1091 電腦攻擊與防禦

GDB & PWNTOOLS INTRO

--BY TA 璟方

OUTLINE

- What is pwn?
- Flow
- Useful Tools
- GDB
- PWNTOOLS

What is PWN

- 碰(O) / 胖(X)
- pwn ← own
- pwn = binary exploitation
  - 利用binary的漏洞,在執行期間控制其control flow 以達到特定行為 (ex get shell in CTF)

Flow

1. Reverse Engineering（逆向工程）：尋找漏洞
   - 通常只會拿到binary,而非程式原始碼
2. Exploitation（漏洞利用）

Useful Tools

1. Reverse Engineering（逆向工程）：尋找漏洞

- 靜態分析
  - objdump
  - ida pro
  - Ghidra
- 動態分析
  - GDB
  - Ollydbg
  - Windbg

Objdump

- dump出執行檔中的組合語言

```
$ objdump -M intel -d <執行檔>
```

  - -M intel ：設定組合語言的syntax為intel，default是AT&T
  - 後面接 | less或 | grep更方便使用

Objdump

- Ex.

```
 minyeon@MinYeon  vivia/Desktop/demo  objdump -M intel -d demo

demo:     file format elf64-x86-64


Disassembly of section .init:

0000000000401000 <_init>:
  401000:       f3 0f 1e fa             endbr64
  401004:       48 83 ec 08             sub    rsp,0x8
  401008:       48 8b 05 e9 2f 00 00    mov    rax,QWORD PTR [rip+0x2fe9]        # 403ff8 <__gmon_start__>
  40100f:       48 85 c0                test   rax,rax
  401012:       74 02                   je     401016 <_init+0x16>
  401014:       ff d0                   call   rax
  401016:       48 83 c4 08             add    rsp,0x8
  40101a:       c3                      ret
```

GDB -- installation

- Origin GDB

```
$ sudo apt-get update
$ sudo apt-get install gdb
```

- 好用插件
  - gef / pwndbg / ~~gdb-peda~~

GDB -- Basic Command

- run <arg1> <arg2> ...
  - r < file.txt : 把檔案內容當作input

```
 gdb ⋆  r < test.txt
Starting program: /mnt/c/Users/vivia/Desktop/demo/demo < test.txt
who r u?
Hello 12345!
```

  - r <<< $(cmd) : 把cmd執行結果當作input

```
 gdb ⋆  r <<< $(echo QQ)
Starting program: /mnt/c/Users/vivia/Desktop/demo/demo <<< $(echo QQ)
who r u?
Hello QQ!
```

GDB -- Basic Command

- disas main ：disassemble main function
- break main ：下斷點在main function
- break *0x4011fb ：下斷點在0x4011fb
- info breakpoint ：查看現在所有斷點
- delete 2：刪除第2個斷點
- disable/enable 2：暫停/恢復第2個斷點

GDB -- Basic Command

- Ex.

```
 gdb ☆  b main
Breakpoint 1 at 0x4011cd
 gdb ☆  b *0x4011d0
Breakpoint 2 at 0x4011d0
 gdb ☆  i b
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x00000000004011cd <main>
2       breakpoint     keep y   0x00000000004011d0 <main+3>
 gdb ☆  d 2
 gdb ☆  dis 1
 gdb ☆  i b
Num     Type           Disp Enb Address            What
1       breakpoint     keep n   0x00000000004011cd <main>
```

GDB -- Basic Command

- continue：繼續執行
- ni / n ：step over, 遇到function不會跟進去
  - ni：是針對assembly
  - n：是針對source code
- si / s ：step in, 遇到function會跟進去

GDB -- Basic Command

- x/10gx <address> ：查看address中的內容
  - b/h/w/g：代表取1/2/4/8 bytes
  - x：以hex形式印出,可替代為
    - i：以指令形式印出
    - u：以unsigned int的形式印出
    - S：以字串形式印出
  - 10：從address開始印出10個

GDB -- Basic Command

- Ex.

```
 gdb*  x/gx 0x601020
0x601020:       0x00000000004005a6
 gdb*  x/3i $rip
=> 0x4006e8 <main+4>:    sub    $0x10,%rsp
   0x4006ec <main+8>:
    mov    0x20096d(%rip),%rax        # 0x601060
   0x4006f3 <main+15>:   mov    $0x0,%ecx
```

- Note: $<Register> 代表暫存器中的值

GDB -- Basic Command

- `set *<address>=<value>`：將address中的值設成value
  - `*` 代表設定4 byte

    可取代成{char/short/long}，分別代表1/2/8 bytes，

    也可以取代成{int}，代表value為int形式
  - Ex

```
gdb ☆ set *0x404000=0xdeadbeef
gdb ☆ x/gx 0x404000
0x404000:       0x00000000deadbeef
gdb ☆ set {int}0x404000=48
gdb ☆ x/gx 0x404000
0x404000:       0x0000000000000030
```

GDB -- Basic Command

- `attach <pid>`：attach一個正在執行的process,
  - 需要root權限
  - `$ echo 0 > /proc/sys/kernel/yama/ptrace_scope`

```
 gdb ⋆  shell ps -ef | grep demo
minyeon     555   519  0 21:28 tty1     00:00:00 ./demo
minyeon     564   562  0 21:28 tty2     00:00:00 zsh -c ps -ef | grep demo
minyeon     566   564  0 21:28 tty2     00:00:00 grep demo
 gdb ⋆  attach 555
Attaching to process 555
Reading symbols from /mnt/c/Users/vivia/Desktop/demo/demo...
```

GDB -- Basic Command

- `set follow-fork-mode <parent|child>`
  - fork之後(eg,system),要繼續debug parent還是child process

GEF -- Installation

- GEF

```
$ wget -O ~/.gdbinit-gef.py -q
  https://github.com/hugsy/gef/raw/master/gef.py

$ echo source ~/.gdbinit-gef.py >> ~/.gdbinit
```

GEF -- Useful feature

- checksec：查看binary有哪些保護機制
- vmmap：查看process mapping狀況
- pattern create/search：可以算overflow offset
- ropper：列出rop gadget
- search-pattern：在process memory中找特定字串

GEF -- Useful feature

- checksec
  - 查看binary有哪些保護機制

```
gef ☆ meow checksec
[+] checksec for '/mnt/c/Users/vivia/Desktop/demo/demo'
Canary                                : ✗
NX                                    : ✓
PIE                                   : ✗
Fortify                               : ✗
RelRO                                 : Partial
```

GEF -- Useful feature

vmmap

- 查看process的mapping状况

```
gef ⚡ meow vmmap
[ Legend:  Code | Heap | Stack ]
Start              End                Offset             Perm Path
0x0000000000400000 0x0000000000401000 0x0000000000000000 r-x /home/peanut0203/2020-pwn/helloctf/helloctf
0x0000000000600000 0x0000000000601000 0x0000000000000000 r-- /home/peanut0203/2020-pwn/helloctf/helloctf
0x0000000000601000 0x0000000000602000 0x0000000000001000 rw- /home/peanut0203/2020-pwn/helloctf/helloctf
0x00007ffff79e4000 0x00007ffff7bcb000 0x0000000000000000 r-x /lib/x86_64-linux-gnu/libc-2.27.so
0x00007ffff7bcb000 0x00007ffff7dcb000 0x00000000001e7000 --- /lib/x86_64-linux-gnu/libc-2.27.so
0x00007ffff7dcb000 0x00007ffff7dcf000 0x00000000001e7000 r-- /lib/x86_64-linux-gnu/libc-2.27.so
0x00007ffff7dcf000 0x00007ffff7dd1000 0x00000000001eb000 rw- /lib/x86_64-linux-gnu/libc-2.27.so
0x00007ffff7dd1000 0x00007ffff7dd5000 0x0000000000000000 rw-
0x00007ffff7dd5000 0x00007ffff7dfc000 0x0000000000000000 r-x /lib/x86_64-linux-gnu/ld-2.27.so
0x00007ffff7feb000 0x00007ffff7fed000 0x0000000000000000 rw-
0x00007ffff7ff7000 0x00007ffff7ffa000 0x0000000000000000 r-- [vvar]
0x00007ffff7ffa000 0x00007ffff7ffc000 0x0000000000000000 r-x [vdso]
0x00007ffff7ffc000 0x00007ffff7ffd000 0x0000000000027000 r-- /lib/x86_64-linux-gnu/ld-2.27.so
0x00007ffff7ffd000 0x00007ffff7ffe000 0x0000000000028000 rw- /lib/x86_64-linux-gnu/ld-2.27.so
0x00007ffff7ffe000 0x00007ffff7fff000 0x0000000000000000 rw-
0x00007ffffffde000 0x00007ffffffff000 0x0000000000000000 rw- [stack]
0xffffffffff600000 0xffffffffff601000 0x0000000000000000 r-x [vsyscall]
```

GEF -- Useful feature

- pattern create/search
  - 算overflow的大小時很好用

```
gef ➤ meow pattern create 30
[+] Generating a pattern of 30 bytes
aaaaaaaabaaaaaaacaaaaaaaadaaaaa
[+] Saved as '$_gef0'
```

  - 如果發現死在ret

```
gef ➤ meow pattern search $rsp
[+] Searching '$rsp'
[+] Found at offset 18 (little-endian search) likely
```

→代表要塞18byte的junk

GEF -- Useful feature

- ropper
  - 可以找ROP gadget

```
gef ★ meow ropper --search "pop r?i; ret"
[INFO] Load gadgets from cache
[LOAD] loading... 100%
[LOAD] removing double gadgets... 100%
[INFO] Searching for gadgets: pop r?i; ret

[INFO] File: /mnt/c/Users/vivia/Desktop/demo/demo.static
0x000000000040188a: pop rdi; ret;
0x000000000040f3fe: pop rsi; ret;
```

GEF -- Useful feature

- search-pattern <str/addr>
  - 可以拿來找字串或地址

```
gef ⭐ meow search-pattern /bin/sh
[+] Searching '/bin/sh' in memory
[+] In '/usr/lib/x86_64-linux-gnu/libc-2.31.so'(
0x7ffffff73d000-0x7ffffff787000), permission=r--
  0x7ffffff7575aa - 0x7ffffff7575b1  →   "/bin/sh"
```

Flow

1. Reverse Engineering（逆向工程）：尋找漏洞
   - 通常只會拿到binary,而非程式原始碼
2. Exploitation（漏洞利用）
   - pwntools：python exploit library

     https://github.com/Gallopsled/pwntools

PWNTOOLS

- from pwn import *

```
'''連接遠端主機'''
r = remote('140.115.59.7', 11001) # usage : remote(host,port)
# exploit code
r.interactive() # 取得shell後可將command傳到terminal上
```

```
'''本地端process'''
p = process('./demo') # usage : process(binary, env)
context.terminal = ['tmux', 'splitw', '-h'] # 在tmux下可以切出視窗跑gdb
gdb.attach(p) # attach到gdb
# explot code
p.interactive()
```

PWNTOOLS

- recv / send

```
'''recv'''
r.recv()
r.recvline() # 接收一行
r.recvlines(num) # 接收(num)行
r.recvuntil(str) # 接收直到碰到(str)
```

```
'''send'''
r.send(payload)
r.sendline(payload) # 會在最後面加一個空字符
```

PWNTOOLS

- Payload construct :

```
'''pack & unpack'''
p32(0xdeadbeef) #'\xef\xbe\xad\xde'
p64(0xdeadbeef) #'\xef\xbe\xad\xde\x00\x00\x00\x00'
hex(u32('\xef\xbe\xad\xde')) # 0xdeadbeef
hex(u64('\xef\xbe\xad\xde\x00\x00\x00\x00')) # 0xdeadbeef
```

```
'''payload可以用flat接起來'''
flat('a'*5, p32(0xdeadbeef)) # b'aaaaa\xef\xbe\xad\xde'
```

PWNTOOLS

- Shellcode
  - 記得先指定架構，或是asm()也可以帶參數

```
'''context'''
context.arch = "amd64"
context.os = 'linux'
context.endian = 'little' # little endian
```

```
'''shllcode & asm'''
asm('mov rax,0; syscall') # b'H\xc7\xc0\x00\x00\x00\x00\x0f\x05'
asm('mov eax, SYS_execve', arch='i386') # b'\xb8\x03\x00\x00\x00'
asm(shellcraft.sh())
shellcraft.i386.mov('eax', 0x20)
```

PWNTOOLS

- ELF
  - 尋找特定Function或library function

```
'''ELF'''
e = ELF(elf_file)
e.got['put'] # puts在got的地址
e.plt['puts'] # puts在plt的地址
```

```
'''libc'''
lib = ELF('libc.so.6')
lib.symbols['system'] # 找system的offset
lib.search('/bin/sh') # 找'/bin/sh' offset
```

PWNTOOLS

- ROP chain
  - 也可以用ropper或ROPgadget找

```
'''ROP'''
rop = ROP(elf_file) # 產生一個空的rop鏈
rop.chain() # 印出目前chain
rop.dump() # dump出chain在stack的樣子
rop.read(0, elf.bss(0x80)) # 如果存在可組成read(0, .bss+0x80)的gadgets,
                           # 就加入rop chain
rop.raw('/bin/sh') # 將'/bin/sh'字串直接加入rop chain
```

ROPgadget

- $ ROPgadget --binary <binary>

```
> ROPgadget --binary helloctf --only 'pop|ret'
Gadgets information
============================================================
0x00000000004007ec : pop r12 ; pop r13 ; pop r14 ; pop r15 ; ret
0x00000000004007ee : pop r13 ; pop r14 ; pop r15 ; ret
0x00000000004007f0 : pop r14 ; pop r15 ; ret
0x00000000004007f2 : pop r15 ; ret
0x00000000004007eb : pop rbp ; pop r12 ; pop r13 ; pop r14 ; pop r15 ; ret
0x00000000004007ef : pop rbp ; pop r14 ; pop r15 ; ret
0x0000000000400648 : pop rbp ; ret
0x00000000004007f3 : pop rdi ; ret
0x00000000004007f1 : pop rsi ; pop r15 ; ret
0x00000000004007ed : pop rsp ; pop r13 ; pop r14 ; pop r15 ; ret
0x000000000040057e : ret
0x0000000000400779 : ret 0

Unique gadgets found: 12
```

THE END