Day 4 - Reverse Engineering

OT
Binary, Exploitation, Pwning - Sean

WJ Slide: http://10.73.3.210/files/8/Binary%20Exploits.pdf

S Twitch (today): http://www.twitch.tv/seanwu/

WU J *需要放一份 Slide 的Internet連結嗎?*

SYA *(y)*

MIAOSKI *+1*

ИССЛЕДО… *+2*

M
demo 1

DH 先洩漏任何一個got.plt欄位
使用現有的puts()
Function call 的 ROP.....

M
```
$ ncat -vc ./vul -kl 127.0.0.1 8888
$ nc -v localhost 8888
```

昨天

1. return to puts()
2. leak base address
3. back to main()
4. exploit

要送複數個 ROP chain 的話，我們需要一個新的方法 -- ROP 之 Combo 技巧

AL Stack Migration

- gadget: leave; ret;
- leave = mov esp, ebp; pop ebp;

ZET T *pop reg = mov reg, ESP; add ESP, 4*
push data = sub ESP, 4; mov data, ESP
enter N bytes= push EBP; mov EBP, ESP; sub ESP, N bytes (required for locals)
call func = push EIP; jmp func address
leave = mov ESP, EBP; pop EBP
ret = pop EIP; jmp EIP

M 我們一般會把它搬到 global variable 的地方，因為 stack 會受 ASLR 影響。

先跳到 pop ebp; ret 的 gadget 可以讓我把 ebp 的值設定好。

AL rop2(stack migration 之後) 的位置會多4個bytes，因為ebp會被pop掉

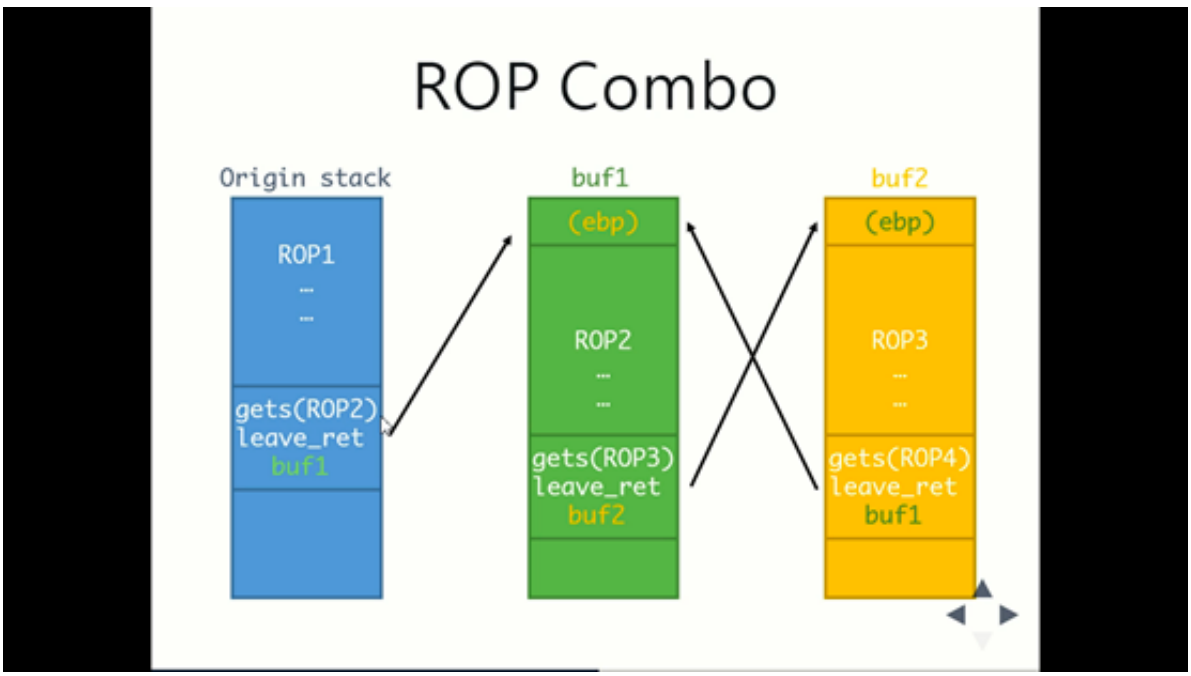
MIAOSKI *這一段好噁心，跪求整理*

馬聖豪 *跪求整理+1*

M leave_ret 加上 pop 總共也才 10 幾個 bytes 所以我們可以多放一些 chain.
一開始在 origin stack overflow 要送的 payload 是 ROP1, 我要找到 gets(ROP2), leave_ret, buf1 要是 block I/O 才行
跳過去就是 stack 搬過來，pop 掉第一個，它就會緊接著 ROP2 繼續往下跑
ROP2 我可能用 puts() leak 某些東西，或許還不夠，再多 leak 一些東西
比如說你想要知道 stack 的位置，libc 裡面有一個 symbol 叫 __libc_stack_end
啊我現在 share library 還沒 load 進來... 要先跑起來。 (gdb) p/x __libc_stack_end
昨天講 puts() 你可以 leak libc 的 base, 也可以 puts(__libc_stack_end) 拿到 stack 位置
後面還是要接一個 gets(ROP3) 把 ROP3 (buf2) 讀進來
你不能把它讀到 buf1 自己，因為你做到一半它會把 stack 洗掉，它會壞掉
所以你就找到兩個有足夠空間的 buffer 讓它可以再跳回 buf1

CF
- 如果 ROP2 中間不會用到 ebp，可以先把 ROP3 要放的位置先寫在 ROP2 上面 （(ebp) 這邊），這樣 ROP1 下面的 leave_ret 的時候就會把它 pop 出來，ROP2 下面的 leave_ret 就會把 stack 搬到那個位置。
- 另一個做法是在 ROP2 上面的 (ebp) 放垃圾，下面多接一個 pop ebp; ret 的 gadget。

X



M 通常你會希望 buffer 愈大愈好，不然爆掉會 crash
cat /proc/`pidof vul`/maps 找 rw-p 像 0804a000-0804b000 這邊都是可以對齊的，

保證後面不會有人用到。我儘量放低一點，是希望它 stack 往上長的時候不要卡到其它東西。function call 的時候它可能會用到 data 段的一些東西，你就儘量放後面一點。
(第21-22行) 昨天的例子是 116 bytes, 然後第一個位子就是要 return 的地方
A*112 + p32(buf1) 一樣是 116 bytes 沒有問題。後面要接的就是 ROP chain
我的 ROP1 是空的，其實沒有做任何事情，要做的就是 gets(ROP2)
main return (main+130) 前它有一個 leave ; ret ，它其實會帶一個 pop ebp
我們要 migrate 所以要控制好 ebp 把它寫成 buf1 再 return.
leave 的時候會把 p32(buf1) pop 到 ebp 去
stack 搬過來之後... 我等一下要做的事情叫做**無限次數的 information leak**
那個 puts() call 一次之後它 stack 會偏一點點，你不能一直往下 call 不然最後 stack 會用完。

AL 原本main是push過來的，可是我們是return過去，所以會少幾個byte

M (行 25-34) 行37第一次 call leak #22 的 gets 其實在等我。
#27 swap 是為了保持我的 ROP chain 要跑在 buf1 上面。它是交換 buf1 和 buf2 就對了
最開始的 pop ebp 會把 #28 的 buf1 pop 出來，
#29 是利用 puts 把記憶體這個地方印出來
#30 再接上 gets 和 levae_ret 把新的 ROP chain 讀到 buf1 上面，然後再跳過去
#36 加上.. print enhex(leak(puts_got))
#37 加上 r.interactive()
你會發現它的 connection 沒有斷，它還在等我。如果我再輸入一些東西它就會斷掉，因為把會把我打的東西當成 ROP chain.
如果我要 leak 其它的東西，假設 addr+1 好了... #36 加 print enhex(leak(puts_got)+1)
如果我之後想要 call system() 我就送一個 call system() 的 ROP 給它
這樣比較簡單，你就不用再考慮...

昨天最後幾分鐘前的 exploit 我把位址都寫死了，這些 offset 全都寫死了，這些是 libc 裡面的 offset. 那如果 DB 裡面查不到怎麼辦，如果不知道對方 libc 版本或 symbol offset 的時候該怎麼辦。
我們剛有個 leak() 可以到處 leak, 問題是 puts() 遇到 \0 會結束
你可以把整個 libc 印出來，然後載下來，這樣可以整個 dump 下來，可是這樣比較久。

```
readelf -a
```

dynsym 和 dynstr 和找 symbol 有關
pwntools: DynELF
information leak 不止有一種方法，format string 一樣可以
你要 return 的東西就是給它一個字串 `d = DynELF(leak, a_ptr_in_libc)`

a_ptr_in_libc 一定要在 libc 裡面某處
比方說我先 leak 出 got 的位置，然後求得 base addr, 在你不知道對方 libc 的版本的時候，你不知道 offset 之多少。所以它每次往前移動一個 page 看有沒有 magic number.
shared library load 進來的時候它和 binary 一樣的，它整段是連續的。
如果你有一個 pointer 在裡面的話，你每次往前找一個 page 一定會找到 ELF header。
接下來的部份比較麻煩，我沒有很想講這個地方 (咦)

ALEX L *https://github.com/Gallopsled/pwntools/blob/master/pwnlib/dynelf.py*

M ~/a/d/p/a.py
#36 我要先找到一個 libc 裡面的 pointer 我們先 leak 一個 got 上的 entry 就好 我們不用去算 base 是多少。
再來就 DynELF(leak, ptr_puts)
你會發現它一個個 page 往前找。這要找一下下，因為我 delay 設比較久。
它找到後會去找 PT_DYNAMIC 這個 section 它還會去找 hash table
然後就找到 system() 在哪個地方 (#41)
你找到後就可以直接跳過去。
我要用 system() 開一個 shell 條件是你要放 /bin/sh 給它。我們昨天在 libc 裡面找，今天我們自己送這個 command 給它 (#46)
我現在送過去的 data 會被當成新的 ROP 跑起來 (#29)
#44 這個不重要，所以我隨便送一個 0 給它
#45 是 return to function 所以我下一個要疊的是 return addr 不過不重要，所以我也填 0 再下一個 buf1+0x10 為什麼呢？這個其實是指到 /bin/sh 字串的開頭
#46 是 system 的參數，我要填什麼都可以. 比方說 ls -la ; sh
delay 設太短它會不穩。
你可以把 offset 剪貼下來，記下來，因為它應該不會每次都變位置。
#46 gets() 會自己補 0 所以 '\n' 會變成 \0

GOT Hijacking 改完 entry 之後要再 call 一次 function
有個限制是 gcc 有個選項，它會把 got 變成唯讀。
.got 的話它是唯讀的，但 .got.plt 就是可讀可寫。

ALEX L *gcc -z relro*

M **demo2**
format string 我沒有很想講，因為它快要消失了，它是歷史遺毒，之後的 libc 可能會拿掉。
exit(0); main 不會 return
printf(str) 平常這邊沒事，你不見得會發現這個 bug. 但是... 我一輸入 %s 它就壞掉了

因為它不是 string 它是 format string. 所以可以打 %x %x %x %x %x %x %4$x %x %7$x %x 這是比較進階的用法. %4$x 是印 arg4.
所以它為什麼會 crash 因為 %s 想把 c8 這個位置印出來，但是它在外面，所以 segfault 。整個 stack 的東西都可以 dump 出來，不過你不能往上... %-1$x 不行 %0$x 也不行。
%s 比較好，你可以指定任意 addr.

ИССЛЕДО… *(題外話:*
format string %s,%x %n 都是好物
補個wiki https://www.owasp.org/index.php/Format_string_attack
and http://www.cgsecurity.org/Articles/SecProg/Art4/)

QUEENIE W *早安大大*

ИССЛЕДО… *台灣第一女駭客早安*

MIAOSKI *樓上黑黑*

ИССЛЕДО… *樓上黑黑*

馬聖豪 *樓上黑黑*

ZHEANLIN *樓上黑黑*

IL 路人補充： %p 很多時候比 %x 更好用

M ~/demo2/a.py
條件是 char str[200] 要在 stack 上面，不能在 global.
我們要先找到 buffer 在 format str 的第幾個參數。偷懶的方式是這樣:
AAAA %x %x %x %x %x %x %x
你找到 41414141 就知道 ... 比方說第五個參數就是我的 buffer. 所以 %5$x 會剛好印到 buffer 的內容
這個趴趴是一個 % AAAA %%%d$x

比方說我要看 printf() 的 GOT 上的內容
#18 改成 print fmt(p32(printf_got) + '%5$x')
它會印出 \x0c\xa0\x0804a00c 可是我要的不是那個數字，我要那個 address 到底是什麼。所以要改成 %5$s 把所在的字串印出來。

```
print enhex(fmt(p32(printf_got) + '%5$.4s'))
```

這樣它印出來的是 printf 在 got 的值
我們可以和 `readelf -s libc.so.6 | grep 'printf@'` 對照一下

ИП 注意印出的資料是以little endian呈現,

M 用這樣的 format string 就可以去讀任意地方，所以我們可以寫一樣的 leak()
format string 不止可以讀，還可以寫... 用法是~~趴恩~~ %n

我們隨便寫點東西好了

```
#include <stdio.h>
int main() {
  int a;
  printf("1234%n\n\n", &a);
  printf("%d\n", a);
}
```

我們來看一下最後 a 變成什麼。
%n 的意思是前面印了幾個字元，然後把它寫到後面的變數。
這可以搭配剛剛的 $ 使用

```
printf("1234asdfasdf%2$n\n", &a, &b);
```

%2$n 會寫到 b 去
利用 %123c 控制輸出的長度，123 個字元，前面補空白，再使用 %hhn 寫入一個 byte 到目標位址。 %hn 寫 2bytes, %hhn 寫 1 byte. 如果我要寫 4 個 bytes 我就寫四次。

system 的位址有了，我要把 printf() 改掉，讓它變成 system()
變成你輸入什麼東西，它就跑那個 command.
你改完一個 byte 它那個 printf() 就壞了，所以你要一次改 4 bytes.
首先我要放 4 個 addr 起來，如果邊做 format string 邊放 addr 會比較麻煩。通通放在最前面的話，addr 剛好是 5$, 6$, 7$, 8$
#26-#32 你要先算好你之前印了多少東西，把它扣掉。所以 printed 我設成 16。
%hhn 它會輸出的東西是印出的字元 mod 256

P 當你pad長度是0的時候不要再印 (不然會踩到地雷, %0c會印出一個字元)

J

M

```
fmt(p32(printf_got) + '%100c' + '%5$hhn')
```

我執行到這個 hhn 的時候它已經印了 104 bytes. 所以它會寫 104 進去

IL

```
$ pidof vul  # 也可以用 pgrep
```

M

```
$ gdb
```

ИП

```
(gdb) dt 那個pid              (attach process)
```

M

```
(gdb) b *0x8048546            (下斷點)
```

ИП

```
(gdb) c                    (continue, 然後就會執行到斷點)
(gdb) x/wx $esp            (這邊就是我們的buffer)
```

M

```
(gdb) x/s 0xff8a3e74
```

P

```
(gdb) ni                   (然後就壞掉了  >/////<)
```

M

```
(gdb) x/wx 0x804a00c          (=> 0xf762c280 等一下 LSB 那個 80 會變換掉)
(gdb) ni
```

(gdb) x/wx 0x804a00c (被換掉惹 => 0xf762c268)

一次改 4 bytes 再重新 attach

(gdb) x/i 0xf761f190 (耶變成 <system> 了)

(gdb) c

ИП

ИССЛЕДО… *下課... ya!!!!*

M **Heap Exploitation**

我沒預期這邊有多少人會聽懂，不過聽聽，大家知道有這樣的東西。
我不會在課堂上做這個東西，因為要做很久 XD
不行，這個要講，好煩 XD
不管有沒有問題我們都要繼續...
我們這邊不講，這邊太複雜 XD
當然會 crash 嘛 ，不會 crash 我就不用講了 XD

MIAOSKI *我會不會被 sean 打死 XD*

AWEIMEOW *大大打逐字稿辛苦了 <(_ _)>*

BLUET *大大打逐字搞辛苦了 m(_ _)m*

ИССЛЕДО… *打死一個還有千千萬萬個.. (逃*

M 32bit 帶 debug symbol 的 libc 記得把 source code 解到 /src/glibc-2.19
它會用一些很複雜的資料結構把 free() 的存起來。之後 malloc() 再拿出來用。
它會有一些資料結構和一些技術來減少 fragmentation
chunk = alloc 的記憶體 + meta data (投影片上寫的是 info)

ИП chunk header:

M 你 free 的時候沒告訴他 size 它怎麼知道要 free 多少東西 --> 看 INTERNAL_SIZE_T size
它其實是一個雙向鏈結串列
heap 的實作超髒，這就是為什麼它有一堆洞可以用
它會 align 到 16-byte (64bit) 或 8-byte (32bit) 傳回來的是 mem_ptr

ИП 我會不會講太快..講太快就算了 (眾人大笑),不然我會講不完

J chunk + size = next chunk

ИП 不清楚可以去trace malloc() code

M malloc() 就是從 chunk list 拿出來 free() 就是塞回去
原則上 **chunk 在記憶體中是連續的**

鄭達達 *多次 malloc or free or realloc 之後才會有間隔*

ИССЛЕДО… *一開始chunk在記憶體是連續的, 然後經過多次不人道的malloc, free, realloc 之後, 就開始出現間隔 .*

M 前一個 chunk 的 user data 在 prev_size 所以 overflow 你會寫掉下一個 chunk 的 prev_size 和 size.
free 時它會檢查前後兩個 chunk 是不是 inuse 如果不是就合併
size 最低的 3bit 有特殊用途，請看投影片，要小心蓋這些東西
chunk 裡的 prev_size 是為了推算前一個 chunk 的 metadata 在哪
這邊有一件奇妙的事情叫 unlink() -- 從雙向鏈結串列裡移除一個 chunk
下面是檢查 next chunk 是不是最後一個 chunk.
最後一個 chunk 要長多大就可以多大
malloc 要多大，你的 exploit 要走哪個 path 每種 bin 都不一樣。
fastbin 是特別小的 chunk. fastbin 跟別人不一樣，它不會參與 chunk 合併，而且是單向鏈結串列。
malloc 時會直接拿 bin 裡的第一個。
看 #3363 行

我 heap 講完囉，你聽不懂就算了 XD
這個 bin 怎麼運作不重要， Use After Free 只和一件事有關... malloc & free

Use After Free

prison/prison.c
p free 掉之後不會出事，等到下一個人用到同一塊記憶體的時候才會出事 (?)

J 如果有人malloc了一塊跟free一樣大小的記憶體，就會有兩個pointer指在同一個chunk上

M 上面如果有 vtable 的話，很不幸你會蓋掉 vtable, 有人 call 就會跳到別的地方去。
#211 行有個 free()
prisoner_s 是一個單向鏈結串列，它 #21 指向下一個. 裡面的 struct 都是 malloc() 出來的。它 free() 之後沒有從 linked list 裡拿掉.
你輸入一些東西，等一下 list 的時候它就會 crash

```
> punish
Cell: 9
> size
36
> note
AAAAAAAAAAAAAAAAAAAAA
> list
segmentation fault
```

note 先指到你要寫的地方，然後再輸入 data .

Heap Buffer overflow

M `-fno-stack-protect`
有 stack guard 的話你就不能 buffer overflow
繞過 stack guard 的方法... 沒有。除非你先有 information leak 你可以得到一些資訊才能繞過去。不過如果只有一個 buffer overflow 大概沒辦法。

LAYS *如果程式是fork出來的，可以每次overflow 1 byte去蓋stack canary，依據程式是否crash去爆出stack canary*

M heap buffer overflow 的好處是沒有 stack guard 這種保護。但 glibc 裡面有各種 sanity check.
舉個例子，unlink() 十年前可以 exploit. 以前只有4行，現在變得很複雜。
get(p) 的時候它會蓋掉 q, 所以 free(q) 的時候會壞掉。
真的跑起來它不會 crash 它會告訴你有東西壞掉了 (double free or corruption)
你並沒有辦法做到控制 EIP 或什麼的。

那 unlink() 怎麼 exploit
以前... 我現在有個 chunk 它有 fd, bk, 那 overflow 的時候可以把 fd bk 兩個蓋掉
它的 linked list 就不是原本的，可以指到任何你想指的地方.
所以這邊就會有一個對任何地方的寫入 (FD->bk = BK; BK->fd = FD)
你可以把其中一個位址指到 shellcode 上，free 的時候就會跑去 shellcode

J 現在unlink會檢查，保證他是double linking list，就沒辦法一個指shellcode 一個指向ret addr

M 但是 unlink 在某些特殊的條件下還是可以繞過。給你們10秒鐘的時間想一下，我喝個水。XD
unlink() 要保證你 double linked list 沒有壞掉，指過去還是要指回來。
(投影片)等一下我要 unlink(P) 如果你 overflow 的時候先改掉 prev_size 的話，它就會以為前一個 chunk 很遠。free() 的時候它就可以和一個很遠的 chunk merge 起來。
它原本是 free chunk 的話就要 buffer overflow，如果不是的話他有可能是data buffer就直接改掉...(?)
反正就是要 trigger unlink(P)
FD 的 bk 是 ptr, BK 的 fd 是 ptr. 所以它可以 bypass 檢查
所以最後 unlink 的結果 ptr 指向 FD->BK

J 有兩次read的話就可以造成任意寫入

M 最後還有十分鐘，我要用非常快的速度講完 PlaidDB 不過一定講不完 XD

PlaidDB

HP
- PlaidCTF 單題最高分
- Key Value DB
- Red-Black Tree
- Single NULL byte overflow

LAYS *write up by sean: http://winesap.logdown.com/posts/261369-plaid-ctf-2015-plaiddb-writeup*

M 它是個紅黑樹寫的，沒有 source code 你要看組語。
HP key value 的 buffer 都在heap 上，這代表 heap 裡同時有 data 跟 structure
M 我們後來是 fuzz 找到這個東西。直接把 payload 貼上去就會壞掉了。
J single NULLL byte overflow
M 它唯一會壞掉的是 overflow 一個 byte, 而且一定是 0

鄭達達 *這種 overflow 叫做 off-by-one*

M 它的問題在字串補 0 就超出去惹.
HP 當字串跟 buffer 大小剛好相同時，在字串捕 0 時就會造成 overflow
M 我現在也看不懂了 XD 可能要重新用 gdb 追一次才看得懂。
single null byte 的會把下一個 chunk 的第一個 byte 改成 0 (pre size)
0x121 --> 蓋掉 size --> 變成 0x100

原本 put 完長第一行那個樣子。
del 右邊藍色(?)那塊，會蓋掉粉紅色的第一個 byte
它覺得下一個 chunk 在最右邊白色那一塊，就是我的 data 就可以控制了
所以白色那邊我可以設一些假的 header 放在那邊
　兩個 chunk 放在一起就會有不好的事情發生
J 我要free掉那個小的粉紅色的chunk，就可以拿到一個超級大的chunk
也會有檢查double linking list的問題，要overflow去改繞過檢查。
M 所以只要你 malloc 的大小正確，他們就都會被拿出來。

時間已經超過了，快速講一下 XD
de-ASLR 只要 put 一個超級大的 chunk 它就黏在一個奇怪的地方 (?)

J .tls上面有一些好東西？
BC
- main_arena: 可用來 leak libc's base address
- stack canary: 可用來 bypass stack guard
- stack address

BRUCE C *Sean's plaiddb writeup: http://winesap.logdown.com/posts/261369-plaid-ctf-2015-plaiddb-writeup*

J fake chunk on stack (另外一個trick)
回從stack拿一些東西出來，chunk是在stack上，而不是在heap上。
還是會檢查，要騙他 (?

M #3383 check_remalloc_chunk() 它只會檢查 size 是否一致
你只要填好一格剛好的 size 下次 malloc() 就會拿到 stack 上面的 memory

MIAOSKI *http://wapiflapi.github.io/2015/04/22/single-null-byte-heap-overflow/*
http://angelboy.logdown.com/posts/262325-plaid-ctf-2015-write-up

OT # The development of CTFs - Tyler Nighswander

M 先回顧昨天講過的 Pin, Z3
Pin is doing kind of counting instructions.
Every time a function is run, print out the arguments.
In qira, you can't access stack sometimes, but with Pin you can do it.

ИП **angr (http://angr.io/ & https://github.com/angr/angr)**

HP
- Created by Shellfish

王
- Cutting edge symbolic execution + binary analysis

HP
- Used in CGC + Defcon 2015
- Python interface!

王 Symbolic Execution

HP
- Run target program with "symbolic" input

```
int a = atoi(argv[1]);
if (a != 42) {
    if ((a * a ^ atoi(argv[2])) != 1) {
        if (a != 42) {
            puts("ok!");
        }
        else {
            puts("huh???");
        }
    }
    else {
        puts("woah");
    }
```

```
}
else {
    puts("lame");
}
```

M The first thing you can do is to break the program into a tree.
We make the variable "a" symbolic. On one side we have a == 42 and we end up here. So we know a != 42. We think of this as we were doing Z3 yesterday. We add a constraint here as a != 42. So know we are gonna make another variable, _argv2. We have the predicate (or the constraint) that a != 42, and the instruction we want to be equal. Hence (ction we want to be equal. Hence (a != 42) AND (a^a^atoi(_argv2) == 1) and we end up here. SAT or UNSAT?

If this is a C program, we proved that we can't end up with this state (puts("huh??");) because (a != 42) AND (a == 42).
We check if it's possible to find the inputs to satisfy their constraints.

re_test. Let's find a better way than to reverse engineering that.

X
```
$ ipython
```
M
```
import angr
prog = angr.Project("hashcity")
magic_str = angr.StringSpec(sym_length=16, nonnull=True)
initial_state = prog.factory.entry_state(args=[angr.StringSpec(string="./hashcity"), magi
c_str], add_options={'BYPASS_UNSUPPORTED_SYSCALL'})
pg = prog.factory.path_group(initial_state, immutable=False)
pg.explore(find=0x4004e3).unstash(from_stash='found', to_stash='active')
pg.active[4].state.satisfiable()  # True
```
HP
```
pg.active[4].state.se._solver.result.model
```
A
```
'726576657273696e675f69735f66756e'.decode('hex')
```
X
M
```
$ ./hashcity reversing_is_fun
```
From Inndy: http://pastebin.com/0D3qnEH8

IL hashcity z3 solution by Inndy: http://pastebin.com/nPHG4jVs

BC Symbolic execution is not 100% magic

- 大程式無法work
- require good IR
- never get past hard crypto

M intermediate language = language between the programme and Z3.
- KLEE works only on LLVM
  - It translates x86 into LLVM

IL
- triton
- S2E

HP

afl

American Fuzzy Lop (from http://rabbitbreeders.us)

王
- State of the art fuzzer by lcamtuf at Google
- Instrumentation based on edge coverage

HP
- Clone processes after startup for increase speed

王
- Responsible for dozens of cool bugs
- Designed for source-based instrumentation
- Can also be used for binary-only with help QEMU

HP
- Capable of finding bugs or generating
  - Test cases *might help to find*

王
- Will not
  - Find exploitable bugs in most CTF problems
  - Help with RE
  - Deal well with networked problems
- Will:
  - Find crashes for many CTF problems
  - Potentially turn spare CPU cycles into bugs

A
  - Waste time with unexploitable bugs

王 Write your own!
- You've seen many examples of custom tools

HP
- Many written by CTF players
- Writing tools helps many ways:
  - Become better at doing X even without tools

王
  - Learn the ins and outs if your tools making you fast

HP
  - Focus in parts that you like

王
- How to get started
  - Research what tools exist
  - Read research papers on the subject

HP
  - Search for tools/libaries to handle basic pieces

王
  - Modify open source projects
  - Use several CTF problems as test cases during develop

Conclusion

- Tools are essential to fast, efficient problem solving
- CTF problems are great for testing new/research tools
- If you want to focus on cool problems you need tools

HP
CTF Meta-Game

- Meta Game?
  - CTFs are largely technical competitions
王
  - But.. CTFs are not perfect mirrors of technical skills
HP
  - What "non-technical" things can be exploited?

王 Organization

- As a team, organization is key
  - Allows teammates to pool together
HP
  - Reduce duplicated work
  - Helps new people learn quickly
- Meeting in person is best!
王
  - "Pair ~~programming~~ hacking" works well
HP
  - People working around you helps motivation
  - Very easy to tell what people are working on
王
  - Very easy to ask question/give updates
HP
  - ...not always practical
A
- Meeting in person doesn't solve file transfer
HP
  - (private) pastbin
  - netcat!
王
  - netcat!
HP
- IRC
  - "traditional" way
王
  - Easy to read and join
  - Channels for competitions/problems
HP
  - ...easy to stop reading/get distracted
A
  - ... hard to tell at people are hate working on
王
  - .. not everyone can stay 24/7
- Slack [not 100% free]
HP
  - (my current favorite for remote communication)
王
  - Web/mobile/native clients with notifications!
HP
  - All the benefits of IRC
王
  - Drag and drop files/code
  - Easy searchable*, keeps history automatically
AL
  - Easy to extend with bots/integrate tools

M Slack is stupid expensive XD

王 Staying Informed

- in general:follow security news/trends
- Before competition: who is running it? when?

HP • During competition: be on IRC/whatever at all times!

王 • After competition read wirte-up!

HP Hints

- Make sure you know where to find them! Read them!

A • Asking for hints:

HP ◦ Asking competition admins for hints is sometimes OK.

◦ If:

王 ▪ Very few people have solved the challenge

HP ▪ You understand what is going on, but hit roadblock

王 ▪ You suspect something is broken

▪ You don't ask all the time

HP • How to ask for hints:

◦ Only ask competition organizers

王 ◦ "i figured all his stuff out but..."

HP ◦ "Am I on the right track?"

王 ◦ Explain in detail what you know, and what you tried

◦ (all this is good general advice outside of CTFs!)

HP Sleep

- Sleep is very important in competition!
  - <36 hours: might be able to avoid sleep (but sleep before!)

陳 ◦ >36 hours: sleep is essential, plan it well!

HP ▪ no breakthrough? sleep when few problems open

王 ▪ don't sleep through end, new info may be given

A ▪ feeling tired ? sleep might be good idea!!!

HP ▪ be careful of caffeine

Problem Solving

- In CTFs, every problem has bugs (hopefully)

王 ◦ (this is mostly true in the real world as well...)

HP ◦ If you wrote the problem, where would you put a bug

- How is problem set up? Does this leak information?

王 • Problem name, description, point value might leak info

- Solve times from teams can leak info
- Keep track of what you've solved/wirteups you've read

    - Same challenge(or same ideas) often repeat
- When possible make mental checklists of paths to take
HP - Web: cookies, SQLi, race conditions, PHP nonsense, XSS, deserialization, command injection, etc
- Keep track of incorrect guesses
    - They might be right in future problems
    - They are great for creating problems :)

The Future of CTFs

王 Disclaimer

- I'm biased and opinionated on CTF development

CTFs are Changing

HP - Saw on day 1, CTFs have been getting harder
王 - Around 2012 CTFs got a whole lot more "serious"
- kyprizel (MSLC) created http://ctftime.org
- Prior to CTF Time teams werent't a big deal
    - Players on rival teams would sometimes play together
HP - No global ranking (so no global)
王 - Today , top CTF teams play in several CTFs per month
HP - Smaller teams combine to larger teams
王 - LC:BC Leet More, Somked Chicken, Blalaika Crew
M - DEFKOR: Cykor, GoN, others
HP - Samurai: (a lot of teams)

王 CTFs Over Time

HP - Lots of CTFs, but how many are "good"?
- Okay, but that always been the case?

王 What gives?

- Why do CTFs seem to be getting worse in quality?
- Running CTFs isn't easy
HP - Running CTFs without playing in them is terrible
王 - Lots of people want to run

CTFs as a Service

- CTFs are great for training, lots of people want training
- CTFs player normally aren't involved... venture
HS -

王 Prediction

HP
- Good CTF org will start getting money...

王 CTF as E-Sport

HP
- "Live CTF" geohot (and myself) in 2014
  - Thousands of viewers
王
  - Quickly lost momentum
  - To be picked up again by Psifertex
  - Very similar to live streaming games
- Fundamental problem: can't re-use challenges

M
- Fundamental problem: CTFs are highly technical

HP What Else is in Store?
- Automation!
  - S well ...

HP DARPA CGC

王
- Someting like DEFCON CTF, but without humans!

HP
- Give out compiled x86 programs + network traffic
  - 王 Automatically find crashing inputs + patch binaries
  - First event already passed

HP
- $2,000,000 prize!
- Systems had 24 hours to find bugs+patch 131 programs
- Bugs vary from stack overflow to subtle heap corruption

王
- Performance and functionality degradation hurt score

HP
- Top team (my team!) result:
  - 王 Found bugs in 77 programs
  - Successfully patched 102 programs
- Why should you care?

HP
- Finals:
  - Each team uploads exploits, patches, and IDS rules
  - 王 Competitors see patches, IDS rules, network traffic
  - Still graded on offenes + defense + performance
  - HP The winning system is going to play against humans

王
- Current system: 77 binaries in 24 hours
  - HP More than 1 every 20 minutes
  - 王 (In fact, 1 per minute for the first ~30 minutes)

HP
- This is a lot faster than even a small team of humans

王
- Competitions may try to make automation fail
- ..or humans will need to get used to AI teammates

S Prediction

HP
- Future contests will go to great lengths to prevent AI

王
  - Crypto + wired architectures + obfuscation...
- Teams will spend more time on "triage" form fuzzers

Lessons
- As discussed, tools are the future! Get used to it!

HP
- For problem authors:

M
  - learn about tools, make challenges that stress them
  - Tons of things tool can't handle for near future

王 CTFs are Changing
- Many teams merging
- CTFs getting more competitive/valuable

HP
- Predictions:
  - Many large "super teams"
  - Several small, elite teams...

王
- Once CTFs ~$100,000/year in prizes, lots of changes

HP
  - This is enough money for 1 person's salary

王
  - People will spend more time practicing CTFs

HP
- At~ $1,000,000/prizes, a lot more changes

王
  - Starting to support players as full time jobs

M
- Sponsors, audiences, etc. will get larger

王
- Will CTFs actually get to the level?
- What other changes will occur?
- How to get on the right side

王茂林 *QQQQQQQQQ*

HSIEH P *太快了啦 QQQQ*

王茂林 *難過*

Slide 越跳越快 Orzzzzzz

HSIEH P *OTZ*

S {Questions?}

M Q: Can you beat your own programs?

Q: What kind of bugs does your program find?

AL symbolic execution : explore every possible path(place?)

M YELLING IS GOOD~

A I'm not sure what to do now QQ

AWEIMEOW	*在旁邊幫忙加油惹*
MIAOSKI	*多人一起合作打投影片比較快啊 XD 不過旁聽教室都會 lag 大大們打好我才看到螢幕*
王茂林	*一直手殘打錯 QAQ*
AWEIMEOW	*我們會幫忙修正的*
SYA	*修正 + 1*
AWEIMEOW	*打架了 QQ*
MIAOSKI	*對不起我只會打垃圾話逐字稿*
AWEIMEOW	*你在嗆 Sean 嗎*
MIAOSKI	*小的不敢 m(-_-)m*
王茂林	*XDDDDDDDD*
AWEIMEOW	*看到 Sleep 一堆人都搶著打字 XD*
王茂林	*可是前兩個都是Sleep呀QQQQQQQQQ*
INNDY L	*什麼都不會，只好睡了*
MIAOSKI	*英文的逐字稿實在沒辦法... 打字不夠快 QQ.*
	ScoreBoard 第一名的可以拿到一盒巧克力！ 賀! bruce30262
王茂林	*早上的逐字稿太猛了><*

OT

Reverse Engineering and Malware Analysis - Erye Hernandez

王

Tools

- hexrays
- Hiding in Plain Sight
- Packers and Unpacker

PS
- Persistence mechanism

AL
- ollydbg

王 Hiding in plain Sight

AL
- Using launchers

王
- Process injection
  - DLL injection
  - Direct injection

AL
- Process replacement
- Search order hijacking

王 Launchers

- Contains embedded EXE or DLL in resource section
- Most of the time embedded EXE or DLL is compressed, encrypted or encoded

AL
- Has privilege escalation code

王 Process injection

- Most popular
- Injects code into a running process
- Comm Windows APU fuctions used:
  - VirtualAllocEx
  - WriteProcessMemory
- Inherits same rights as therunning process

Dynamic Link Library (DLL)

- Similar to shared objects in Linux
- Executable file
  - Uses the PE file format
  - Use DllMain instead of main (.exe file)

劉
- More function exports than imports

HP
DLL Injection

ALEX L *OpenProcess() -> VirtualAllocEx() -> CreateRemoteThread()*

HP Launcher -> CreateRemoteThread() -> Internet Explorer (malicious DLL)

```
LPVOID WINAPI VirtualAllocEx(
  _In_     HANDLE hProcess,
  _In_opt_ LPVOID lpAddress,
  _In_     SIZE_T dwSize,
  _In_     DWORD  flAllocationType,
  _In_     DWORD  flProtect
);
```

王
```
HANDLE WINAPI CreateRemoteThread(
```
HP
```
  _In_  HANDLE                 hProcess,
  _In_  LPSECURITY_ATTRIBUTES  lpThreadAttributes,
  _In_  SIZE_T                 dwStackSize,
  _In_  LPTHREAD_START_ROUTINE lpStartAddress,
  _In_  LPVOID                 lpParameter,
  _In_  DWORD                  dwCreationFlags,
```

```
    _Out_ LPDWORD           lpThreadId
);
```

王 Process Replacement

- Overwrite a legitimate process in memory with malicious program
- Same permissions as a the replaced process

YJ • Need to know victim process' environment

王 • CreateProcess() with CREATE_SUSPEND
- ZwUnmapViewOfSection()
- VirtualAllocEx()
- WriteProcessMemory
- SetThreadContext()
- ResumeThread()

Launcher -> -> svchost.exe

Search Order Hijacking

- Does not require a launcher
- Use legitimate program to launch malicious DLL

Y • Also a persistence mechanism

王 • Default search order(SafeDLLSearchMode disabled):
  - application directory
  - current directory
  - system directory
  - 16-bit system directory

AL - The Windows directory
  - The directories that are listed in the PATH environment variable

ALEX L *see also https://msdn.microsoft.com/en-us/library/windows/desktop/ms682586(v=vs.85).aspx*

王 Packed Binaries

Y OEP-> .text section ------> OEP-> unpacking stub

蔡 .rsrc section -----> packed code

王 Unpacking

- Automated unpackers(i.e. PE Explorer)
- Manual unpacking
  - Lean the packing algorithm and write a script to reverse

楊 - run the unpacking stub then dump the process and fix the PE header

王 Unpacking Binaries
蔡 OEP ---> unpacking stub OEP ----> .text section

Y Common Packers
IL • UPX（支援多種平台 / 多種ISA）
• ASPack（跟UPX一樣輕量好脫殼）
蔡 • AsProtect
Y • Zprotect v1.3
楊 • PECompact
• Themida (TMD!很難脫殼，超級難, 真)
LAYS *vm 處理起來有點麻煩而已，不過舊版本用 od plugin (Oreans UnVirtualizer) 效果還可以*
INNDY L *l4ys <(_ _)>*
MIAOSKI *l4yx m(-_-)m*
PF *l4yx <(_ _)rz*

IL ◦ WinLicense，Themida變種，focus在license管理
楊 • WinUpack
IL • VMProtect
◦ 超強VM

Y **Demo**

TOOL->PE Expjlorer
IL （講師在demo的時候不小心讓malware.exe跑起來了 .. GG）
ИП (好在這個malware沒有wipe功能或cryptolock功能, 或重開機功能)
S (講師口頭禪 Oh man! XD , Cool. All right!! Awesome!!

IL UPX特徵：
• section 數量變少
• UPX0, UPX1

HP Cool!~~哭喔~~
C 範例：AIS3_packed.exe // UPX脫殼
HP 設置硬體訪問斷點當成是再次存取到記憶體時就會被中斷
很好，剛好停在 JMP 前面，我們跟進去看看
之後把它 dump 出來，這會花一段間
Cool!
之後我們把他丟進 IDA Pro 裡面

你看看有多出好多個 function 看起來就像正確的 binary 了
Cool!

Y Persistence Mechanisms
  Windows Registry
  Trojanized System Binaries
楊 Search Order Hijacking

王 Windows Registry
- Run key
  - HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
- Applinit
  - Y HKLM\...\CurrentVersion\Windows
- 王 SvcHost

王 Trojanized System Binaries
- Modify system DLLs by patching the entry function
- Jumps to malicious code in an empty space of the DLL
- 楊 Jumps back to the original DLL code

Y Debuggers
- Ollydbg v1.10
- 王 Immunity Debugger
- WinDbg
  - HP by Microsoft
  - support x64
  - Y can be used to debug kernel modules

СЛЕДО… *(註: IDA Pro也有debugger喔 https://www.hex-rays.com/products/ida/debugger/)*

YURILOVE… *ce也很好用阿...*

СЛЕДО… *課程結束了!????[y/n]*
今日課程好玩嗎[y/n]
請下斷點以結束本日課程：^C (int3) (CC)

S 本日密碼：infected (如果解不開請自行爆破)

СЛЕДО… *(不是 GGININDER !?)*

HP DWORD WINAPI GetModuleFileName(
_In_opt_ HMODULE hModule,
Out LPTSTR lpFilename,
_In_ DWORD nSize
);

when hModule = 0 returns itself

ИССЛЕДО… *廣告一下...有一個CTF : https://ctf.ekoparty.org/*

CEBRUSFS *這個是?? 哪邊找到的啊XD*

ИССЛЕДО… *就有人寄給我問我要不要玩...*

CEBRUSFS *看起來頗新的，還沒在ctftime上看到XD*

ИССЛЕДО… *前五名可以去阿根廷玩喔*

CEBRUSFS *看起來機票還是要自己買啊? 而且只有一人份的conf ticket XD*

ИССЛЕДО… *不知道耶, 不過可以找趨勢科技贊助, 我可以幫忙看看*

CEBRUSFS *基本上可以玩玩啦，但是要不要去又是另一回事了(?)*

ИССЛЕДО… *呵呵. 一開始是線上的*

CEBRUSFS *幫忙宣傳的話叫他放到ctftime上啊，人會更多*

ИССЛЕДО… *我猜應該是新手弄的 這個security conference 以前沒聽過*

CEBRUSFS *還是可以放到ctftime啦，最近越來越多這種的XD Tyler就說這種比賽就越來越多啊XD*

ИССЛЕДО… *對呀. 我去叫他放上去*
看來以後每個國家都有很多..光是打比賽賺錢就夠了(!?@@@)

CEBRUSFS *有錢的ctf不太多啊.....而且入帳通常會很久 (?)*
而且一個人打不起來，目前的話應該還是會餓死，而且一整個team分完獎金其實就沒剩多少了。

ИССЛЕДО… *(XD)*
ㄟ...https://ctftime.org/team/321/ 這個team辦的研討會

SYA *語助詞好多 XD 講師*

ИССЛЕДО…

葉泳志 *這是我做的共筆要是不嫌棄可以把統整後的資料放在這裡。->AIS3*