

Reverse 0x01

ss8651twtw

<https://ppt.cc/f3O5Fx>

<https://bamboofox.cs.nctu.edu.tw/>

<https://ppt.cc/fEhfpx>

wget https://ppt.cc/fEhfpx

環境設定

Windows

- 下載安裝 virtualbox / vmware
- 下載 ubuntu / kali iso
- 安裝虛擬機

Linux / Mac

- 不用做事 (´·ω·`)
- 要裝 strace ltrace

懶人包

- 註冊 picoCTF 2018
- 直接用他的 Shell
- <https://2018game.picoctf.com/shell>

Basic

Outline

- 檔案類型
- 包含字串
- strace / ltrace
- objdump

檔案類型

\$ file <something>

查看檔案類型

```
22:59 ss8651tw@tcp(10.140.0.2)[/tmp/lala]
[XD] % ls
file1 file2 file3
22:59 ss8651tw@tcp(10.140.0.2)[/tmp/lala]
[XD] % file file1
file1: ASCII text
22:59 ss8651tw@tcp(10.140.0.2)[/tmp/lala]
[XD] % file file2
file2: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically
linked, interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 2
.6.32, BuildID[sha1]=197b437a62d4bf0abc6f5d79aa19a98c8bd5addb, not s
tripped
22:59 ss8651tw@tcp(10.140.0.2)[/tmp/lala]
[XD] % file file3
file3: POSIX tar archive (GNU)
```

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 ss8651ttw@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 ss8651tw@tcp(10.140.0.2)[/tmp/lala]  
[XD] % strings file2 | grep "read"  
read  
read@@GLIBC_2.2.5
```

小練習

- EasyCTF IV
 - hexedit
- EasyCTF 2017
 - Hexable

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 ss8651tw@tcp(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 ss8651tw@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) +++
```


小練習

- CSIE 2017
 - strace

objdump

```
$ objdump -M intel -d <binary>
```

以 intel 格式顯示 binary 反組譯的結果 (組合語言)

```
$ objdump -M intel -d <binary> | less
```

把輸出結果導向到 less 方便查詢閱讀

objdump

```
23:37 ss8651tw@tcp(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:

小練習

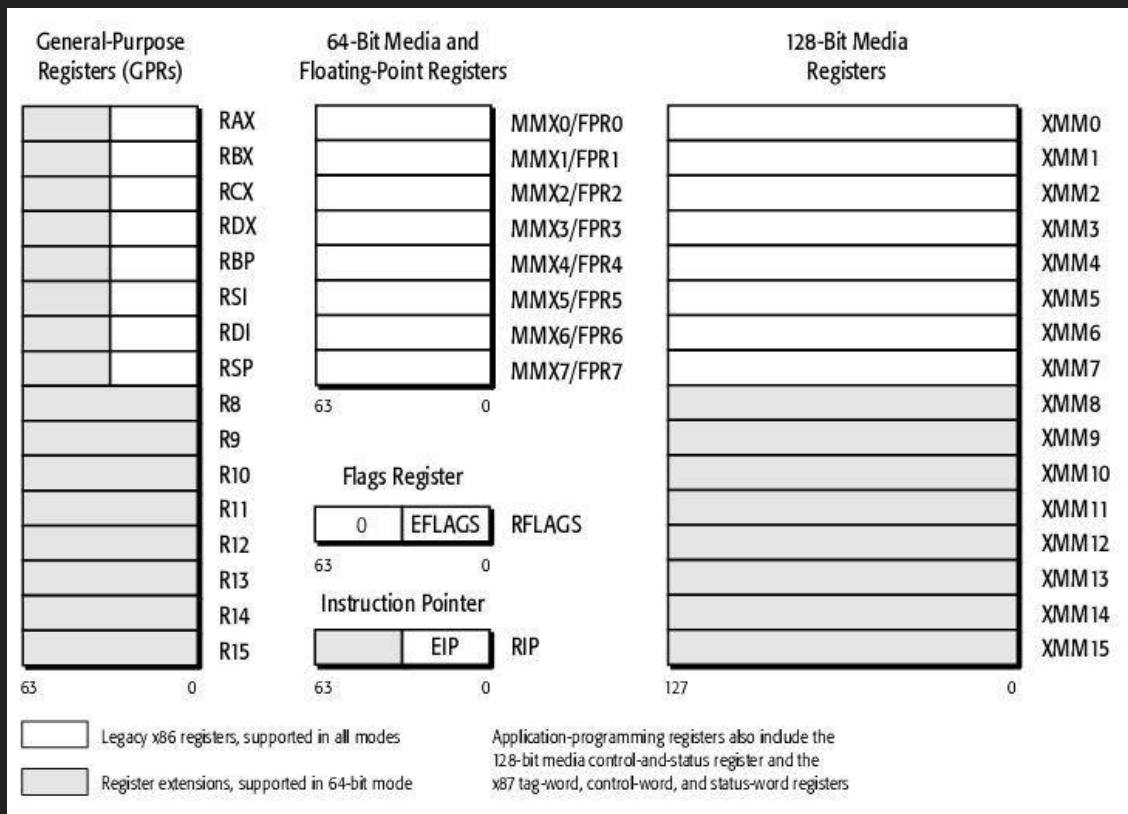
- Reverse CTF
 - find
 - search

組合語言

Outline

- 暫存器
- stack
- 組合語言指令
- 組合語言與 C 的轉換

暫存器

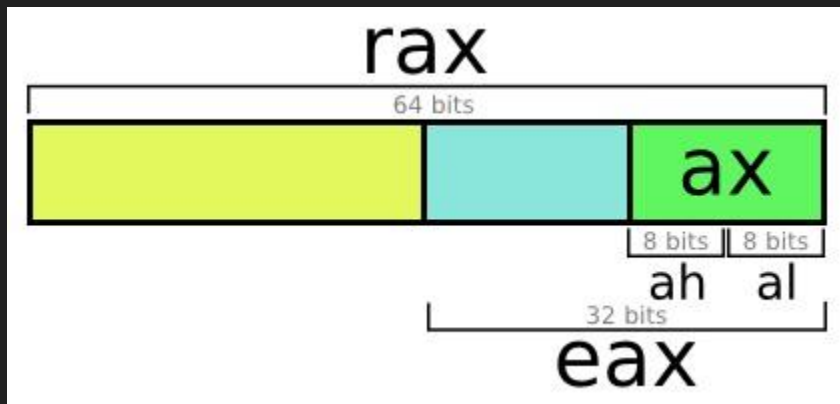


暫存器

- rax - accumulator
- rbx - base
- rcx - count
- rdx - data
- rsi - source index
- rdi - destination index
- rbp - base pointer
- rsp - stack pointer

暫存器

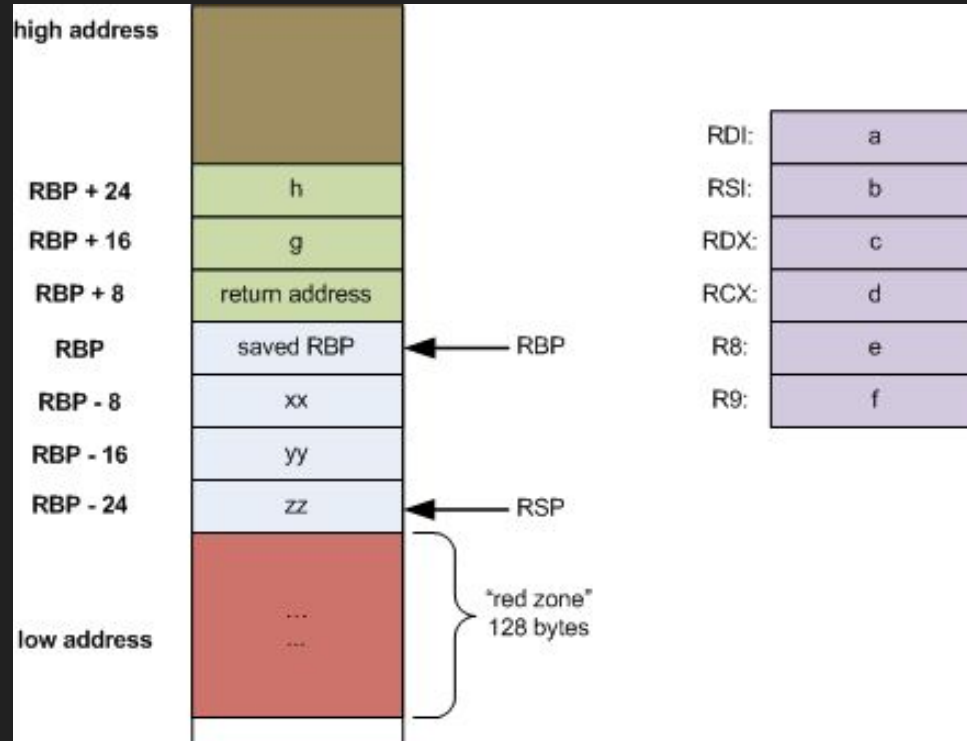
- r[a-d]x register layout



Stack

- rsp、rbp 所指向的記憶體間的空間
- 可以用來記錄
 - return address
 - local variable

Stack



組合語言指令

- **mov**
- syntax
 - **mov dest, source**
- example
 - `mov rax, rbx` `// rax = rbx`
 - `mov rax, [rbp - 4]` `// rax = *(rbp - 4)`
 - `mov [rax], rbx` `// *rax = rbx`

組合語言指令

- **add、sub、imul、idiv、and、or、xor**

- syntax

- add dest, source

- example

- sub rbx, [rbp - 4] // rbx = rbx - *(rbp - 4)
 - mul rcx, 2 // rcx = rcx * 2
 - xor [rsp], rax // *rsp = (*rsp) ^ rax

組合語言指令

- **inc、dec、neg、not**

- syntax

- inc dest

- example

- dec rbx

// rbx = rbx - 1

- neg rcx

// rcx = -rcx

- not byte [rsp]

// convert [rsp] to byte
*rsp = ~(*rsp)

組合語言指令

- if dest is memory access and source is not register, need to specify the dest's type
- example
 - `mov byte [rcx], 0x61` // byte = 8 bits
 - `mul word [rax], 0x87` // word = 2 bytes
 - `inc dword [rbp]` // dword = 2 words
 - `not qword [rsp]` // qword = 2 dwords

組合語言指令

- dest and source must be the **same type**
- rip could NOT be dest

組合語言指令

- **cmp**
- syntax
 - **cmp value1, value2**
- example
 - `cmp rax, 5` `// compare the values and set the flag`
 - `cmp rbx, rcx`
 - `cmp word [rsp], 0x1234`

組合語言指令

- **jmp**
- syntax
 - **jmp label**
- example
 - loop: // set a label
 - ; do something
 - jmp loop // jump to loop label

組合語言指令

- **ja、jb、jna、jbe、je、jne、jz**
- syntax
 - ja label
- example
 - `cmp rax, 10` // compare the values and set flag
 - `je quit` // check flag if equal jump to quit

<http://www.felixcloutier.com/x86/Jcc.html>

組合語言指令

- **nop**
- syntax
 - **nop**
- example
 - `nop`

`// do nothing`

`// 在 patch 的時後很好用`

小練習

- picoCTF 2017
 - Programmers Assemble
 - 題目為 AT&T 格式
 - 可參考下一頁 Intel 格式的題目

小練習

.global main

main:

mov eax, XXXXXXXX

mov ebx, 0

mov ecx, 0x5

loop:

test eax, eax

jz fin

add ebx, ecx

dec eax

jmp loop

fin:

cmp ebx, 0x7ee0

je good

mov eax, 0

jmp end

good:

mov eax, 1

end:

ret

小練習

```
eax = xxxxxxxx
```

```
ebx = 0
```

```
ecx = 5
```

```
while (eax != 0) {
```

```
    ebx += ecx
```

```
    eax--
```

```
}
```

```
if (ebx == 0x7ee0) {
```

```
    // good
```

```
    eax = 1
```

```
    return
```

```
}
```

```
else {
```

```
    eax = 0
```

```
    return
```

組合語言指令

- **push、pop**
- syntax
 - push source
 - pop dest
- example
 - push rax
 - push 0
 - pop rcx
 - pop word [rbx]

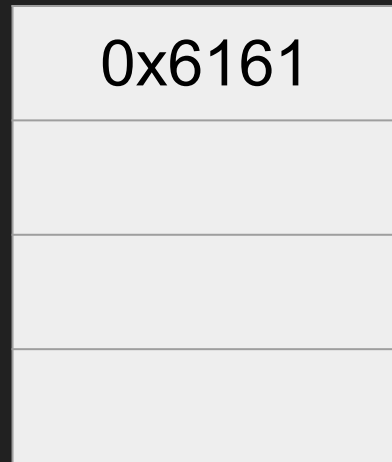
組合語言指令

- **push rax**
- push 0
- pop rcx
- pop word [rbx]

rax = 0x6161

rbx = 0x601000

rcx = 0x1234



組合語言指令

- `push rax`
- **`push 0`**
- `pop rcx`
- `pop word [rbx]`

`rax = 0x6161`

`rbx = 0x601000`

`rcx = 0x1234`

0
0x6161

組合語言指令

- push rax
- push 0
- **pop rcx**
- pop word [rbx]

rax = 0x6161

rbx = 0x601000

rcx = 0

0x6161

組合語言指令

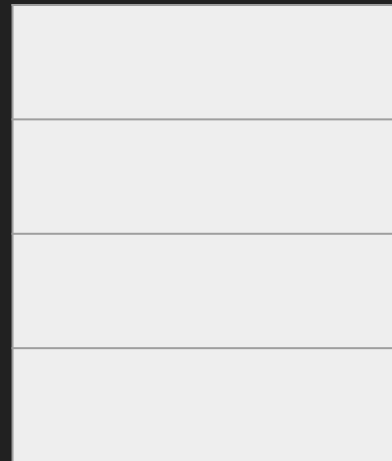
- push rax
- push 0
- pop rcx
- **pop word [rbx]**

rax = 0x6161

rbx = 0x601000

rcx = 0

***(0x601000) = 0x6161**



小練習

- picoCTF 2017
 - A Thing Called the Stack
 - 題目為 AT&T 格式
 - 可參考下一頁 Intel 格式的題目

小練習

foo:

push ebp

mov ebp, esp

push edi

push esi

push ebx

sub esp, 0xf4

mov dword [esp], 0x1

mov dword [esp + 0x4], 0x2

mov dword [esp + 0x8], 0x3

mov dword [esp + 0xc], 0x4

組合語言指令

- **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/

組合語言與 C 的轉換

- conditional statement

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


組合語言與 C 的轉換

- 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:
```

組合語言與 C 的轉換

- loop

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

組合語言與 C 的轉換

- loop

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

組合語言與 C 的轉換

- **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
```

組合語言與 C 的轉換

- function

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

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

組合語言與 C 的轉換

- 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
sub     rsp, 0x10
mov     DWORD PTR [rbp-0x8], 0x1
mov     DWORD PTR [rbp-0x4], 0x2
mov     edx, DWORD PTR [rbp-0x4]
mov     eax, DWORD PTR [rbp-0x8]
mov     esi, edx
mov     edi, eax
call    660 <foo>
mov     eax, 0x0
leave
ret
```

Compile

- **nasm -f elf64 <asm source code>**
 - -f elf64 // output elf 64 format
- **ld -m elf_x86_64 -o <output file name> <object file>**
 - -m elf_x86_64 // elf x86-64 format
 - -o // output file name

Disassemble

- **objdump -M intel -d <binary file>**
 - -M intel // intel syntax
 - -d // disassemble
- using IDA pro will be happy

Practice

- <https://github.com/ss8651twtw/asm-practice.git>

git clone https://github.com/ss8651twtw/asm-practice.git

Practice

- **Hello World!**
 - using **write** syscall to show “Hello World!”

Practice

- **string split**
 - input string s ($1 \leq \text{len}(s) \leq 9$)
 - output a character every line in input order

Practice

- **calculate N!**
 - input N ($1 \leq N \leq 9$)
 - using recursive function to calculate N!
 - output store in rax

Reference

- <https://en.wikipedia.org/wiki/X86-64>
- <https://software.intel.com/en-us/articles/introduction-to-x64-assembly>
- http://blog.rchapman.org/posts/Linux_System_Call_Table_for_x86_64/
- <http://www.felixcloutier.com/x86/>
- <https://eli.thegreenplace.net/2011/09/06/stack-frame-layout-on-x86-64>

Thanks for listening!