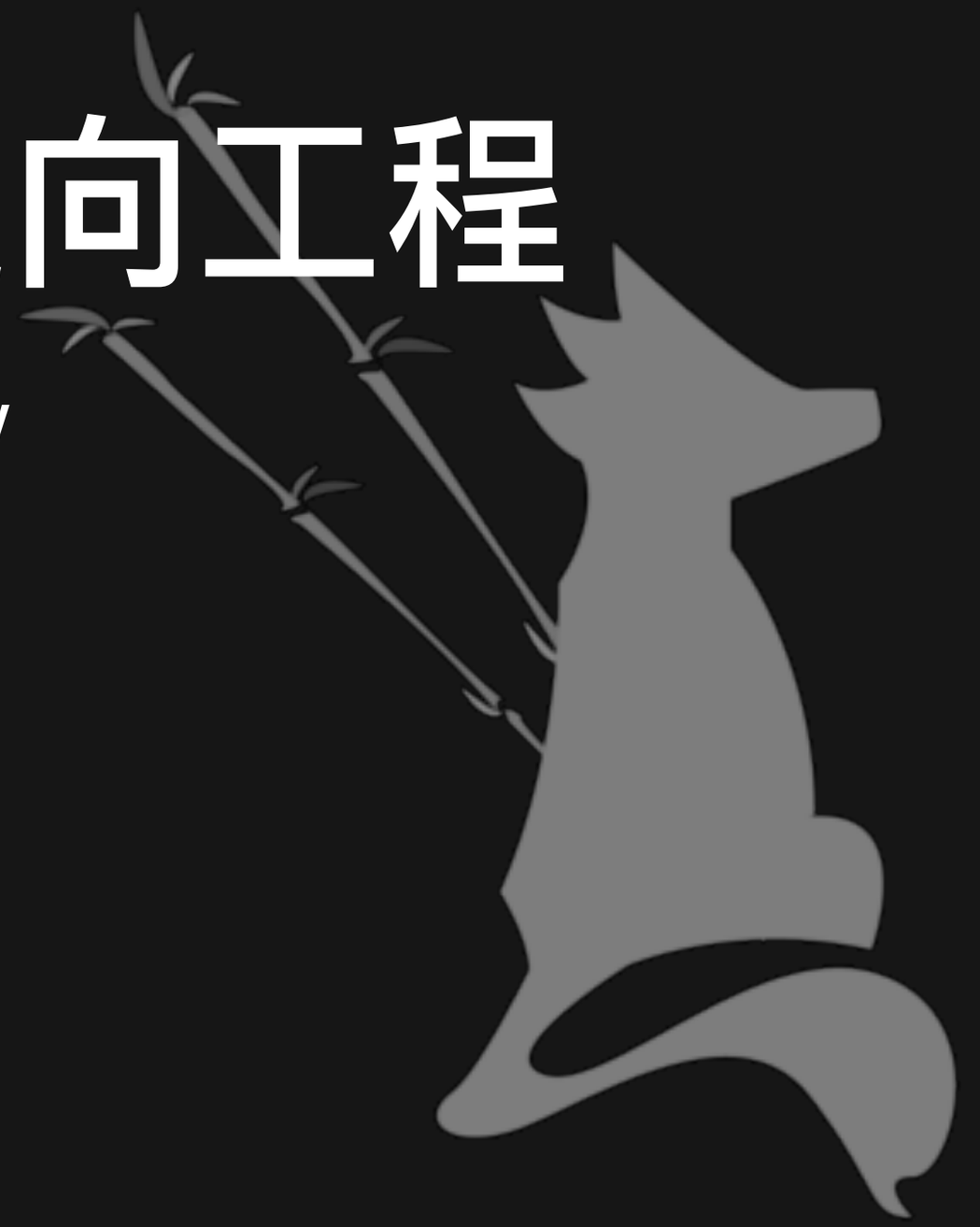
初探 CTF 逆向工程

ss8651twtw



Day1

- 基本工具介紹
- x64 組合語言

Day1

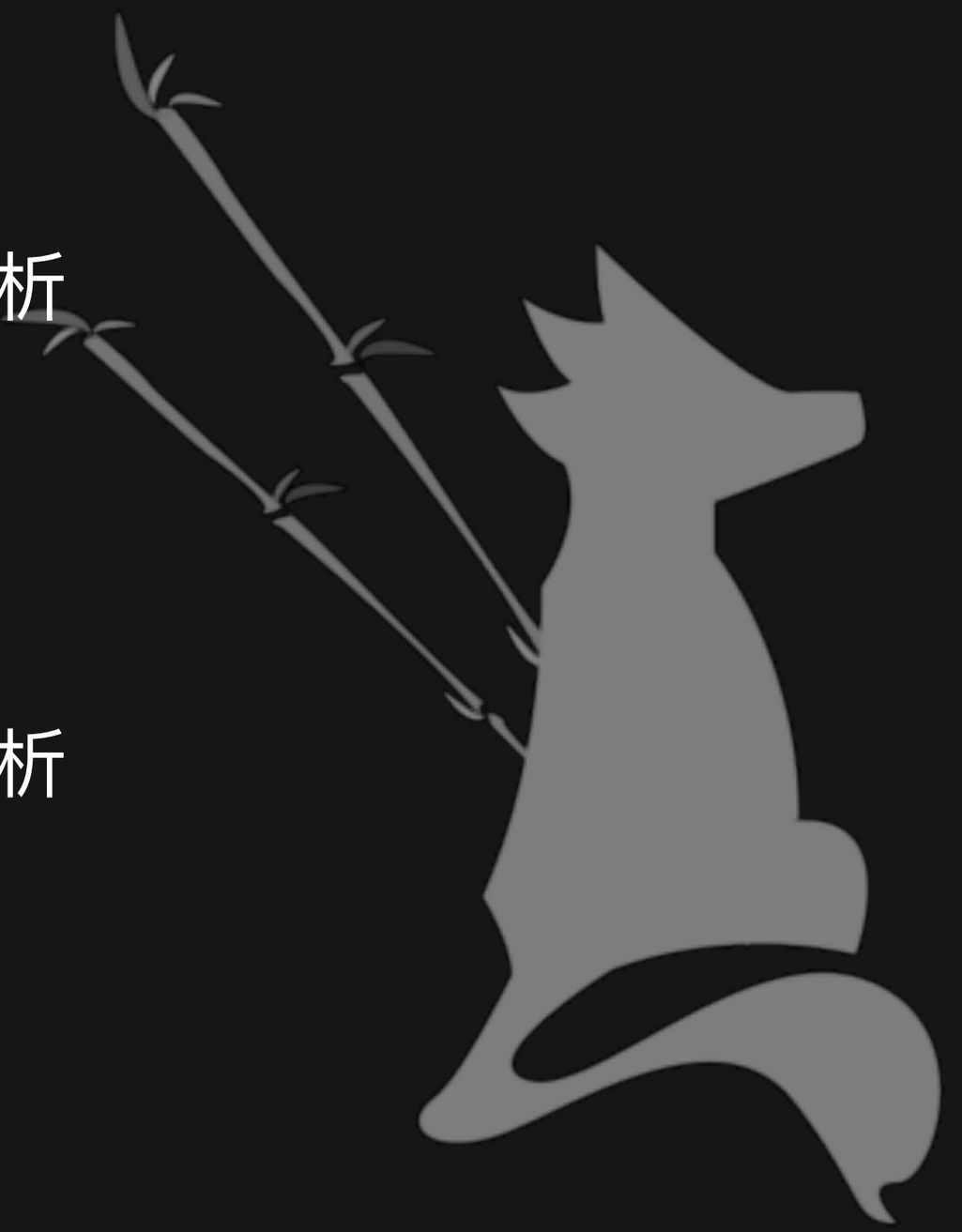
- 基本工具介紹
- x64 組合語言

基本工具介紹

- 起手式
- objdump
- strace / ltrace
- gdb

→ 靜態分析

→ 動態分析



基本工具介紹

❖ 起手式

- 查看檔案類型
  - $ file <something>

基本工具介紹

```
0:14:25 ss8651twtw @ ub18 in ~/csc/lab
→ ls
find  hexable  hexedit  search  strace

0:14:26 ss8651twtw @ ub18 in ~/csc/lab
→ file find
find: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV),
dynamically linked, interpreter /lib64/l, for GNU/Linux 2.6.32
, BuildID[sha1]=62f04aeff7d39314bca80039fd545b2f817e12c8, not
stripped
```

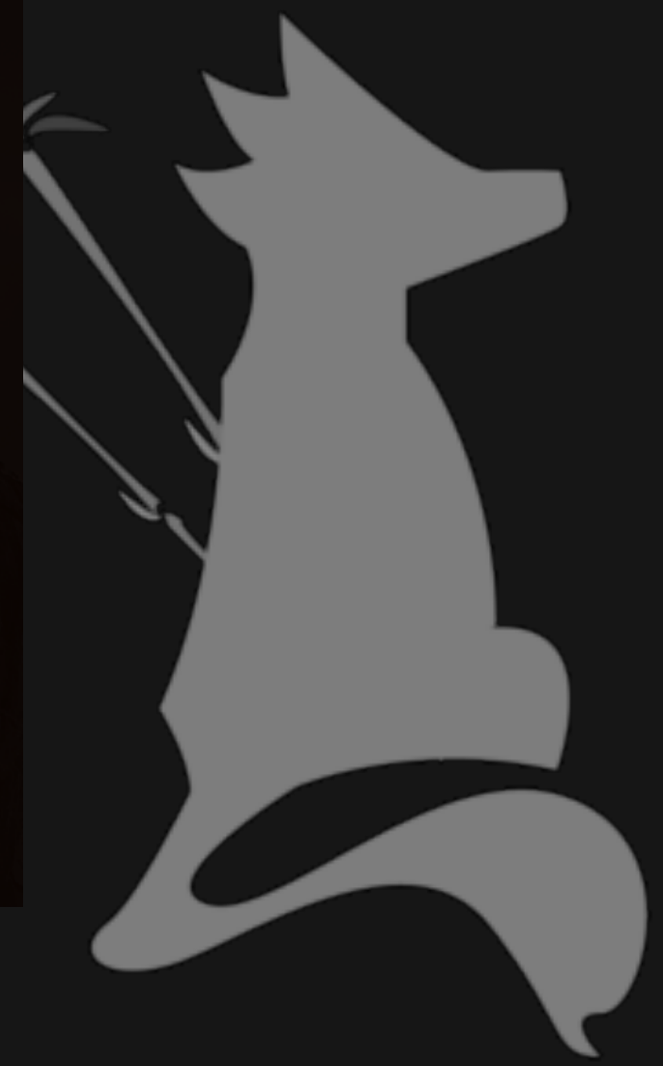
基本工具介紹

❖ 起手式

- 印出可視字串
  - $ strings <something>
- 印出最短長度為 min-len 的可視字串
  - $ strings -n <min-len> <something>

基本工具介紹

```
0:18:23 ss8651twtw @ ub18 in ~/csc/lab
→ strings ./find
/lib64/ld-linux-x86-64.so.2
libc.so.6
puts
__cxa_finalize
__libc_start_main
__gmon_start__
_Jv_RegisterClasses
_ITM_deregisterTMCloneTable
_ITM_registerTMCloneTable
GLIBC_2.2.5
AWAVA
AUATL
[]A\A]A^A_
nothing here!
;*3$"
```



基本工具介紹

❖ 起手式

- 在可視字串中尋找特定字串
  - $ strings <something> | grep <target>

```
0:20:16 ss8651twtw @ ub18 in ~/csc/lab
→ strings ./find | grep "puts"
puts
puts@@GLIBC_2.2.5
```

基本工具介紹

❖ 起手式 🌰

- EasyCTF IV - hexedit
- EasyCTF 2017 - hexable
- Reverse CTF - find

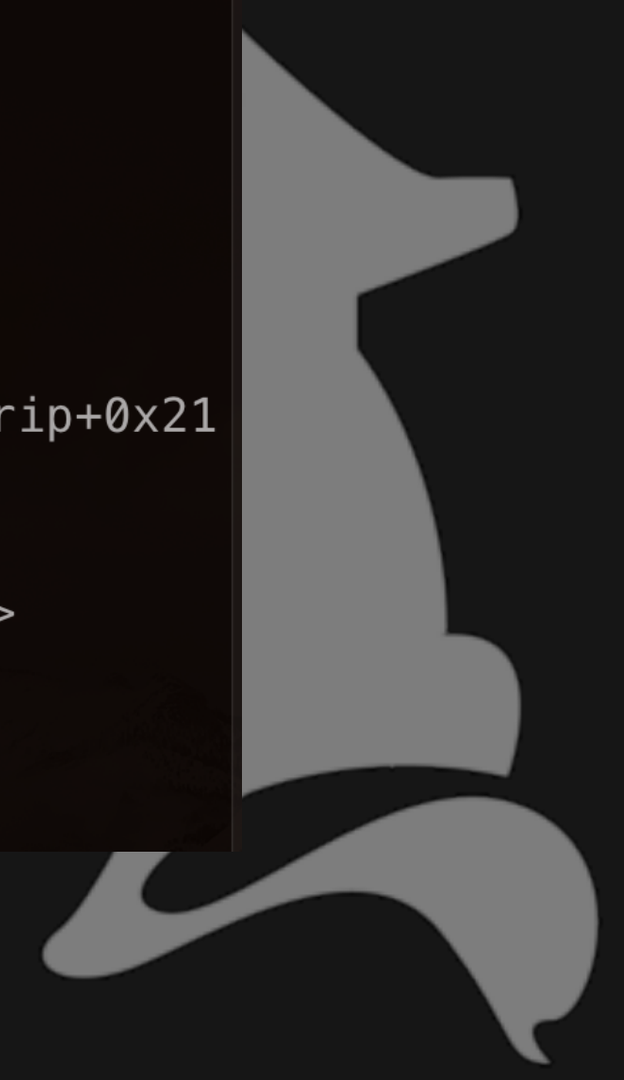
基本工具介紹

- objdump
  - 以 intel 格式顯示 binary 反組譯結果 (組合語言)
    - $ objdump -M intel -d <binary>
  - 把輸出結果導向到 less 方便查詢閱讀
    - $ objdump -M intel -d <binary> | less

基本工具介紹

$ alias objdump="objdump -M intel"

```
0:33:30 ss8651twtw @ ub18 in ~/csc/lab
→ objdump -d ./search

./search:     file format elf64-x86-64


Disassembly of section .init:

0000000000000530 <_init>:
 530:   48 83 ec 08             sub    rsp,0x8
 534:   48 8b 05 a5 1a 21 00    mov    rax,QWORD PTR [rip+0x21
1aa5]        # 211fe0 <__gmon_start__>
 53b:   48 85 c0                test   rax,rax
 53e:   74 02                   je     542 <_init+0x12>
 540:   ff d0                   call   rax
 542:   48 83 c4 08             add    rsp,0x8
 546:   c3                      ret
```



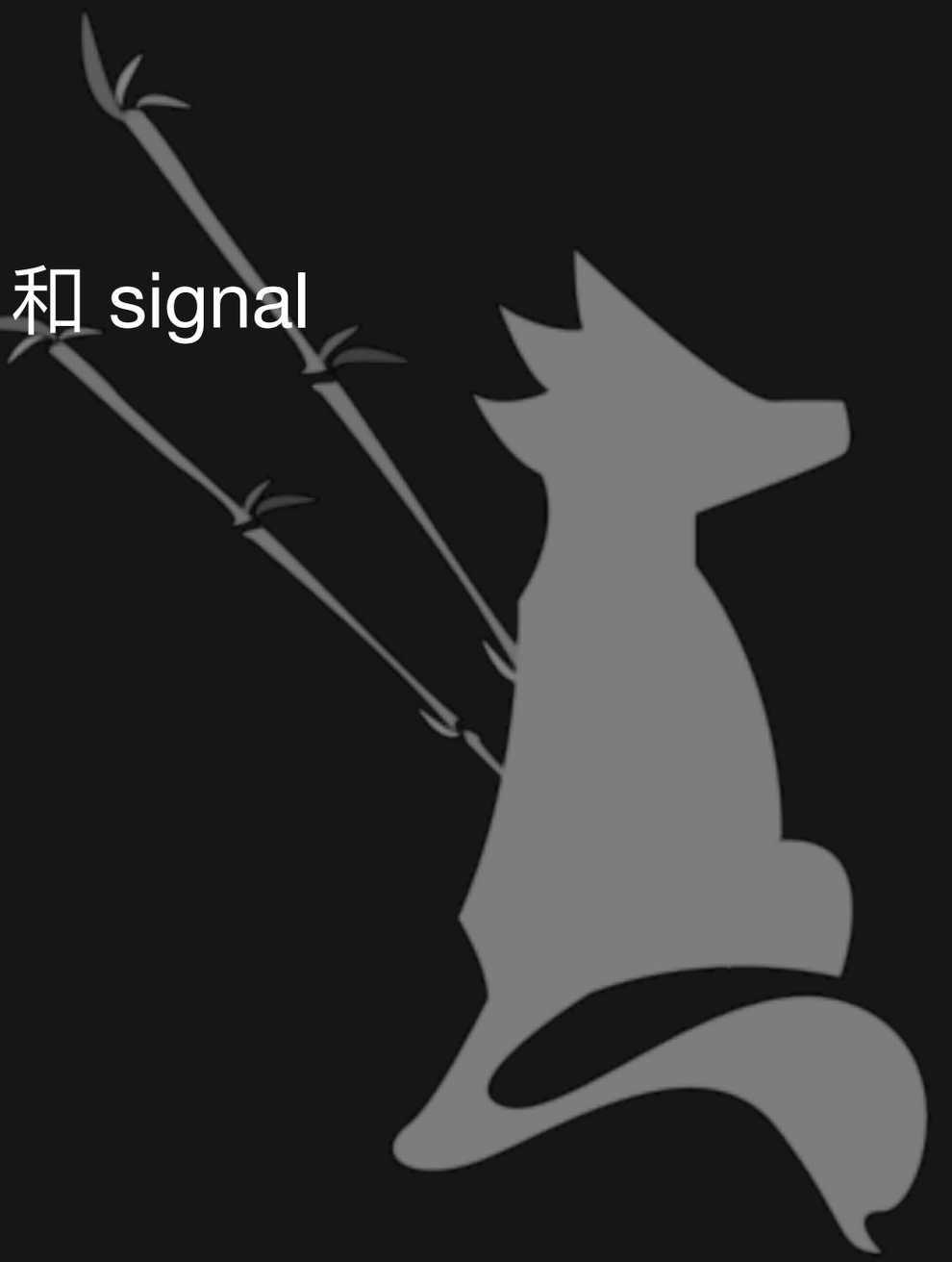
基本工具介紹

❖ objdump 🌰

- Reverse CTF - search
  - objdump
  - string processing

基本工具介紹

- strace / ltrace
  - 查看 binary 執行時的 system call 和 signal
    - $ strace <binary>
  - 查看 binary 執行時的 library call
    - $ ltrace <binary>



基本工具介紹

```
0:37:00 ss8651twtw @ ub18 in ~/csc/lab
→ strace ./strace
execve("./strace", ["./strace"], 0x7fffa5560aa0 /* 38 vars */)
 = 0
uname({sysname="Linux", nodename="ub18", ...}) = 0
brk(NULL)                                       = 0x19b2000
brk(0x19b31c0)                                  = 0x19b31c0
arch_prctl(ARCH_SET_FS, 0x19b2880)              = 0
readlink("/proc/self/exe", "/home/ss8651twtw/csc/lab/strace",
4096) = 31
```

基本工具介紹

```
0:41:29 ss8651twtw @ ub18 in ~/csc/lab
➜ ltrace ./hexedit
__libc_start_main(0x40052d, 1, 0x7ffe1644f138, 0x400550 <unfinished ...>
puts("Find the flag!"Find the flag!
)                                 = 15
+++ exited (status 0) +++
```

基本工具介紹

- strace / ltrace 🌰
  - CSIE 2017 - strace

基本工具介紹

❖ gdb

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

```
attach <pid>
```

gdb

基本工具介紹

- gdb
  - 套件安裝
    - peda - https://github.com/longld/peda
    - Pwngdb - https://github.com/scwuaptx/Pwngdb

基本工具介紹

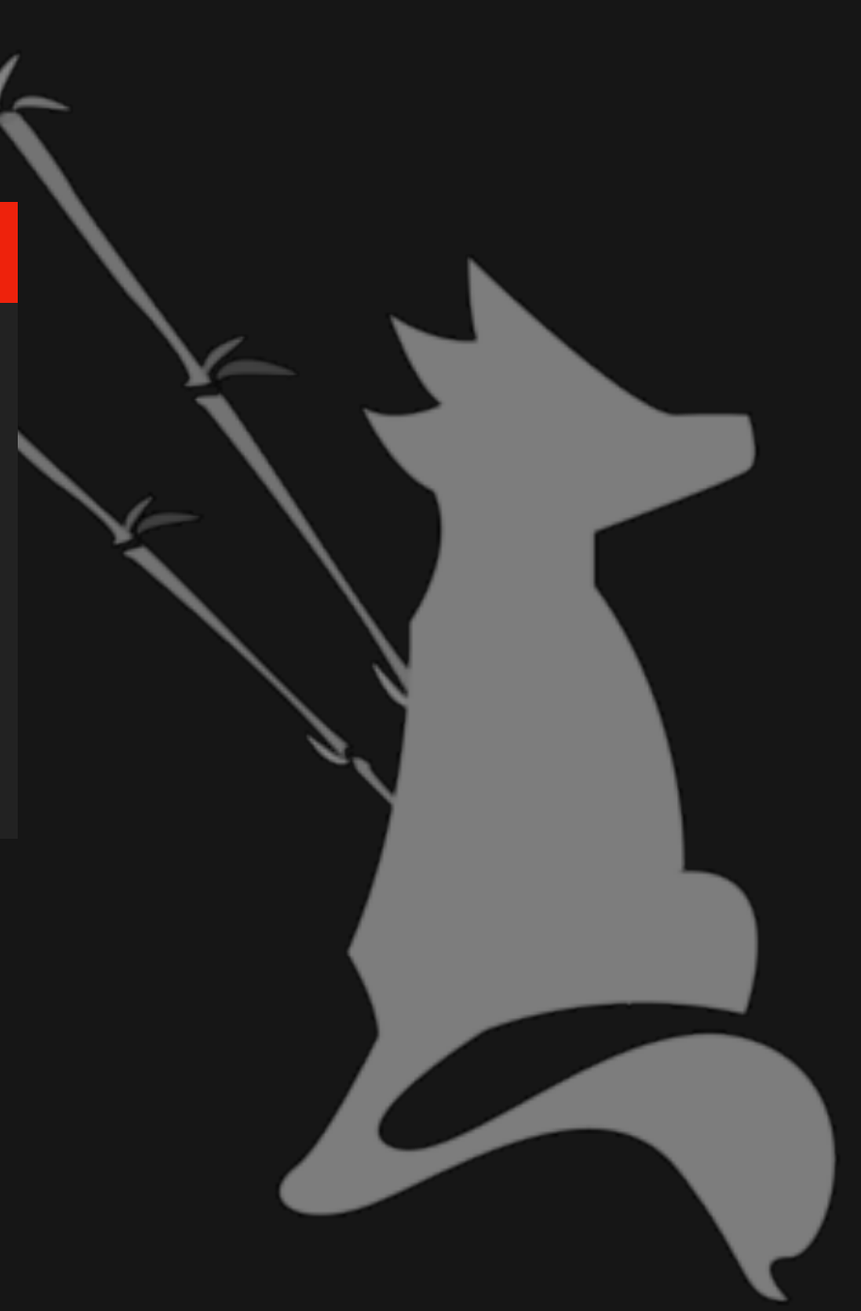
❖ gdb

- 套件安裝
  - $ git clone https://github.com/longld/peda.git ~/peda
  - $ git clone https://github.com/scwuaptx/Pwngdb.git ~/Pwngdb
  - $ cp ~/Pwngdb/.gdbinit ~/

基本工具介紹

gdb

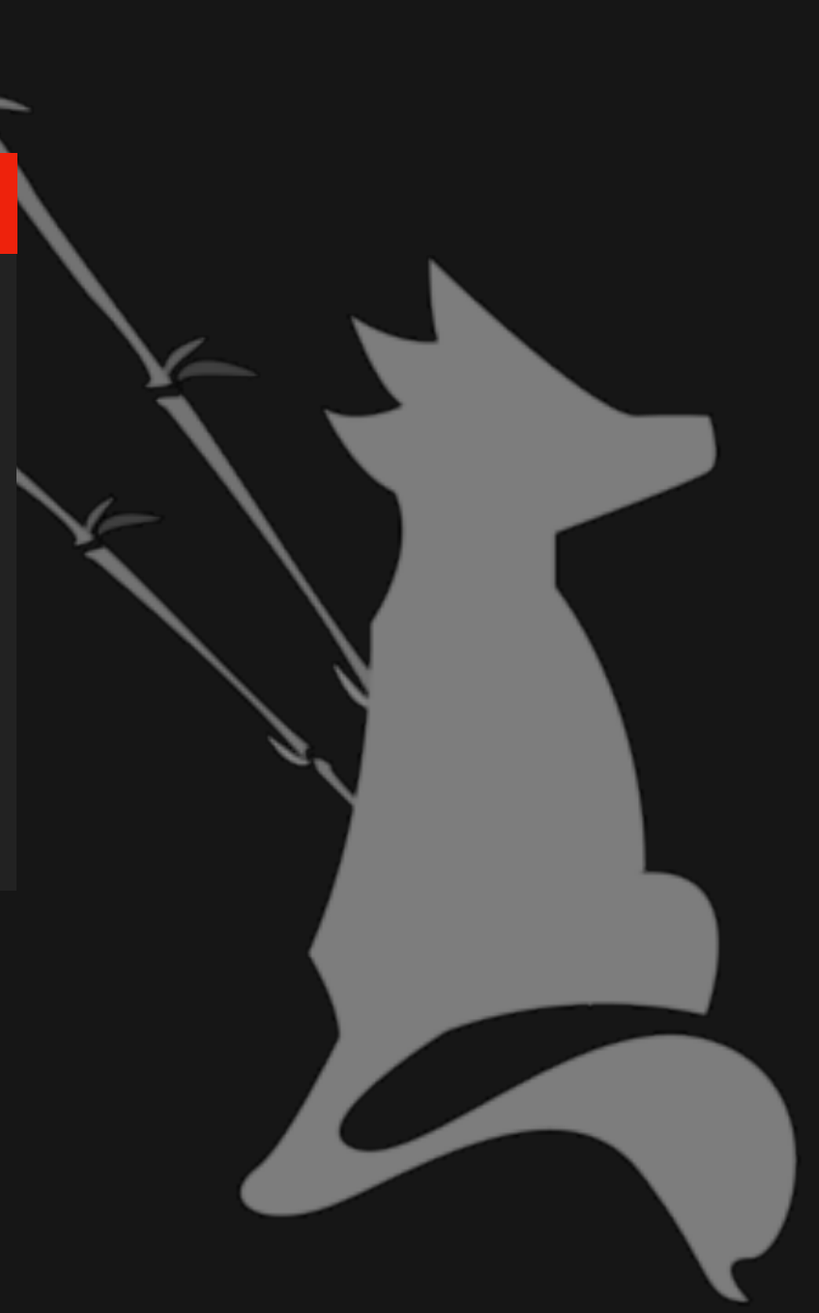
```
# 設定斷點
# break *<address>
break *0x4004d7

# 執行程式
run
```



基本工具介紹

gdb

```
# 執行下一個指令
# 會追進 function 中
step

# 執行下一個指令
# 不會追進 function 中
next
```



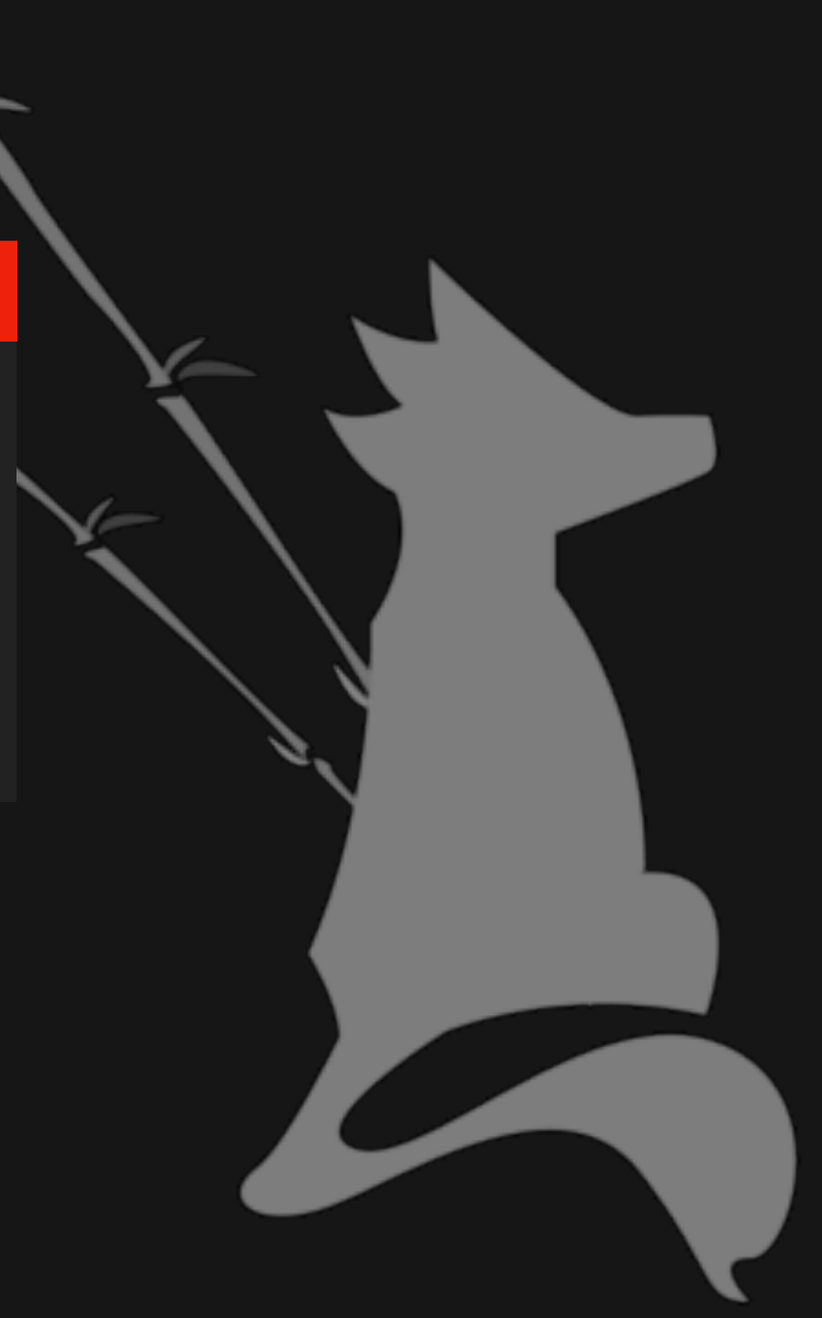
gdb

基本工具介紹

gdb

gdb

```
# 繼續執行
continue

# 執行至 function 結束
finish
```

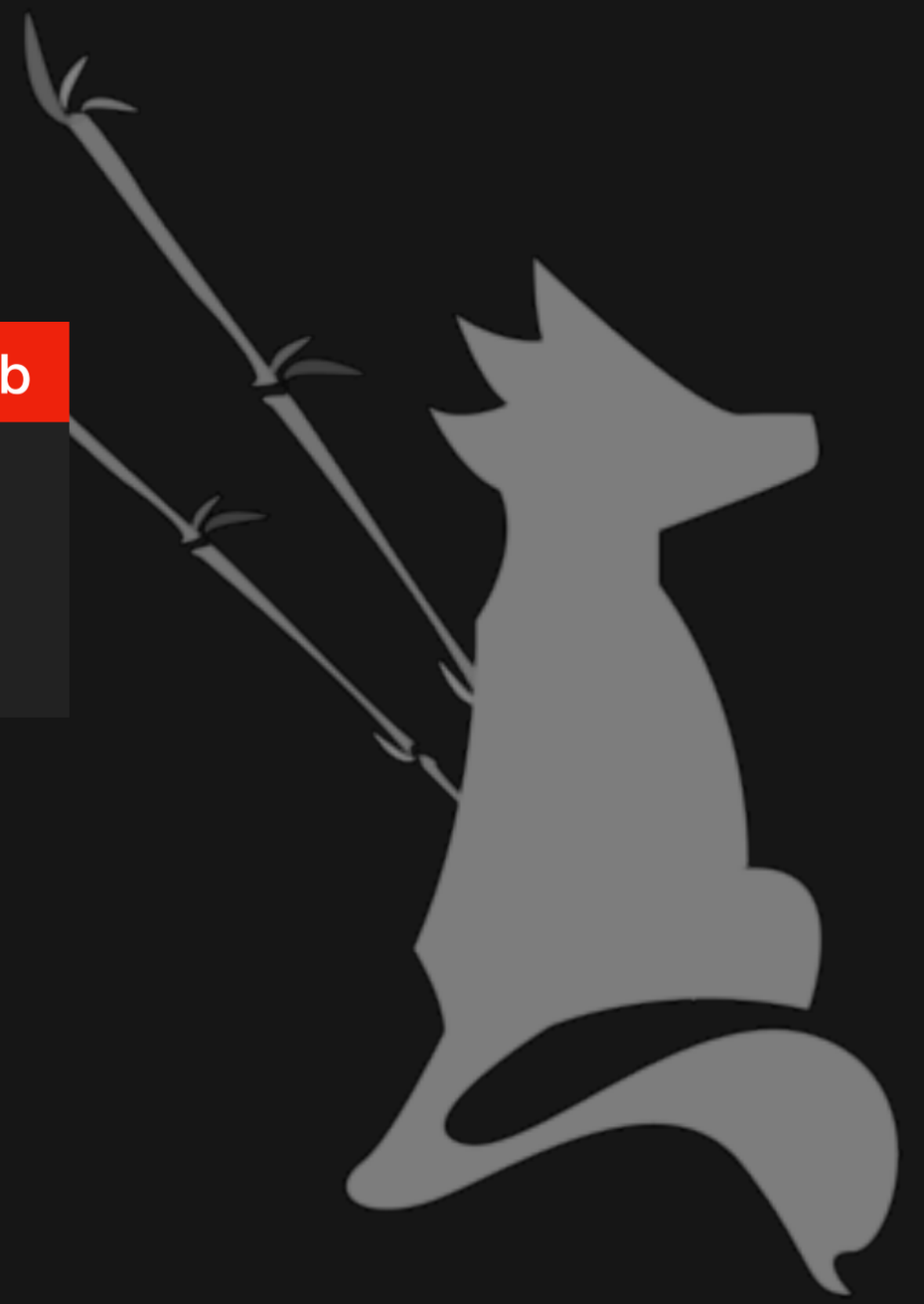


基本工具介紹

gdb

gdb

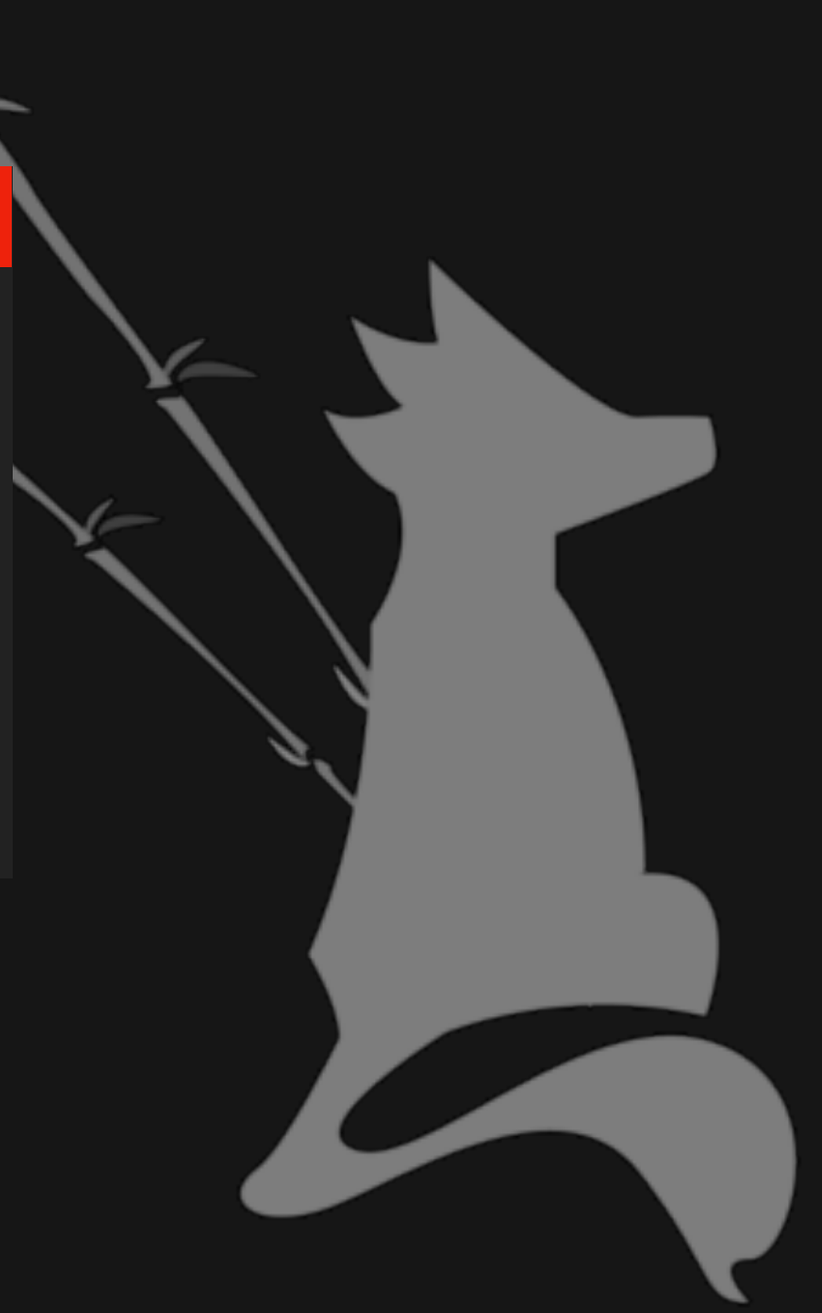
```
# 跳轉
# jump *<address>
jump *0x4004d7
```



基本工具介紹

gdb

```
# 印出暫存器的值
# print $<register>
print $rax

# 印出記憶體的值
# x <memory address>
x 0x7fffffffe920
```



基本工具介紹

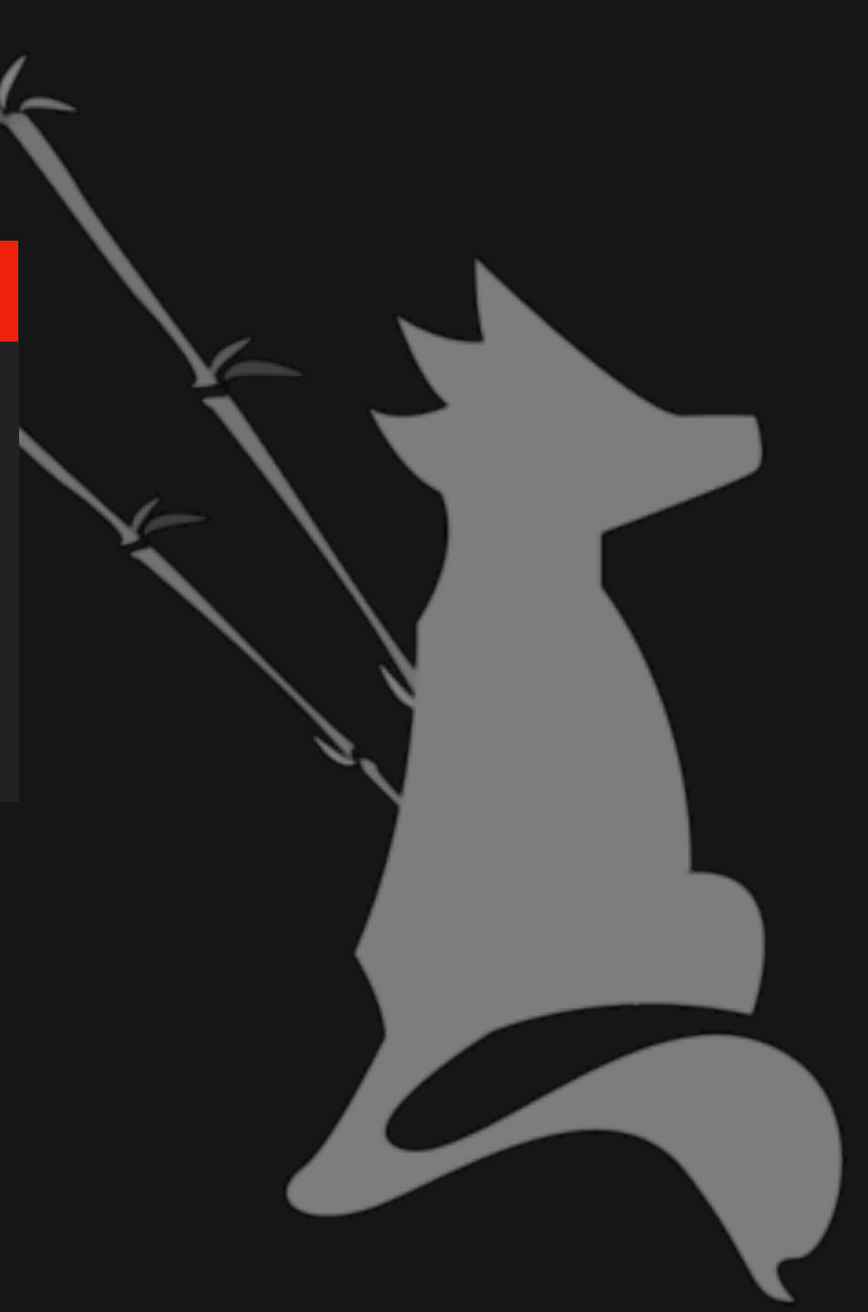
gdb

```gdb
# 設定暫存器的值
# set $<register> = <value>
set $rsp = 0x7fffffffe920

# 設定記憶體的值
# set {<size>} <memory address> = <value>
set {int} 0x7fffffffe920 = 2
```

基本工具介紹

gdb

```gdb
# 查看斷點狀態
info break

# 查看暫存器狀態
info register
```



Day1

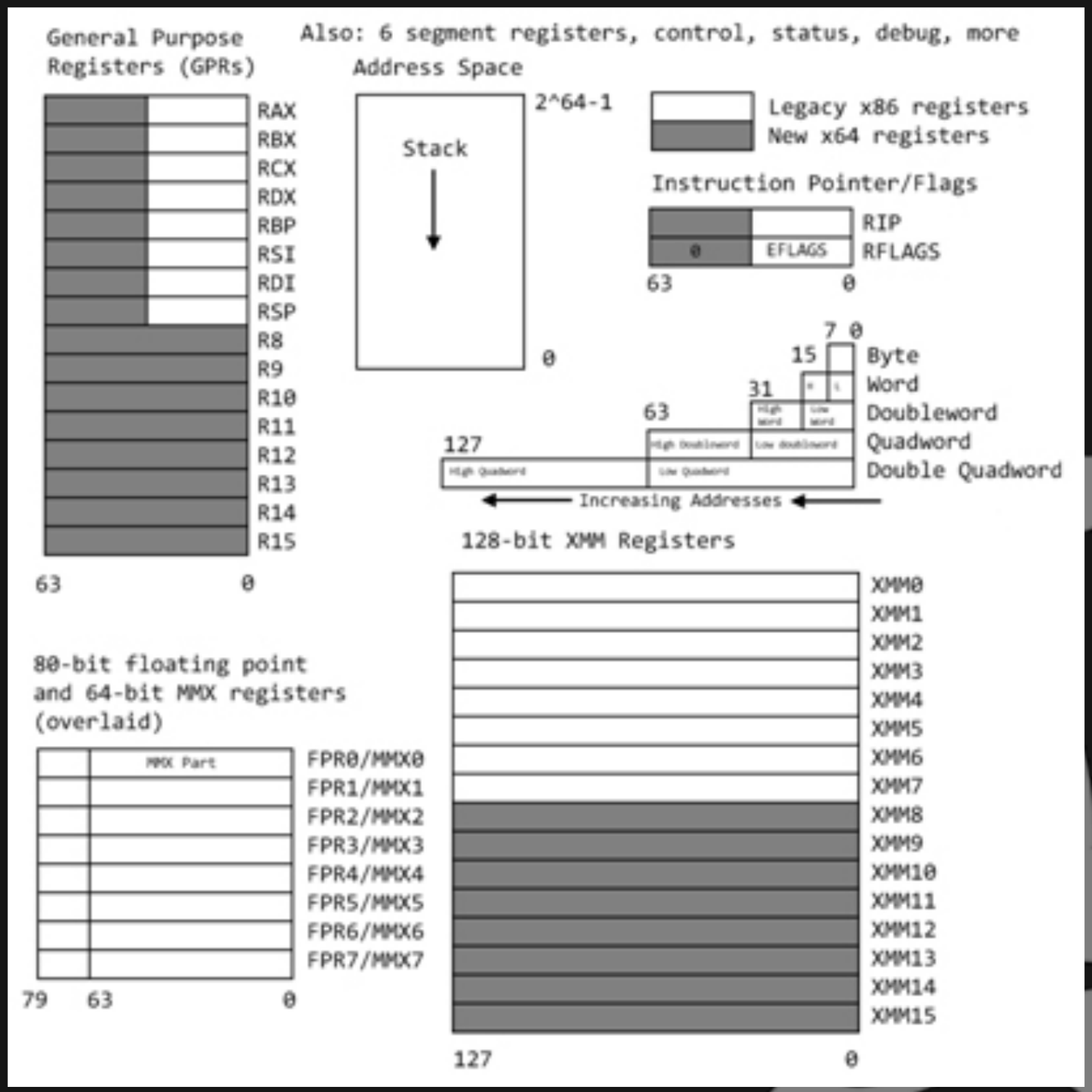
- 基本工具介紹
- x64 組合語言

x64 組合語言

- 暫存器
- stack
- 組合語言指令
- 編譯與執行
- 組合語言與 C 的轉換

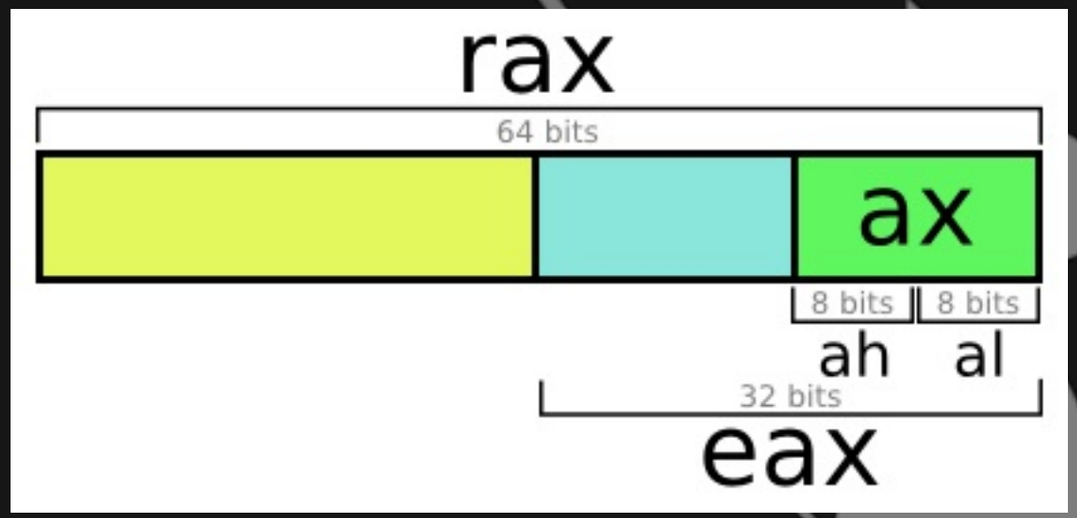
x64 組合語言

❖ 暫存器


General Purpose Registers (GPRs)
Also: 6 segment registers, control, status, debug, more
Address Space
RAX
RBX
RCX
RDX
RBP
RSI
RDI
RSP
R8
R9
R10
R11
R12
R13
R14
R15
63
0
Stack
2^64-1
0
Legacy x86 registers
New x64 registers
Instruction Pointer/Flags
RIP
EFLAGS
RFLAGS
63
0
7 0
15
31
63
127
Byte
Word
Doubleword
Quadword
Double Quadword
Increasing Addresses
80-bit floating point and 64-bit MMX registers (overlaid)
MMX Part
FPR0/MMX0
FPR1/MMX1
FPR2/MMX2
FPR3/MMX3
FPR4/MMX4
FPR5/MMX5
FPR6/MMX6
FPR7/MMX7
79
63
0
128-bit XMM Registers
XMM0
XMM1
XMM2
XMM3
XMM4
XMM5
XMM6
XMM7
XMM8
XMM9
XMM10
XMM11
XMM12
XMM13
XMM14
XMM15
127
0

x64 組合語言

- 通用暫存器
  - r[a-d]x
  - rsi, rdi
  - rbp, rsp
  - r[8-15]



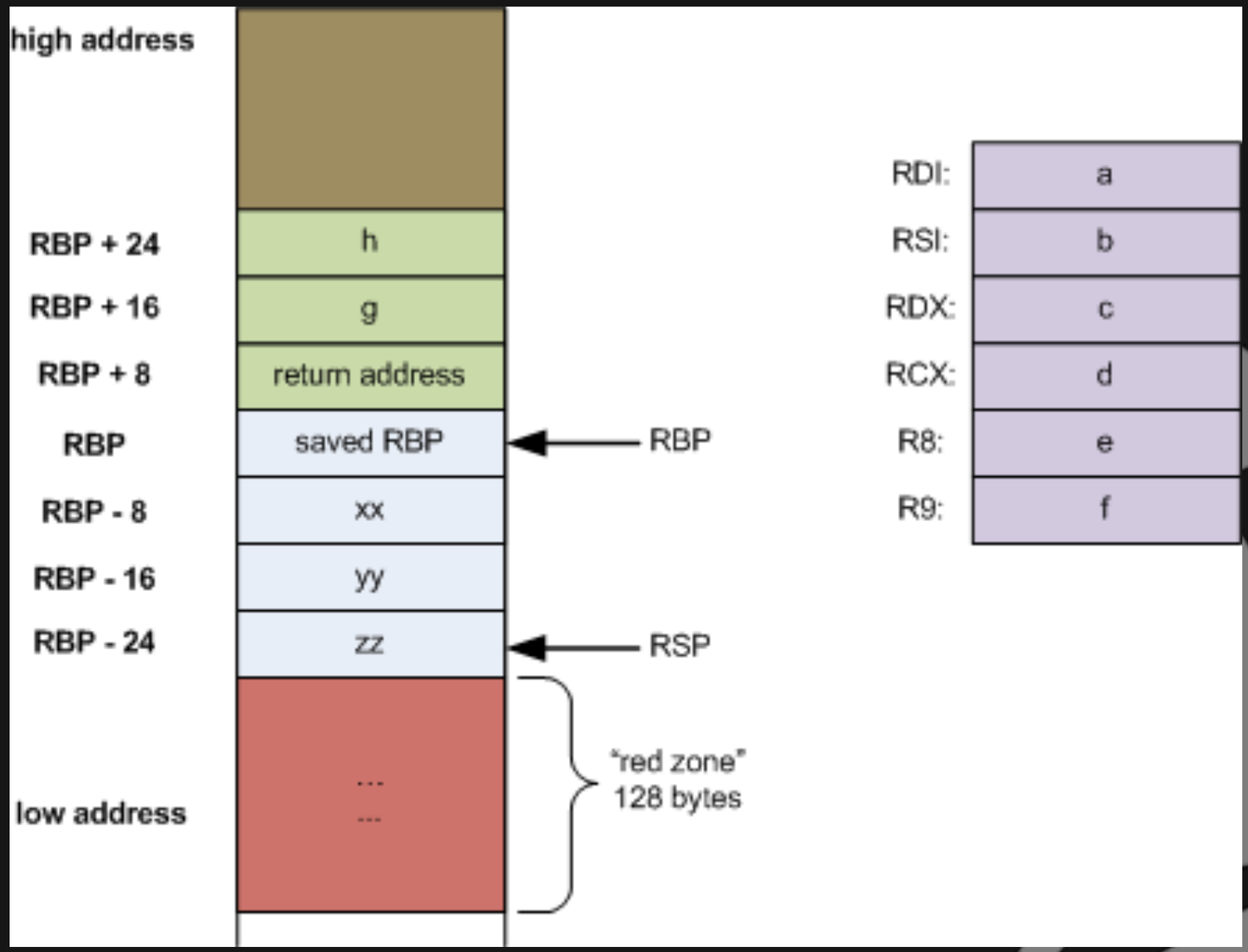
x64 組合語言

❖ stack

- rsp (top) 及 rbp (bottom) 所指向記憶體間的空間
- 紀錄區域變數、程式返回位置

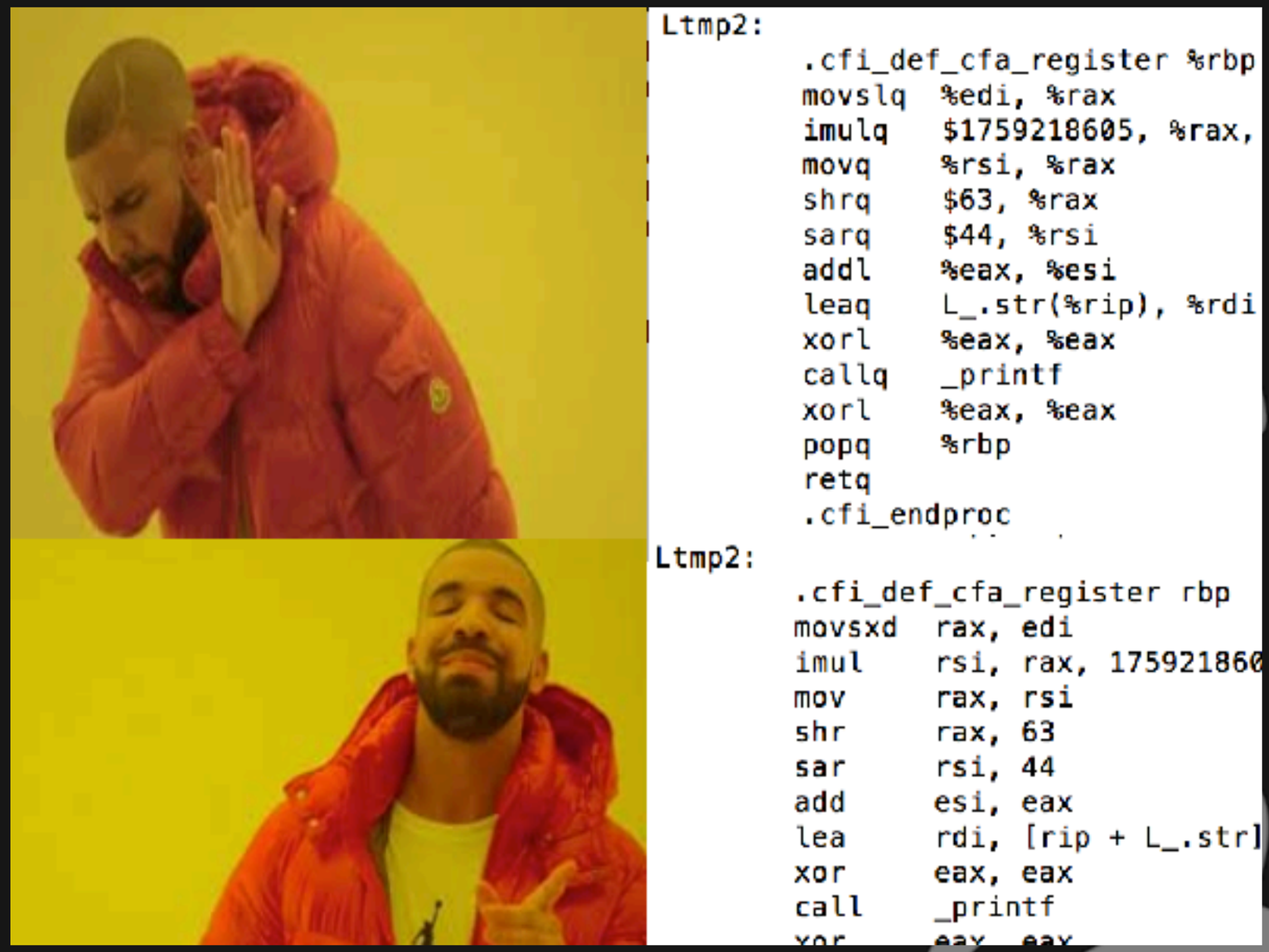
x64 組合語言

- stack



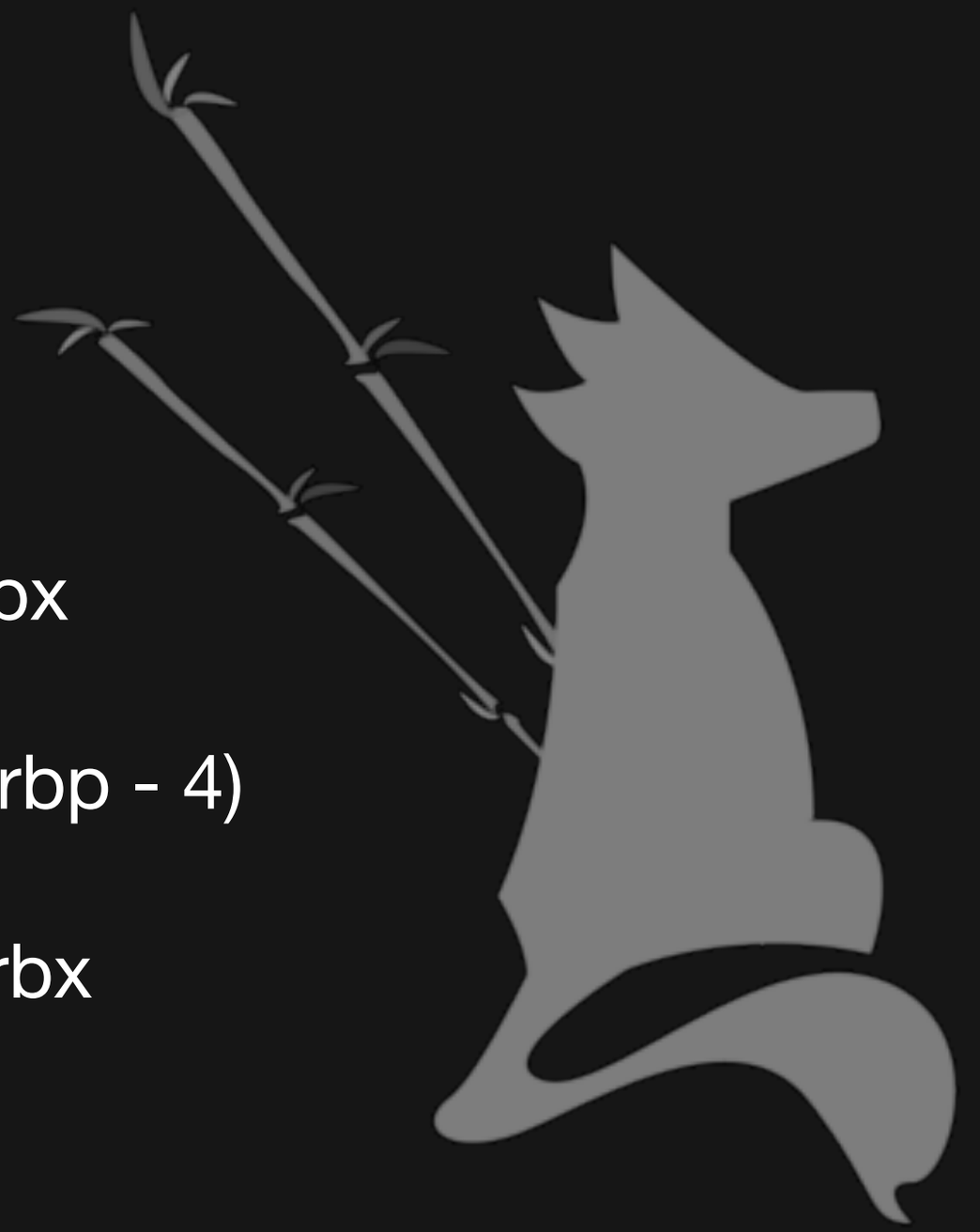
x64 組合語言

- ❖ 組合語言格式
  - AT&T
  - Intel



x64 組合語言

- 組合語言指令
- mov dst, src
- example
  - mov rax, rbx // rax = rbx
  - mov rax, [rbp - 4] // rax = *(rbp - 4)
  - mov [rax], rbx // *rax = rbx



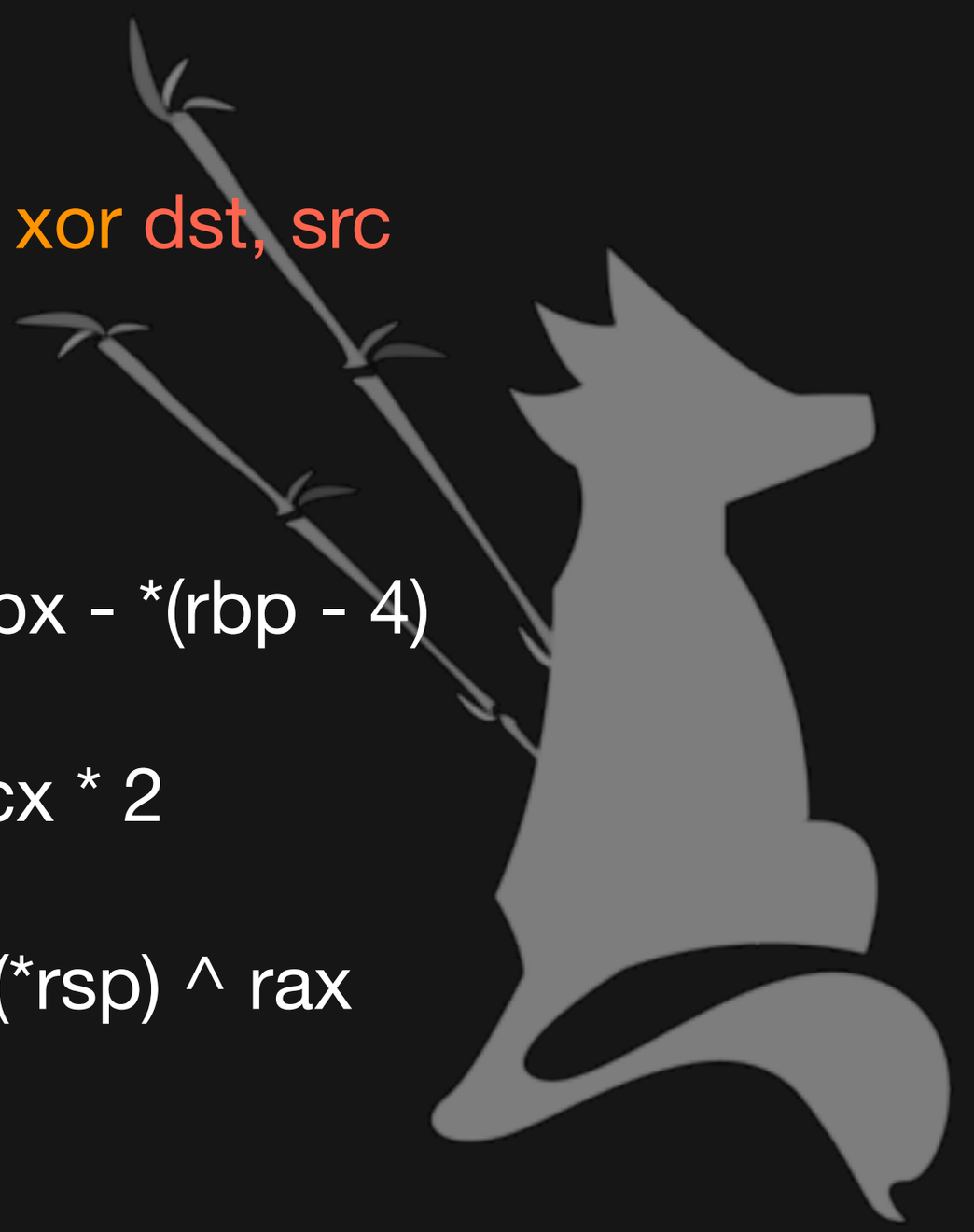
x64 組合語言

❖ 組合語言指令

❖ add、sub、imul、idiv、and、or、xor dst, src

❖ example

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



x64 組合語言

❖ 組合語言指令

❖ inc、dec、neg、not dst

❖ example

- dec rbx // rbx -= 1
- neg rcx // rcx = -rcx
- not byte [rsp] // convert [rsp] to byte
  *rsp = ~(*rsp)

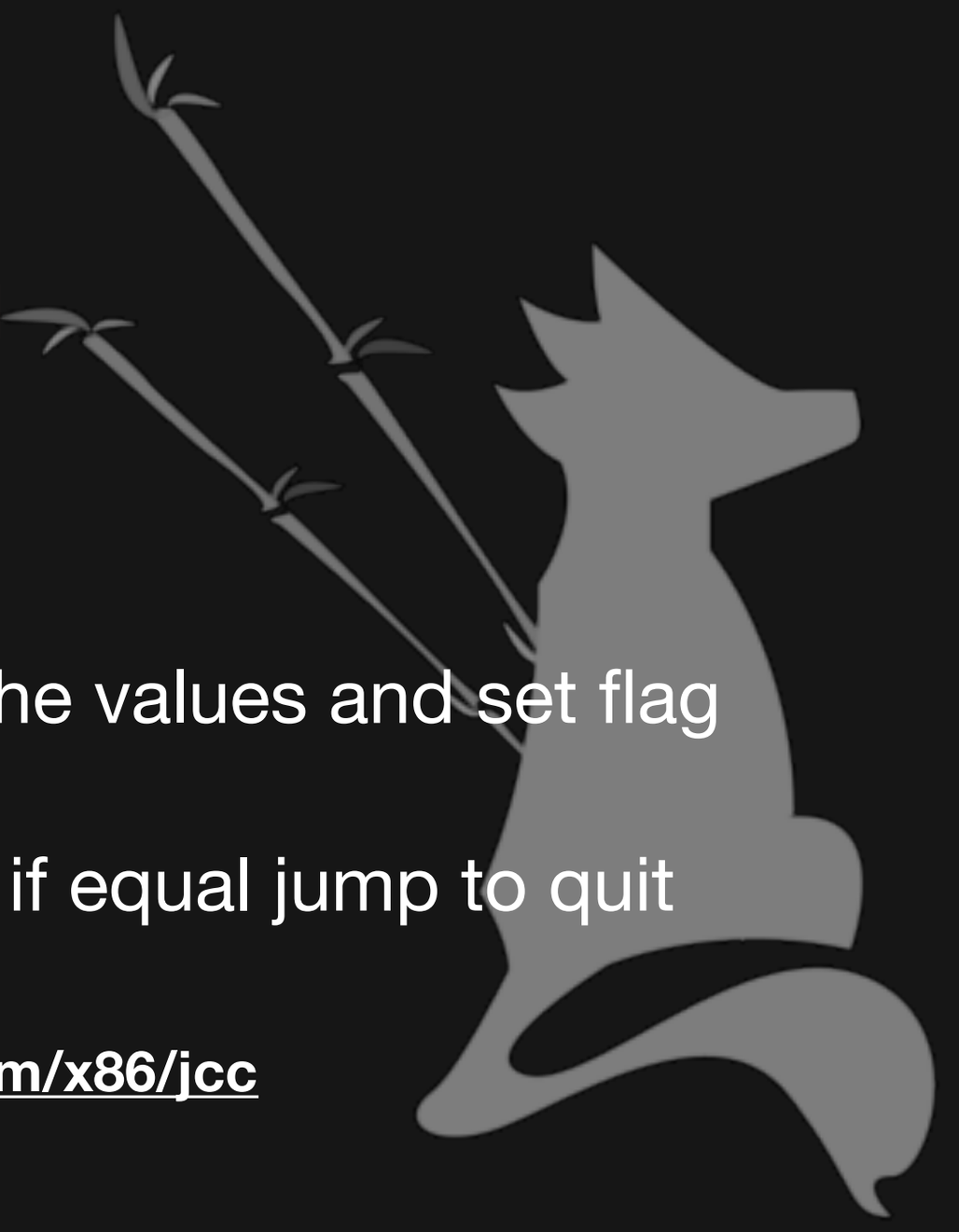
x64 組合語言

- 組合語言指令
- cmp val1, val2
- example
  - cmp rax, 5　　// compare the values and set the flag
  - cmp rbx, rcx
  - cmp word [rsp], 0x1234

x64 組合語言

- 組合語言指令
- jmp label
- example
  - loop: // set a label
  - ; do something
  - jmp loop // jump to loop label

x64 組合語言



❖ 組合語言指令

❖ ja、jb、jna、jbe、je、jne、jz label

❖ example

- cmp rax, 10 // compare the values and set flag
- je quit // check flag if equal jump to quit

https://www.felixcloutier.com/x86/jcc

x64 組合語言

- 組合語言指令
- nop
- example
  - nop　　// 什麼事都不做
  - 在 patch 的時候很好用!!!

x64 組合語言

- 組合語言指令
- syscall
- example

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

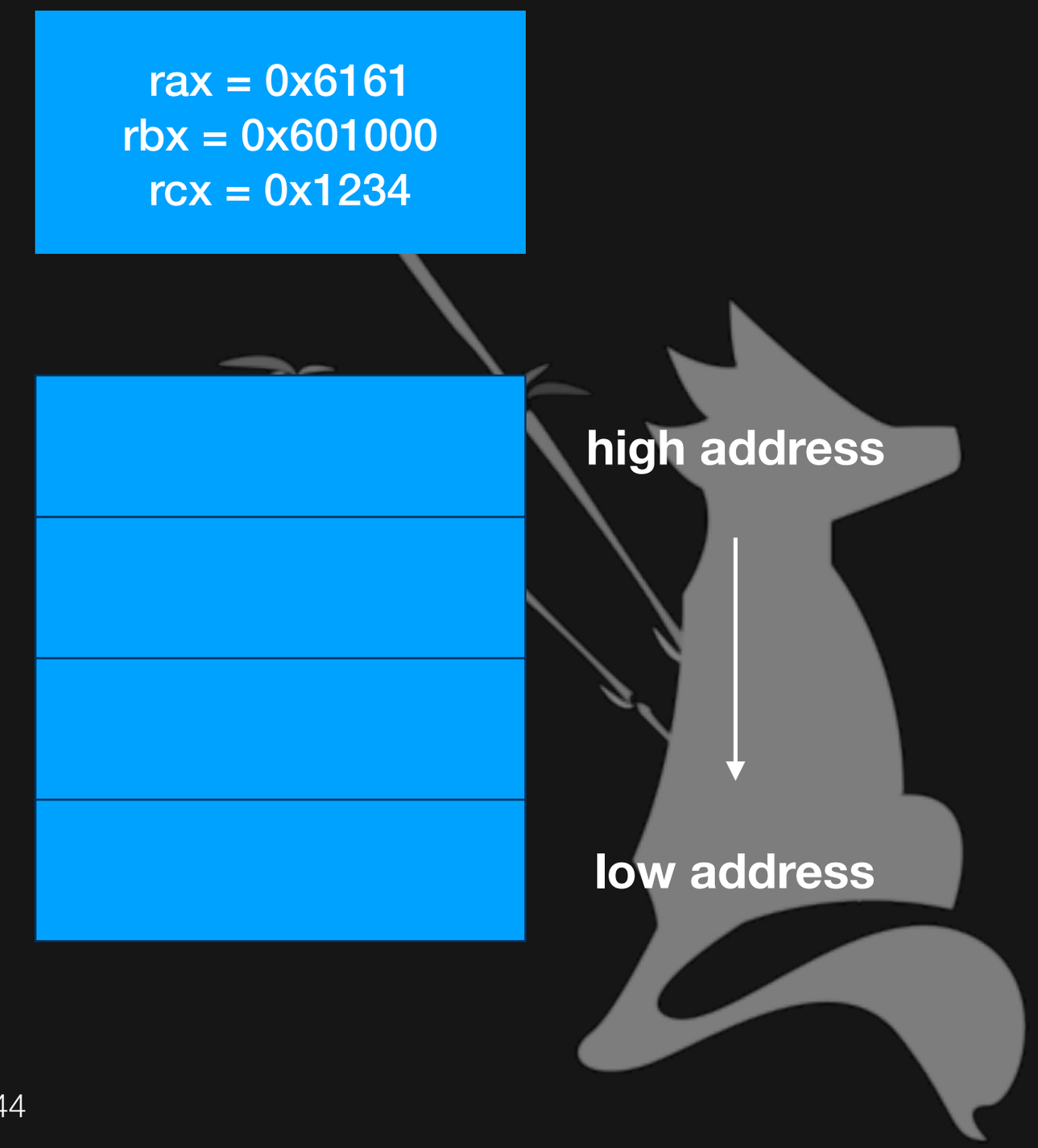
http://blog.rchapman.org/posts/Linux_System_Call_Table_for_x86_64/

x64 組合語言

- 組合語言指令
- push、pop val
- example
  - push rax
  - push 0
  - pop rcx
  - pop word [rbx]

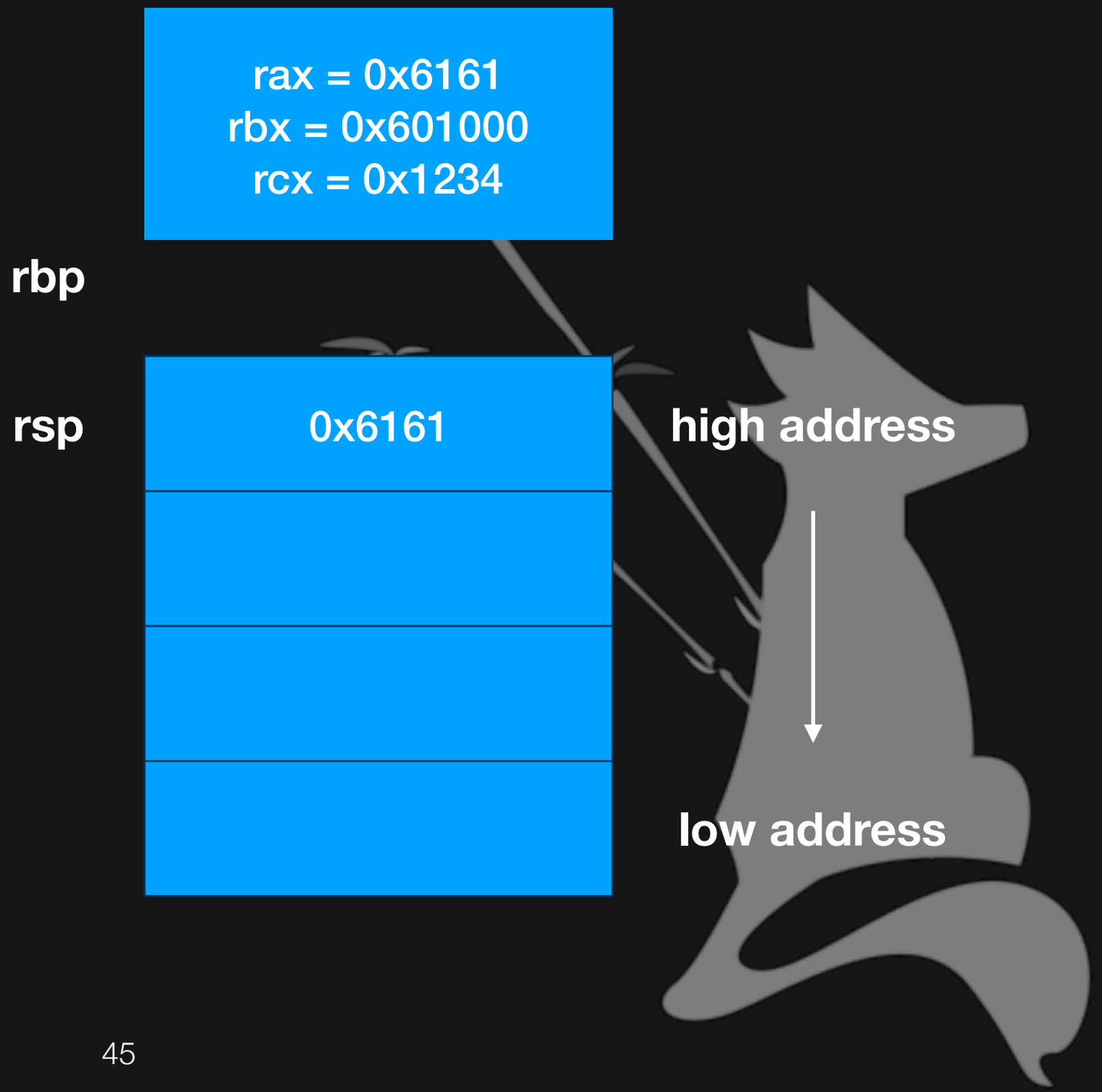
x64 組合語言

- 組合語言指令
- push、pop val
- example
  - push rax
  - push 0
  - pop rcx
  - pop word [rbx]



x64 組合語言

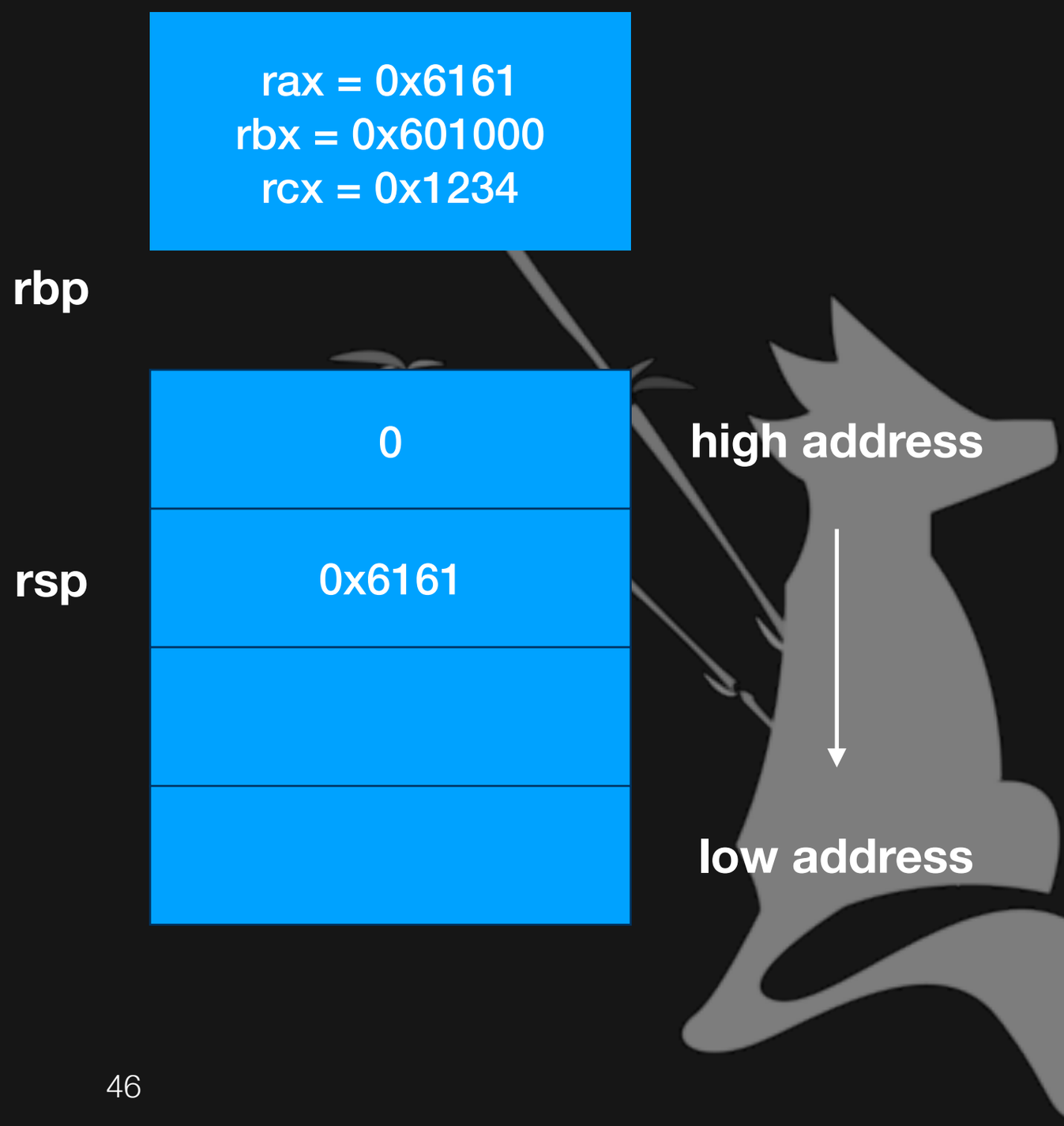
- 組合語言指令
- push、pop val
- example
  - push rax
  - push 0
  - pop rcx
  - pop word [rbx]



x64 組合語言

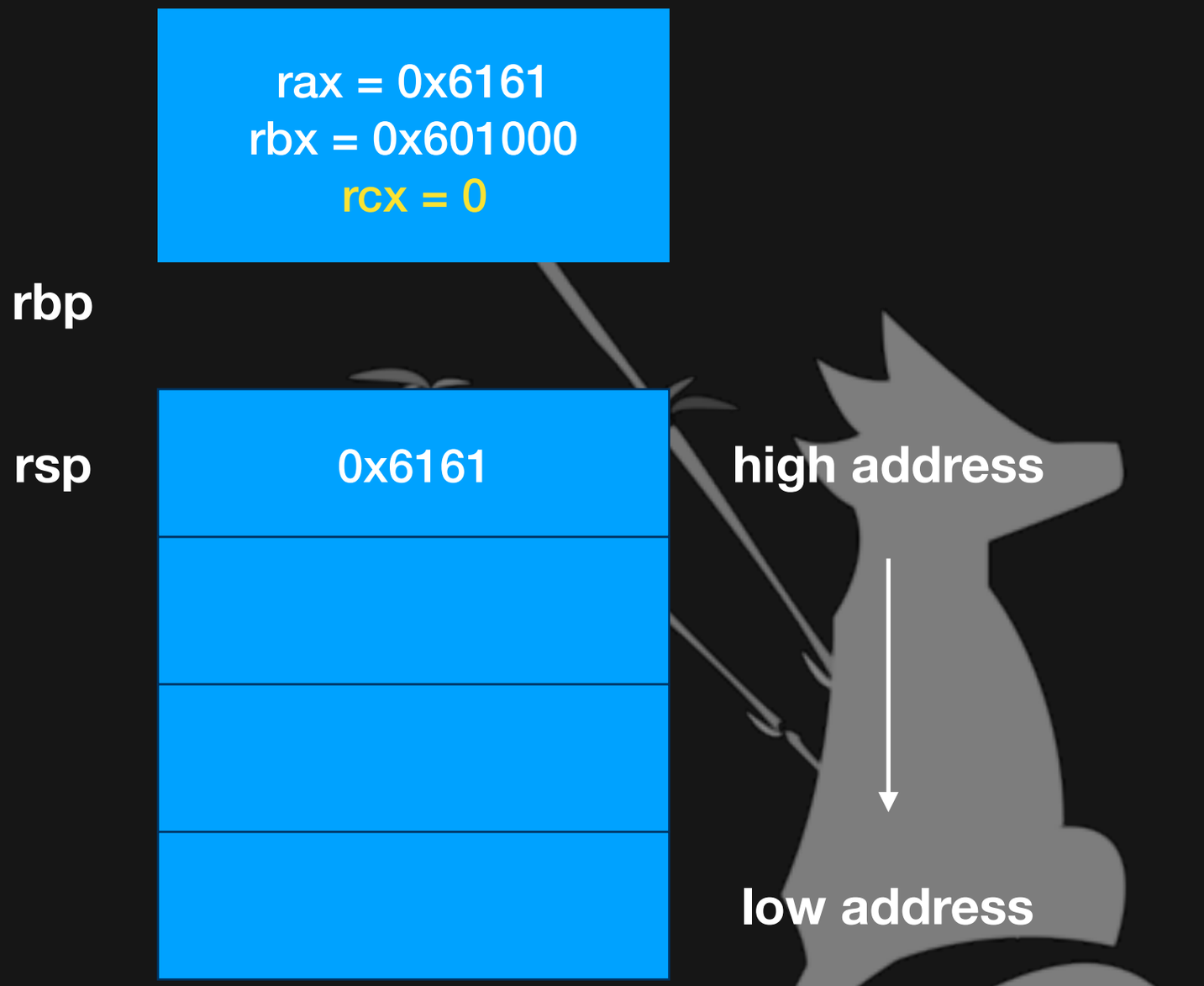
- 組合語言指令
- push、pop val
- example
  - push rax
  - push 0
  - pop rcx
  - pop word [rbx]

rax = 0x6161
rbx = 0x601000
rcx = 0x1234



x64 組合語言

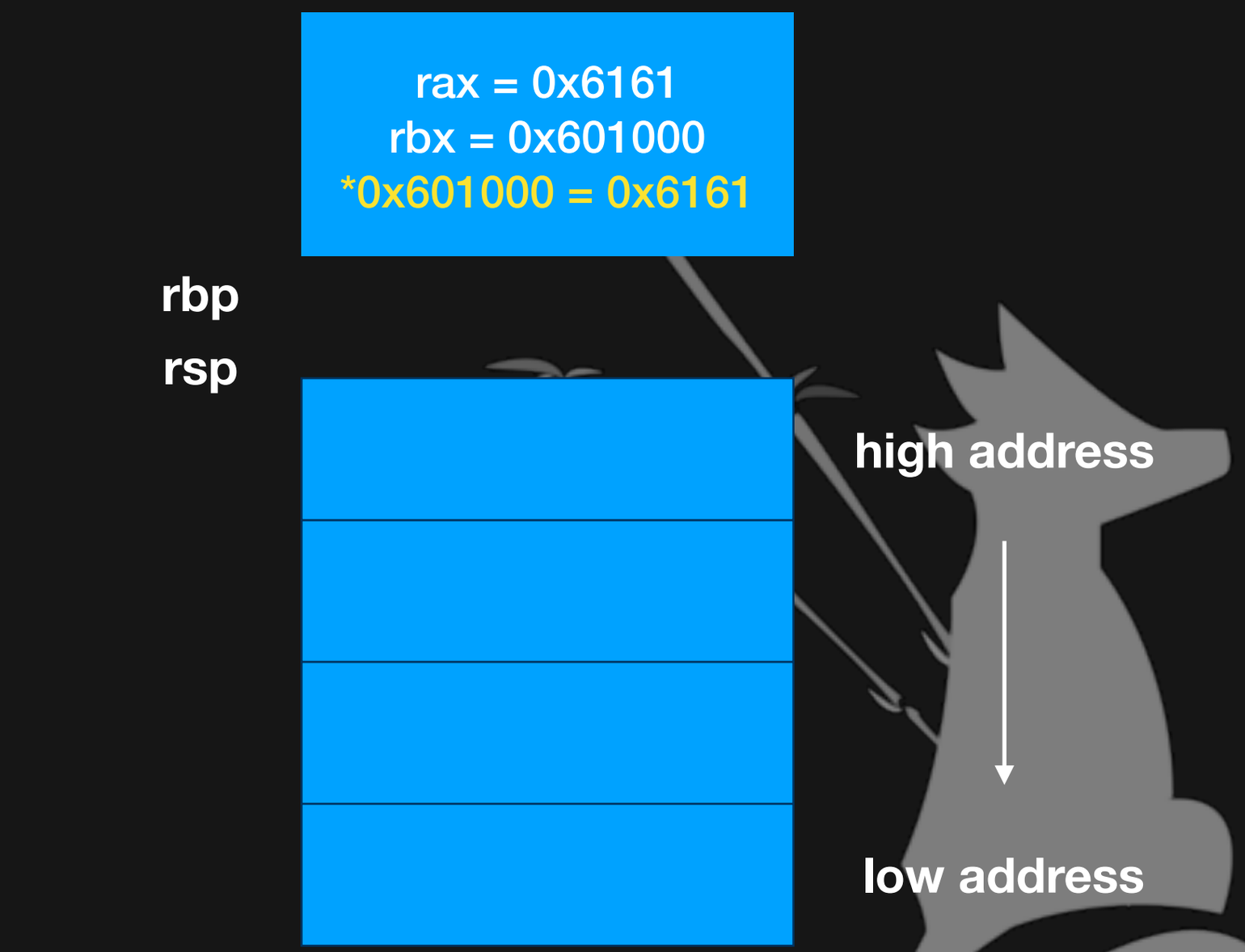
- 組合語言指令
- push、pop val
- example
  - push rax
  - push 0
  - pop rcx
  - pop word [rbx]



x64 組合語言

- 組合語言指令
- push、pop val
- example
  - push rax
  - push 0
  - pop rcx
  - pop word [rbx]

rax = 0x6161
rbx = 0x601000
*0x601000 = 0x6161

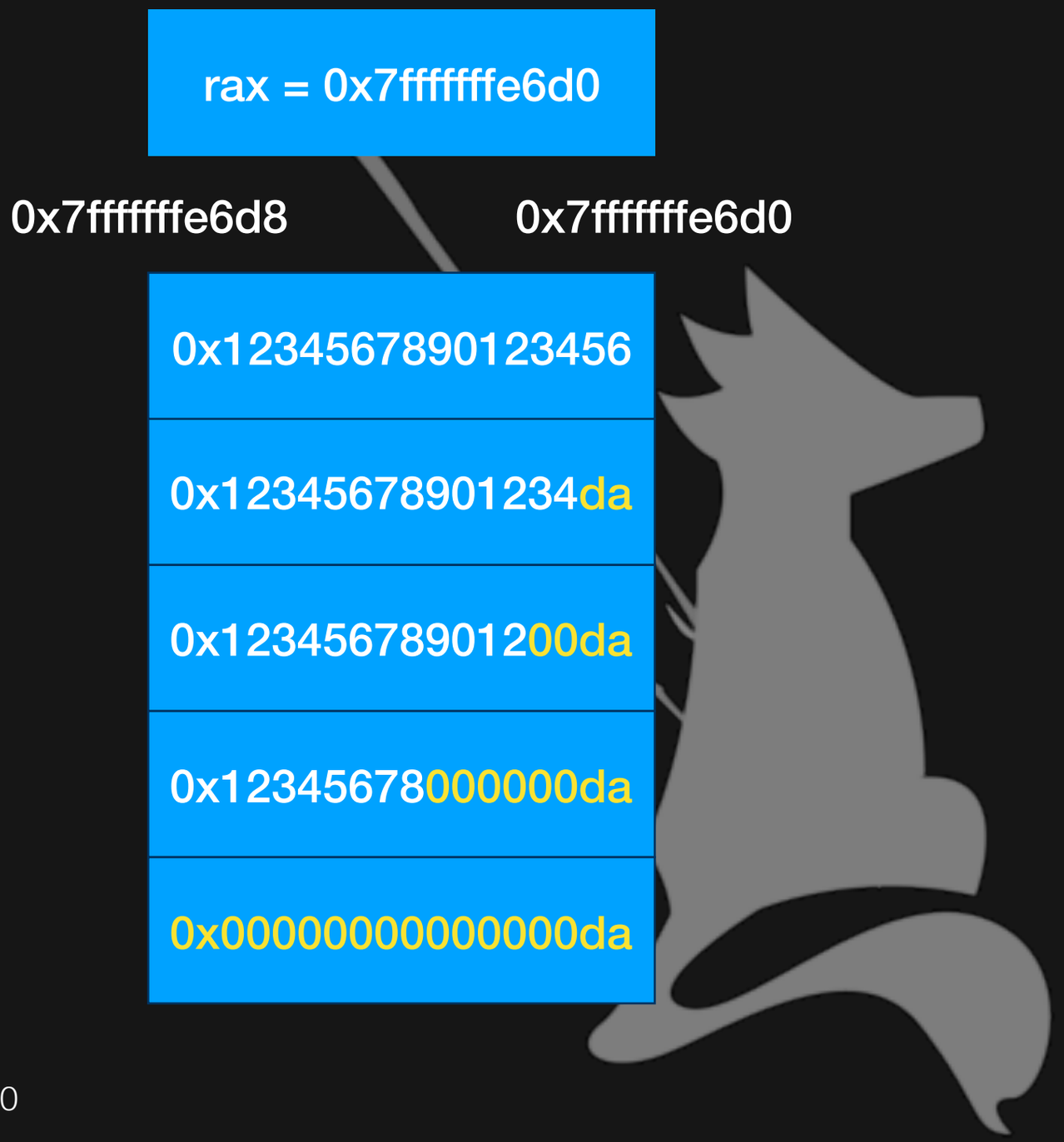


x64 組合語言

- ❖ 組合語言指令
- ❖ 標注型態
  - ▸ 若 dst 為 memory address 且 src 不為 register
  - ▸ 則需要標註 dst 的型態
  - ▸ byte → word → dword → qword
  - ▸ 8 bits → 2 bytes → 2 words → 2 dwords

x64 組合語言

- 組合語言指令
- 標注型態
- example
  - mov byte [rax], 0xda
  - mov word [rax], 0xda
  - mov dword [rax], 0xda
  - mov qword [rax], 0xda



x64 組合語言

❖ 組合語言指令 🌰

- Reverse CTF - read-asm

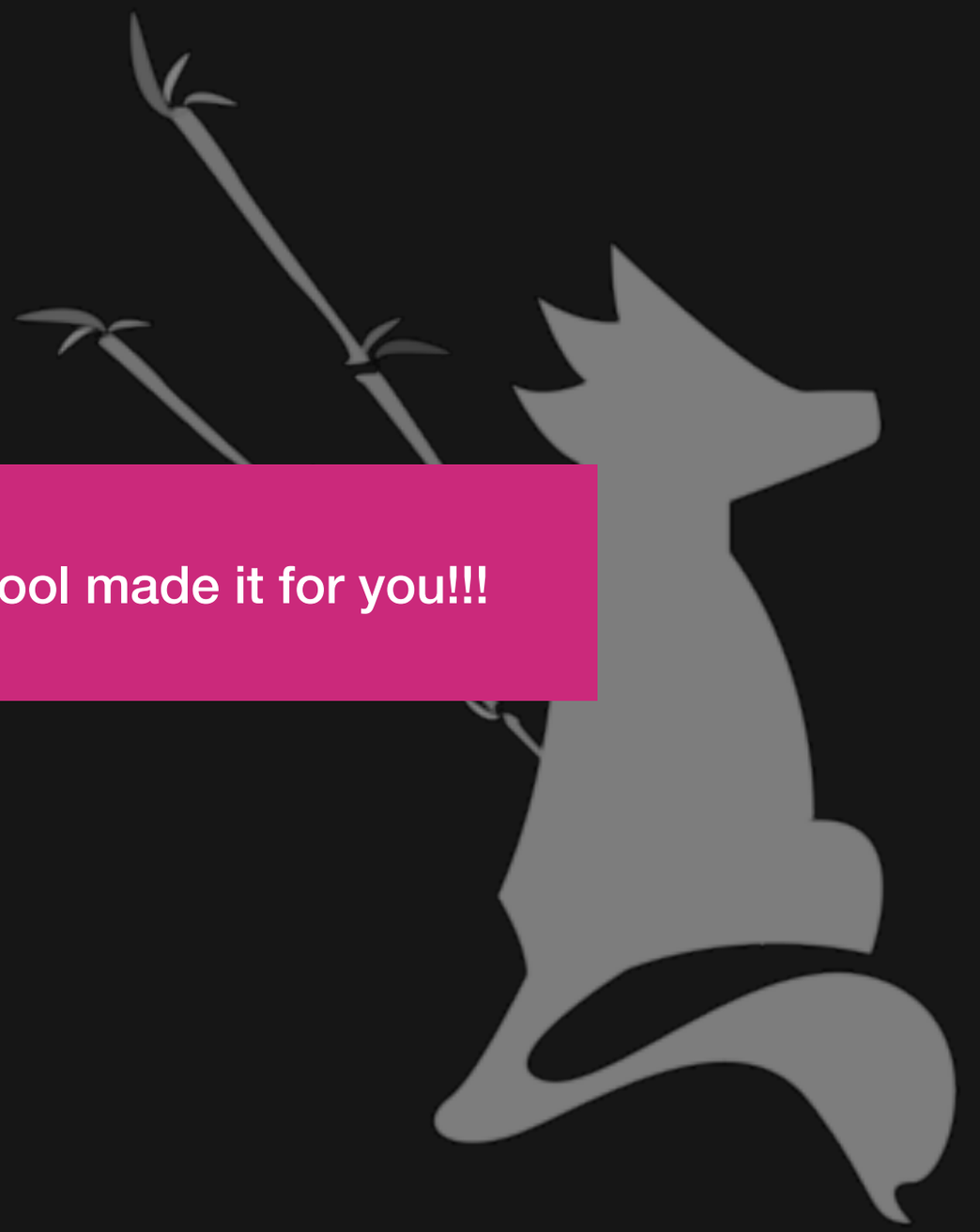
x64 組合語言

- ❖ 動手試試看
  - ▸ 寫組合語言
  - ▸ 編譯成機器碼
  - ▸ 連結成可執行檔
  - ▸ 執行!!!

x64 組合語言

❖ 動手試試看

- 寫組合語言
- 編譯成機器碼
- 連結成可執行檔
- 執行!!!



x64 組合語言

python2

```
from pwn import *

# write asm here
sc = """
    xor rax, rax
    inc rax
    mov rbx, 7
    add rax, rbx
    neg rbx
"""

# set x64 arch
context.arch = "amd64"

# print the binary path
print(make_elf_from_assembly(sc))
```

x64 組合語言

❖ 編譯與執行

- $ nasm -f elf64 solve.asm
  - -f elf64 // output elf 64 format
- $ ld -m elf_x86_64 -o solve solve.o
  - -m elf_x86_64 // x86-64 elf format
  - -o // output file name

x64 組合語言

❖ 編譯與執行 🌰

- Reverse CTF - run-asm

x64 組合語言

- 組合語言與 C 的轉換
- 條件語句

c

```c
if (rax < 5) {
    foo1();
}
else if (rax >= 5 && rax < 10) {
    foo2();
}
else {
    foo3();
}
```

asm

```asm
    cmp     rax, 0x4
    jg      Lelseif
    call    foo1
    jmp     Lend
Lelseif:
    cmp     rax, 0x4
    jle     Lelse
    cmp     rax, 0x9
    jg      Lelse
    call    foo2
    jmp     Lend
Lelse:
    call    foo3
Lend:
    ret
```

x64 組合語言

- 組合語言與 C 的轉換
- 循環語句

```c
for (rax = 0; rax < 10; rax++) {
    foo1();
}
```

```asm
    mov     rax, 0x0
    jmp     Lchk
Lbody:
    call    foo1
    add     rax, 0x1
Lchk:
    cmp     rax, 0x9
    jle     Lbody
    ret
```

x64 組合語言

❖ 組合語言與 C 的轉換

❖ 函數

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

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

x64 組合語言

❖ 組合語言與 C 的轉換

❖ 函數

- prologue and epilogue
- TBD

x64 組合語言

- 組合語言與 C 的轉換
- 函數 TBD

x64 組合語言

❖ 組合語言與 C 的轉換 🌰

- EasyCTF IV - adder
- EasyCTF 2017 - LuckyGuess
- EasyCTF IV - liar
- EasyCTF IV - ezreverse

x64 組合語言

❖ 組合語言練習

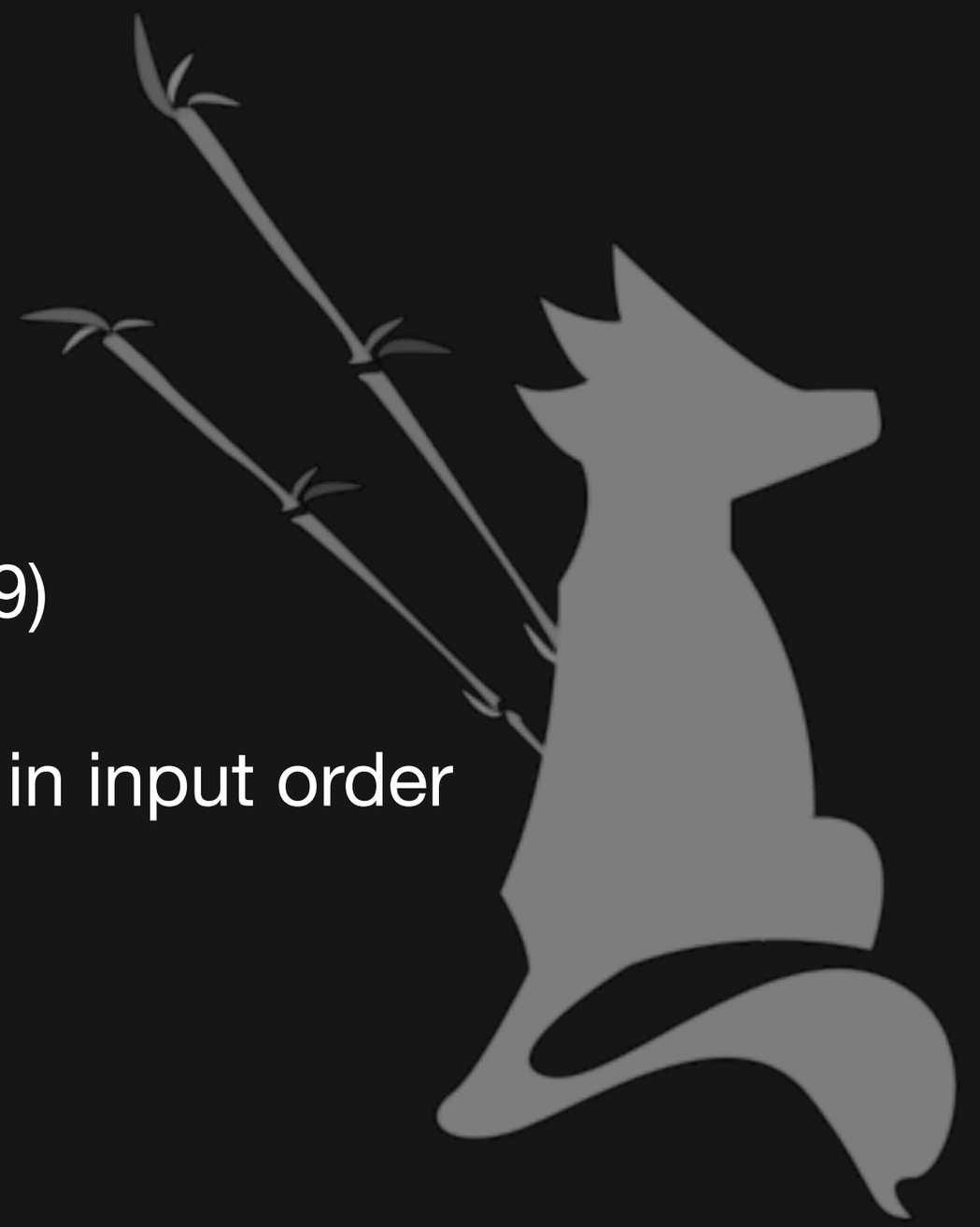
- 簡單的實作題目
  - $ git clone https://github.com/ss8651twtw/asm-practice.git

x64 組合語言

❖ 組合語言練習

- Hello World!
  - using write syscall to print “Hello World!”

x64 組合語言



❖ 組合語言練習

- string split
  - input string s (1 <= len(s) <= 9)
  - output a character every line in input order

x64 組合語言

❖ 組合語言練習

- calculate N!
  - input N (1 <= N <= 9)
  - using recursive function to calculate N!
  - output store in rax

Reference

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

Thanks for listening!