新竹人最愛逆向(工程)

terrynini38514

terrynini

WHO AM I ?

ID : Terrynini38514

- 跟我有點熟：
  逆逆
- 跟我麻吉麻：
  國立交通大學SENSE LAB
  資電玄客與安全碩士學位學程陳廷宇
- 稍微能拿來吹的東西：
  2019 年金盾獎冠軍
  2019, 2020 FireEye Flare On Challenge 破台
- CTF Team：
  DoubleSigma（已慘遭 Balsn 併吞化作其血肉）
  Balsn

沒照片可用 救命

INTRODUCTION & REVIEW

(軟體)逆向工程是什麼？

- 在缺少原始碼的情況下分析程式的設計
  - 惡意程式分析 e.g. WannaCry
  - 軟體行為分析 e.g. zoom
  - 開發軟體周邊應用 e.g. nouveau, Samba, Skype protocol
  - 挖掘軟體漏洞 e.g. Windows, iPhone
  - 非法行為 e.g. 盜版軟體, Keygen, 把手機遊戲裡的貼圖尻出來放進 telegram

抵禦逆向工程

- 增加逆向工程的成本
  - 加密
  - 裝死
  - 誤導
  - 程式碼混淆
  - 使執行檔難以取得
  - 經常性更新

主要分析方法

- 靜態分析
  - 不執行目標程式，直接分析程式的執行檔
  - 分析 program
- 動態分析
  - 完整或部分執行目標程式，直接或間接觀察其行為
  - 分析 process

主要分析工具

靜態分析

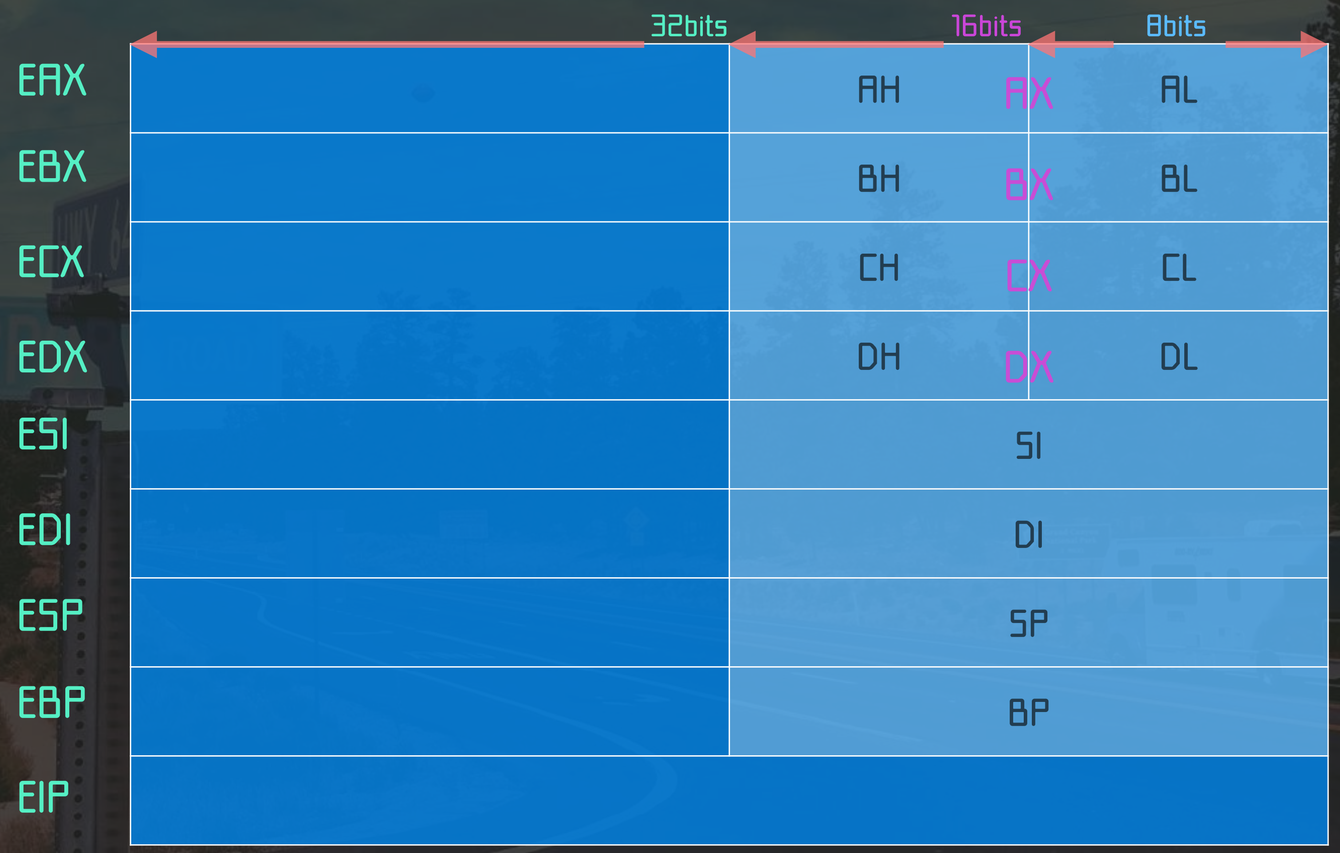
- IDA pro
  貴,潮,強大
- Ghidra (建議課堂中使用)
  NSA 開源工具, Java
- radare2
  開源工具, CLI

動態分析

- Windbg preview
  強大
- x64dbg (建議課堂中使用)
  GUI, 開源工具, 擴充功能
- gdb (建議課堂中使用)
  CLI, 強大
- edb
  GUI, 不習慣 CLI 可以先試試

x86(IA-32) register

- Accumulator register
- Base register
- Counter register
- Data register
- Source Index
- Destination Index
- Stack Pointer
- Stack Base Pointer
- Instruction Pointer



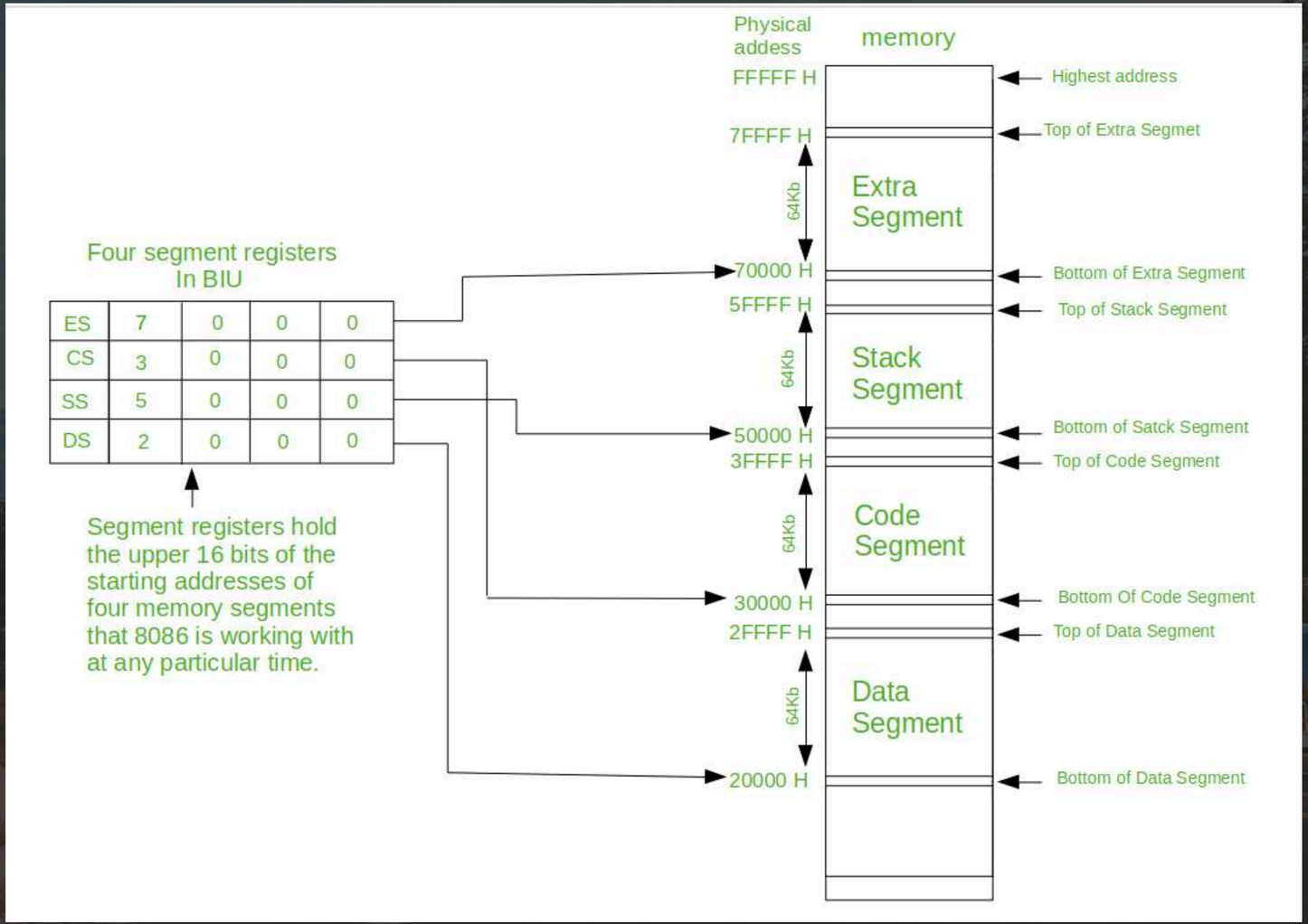
x86-64(AMD-64) register

- additional r8~r15

	64bits	32bits	16bits	8bits	
RAX		EAX	AH	AL	AX
RBX		EBX	BH	BL	BX
RCX		ECX	CH	CL	CX
RDX		EDX	DH	DL	DX
RSI		ESI		sil	SI
RDI		EDI		dil	DI
RSP		ESP		spl	SP
RBP		EBP		bpl	BP
R8		R8d		R8b	R8W
RIP					

Segment Registers

- CS, DS, SS, ES, FS, GS
- cs:0x1000 (= 0x31000)
- OS use flat memory model nowadays
- Segment Register no longer represent the base of a segment, but the index in Descriptor Table



Instruction

```
//mov
mov   rax, rbx              //rax = rbx
mov   bl,  byte ptr [rbx]   //char bl = *(char *)rbx
// Load Effective Address
lea     rax, [0x12345678]   // rax = 0x12345678
lea     rax, [rax*2+16]     // rax = 0x12345678*2+16
```

Instruction

```
add rax, rbx     // rax = rax + rbx
sub rax, rbx     // rax = rax - rbx
inc rax          // rax++
dec rax          // rax--
and rax, rbx     // rax = rax & rbx
or  rax, rbx     // rax = rax | rbx
shl rax, 2       // rax = rax << 2
shr rax, 2       // rax = rax >> 2
```

```
xor rax, rbx     // rax = rax xor rbx
not rax          // rax = !rax
neg rax
// rax = -rax (2's complement)
test rax, rbx
// rax & rbx, won't write back
cmp rax, rbx
// rax - rbx, won't write back
```

FLAGS register (EFLAGS, RFLAGS)

- Zero Flag
  - set if result is 0
  - e.g. 100-100=0
- Carry Flag
  - set if carry of borrow a bit beyond the size of register
  - e.g. 0 - 1 = 4294967295
  - e.g. 4294967295 + 1 = 0

FLAGS register (EFLAGS, RFLAGS)

- Overflow Flag
  - set if singed result overflow
  - e.g. 2147483647 + 1 = - 2147483648
- Sign Flag
  - set if operation result is negative (sign bit is 1)
  - e.g. 0 - 1 = -1

FLAGS register (EFLAGS, RFLAGS)

```
cmp eax, ebx
```

無號整數

ZF	CF	代表結果
FALSE	TRUE	eax < ebx
FALSE	FALSE	eax > ebx
TRUE	FALSE	eax = ebx

有號整數

EFLAG	代表結果
sign flag ≠ overflow flag	eax < ebx
sign flag = overflow flag	eax > ebx
ZF = True	eax = ebx

compare 時並不管有號還無號、flag 全部會設好，Jcc 指令決定看什麼

Jcc

無號整數

JA	Jump if above
JNBE	Jump if not below or not equal (Jump if above)
JAE	Jump if above or equal
JNB	Jump if not below (=JAE)
JB	Jump if below
JNAE	Jump if not above or not equal(=JB)
JBE	Jump if below or equal
JNA	Jump if not above(=JBE)

有號整數

JG	Jump if greater
JNLE	Jump if not less or not equal(=JG)
JGE	Jump if greater or equal
JNL	Jump if not less (=JGE)
JL	Jump if less
JNGE	Jump if not greater or not equal(=JL)
JLE	Jump if less or equal
JNG	Jump if not greater (=JLE)

普通

JE	Jump if equal
JNE	Jump if not equal
JMP	不管，跳

其他

JC	Jump if carry flag set
JS	Jump if sign flag set
各種...	

Instruction

```
mov eax, 0x20 //eax = 0x20
mov ecx, 3
mul ecx
//edx:eax = eax * ecx
div ecx
// edx:eax / 3
// quotient in eax, remainder in edx
```

```
nop //xchg eax, eax
```

Instruction

```
global _start
section .text

_start:
//memcpy(target,source,6)
    mov rcx, 6
    mov rsi, source
    mov rdi, target
    cld
    rep movsb

section .data

source: db    'hello',0
target: times 6 db 1
```

Instruction

```
mov ebx , 110
mov eax , 0
mov eax , ebx
add eax , ebx
sub eax , ebx
inc eax
dec eax
```

Intel syntax

ebx = 100 的概念

```
movl $110, %ebx
movl $0, %eax
mov %ebx , %eax
add %eax , %ebx
sub %ebx , %eax
inc %eax
dec %eax
```

AT&T syntax

100 -> ebx 的概念

ABEX'S CRACKME #1

HELLO

Program

- EntryPoint
- StubCode
- Main 非 Main

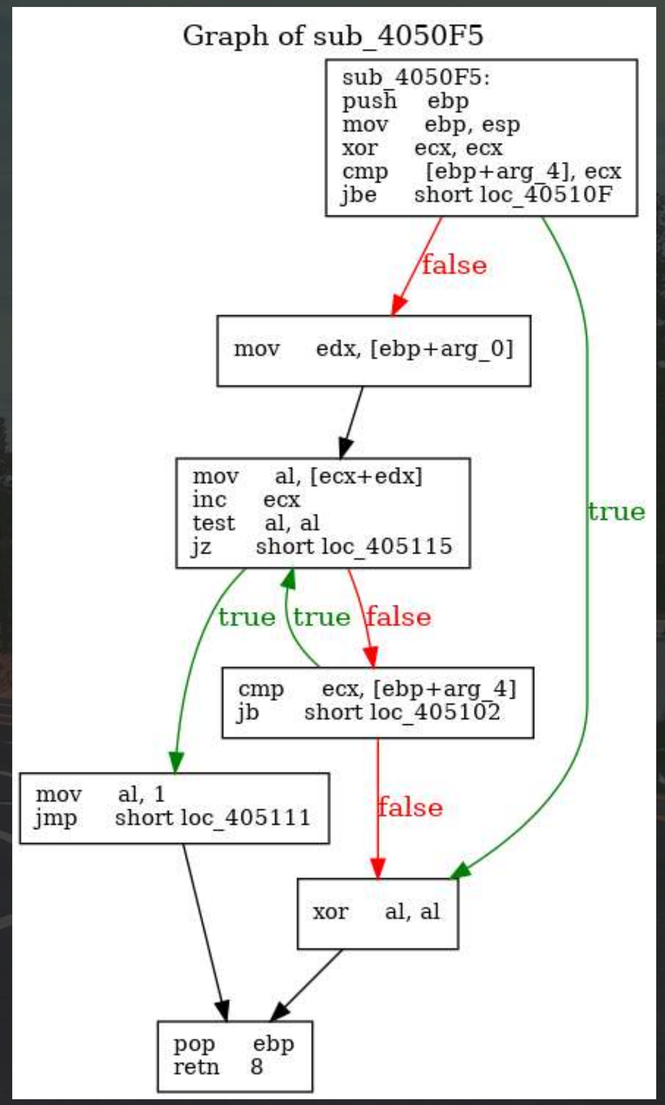
找到關鍵 code 的方法

- 直接滑起來
- 尋找參數 (string or magic number)
- Break on API
- Break in API (run time packer, protector)

BABY STEP

Control-flow graph

- 一個節點代表一個 BB (Basic Block)
- BB 只有一個入口點以及出口點
- BB 就是 instruction 的 sequence
- BB 中一個 instruction 的執行暗示了同 BB 中的其他 instruction 已經或即將被執行（不考慮 exception）



Control-flow graph

- 這是一個平淡無奇的 BB (Basic Block)

```
mov     edi, offset s   ;"Do the right thing"
call    _puts
jmp     short loc_4008AB
```

Control-flow graph

- 這是一個不知道要往哪的 BB

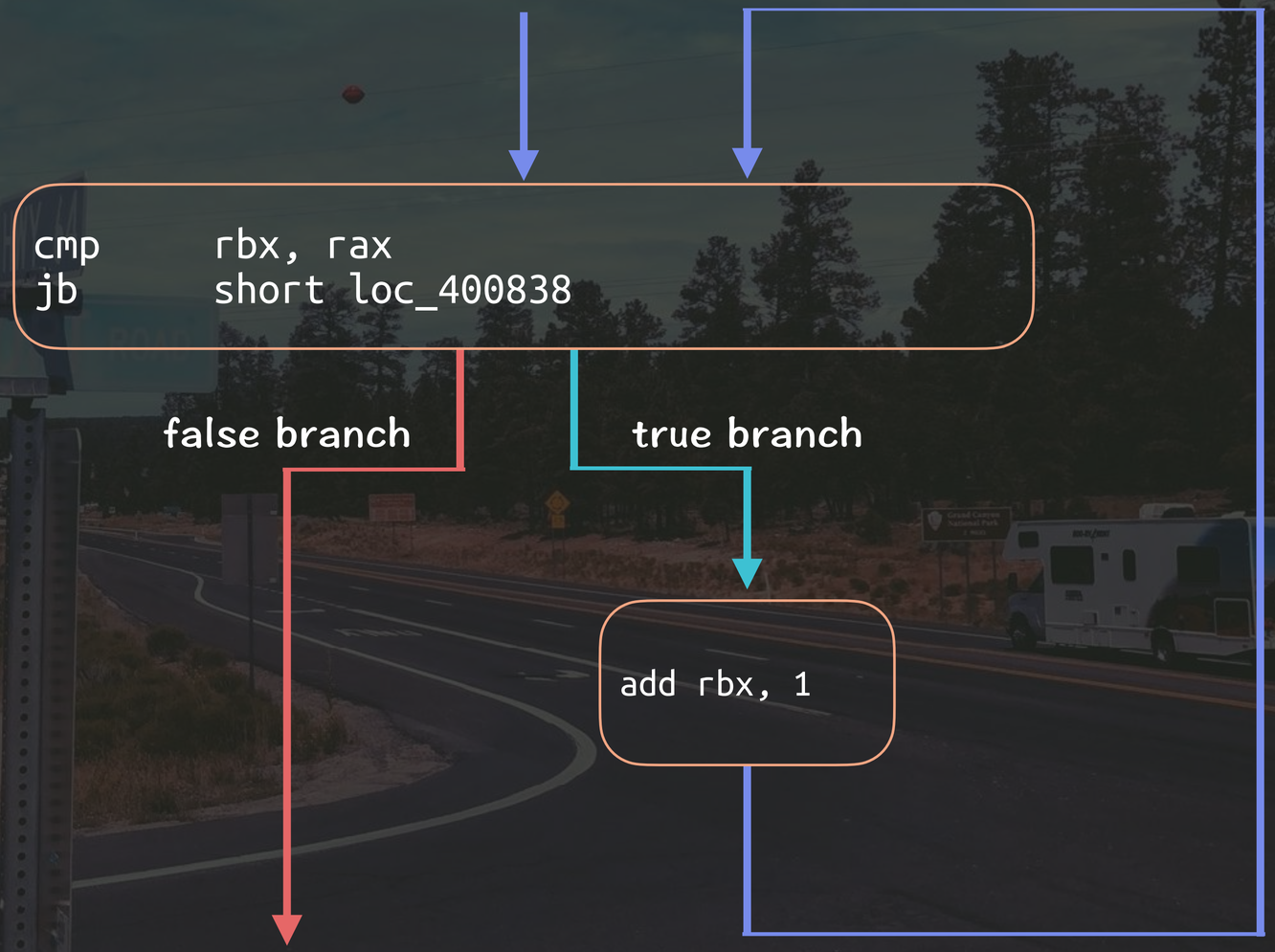
```
mov     eax, [rbp+var_7C]
movsxd  rbx, eax
lea     rax, [rbp+ptr]
mov     rdi, rax
call    _strlen
cmp     rbx, rax
jb      short loc_400838
```

false branch

true branch

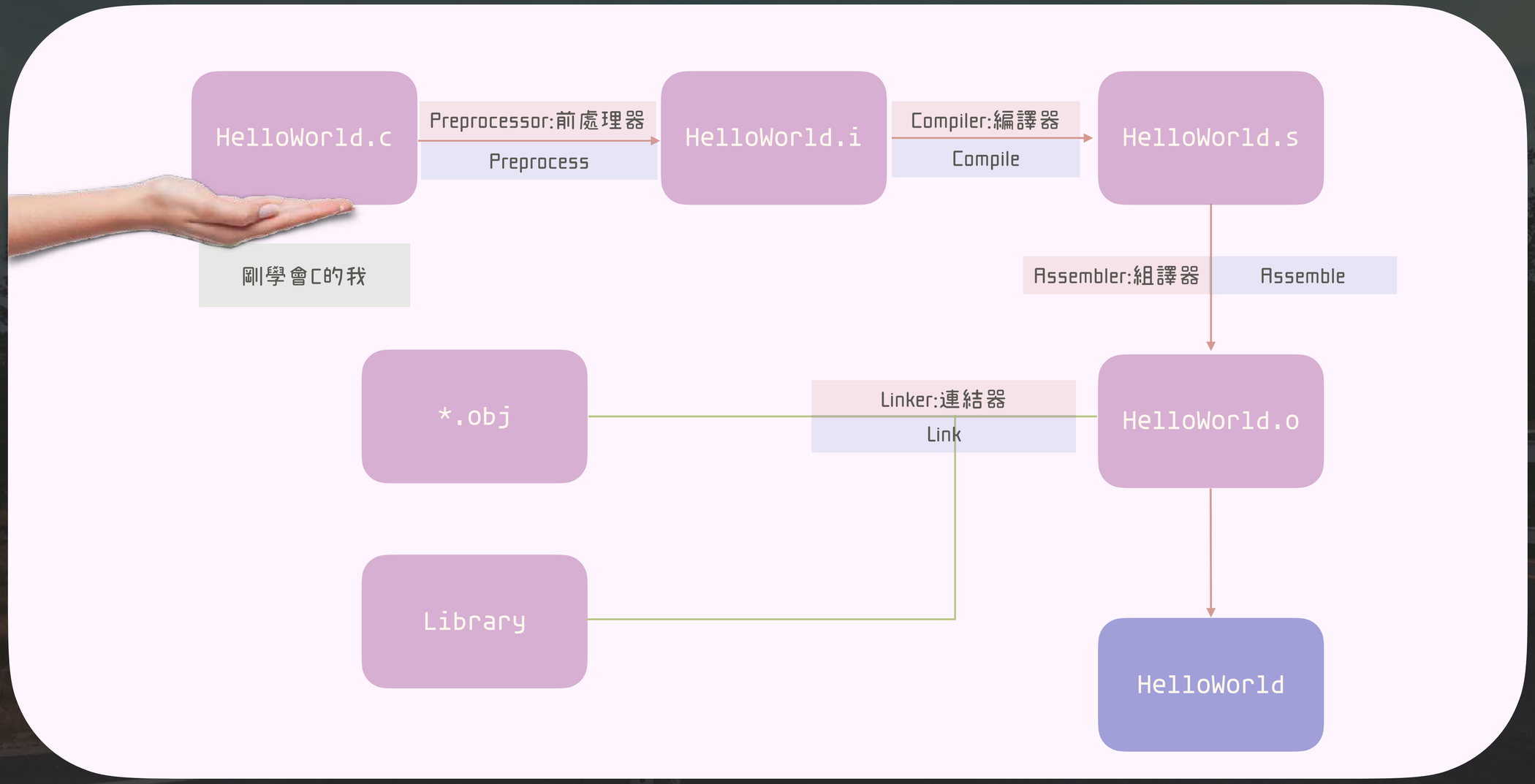
Control-flow graph

- 這是一個走回頭路的 BB



C

compile walk throught



if

```c
#include <stdio.h>

int main(int argc, char** argv, char** evnp){
    if (argc == 0){
        printf("no argc");
    }
    return 0;
}
```

```
64a:   55                      push   rbp
64b:   48 89 e5                mov    rbp,rsp
64e:   48 83 ec 20             sub    rsp,0x20
652:   89 7d fc                mov    DWORD PTR [rbp-0x4],edi
655:   48 89 75 f0             mov    QWORD PTR [rbp-0x10],rsi
659:   48 89 55 e8             mov    QWORD PTR [rbp-0x18],rdx
65d:   83 7d fc 00             cmp    DWORD PTR [rbp-0x4],0x0
661:   75 11                   jne    674 <main+0x2a>
663:   48 8d 3d 9a 00 00 00    lea    rdi,[rip+0x9a]
66a:   b8 00 00 00 00          mov    eax,0x0
66f:   e8 ac fe ff ff          call   520 <printf@plt>
674:   b8 00 00 00 00          mov    eax,0x0
679:   c9                      leave
67a:   c3                      ret
```

if-else

```c
#include <stdio.h>

int main(int argc, char** argv, char** evnp){
    if (argc == 0){
        printf("no argc");
    }else{
        printf("there are %d args", argc);
    }
    return 0;
}
```

```
64a:   55                    push   rbp
64b:   48 89 e5              mov    rbp,rsp
64e:   48 83 ec 20           sub    rsp,0x20
652:   89 7d fc              mov    DWORD PTR [rbp-0x4],edi
655:   48 89 75 f0           mov    QWORD PTR [rbp-0x10],rsi
659:   48 89 55 e8           mov    QWORD PTR [rbp-0x18],rdx
65d:   83 7d fc 00           cmp    DWORD PTR [rbp-0x4],0x0
661:   75 13                 jne    676 <main+0x2c>
...
66f:   e8 ac fe ff ff        call   520 <printf@plt>
674:   eb 16                 jmp    68c <main+0x42>
...
687:   e8 94 fe ff ff        call   520 <printf@plt>
68c:   b8 00 00 00 00        mov    eax,0x0
691:   c9                    leave
692:   c3                    ret
```

PRACTICE