Лабораторна робота №2
Засоби автоматизації аналізу

Виконав:
Студент 3 курсу ФТІ
групи ФІ-92
Поночевний Назар Юрійович
Варіант 6

Мета роботи
Отримати навички автоматизації методів аналізу програмного коду.

Завдання 1:
Проаналізуйте обфускатор (encoder) з Metasploit за варіантом (x86/nonupper);

1) Спочатку спробуймо обфускувати звичайний текст і простий ELF файл

```
┌──(kali㉿kali)-[~/lab2-reverse]
└─$ echo -en '\xccHappy kitty, sleepy kitty, purr purr purr' > sc

┌──(kali㉿kali)-[~/lab2-reverse]
└─$ msfvenom -p generic/custom payloadfile=sc -f raw -o payload_raw.bin -e x86/nonupper
[-] No platform was selected, choosing Msf::Module::Platform from the payload
[-] No arch selected, selecting arch: x86 from the payload
Found 1 compatible encoders
Attempting to encode payload with 1 iterations of x86/nonupper
x86/nonupper succeeded with size 78 (iteration=0)
x86/nonupper chosen with final size 78
Payload size: 78 bytes
Saved as: payload_raw.bin

┌──(kali㉿kali)-[~/lab2-reverse]
└─$ msfvenom -a x86 --platform linux -p linux/x86/exec CMD=whoami -f elf -o payload_elf.bin -e x86/nonupper
Found 1 compatible encoders
Attempting to encode payload with 1 iterations of x86/nonupper
x86/nonupper succeeded with size 78 (iteration=0)
x86/nonupper chosen with final size 78
Payload size: 78 bytes
Final size of elf file: 162 bytes
Saved as: payload_elf.bin

┌──(kali㉿kali)-[~/lab2-reverse]
└─$ xxd payload_raw.bin
00000000: 66b9 ffff eb19 5e8b fe83 c700 8bd7 3bf2  f.....^.......;.
00000010: 7d0b b07b f2ae ffcf ac28 07eb f1eb 05e8  }..{.....(......
00000020: e2ff ffff cc48 6170 7079 206b 6974 7479  .....Happy kitty
00000030: 2c20 736c 6565 7079 206b 6974 7479 2c20  , sleepy kitty,
00000040: 7075 7272 2070 7572 7220 7075 7272       purr purr purr

┌──(kali㉿kali)-[~/lab2-reverse]
└─$ xxd payload_elf.bin
00000000: 7f45 4c46 0101 0100 0000 0000 0000 0000  .ELF............
00000010: 0200 0300 0100 0000 5480 0408 3400 0000  ........T...4...
00000020: 0000 0000 0000 0000 3400 2000 0100 0000  ........4. .....
00000030: 0000 0000 0100 0000 0000 0000 0080 0408  ................
00000040: 0080 0408 a200 0000 f000 0000 0700 0000  ................
00000050: 0010 0000 66b9 ffff eb19 5e8b fe83 c700  ....f.....^.....
00000060: 8bd7 3bf2 7d0b b07b f2ae ffcf ac28 07eb  ..;.}..{.....(..
00000070: f1eb 05e8 e2ff ffff 6a0b 5899 5266 682d  ........j.X.Rfh-
00000080: 6389 e768 2f73 6800 682f 6269 6e89 e352  c..h/sh.h/bin..R
00000090: e807 0000 0077 686f 616d 6900 5753 89e1  .....whoami.WS..
000000a0: cd80                                     ..

┌──(kali㉿kali)-[~/lab2-reverse]
└─$ chmod u+x payload_elf.bin && ./payload_elf.bin
kali
```

2) Тепер проаналізуємо [вихідний код обфускатора]

.encode(buf) ⇒ Object

[Hide source]

```ruby
# File 'lib/rex/encoder/nonupper.rb', line 47

def NonUpper.encode(buf)
  table = ""
  tablelen = 0
  nonascii = ""
  encoded = gen_decoder()
  buf.each_byte {
    |block|

    newchar, table, tablelen = encode_byte(block.unpack('C')[0], table, tablelen)
    nonascii += newchar
  }
  encoded.gsub!(/A/, tablelen)
  encoded.gsub!(/B/, tablelen+5)
  encoded += table
  encoded += nonascii
end
```

.encode_byte(badchars, block, table, tablelen) ⇒ Object

[Hide source]

```ruby
# File 'lib/rex/encoder/nonupper.rb', line 31

def NonUpper.encode_byte(badchars, block, table, tablelen)
  if (tablelen > 255) or (block == 0x40)
    raise RuntimeError, "BadChar"
  end

  if (block >= 0x41 and block <= 0x40) or (badchars =~ block)
    # gen offset, return magic
    offset = 0x40 - block;
    table += offset.chr
    tablelen = tablelen + 1
    block = 0x40
  end

  return [block.chr, table, tablelen]
end
```

```
.gen_decoder ⇒ Object
```

[Hide source]

```
    # File 'lib/rex/encoder/nonupper.rb', line 11

11  def NonUpper.gen_decoder()
12    decoder =
13      "\x66\xB9\xFF\xFF" +
14      "\xEB\x19"  +               # Jmp to table
15      "\x5E"      +               # pop esi
16      "\x8B\xFE"  +               # mov edi, esi      - Get table addr
17      "\x83\xC7"  + "A" +         # add edi, tablelen - Get shellcode addr
18      "\x8B\xD7"  +               # mov edx, edi      - Hold end of table ptr
19      "\x3B\xF2"  +               # cmp esi, edx
20      "\x7D\x0B"  +               # jle to end
21      "\xB0\x7B"  +               # mov eax, 0x7B     - Set up eax with magic
22      "\xF2\xAE"  +               # repne scasb       - Find magic!
23      "\xFF\xCF"  +               # dec edi           - scasb purs us one ahead
24      "\xAC"      +               # lodsb
25      "\x28\x07"  +               # subb [edi], al
26      "\xEB\xF1"  +               # jmp BACK!
27      "\xEB"      + "B" +         # jmp [shellcode]
28      "\xE8\xE2\xFF\xFF\xFF"
29  end
```

3) Спробуємо знаходити деобфускований шел-код по знайденій сигнатурі

```
SIGNATURE = bytearray(
    [102, 185, 255, 255, 235, 25, 94, 139,
     254, 131, 199, 0, 139, 215, 59, 242,
     125, 11, 176, 123, 242, 174, 255, 207,
     172, 40, 7, 235, 241, 235, 5, 232,
     226, 255, 255, 255]
)
```

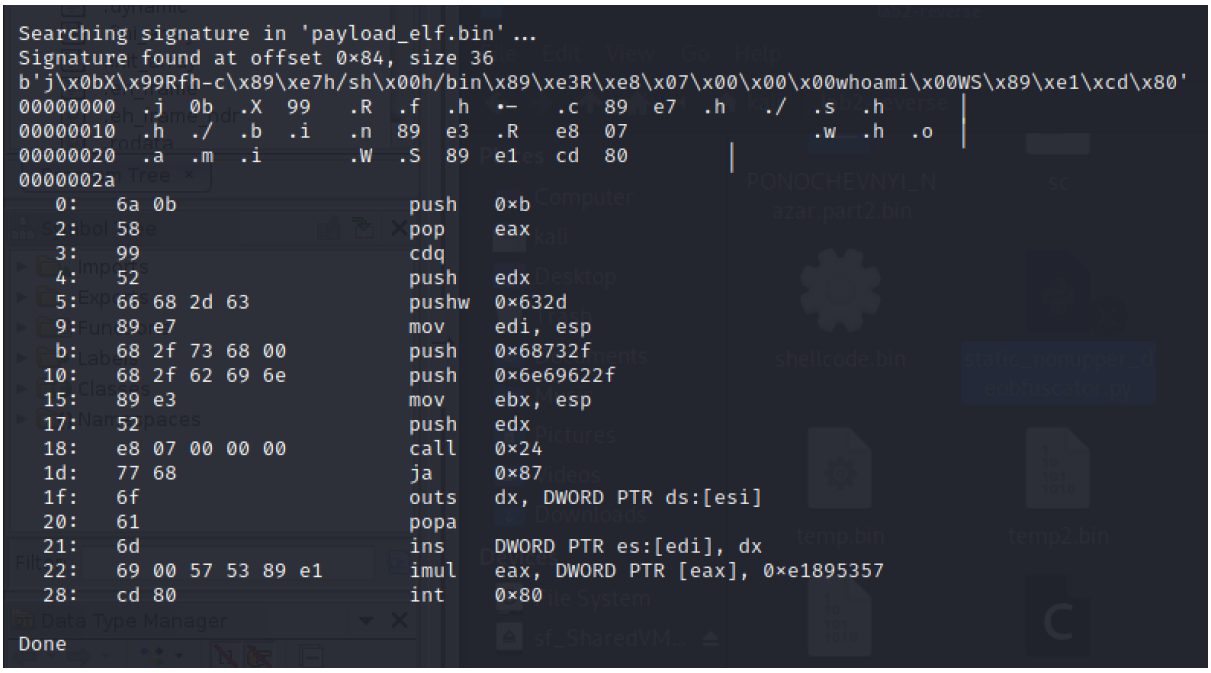
Завдання 2:

Реалізуйте статичний деобфускатор для Вашого варіанту (x86/nonupper);

```
import sys
from pwn import *

SIGNATURE = bytearray(
    [102, 185, 255, 255, 235, 25, 94, 139,
     254, 131, 199, 0, 139, 215, 59, 242,
     125, 11, 176, 123, 242, 174, 255, 207,
     172, 40, 7, 235, 241, 235, 5, 232,
     226, 255, 255, 255]
)

if len(sys.argv) < 2:
    print(f"USAGE: {sys.argv[0]} [filename] ...")
```

```
    sys.exit()

print("Start file(s) processing...")

for filename in sys.argv[1:]:
    print(f"\nSearching signature in '{filename}'...")
    data = read(filename)
    offset = data.find(SIGNATURE)
    if offset == -1:
        print("Signature not found!")
        continue
    print(f"Signature found at offset 0x{offset}, size {len(SIGNATURE)}")
    shellcode = data[offset + len(SIGNATURE):]
    print(shellcode)
    print(hexdump(shellcode, hexii=True))
    print(disasm(shellcode))

print("\nDone")
```

```
┌──(kali㉿kali)-[~/lab2-reverse]
└─$ sudo python3 static_nonupper_deobfuscator.py payload_raw.bin payload_elf.bin
Start file(s) processing...

Searching signature in 'payload_raw.bin'...
Signature found at offset 0x0, size 36
b'\xccHappy kitty, sleepy kitty, purr purr purr'
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
   0:   cc                      int3
   1:   48                      dec    eax
   2:   61                      popa
   3:   70 70                   jo     0x75
   5:   79 20                   jns    0x27
   7:   6b 69 74 74             imul   ebp, DWORD PTR [ecx+0x74], 0x74
   b:   79 2c                   jns    0x39
   d:   20 73 6c                and    BYTE PTR [ebx+0x6c], dh
  10:   65 65 70 79             gs gs jo 0x8d
  14:   20 6b 69                and    BYTE PTR [ebx+0x69], ch
  17:   74 74                   je     0x8d
  19:   79 2c                   jns    0x47
  1b:   20 70 75                and    BYTE PTR [eax+0x75], dh
  1e:   72 72                   jb     0x92
  20:   20 70 75                and    BYTE PTR [eax+0x75], dh
  23:   72 72                   jb     0x97
  25:   20 70 75                and    BYTE PTR [eax+0x75], dh
  28:   72 72                   jb     0x9c
```



Завдання 3:

Реалізуйте динамічний деобфускатор для Вашого варіанту (x86/nonupper);

```
import sys
from unicorn import *
from unicorn.x86_const import *
from capstone import *

if len(sys.argv) != 2:
    print(f"USAGE: {sys.argv[0]} [filepath]")
    sys.exit()

cs = Cs(CS_ARCH_X86, CS_MODE_64)


def hook_code(uc, address, size, user_data):
    global cs
    mem = uc.mem_read(address, size)
    for i in cs.disasm(mem, size):
        print("hook 0x{:03x} size {:2d}: {:20s} {} {}".format(
            address, size, i.bytes.hex(), i.mnemonic, i.op_str))
        if i.bytes.hex() == "cd80":
            rax = mu.reg_read(UC_X86_REG_RAX)
            rbx = mu.reg_read(UC_X86_REG_RBX)
            rsi = mu.reg_read(UC_X86_REG_RSI)
            if rax == 11:
                sh_data = [bytes(x) for x in mu.mem_read(
                    rbx, 0x1000).split(b"\0")[:16] if x != b""]
                sh, args = (sh_data[0] + sh_data[1]
```

```
                            ).decode("utf8"), sh_data[2].decode("utf8")
                cmd_data = [bytes(x) for x in mu.mem_read(
                    rsi, 0x1000).split(b"\0")[:16] if x != b""]
                cmd = cmd_data[2].decode("utf8")
                print(f"[SYSCALL] sys_execve {sh} {args} {cmd}")
            else:
                print(f"[SYSCALL] rax = 0x{rax}, rbx = 0x{rbx}, rsi = 0x{rsi}")


print(f"\nOpening {sys.argv[1]} file for processing...\n")
with open(sys.argv[1], "rb") as file:
    code = file.read()
address = 0x08048000

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)

try:
    mu.emu_start(0x08048054, address + len(code))
except:
    pass
print("\nDone")
```

```
┌──(kali㉿kali)-[~/lab2-reverse]
└─$ sudo python3 dynamic_nonupper_deobfuscator.py payload_elf.bin

Opening payload_elf.bin file for processing...

hook 0x8048054 size  4: 66b9ffff            mov cx, 0xffff
hook 0x8048058 size  2: eb19                jmp 0x1d
hook 0x8048073 size  5: e8e2ffffff          call 0xffffffffffffffec
hook 0x804805a size  1: 5e                  pop rsi
hook 0x804805b size  2: 8bfe                mov edi, esi
hook 0x804805d size  3: 83c700              add edi, 0
hook 0x8048060 size  2: 8bd7                mov edx, edi
hook 0x8048062 size  2: 3bf2                cmp esi, edx
hook 0x8048064 size  2: 7d0b                jge 0xf
hook 0x8048071 size  2: eb05                jmp 9
hook 0x8048078 size  2: 6a0b                push 0xb
hook 0x804807a size  1: 58                  pop rax
hook 0x804807b size  1: 99                  cdq
hook 0x804807c size  1: 52                  push rdx
hook 0x804807d size  4: 66682d63            push 0x632d
hook 0x8048081 size  2: 89e7                mov edi, esp
hook 0x8048083 size  5: 682f736800          push 0x68732f
hook 0x8048088 size  5: 682f62696e          push 0x6e69622f
hook 0x804808d size  2: 89e3                mov ebx, esp
hook 0x804808f size  1: 52                  push rdx
hook 0x8048090 size  5: e807000000          call 0x11
hook 0x804809c size  1: 57                  push rdi
hook 0x804809d size  1: 53                  push rbx
hook 0x804809e size  2: 89e1                mov ecx, esp
hook 0x80480a0 size  2: cd80                int 0x80
[SYSCALL] sys_execve /bin/sh -c whoami

Done
```