Basic Reverse

ss8651twtw

Outline

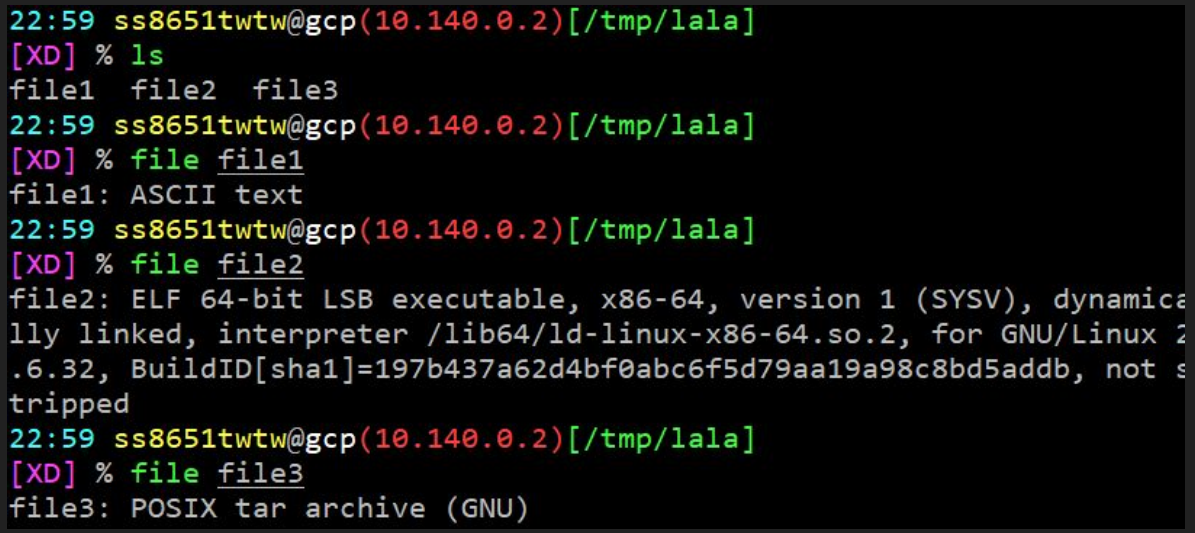
- Reverse 101
- x86 組合語言
- C / C++ Reverse
- IDA 使用
- 各種保護
- 總結

Reverse 101

檔案類型

$ file <something>

查看檔案類型



包含字串

$ strings <something>

印出檔案中的可視字串

$ strings -n <min-len> <something>

印出長度最短為 min-len 的可視字串

```
22:59 ss8651twtw@gcp(10.140.0.2)
[XD] % strings file2
/lib64/ld-linux-x86-64.so.2
{Czb
libc.so.6
fflush
exit
puts
__stack_chk_fail
putchar
printf
read
stdout
sleep
__libc_start_main
```

包含字串

$ strings <something> | grep "read"

在 strings 的結果中有包含 "read" 字串的結果

```
23:08 ss8651twtw@gcp(10.140.0.2)[/tmp/lala]
[XD] % strings file2 | grep "read"
read
read@@GLIBC_2.2.5
```

objdump

$ objdump -M intel -d <binary>

以 intel 格式顯示 binary 反組譯的結果 (組合語言)

objdump

```
23:37 ss8651twtw@gcp(10.140.0.2)[/tmp/lala]
[XD] % objdump -M intel -d ./file2

./file2:     file format elf64-x86-64


Disassembly of section .init:

00000000004005b8 <_init>:
  4005b8:       48 83 ec 08             sub    rsp,0x8
  4005bc:       48 8b 05 35 1a 20 00    mov    rax,QWORD PTR [rip+0x
201a35]        # 601ff8 <__gmon_start__>
  4005c3:       48 85 c0                test   rax,rax
  4005c6:       74 05                   je     4005cd <_init+0x15>
  4005c8:       e8 b3 00 00 00          call   400680 <__gmon_start_
_@plt>
  4005cd:       48 83 c4 08             add    rsp,0x8
  4005d1:       c3                      ret

Disassembly of section .plt:
```

Lab 2-1

- add

Lab 2-1

- add
- 起手式
  - **$ file** 查看題目檔案類型
  - **$ strings** 找出一些可視字串
  - 如果是執行檔就執行看看

Lab 2-1

- 分析執行檔
  - objdump 反組譯看組合語言
  - IDA pro 分析

strace / ltrace

$ strace <binary>

查看 binary 執行時的 system call 和 signal

$ ltrace <binary>

查看 binary 執行時的 library call

strace / ltrace

```
23:34 ss8651twtw@gcp(10.140.0.2)[/tmp/lala]
[XD] % strace ./file2
execve("./file2", ["./file2"], [/* 21 vars */]) = 0
brk(NULL)                                       = 0x1d99000
access("/etc/ld.so.nohwcap", F_OK)              = -1 ENOENT (No such file or
 directory)
access("/etc/ld.so.preload", R_OK)              = -1 ENOENT (No such file or
 directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=185501, ...}) = 0
mmap(NULL, 185501, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fbfa7185000
close(3)                                        = 0
access("/etc/ld.so.nohwcap", F_OK)              = -1 ENOENT (No such file or
 directory)
```

strace / ltrace

```
23:35 ss8651twtw@gcp(10.140.0.2)[/tmp/lala]
[XD] % ltrace ./file2
__libc_start_main(0x400f72, 1, 0x7ffc831c64a8, 0x400fd0 <unfinished
...>
puts("ANGRMAN X"ANGRMAN X
)                          = 10
puts("1 GAME START"1 GAME START
)                         = 13
puts("2 PASS WORD"2 PASS WORD
)                         = 12
puts("3 EXIT GAME"3 EXIT GAME
)                         = 12
read(0a
, "a\n", 2)                              = 2
+++ exited (status 0) +++
```

gdb

- 執行 binary 並且使用 gdb 來 debug
  - **$ gdb <binary>**
- 先執行 gdb 之後再 attach 上要 debug 的 process
  - **$ gdb**
  - **attach <pid>**

patch

- 使用 nop 將檢查判斷蓋掉
- 改掉要執行的 function
- 插 code 紀錄資訊

Lab 2-2

- guess

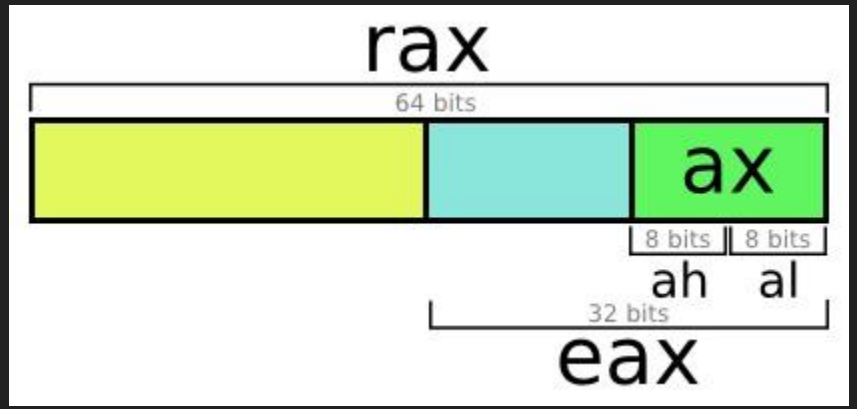
x86 組合語言

x86 組合語言

- **與 x64 的差別**
  - 暫存器大小
  - function call 傳參方式
  - system call 方法

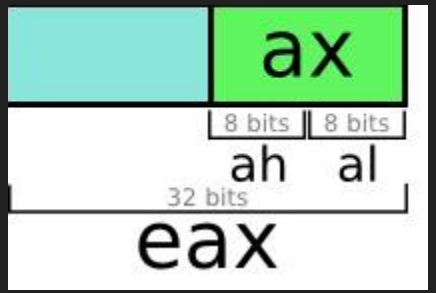
x64 暫存器

- r[a-d]x register layout



x86 暫存器

- e[a-d]x register layout



x64 System call

- **syscall**
- syntax
  - syscall
- example

%rax	System call	%rdi	%rsi	%rdx
0	sys_read	unsigned int fd	char *buf	size_t count
1	sys_write	unsigned int fd	const char *buf	size_t count
2	sys_open	const char *filename	int flags	int mode

http://blog.rchapman.org/posts/Linux_System_Call_Table_for_x86_64/

x86 System call

- **int 0x80**
- syntax
  - int 0x80
- example

#	Name	Registers			
		eax	ebx	ecx	edx
0	sys_restart_syscall	0x00	-	-	-
1	sys_exit	0x01	int error_code	-	-
2	sys_fork	0x02	struct pt_regs *	-	-
3	sys_read	0x03	unsigned int fd	char __user *buf	size_t count

https://syscalls.kernelgrok.com/

x64 function call

- **function**
- parameter passing
  - rdi、rsi、rdx、rcx、r8、r9、push stack

```
int foo(int a, int b, int c, int d, int e, int f, int g)
        rdi    rsi    rdx    rcx    r8     r9     stack
```

x86 function call

- **function**
- parameter passing
  - push stack

```
int foo(int a, int b, int c)
        stack  stack  stack
```

```
push c
push b
push a
call foo
```

C / C++ Reverse

C Reverse

- conditional statement

```
if (rax < 5) {
  // do something
}
else if (rax >= 5 && rax < 10) {
  // do something
}
else {
  // do something
}
```

C Reverse

- conditional statement

```
cmp rax, 5
jae Lelseif
; do something
jmp Lend
Lelseif:
cmp rax, 10
jae Lelse
; do something
jmp Lend
Lelse:
; do something
Lend:
```

C Reverse

- loop

```c
for (int i = 0; i < 10; i++) {
  // do something
}
```

C Reverse

- loop

```
mov rcx, 0
Lloop:
cmp rcx, 10
jae Lend
; do something
inc rcx
jmp Lloop
Lend:
```

C Reverse

- **function**
- parameter passing (x64)
  - rdi、rsi、rdx、rcx、r8、r9、push stack

```
int foo(int a, int b, int c, int d, int e, int f, int g)
        rdi    rsi    rdx    rcx    r8     r9     stack
```

C Reverse

- function

```c
int foo(int a, int b) {
  return a + b;
}

int main() {
  int a = 1, b = 2;
  foo(a, b);
}
```

C Reverse

- function

```
foo:
push   rbp
mov    rbp,rsp
mov    DWORD PTR [rbp-0x4],edi
mov    DWORD PTR [rbp-0x8],esi
mov    edx,DWORD PTR [rbp-0x4]
mov    eax,DWORD PTR [rbp-0x8]
add    eax,edx
mov    rsp, rbp
pop    rbp
ret
```

```
main:
push   rbp
mov    rbp,rsp                    prolog
sub    rsp,0x10
mov    DWORD PTR [rbp-0x8],0x1
mov    DWORD PTR [rbp-0x4],0x2    存(init) local var
mov    edx,DWORD PTR [rbp-0x4]
mov    eax,DWORD PTR [rbp-0x8]    抓local var
mov    esi,edx
mov    edi,eax                    放參數
call   660 <foo>
mov    eax,0x0                    return 0
leave
ret
```

C++ Reverse

- **Name Mangling**
  - 修飾名稱並附加資訊
  - 目的
    - compiler 和 linker 可以辨識同名參數不同的 function
    - 在不同的 class、template、namespace 底下可以有同樣名稱的 function

C++ Reverse

```
int foo(int a, int b) {
    return a + b;
}

int foo(int a) {
    return a * 2;
}

int main() {
    cout << foo(5) << endl;
    cout << foo(2, 4) << endl;
    return 0;
}
```

C++ Reverse

```
0000000000001165 <_Z3fooii>:
    1165:       55                      push   rbp
    1166:       48 89 e5                mov    rbp,rsp
    1169:       89 7d fc                mov    DWORD PTR [rbp-0x4],edi
    116c:       89 75 f8                mov    DWORD PTR [rbp-0x8],esi
    116f:       8b 55 fc                mov    edx,DWORD PTR [rbp-0x4]
    1172:       8b 45 f8                mov    eax,DWORD PTR [rbp-0x8]
    1175:       01 d0                   add    eax,edx
    1177:       5d                      pop    rbp
    1178:       c3                      ret

0000000000001179 <_Z3fooi>:
    1179:       55                      push   rbp
    117a:       48 89 e5                mov    rbp,rsp
    117d:       89 7d fc                mov    DWORD PTR [rbp-0x4],edi
    1180:       8b 45 fc                mov    eax,DWORD PTR [rbp-0x4]
    1183:       01 c0                   add    eax,eax
    1185:       5d                      pop    rbp
    1186:       c3                      ret
```

C++ Reverse

- 還原名稱
  - $ c++filt <name>

```
[XD] # c++filt _ZSt4endlIcSt11char_traitsIcEERSt13basic_ostreamIT_T0_ES6_
std::basic_ostream<char, std::char_traits<char> >& std::endl<char, std::char_traits<char> >(std::basic_ostream<char, std::char
_traits<char> >&)
8:55 root@kali(10.0.2.15)[~/course]
[XD] # c++filt _ZStL19piecewise_construct
std::piecewise_construct
```

- gdb 中可以下
  - set print asm-demangle on

IDA 使用

IDA 使用

- 反編譯大法
- 字串表
- 標記
  - 變數名
  - function 參數
  - struct 結構

反編譯大法

把 binary 反編譯回 C code

1. 在 functions window 點選想看的 function
2. 按下 F5
3. 完成!!!

字串表

列出可視字串表

- View => Open subviews => Strings
- shift + F12

標記

標記變數名

1. 先點擊要命名的變數
2. 按下 n
3. 輸入新的變數名

標記

標記 function 參數

1. 先點擊該 function
2. 按下 y
3. 輸入正確的 function 參數

標記

標記 struct 結構

1. 切到 Structures 頁面
2. 看裡面的說明
3. 新增 struct 並標記裡面的內容

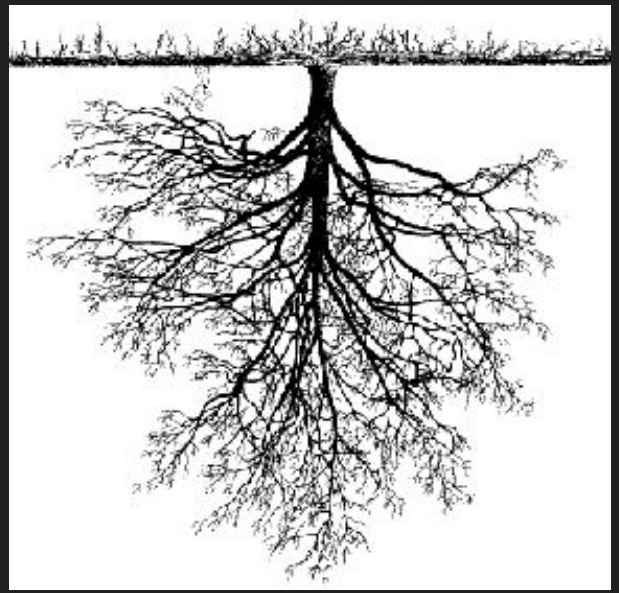
標記

標記 struct 結構

```
; Ins/Del : create/delete structure
; D/A/*   : create structure member (data/ascii/array)
; N       : rename structure or structure member
; U       : delete structure member
```

Lab 2-3

- forest



各種保護

代碼混淆

- 插入垃圾指令
  - 前提是不影響原程式正常執行

```
add rax, 5 =>  xor ecx, ecx
mov rdi, rax   sub rdx, r8
               add rax, 5
               mul rbx, rcx
               mov rdi, rax
```

代碼混淆

- 替換與原指令等價的指令

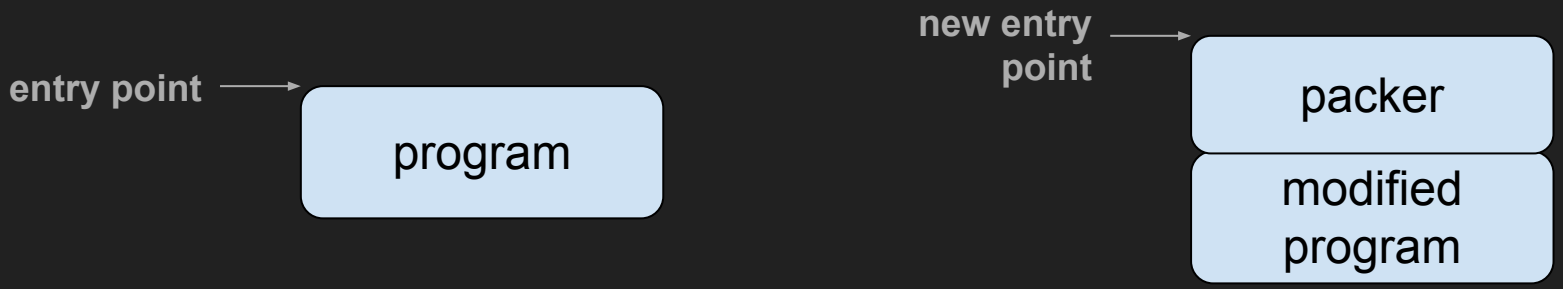
```
add rax, 5 => add rax, 10
              sub rax, 3
              sub rax, 5
              add rax, 3

mov rdi, rax => push rax
                pop rdi

jmp label    => push label
                ret
```

殼

在原本的程式外加一層保護，用來防止修改或反編譯



有加殼的程式在 runtime 才將真正的 program 解回來執行

殼

- 壓縮殼
  - UPX、ASPCAK、TELOCK
- 加密殼
  - ASPROTECT、ARMADILLO
- 自己實作的殼

殼

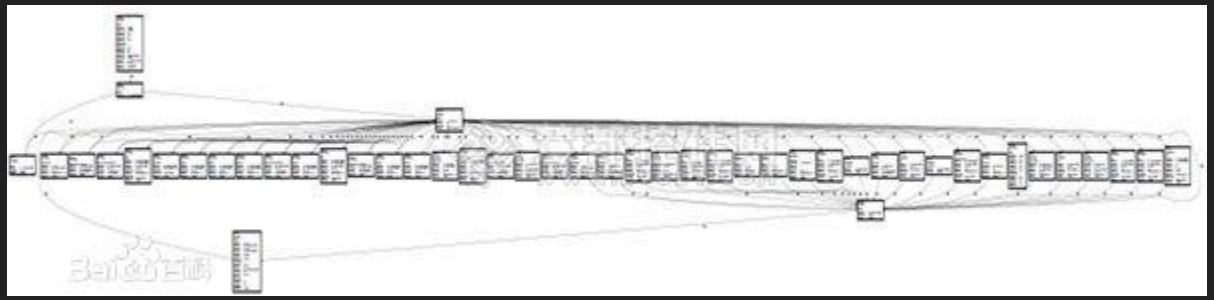
- 查殼
  - PEiD
  - http://www.softpedia.com/get/Programming/Packers-Crypters-Protectors/PEiD-updated.shtml

UPX

- 安裝執行檔
  - https://github.com/upx/upx/releases
- 加殼
  - 原檔案要 40 Kb 以上
- 判定 UPX
  - **$ strings** 尋找 UPX 字串

VM 保護

- 自行實作指令集並交由 VM 執行
- 難以破解 Orz



總結

總結

- 指令集
  - x86、x64、ARM、MIPS…
- OS
  - Linux、Windows、macOS…
- 程式語言
  - C、C++、Go、Rust…
- 各種保護
  - 代碼混淆、殼、VM 保護…

Reference

- http://blog.rchapman.org/posts/Linux_System_Call_Table_for_x86_64/
- https://syscalls.kernelgrok.com/
- https://ithelp.ithome.com.tw/articles/10188209
- https://wizardforcel.gitbooks.io/re-for-beginners/content/Part-III/Chapter-50.html
- https://baike.baidu.com/item/VMProtect

Thanks for listening!