網安實務 Hw3

網工所碩一 309552005 吳偉誠

用到的工具:angr(該HW的練習工具、版本: 8.20.1.7)、python3.8、gdb-peda、pwntool
後兩者是用於跑target來觀看其overflow問題的工具

- angr裝於在虛擬中，本來angr是裝最新版(9)，但API似乎有所更動，造成助教提供的腳本跑不起來，因此才降版本
- gdb-peda安裝教學參考:https://ithelp.ithome.com.tw/articles/10227380
- pwntool安裝教學參考:https://docs.pwntools.com/en/stable/install.html

Question1

題目中，可以看到有兩個overflow漏洞:

```
void func()
{
    char pwd[0x10]={0};
    puts("input admin password:");
    read(0,pwd,0x20);
}
void over()
{
    puts("over!");
    char c[0x10]={0};
    read(0,c,0x20);
}
```

> 超出該字元長度

1. Describe the overall code structure of the script.

- simply_exploit.py
  - 透過unconstrained狀態(with the instruction pointer controlled by user data or some other source of symbolic data)來搜索漏洞
    - 會存在unconstrained stash中
    - 這種狀態的路徑一般來說就是rip值不可約束(不受控制它的值符號化了、)才會產生的，例如一般發生Stack Overflow時，rip的值通常是標準輸入的某段字符串，而在angr中，stdin也會被符號化，所以說當rip值變成stdin的部分值時，也就當作rip的值也是符號化的，這樣就出現了unconstrained狀態
  - 讓simulation_manager重複單步執行
    - 如果有unconstrained狀態
      - 就顯示其結果
- full_exploit.py
  - 透過檢查指令"push rbp;"、"mov rbp,rsp;"來判斷是否是函數的開頭(check_head(state))
  - 透過檢查指令"leave;"、"ret;"來判斷是否是函數的結尾(check_end(state))
  - 每次進入新函式時，利用字典的方式儲存rbp值(key為正確返回的地址)
    - 不管多複雜，都可以透過唯一的返回地址鎖定rbp的正確值
  - 透過檢測某地址的值是否符號化，來計算出overflow的具體字節
    - 如果overflow到rbp或返回地址，可以透過檢測順序來解決

            - 例:overflow到返回地址，那必然overflow了rbp
                - 直接報出pc overflow
            - 先檢測返回地址是否被overflow，再檢測是否overflow到了rbp
                - 如果只overflow到rbp則報出rbp overflow
    - 再利用跟"simply_exploit.py"類似的方式(unconstrained狀態)來檢驗
        - 讓simulation_manager重複單步執行
            - 檢測函式的head跟end
                - 如果rbp或返回地址被符號化了
                - 印出相對應的訊息(pc overflow or rbp overflow)

2. Explain the purpose of the code on the lines marked with comment symbol.

#1-1:

如果sm(simulation_manager)有unconstrained的狀態，就遍歷其內容，然後去列出stdout、stdin內容

#1-2:

用於檢查函式一開始是不是"push rbp;"、"mov rbp,rsp;"，來判斷是否是該函式的開頭
如果是的話將返回地址與rbp給存起來

#1-3:

當flag等於2，則代表已是函式的結尾了，其代碼是將stack上的rbp跟ret給讀出來，並把之前存的(在check_head()裡做的)返回地址與rbp給讀出來，用於後面程式的進行

#1-4:

先檢查返回地址是否有被overflow(透過其內容是否有被符號化來判定)，如果有則印出pc_overflow的訊息，並將原本存的rbp跟返回地址給變數rbp、rbp+byte_s使其復原(可讓程式在正常往下執行找更多Bug)

#1-5:

跟1-4類似，檢查rbp的值是否有被先檢查返回地址是否有被overflow(透過其內容是否有被符號化來判定)，如果有則印出rbp_overflow的訊息，並將原本存的rbp跟返回地址給變數rbp使其復原(可讓程式在正常往下執行找更多Bug)

3. Test the exploit(input) on the C program and show your results

Run simply_exploit.py:

```
wulearn@wulearn-MS-7B24: ~/Desktop/NS_hw3/Hw3/Question_A
(angr) wulearn@wulearn-MS-7B24:~/Desktop/NS_hw3/Hw3/Question_A$ python simply_exploit.py
WARNING | 2021-04-28 15:14:37,662 | cle.loader | The main binary is a position-independent e
xecutable. It is being loaded with a base address of 0x400000.
WARNING | 2021-04-28 15:14:38,802 | angr.engines.successors | Exit state has over 256 possib
le solutions. Likely unconstrained; skipping. <BV64 Reverse(packet_1_stdin_43_256[63:0])>
stdout:
 b'input your name:\nover!\n'
stdin:
 b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\
x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00Q\x01\x11\x14u\x03T\x
10'
```

> 可以發現它只找到了over函數的overflow問題，無法檢查到func函數的overflow問題
> 因為執行到over函數時就結束了後續的路徑探索，因為overflow使rip的值unconstrained了

Run full_exploit.py:

```
wulearn@wulearn-MS-7B24: ~/Desktop/NS_hw3/Hw3/Question_A
(angr) wulearn@wulearn-MS-7B24:~/Desktop/NS_hw3/Hw3/Question_A$ python full_exploit.py
WARNING | 2021-04-28 15:17:15,844 | cle.loader | The main binary is a position-independent executable.
It is being loaded with a base address of 0x400000.

[========find a pc overflow========]
over for  0x8 bytes
[PC]stdout:
 b'input your name:\nover!\n'
[PC]trigger overflow input:
b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\
x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'
WARNING | 2021-04-28 15:17:16,216 | angr.state_plugins.symbolic_memory | The program is accessing memor
y or registers with an unspecified value. This could indicate unwanted behavior.
WARNING | 2021-04-28 15:17:16,216 | angr.state_plugins.symbolic_memory | angr will cope with this by ge
nerating an unconstrained symbolic variable and continuing. You can resolve this by:
WARNING | 2021-04-28 15:17:16,216 | angr.state_plugins.symbolic_memory | 1) setting a value to the init
ial state
WARNING | 2021-04-28 15:17:16,216 | angr.state_plugins.symbolic_memory | 2) adding the state option ZER
O_FILL_UNCONSTRAINED_{MEMORY,REGISTERS}, to make unknown regions hold null
WARNING | 2021-04-28 15:17:16,217 | angr.state_plugins.symbolic_memory | 3) adding the state option SYM
BOL_FILL_UNCONSTRAINED_{MEMORY_REGISTERS}, to suppress these messages.
WARNING | 2021-04-28 15:17:16,217 | angr.state_plugins.symbolic_memory | Filling memory at 0x7fffffffff
efff8 with 88 unconstrained bytes referenced from 0x1000020 (strstr+0x0 in extern-address space (0x20))
WARNING | 2021-04-28 15:17:16,217 | angr.state_plugins.symbolic_memory | Filling memory at 0x7fffffffff
eff70 with 8 unconstrained bytes referenced from 0x1000020 (strstr+0x0 in extern-address space (0x20))

[========find a pc overflow========]
over for  0x8 bytes
[PC]stdout:
 b'input your name:\nover!\ninput admin password:\n'
[PC]trigger overflow input:
b'admin\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x0
0\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'
(angr) wulearn@wulearn-MS-7B24:~/Desktop/NS_hw3/Hw3/Question_A$
```

> 除了偵測到over函數的overflow問題外，還檢查到了func函數的overflow問題

Run C program:

在跑程式之前先將程式改成可執行(`$ chmod +x stack1`)
因為有特殊的byte的input、以及要觀看overflow問題，因此就用最初提到的pwntool與gdb-peda來觀看

寫個簡單的腳本:

```
#!/usr/bin/env python3
# -*- encoding: utf-8 -*-
from pwn import *
elf  = ELF('./stack1')
```

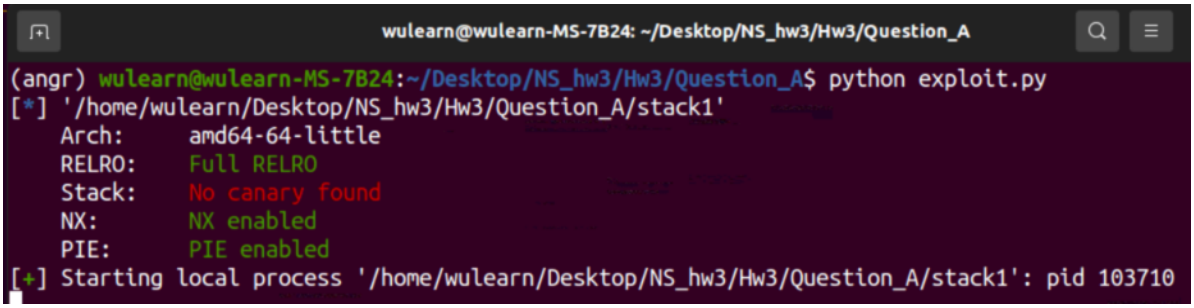
```
def main():
    proc=elf.process()
    input()#用於handle住程式，方便gdb trace
    proc.recvuntil(":")#input your name:
    proc.send(b'123')
    proc.recvuntil("!")#over!
    proc.send(b'aaa')
    proc.interactive()

if __name__ == '__main__':
    main()
```

> 此為正常執行程式無觸發overflow問題的腳本

Run:

`$ python exploit.py`



> 得到pid:103710(下一步會用到)

gdb:

`$ gdb -p 103710`

```
wulearn@wulearn-MS-7B24: ~/Desktop/NS_hw3/Hw3/Question_A

──────────────────────────────────────── Registers ────────────────────────────────────────
RAX: 0xffffffffffffffe00
RBX: 0x55a39ad2b840 (<__libc_csu_init>: push   r15)
RCX: 0x7f4361031142 (<__GI___libc_read+18>:     cmp    rax,0xfffffffffffff000)
RDX: 0x10
RSI: 0x7ffebd462bf0 --> 0x0
RDI: 0x0
RBP: 0x7ffebd462c00 --> 0x0
RSP: 0x7ffebd462bd8 --> 0x55a39ad2b7e4 (<main+70>:      mov    eax,0x0)
RIP: 0x7f4361031142 (<__GI___libc_read+18>:     cmp    rax,0xfffffffffffff000)
R8 : 0x11
R9 : 0x7c ('|')
R10: 0x7f436110bbe0 --> 0x55a39cf0d6a0 --> 0x0
R11: 0x246
R12: 0x55a39ad2b610 (<_start>:  xor    ebp,ebp)
R13: 0x7ffebd462cf0 --> 0x1
R14: 0x0
R15: 0x0
EFLAGS: 0x246 (carry PARITY adjust ZERO sign trap INTERRUPT direction overflow)
──────────────────────────────────────── Code ────────────────────────────────────────
   0x7f436103113c <__GI___libc_read+12>:        test   eax,eax
   0x7f436103113e <__GI___libc_read+14>:        jne    0x7f4361031150 <__GI___libc_read+32>
   0x7f4361031140 <__GI___libc_read+16>:        syscall
=> 0x7f4361031142 <__GI___libc_read+18>:        cmp    rax,0xfffffffffffff000
   0x7f4361031148 <__GI___libc_read+24>:        ja     0x7f43610311a0 <__GI___libc_read+112>
   0x7f436103114a <__GI___libc_read+26>:        ret
   0x7f436103114b <__GI___libc_read+27>:        nop    DWORD PTR [rax+rax*1+0x0]
   0x7f4361031150 <__GI___libc_read+32>:        sub    rsp,0x28
──────────────────────────────────────── Stack ────────────────────────────────────────
0000| 0x7ffebd462bd8 --> 0x55a39ad2b7e4 (<main+70>:      mov    eax,0x0)
0008| 0x7ffebd462be0 --> 0x7ffebd462cf8 --> 0x7ffebd4630b9 ("/home/wulearn/Desktop/NS_hw3/Hw3/Questio
n_A/stack1")
0[ LibreOffice Impress be8 --> 0x19ad2b610
0024| 0x7ffebd462bf0 --> 0x0
0032| 0x7ffebd462bf8 --> 0x0
0040| 0x7ffebd462c00 --> 0x0
0048| 0x7ffebd462c08 --> 0x7f4360f470b3 (<__libc_start_main+243>:       mov    edi,eax)
0056| 0x7ffebd462c10 --> 0x7f4361153620 --> 0x5081200000000

Legend: code, data, rodata, heap, value
0x00007f4361031142 in __GI___libc_read (fd=0x0, buf=0x7ffebd462bf0, nbytes=0x10)
    at ../sysdeps/unix/sysv/linux/read.c:26
26      ../sysdeps/unix/sysv/linux/read.c: No such file or directory.
gdb-peda$
```

在exploit那邊終端機畫面按下`enter`後，gdb這邊輸入`fin`(剩餘基本操作步驟就省略，基本上就都使用`si`來單步執行到要的位置)

讓gdb進到over的函式裡面，去看它的return address是不是正常的

```
──────────────────────────────────────── Code ────────────────────────────────────────
   0x558dfa241796 <over+58>:    call   0x558dfa2415e0 <read@plt>
   0x558dfa24179b <over+63>:    nop
   0x558dfa24179c <over+64>:    leave
=> 0x558dfa24179d <over+65>:    ret
   0x558dfa24179e <main>:       push   rbp
   0x558dfa24179f <main+1>:     mov    rbp,rsp
   0x558dfa2417a2 <main+4>:     sub    rsp,0x20
   0x558dfa2417a6 <main+8>:     mov    DWORD PTR [rbp-0x14],edi
TeamViewer dfa2417ee <main+80>:        lea    rax,[rbp-0x10]
   0x558dfa2417f2 <main+84>:        lea    rsi,[rip+0xf8]        # 0x558dfa2418f1
   0x558dfa2417f9 <main+91>:        mov    rdi,rax
   0x558dfa2417fc <main+94>:        call   0x558dfa2415f0 <strstr@plt>
──────────────────────────────────────── Stack ────────────────────────────────────────
0000| 0x7ffc84c7de08 --> 0x558dfa2417ee (<main+80>:      lea    rax,[rbp-0x10])
0008| 0x7ffc84c7de10 --> 0x7ffc84c7df28 --> 0x7ffc84c7f0b9 ("/home/wulearn/Desktop/NS_hw3/Hw3/Questio
n_A/stack1")
0016| 0x7ffc84c7de18 --> 0x1fa241610
0024| 0x7ffc84c7de20 --> 0x6e696d6461 ('admin')
0032| 0x7ffc84c7de28 --> 0x0
0040| 0x7ffc84c7de30 --> 0x0
0048| 0x7ffc84c7de38 --> 0x7ff9dd7640b3 (<__libc_start_main+243>:       mov    edi,eax)
0056| 0x7ffc84c7de40 --> 0x7ff9dd970620 --> 0x5081200000000
```

> 可以看到它返回位址只是正常的

接著來驗證兩個overflow的問題。
第一個overflow:(over函式裡的)
將上述exploit.py的`proc.send(b'aaa')`改成:
`proc.send(b'\x00'*48)`

> 由上述full_exploit第一個stdin提供

然後再來跟剛剛上述run的流程一樣，去觀看其return的位置:

```
─────────────────────────────────────────────────────────────────────────── Code ───────────────────────────────────────────────────────────────────────────
   0x5566473ae796 <over+58>:    call   0x5566473ae5e0 <read@plt>
   0x5566473ae79b <over+63>:    nop
   0x5566473ae79c <over+64>:    leave
=> 0x5566473ae79d <over+65>:    ret
   0x5566473ae79e <main>:       push   rbp
   0x5566473ae79f <main+1>:     mov    rbp,rsp
   0x5566473ae7a2 <main+4>:     sub    rsp,0x20
   0x5566473ae7a6 <main+8>:     mov    DWORD PTR [rbp-0x14],edi
 TeamViewer                                                                    JUMP is NOT taken
─────────────────────────────────────────────────────────────────────────── Stack ──────────────────────────────────────────────────────────────────────────
0000| 0x7ffd963bdde8 --> 0x0
0008| 0x7ffd963bddf0 --> 0x7ffd963bdf08 --> 0x7ffd963c00b9 ("/home/wulearn/Desktop/NS_hw3/Hw3/Questio
n_A/stack1")
0016| 0x7ffd963bddf8 --> 0x1473ae610
0024| 0x7ffd963bde00 --> 0x333231 ('123')
0032| 0x7ffd963bde08 --> 0x0
0040| 0x7ffd963bde10 --> 0x0
0048| 0x7ffd963bde18 --> 0x7f653a7f50b3 (<__libc_start_main+243>:       mov    edi,eax)
0056| 0x7ffd963bde20 --> 0x7f653aa01620 --> 0x5081200000000

Legend: code, data, rodata, heap, value
0x00005566473ae79d in over ()
```

> 可以看到返回位址已被更改(overflow問題)
> 造成程式無法運作

第二個overflow:(func函式裡的)
將腳本改成以下這樣:

```
#!/usr/bin/env python3
# -*- encoding: utf-8 -*-
from pwn import *
elf  = ELF('./stack1')

def main():
    proc=elf.process()
    input()
    proc.recvuntil(":")
    proc.send(b'admin')
    proc.recvuntil("!")
    proc.send( b'\x00')
    proc.recvuntil("password:")
    proc.send( b'\x00'*48)
    proc.interactive()

if __name__ == '__main__':
    main()
```

然後跟剛剛一樣透過gdb來trace，這次是要看在func()的回傳位址:

```
──────────────────────────────────────────────────────────────────── Code ────────────────────────────────────────────────────────────────────
   0x5589f0b0e754 <func+58>:    call   0x5589f0b0e5e0 <read@plt>
   0x5589f0b0e759 <func+63>:    nop
   0x5589f0b0e75a <func+64>:    leave
=> 0x5589f0b0e75b <func+65>:    ret
   0x5589f0b0e75c <over>:       push   rbp
   0x5589f0b0e75d <over+1>:     mov    rbp,rsp
   0x5589f0b0e760 <over+4>:     sub    rsp,0x10
   0x5589f0b0e764 <over+8>:     lea    rdi,[rip+0x16f]        # 0x5589f0b0e8da
                                                               JUMP is NOT taken
──────────────────────────────────────────────────────────────────── Stack ───────────────────────────────────────────────────────────────────
0000| 0x7fff93baf628 --> 0x0
0008| 0x7fff93baf630 --> 0x7fff93baf748 --> 0x7fff93bb10b9 ("/home/wulearn/Desktop/NS_hw3/Hw3/Questio
n_A/stack1")
0016| 0x7fff93baf638 --> 0x1f0b0e610
0024| 0x7fff93baf640 --> 0x6e696d6461 ('admin')
0032| 0x7fff93baf648 --> 0x0
0040| 0x7fff93baf650 --> 0x0
0048| 0x7fff93baf658 --> 0x7f8c188810b3 (<__libc_start_main+243>:       mov    edi,eax)
0056| 0x7fff93baf660 --> 0x7f8c18a8d620 --> 0x5081200000000
──────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────
Legend: code, data, rodata, heap, value
0x00005589f0b0e75b in func ()
```

> 可以看到返回位址已被更改(overflow問題)
> 造成程式無法運作

如果正常的會是這樣(將上面的`*48`拿掉即可):

```
──────────────────────────────────────────────────────────────────── Code ────────────────────────────────────────────────────────────────────
   0x558c7f6d9754 <func+58>:    call   0x558c7f6d95e0 <read@plt>
   0x558c7f6d9759 <func+63>:    nop
   0x558c7f6d975a <func+64>:    leave
=> 0x558c7f6d975b <func+65>:    ret
   0x558c7f6d975c <over>:       push   rbp
   0x558c7f6d975d <over+1>:     mov    rbp,rsp
   0x558c7f6d9760 <over+4>:     sub    rsp,0x10
   0x558c7f6d9764 <over+8>:     lea    rdi,[rip+0x16f]        # 0x558c7f6d98da
 >    0x558c7f6d9810 <main+114>:       lea    rdi,[rip+0xe0]        # 0x558c7f6d98f7
      0x558c7f6d9817 <main+121>:       call   0x558c7f6d95c0 <puts@plt>
      0x558c7f6d981c <main+126>:       jmp    0x558c7f6d9836 <main+152>
      0x558c7f6d981e <main+128>:       lea    rax,[rbp-0x10]
──────────────────────────────────────────────────────────────────── Stack ───────────────────────────────────────────────────────────────────
0000| 0x7ffe9a4768e8 --> 0x558c7f6d9810 (<main+114>:    lea    rdi,[rip+0xe0]        # 0x558c7f6d98f7
)
0008| 0x7ffe9a4768f0 --> 0x7ffe9a476a08 --> 0x7ffe9a4780b9 ("/home/wulearn/Desktop/NS_hw3/Hw3/Questio
n_A/stack1")
0016| 0x7ffe9a4768f8 --> 0x17f6d9610
0024| 0x7ffe9a476900 --> 0x6e696d6461 ('admin')
0032| 0x7ffe9a476908 --> 0x0
0040| 0x7ffe9a476910 --> 0x0
0048| 0x7ffe9a476918 --> 0x7f370943d0b3 (<__libc_start_main+243>:       mov    edi,eax)
0056| 0x7ffe9a476920 --> 0x7f3709649620 --> 0x5081200000000
──────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────
Legend: code, data, rodata, heap, value
0x0000558c7f6d975b in func ()
```

> 會有正確位址的回傳值