自己的除錯器自己寫

Hacks in Taiwan Conference 2014

黃建忠 (from TAIWAN)

whoami

- 現任研究生
- Linux 愛好者，關注系統安全領域
- **不是**哪個資 (ㄏㄢˋ) 安 (ㄎㄜˋ) 組織或公司的成員
- 簡單說就是一個**路人** QQ

今天講什麼？

- 一點點的除錯器 (debugger) 實現原理
- Memory search
- Code injection
- **Dynamic library function hooking**

題目有點簡單……

今天的目標

- 本投影片提供足夠的資訊，希望讓各位看完後，都有能力自己實作相關技術

我們需要的東西

- 一台 Linux 機器 (64 位元) Linux Mint 17
- Python(2.7.6)
- 一點點的 c 語言和 intel x64 組語

Part 1

- 一點點的除錯器實現原理

除錯器是啥？

- 用來惡搞或蹂躪其他 process 的**遙控器**
- **GDB**,ollydbg..... 等

```
(gdb) info registers
rax            0xfffffffffffffdfc       -516
rbx            0x8e4010 9322512
rcx            0xffffffffffffffff       -1
rdx            0xfa0    4000
rsi            0x8      8
rdi            0xab89a0 11241888
rbp            0xab89a0 0xab89a0
rsp            0x7fffb0f42bd0   0x7fffb0f42bd0
r8             0x0      0
r9             0x5b93   23443
r10            0x0      0
r11            0x293    659
r12            0x7fe695fe4f00   140628335677184
r13            0xfa0    4000
r14            0x8      8
r15            0x1      1
rip            0x7fe697a4ba43   0x7fe697a4ba43 <poll+83>
eflags         0x293    [ CF AF SF IF ]
cs             0x33     51
ss             0x2b     43
ds             0x0      0
es             0x0      0
fs             0x0      0
gs             0x0      0
(gdb) detach
```

看起來很好很強大 ,BUT.....

- 我想找哪條指令或數值在 memory 的哪裡？
- 我想對別的 process 加很多料
- 我們需要一個客製 , 自動化的工具

你不會想一直拿著遙控器的 ...

自己的工具自己寫 !!

- 除錯器是怎麼運作的？
- 怎麼去改或讀其他 process 的 memory？

如何讀寫其他 process 的 memory?

- **ptrace** (linux 提供的一個 system call)
- **/proc/pid/mem** (kernel >= 2.6.39)
- process_vm_readv, process_vm_writev(kernel >= 3.2)

下斷點 (set breakpoint)

- 讓 process 停在想停的地方
- 寫入 int 3(0xcc) 指令
- 用 wait() 等待 trap 事件的發生

```
int main()
{
    asm("int $3");

}
```

```
00000000004004b4 <main>:
  4004b4:       55                      push   %rbp
  4004b5:       48 89 e5                mov    %rsp,%rbp
  4004b8:       cc                      int3
  4004b9:       5d                      pop    %rbp
  4004ba:       c3                      retq
```

執行後

```
Trace/breakpoint trap (core dumped)
```

用 wait() 來等待 trap

ptrace(1)

```
long ptrace(enum __ptrace_request request, pid_t pid,
            void *addr, void *data);
```

- 功能很多，但我們只有下列需求
  - → 讀寫 data 從 (到) 某個地址 (8 bytes in x64)
  - → Attach,continue,detach process
  - → 讀或寫暫存器的值

```
long ptrace(enum __ptrace_request request, pid_t pid,
            void *addr, void *data);
```

PTRACE_PEEKDATA
PTRACE_POKEDATA

→ 記憶體讀寫

PTRACE_ATTACH
PTRACE_DETACH
PTRACE_CONT

→ Process 控制

PTRACE_GETREGS
PTRACE_SETREGS

→ 暫存器讀寫

那用什麼語言寫？

- 我就是不想用 C 語言寫

- 人生苦短，請用 **python**

ctypes

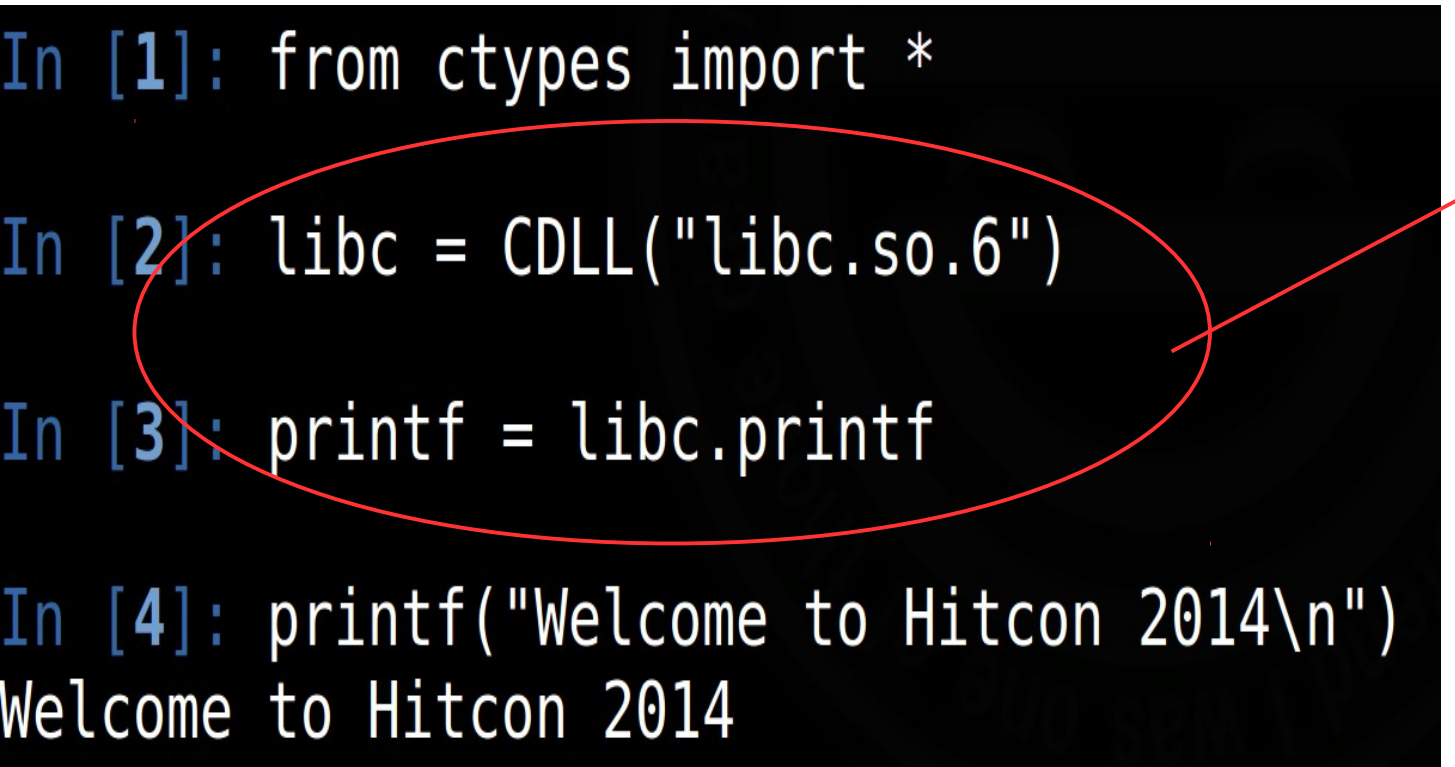
- python 的標準模組
- 在 python 用 c 的 library
- 要知道我們要的 function 在哪個 library

```
In [1]: from ctypes import *

In [2]: libc = CDLL("libc.so.6")

In [3]: printf = libc.printf

In [4]: printf("Welcome to Hitcon 2014\n")
Welcome to Hitcon 2014
```



printf 在 libc.so.6

ptrace,wait, 常用 function....

ctypes- 定義變數 (1)

在 python 定義 c 語言型態的變數

ctypes type	C type	Python type
c_char	char	1-character string
c_long	long	int/long
c_ulong	unsigned long	int/long
c_char_p	char * (NUL terminated)	string or None
c_void_p	void *	int/long or None

ctypes- 定義變數 (2)

```
In [1]: from ctypes import *

In [2]: x = c_ulong()

In [3]: x.value = 2014

In [4]: s = c_char_p("Welcome to Hitcon 2014")

In [5]: s.value
Out[5]: 'Welcome to Hitcon 2014'

In [6]: array = (c_int * 10)()
```

定義 Array

ctypes- 定義結構體

```c
struct user_regs_struct
{
  unsigned long int r15;
  unsigned long int r14;
  unsigned long int r13;
  unsigned long int r12;
  unsigned long int rbp;
  unsigned long int rbx;
  unsigned long int r11;
  unsigned long int r10;
  unsigned long int r9;
  unsigned long int r8;
  unsigned long int rax;
  unsigned long int rcx;
  unsigned long int rdx;
  unsigned long int rsi;
  unsigned long int rdi;
  unsigned long int orig_rax;
  unsigned long int rip;
  unsigned long int cs;
  unsigned long int eflags;
  unsigned long int rsp;
  unsigned long int ss;
  unsigned long int fs_base;
  unsigned long int gs_base;
  unsigned long int ds;
  unsigned long int es;
  unsigned long int fs;
  unsigned long int gs;
};
```

C

```python
class user_regs_struct(Structure):
    _fields_ = [
            ("r15", c_ulong),
            ("r14", c_ulong),
            ("r13", c_ulong),
            ("r12", c_ulong),
            ("rbp", c_ulong),
            ("rbx", c_ulong),
            ("r11", c_ulong),
            ("r10", c_ulong),
            ("r9", c_ulong),
            ("r8", c_ulong),
            ("rax", c_ulong),
            ("rcx", c_ulong),
            ("rdx", c_ulong),
            ("rsi", c_ulong),
            ("rdi", c_ulong),
            ("orig_rax", c_ulong),
            ("rip", c_ulong),
            ("cs", c_ulong),
            ("eflags", c_ulong),
            ("rsp", c_ulong),
            ("ss", c_ulong),
            ("fs_base", c_ulong),
            ("gs_base", c_ulong),
            ("ds", c_ulong),
            ("es", c_ulong),
            ("fs", c_ulong),
            ("gs", c_ulong)
            ]
```

Python

微 debugger


In [5]: debug.attach(2945) ; debug.wait()
Out[5]: 2945
In [6]: debug.bpset(0x7f4accca7c80)
In [7]: debug.cont() ; debug.wait()
Out[7]: 2945
In [8]: debug.getregs(regs)
In [9]: print hex(regs.rip)
0x7f4accca7c81L
In [10]: debug.bpclear(0x7f4accca7c80)
In [11]: debug.detach()

Part 2

- Memory search

Memory search

- 尋找特定**指令**或**數值**的位址
- 應用

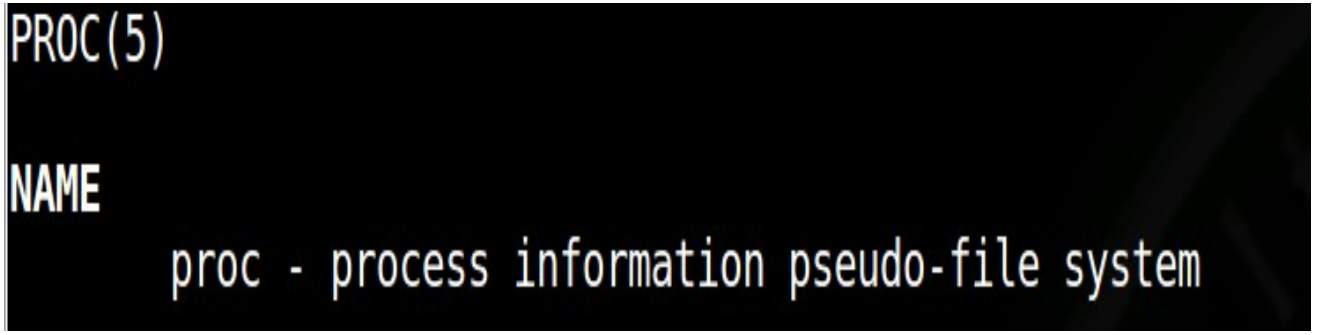
  → 找 ROP gadget ？

  → 遊戲修改器 ？

但是我後面都不是以上用途 .....

Memory search- 怎麼找

- man proc



- /proc/[pid]/maps （看地圖）

  → 收集相關資訊

- /proc/[pid]/mem

  → 從這邊找

/proc/[pid]/maps

r-xp code segment

r—p 和 rw-p data segment

載入 so

```
demo@demo-K43SV ~/example $ cat /proc/self/maps
00400000-0040b000 r-xp 00000000 08:06 4980759                            /bin/cat
0060a000-0060b000 r--p 0000a000 08:06 4980759                            /bin/cat
0060b000-0060c000 rw-p 0000b000 08:06 4980759                            /bin/cat
01693000-016b4000 rw-p 00000000 00:00 0                                  [heap]
7f8534128000-7f8534432000 r--p 00000000 08:06 2890232                    /usr/lib/locale/locale-archive
7f8534432000-7f85345ee000 r-xp 00000000 08:06 4725047                    /lib/x86_64-linux-gnu/libc-2.19.so
7f85345ee000-7f85347ed000 ---p 001bc000 08:06 4725047                    /lib/x86_64-linux-gnu/libc-2.19.so
7f85347ed000-7f85347f1000 r--p 001bb000 08:06 4725047                    /lib/x86_64-linux-gnu/libc-2.19.so
7f85347f1000-7f85347f3000 rw-p 001bf000 08:06 4725047                    /lib/x86_64-linux-gnu/libc-2.19.so
7f85347f3000-7f85347f8000 rw-p 00000000 00:00 0
7f85347f8000-7f853481b000 r-xp 00000000 08:06 4725035                    /lib/x86_64-linux-gnu/ld-2.19.so
7f85349f6000-7f85349f9000 rw-p 00000000 00:00 0
7f8534a18000-7f8534a1a000 rw-p 00000000 00:00 0
7f8534a1a000-7f8534a1b000 r--p 00022000 08:06 4725035                    /lib/x86_64-linux-gnu/ld-2.19.so
7f8534a1b000-7f8534a1c000 rw-p 00023000 08:06 4725035                    /lib/x86_64-linux-gnu/ld-2.19.so
7f8534a1c000-7f8534a1d000 rw-p 00000000 00:00 0
7fffed344000-7fffed365000 rw-p 00000000 00:00 0                          [stack]
7fffed391000-7fffed393000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
```

記憶體區段位址範圍

也包含 heap 和 stack 相關資訊

/proc/[pid]/mem(1)

- Process 的 memory 內容
- 可當作 file 存取 (open,read,lseek)

```
/proc/[pid]/mem
       This file can be used to access the pages of a process's memory through open(2), read(2), and
       lseek(2).
```

- 需要先用 ptrace attach 後才可以存取

需要 ROOT 權限

/proc/[pid]/mem(2)

- 在 python 怎麼存取？

```
memory = open("/proc/"+pid+"/mem","r", 0) # 0(zero) means no buffering

memory.seek(start_address) # 從哪裡開始讀

code = memory.read(1) # 讀一個bytes
```

- 就像處理普通的 file

/proc/[pid]/mem- 範例

```
def searchmem(self,opcode,attr = "r-xp",segment = ""):
```

“48 83 c4 08 c3”
複製貼上就好

相關訊息都由 proc/[pid]/maps 取得

```
400641:       48 83 c4 08                 add    $0x8,%rsp
400645:       c3                          retq
```

/proc/[pid]/mem- 執行

```
In [1]: import mydebug

In [2]: debug = mydebug.debug()

In [3]: debug.attach(13283)

In [4]: debug.searchmem("48 83 c4 08",segment = "out")
['48', '83', 'c4', '08']
search between 00400000 and 00401000
[*] found at 0x400413
[*] found at 0x40049e
[*] found at 0x40050d
[*] found at 0x40062f
[*] found at 0x400641
```

Part 3

- Code injection

Code injection

- 把指令寫入到目標 process 的 memory

  → 白話一點，給目標 process 加點料

- 以**注入 so** 為範例

在 windows 這叫 DLL

```
7f8534432000-7f85345ee000 r-xp 00000000 08:06 4725047    /lib/x86_64-linux-gnu/libc-2.19.so
7f85345ee000-7f85347ed000 ---p 001bc000 08:06 4725047    /lib/x86_64-linux-gnu/libc-2.19.so
7f85347ed000-7f85347f1000 r--p 001bb000 08:06 4725047    /lib/x86_64-linux-gnu/libc-2.19.so
7f85347f1000-7f85347f3000 rw-p 001bf000 08:06 4725047    /lib/x86_64-linux-gnu/libc-2.19.so
```

- 唯一用 C 語言和 x64 組語的地方 (只有一點點 ...)

要生出一個 so

注入 SO 能幹啥？

- Hotpatch (不用重啟程式就可完成更新)
- **function hooking**
- 作防毒軟體！？

順便一提，有些**惡意軟體**也採用類似的技巧

如何注入 SO？

- 既然都可以用 ptrace 控制目標的暫存器了 .....

  → 讓目標 process 執行載入 so 的 api 即可

IDEA 源自 BLACK HAT 2001 injectso

- 哪一個 API？

  → __libc_dlopen_mode (位在 libc.so)

```
void * __libc_dlopen_mode (const char *name, int mode)
```

參考 glibc-2.19 的原始碼

如何注入 SO - 找位址 (1)

- 因為 ASLR(Address space layout randomization)

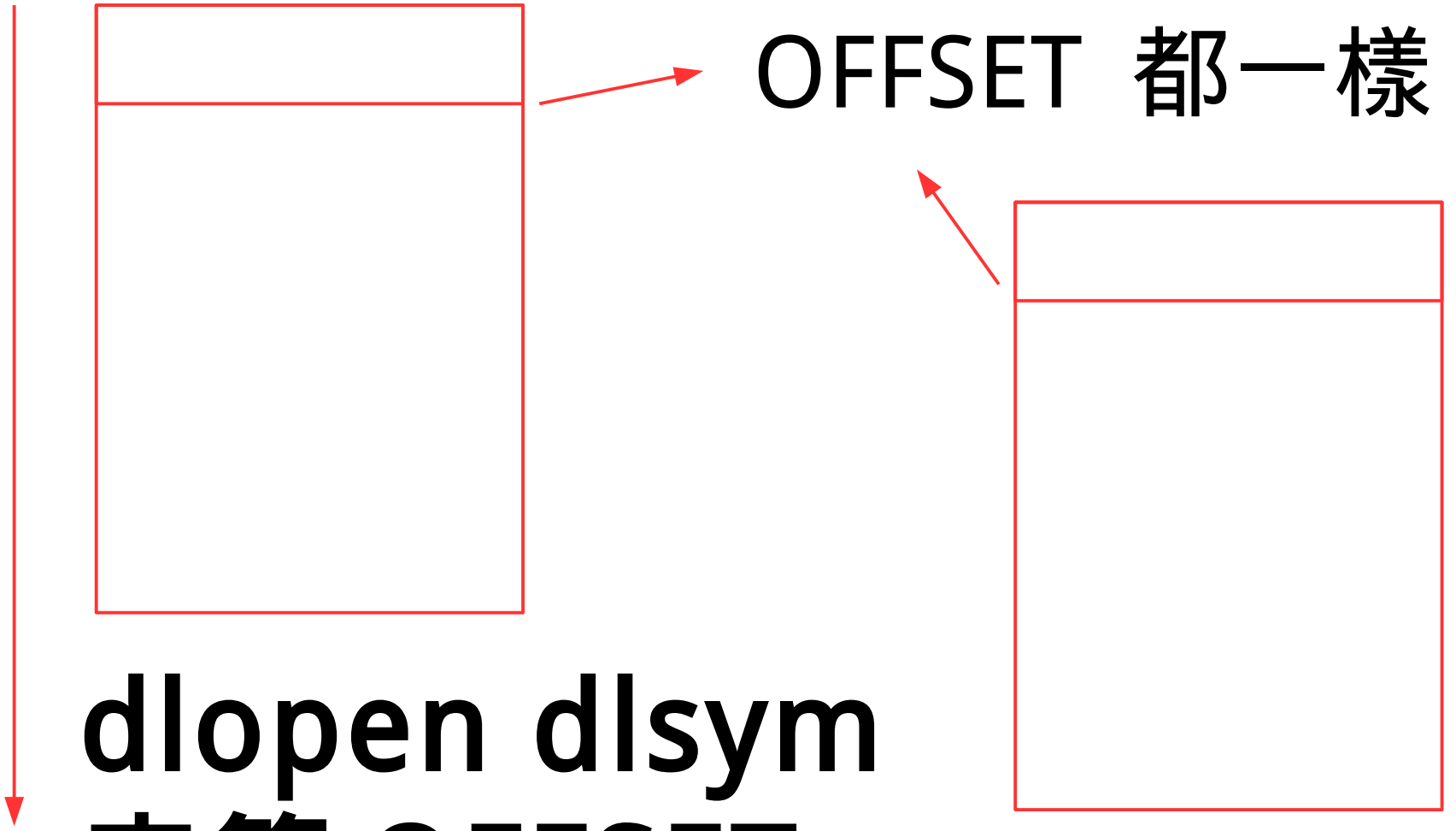
  → 每次載入 SO 的位址都不一樣

```
In [3]: os.system("ldd ./a.out")
        linux-vdso.so.1 =>  (0x00007fff7b73c000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f1907dd5000)
        /lib64/ld-linux-x86-64.so.2 (0x00007f19081b9000)
Out[3]: 0

In [4]: os.system("ldd ./a.out")
        linux-vdso.so.1 =>  (0x00007fffa0396000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007fef9b1ed000)
        /lib64/ld-linux-x86-64.so.2 (0x00007fef9b5d1000)
Out[4]: 0
```

- 但是 **OFFSET** 不變

  → 所需 API 的位址**相對位址**都一樣



dlopen dlsym 來算 OFFSET

如何注入 SO - 找位址 (2)

```
def find_hook_point(self,pid,libname,funcname):
```

```
In [1]: import mydebug

In [2]: debug = mydebug.debug()

In [3]: debug.find_hook_point(8397,"libc-2","__libc_dlopen_mode")
[*] get remote start addr 7f951914b000
[*] get remote libpath /lib/x86_64-linux-gnu/libc-2.19.so
my pid is 8529
my_start 7f0f07722000
mylibpath /lib/x86_64-linux-gnu/libc-2.19.so
[*] get offset 0x137070
[*] target function __libc_dlopen_mode
[*] hook point 0x7f9519282070
```

自己也載入相同的 SO 來算 offset

+

參考目標 /proc/pid/maps so 的起始位址

如何注入 SO - 準備參數

- 在 Intel x64，是如何傳遞參數的？

```c
#include<stdio.h>
int hit(int a1,int a2,int a3,int a4,int a5, int a6)
{
    return a1+a2+a3+a4+a5+a6;
}

int main()
{
    asm("nop");
    asm("nop");
    asm("nop");
    asm("nop");
    asm("nop");
    hit(1,2,3,4,5,6);
    asm("nop");
    asm("nop");
    asm("nop");
    asm("nop");
    asm("nop");

}
```

```
push   %rbp
mov    %rsp,%rbp
nop
nop
nop
nop
nop
mov    $0x6,%r9d
mov    $0x5,%r8d
mov    $0x4,%ecx
mov    $0x3,%edx
mov    $0x2,%esi
mov    $0x1,%edi
callq  0x4004b4 <hit>
nop
nop
nop
nop
nop
pop    %rbp
retq
```

把參數丟到暫存器
(32 位元是丟到 stack)

如何注入 SO - 實現

- 準備一個要注入的 so

```
In [3]: debug.attach(8397)

In [4]: debug.inject("/tmp/hitcon.so")
[*] get remote start addr 7f951914b000
[*] get remote libpath /lib/x86_64-linux-gnu/libc-2.19.so
my pid is 8676
my_start 7f3afe9bb000
mylibpath /lib/x86_64-linux-gnu/libc-2.19.so
[*] get offset 0x137070
[*] target function __libc_dlopen_mode
[*] hook point 0x7f9519282070
we'll execute at 0x7f9519282070L
control again.....because of 0x0 return address

demo@demo-K43SV ~/hooking $ cat /proc/8397/maps
00400000-00401000 r-xp 00000000 08:07 9175230                            /home/demo/example/a.out
00600000-00601000 r--p 00000000 08:07 9175230                            /home/demo/example/a.out
00601000-00602000 rw-p 00001000 08:07 9175230                            /home/demo/example/a.out
01e68000-01e89000 rw-p 00000000 00:00 0                                  [heap]
7f9518f49000-7f9518f4a000 r-xp 00000000 08:06 3938272                    /tmp/hitcon.so
7f9518f4a000-7f9519149000 ---p 00001000 08:06 3938272                    /tmp/hitcon.so
7f9519149000-7f951914a000 r--p 00000000 08:06 3938272                    /tmp/hitcon.so
7f951914a000-7f951914b000 rw-p 00001000 08:06 3938272                    /tmp/hitcon.so
```

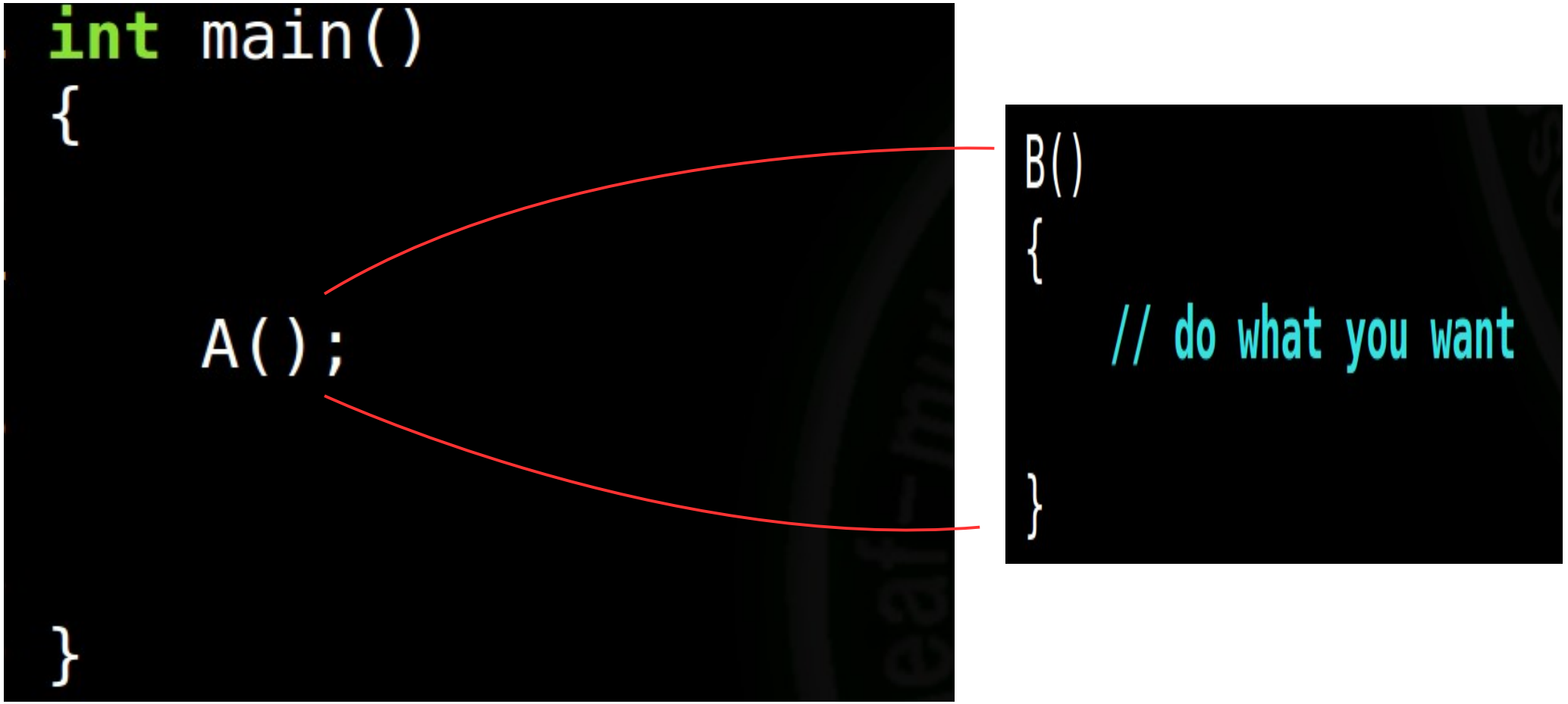
Part 4

- Hooking

function hooking(1)

- Hooking 是什麼？

  → 攔截或替換 process 的某一個 function 的一種**技巧**



function hooking(2)

- 那我用 hook 作什麼事 ？
  - → 統計某個 function 被呼叫了幾次
  - → 替換或監視這個 function 的參數內容

不是這樣就搞定了嗎 ？ **BUT.....**

1. 下斷點
2. 看參數或統計斷點觸發次數
3. 移除斷點，讓目標恢復執行

function hooking(3)

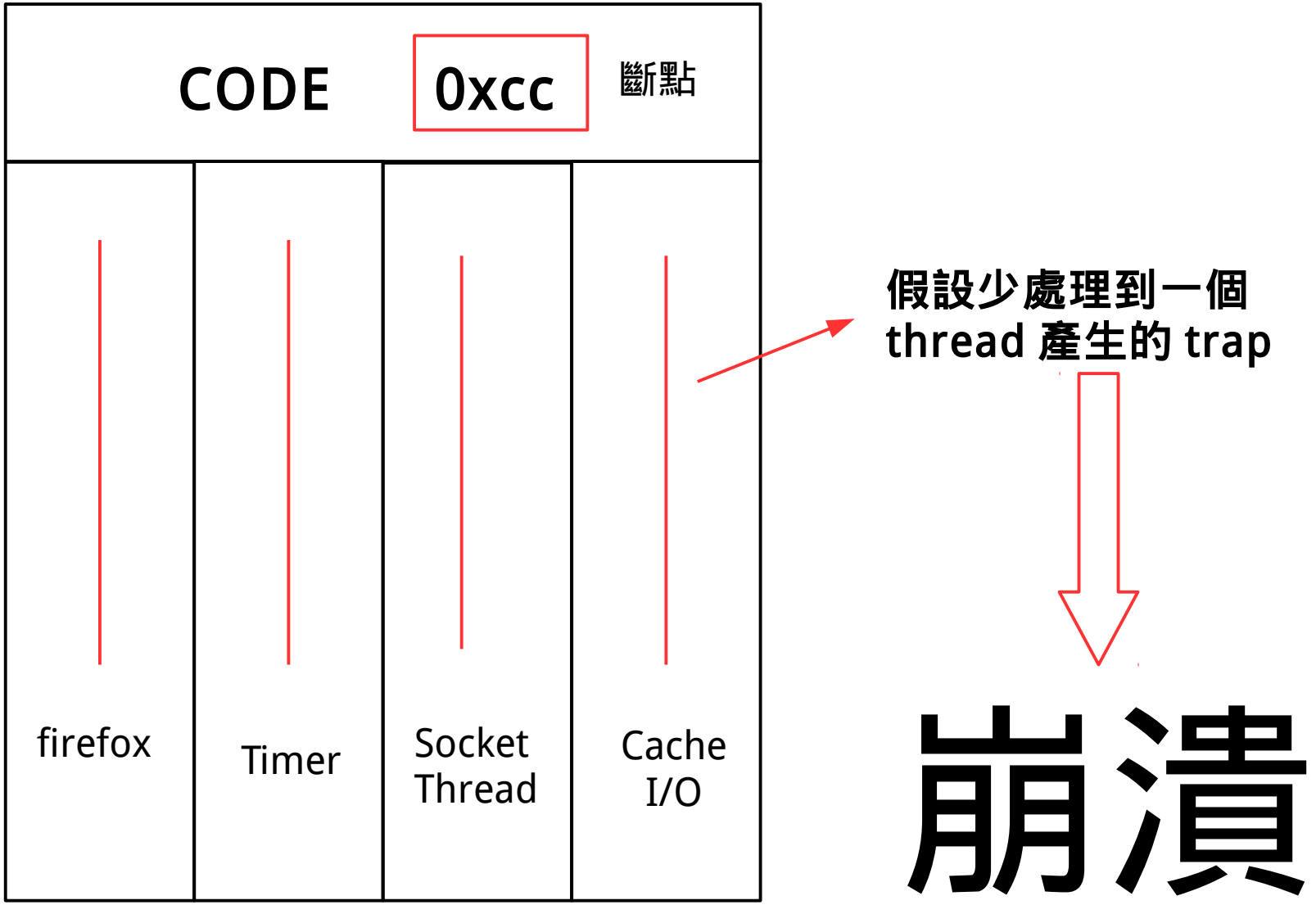
- 如果目標是 multi-thread 性質的程式
- 如果目標 function 會被大量呼叫

→ 瀏覽器符合以上特性（後面以 firefox 為例）

可～～～能～～～會～～～很～～～慢～～～

1. 下斷點
2. 看參數或統計斷點觸發次數
3. 移除斷點，讓目標恢復執行

有時候目標還會崩潰 QQ


CODE
0xcc
斷點
firefox
Timer
Socket
Thread
Cache
I/O
假設少處理到一個
thread 產生的 trap
崩潰

放張崩潰圖

HOOK 前注意事項

- 目標

  → 不要讓目標崩潰 (crash)

  → 對目標執行速度影響要小

  → hook 方法要容易實現

HOOK 方法

- 採用的方法

  → 1. 先注入 SO (前面我們已經會了)

  → 2. 讓要 hook 的 function 跳到注入的 so 裡

  → 3. 再從注入的 so 跳回去原本的 function

把前面的功能拼一拼就可實現

hook 範例

- 一個非常簡單的例子

```c
#include<stdio.h>
#include<unistd.h>
int main()
{
    char msg[30] = "I'm sleeping.....\n";
    while(1) {
        puts(msg);
        sleep(1);
    }
    return 0;
}
```

- 目標

  → 攔截 puts，讓它不要在睡了

```
I'm sleeping.....
I'm sleeping.....
I'm sleeping.....
```

如何改變執行流程 (1) ？

- 先來看點組合語言（在呼叫 puts 之前）

```
0x000000000040058c <+72>:    lea    -0x30(%rbp),%rax
0x0000000000400590 <+76>:    mov    %rax,%rdi
0x0000000000400593 <+79>:    callq  0x400430 <puts@plt>
0x0000000000400598 <+84>:    mov    $0x1,%edi
```

跳到 0x400430 位址

```
(gdb) x/3i 0x400430
   0x400430 <puts@plt>: jmpq   *0x200bca(%rip)        # 0x601000 <puts@got.plt>
   0x400436 <puts@plt+6>:       pushq  $0x0
   0x40043b <puts@plt+11>:      jmpq   0x400420
```

跳到 0x601000 所存的位址

```
(gdb) x/2x 0x601000
0x601000 <puts@got.plt>:    0x00400436    0x00000000
```

如何改變執行流程 (2) ？

- 還是再來點組合語言（在呼叫 puts 之後）

```
0x000000000040058c <+72>:    lea    -0x30(%rbp),%rax
0x0000000000400590 <+76>:    mov    %rax,%rdi
0x0000000000400593 <+79>:    callq  0x400430 <puts@plt>
0x0000000000400598 <+84>:    mov    $0x1,%edi
```

跳到 0x400430 位址（跟剛剛一樣）

```
(gdb) x/3i 0x400430
   0x400430 <puts@plt>: jmpq   *0x200bca(%rip)        # 0x601000 <puts@got.plt>
   0x400436 <puts@plt+6>:       pushq  $0x0
   0x40043b <puts@plt+11>:      jmpq   0x400420
```

值不一樣了，這是 puts 的實際所在位址

```
(gdb) x/3x 0x601000
0x601000 <puts@got.plt>:        0xf7a8ace0      0x00007fff
(gdb) p puts
$1 = {<text variable, no debug info>} 0x7ffff7a8ace0 <puts>
(gdb)
```

如何改變執行流程 (3) ？

- Linux 的動態連結實現機制

  → plt → got → 實際 function 的所在位址
- 其實這樣作是有道理的 ... 但是先不管它

- 只要改寫 GOT 表格的值即可搞定

  → 用 memory search 的功能找要改的值在哪？

如何改變執行流程 (4) ？

- 那注入的 SO 要做什麼？
  - → 備份所有的暫存器
  - → 做你想做的事
  - → 還原所有的暫存器
  - → 跳回原本被 hook 的 function 位址
- 怎麼跳回去
  - → push Address → ret

都使用組合語言

64 位元要迂迴一點，不能直接 push 8bytes 的值到 stack

注入的 SO 長這樣

```
int hello(char *buf)
{
    backup();

    // do something
    char str[30] = "Welcome to HITCON 2014";
    printf("Hooked %d\n",i);
    strncpy(buf,str,30);
    i++;

    restore();

    asm("leave"); // mov %rbp,%rsp , pop %rbp
    // backup r15
    asm("sub $16,%rsp");
    asm("mov %r15,(%rsp)");
    asm("add $16,%rsp");

    // 跳回去
    pattern();
    asm("mov $0x4141414141414141,%r15");
    asm("push %r15");

    // restore r15
    asm("sub $8,%rsp");
    asm("mov (%rsp),%r15");
    asm("add $8,%rsp");

    asm("ret");
}
```

static inline function
把暫存器丟到 stack
備份 or 還原

先用 memory search 找到後改為被 hook 的位址

DEMO

```
I'm sleeping......

I'm sleeping......

I'm sleeping......

I'm sleeping......

I'm sleeping......

Hooked 0
Welcome to HITCON 2014
Hooked 1
Welcome to HITCON 2014
Hooked 2
Welcome to HITCON 2014
Hooked 3
Welcome to HITCON 2014
Hooked 4
Welcome to HITCON 2014
```

以 Firefox 為例

- 要攔截哪一個 function？

  → PR_Write 位在 libnspr4.so

```
/*
 ************************************************************************
 * FUNCTION: PR_Write
 * DESCRIPTION:
 *     Write a specified number of bytes to a file or socket.  The thread
 *     invoking this function blocks until all the data is written.
NSPR_API(PRInt32) PR_Write(PRFileDesc *fd,const void *buf,PRInt32 amount);
```

- 為什麼攔它 ？

  → 會被許多 thread 大量呼叫，很好的測試點

  → HTTP 的 GET,**POST** 請求

可以攔到神奇的東西

DEMO

結論

- Thanks

参考資料

- 1.injectso
  → https://github.com/ice799/injectso64
- 2. playing with ptrace
  → http://www.linuxjournal.com/article/6100
- 3. python ctypes
  →https://docs.python.org/2/library/ctypes.html