IDA 不是只有 F5 可以按

進階資安攻防競技

TwinkleStar03

whoami

- TwinkleStar03
  - Member of UNDEFINED
  - Reverse Engineering, Pwning @ ${CyStick}
  - StarHack Academy 講師
- Blog: blog.star03.me
- Email: star@undefined.zip

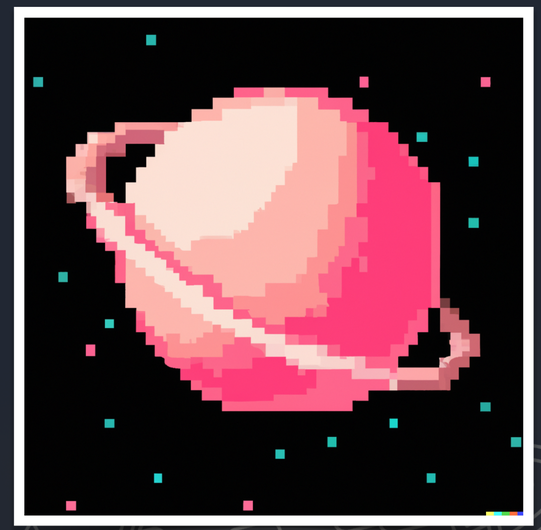
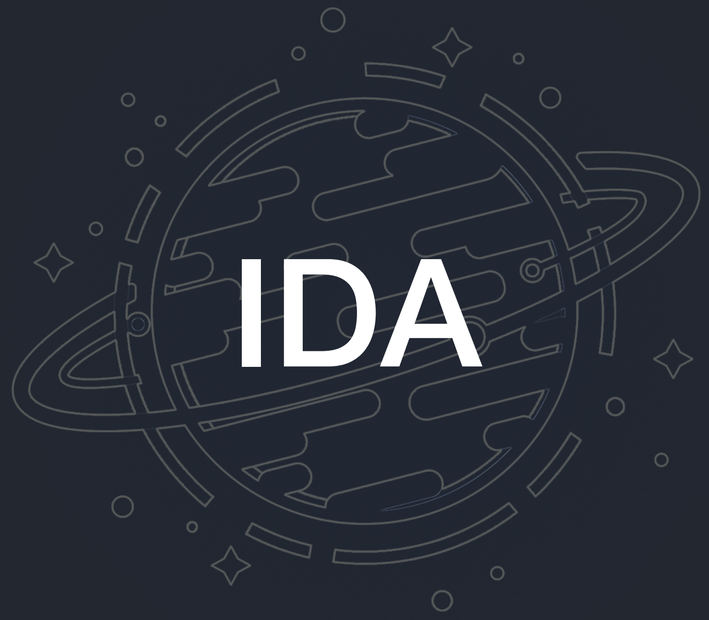


Table of Content

- IDAFree 介紹
- Binary 分析起手式
- IDA Workspace
- 標記各種東西
- Lab: Fix idb
- Inspect ELF Binary
- Local Debugger
- Patching Binary
- Lab: patch & debug
- Structure Recovery

Lab URL: https://ais3-practice.zoolab.org/



IDA

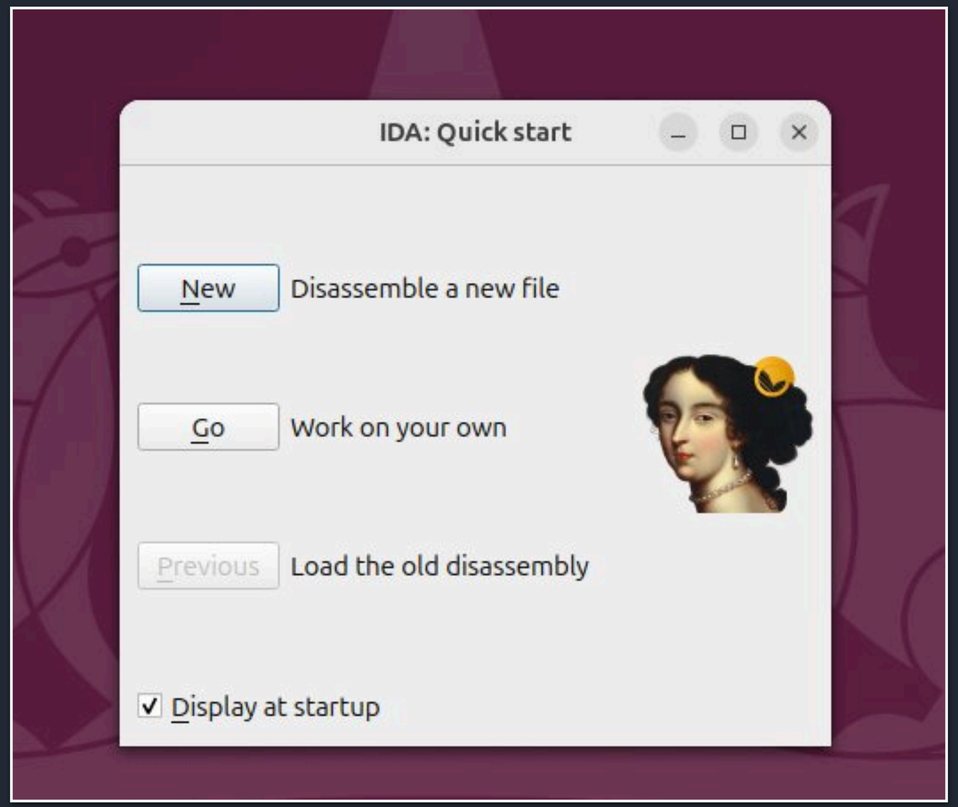
IDA

- 家喻戶曉的逆向工程工具
- IDAFree / IDAPro / IDATeam
- 可同時勝任動態與靜態分析
- 本堂課使用 IDAFree

新手怎麼使用 IDA

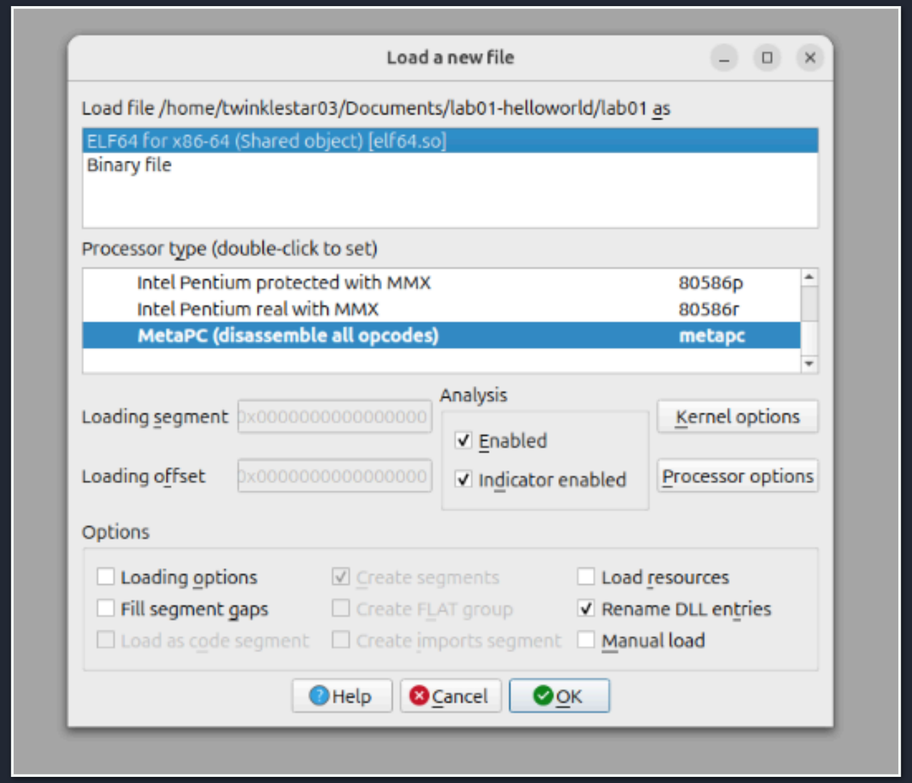
- 打開一個 Binary
- 不在乎一路彈出來的視窗都寫了什麼，一路點 Ok / Next / Done
- 在左邊的 Function List 找到 main 或是 start 點兩下
- 不管三七二十一，按下 F5

新手怎麼使用 IDA



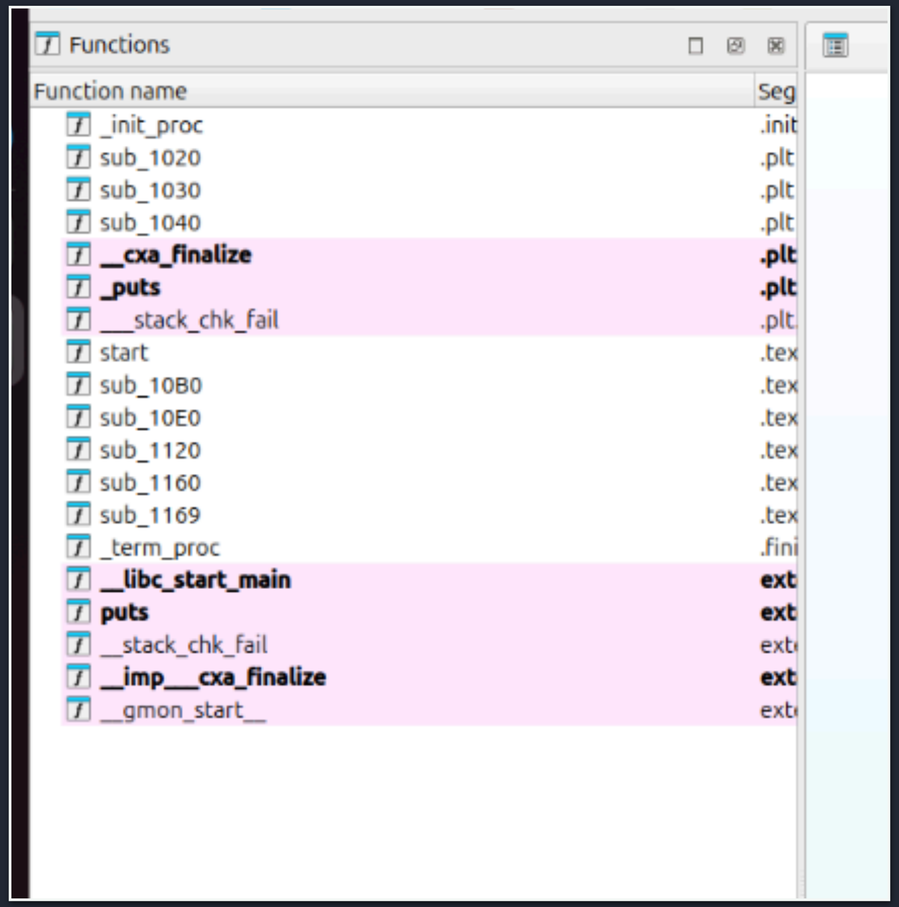
打開 IDA

新手怎麼使用 IDA



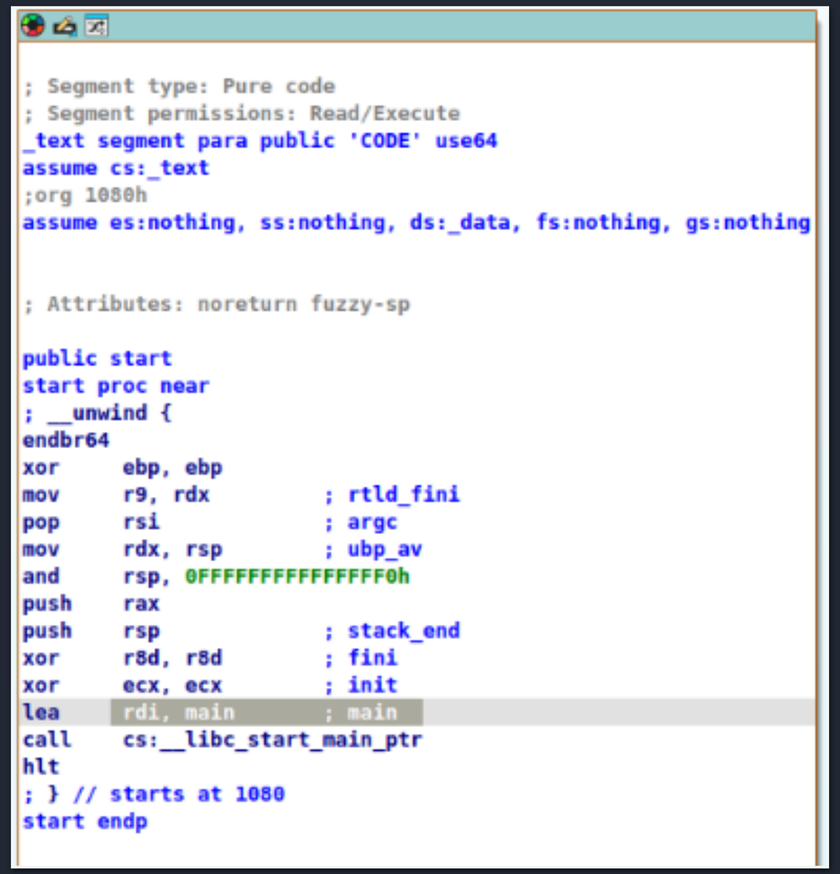
載入 Binary

新手怎麼使用 IDA



在 Function List 裡面找 start 或 main

新手怎麼使用 IDA



```
; Segment type: Pure code
; Segment permissions: Read/Execute
_text segment para public 'CODE' use64
assume cs:_text
;org 1080h
assume es:nothing, ss:nothing, ds:_data, fs:nothing, gs:nothing


; Attributes: noreturn fuzzy-sp

public start
start proc near
; __unwind {
endbr64
xor     ebp, ebp
mov     r9, rdx         ; rtld_fini
pop     rsi             ; argc
mov     rdx, rsp        ; ubp_av
and     rsp, 0FFFFFFFFFFFFFFF0h
push    rax
push    rsp             ; stack_end
xor     r8d, r8d        ; fini
xor     ecx, ecx        ; init
lea     rdi, main       ; main
call    cs:__libc_start_main_ptr
hlt
; } // starts at 1080
start endp
```

看到 main 開心的點兩下

新手怎麼使用 IDA

```
.text:0000000000001210 main:                                   ; DATA XREF: start+18↑o
.text:0000000000001210 ; __unwind {
.text:0000000000001210                 endbr64
.text:0000000000001214                 push    rbp
.text:0000000000001215                 mov     rbp, rsp
.text:0000000000001218                 sub     rsp, 10h
.text:000000000000121C                 mov     [rbp-4], edi
.text:000000000000121F                 mov     [rbp-10h], rsi
.text:0000000000001223                 lea     rax, aHelloworld ; "HelloWorld!"
.text:000000000000122A                 mov     rdi, rax
.text:000000000000122D                 call    _puts
.text:0000000000001232
.text:0000000000001232 loc_1232:                               ; CODE XREF: .text:loc_1232↑j
.text:0000000000001232                 jmp     short near ptr loc_1232+1
.text:0000000000001232 ; ---------------------------------------------------------------------------
.text:0000000000001234                 dd 58D48C0h
.text:0000000000001238                 dq 0E8C7894800000E54h, 0B8FFFFFE1Ch
.text:0000000000001248                 db 0, 0C9h, 0C3h
.text:0000000000001248 ; } // starts at 1210
.text:0000000000001248 _text           ends
.text:0000000000001248
LOAD:000000000000124B ; ===========================================================================
```

新手怎麼使用 IDA

```
.text:0000000000001210 main:                                   ; DATA XREF: start+18↑o
.text:0000000000001210 ; __unwind {
.text:0000000000001210                 endbr64
.text:0000000000001214                 push    rbp
.text:0000000000001215                 mov     rbp, rsp
.text:0000000000001218                 sub     rsp, 10h
.text:000000000000121C                 mov     [rbp-4], edi
.text:000000000000121F                 mov     [rbp-10h], rsi
.text:0000000000001223                 lea     rax, aHelloworld ; "HelloWorld
.text:000000000000122A                 mov     rdi, rax
.text:000000000000122D                 call    _puts
.text:0000000000001232
.text:0000000000001232 loc_1232:                               ; CODE XREF: .te
.text:0000000000001232                 jmp     short near ptr loc_1232+1
.text:0000000000001232 ; ---------------------------------------------------------------------------
.text:0000000000001234                 dd 58D48C0h
.text:0000000000001238                 dq 0E8C7894800000E54h, 0B8FFFFFE1Ch
.text:0000000000001248                 db 0, 0C9h, 0C3h
.text:0000000000001248 ; } // starts at 1210
.text:0000000000001248 _text           ends
.text:0000000000001248
LOAD:000000000000124B ; ===========================================================================
```

壞了，IDA 成垃圾了，沒 F5 可按了

新手怎麼使用 IDA Cont.

- 壞掉了怎麼辦？
  - 對 IDA 的功能沒概念不會修
  - 過往只看 decompile 不看 disassembly 漏掉關鍵資訊
- 沒有善用強大的工具！
- 這堂課教會大家使用 IDA，讓逆向工程的旅途變得愉快些

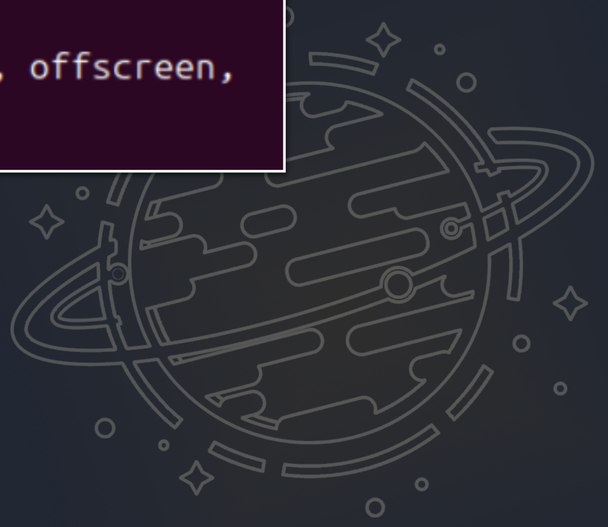
Binary 分析起手式

Launching IDAFree

- 在 Ubuntu 中打開 IDAFree 可能會長這樣...

Launching IDAFree - Fix

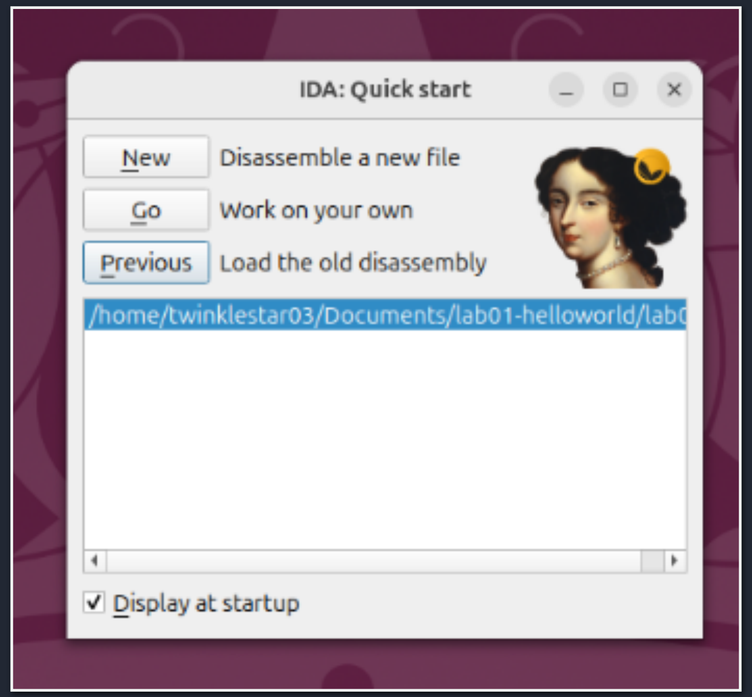
```
twinklestar03@twinklestar03-Standard-PC-i440FX-PIIX-1996:~/idafree-8.4$ ./ida64
Warning: Ignoring XDG_SESSION_TYPE=wayland on Gnome. Use QT_QPA_PLATFORM=wayland
 to run on Wayland anyway.
qt.qpa.plugin: Could not load the Qt platform plugin "xcb" in "" even though it
was found.
This application failed to start because no Qt platform plugin could be initiali
zed. Reinstalling the application may fix this problem.

Available platform plugins are: eglfs, linuxfb, minimal, minimalegl, offscreen,
vnc, xcb.
```



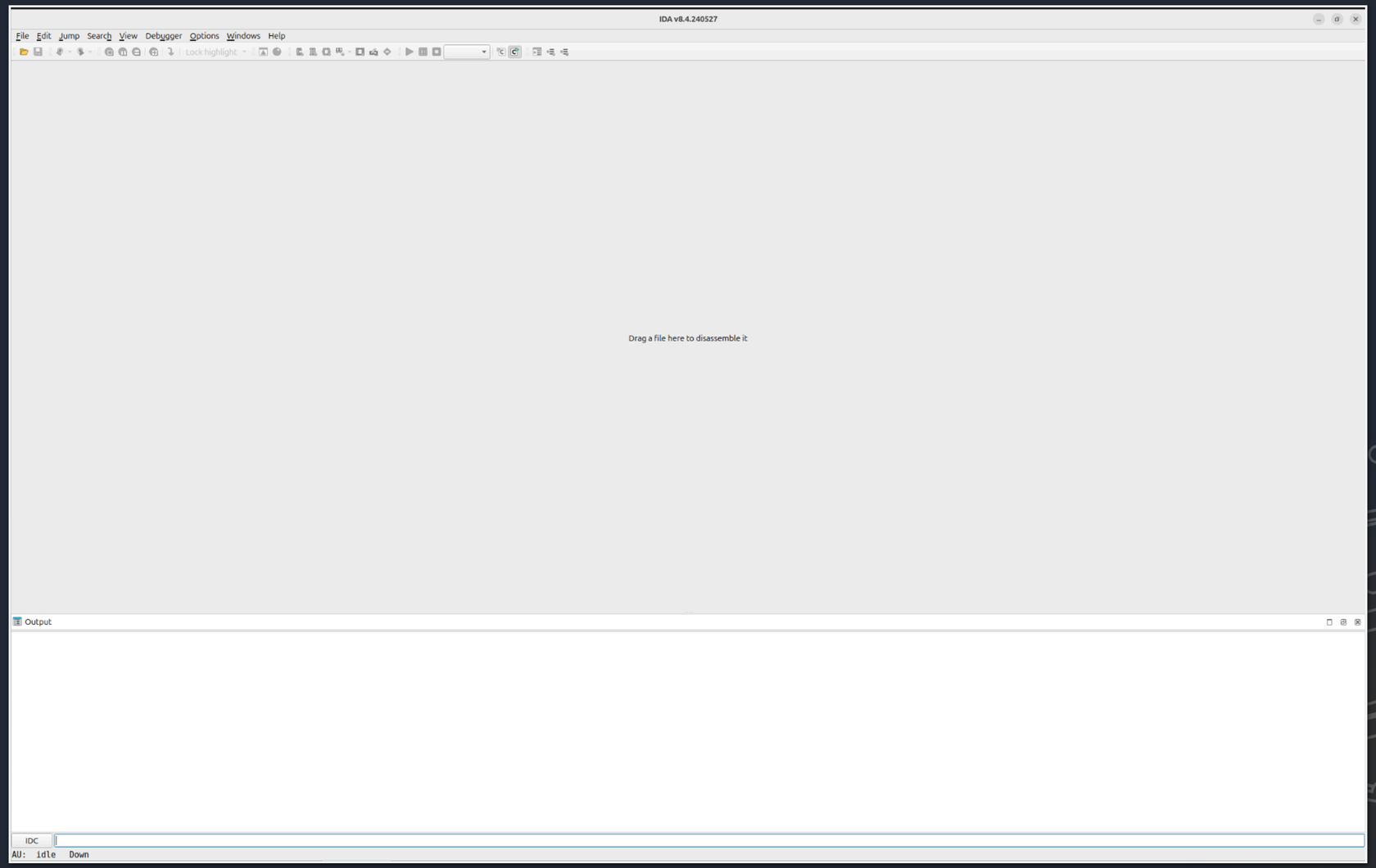
Launching IDAFree - Fix

- 安裝缺少的 Qt 套件
  - sudo apt install libxcb-xinerama0

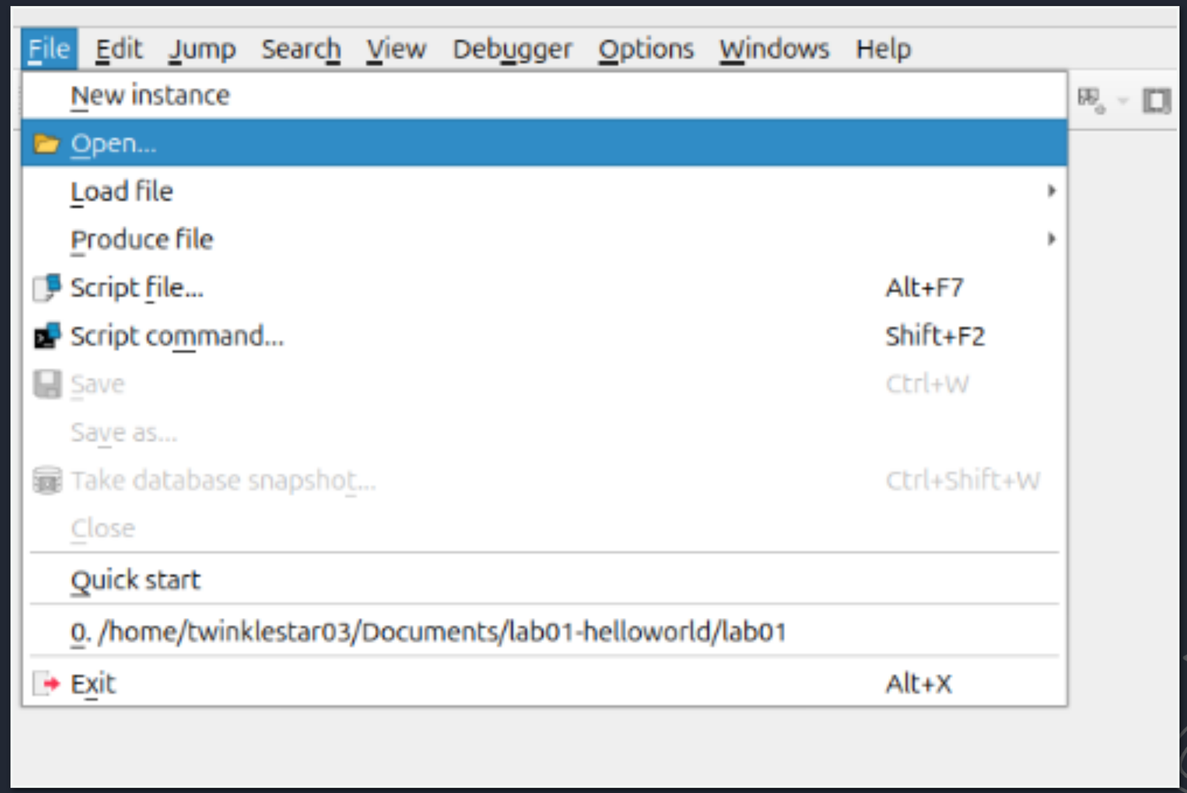
Launching IDAFree - Start Page



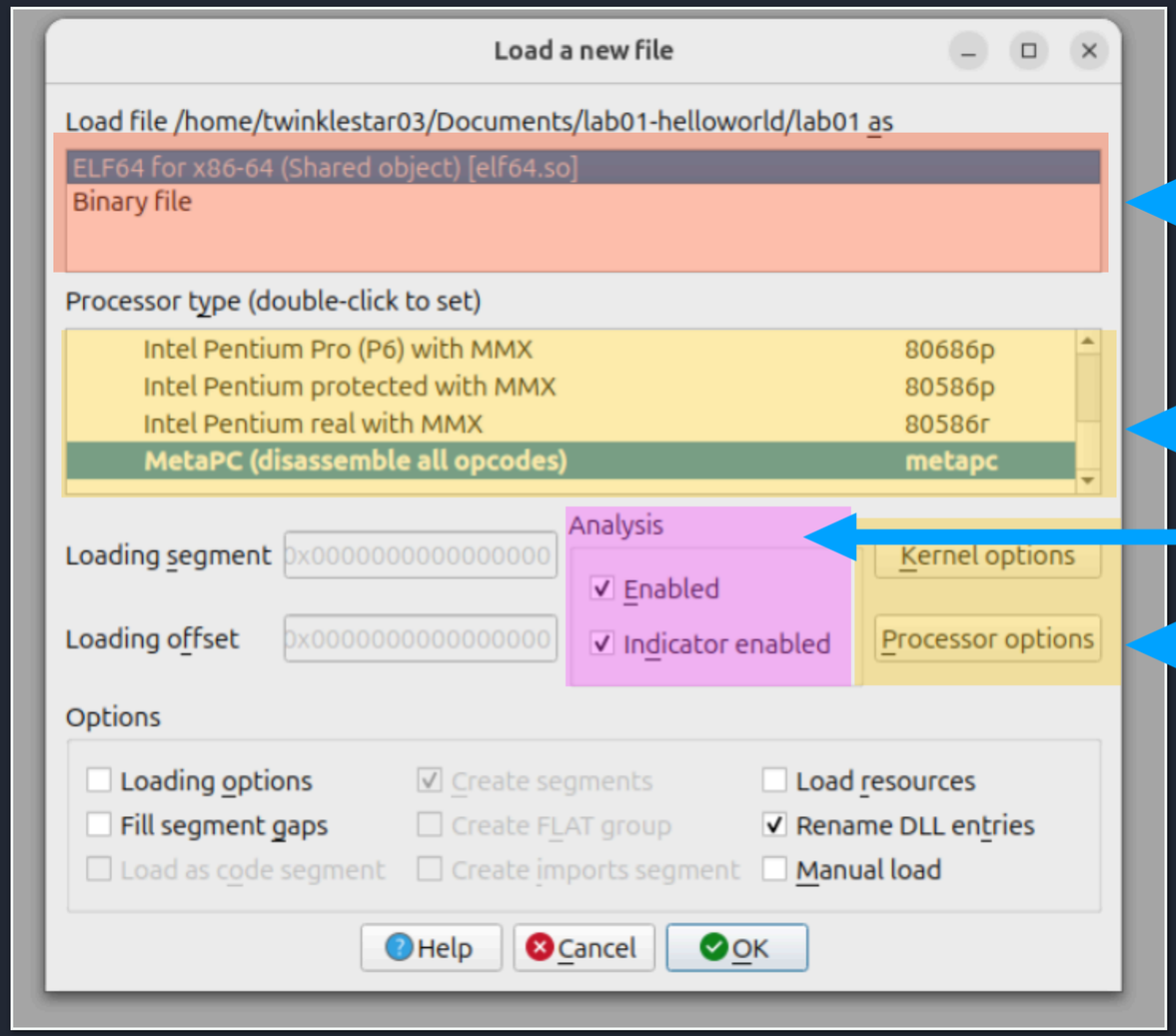
Launching IDAFree - Main Page



Launching IDAFree - Load



Launching IDAFree - Load

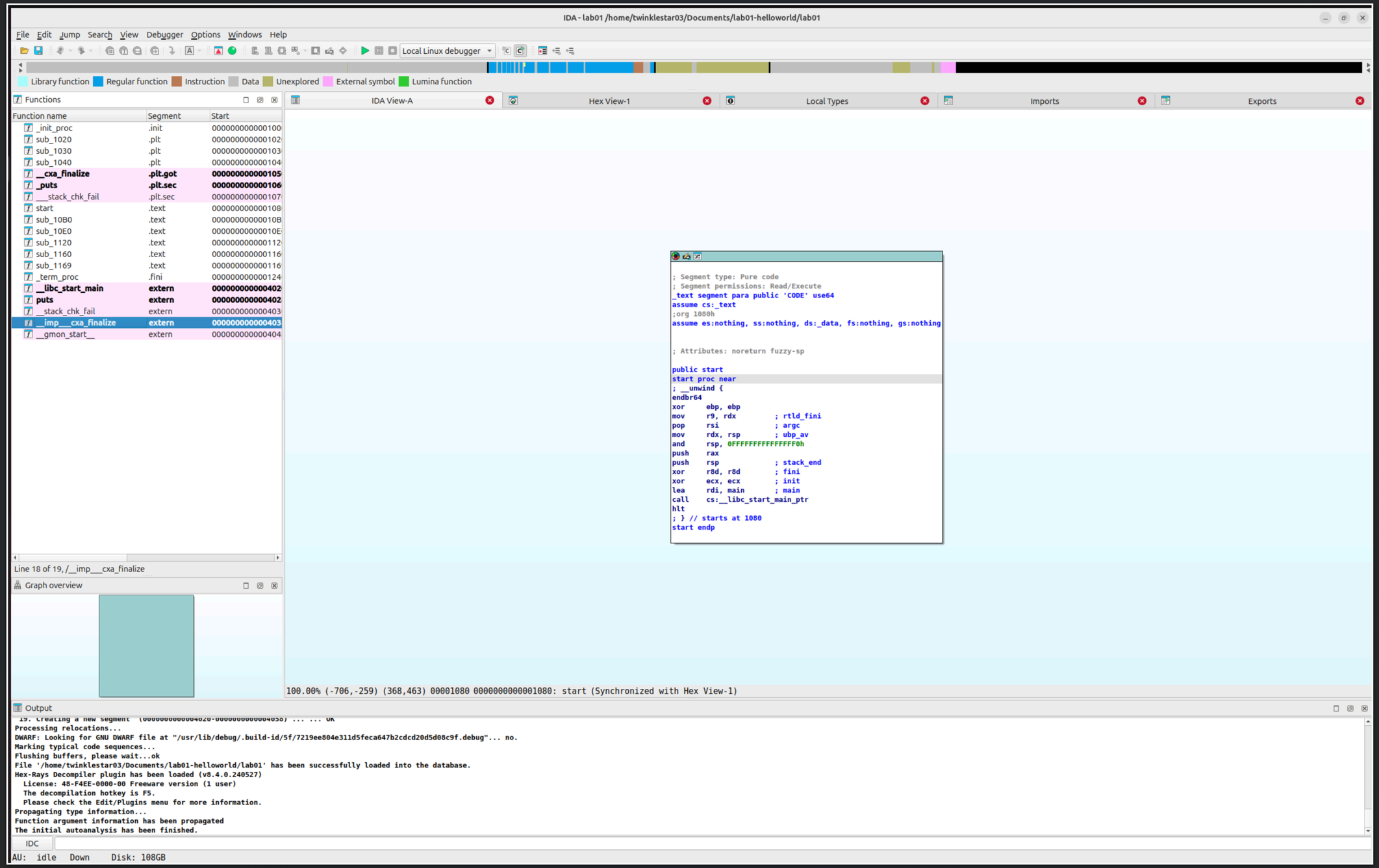


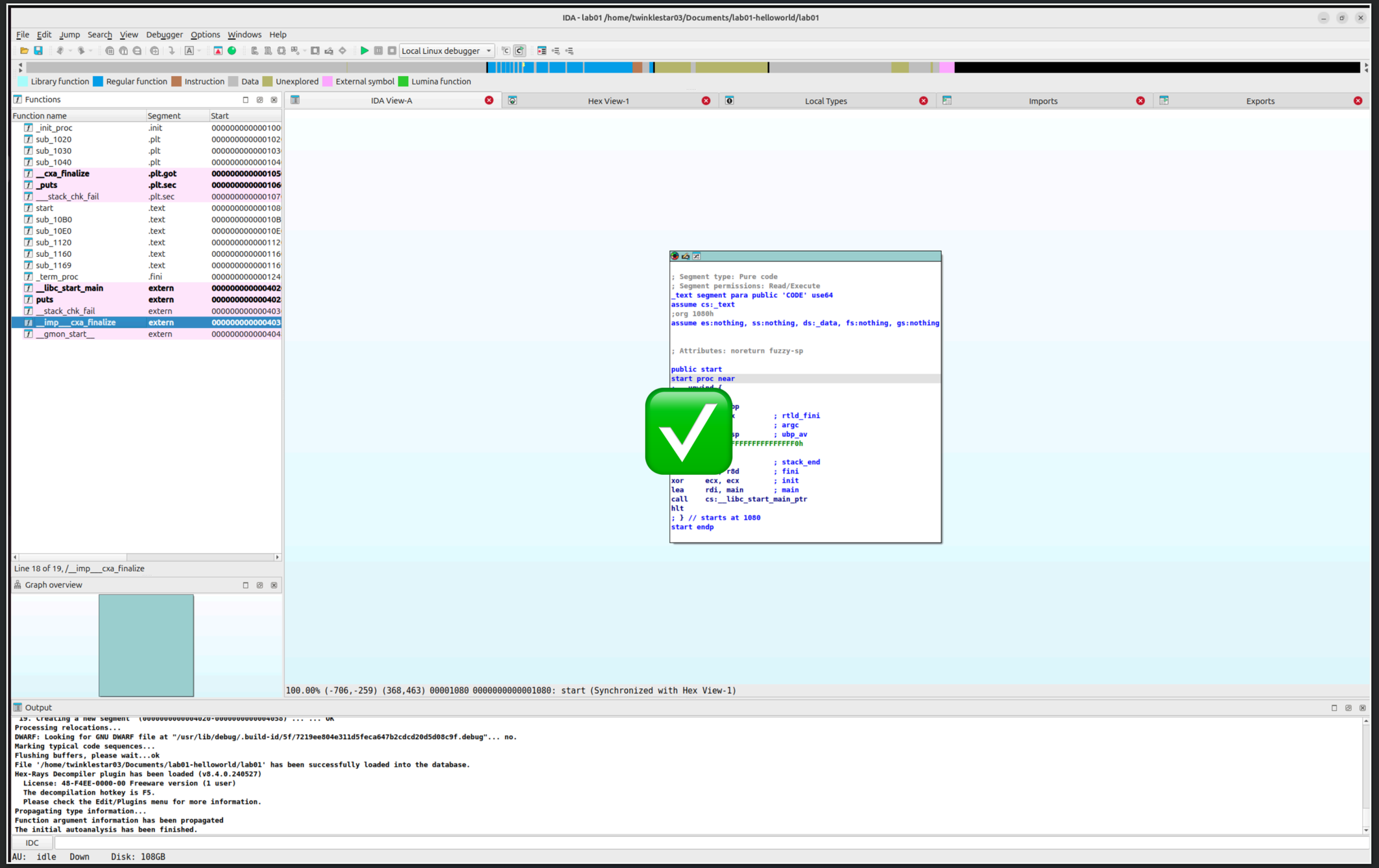
選擇解析時的格式 ELF/PE/MachO

選擇指令集

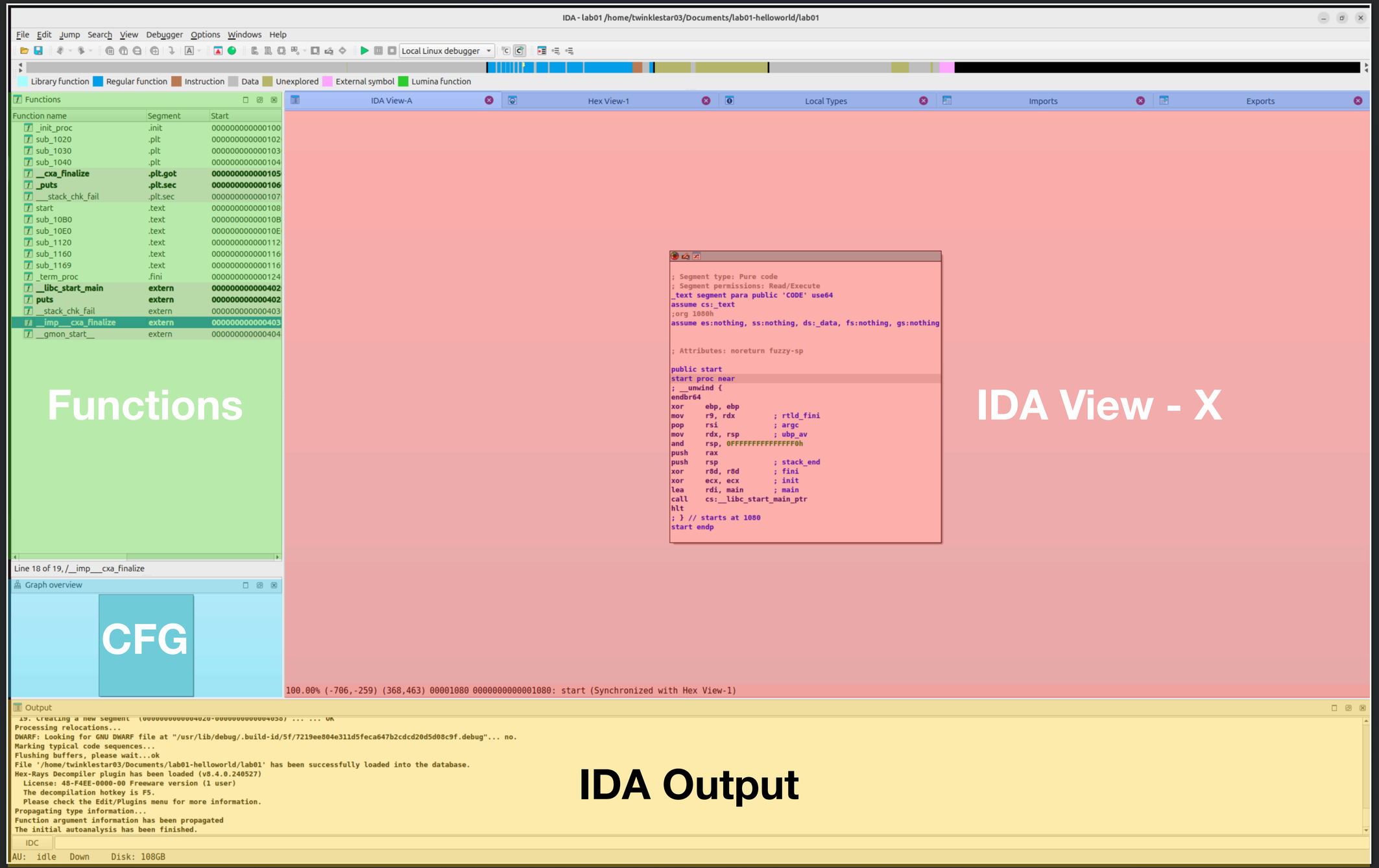
自動分析

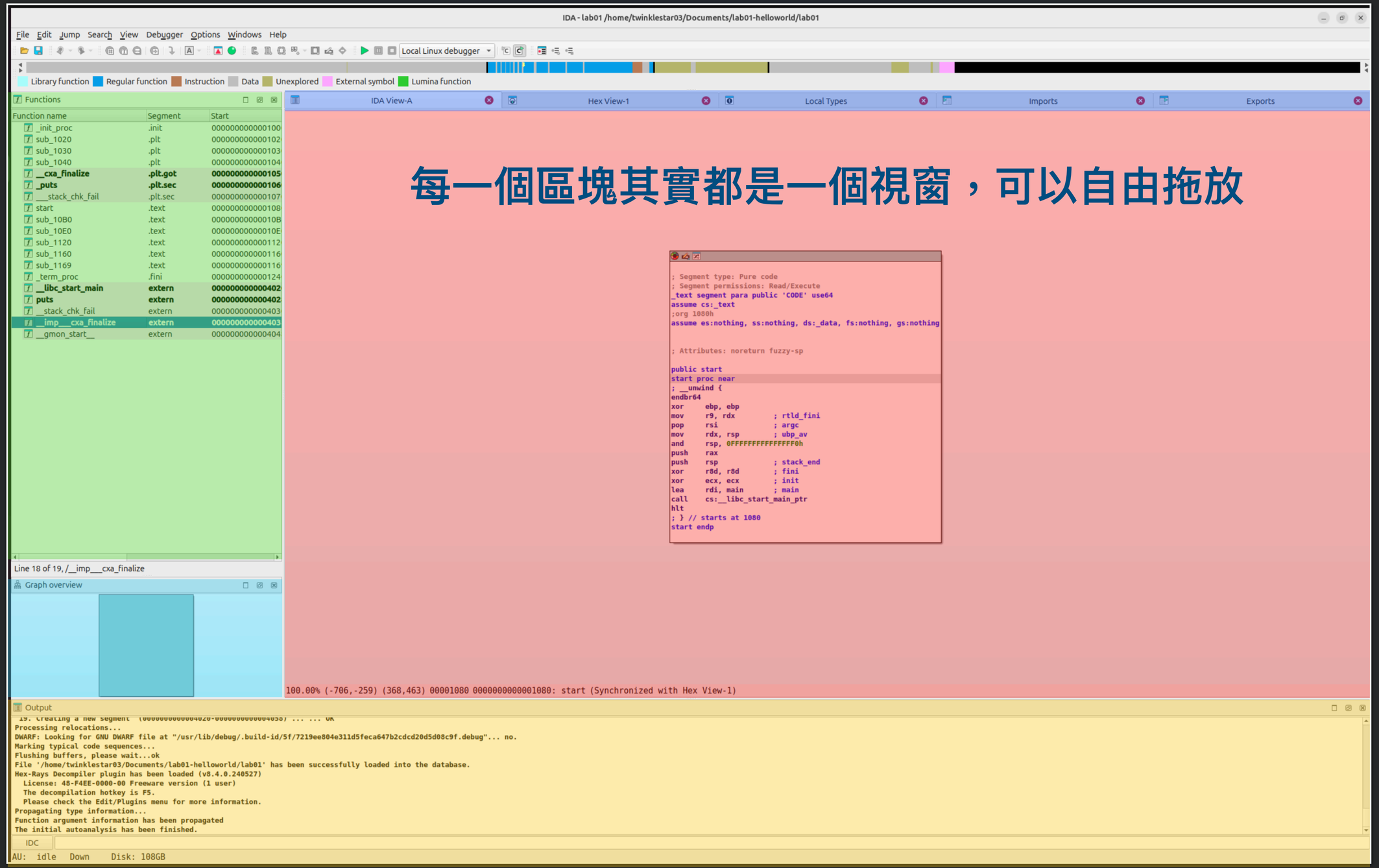
細節設定

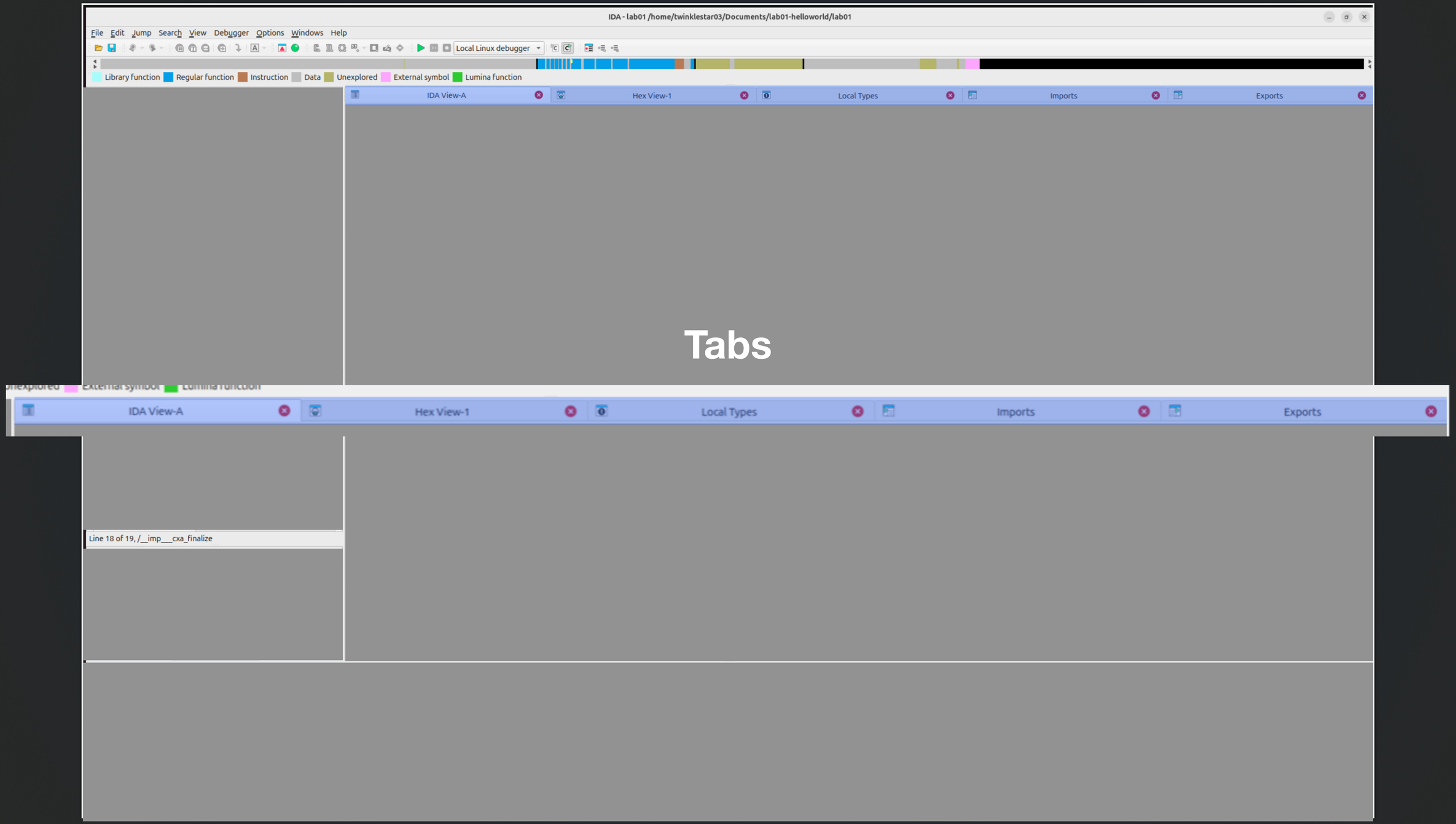

IDA - lab01 /home/twinklestar03/Documents/lab01-helloworld/lab01
File Edit Jump Search View Debugger Options Windows Help
Local Linux debugger
Library function Regular function Instruction Data Unexplored External symbol Lumina function
Functions
Function name Segment Start
_init_proc .init
sub_1020 .plt
sub_1030 .plt
sub_1040 .plt
__cxa_finalize .plt.got
_puts .plt.sec
__stack_chk_fail .plt.sec
start .text
sub_10B0 .text
sub_10E0 .text
sub_1120 .text
sub_1160 .text
sub_1169 .text
_term_proc .fini
__libc_start_main extern
puts extern
__stack_chk_fail extern
__imp___cxa_finalize extern
__gmon_start__ extern
Line 18 of 19, /__imp___cxa_finalize
Graph overview
IDA View-A
Hex View-1
Local Types
Imports
Exports
; Segment type: Pure code
; Segment permissions: Read/Execute
_text segment para public 'CODE' use64
assume cs:_text
;org 1080h
assume es:nothing, ss:nothing, ds:_data, fs:nothing, gs:nothing
; Attributes: noreturn fuzzy-sp
public start
start proc near
; __unwind {
endbr64
xor ebp, ebp
mov r9, rdx ; rtld_fini
pop rsi ; argc
mov rdx, rsp ; ubp_av
and rsp, 0FFFFFFFFFFFFFFF0h
push rax
push rsp ; stack_end
xor r8d, r8d ; fini
xor ecx, ecx ; init
lea rdi, main ; main
call cs:__libc_start_main_ptr
hlt
; } // starts at 1080
start endp
100.00% (-706,-259) (368,463) 00001080 0000000000001080: start (Synchronized with Hex View-1)
Output
Processing relocations...
DWARF: Looking for GNU DWARF file at "/usr/lib/debug/.build-id/5f/7219ee804e311d5feca647b2cdcd20d5d08c9f.debug"... no.
Marking typical code sequences...
Flushing buffers, please wait...ok
File '/home/twinklestar03/Documents/lab01-helloworld/lab01' has been successfully loaded into the database.
Hex-Rays Decompiler plugin has been loaded (v8.4.0.240527)
License: 48-F4EE-0000-00 Freeware version (1 user)
The decompilation hotkey is F5.
Please check the Edit/Plugins menu for more information.
Propagating type information...
Function argument information has been propagated
The initial autoanalysis has been finished.
IDC
AU: idle Down Disk: 108GB


IDA - lab01 /home/twinklestar03/Documents/lab01-helloworld/lab01
File Edit Jump Search View Debugger Options Windows Help
Local Linux debugger
Library function Regular function Instruction Data Unexplored External symbol Lumina function
Functions
Function name Segment Start
_init_proc .init 0000000000001000
sub_1020 .plt 0000000000001020
sub_1030 .plt 0000000000001030
sub_1040 .plt 0000000000001040
__cxa_finalize .plt.got 0000000000001050
_puts .plt.sec 0000000000001060
__stack_chk_fail .plt.sec 0000000000001070
start .text 0000000000001080
sub_10B0 .text 00000000000010B0
sub_10E0 .text 00000000000010E0
sub_1120 .text 0000000000001120
sub_1160 .text 0000000000001160
sub_1169 .text 0000000000001169
_term_proc .fini 0000000000001240
__libc_start_main extern 0000000000004020
puts extern 0000000000004028
__stack_chk_fail extern 0000000000004030
__imp___cxa_finalize extern 0000000000004038
__gmon_start__ extern 0000000000004040
Line 18 of 19, /__imp___cxa_finalize
Graph overview
IDA View-A
Hex View-1
Local Types
Imports
Exports
; Segment type: Pure code
; Segment permissions: Read/Execute
_text segment para public 'CODE' use64
assume cs:_text
;org 1080h
assume es:nothing, ss:nothing, ds:_data, fs:nothing, gs:nothing
; Attributes: noreturn fuzzy-sp
public start
start proc near
; rtld_fini
; argc
; ubp_av
; stack_end
r8d ; fini
xor ecx, ecx ; init
lea rdi, main ; main
call cs:__libc_start_main_ptr
hlt
; } // starts at 1080
start endp
100.00% (-706,-259) (368,463) 00001080 0000000000001080: start (Synchronized with Hex View-1)
Output
Processing relocations...
DWARF: Looking for GNU DWARF file at "/usr/lib/debug/.build-id/5f/7219ee804e311d5feca647b2cdcd20d5d08c9f.debug"... no.
Marking typical code sequences...
Flushing buffers, please wait...ok
File '/home/twinklestar03/Documents/lab01-helloworld/lab01' has been successfully loaded into the database.
Hex-Rays Decompiler plugin has been loaded (v8.4.0.240527)
License: 48-F4EE-0000-00 Freeware version (1 user)
The decompilation hotkey is F5.
Please check the Edit/Plugins menu for more information.
Propagating type information...
Function argument information has been propagated
The initial autoanalysis has been finished.
IDC
AU: idle Down Disk: 108GB

認識 Workspace

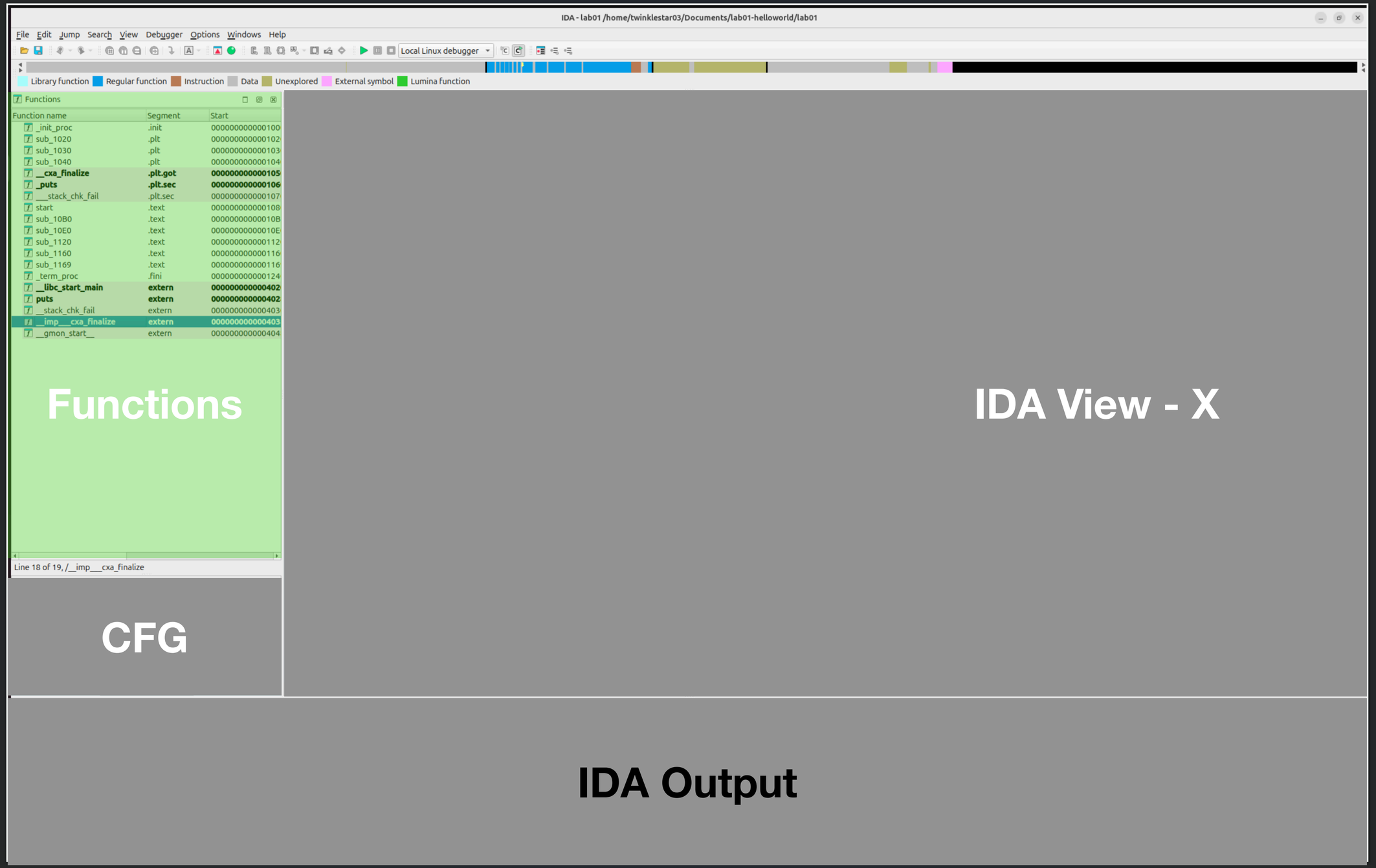

IDA - lab01 /home/twinklestar03/Documents/lab01-helloworld/lab01
File Edit Jump Search View Debugger Options Windows Help
Local Linux debugger
Library function Regular function Instruction Data Unexplored External symbol Lumina function
Functions
Function name Segment Start
_init_proc .init 0000000000001000
sub_1020 .plt 0000000000001020
sub_1030 .plt 0000000000001030
sub_1040 .plt 0000000000001040
__cxa_finalize .plt.got 0000000000001050
_puts .plt.sec 0000000000001060
__stack_chk_fail .plt.sec 0000000000001070
start .text 0000000000001080
sub_10B0 .text 00000000000010B0
sub_10E0 .text 00000000000010E0
sub_1120 .text 0000000000001120
sub_1160 .text 0000000000001160
sub_1169 .text 0000000000001169
_term_proc .fini 0000000000001240
__libc_start_main extern 0000000000004020
puts extern 0000000000004028
__stack_chk_fail extern 0000000000004030
__imp___cxa_finalize extern 0000000000004038
__gmon_start__ extern 0000000000004040
Line 18 of 19, /__imp___cxa_finalize
Graph overview
CFG
IDA View-A
Hex View-1
Local Types
Imports
Exports
; Segment type: Pure code
; Segment permissions: Read/Execute
_text segment para public 'CODE' use64
assume cs:_text
;org 1080h
assume es:nothing, ss:nothing, ds:_data, fs:nothing, gs:nothing
; Attributes: noreturn fuzzy-sp
public start
start proc near
; __unwind {
endbr64
xor ebp, ebp
mov r9, rdx ; rtld_fini
pop rsi ; argc
mov rdx, rsp ; ubp_av
and rsp, 0FFFFFFFFFFFFFFF0h
push rax
push rsp ; stack_end
xor r8d, r8d ; fini
xor ecx, ecx ; init
lea rdi, main ; main
call cs:__libc_start_main_ptr
hlt
; } // starts at 1080
start endp
100.00% (-706,-259) (368,463) 00001080 0000000000001080: start (Synchronized with Hex View-1)
Output
Processing relocations...
DWARF: Looking for GNU DWARF file at "/usr/lib/debug/.build-id/5f/7219ee804e311d5feca647b2cdcd20d5d08c9f.debug"... no.
Marking typical code sequences...
Flushing buffers, please wait...ok
File '/home/twinklestar03/Documents/lab01-helloworld/lab01' has been successfully loaded into the database.
Hex-Rays Decompiler plugin has been loaded (v8.4.0.240527)
License: 48-F4EE-0000-00 Freeware version (1 user)
The decompilation hotkey is F5.
Please check the Edit/Plugins menu for more information.
Propagating type information...
Function argument information has been propagated
The initial autoanalysis has been finished.
IDC
AU: idle Down Disk: 108GB
Tabs
Functions
IDA View - X
IDA Output


IDA - lab01 /home/twinklestar03/Documents/lab01-helloworld/lab01
File Edit Jump Search View Debugger Options Windows Help
Local Linux debugger
Library function Regular function Instruction Data Unexplored External symbol Lumina function
Functions
Function name Segment Start
_init_proc .init 0000000000001000
sub_1020 .plt 0000000000001020
sub_1030 .plt 0000000000001030
sub_1040 .plt 0000000000001040
__cxa_finalize .plt.got 0000000000001050
_puts .plt.sec 0000000000001060
__stack_chk_fail .plt.sec 0000000000001070
start .text 0000000000001080
sub_10B0 .text 00000000000010B0
sub_10E0 .text 00000000000010E0
sub_1120 .text 0000000000001120
sub_1160 .text 0000000000001160
sub_1169 .text 0000000000001169
_term_proc .fini 0000000000001240
__libc_start_main extern 0000000000004020
puts extern 0000000000004028
__stack_chk_fail extern 0000000000004030
__imp___cxa_finalize extern 0000000000004038
__gmon_start__ extern 0000000000004040
Line 18 of 19, /__imp___cxa_finalize
Graph overview
IDA View-A
Hex View-1
Local Types
Imports
Exports
每一個區塊其實都是一個視窗，可以自由拖放
; Segment type: Pure code
; Segment permissions: Read/Execute
_text segment para public 'CODE' use64
assume cs:_text
;org 1080h
assume es:nothing, ss:nothing, ds:_data, fs:nothing, gs:nothing
; Attributes: noreturn fuzzy-sp
public start
start proc near
; __unwind {
endbr64
xor ebp, ebp
mov r9, rdx ; rtld_fini
pop rsi ; argc
mov rdx, rsp ; ubp_av
and rsp, 0FFFFFFFFFFFFFFF0h
push rax
push rsp ; stack_end
xor r8d, r8d ; fini
xor ecx, ecx ; init
lea rdi, main ; main
call cs:__libc_start_main_ptr
hlt
; } // starts at 1080
start endp
100.00% (-706,-259) (368,463) 00001080 0000000000001080: start (Synchronized with Hex View-1)
Output
Processing relocations...
DWARF: Looking for GNU DWARF file at "/usr/lib/debug/.build-id/5f/7219ee804e311d5feca647b2cdcd20d5d08c9f.debug"... no.
Marking typical code sequences...
Flushing buffers, please wait...ok
File '/home/twinklestar03/Documents/lab01-helloworld/lab01' has been successfully loaded into the database.
Hex-Rays Decompiler plugin has been loaded (v8.4.0.240527)
License: 48-F4EE-0000-00 Freeware version (1 user)
The decompilation hotkey is F5.
Please check the Edit/Plugins menu for more information.
Propagating type information...
Function argument information has been propagated
The initial autoanalysis has been finished.
IDC
AU: idle Down Disk: 108GB


IDA - lab01 /home/twinklestar03/Documents/lab01-helloworld/lab01
File Edit Jump Search View Debugger Options Windows Help
Local Linux debugger
Library function Regular function Instruction Data Unexplored External symbol Lumina function
IDA View-A
Hex View-1
Local Types
Imports
Exports
Tabs
IDA View-A
Hex View-1
Local Types
Imports
Exports
Line 18 of 19, /__imp___cxa_finalize

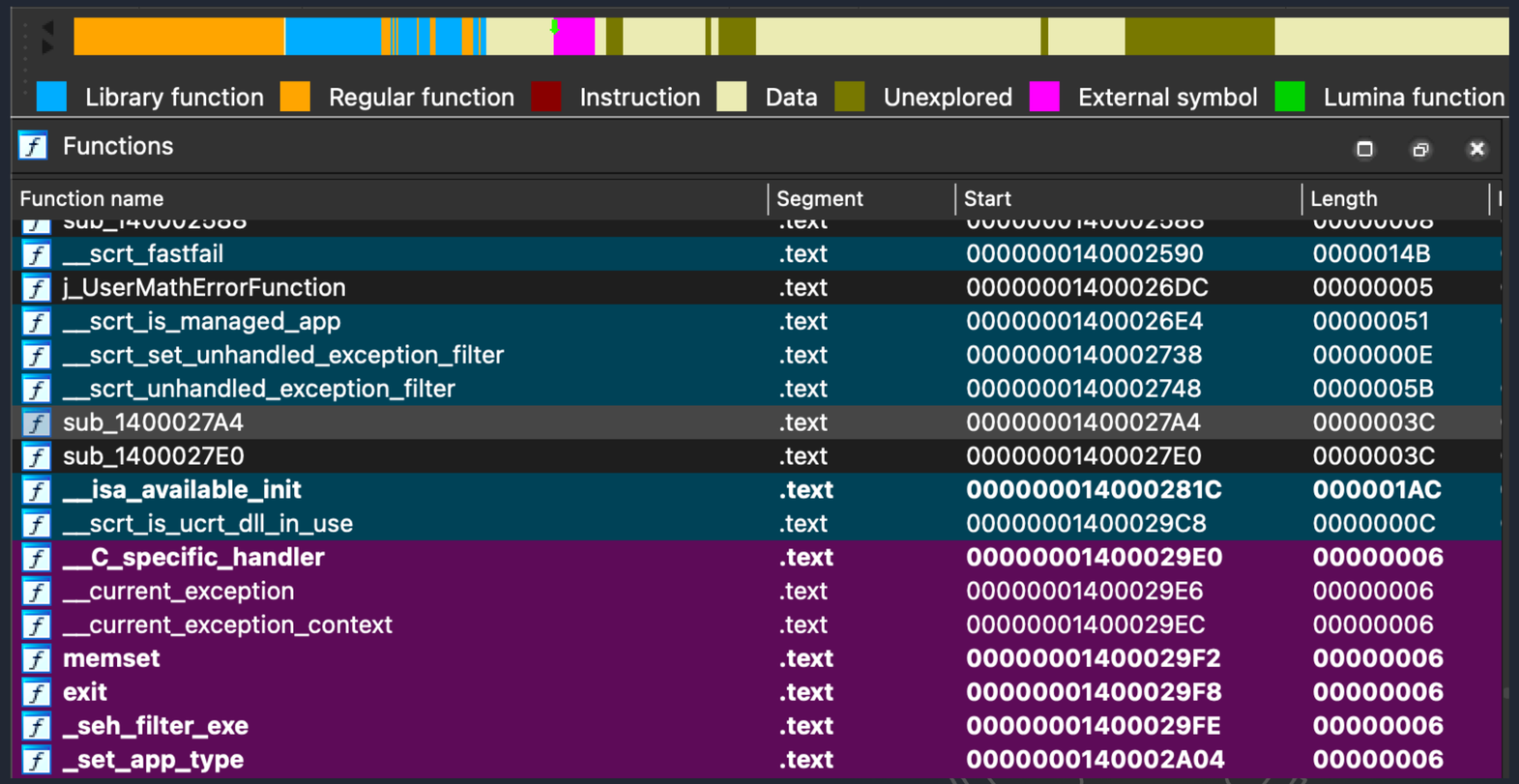
Tabs

- 視窗的集合
- 可以自行決定要有哪些 Tab / Pane
- 通常會有的 Tab
  - Disassembly
  - Decompile
  - Hex View

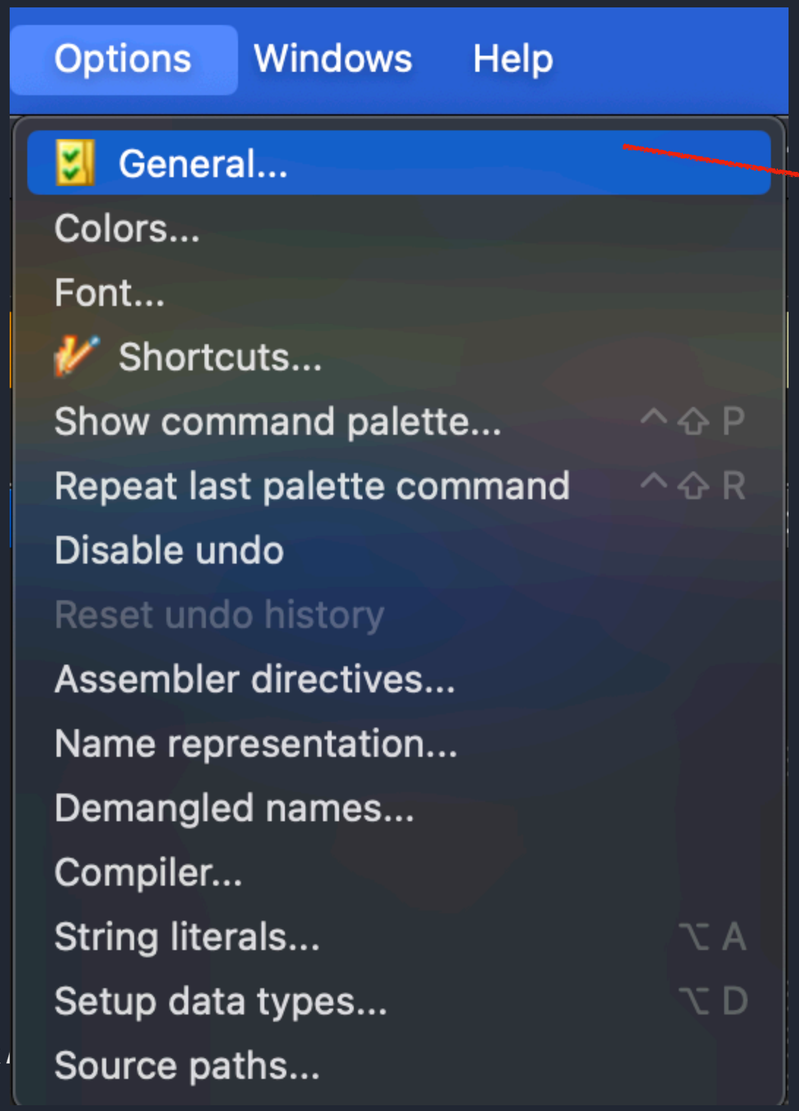

IDA - lab01 /home/twinklestar03/Documents/lab01-helloworld/lab01
File Edit Jump Search View Debugger Options Windows Help
Local Linux debugger
Library function Regular function Instruction Data Unexplored External symbol Lumina function
Functions
Function name Segment Start
_init_proc .init 0000000000001000
sub_1020 .plt 0000000000001020
sub_1030 .plt 0000000000001030
sub_1040 .plt 0000000000001040
__cxa_finalize .plt.got 0000000000001050
_puts .plt.sec 0000000000001060
__stack_chk_fail .plt.sec 0000000000001070
start .text 0000000000001080
sub_10B0 .text 00000000000010B0
sub_10E0 .text 00000000000010E0
sub_1120 .text 0000000000001120
sub_1160 .text 0000000000001160
sub_1169 .text 0000000000001169
_term_proc .fini 0000000000001240
__libc_start_main extern 0000000000004020
puts extern 0000000000004028
__stack_chk_fail extern 0000000000004030
__imp___cxa_finalize extern 0000000000004038
__gmon_start__ extern 0000000000004040
Line 18 of 19, /__imp___cxa_finalize
Functions
CFG
IDA View - X
IDA Output
Tabs

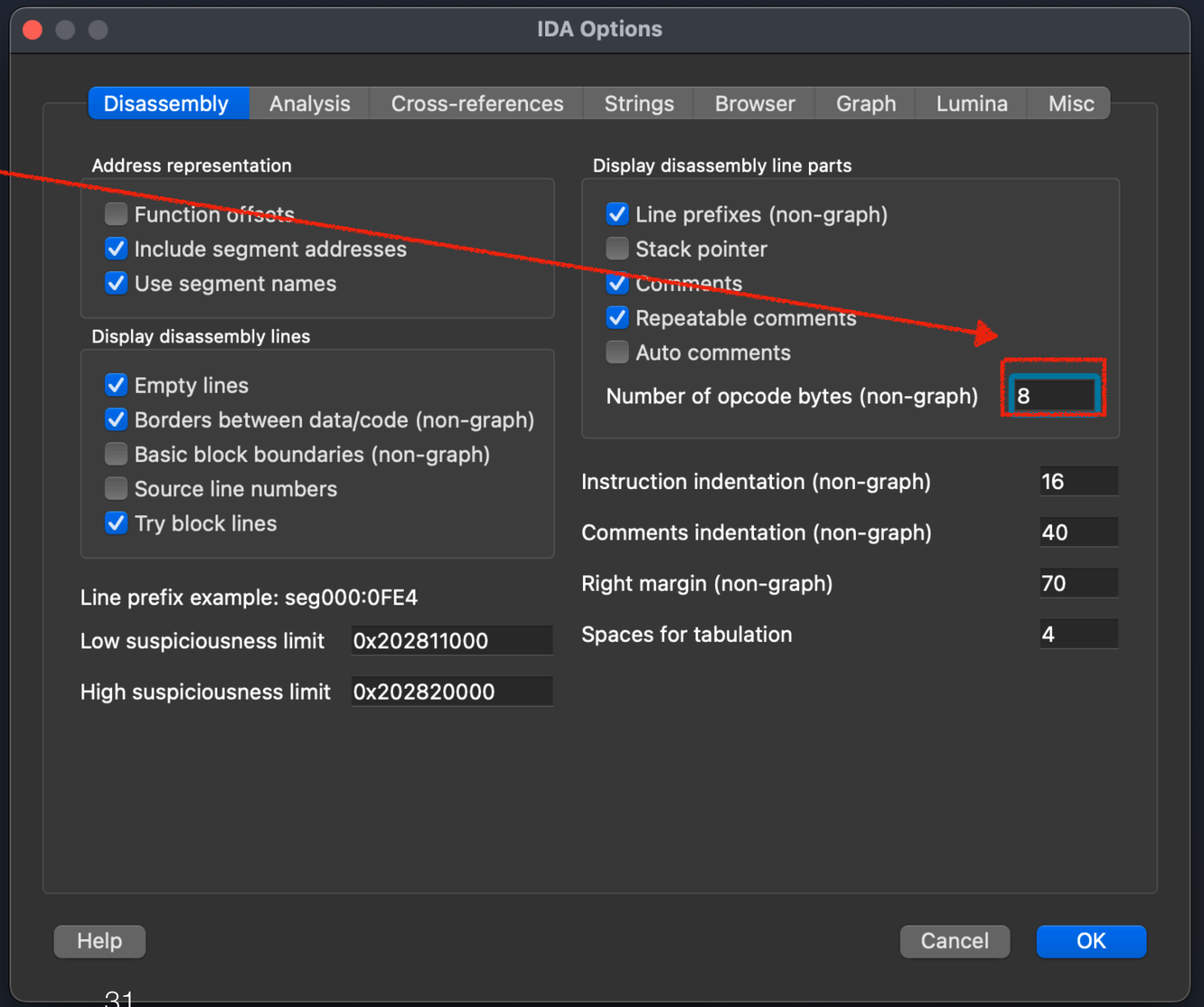
Function View

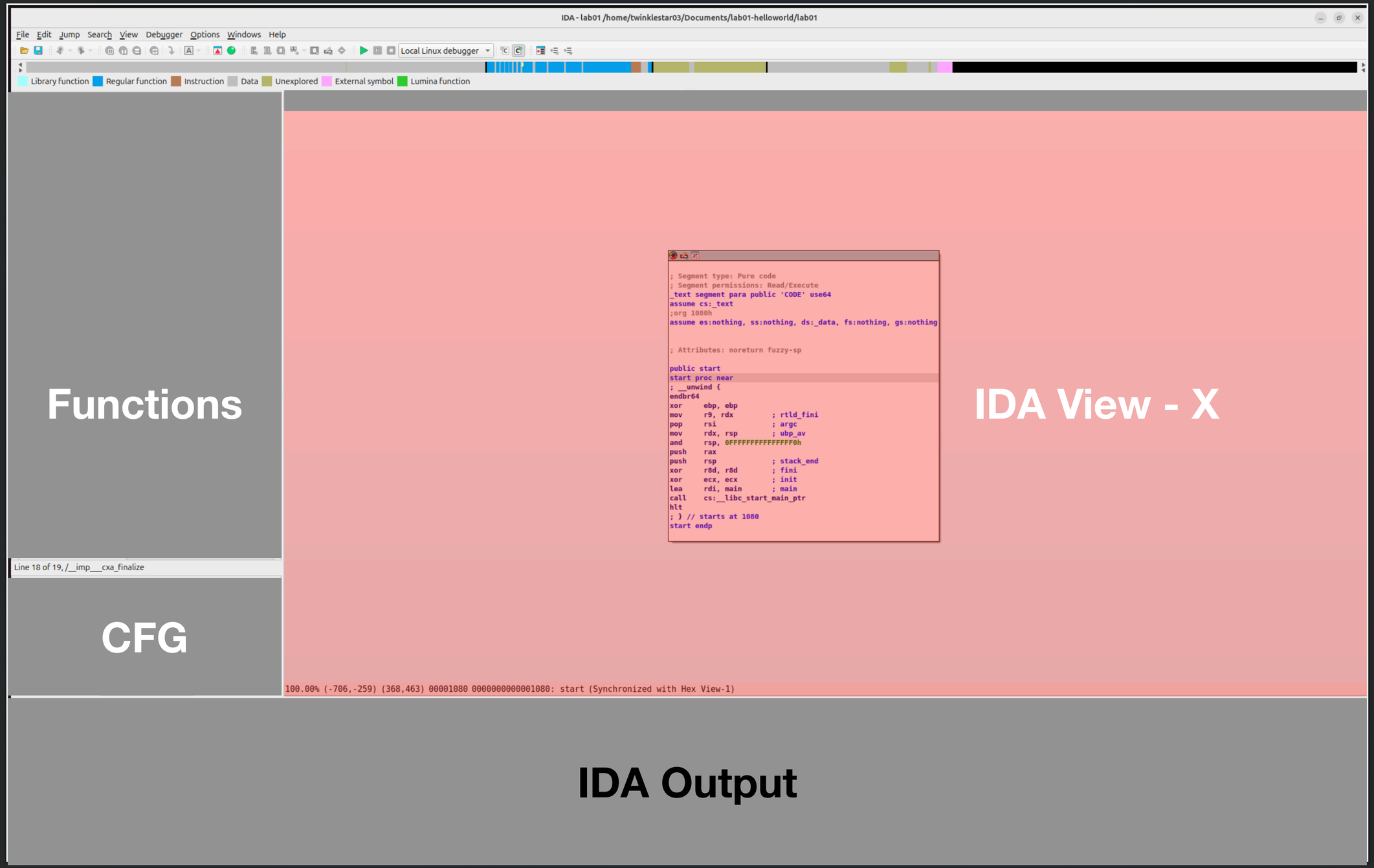
- 所有 IDA 自動分析找到的 function
- 顏色標示 function 類型
  - Library function
  - External symbol
  - **Regular function (重點)**
- 搜尋
  - 直接輸入：從頭匹配
  - [Ctrl+ f]：模糊搜尋



Show Op Code

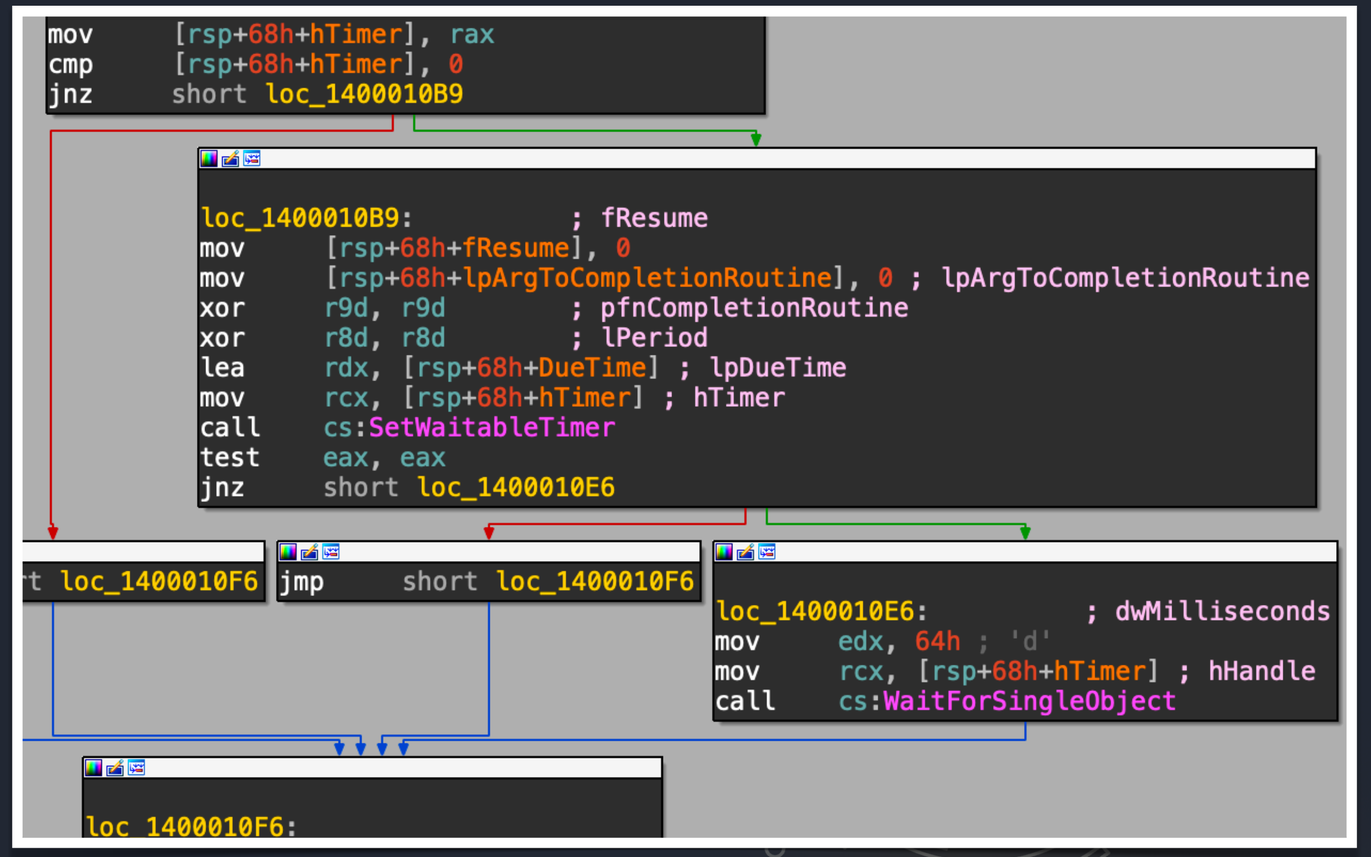

Options
Windows
Help
General...
Colors...
Font...
Shortcuts...
Show command palette... ^ ⇧ P
Repeat last palette command ^ ⇧ R
Disable undo
Reset undo history
Assembler directives...
Name representation...
Demangled names...
Compiler...
String literals... ⌥ A
Setup data types... ⌥ D
Source paths...


IDA Options
Disassembly
Analysis
Cross-references
Strings
Browser
Graph
Lumina
Misc
Address representation
Function offsets
Include segment addresses
Use segment names
Display disassembly lines
Empty lines
Borders between data/code (non-graph)
Basic block boundaries (non-graph)
Source line numbers
Try block lines
Line prefix example: seg000:0FE4
Low suspiciousness limit 0x202811000
High suspiciousness limit 0x202820000
Display disassembly line parts
Line prefixes (non-graph)
Stack pointer
Comments
Repeatable comments
Auto comments
Number of opcode bytes (non-graph) 8
Instruction indentation (non-graph) 16
Comments indentation (non-graph) 40
Right margin (non-graph) 70
Spaces for tabulation 4
Help
Cancel
OK


IDA - lab01 /home/twinklestar03/Documents/lab01-helloworld/lab01
File Edit Jump Search View Debugger Options Windows Help
Local Linux debugger
Library function Regular function Instruction Data Unexplored External symbol Lumina function
Tabs
Functions
; Segment type: Pure code
; Segment permissions: Read/Execute
_text segment para public 'CODE' use64
assume cs:_text
;org 1080h
assume es:nothing, ss:nothing, ds:_data, fs:nothing, gs:nothing
; Attributes: noreturn fuzzy-sp
public start
start proc near
; __unwind {
endbr64
xor ebp, ebp
mov r9, rdx ; rtld_fini
pop rsi ; argc
mov rdx, rsp ; ubp_av
and rsp, 0FFFFFFFFFFFFFFF0h
push rax
push rsp ; stack_end
xor r8d, r8d ; fini
xor ecx, ecx ; init
lea rdi, main ; main
call cs:__libc_start_main_ptr
hlt
; } // starts at 1080
start endp
IDA View - X
Line 18 of 19, /__imp___cxa_finalize
CFG
100.00% (-706,-259) (368,463) 00001080 0000000000001080: start (Synchronized with Hex View-1)
IDA Output

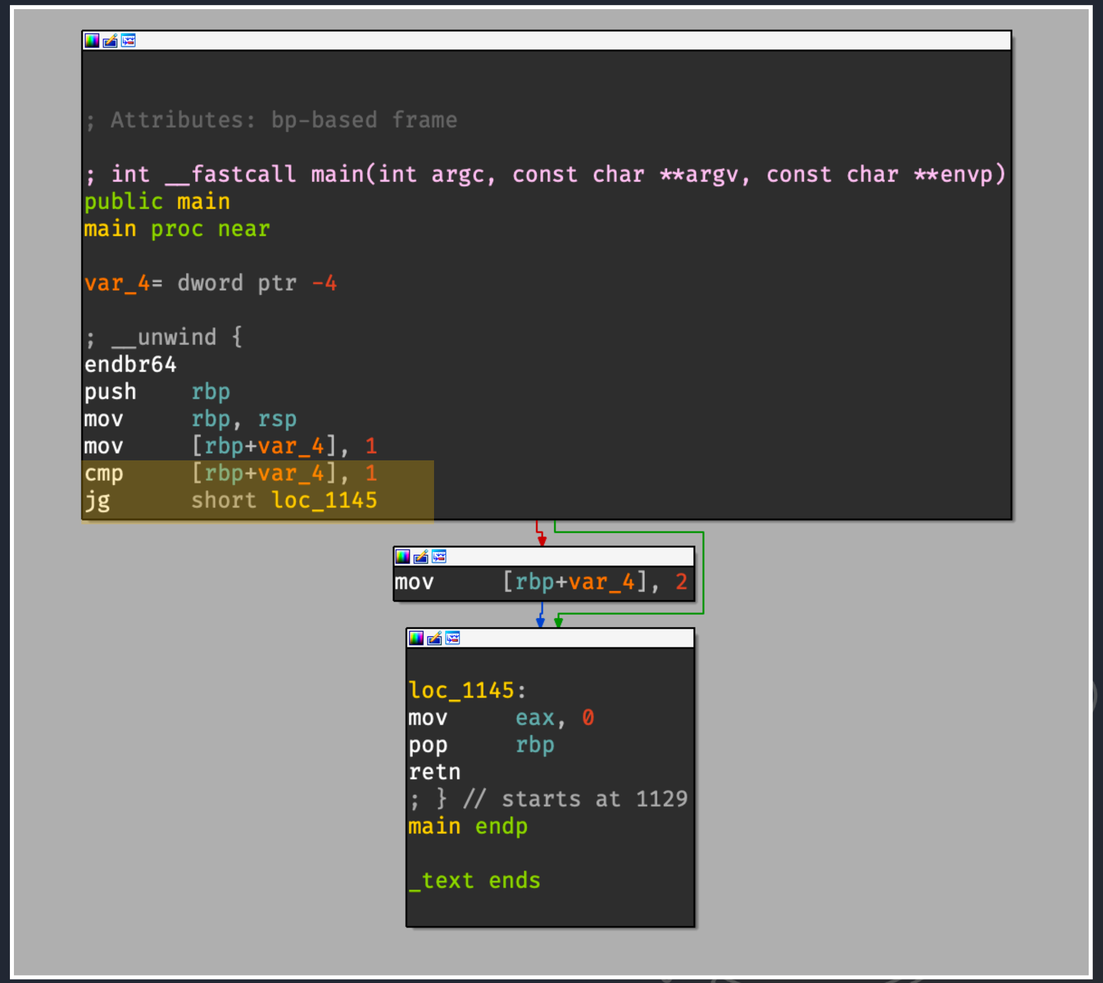
Disassembly View (Graph)

- 優點
  - 視覺化 Control Flow
- 缺點
  - 圖大、複雜
  - 難 navigate
- [Space] 切換為 Text view



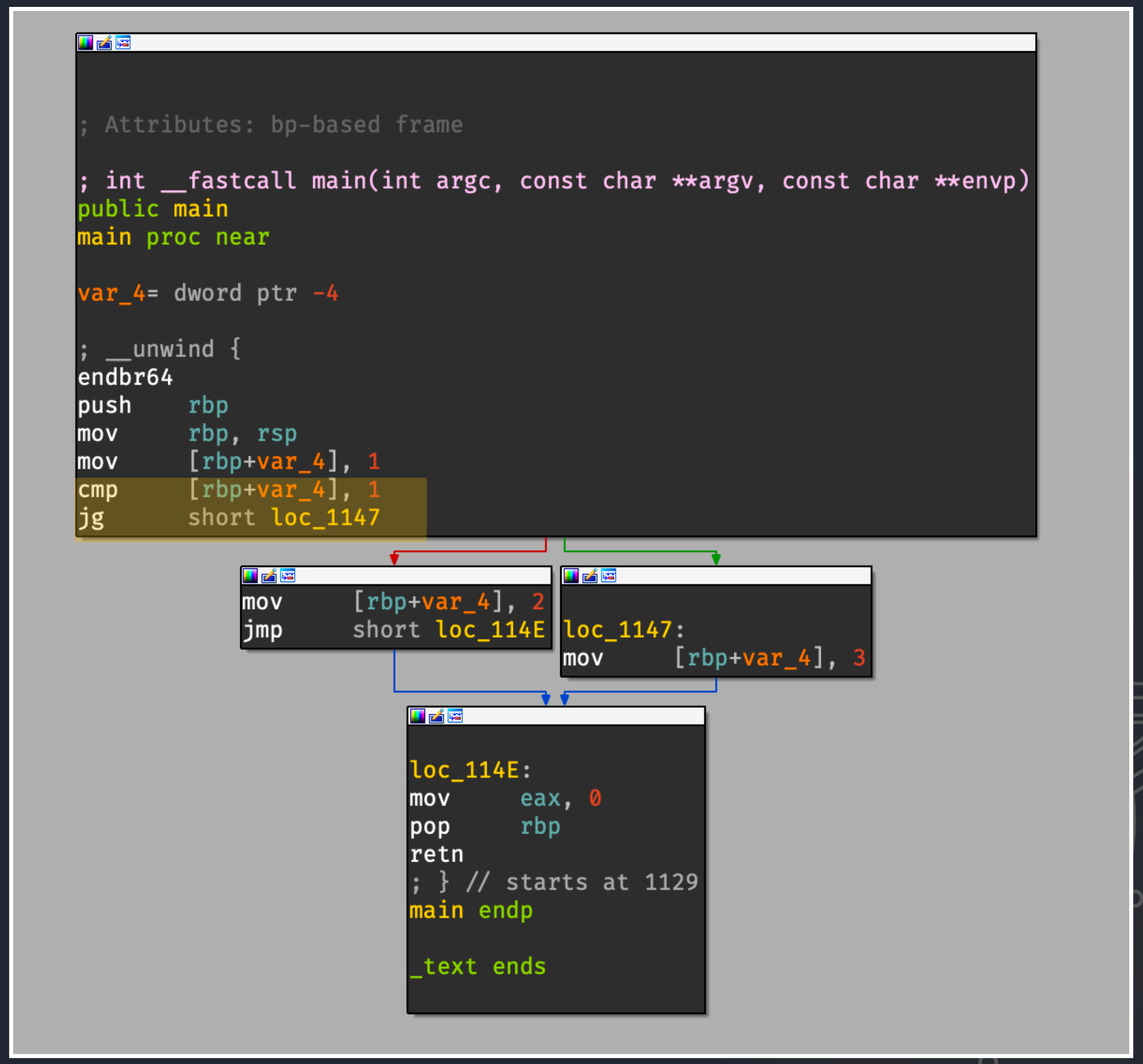
IF

```c
int main() {
    int a = 1;
    if (a < 2) {
        a = 2;
    }

    return 0;
}
```



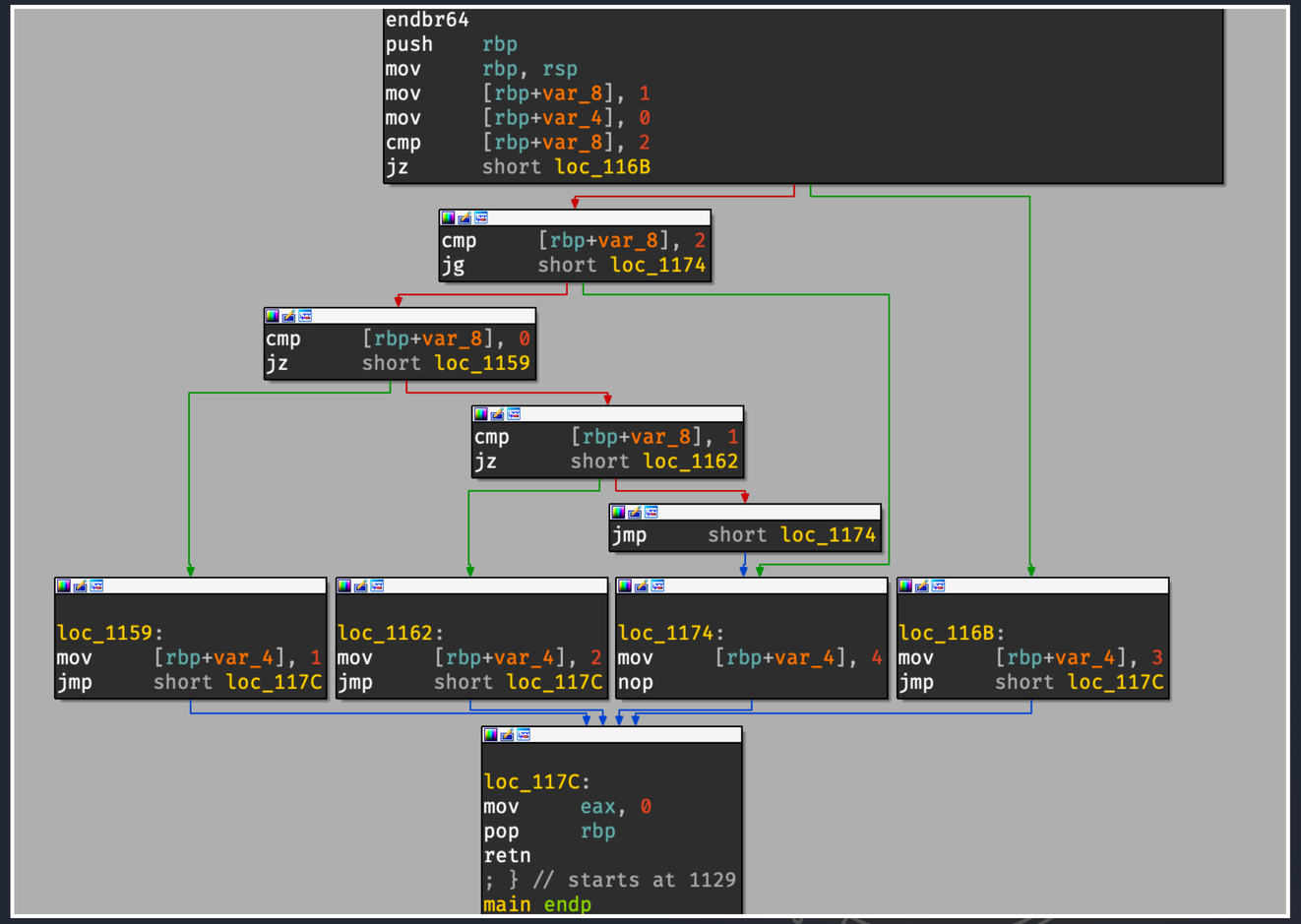
IF-ELSE

```c
int main() {
    int a = 1;
    if (a < 2)
{       a = 2;
    } else {
        a = 3;
    }

    return 0;
}
```



Switch (3 Cases)

```c
int main() {
    int a = 1;
    int b = 0;

    switch (a) {
        case 0:
            b = 1;
            break;
        case 1:
            b = 2;
            break;
        case 2:
            b = 3;
            break;
        default:
            b = 4;
            break;
    }

    return 0;
}
```



Switch (5 Cases)

Case夠多的時候，用 jump table 來做 switch
因為直接 if-else 會太多 instruction

Default Case

Disassembly View (Text)

- 優點
  - 簡潔
  - 方便看 op code
  - 顯示資訊 (e.g. xref)
- [Space] 切換為 Graph view

```
10B9                    loc_1400010B9:                          ; CODE XREF: sub_140001020+95↑j
10B9 C7 44 24 28 00 00                  mov     [rsp+68h+fResume], 0 ; fResume
10B9 00 00
10C1 48 C7 44 24 20 00                  mov     [rsp+68h+lpArgToCompletionRoutine], 0 ; lpArgTo
10C1 00 00 00
10CA 45 33 C9                           xor     r9d, r9d        ; pfnCompletionRoutine
10CD 45 33 C0                           xor     r8d, r8d        ; lPeriod
10D0 48 8D 54 24 50                     lea     rdx, [rsp+68h+DueTime] ; lpDueTime
10D5 48 8B 4C 24 30                     mov     rcx, [rsp+68h+hTimer] ; hTimer
10DA FF 15 50 20 00 00                  call    cs:SetWaitableTimer
10E0 85 C0                              test    eax, eax
10E2 75 02                              jnz     short loc_1400010E6
10E4 EB 10                              jmp     short loc_1400010F6
10E6                    ; ---------------------------------------------------------------------------
10E6
10E6                    loc_1400010E6:                          ; CODE XREF: sub_140001020+C2↑j
10E6 BA 64 00 00 00                     mov     edx, 64h ; 'd'  ; dwMilliseconds
10EB 48 8B 4C 24 30                     mov     rcx, [rsp+68h+hTimer] ; hHandle
10F0 FF 15 1A 20 00 00                  call    cs:WaitForSingleObject
10F6
10F6                    loc_1400010F6:                          ; CODE XREF: sub_140001020+6B↑j
10F6                                                            ; sub_140001020+97↑j ...
10F6 48 8B 4C 24 58                     mov     rcx, [rsp+68h+var_10]
10FB 48 33 CC                           xor     rcx, rsp        ; StackCookie
10FE E8 0D 0D 00 00                     call    __security_check_cookie
```

Decompile View

- 編譯 compile (C -> assembly)
- 反編譯 Decompile (assembly -> C)
- 看 code 速度 ⬆️⬆️⬆️
- 有時候會出錯 / 被搞事 💥🔥😡

  ➡ 乖乖回去看 Disassembly view 🫠

```
1 int __fastcall main(int argc, const char
2 {
3   Sleep(0x1B7740u);
4   sub_140001C80();
5   sub_140001030();
6   sub_140001120();
7   sub_140001BF0();
8   return 0;
9 }
```

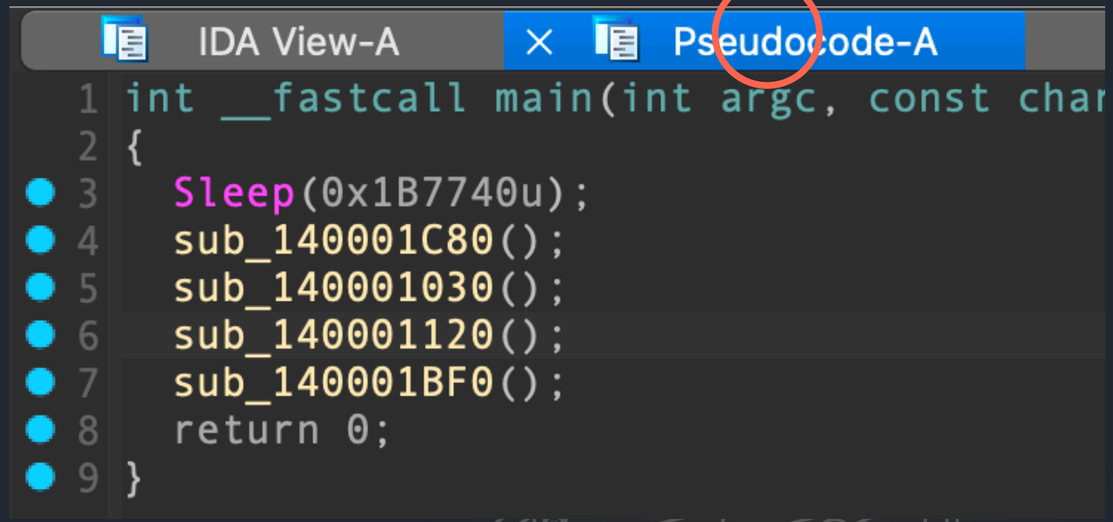
召喚方式：[F5] / [Tab]

Decompile View Shortcuts

- [Tab]：切換 Decompile / Disassembly View
- **[Enter]**：進入 function / data
- **[Esc]**：返回上一步
- [↑↓←→]：移動
- [Opt/Alt + ↑↓]：跳到上 / 下次出現位置
- **[n]**：重新命名 function, variables
- **[y]**：指定 function, variables 的型別
- [Opt/Alt+a]：將 data 轉為字串
- [*]：將 data 轉為 array
- **[/]**：新增、編輯註解
- **[h]**：將常數顯示為 dec / hex
- **[r]**：將常數顯示為 char
- [x]：顯示 cross reference
- [Ctrl+e]：顯示 entry points

Disassembly, Decompile 我全都要

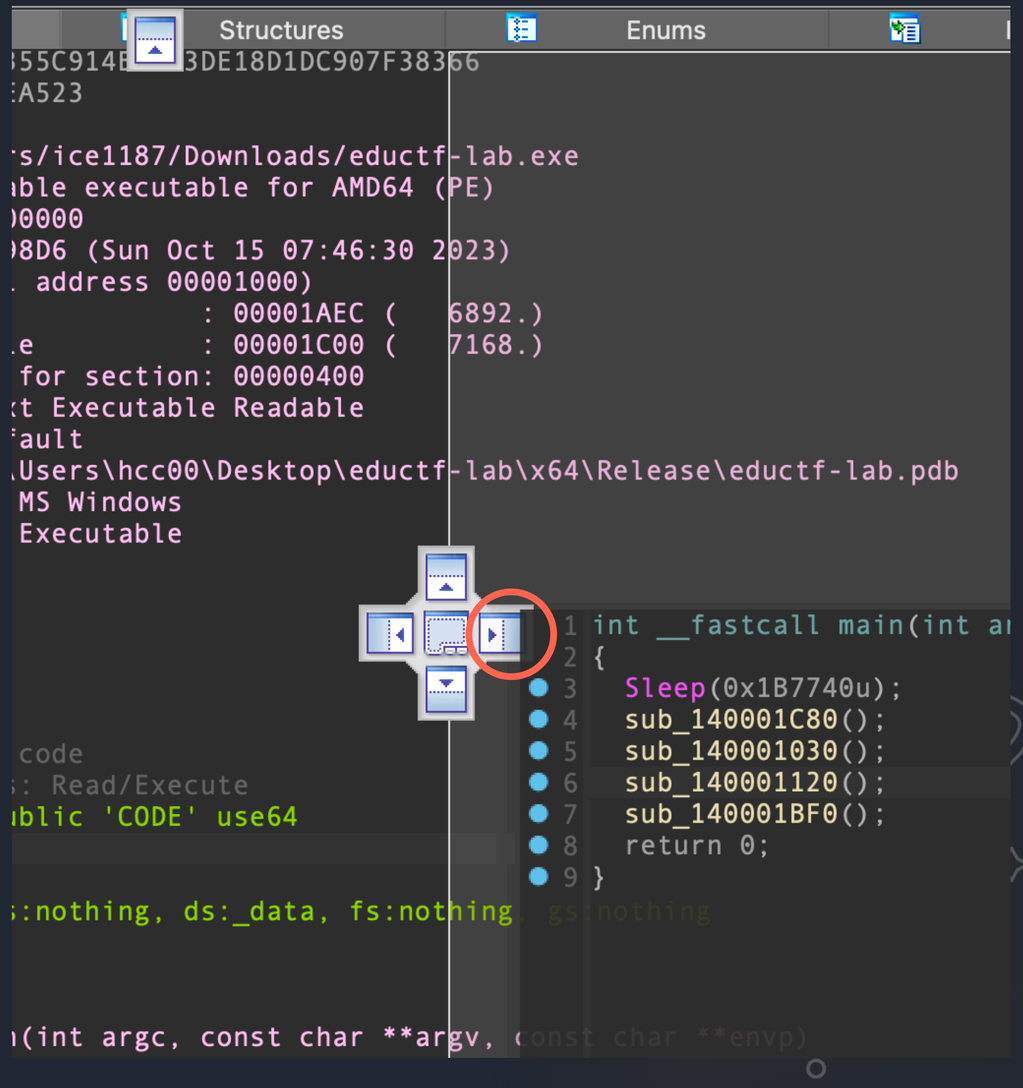
1. F5 或 Tab 叫出 decompile view
2. **按住 Pseudocode-A**



Disassembly, Decompile 我全都要

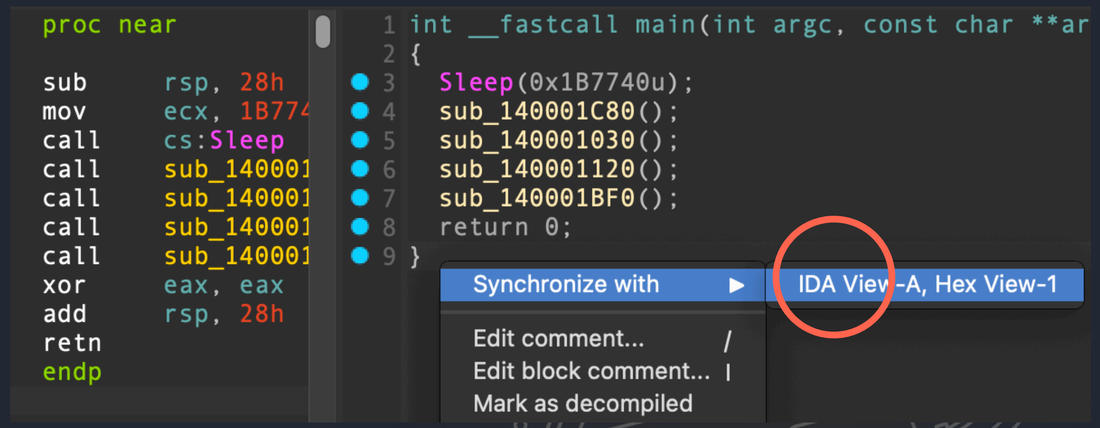
1. F5 或 Tab 叫出 decompile view
2. 按住 Pseudocode-A
3. **拖動到畫面中間的向右箭頭區域**

這一招在所有 Pane 都可以用



Disassembly, Decompile 我全都要

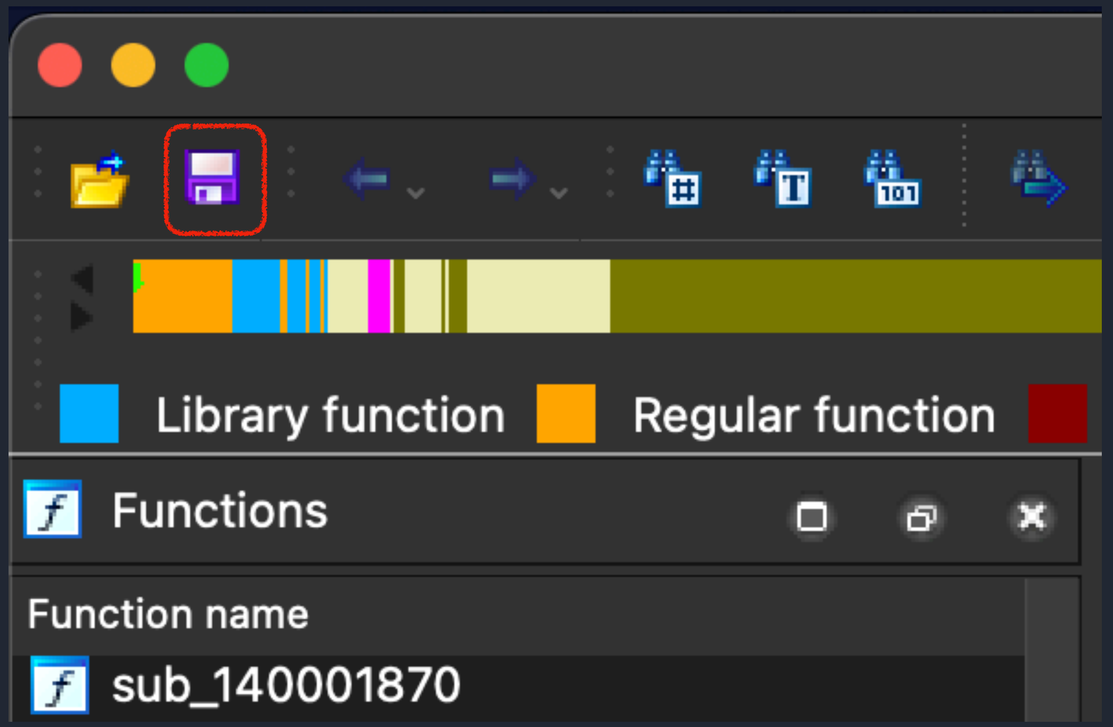
1. F5 或 Tab 叫出 decompile view
2. 按住 Pseudocode-A
3. 拖動到畫面中間的向右箭頭區域
4. **右鍵 -> "Synchronize with"**

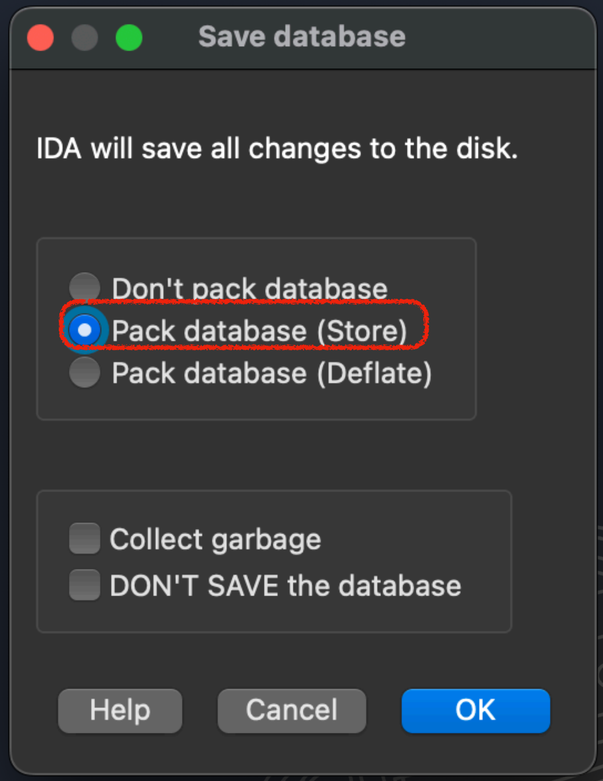


Disassembly, Decompile 我全都要

1. F5 或 Tab 叫出 decompile view
2. 按住 Pseudocode-A
3. 拖動到畫面中間的向右箭頭區域
4. 右鍵 -> "Synchronize with"
5. **左右邊 sync 在一起！**

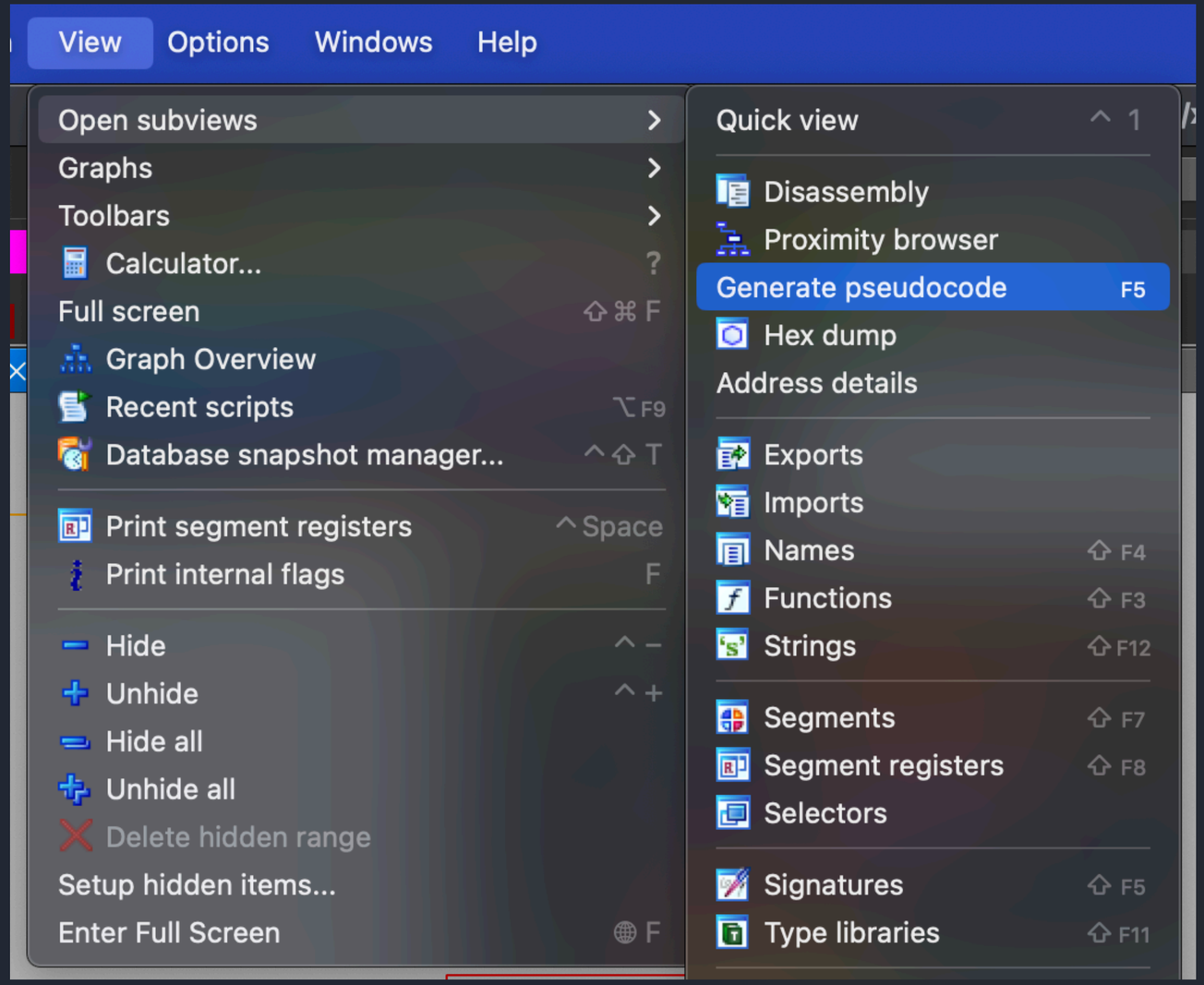
Save Your Analyzed Result !





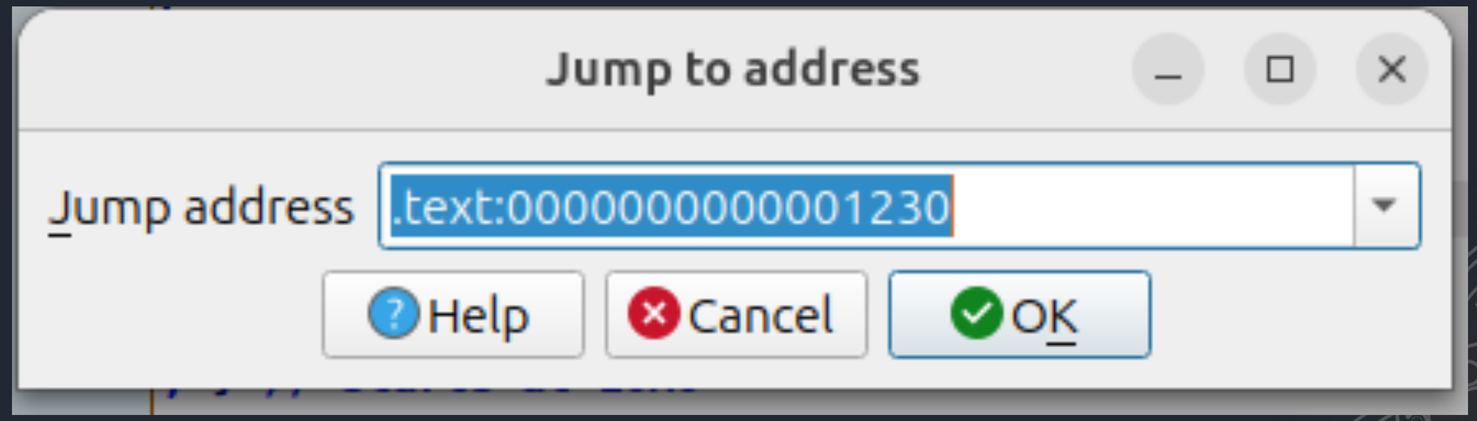
有視窗被我弄不見了 QQ

1. Menu Bar
2. 點開 "View"
3. 選擇 "Open subviews"
4. 你要的 View



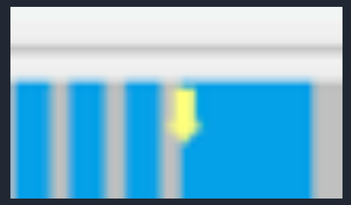
Goto Address / Symbol

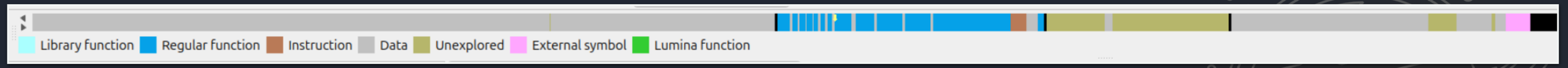
在視窗按 [G]
輸入 Address / Symbol



經常被忽視的小東西

用滑鼠在狀態條上可以改變位址
小箭頭代表當前位址

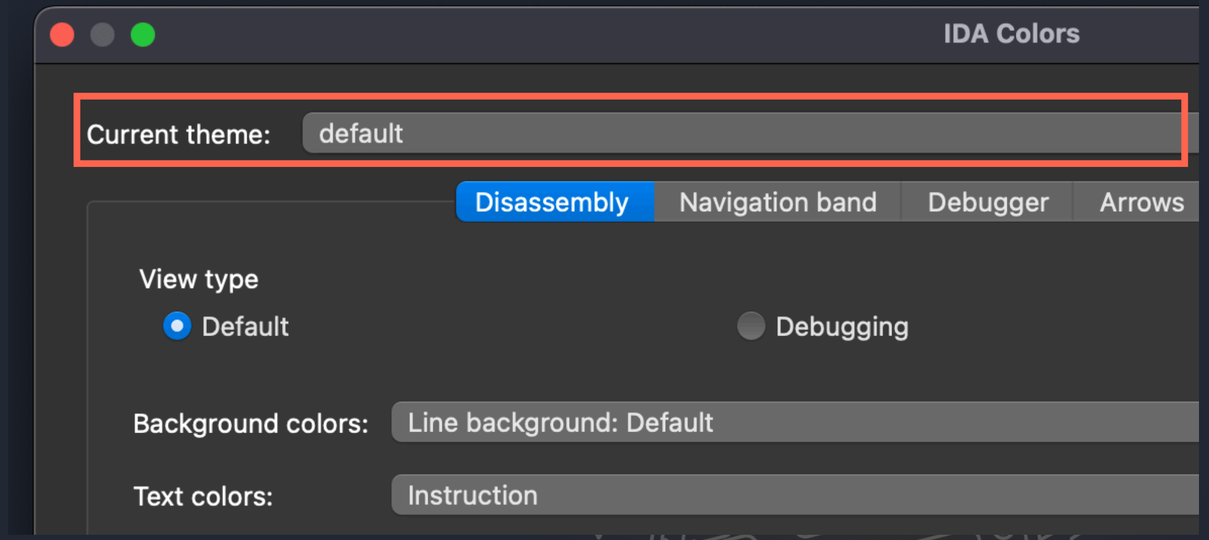




顏色代表標記種類

Dark Theme 😎

1. Menu Bar
2. 點開 "Options"
3. 選擇 "Colors..."
4. "Current Theme" 選擇 "Dark"



DEMO

Mark Everything

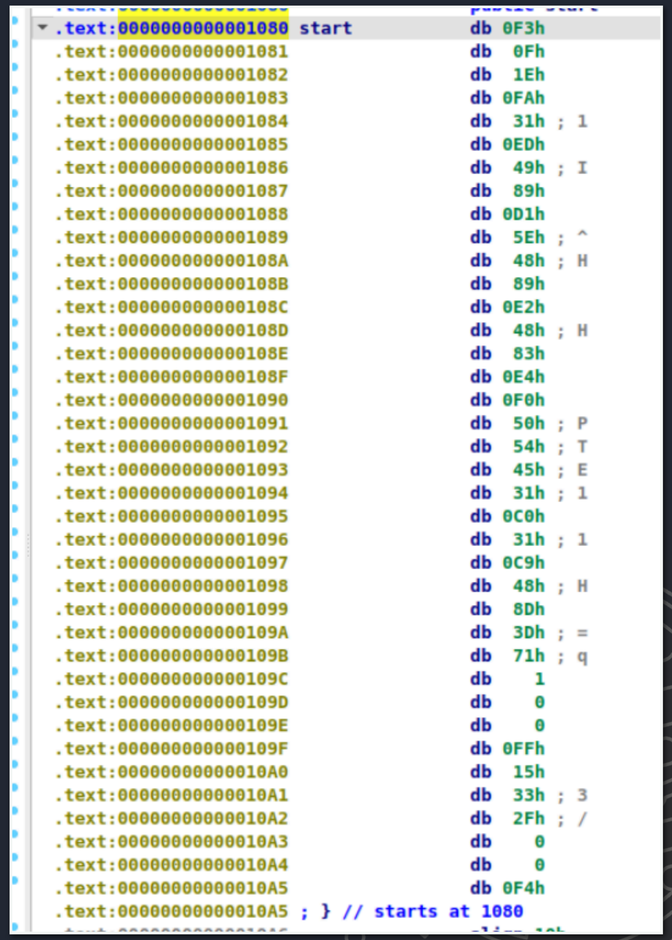
Mark Code

Mark Code

- IDA 不認為現在這一塊是可執行的機器碼
  - 不是 Code 就不能當成 Function 就不能 F5
  - 不能 Decompile = 悲劇
- 需要人工介入分析

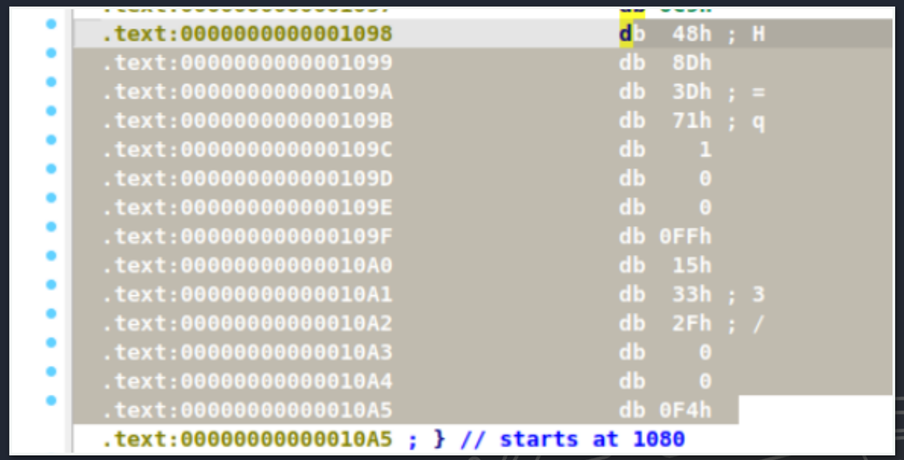
Mark Code

- 在 Text View 下會比較好工作
- 右邊這是一塊沒有被定義的區塊
  - 土黃色



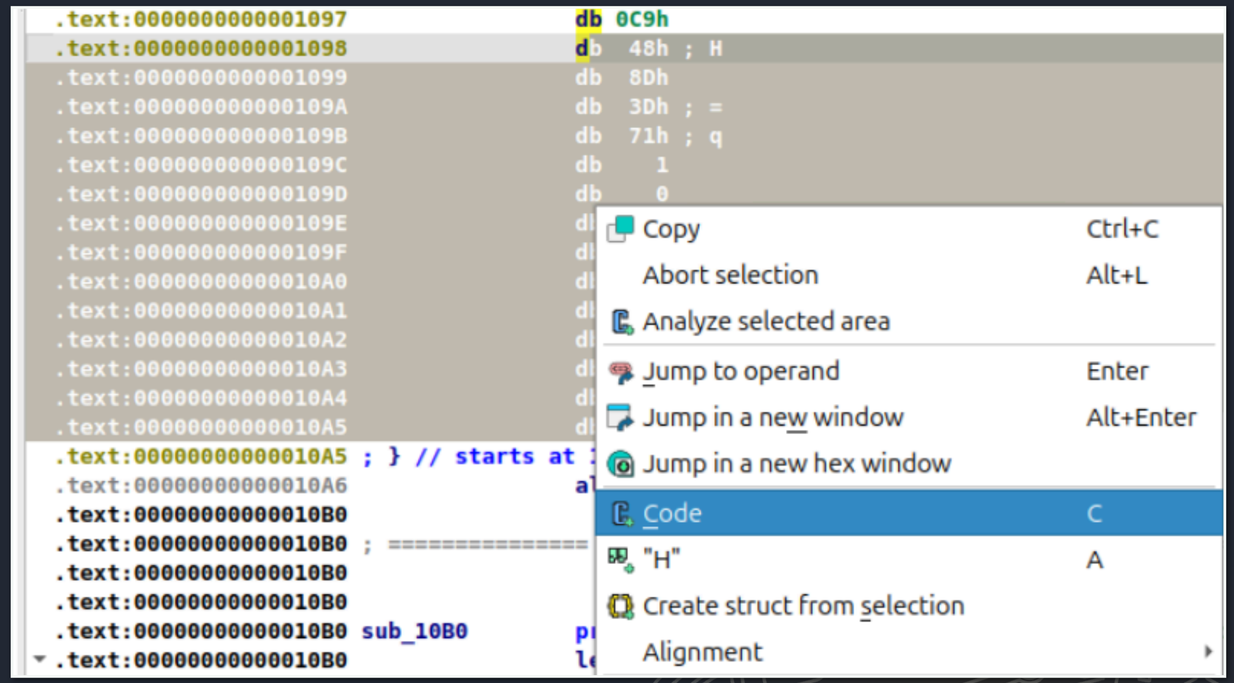
Mark Code

- 框選你想要標記的區域
  - 行數有選到就好，不用選滿



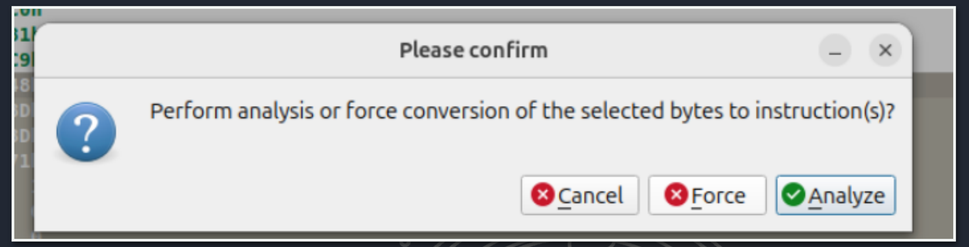
Mark Code

- 右鍵 “Code”
  - 或是 [C]



Mark Code

- 有自信自己選的範圍是對的就 Force
- 沒自信就 Analysis 讓 IDA 幫忙補齊



Mark Code

- 標記完就會變成 Instruction
  - 紅棕色
- 此時還是不能 F5
  - Decompile 需要 Function 的資訊
  - Mark Function!

```
.text:0000000000001098 ; ---------------------------------------------------------------------------
.text:0000000000001098                 lea     rdi, main
.text:000000000000109F                 call    cs:off_3FD8
.text:00000000000010A5                 hlt
.text:00000000000010A5 ; } // starts at 1080
.text:00000000000010A5 ; ---------------------------------------------------------------------------
```

Mark Code - Notice

- 因為 x64 指令集密度很高，很可能會解錯
  - 解錯會影響到語意
- Decompile 會解錯甚至是無法運作

Mark Function

Mark Function

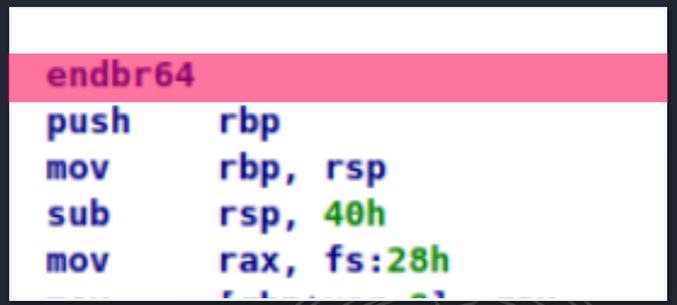
- 標記 Function 要先找出開頭與結尾
- Function Prologue / Function Epilogue
  - 用來分配與收回 Stack Frame 的片段

```
endbr64
push    rbp
mov     rbp, rsp
sub     rsp, 40h
mov     rax, fs:28h
```

```
leave
retn
```

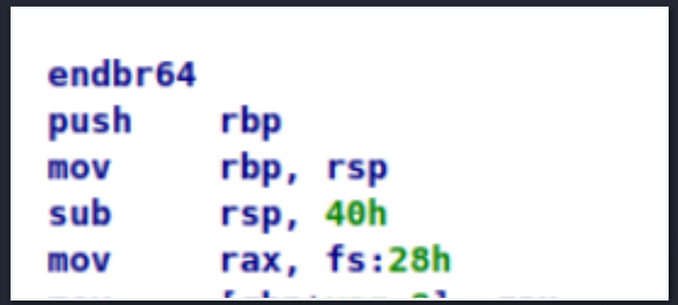
Mark Function

- 在 Function 開頭 [P]
  - 或是右鍵 “Create Function”
  - IDA 自動分析 Function 結構並標記



Mark Function

- 凡事總會有個萬一...
  - 自動分析壞掉，IDA 認不出 Function
- 手動框選整個 Function Body 後再 [P]



Mark Function

- 我明明都框起來了，怎麼還是不能 Mark Function?
- 看看 IDA Output 有沒有錯誤訊息
  - 

  - 看著訊息上的記憶體位置修修補補

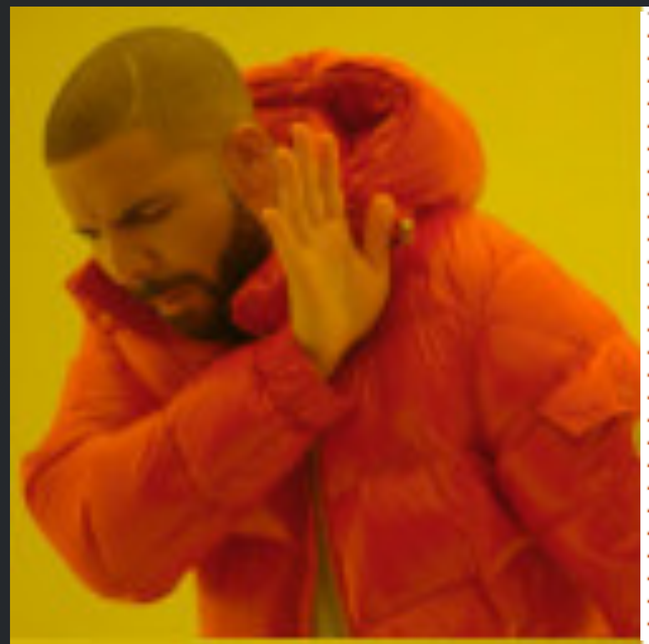
Mark Function

```
.text:0000000000001080 ; ===========================================================================
.text:0000000000001080
.text:0000000000001080 ; Segment type: Pure code
.text:0000000000001080 ; Segment permissions: Read/Execute
.text:0000000000001080 _text           segment para public 'CODE' use64
.text:0000000000001080                 assume cs:_text
.text:0000000000001080                 ;org 1080h
.text:0000000000001080                 assume es:nothing, ss:nothing, ds:_data, fs:nothing, gs:nothing
.text:0000000000001080
.text:0000000000001080                 public start
.text:0000000000001080 start:                              ; DATA XREF: LOAD:0000000000000018↑o
.text:0000000000001080 ; __unwind {
.text:0000000000001080                 endbr64
.text:0000000000001084                 xor     ebp, ebp
.text:0000000000001086                 mov     r9, rdx
.text:0000000000001089                 pop     rsi
.text:000000000000108A                 mov     rdx, rsp
.text:000000000000108D                 and     rsp, 0FFFFFFFFFFFFFFF0h
.text:0000000000001091                 push    rax
.text:0000000000001092                 push    rsp
.text:0000000000001093                 xor     r8d, r8d
.text:0000000000001096                 xor     ecx, ecx
.text:0000000000001098                 lea     rdi, main
.text:000000000000109F                 call    cs:off_3FD8
.text:00000000000010A5                 hlt
.text:00000000000010A5 ; } // starts at 1080
.text:00000000000010A5 ; ---------------------------------------------------------------------------
```

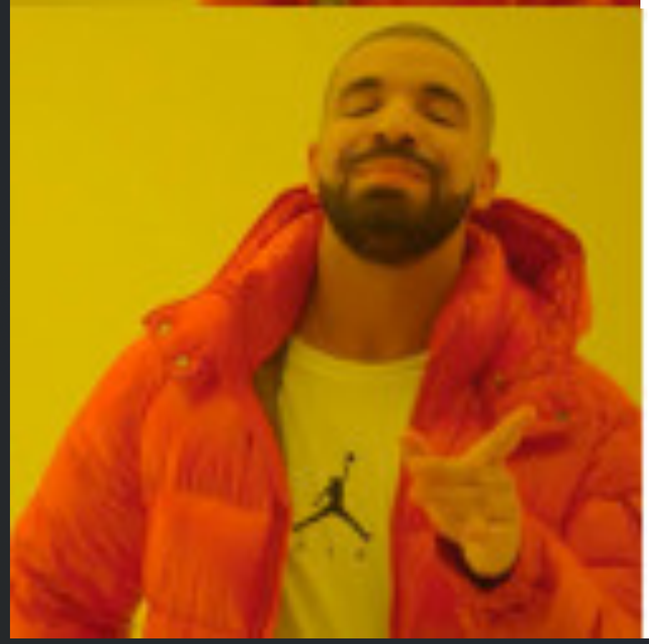
不是 Function

```
.text:0000000000001080
.text:0000000000001080                 public start
.text:0000000000001080 start           proc near               ; DATA XREF: LOAD:000
.text:0000000000001080 ; __unwind {
.text:0000000000001080                 endbr64
.text:0000000000001084                 xor     ebp, ebp
.text:0000000000001086                 mov     r9, rdx         ; rtld_fini
.text:0000000000001089                 pop     rsi             ; argc
.text:000000000000108A                 mov     rdx, rsp        ; ubp_av
.text:000000000000108D                 and     rsp, 0FFFFFFFFFFFFFFF0h
.text:0000000000001091                 push    rax
.text:0000000000001092                 push    rsp             ; stack_end
.text:0000000000001093                 xor     r8d, r8d        ; fini
.text:0000000000001096                 xor     ecx, ecx        ; init
.text:0000000000001098                 lea     rdi, main       ; main
.text:000000000000109F                 call    cs:off_3FD8
.text:00000000000010A5                 hlt
.text:00000000000010A5 ; } // starts at 1080
.text:00000000000010A5 start           endp
.text:00000000000010A5
.text:00000000000010A5 ; ---------------------------------------------------------------------------
```

是 Function
(可以 F5)



```
.text:0000000000001080 ; ===========================================================================
.text:0000000000001080
.text:0000000000001080 ; Segment type: Pure code
.text:0000000000001080 ; Segment permissions: Read/Execute
.text:0000000000001080 _text           segment para public 'CODE' use64
.text:0000000000001080                 assume cs:_text
.text:0000000000001080                 ;org 1080h
.text:0000000000001080                 assume es:nothing, ss:nothing, ds:_data, fs:nothing, gs:nothing
.text:0000000000001080
.text:0000000000001080                 public start
.text:0000000000001080 start:                                  ; DATA XREF: LOAD:0000000000000018↑o
.text:0000000000001080 ; __unwind {
.text:0000000000001080                 endbr64
.text:0000000000001084                 xor     ebp, ebp
.text:0000000000001086                 mov     r9, rdx
.text:0000000000001089                 pop     rsi
.text:000000000000108A                 mov     rdx, rsp
.text:000000000000108D                 and     rsp, 0FFFFFFFFFFFFFFF0h
.text:0000000000001091                 push    rax
.text:0000000000001092                 push    rsp
.text:0000000000001093                 xor     r8d, r8d
.text:0000000000001096                 xor     ecx, ecx
.text:0000000000001098                 lea     rdi, main
.text:000000000000109F                 call    cs:off_3FD8
.text:00000000000010A5                 hlt
.text:00000000000010A5 ; } // starts at 1080
.text:00000000000010A5 ; ---------------------------------------------------------------------------
```



```
.text:0000000000001080
.text:0000000000001080                 public start
.text:0000000000001080 start           proc near               ; DATA XREF: LOAD:000
.text:0000000000001080 ; __unwind {
.text:0000000000001080                 endbr64
.text:0000000000001084                 xor     ebp, ebp
.text:0000000000001086                 mov     r9, rdx         ; rtld_fini
.text:0000000000001089                 pop     rsi             ; argc
.text:000000000000108A                 mov     rdx, rsp        ; ubp_av
.text:000000000000108D                 and     rsp, 0FFFFFFFFFFFFFFF0h
.text:0000000000001091                 push    rax
.text:0000000000001092                 push    rsp             ; stack_end
.text:0000000000001093                 xor     r8d, r8d        ; fini
.text:0000000000001096                 xor     ecx, ecx        ; init
.text:0000000000001098                 lea     rdi, main       ; main
.text:000000000000109F                 call    cs:off_3FD8
.text:00000000000010A5                 hlt
.text:00000000000010A5 ; } // starts at 1080
.text:00000000000010A5 start           endp
.text:00000000000010A5
.text:00000000000010A5 ; ---------------------------------------------------------------
```

Mark Data

Mark Data - Visual Cue

這是 Data

```
.rodata:0000000000002008 byte_2008        db 46h                  ; DATA XREF: .data:0000000000004010↓o
```

這啥都不是 (undefine)

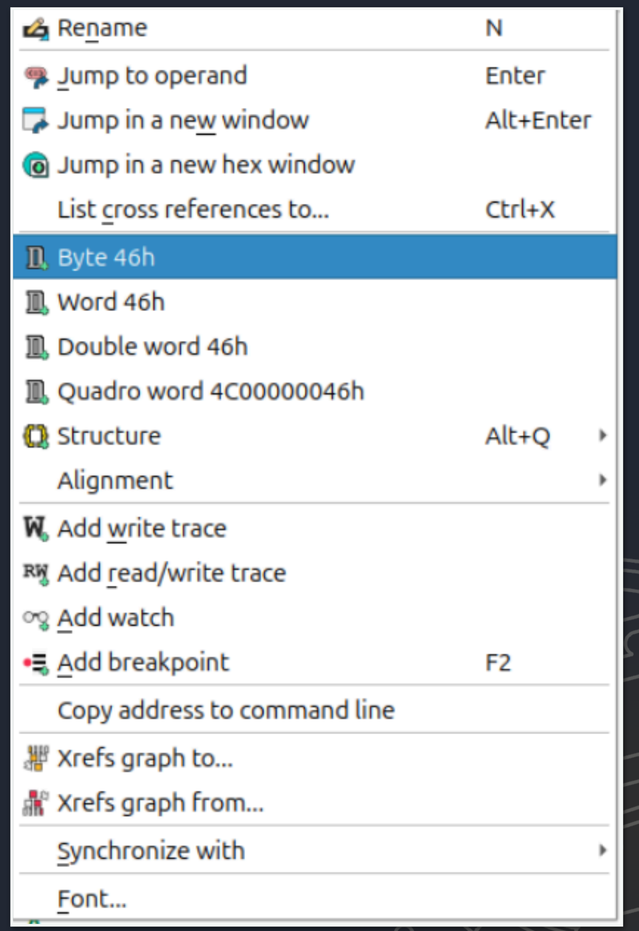
```
.rodata:0000000000002008 unk_2008         db  46h ; F             ; DATA XREF: .data:0000000000004010↓o
```

Mark Data

- 如果你看出當前某塊記憶體是什麼 Type
- 框選後 [D] 成資料
  - 按的次數會影響到標記出來的 Data 大小
  - db / dw / dd / dq 分別為 1 / 2 / 4 / 8 Bytes

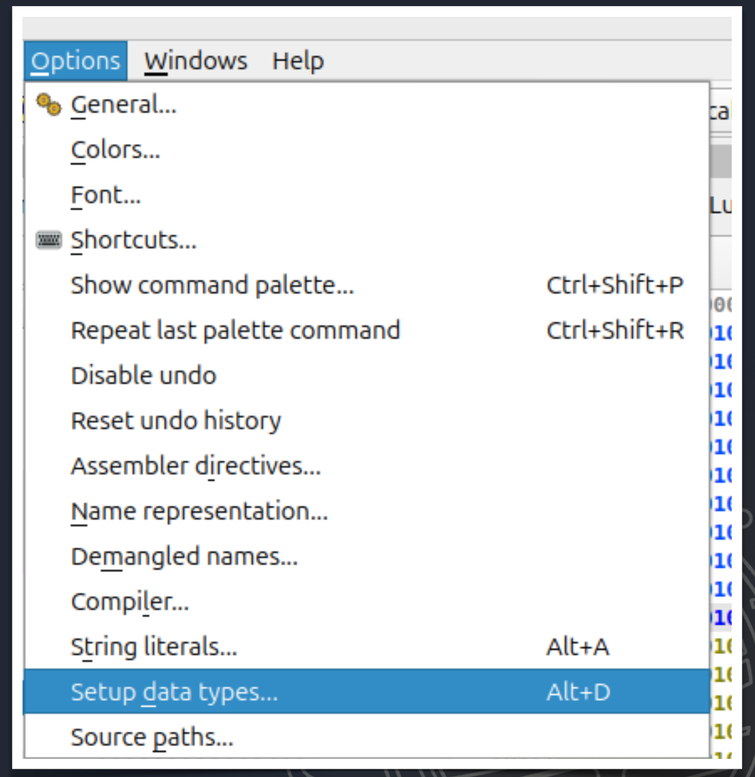
Mark Data

- 也可以右鍵選

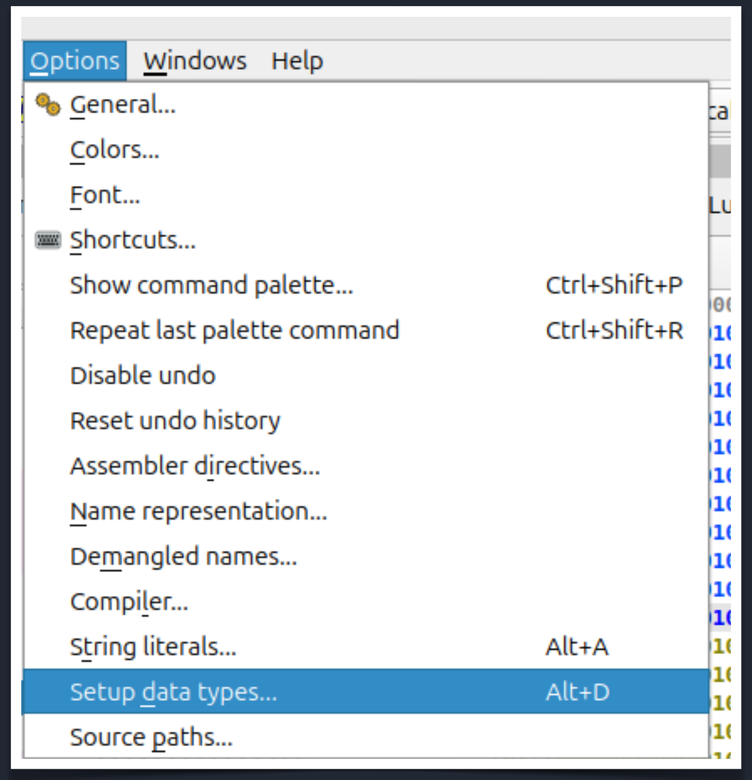


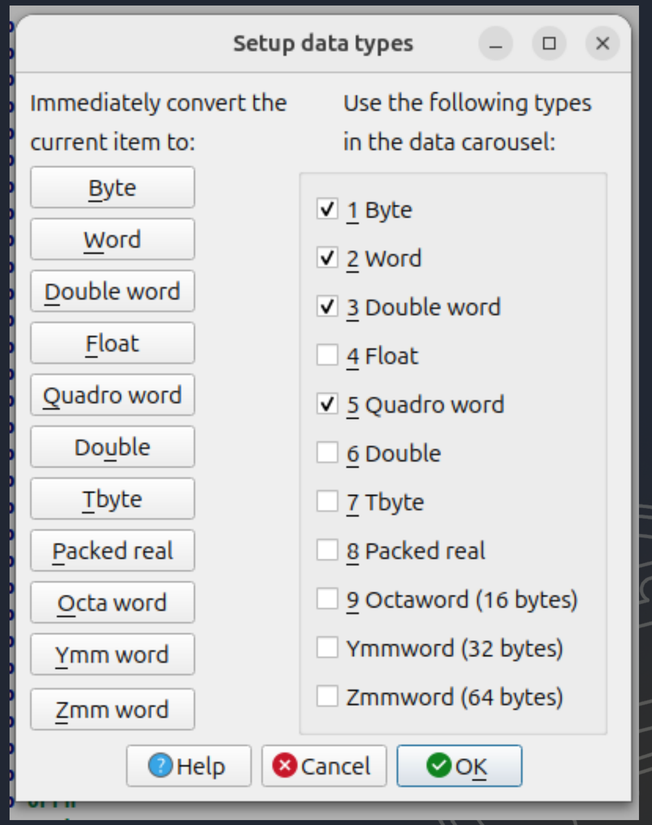
Mark Data - More Types

- 因為 Register 不只有到 64bit
  - SSE, AVX 之類的可以到 512 bit
- 這些不能在右鍵選單選出來
  - [Alt+D]
  - Options > Setup data types

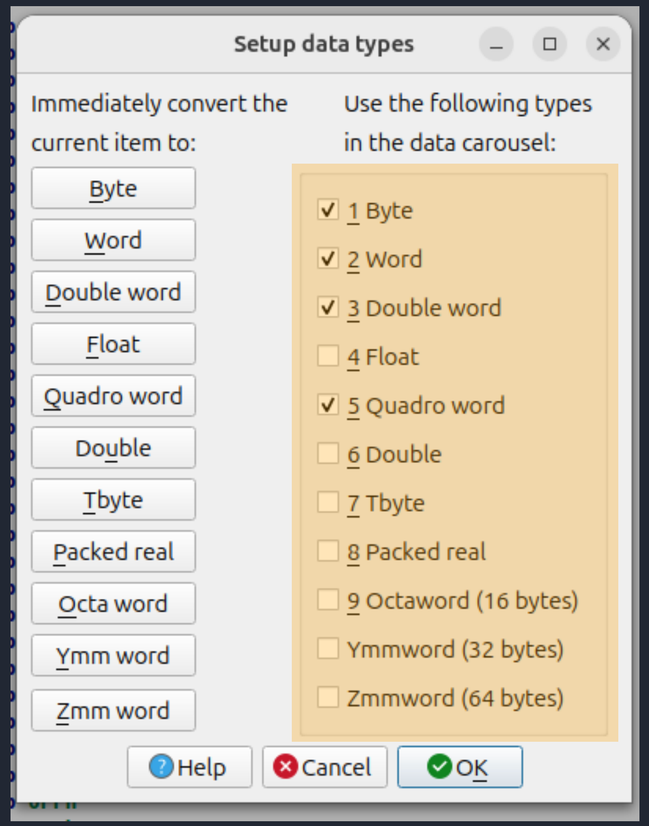


Mark Data - More Types





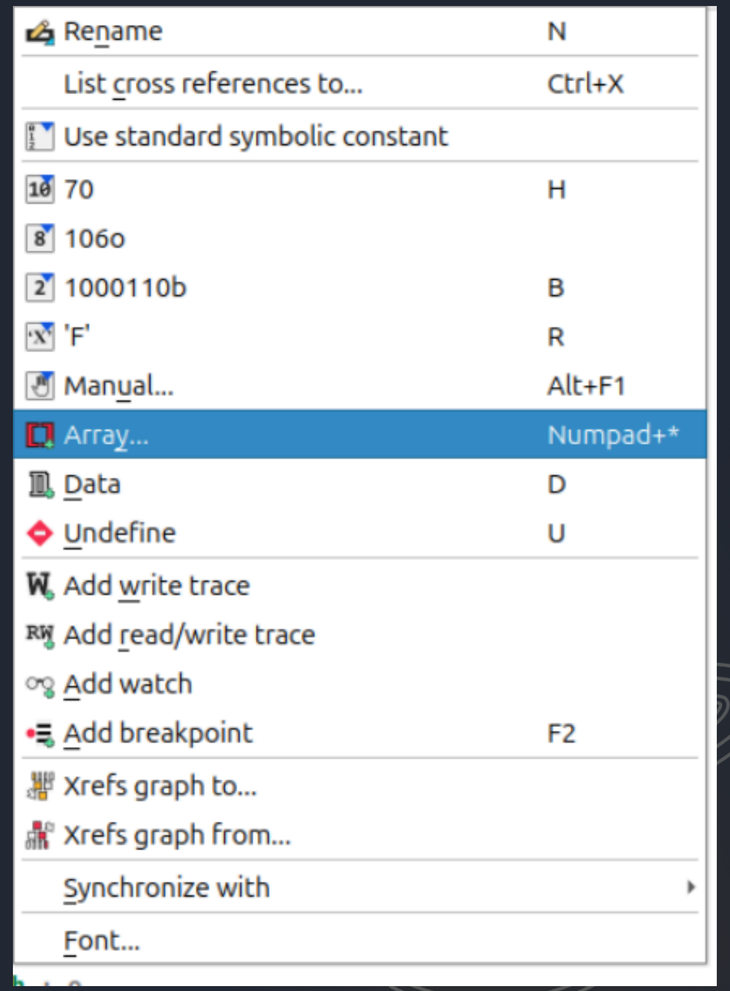
Mark Data - More Types



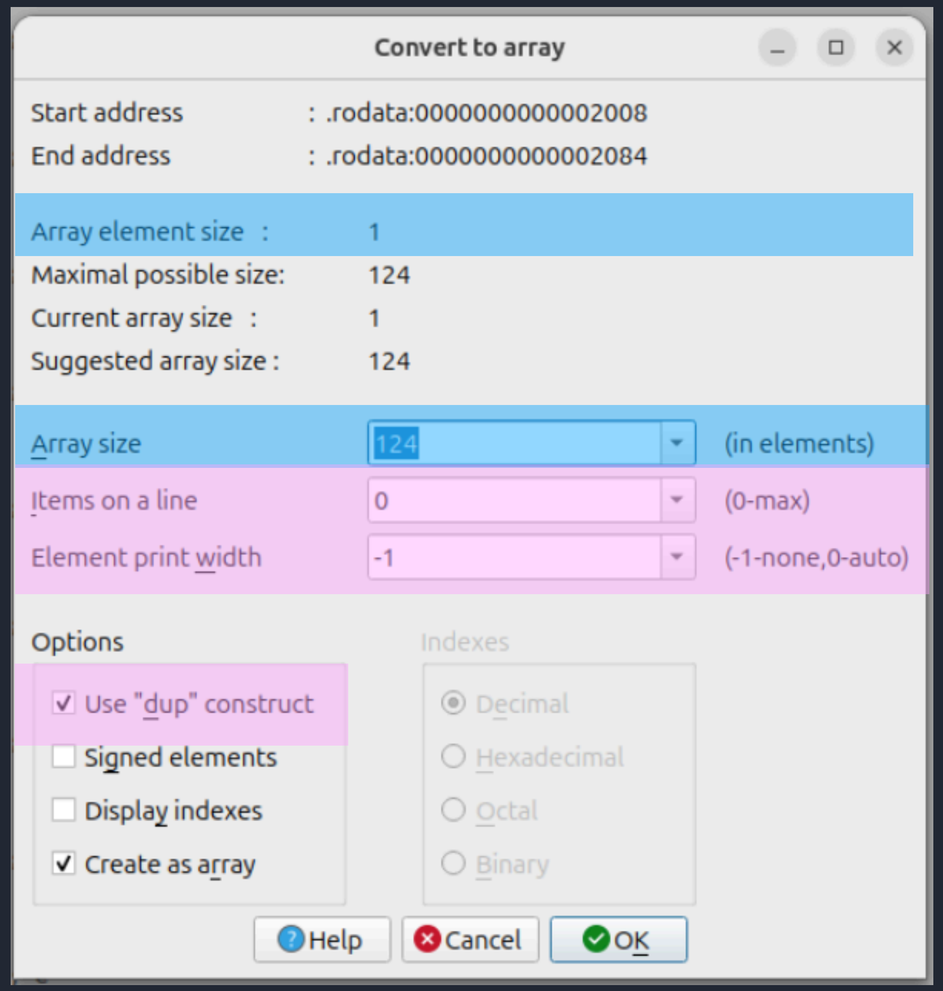
這邊勾起來的，以後就可以直接 [D] 出來

Mark Data - Array

- 標記成 Array
  - 先設定好第一個 Element 的大小
  - 按 [*] 或是右鍵 “Array”



Mark Data - Array



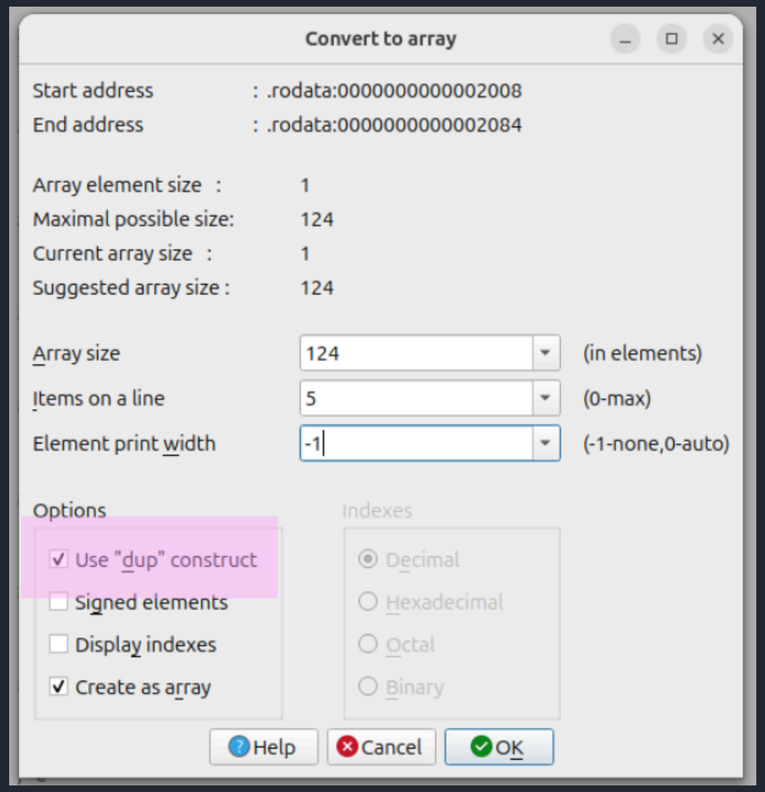
每個 Element 的大小

Array 多大

設定顯示上的細節

用 dup 來替換掉重複的元素
(也是顯示細節)

Mark Data - Array

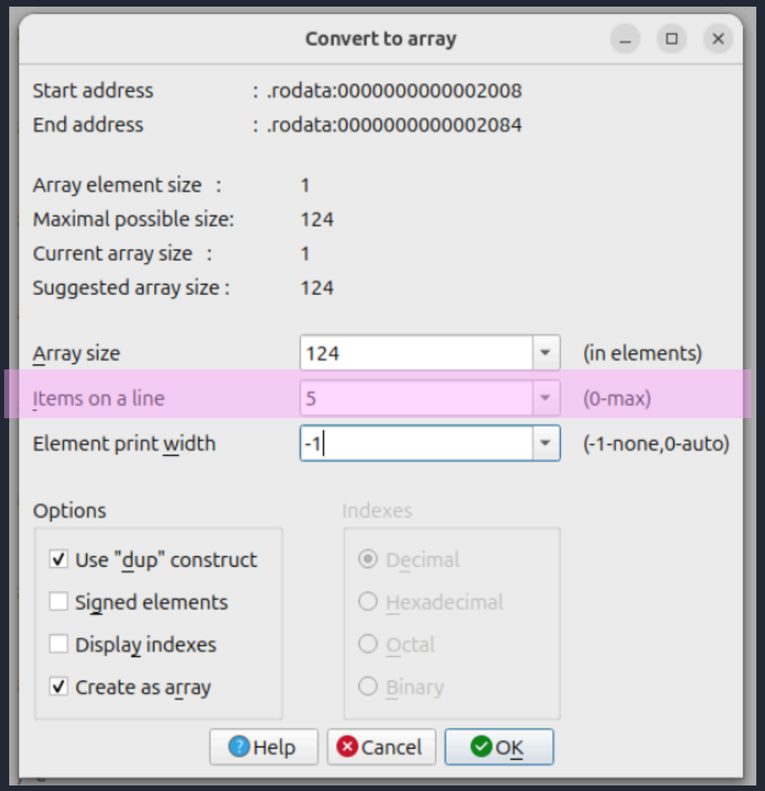


重複的元素會變成 dup 折疊

3 dup(0) = 0, 0, 0

```
.rodata:0000000000002008 byte_2008      db 46h, 3 dup(0), 4Ch, 3 dup(0), 41h
.rodata:0000000000002008                                  ; DATA XREF: .data:0000000000004010↓o
.rodata:0000000000002011                db 3 dup(0), 47h, 3 dup(0), 7Bh, 3 dup(0)
.rodata:000000000000201C                db 72h, 3 dup(0), 65h, 3 dup(0), 63h
.rodata:0000000000002025                db 3 dup(0), 6Fh, 3 dup(0), 76h, 3 dup(0)
.rodata:0000000000002030                db 65h, 3 dup(0), 72h, 3 dup(0), 65h
.rodata:0000000000002039                db 3 dup(0), 64h, 3 dup(0), 5Fh, 3 dup(0)
.rodata:0000000000002044                db 73h, 3 dup(0), 74h, 3 dup(0), 72h
.rodata:000000000000204D                db 3 dup(0), 69h, 3 dup(0), 6Eh, 3 dup(0)
.rodata:0000000000002058                db 67h, 3 dup(0), 5Fh, 3 dup(0), 6Ch
.rodata:0000000000002061                db 3 dup(0), 69h, 3 dup(0), 74h, 3 dup(0)
.rodata:000000000000206C                db 65h, 3 dup(0), 72h, 3 dup(0), 61h
.rodata:0000000000002075                db 3 dup(0), 6Ch, 3 dup(0), 7Dh, 7 dup(0)
```

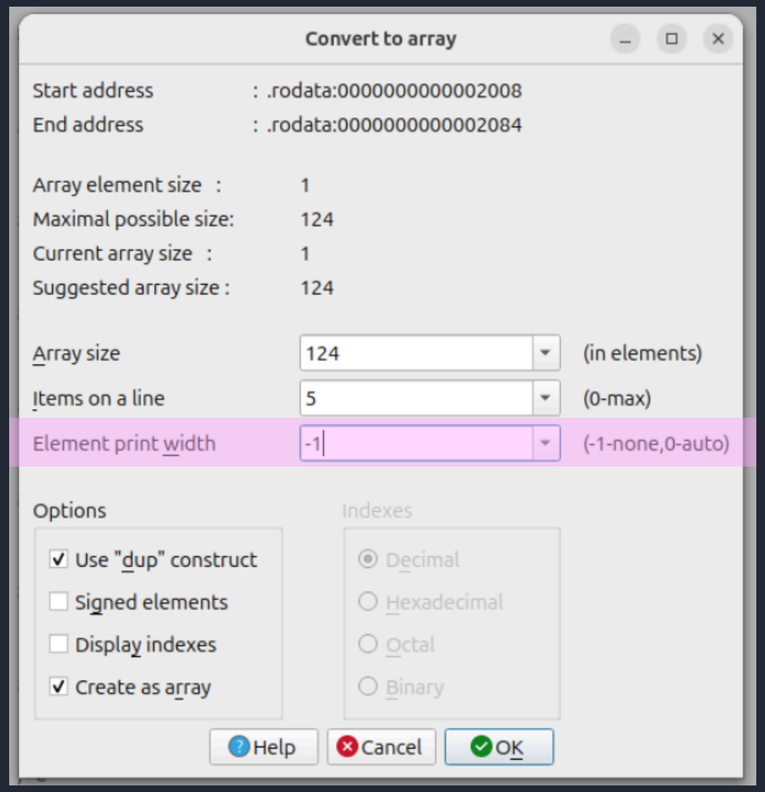
Mark Data - Array



5 個元素

```
.rodata:0000000000002008 byte_2008      db 46h, 3 dup(0), 4Ch, 3 dup(0), 41h
.rodata:0000000000002008                                        ; DATA XREF: .data:0000000000004010↓o
.rodata:0000000000002011                db 3 dup(0), 47h, 3 dup(0), 7Bh, 3 dup(0)
.rodata:000000000000201C                db 72h, 3 dup(0), 65h, 3 dup(0), 63h
.rodata:0000000000002025                db 3 dup(0), 6Fh, 3 dup(0), 76h, 3 dup(0)
.rodata:0000000000002030                db 65h, 3 dup(0), 72h, 3 dup(0), 65h
.rodata:0000000000002039                db 3 dup(0), 64h, 3 dup(0), 5Fh, 3 dup(0)
.rodata:0000000000002044                db 73h, 3 dup(0), 74h, 3 dup(0), 72h
.rodata:000000000000204D                db 3 dup(0), 69h, 3 dup(0), 6Eh, 3 dup(0)
.rodata:0000000000002058                db 67h, 3 dup(0), 5Fh, 3 dup(0), 6Ch
.rodata:0000000000002061                db 3 dup(0), 69h, 3 dup(0), 74h, 3 dup(0)
.rodata:000000000000206C                db 65h, 3 dup(0), 72h, 3 dup(0), 61h
.rodata:0000000000002075                db 3 dup(0), 6Ch, 3 dup(0), 7Dh, 7 dup(0)
```

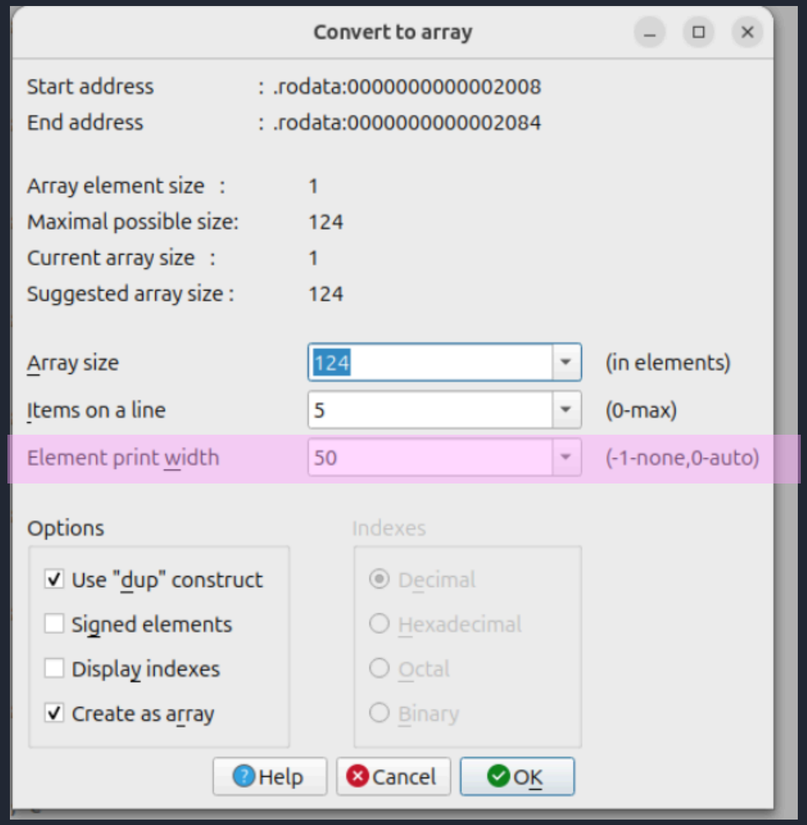
Mark Data - Array



顯示對齊 (None)

```
.rodata:0000000000002008 byte_2008      db 46h, 3 dup(0), 4Ch, 3 dup(0), 41h
.rodata:0000000000002008                                  ; DATA XREF: .data:0000000000004010↓o
.rodata:0000000000002011                db 3 dup(0), 47h, 3 dup(0), 7Bh, 3 dup(0)
.rodata:000000000000201C                db 72h, 3 dup(0), 65h, 3 dup(0), 63h
.rodata:0000000000002025                db 3 dup(0), 6Fh, 3 dup(0), 76h, 3 dup(0)
.rodata:0000000000002030                db 65h, 3 dup(0), 72h, 3 dup(0), 65h
.rodata:0000000000002039                db 3 dup(0), 64h, 3 dup(0), 5Fh, 3 dup(0)
.rodata:0000000000002044                db 73h, 3 dup(0), 74h, 3 dup(0), 72h
.rodata:000000000000204D                db 3 dup(0), 69h, 3 dup(0), 6Eh, 3 dup(0)
.rodata:0000000000002058                db 67h, 3 dup(0), 5Fh, 3 dup(0), 6Ch
.rodata:0000000000002061                db 3 dup(0), 69h, 3 dup(0), 74h, 3 dup(0)
.rodata:000000000000206C                db 65h, 3 dup(0), 72h, 3 dup(0), 61h
.rodata:0000000000002075                db 3 dup(0), 6Ch, 3 dup(0), 7Dh, 7 dup(0)
```

Mark Data - Array



顯示對齊 (50)

```
db                                      46h,3 dup(                      0),                     4Ch,3 dup(
                                ; DATA XREF: .data:0000000000004010↓o
db 3 dup(                               0),                     47h,3 dup(                      0),
db                                      72h,3 dup(                      0),                     65h,3 dup(
db 3 dup(                               0),                     6Fh,3 dup(                      0),
db                                      65h,3 dup(                      0),                     72h,3 dup(
db 3 dup(                               0),                     64h,3 dup(                      0),
db                                      73h,3 dup(                      0),                     74h,3 dup(
db 3 dup(                               0),                     69h,3 dup(                      0),
db                                      67h,3 dup(                      0),                     5Fh,3 dup(
db 3 dup(                               0),                     69h,3 dup(                      0),
db                                      65h,3 dup(                      0),                     72h,3 dup(
db 3 dup(                               0),                     6Ch,3 dup(                      0),
```

Mark Data - String Literal

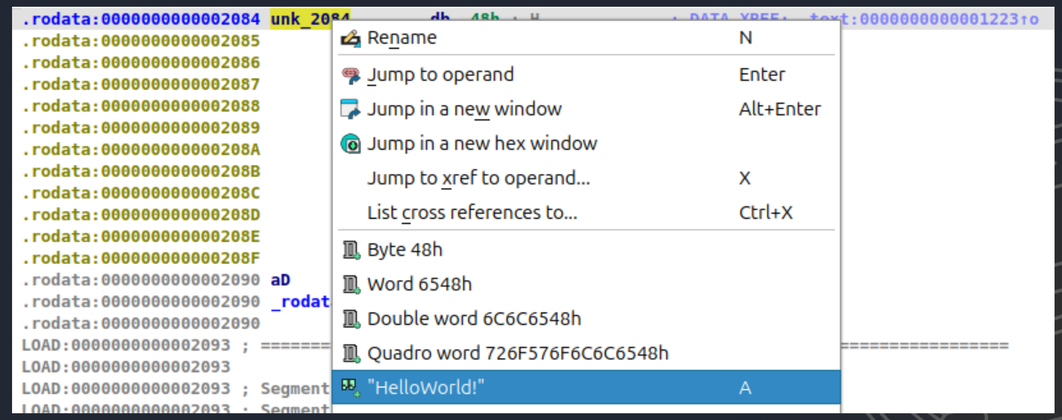
很顯然他是一個 char *

```
.rodata:0000000000002084 unk_2084        db  48h ; H             ; DATA XREF: .text:0000000000001223↑o
.rodata:0000000000002085                 db  65h ; e
.rodata:0000000000002086                 db  6Ch ; l
.rodata:0000000000002087                 db  6Ch ; l
.rodata:0000000000002088                 db  6Fh ; o
.rodata:0000000000002089                 db  57h ; W
.rodata:000000000000208A                 db  6Fh ; o
.rodata:000000000000208B                 db  72h ; r
.rodata:000000000000208C                 db  6Ch ; l
.rodata:000000000000208D                 db  64h ; d
.rodata:000000000000208E                 db  21h ; !
.rodata:000000000000208F                 db    0
```

Mark Data - String Literal

選擇開頭後 [a]

```
.rodata:0000000000002084 unk_2084        db  48h ; H          ; DATA XREF: .text:0000000000001223↑o
.rodata:0000000000002085                 db  65h ; e
.rodata:0000000000002086                 db  6Ch ; l
.rodata:0000000000002087                 db  6Ch ; l
.rodata:0000000000002088                 db  6Fh ; o
.rodata:0000000000002089                 db  57h ; W
.rodata:000000000000208A                 db  6Fh ; o
.rodata:000000000000208B                 db  72h ; r
.rodata:000000000000208C                 db  6Ch ; l
.rodata:000000000000208D                 db  64h ; d
.rodata:000000000000208E                 db  21h ; !
.rodata:000000000000208F                 db    0
```



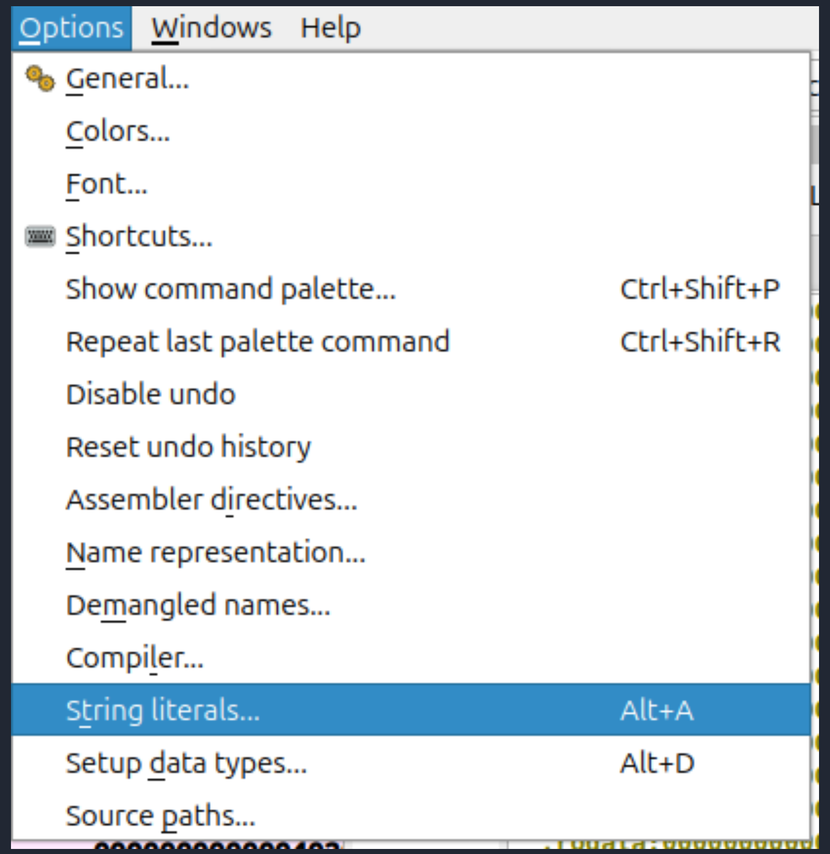
Mark Data - String Literal

自動根據 null-terminate 決定字串大小

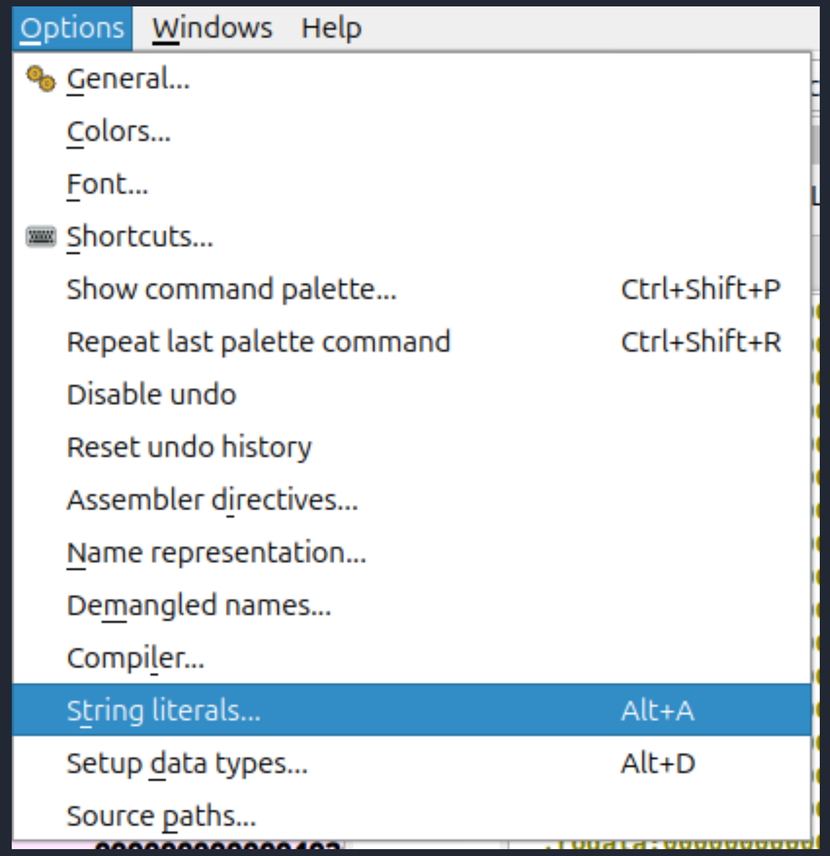
```
.rodata:0000000000002084 aHelloworld      db 'HelloWorld!',0       ; DATA XREF: .text:0000000000001223↑o
```

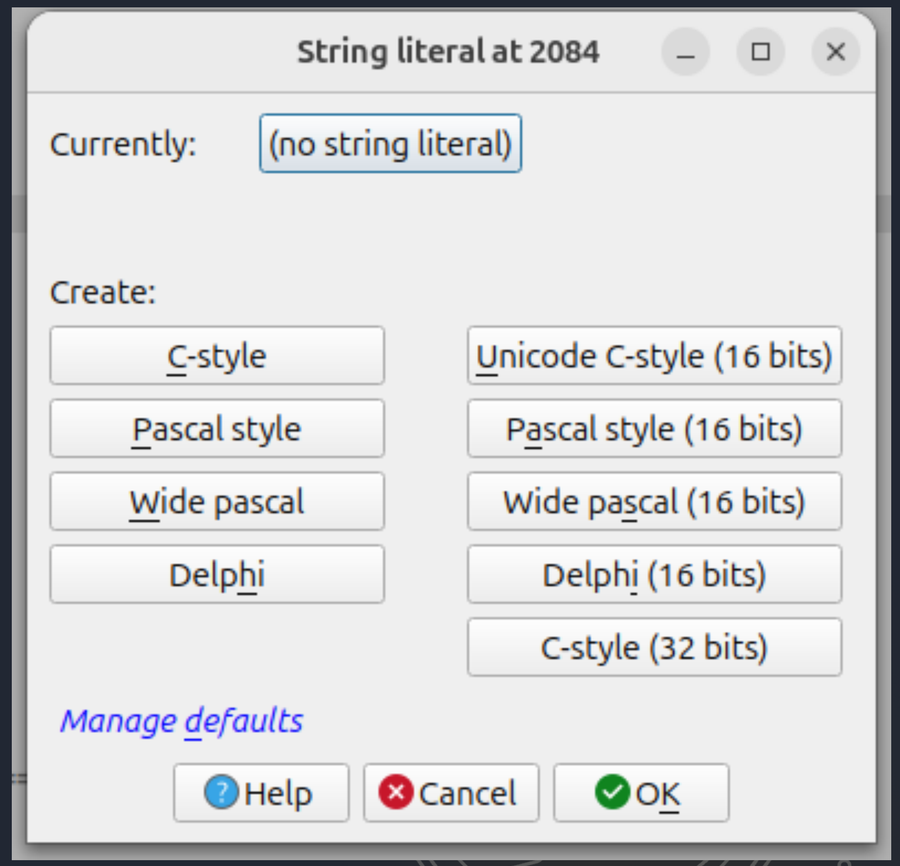
Mark Data - String Literal

如果不是 char* 而是其他種字串 (UTF16LE, UTF16BE...)



Mark Data - String Literal





Mark Data - String Literal

- Rust, Golang 這些語言的字串經常都是多個 Pack 在一塊
- IDA 預設不裝插件就幫不上忙了 QQ

Mark Enum

Mark Enum

- Decompile 的 Library Function 上經常出現神秘數字嗎？
- Enum 是你的好幫手！

Mark Enum - Example

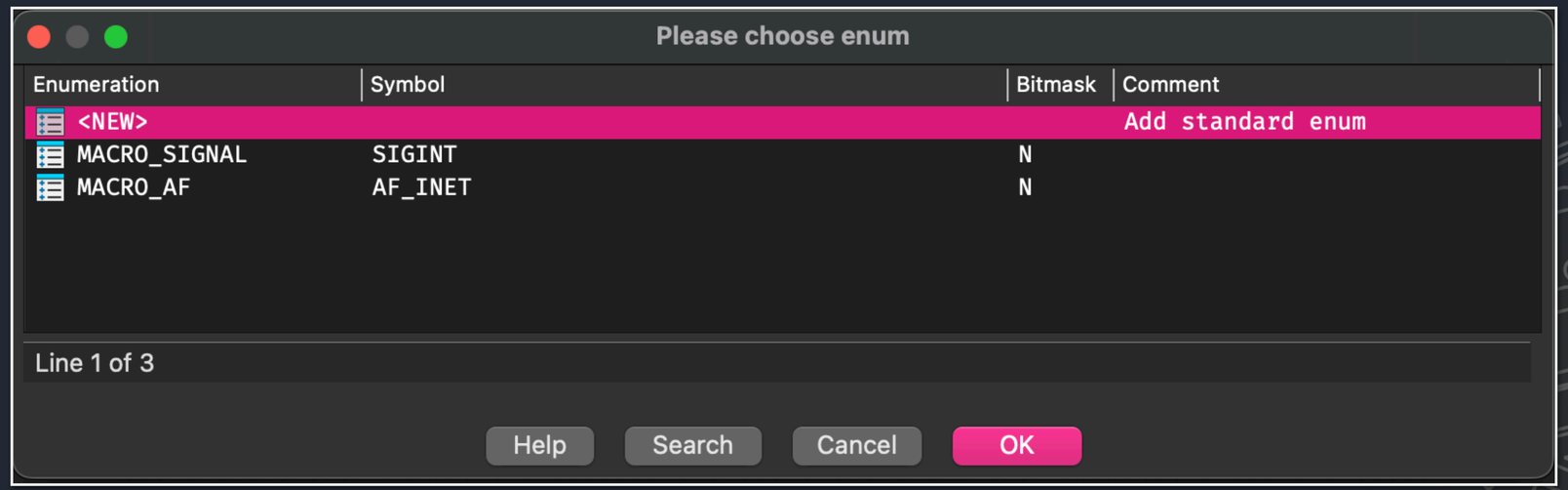
socket(Domain, Type, Protocol)

```
v0 = socket_0(1, 2, 0);
if ( v0 >= 0 )
{
```

Mark Enum - Example

```
v0 = socket_0(1, 2, 0);
if ( v0 ≥ 0 )
{
```

框起來之後 [M] 就可以選擇 Enum，如果沒出現的話可以去 NEW 裡面找



Mark Enum - Example

都做完之後就會有好看的 decompile 了

```
v0 = socket_0(AF_UNIX, SOCK_DGRAM, 0);
if ( v0 >= 0 )
{
```

Mark Enum

- 這邊舉的例子是 standard library 所以 IDA 裡面有得用
- 如果是開發者實作的 Enum，就要根據功能還原出 Enum 結構

Something F up

- 如果有東西標錯怎麼辦？
- 把做錯的地方框起來 Undefine 後重新來過
  - [U] 或是右鍵 “Undefine”

lab01.idb

lab01.idb

- 目標
  - 成功打開 IDAFree 並載入此 idb
  - 在 0x2008 有一把 Flag，修成 String literal
  - 有一個 Unreachable Function，找到並解開 Flag

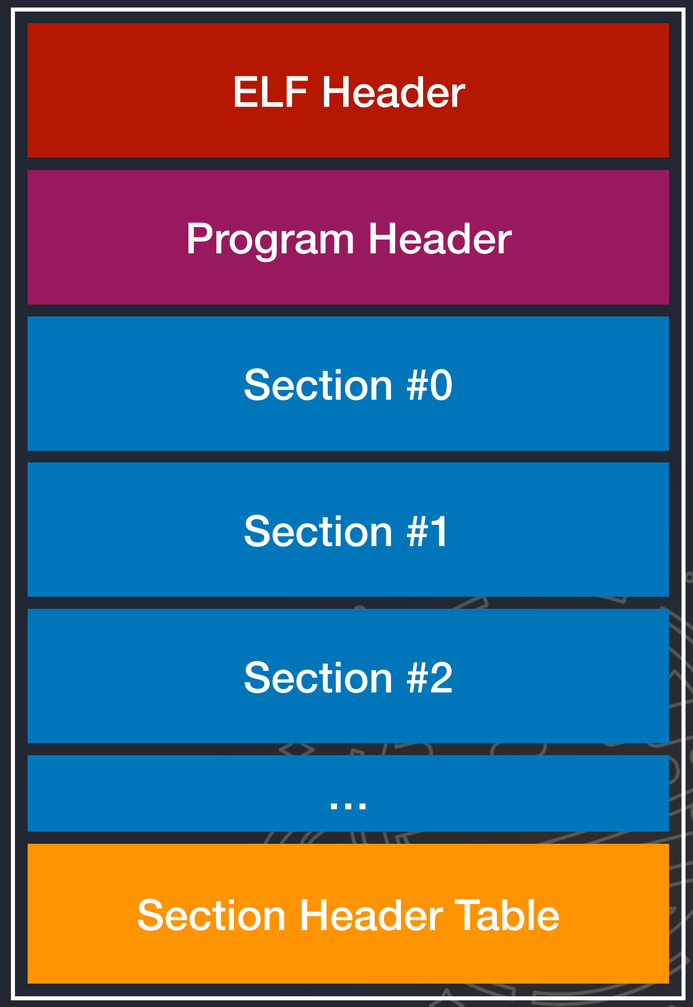
Inspect ELF Components

Executable & Linkable Format

- Linux 上的執行檔案格式
- 可以通過 readelf 來檢視 ELF 檔案的詳細內容

ELF Overview

- Major Components
  - ELF Header
  - Program Header
  - Sections
  - Section Header Table
- 我們來看看 Raw Data 怎麼 Map
  - 結構都可以在 elf.h 上面找到



```
typedef struct {
    unsigned char e_ident[EI_NIDENT];
    uint16_t      e_type;
    uint16_t      e_machine;
    uint32_t      e_version;
    ElfN_Addr     e_entry;
    ElfN_Off      e_phoff;
    ElfN_Off      e_shoff;
    uint32_t      e_flags;
    uint16_t      e_ehsize;
    uint16_t      e_phentsize;
    uint16_t      e_phnum;
    uint16_t      e_shentsize;
    uint16_t      e_shnum;
    uint16_t      e_shstrndx;
} ElfN_Ehdr;
```

ELF Header (Ehdr)

```
00000000: 7f45 4c46 0201 0100 0000 0000 0000 0000  .ELF............
00000010: 0300 3e00 0100 0000 6010 0000 0000 0000  ..>.....`.......
00000020: 4000 0000 0000 0000 a036 0000 0000 0000  @........6......
00000030: 0000 0000 4000 3800 0d00 4000 1f00 1e00  ....@.8...@.....
00000040: 0600 0000 0400 0000 4000 0000 0000 0000  ........@.......
00000050: 4000 0000 0000 0000 4000 0000 0000 0000  @.......@.......
00000060: d802 0000 0000 0000 d802 0000 0000 0000  ................
00000070: 0800 0000 0000 0000 0300 000
00000080: 1803 0000 0000 0000 1803 000
00000090: 1803 0000 0000 0000 1c00 000
000000a0: 1c00 0000 0000 0000 0100 000
000000b0: 0100 0000 0400 0000 0000 000
000000c0: 0000 0000 0000 0000 0000 000
000000d0: 2806 0000 0000 0000 2806 000
000000e0: 0010 0000 0000 0000 0100 000
000000f0: 0010 0000 0000 0000 0010 000
00000100: 0010 0000 0000 0000 8101 000
```

```
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              DYN (Position-Independent Executable file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x1060
  Start of program headers:          64 (bytes into file)
  Start of section headers:          13984 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           56 (bytes)
  Number of program headers:         13
  Size of section headers:           64 (bytes)
  Number of section headers:         31
  Section header string table index: 30
```

ELF Header (Ehdr)

```
00000000: 7f45 4c46 0201 0100 0000 0000 0000 0000  .ELF............
00000010: 0300 3e00 0100 0000 6010 0000 0000 0000  ..>.....`.......
00000020: 4000 0000 0000 0000 a036 0000 0000 0000  @........6......
00000030: 0000 0000 40
00000040: 0600 0000 04
00000050: 4000 0000 00
00000060: d802 0000 00
00000070: 0800 0000 00
00000080: 1803 0000 00
00000090: 1803 0000 00
000000a0: 1c00 0000 00
000000b0: 0100 0000 04
000000c0: 0000 0000 00
000000d0: 2806 0000 00
000000e0: 0010 0000 00
000000f0: 0010 0000 00
00000100: 0010 0000 00
```


```
00 00 00

t, little endian

V

-Independent Executable file)
o Devices X86-64

o file)
into file)

Size of this header:                64 (bytes)
Size of program headers:            56 (bytes)
Number of program headers:          13
Size of section headers:            64 (bytes)
Number of section headers:          31
Section header string table index: 30
```

ELF Header (Ehdr)

```
00000000: 7f45 4c46 0201 0100 0000 0000 0000 0000  .ELF............
00000010: 0300 3e00 0100 0000 6010 0000 0000 0000  ..>.....`.......
00000020: 4000 0000 0000 0000 a036 0000 0000 0000  @........6......
00000030: 0000 0000 0000 0000 0d00 4000 1f00 1e00  ....@.8...@.....
00000040: 0600 0000 0400 0000 4000 0000 0000 0000  ....@.8...@.....
00000050: 4000 0000 0000 0000 4000 0000 0000 0000  @.8...@.....
00000060: d802 0000 0000 0000 d802 0000 0000 0000  ................
00000070: 0800 0000 0000 0000 0300 0000 0400 0000  ................
00000080: 1803 0000 0000 0000 1803 0000 0000 0000  ................
00000090: 1803 0000 0000 0000 1c00 0000 0000 0000  ................
000000a0: 1c00 0000 0000 0000 0100 0000 0400 0000  ................
000000b0: 0100 0000 0400 0000 0000 0000 0000 0000  ................
000000c0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000d0: 2806 0000 0000 0000 2806 0000 0000 0000  (.......(.......
000000e0: 0010 0000 0000 0000 0100 0000 0500 0000  ................
000000f0: 0010 0000 0000 0000 0010 0000 0000 0000  ................
00000100: 0010 0000 0000 0000 8101 0000 0000 0000  ................
```

E_IDENT

這是一個 Array 會告訴解析器怎麼分析這個 ELF 檔案

```
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              DYN (Position-Independent Executable file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x1060
  Start of program headers:          64 (bytes into file)
  Start of section headers:          13984 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           56 (bytes)
  Number of program headers:         13
  Size of section headers:           64 (bytes)
  Number of section headers:         31
  Section header string table index: 30
```

ELF Header (Ehdr)

```
00000000: 7f45 4c46 0201 0100 0000 0000 0000 0000  .ELF............
00000010: 0300 3e00 0100 0000 6010 0000 0000 0000  ..>.....`.......
00000020: 4000 0000 0000 0000 a036 0000 0000 0000  @........6......
00000030: 0000 0(                                  @.
00000040: 0600 0(                                  ..
00000050: 4000 0(
00000060: d802 0(
00000070: 0800 0(
00000080: 1803 0000 0000 0000 1803 000
00000090: 1803 0000 0000 0000 1c00 000
000000a0: 1c00 0000 0000 0000 0100 000
000000b0: 0100 0000 0400 0000 0000 000
000000c0: 0000 0000 0000 0000 0000 000
000000d0: 2806 0000 0000 0000 2806 000
000000e0: 0010 0000 0000 0000 0100 000
000000f0: 0010 0000 0000 0000 0010 000
00000100: 0010 0000 0000 0000 8101 000
```

EI_MAG0 **EI_MAG1** **EI_MAG2** **EI_MAG3**

在 memdump 中如果看到這幾個 byte 就知道是 ELF 了

```
Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
Class:                             ELF64
Data:                              2's complement, little endian
Version:                           1 (current)
OS/ABI:                            UNIX - System V
ABI Version:                       0
Type:                              DYN (Position-Independent Executable file)
Machine:                           Advanced Micro Devices X86-64
Version:                           0x1
Entry point address:               0x1060
Start of program headers:          64 (bytes into file)
Start of section headers:          13984 (bytes into file)
Flags:                             0x0
Size of this header:               64 (bytes)
Size of program headers:           56 (bytes)
Number of program headers:         13
Size of section headers:           64 (bytes)
Number of section headers:         31
Section header string table index: 30
```

ELF Header (Ehdr)

```
00000000: 7f45 4c46 0201 0100 0000 0000 0000 0000  .ELF............
00000010: 0300 3e00 0100 0000 6010 0000 0000 0000  ..>.....`.......
00000020: 4000 0000 0000 0000 a036 0000 0000 0000  @........6......
00000030: 0000 0000 4[illegible]00 4000 1f00 1e00  ....@.8...@.....
00000040: 0600 0000 0[illegible]                   
00000050: 4000 0000 0[illegible]                   
00000060: d802 0000 0[illegible]                   
00000070: 0800 0000 0000 0000 0300 000[illegible]
00000080: 1803 0000 0000 0000 1803 000[illegible]
00000090: 1803 0000 0000 0000 1c00 000[illegible]
000000a0: 1c00 0000 0000 0000 0100 000[illegible]
000000b0: 0100 0000 0400 0000 0000 000[illegible]
000000c0: 0000 0000 0000 0000 0000 000[illegible]
000000d0: 2806 0000 0000 0000 2806 000[illegible]
000000e0: 0010 0000 0000 0000 0100 000[illegible]
000000f0: 0010 0000 0000 0000 0010 000[illegible]
00000100: 0010 0000 0000 0000 8101 000[illegible]
```

E_TYPE

ELF 的種類 ET_NONE, ET_REL, ET_DYN, ET_CORE

```
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              DYN (Position-Independent Executable file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x1060
  Start of program headers:          64 (bytes into file)
  Start of section headers:          13984 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           56 (bytes)
  Number of program headers:         13
  Size of section headers:           64 (bytes)
  Number of section headers:         31
  Section header string table index: 30
```

ELF Header (Ehdr)

```
00000000: 7f45 4c46 0201 0100 0000 0000 0000 0000  .ELF............
00000010: 0300 3e00 0100 0000 6010 0000 0000 0000  ..>.....`.......
00000020: 4000 0000 0000 0000 a036 0000 0000 0000  @........6......
00000030: 0000 0000 4000 3800 0d00 4000 1f00 1e00  ....@.8...@.....
00000040: 0600 0000 0400 0000 4000 0000 0000 0000  ........@.......
00000050: 4000 0000 0000 0000 4000 0000 0000 0000  @.......@.......
00000060: d802 0000 0000 0000 d802 0000 0000 0000  ................
00000070: 0800 0000 0000 0000 0300 000
00000080: 1803 0000 0000 0000 1803 000
00000090: 1803 0000 0000 0000 1c00 000
000000a0: 1c00 0000 0000 0000 0100 000
000000b0: 0100 0000 0400 0000 0000 000
000000c0: 0000 0000 0000 0000 0000 000
000000d0: 2806 0000 0000 0000 2806 000
000000e0: 0010 0000 0000 0000 0100 000
000000f0: 0010 0000 0000 0000 0010 000
00000100: 0010 0000 0000 0000 8101 000
```

E_ENTRY

程式進入點位址 (Entrypoint)

```
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              DYN (Position-Independent Executable file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x1060
  Start of program headers:          64 (bytes into file)
  Start of section headers:          13984 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           56 (bytes)
  Number of program headers:         13
  Size of section headers:           64 (bytes)
  Number of section headers:         31
  Section header string table index: 30
```

ELF Header (Ehdr)

```
00000000: 7f45 4c46 0201 0100 0000 0000 0000 0000  .ELF............
00000010: 0300 3e00 0100 0000 6010 0000 0000 0000  ..>.....`.......
00000020: 4000 0000 0000 0000 a036 0000 0000 0000  @........6......
00000030: 0000 0000 4000 3800 0d00 4000 1f00 1e00  ....@.8...@.....
00000040: 0600 0000 0400 0000 4000 0000 0000 0000  ........@.......
00000050: 4000 0000 0000 0000 4000 0000 0000 0000  @.......@.......
00000060: 0000 0000 d802 0000 0000 0000 0000 0000  ................
00000070:
00000080:
00000090: 1803 0000 0000 0000 1c00 000
000000a0: 1c00 0000 0000 0000 0100 000
000000b0: 0100 0000 0400 0000 0000 000
000000c0: 0000 0000 0000 0000 0000 000
000000d0: 2806 0000 0000 0000 2806 000
000000e0: 0010 0000 0000 0000 0100 000
000000f0: 0010 0000 0000 0000 0010 000
00000100: 0010 0000 0000 0000 8101 000
```

E_PHOFF

Program Header 陣列的起點

```
c 46 02 01 01 00 00 00 00 00 00 00 00 00
Class:                             ELF64
Data:                              2's complement, little endian
Version:                           1 (current)
OS/ABI:                            UNIX - System V
ABI Version:                       0
Type:                              DYN (Position-Independent Executable file)
Machine:                           Advanced Micro Devices X86-64
Version:                           0x1
Entry point address:               0x1060
Start of program headers:          64 (bytes into file)
Start of section headers:          13984 (bytes into file)
Flags:                             0x0
Size of this header:               64 (bytes)
Size of program headers:           56 (bytes)
Number of program headers:         13
Size of section headers:           64 (bytes)
Number of section headers:         31
Section header string table index: 30
```

ELF Header (Ehdr)

```
00000000: 7f45 4c46 0201 0100 0000 0000 0000 0000  .ELF............
00000010: 0300 3e00 0100 0000 6010 0000 0000 0000  ..>.....`.......
00000020: 4000 0000 0000 0000 a036 0000 0000 0000  @........6......
00000030: 0000 0000 4000 3800 0d00 4000 1f00 1e00  ....@.8...@.....
00000040: 0600 0000 0400 0000 4000 0000 0000 0000  ........@.......
00000050: 4000 0000 0000 0000 4000 0000 0000 0000  @.......@.......
00000060: 0803 0000 0000 0000 0803 0000 0000 0000  ................
00000070:
00000080:
00000090: 1803 0000 0000 0000 1c00 000
000000a0: 1c00 0000 0000 0000 0100 000
000000b0: 0100 0000 0400 0000 0000 000
000000c0: 0000 0000 0000 0000 0000 000
000000d0: 2806 0000 0000 0000 2806 000
000000e0: 0010 0000 0000 0000 0100 000
000000f0: 0010 0000 0000 0000 0010 000
00000100: 0010 0000 0000 0000 8101 000
```

E_PHENTSIZE

每個 Program Header 多大

```
5 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
Class:                             ELF64
Data:                              2's complement, little endian
Version:                           1 (current)
OS/ABI:                            UNIX - System V
ABI Version:                       0
Type:                              DYN (Position-Independent Executable file)
Machine:                           Advanced Micro Devices X86-64
Version:                           0x1
Entry point address:               0x1060
Start of program headers:          64 (bytes into file)
Start of section headers:          13984 (bytes into file)
Flags:                             0x0
Size of this header:               64 (bytes)
Size of program headers:           56 (bytes)
Number of program headers:         13
Size of section headers:           64 (bytes)
Number of section headers:         31
Section header string table index: 30
```

ELF Header (Ehdr)

```
00000000: 7f45 4c46 0201 0100 0000 0000 0000 0000  .ELF............
00000010: 0300 3e00 0100 0000 6010 0000 0000 0000  ..>.....`.......
00000020: 4000 0000 0000 0000 a036 0000 0000 0000  @........6......
00000030: 0000 0000 4000 3800 0d00 4000 1f00 1e00  ....@.8...@.....
00000040: 0600 0000 0400 0000 4000 0000 0000 0000  ........@.......
00000050: 0000 0000 0000 0000 4000 0000 0000 0000  @.......@.......
00000060: 0000 0000 d802 0000 0000 0000 0000 0000  ................
00000070:
00000080:
00000090: 1803 0000 0000 0000 1c00 000
000000a0: 1c00 0000 0000 0000 0100 000
000000b0: 0100 0000 0400 0000 0000 000
000000c0: 0000 0000 0000 0000 0000 000
000000d0: 2806 0000 0000 0000 2806 000
000000e0: 0010 0000 0000 0000 0100 000
000000f0: 0010 0000 0000 0000 0010 000
00000100: 0010 0000 0000 0000 8101 000
```

E_PHNUM

有幾個 Program Header

```
$ readelf -h ./ehdr
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              DYN (Position-Independent Executable file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x1060
  Start of program headers:          64 (bytes into file)
  Start of section headers:          13984 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           56 (bytes)
  Number of program headers:         13
  Size of section headers:           64 (bytes)
  Number of section headers:         31
  Section header string table index: 30
```

ELF Header (Ehdr)

```
00000000: 7f45 4c46 0201 0100 0000 0000 0000 0000  .ELF............
00000010: 0300 3e00 0100 0000 6010 0000 0000 0000  ..>.....`.......
00000020: 4000 0000 0000 0000 a036 0000 0000 0000  @........6......
00000030: 0000 0000 4000 3800 0d00 4000 1f00 1e00  ....@.8...@.....
00000040: 0600 0000 0400 0000 4000 0000 0000 0000  ........@.......
00000050: 0000 0000 0000 0000 4000 0000 0000 0000  @.......@.......
00000060: 0000 0000 d802 0000 0000 0000 0000 0000  ................
00000070:
00000080:
00000090: 1803 0000 0000 0000 1c00 000
000000a0: 1c00 0000 0000 0000 0100 000
000000b0: 0100 0000 0400 0000 0000 000
000000c0: 0000 0000 0000 0000 0000 000
000000d0: 2806 0000 0000 0000 2806 000
000000e0: 0010 0000 0000 0000 0100 000
000000f0: 0010 0000 0000 0000 0010 000
00000100: 0010 0000 0000 0000 8101 000
```

E_SHOFF

Section Header 的起點

```
er:
    7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
lass:                              ELF64
Data:                              2's complement, little endian
Version:                           1 (current)
OS/ABI:                            UNIX - System V
ABI Version:                       0
Type:                              DYN (Position-Independent Executable file)
Machine:                           Advanced Micro Devices X86-64
Version:                           0x1
Entry point address:               0x1060
Start of program headers:          64 (bytes into file)
Start of section headers:          13984 (bytes into file)
Flags:                             0x0
Size of this header:               64 (bytes)
Size of program headers:           56 (bytes)
Number of program headers:         13
Size of section headers:           64 (bytes)
Number of section headers:         31
Section header string table index: 30
```

ELF Header (Ehdr)

```
00000000: 7f45 4c46 0201 0100 0000 0000 0000 0000  .ELF............
00000010: 0300 3e00 0100 0000 6010 0000 0000 0000  ..>.....`.......
00000020: 4000 0000 0000 0000 a036 0000 0000 0000  @........6......
00000030: 0000 0000 4000 3800 0d00 4000 1f00 1e00  ....@.8...@.....
00000040: 0600 0000 0400 0000 4000 0000 0000 0000  ........@.......
00000050: 4000 0000 0000 0000 4000 0000 0000 0000  @.......@.......
00000060: [illegible]                              ................
00000070:
00000080:
00000090: 1803 0000 0000 0000 1c00 000
000000a0: 1c00 0000 0000 0000 0100 000
000000b0: 0100 0000 0400 0000 0000 000
000000c0: 0000 0000 0000 0000 0000 000
000000d0: 2806 0000 0000 0000 2806 000
000000e0: 0010 0000 0000 0000 0100 000
000000f0: 0010 0000 0000 0000 0010 000
00000100: 0010 0000 0000 0000 8101 000
```

E_SHENTSIZE

每個 Section Header 多大

```
45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00 00
Class:                             ELF64
Data:                              2's complement, little endian
Version:                           1 (current)
OS/ABI:                            UNIX - System V
ABI Version:                       0
Type:                              DYN (Position-Independent Executable file)
Machine:                           Advanced Micro Devices X86-64
Version:                           0x1
Entry point address:               0x1060
Start of program headers:          64 (bytes into file)
Start of section headers:          13984 (bytes into file)
Flags:                             0x0
Size of this header:               64 (bytes)
Size of program headers:           56 (bytes)
Number of program headers:         13
Size of section headers:           64 (bytes)
Number of section headers:         31
Section header string table index: 30
```

ELF Header (Ehdr)

```
00000000: 7f45 4c46 0201 0100 0000 0000 0000 0000  .ELF............
00000010: 0300 3e00 0100 0000 6010 0000 0000 0000  ..>.....`.......
00000020: 4000 0000 0000 0000 a036 0000 0000 0000  @........6......
00000030: 0000 0000 4000 3800 0d00 4000 1f00 1e00  ....@.8...@.....
00000040: 0600 0000 0400 0000 4000 0000 0000 0000  ........@.......
00000050: 0000 0000 0000 0000 4000 0000 0000 0000  @.......@.......
00000060: 0000 0000 d802 0000 0000 0000 0000 0000  ................
00000070:
00000080:
00000090: 1803 0000 0000 0000 1c00 000
000000a0: 1c00 0000 0000 0000 0100 000
000000b0: 0100 0000 0400 0000 0000 000
000000c0: 0000 0000 0000 0000 0000 000
000000d0: 2806 0000 0000 0000 2806 000
000000e0: 0010 0000 0000 0000 0100 000
000000f0: 0010 0000 0000 0000 0010 000
00000100: 0010 0000 0000 0000 8101 000
```

E_SHNUM

Section Header 數量

```
 Header:
 gic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
 lass:                              ELF64
 Data:                              2's complement, little endian
 Version:                           1 (current)
 OS/ABI:                            UNIX - System V
 ABI Version:                       0
 Type:                              DYN (Position-Independent Executable file)
 Machine:                           Advanced Micro Devices X86-64
 Version:                           0x1
 Entry point address:               0x1060
 Start of program headers:          64 (bytes into file)
 Start of section headers:          13984 (bytes into file)
 Flags:                             0x0
 Size of this header:               64 (bytes)
 Size of program headers:           56 (bytes)
 Number of program headers:         13
 Size of section headers:           64 (bytes)
 Number of section headers:         31
 Section header string table index: 30
```

ELF Header (Ehdr)

```
00000000: 7f45 4c46 0201 0100 0000 0000 0000 0000  .ELF............
00000010: 0300 3e00 0100 0000 6010 0000 0000 0000  ..>.....`.......
00000020: 4000 0000 0000 0000 a036 0000 0000 0000  @........6......
00000030: 0000 0000 4000 3800 0d00 4000 1f00 1e00  ....@.8...@.....
00000040: 0600 0000 0400 0000 4000 0000 0000 0000  ........@.......
00000050: 0000 0000 0000 0000 4000 0000 0000 0000  @.......@.......
00000060:
00000070:
00000080:
00000090: 1803 0000 0000 0000 1c00 000
000000a0: 1c00 0000 0000 0000 0100 000
000000b0: 0100 0000 0400 0000 0000 000
000000c0: 0000 0000 0000 0000 0000 000
000000d0: 2806 0000 0000 0000 2806 000
000000e0: 0010 0000 0000 0000 0100 000
000000f0: 0010 0000 0000 0000 0010 000
00000100: 0010 0000 0000 0000 8101 000
```

E_SHSTRNDX

String Table 是第幾個 Section Header

```
0 00 00 00 00 00 00 00
Class:                             ELF64
Data:                              2's complement, little endian
Version:                           1 (current)
OS/ABI:                            UNIX - System V
ABI Version:                       0
Type:                              DYN (Position-Independent Executable file)
Machine:                           Advanced Micro Devices X86-64
Version:                           0x1
Entry point address:               0x1060
Start of program headers:          64 (bytes into file)
Start of section headers:          13984 (bytes into file)
Flags:                             0x0
Size of this header:               64 (bytes)
Size of program headers:           56 (bytes)
Number of program headers:         13
Size of section headers:           64 (bytes)
Number of section headers:         31
Section header string table index: 30
```

Program Header (Phdr)

每個 Program Header 都是 0x38

Program Header 會告訴 OS 該如何映射資料

```
00000000: 7f
00000010: 03
00000020: 40
00000030: 00
00000040: 0600 0000 0400 0000 4000 0000 0000 0000  ........@.......
00000050: 4000 0000 0000 0000 4000 0000 0000 0000  @.......@.......
00000060: d802 0000 0000 0000 d802 0000 0000 0000  ................
00000070: 0800 0000 0000 0000 0300 0000 0400 0000  ................
00000080: 1803 0000 0000 0000 1803 0000 0000 0000  ................
00000090: 1803 0000 0000 0000 1c00 0000 0000 0000  ................
000000a0: 1c00 0000 0000 0000 0100 0000 0000 0000  ................
000000b0: 0100 0000 0400 0000 0000 00
000000c0: 0000 0000 0000 0000 0000 00
000000d0: 2806 0000 0000 0000 2806 00
000000e0: 0010 0000 0000 0000 0100 00
000000f0: 0010 0000 0000 0000 0010 00
00000100: 0010 0000 0000 0000 8101 00
```

```
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
                 0x0000000000000628 0x0000000000000628  R      0x1000
  LOAD           0x0000000000001000 0x0000000000001000 0x0000000000001000
                 0x0000000000000181 0x0000000000000181  R E    0x1000
  LOAD           0x0000000000002000 0x0000000000002000 0x0000000000002000
```

Program Header (Phdr)

每個 Program Header 都是 0x38

Program Header 會告訴 OS 該如何映射資料

```
00000000: 7f
00000010: 03
00000020: 40
00000030: 00
00000040: 0600 0000 0400 0000 4000 0000 0000 0000  ........@.......
00000050: 4000 0000 0000 0000 4000 0000 0000 0000  @.......@.......
00000060: d802 0000 0000 0000 d802 0000 0000 0000  ................
00000070: 0800 0000 0000 0000 0300 0000 0400 0000  ................
00000080: 1803 0000 0000 0000 1803 0000 0000 0000  ................
00000090: 1803 0000 0000 0000 1c00 0000 0000 0000  ................
000000a0: 1c00 0000 0000 0000 0100 0000 0000 0000  ................
000000b0: 0100 0000 0400 0000 0000 00
000000c0: 0000 0000 0000 0000 0000 00
000000d0: 2806 0000 0000 0000 2806 00
000000e0: 0010 0000 0000 0000 0100 00
000000f0: 0010 0000 0000 0000 0010 00
00000100: 0010 0000 0000 0000 8101 00
```

```
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
                 0x0000000000000628 0x0000000000000628  R      0x1000
  LOAD           0x0000000000001000 0x0000000000001000 0x0000000000001000
                 0x0000000000000181 0x0000000000000181  R E    0x1000
  LOAD           0x0000000000002000 0x0000000000002000 0x0000000000002000
```

Program Header (Phdr)

p_type

Program Header 種類

```
00000000: 7f45 4c46 02                                    .ELF.....
00000010: 0300 3e00 01                                    ..>.....
00000020: 4000 0000 00                                    @.....
00000030: 0000 0000 40                                    ....@...
00000040: 0600 0000 0400 0000 4000 0000 0000 0000  ........@.......
00000050: 4000 0000 0000 0000 4000 0000 0000 0000  @.......@.......
00000060: d802 0000 0000 0000 d802 0000 0000 0000  ................
00000070: 0800 0000 0000 0000 0300 0000 0400 0000  ................
00000080: 1803 0000 0000 0000 1803 0000 0000 0000  ................
00000090: 1803 0000 0000 0000 1c00 0000 0000 0000  ................
000000a0: 1c00 0000 0000 0000 0100 0000 0000 0000  ................
000000b0: 0100 0000 0400 0000 0000 00              ................
000000c0: 0000 0000 0000 0000 0000 00              ................
000000d0: 2806 0000 0000 0000 2806 00              ....(.....(.....
000000e0: 0010 0000 0000 0000 0100 00              ................
000000f0: 0010 0000 0000 0000 0010 00              ................
00000100: 0010 0000 0000 0000 8101 00              ................
```

```
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
                 0x0000000000000628 0x0000000000000628  R      0x1000
  LOAD           0x0000000000001000 0x0000000000001000 0x0000000000001000
                 0x0000000000000181 0x0000000000000181  R E    0x1000
  LOAD           0x0000000000002000 0x0000000000002000 0x0000000000002000
```

Program Header (Phdr)

p_flag
這一個 Phdr 的權限 R/W/X

```
00000000: 7f45 4c46 02                    000 0000 0000 0000   .ELF............
00000010: 0300 3e00 01                    010 0000 0000 0000   ..>.....@.......
00000020: 4000 0000 00                                         @.....
00000030: 0000 0000 40                                         @.....
00000040: 0600 0000 0400 0000 4000 0000 0000 0000   ........@.......
00000050: 4000 0000 0000 0000 4000 0000 0000 0000   @.......@.......
00000060: d802 0000 0000 0000 d802 0000 0000 0000   ................
00000070: 0800 0000 0000 0000 0300 0000 0400 0000   ................
00000080: 1803 0000 0000 0000 1803 0000 0000 0000   ................
00000090: 1803 0000 0000 0000 1c00 0000 0000 0000   ................
000000a0: 1c00 0000 0000 0000 0100 0000 0000 0000   ................
000000b0: 0100 0000 0400 0000 0000 00               ................
000000c0: 0000 0000 0000 0000 0000 00               ................
000000d0: 2806 0000 0000 0000 2806 00               ................
000000e0: 0010 0000 0000 0000 0100 00               ................
000000f0: 0010 0000 0000 0000 0010 00               ................
00000100: 0010 0000 0000 0000 8101 00               ................
```

```
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
                 0x0000000000000628 0x0000000000000628  R      0x1000
  LOAD           0x0000000000001000 0x0000000000001000 0x0000000000001000
                 0x0000000000000181 0x0000000000000181  R E    0x1000
  LOAD           0x0000000000002000 0x0000000000002000 0x0000000000002000
```

Program Header (Phdr)

p_offset
這一段資料在檔案的 Offset

```
00000000: 7f45 4c46 02                    0000 0000 0000   .ELF.....
00000010: 0300 3e00 01                    0000 0000 0000   ..>.....
00000020: 4000 0000 00                                     @.....
00000030: 0000 0000 40                                     @.....
00000040: 0600 0000 0400 0000 4000 0000 0000 0000  ........@.......
00000050: 4000 0000 0000 0000 4000 0000 0000 0000  @.......@.......
00000060: d802 0000 0000 0000 d802 0000 0000 0000  ................
00000070: 0800 0000 0000 0000 0300 0000 0400 0000  ................
00000080: 1803 0000 0000 0000 1803 0000 0000 0000  ................
00000090: 1803 0000 0000 0000 1c00 0000 0000 0000  ................
000000a0: 1c00 0000 0000 0000 0100 0000 0000 0000  ................
000000b0: 0100 0000 0400 0000 0000 00              ................
000000c0: 0000 0000 0000 0000 0000 00              ................
000000d0: 2806 0000 0000 0000 2806 00              ................
000000e0: 0010 0000 0000 0000 0100 00              ................
000000f0: 0010 0000 0000 0000 0010 00              ................
00000100: 0010 0000 0000 0000 8101 00              ................
```

```
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
                 0x0000000000000628 0x0000000000000628  R      0x1000
  LOAD           0x0000000000001000 0x0000000000001000 0x0000000000001000
                 0x0000000000000181 0x0000000000000181  R E    0x1000
  LOAD           0x0000000000002000 0x0000000000002000 0x0000000000002000
```

Program Header (Phdr)

p_vaaddr

Virtual Address

```
00000000: 7f45 4c46 02                                    .ELF.....
00000010: 0300 3e00 01                                    ..>.....
00000020: 4000 0000 00                                    @.....
00000030: 0000 0000 40                                    ....@.
00000040: 0600 0000 0400 0000 4000 0000 0000 0000         ........@.......
00000050: 4000 0000 0000 0000 4000 0000 0000 0000         @.......@.......
00000060: d802 0000 0000 0000 d802 0000 0000 0000         ................
00000070: 0800 0000 0000 0000 0300 0000 0400 0000         ................
00000080: 1803 0000 0000 0000 1803 0000 0000 0000         ................
00000090: 1803 0000 0000 0000 1c00 0000 0000 0000         ................
000000a0: 1c00 0000 0000 0000 0100 0000 0000 0000         ................
000000b0: 0100 0000 0400 0000 0000 00
000000c0: 0000 0000 0000 0000 0000 00
000000d0: 2806 0000 0000 0000 2806 00
000000e0: 0010 0000 0000 0000 0100 00
000000f0: 0010 0000 0000 0000 0010 00
00000100: 0010 0000 0000 0000 8101 00
```

```
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
                 0x0000000000000628 0x0000000000000628  R      0x1000
  LOAD           0x0000000000001000 0x0000000000001000 0x0000000000001000
                 0x0000000000000181 0x0000000000000181  R E    0x1000
  LOAD           0x0000000000002000 0x0000000000002000 0x0000000000002000
```

Program Header (Phdr)

p_paaddr

Physical Address，現在基本上不會用到

```
00000000: 7f45 4c46 02                    0000 0000 0000   .ELF..........
00000010: 0300 3e00 01                    0000 0000 0000   ..>.....`.....
00000020: 4000 0000 00                    0000 0000 0000   @.....`.....
00000030: 0000 0000 40                    0000 0000 0000   ....@.8.....
00000040: 0600 0000 0400 0000 4000 0000 0000 0000  ........@.......
00000050: 4000 0000 0000 0000 4000 0000 0000 0000  @.......@.......
00000060: d802 0000 0000 0000 d802 0000 0000 0000  ................
00000070: 0800 0000 0000 0000 0300 0000 0400 0000  ................
00000080: 1803 0000 0000 0000 1803 0000 0000 0000  ................
00000090: 1803 0000 0000 0000 1c00 0000 0000 0000  ................
000000a0: 1c00 0000 0000 0000 0100 0000 0000 0000  ................
000000b0: 0100 0000 0400 0000 0000 00              ................
000000c0: 0000 0000 0000 0000 0000 00              ................
000000d0: 2806 0000 0000 0000 2806 00              ................
000000e0: 0010 0000 0000 0000 0100 00              ................
000000f0: 0010 0000 0000 0000 0010 00              ................
00000100: 0010 0000 0000 0000 8101 00              ................
```

```
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
                 0x0000000000000628 0x0000000000000628  R      0x1000
  LOAD           0x0000000000001000 0x0000000000001000 0x0000000000001000
                 0x0000000000000181 0x0000000000000181  R E    0x1000
  LOAD           0x0000000000002000 0x0000000000002000 0x0000000000002000
```

Program Header (Phdr)

p_filesz

從檔案 offset 開始讀多少資料

```
00000000: 7f45 4c46 02                    0 0000 0000 0000   .ELF.....
00000010: 0300 3e00 01                    0 0000 0000 0000   ..>.....
00000020: 4000 0000 00                                       @.....
00000030: 0000 0000 40                                       ....
00000040: 0600 0000 0400 0000 4000 0000 0000 0000  ........@.......
00000050: 4000 0000 0000 0000 4000 0000 0000 0000  @.......@.......
00000060: d802 0000 0000 0000 d802 0000 0000 0000  ................
00000070: 0800 0000 0000 0000 0300 0000 0400 0000  ................
00000080: 1803 0000 0000 0000 1803 0000 0000 0000  ................
00000090: 1803 0000 0000 0000 1c00 0000 0000 0000  ................
000000a0: 1c00 0000 0000 0000 0100 0000 0000 0000  ................
000000b0: 0100 0000 0400 0000 0000 00              ................
000000c0: 0000 0000 0000 0000 0000 00              ................
000000d0: 2806 0000 0000 0000 2806 00              ................
000000e0: 0010 0000 0000 0000 0100 00              ................
000000f0: 0010 0000 0000 0000 0010 00              ................
00000100: 0010 0000 0000 0000 8101 00              ................
```

```
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
                 0x0000000000000628 0x0000000000000628  R      0x1000
  LOAD           0x0000000000001000 0x0000000000001000 0x0000000000001000
                 0x0000000000000181 0x0000000000000181  R E    0x1000
  LOAD           0x0000000000002000 0x0000000000002000 0x0000000000002000
```

Program Header (Phdr)

p_memsz

映射記憶體區塊的大小，可以跟檔案大小不一樣

```
00000000: 7f45 4c46 02[illegible]000 0000 0000   .ELF.....[illegible]
00000010: 0300 3e00 01[illegible]000 0000 0000   [illegible]
00000020: 4000 0000 00[illegible]                [illegible]
00000030: 0000 0000 40[illegible]                [illegible]
00000040: 0600 0000 0400 0000 4000 0000 0000 0000  ........@.......
00000050: 4000 0000 0000 0000 4000 0000 0000 0000  @.......@.......
00000060: d802 0000 0000 0000 d802 0000 0000 0000  ................
00000070: 0800 0000 0000 0000 0300 0000 0400 0000  ................
00000080: 1803 0000 0000 0000 1803 0000 0000 0000  ................
00000090: 1803 0000 0000 0000 1c00 0000 0000 0000  ................
000000a0: 1c00 0000 0000 0000 0100 0000 0000 0000  ................
000000b0: 0100 0000 0400 0000 0000 00[illegible]
000000c0: 0000 0000 0000 0000 0000 00[illegible]
000000d0: 2806 0000 0000 0000 2806 00[illegible]
000000e0: 0010 0000 0000 0000 0100 00[illegible]
000000f0: 0010 0000 0000 0000 0010 00[illegible]
00000100: 0010 0000 0000 0000 8101 00[illegible]
```

```
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
                 0x0000000000000628 0x0000000000000628  R      0x1000
  LOAD           0x0000000000001000 0x0000000000001000 0x0000000000001000
                 0x0000000000000181 0x0000000000000181  R E    0x1000
  LOAD           0x0000000000002000 0x0000000000002000 0x0000000000002000
                 0x[illegible]
```

Program Header (Phdr)

p_align

Segment Alignment.

```
00000000: 7f45 4c46 02                                    .ELF.....
00000010: 0300 3e00 01                                    ..>.....
00000020: 4000 0000 00                                    @.......
00000030: 0000 0000 40                                    ....@.8.
00000040: 0600 0000 0400 0000 4000 0000 0000 0000  ........@.......
00000050: 4000 0000 0000 0000 4000 0000 0000 0000  @.......@.......
00000060: d802 0000 0000 0000 d802 0000 0000 0000  ................
00000070: 0800 0000 0000 0000 0300 0000 0400 0000  ................
00000080: 1803 0000 0000 0000 1803 0000 0000 0000  ................
00000090: 1803 0000 0000 0000 1c00 0000 0000 0000  ................
000000a0: 1c00 0000 0000 0000 0100 0000 0000 0000  ................
000000b0: 0100 0000 0400 0000 0000 00              ................
000000c0: 0000 0000 0000 0000 0000 00              ................
000000d0: 2806 0000 0000 0000 2806 00              ....(.....(.....
000000e0: 0010 0000 0000 0000 0100 00              ................
000000f0: 0010 0000 0000 0000 0010 00              ................
00000100: 0010 0000 0000 0000 8101 00              ................
```

```
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
                 0x0000000000000628 0x0000000000000628  R      0x1000
  LOAD           0x0000000000001000 0x0000000000001000 0x0000000000001000
                 0x0000000000000181 0x0000000000000181  R E    0x1000
  LOAD           0x0000000000002000 0x0000000000002000 0x0000000000002000
```

Program Header (Phdr)

```
LOAD           0x0000000000002db8 0x0000000000003db8 0x0000000000003db8
               0x0000000000000258 0x0000000000000260  RW     0x1000
```

ELF 通過這些 Program Header 來告訴 OS 映射資料進入記憶體
以 segment 作為單位載入的，一個 segment 可以有多個 section

```
Section to Segment mapping:
 Segment Sections...
  00
  01     .interp
  02     .interp .note.gnu.property .note.gnu.build-id .note.ABI-tag .gnu.hash .dynsym .dynstr .gnu.version .gnu.version_r .rela.dyn .rela.plt
  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
  09     .note.gnu.property
  10     .eh_frame_hdr
  11
  12     .init_array .fini_array .dynamic .got
```

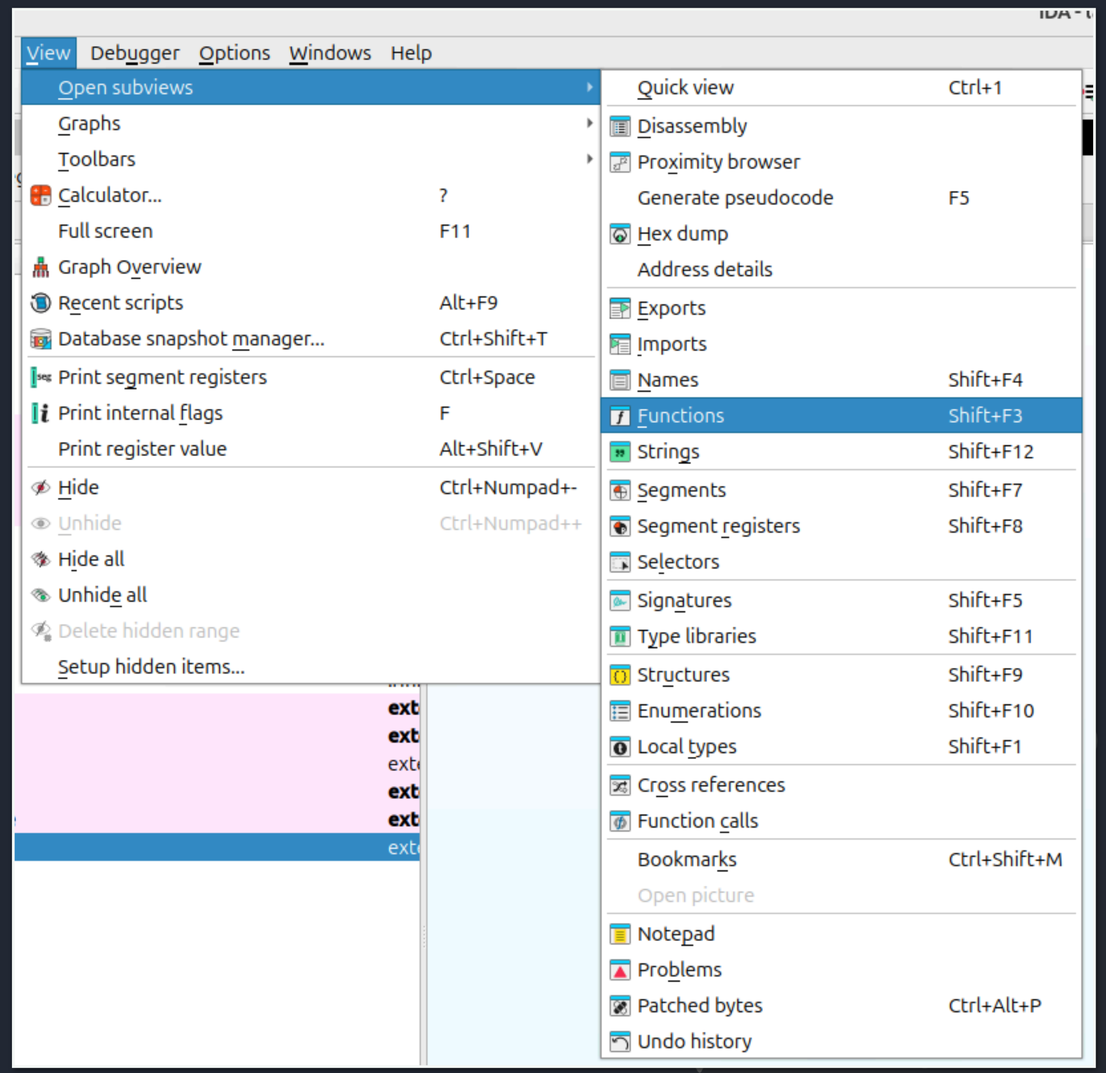
Common Section

Section	Perm	Description
.text	R-X	Executable Code (Instructions)
.data	RW-	Global with initial data
.rodata	R—	ReadOnly Data
.bss	RW-	Global without initial data

Back to IDA

Back to IDA

- 這些資訊 IDA 會幫你分析
- 可以從 sub-view 叫出來
  - 接下來介紹的東西都從這按出來



Back to IDA

- 這些資訊 IDA 會幫你分析

```
LOAD:0000000000000000                         LOAD            segment mempage public 'DATA' use64
LOAD:0000000000000000                                         assume cs:LOAD
LOAD:0000000000000000 7F 45 4C 46             dword_0         dd 464C457Fh                ; DATA XREF: LOAD:00000000000003F0↓o
LOAD:0000000000000000                                                                     ; LOAD:0000000000000408↓o ...
LOAD:0000000000000000                                                                     ; File format: \x7FELF
LOAD:0000000000000004 02                                      db 2                        ; File class: 64-bit
LOAD:0000000000000005 01                                      db 1                        ; Data encoding: little-endian
LOAD:0000000000000006 01                                      db 1                        ; File version
LOAD:0000000000000007 00                                      db 0                        ; OS/ABI: UNIX System V ABI
LOAD:0000000000000008 00                                      db 0                        ; ABI Version
LOAD:0000000000000009 00 00 00 00 00 00 00                    db 7 dup(0)                 ; Padding
LOAD:0000000000000010 03 00                                   dw 3                        ; File type: Shared object
LOAD:0000000000000012 3E 00                                   dw 3Eh                      ; Machine: x86-64
LOAD:0000000000000014 01 00 00 00                             dd 1                        ; File version
LOAD:0000000000000018 A0 10 00 00 00 00 00 00                 dq offset start             ; Entry point
LOAD:0000000000000020 40 00 00 00 00 00 00 00                 dq 40h                      ; PHT file offset
LOAD:0000000000000028 48 31 00 00 00 00 00 00                 dq 3148h                    ; SHT file offset
LOAD:0000000000000030 00 00 00 00                             dd 0                        ; Processor-specific flags
LOAD:0000000000000034 40 00                                   dw 40h                      ; ELF header size
LOAD:0000000000000036 38 00                                   dw 38h                      ; PHT entry size
LOAD:0000000000000038 0D 00                                   dw 0Dh                      ; Number of entries in PHT
LOAD:000000000000003A 40 00                                   dw 40h                      ; SHT entry size
LOAD:000000000000003C 1D 00                                   dw 1Dh                      ; Number of entries in SHT
LOAD:000000000000003E 1C 00                                   dw 1Ch                      ; SHT entry index for string table
LOAD:0000000000000040                         ; ELF64 Program Header
LOAD:0000000000000040                         ; PHT Entry 0
LOAD:0000000000000040 06 00 00 00             dword_40        dd 6                        ; DATA XREF: LOAD:0000000000000050↓o
LOAD:0000000000000040                                                                     ; Type: PHDR
LOAD:0000000000000044 04 00 00 00                             dd 4                        ; Flags
LOAD:0000000000000048 40 00 00 00 00 00 00 00                 dq 40h                      ; File offset
LOAD:0000000000000050 40 00 00 00 00 00 00 00                 dq offset dword_40          ; Virtual address
LOAD:0000000000000058 40 00 00 00 00 00 00 00                 dq 40h                      ; Physical address
LOAD:0000000000000060 D8 02 00 00 00 00 00 00                 dq 2D8h                     ; Size in file image
LOAD:0000000000000068 D8 02 00 00 00 00 00 00                 dq 2D8h                     ; Size in memory image
LOAD:0000000000000070 08 00 00 00 00 00 00 00                 dq 8                        ; Alignment
LOAD:0000000000000078                         ; PHT Entry 1
LOAD:0000000000000078 03 00 00 00                             dd 3                        ; Type: INTERP
LOAD:000000000000007C 04 00 00 00                             dd 4                        ; Flags
LOAD:0000000000000080 18 03 00 00 00 00 00 00                 dq 318h                     ; File offset
LOAD:0000000000000088 18 03 00 00 00 00 00 00                 dq offset aLib64LdLinuxX8 ; Virtual address
LOAD:0000000000000090 18 03 00 00 00 00 00 00                 dq 318h                     ; Physical address
LOAD:0000000000000098 1C 00 00 00 00 00 00 00                 dq 1Ch                      ; Size in file image
LOAD:00000000000000A0 1C 00 00 00 00 00 00 00                 dq 1Ch                      ; Size in memory image
LOAD:00000000000000A8 01 00 00 00 00 00 00 00                 dq 1                        ; Alignment
LOAD:00000000000000B0                         ; PHT Entry 2
LOAD:00000000000000B0 01 00 00 00                             dd 1                        ; Type: LOAD
LOAD:00000000000000B4 04 00 00 00                             dd 4                        ; Flags
LOAD:00000000000000B8 00 00 00 00 00 00 00 00                 dq 0                        ; File offset
LOAD:00000000000000C0 00 00 00 00 00 00 00 00                 dq 0                        ; Virtual address
LOAD:00000000000000C8 00 00 00 00 00 00 00 00                 dq 0                        ; Physical address
LOAD:00000000000000D0 D8 06 00 00 00 00 00 00                 dq 6D8h                     ; Size in file image
LOAD:00000000000000D8 D8 06 00 00 00 00 00 00                 dq 6D8h                     ; Size in memory image
LOAD:00000000000000E0 00 10 00 00 00 00 00 00                 dq 1000h                    ; Alignment
LOAD:00000000000000E8                         ; PHT Entry 3
LOAD:00000000000000E8 01 00 00 00                             dd 1                        ; Type: LOAD
LOAD:00000000000000EC 05 00 00 00                             dd 5                        ; Flags
LOAD:00000000000000F0 00 10 00 00 00 00 00 00                 dq 1000h                    ; File offset
LOAD:00000000000000F8 00 10 00 00 00 00 00 00                 dq offset _init_proc        ; Virtual address
LOAD:0000000000000100 00 10 00 00 00 00 00 00                 dq 1000h                    ; Physical address
LOAD:0000000000000108 8D 02 00 00 00 00 00 00                 dq 28Dh                     ; Size in file image
LOAD:0000000000000110 8D 02 00 00 00 00 00 00                 dq 28Dh                     ; Size in memory image
LOAD:0000000000000118 00 10 00 00 00 00 00 00                 dq 1000h                    ; Alignment
00000000 0000000000000000: LOAD:dword_0 (Synchronized with Hex View-1)
```

Segment

Name	Start	End	R	W	X	D	L	Align	Base	Type	Class	AD	es	ss	ds	fs	gs
LOAD	0000000000000000	00000000000006D8	R	.	.	.	L	mempage	0001	public	DATA	64	0000	0000	0014	0000	0000
.init	0000000000001000	000000000000101B	R	.	X	.	L	dword	0008	public	CODE	64	FFFFF...	FFFFF...	0014	FFFFF...	FFFFFFFFFFFFFFF
LOAD	000000000000101B	0000000000001020	R	.	X	.	L	mempage	0002	public	CODE	64	0000	0000	0014	0000	0000
.plt	0000000000001020	0000000000001060	R	.	X	.	L	para	0009	public	CODE	64	FFFFF...	FFFFF...	0014	FFFFF...	FFFFFFFFFFFFFFF
.plt.got	0000000000001060	0000000000001070	R	.	X	.	L	para	000A	public	CODE	64	FFFFF...	FFFFF...	0014	FFFFF...	FFFFFFFFFFFFFFF
.plt.sec	0000000000001070	00000000000010A0	R	.	X	.	L	para	000B	public	CODE	64	FFFFF...	FFFFF...	0014	FFFFF...	FFFFFFFFFFFFFFF
.text	00000000000010A0	000000000000127F	R	.	X	.	L	para	000C	public	CODE	64	FFFFF...	FFFFF...	0014	FFFFF...	FFFFFFFFFFFFFFF
LOAD	000000000000127F	0000000000001280	R	.	X	.	L	mempage	0002	public	CODE	64	0000	0000	0014	0000	0000
.fini	0000000000001280	000000000000128D	R	.	X	.	L	dword	000D	public	CODE	64	FFFFF...	FFFFF...	0014	FFFFF...	FFFFFFFFFFFFFFF
.rodata	0000000000002000	0000000000002140	R	.	.	.	L	qword	000E	public	CONST	64	FFFFF...	FFFFF...	0014	FFFFF...	FFFFFFFFFFFFFFF
.eh_frame_hdr	0000000000002140	000000000000217C	R	.	.	.	L	dword	000F	public	CONST	64	FFFFF...	FFFFF...	0014	FFFFF...	FFFFFFFFFFFFFFF
LOAD	000000000000217C	0000000000002180	R	.	.	.	L	mempage	0003	public	DATA	64	0000	0000	0014	0000	0000
.eh_frame	0000000000002180	000000000000224C	R	.	.	.	L	qword	0010	public	CONST	64	FFFFF...	FFFFF...	0014	FFFFF...	FFFFFFFFFFFFFFF
.init_array	0000000000003DA8	0000000000003DB0	R	W	.	.	L	qword	0011	public	DATA	64	FFFFF...	FFFFF...	0014	FFFFF...	FFFFFFFFFFFFFFF
.fini_array	0000000000003DB0	0000000000003DB8	R	W	.	.	L	qword	0012	public	DATA	64	FFFFF...	FFFFF...	0014	FFFFF...	FFFFFFFFFFFFFFF
LOAD	0000000000003DB8	0000000000003FA8	R	W	.	.	L	mempage	0004	public	DATA	64	0000	0000	0014	0000	0000
.got	0000000000003FA8	0000000000004000	R	W	.	.	L	qword	0013	public	DATA	64	FFFFF...	FFFFF...	0014	FFFFF...	FFFFFFFFFFFFFFF
.data	0000000000004000	0000000000004018	R	W	.	.	L	qword	0014	public	DATA	64	FFFFF...	FFFFF...	0014	FFFFF...	FFFFFFFFFFFFFFF
.bss	0000000000004018	0000000000004020	R	W	.	.	L	byte	0015	public	BSS	64	FFFFF...	FFFFF...	0014	FFFFF...	FFFFFFFFFFFFFFF
extern	0000000000004020	0000000000004060	?	?	?	.	L	qword	0016	public		64	FFFFF...	FFFFF...	FFFFF...	FFFFF...	FFFFFFFFFFFFFFF

Segment

- 就跟先前介紹的一樣

Strings

Address	Length	Type	String
LOAD:000000…	0000001C	C	/lib64/ld-linux-x86-64.so.2
LOAD:000000…	00000008	C	wprintf
LOAD:000000…	00000011	C	__stack_chk_fail
LOAD:000000…	00000012	C	__libc_start_main
LOAD:000000…	0000000F	C	__cxa_finalize
LOAD:000000…	0000000A	C	libc.so.6
LOAD:000000…	0000000A	C	GLIBC_2.4
LOAD:000000…	0000000C	C	GLIBC_2.2.5
LOAD:000000…	0000000B	C	GLIBC_2.34
LOAD:000000…	0000001C	C	_ITM_deregisterTMCloneTable
LOAD:000000…	0000000F	C	__gmon_start__
LOAD:000000…	0000001A	C	_ITM_registerTMCloneTable
.rodata:00000…	0000000C	C	HelloWorld!
.eh_frame:000…	00000006	C	9*3$\"

Strings

- 被 IDA 認成 String Literals 的資訊都會出現在這
  - 也就是說，如果沒有被標記的不會在這
- 特殊長相的字串不會自動跑出來
  - 但通常特殊長相的字串用 strings 也讀不出來

Import / Export

Import Table / Export Table

- 雖然 ELF 不像 PE 真的有個叫做 Import / Export Table 的結構
  - Windows 跟 Linux 的動態裝載的機制不同
- ELF 中的 .dynsym 會用來乘載這些動態連結的資訊
- IDA 將動態裝載的資訊放在 Import / Export Table View

Import Table

Address	Ordinal	Name	Library
.dynsym			
0000000000004020		**__libc_start_main@@GLIBC_2.34**	**.dynsym**
0000000000004028		**puts@@GLIBC_2.2.5**	**.dynsym**
0000000000004030		__stack_chk_fail@@GLIBC_2.4	.dynsym
0000000000004038		**wprintf@@GLIBC_2.2.5**	**.dynsym**
0000000000004040		**__cxa_finalize@@GLIBC_2.2.5**	**.dynsym**
0000000000004048		_ITM_deregisterTMCloneTable	.dynsym
0000000000004050		__gmon_start__	.dynsym
0000000000004058		_ITM_registerTMCloneTable	.dynsym

Import Table

- 當前 Executable 用到了來自外部動態連結的 Symbol
  - Function, Variable…
- 可以用來找這個 Executable 有沒有用到特別的 Library Function
  - mprotect, mmap…

Export Table

Name	Address	Ordinal
.init_proc	0000000000001000	
.term_proc	0000000000001280	
start	00000000000010A0	[main entry]

Export Table

- 當前 Executable 供外部動態連結使用的資源
- 使用場景
  - 逆向工程的目標是一個 Library

x64 Calling Convention

Calling Convention

- 程式呼叫子邏輯的時候用來傳遞參數的慣例
- x86 呼叫慣例大部分會將 Parameters 往 Stack 上面堆
- x64 則是依賴 Register 傳參

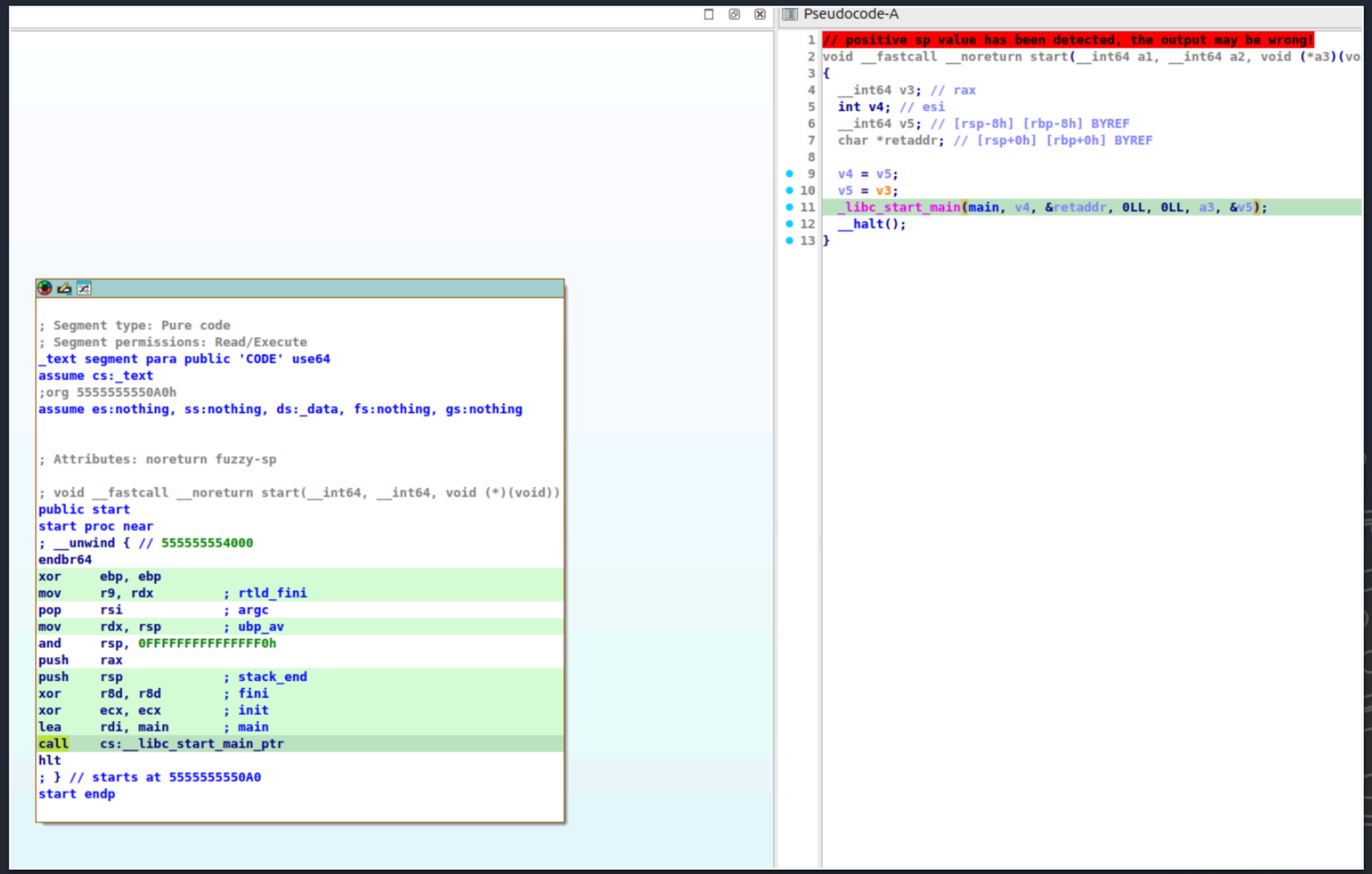
x64 Calling Convention

- Syscall 跟 C ABI 有差異

x64 Calling Convention

- *C ABI*
  - Return Value: RDX:RAX
  - Parameters: RDI, RSI, RDX, RCX, r8, r9, Stack
- *Syscall Calling Convention*
  - Syscall Number: RAX
  - Parameters: RDI, RSI, RDX, r10, r8, r9, Stack
  - Return Value: RAX

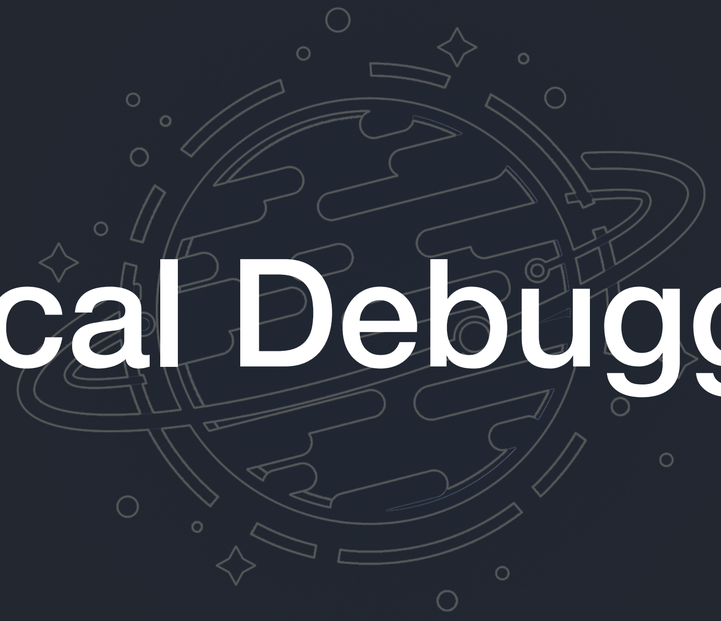
x64 Calling Convention



```
; Segment type: Pure code
; Segment permissions: Read/Execute
_text segment para public 'CODE' use64
assume cs:_text
;org 5555555550A0h
assume es:nothing, ss:nothing, ds:_data, fs:nothing, gs:nothing


; Attributes: noreturn fuzzy-sp

; void __fastcall __noreturn start(__int64, __int64, void (*)(void))
public start
start proc near
; __unwind { // 555555554000
endbr64
xor     ebp, ebp
mov     r9, rdx         ; rtld_fini
pop     rsi             ; argc
mov     rdx, rsp        ; ubp_av
and     rsp, 0FFFFFFFFFFFFFFF0h
push    rax
push    rsp             ; stack_end
xor     r8d, r8d        ; fini
xor     ecx, ecx        ; init
lea     rdi, main       ; main
call    cs:__libc_start_main_ptr
hlt
; } // starts at 5555555550A0
start endp
```

Pseudocode-A

```
1  // positive sp value has been detected, the output may be wrong!
2  void __fastcall __noreturn start(__int64 a1, __int64 a2, void (*a3)(vo
3  {
4    __int64 v3; // rax
5    int v4; // esi
6    __int64 v5; // [rsp-8h] [rbp-8h] BYREF
7    char *retaddr; // [rsp+0h] [rbp+0h] BYREF
8
9    v4 = v5;
10   v5 = v3;
11   _libc_start_main(main, v4, &retaddr, 0LL, 0LL, a3, &v5);
12   __halt();
13 }
```



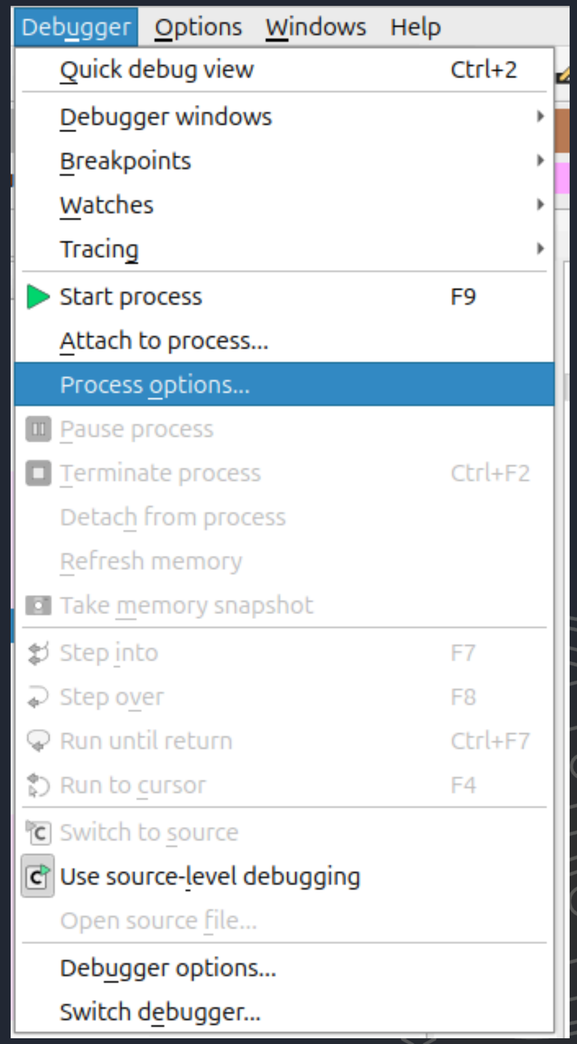
Local Debugger

Local Debugger

- IDAFree 有 local debugger
  - IDAPro 有 remote debugger，可以跨平台 debug
- IDAFree on Linux 可以 debug ELF
- 該有的 Debugger 功能都有，還多了 Decompiler

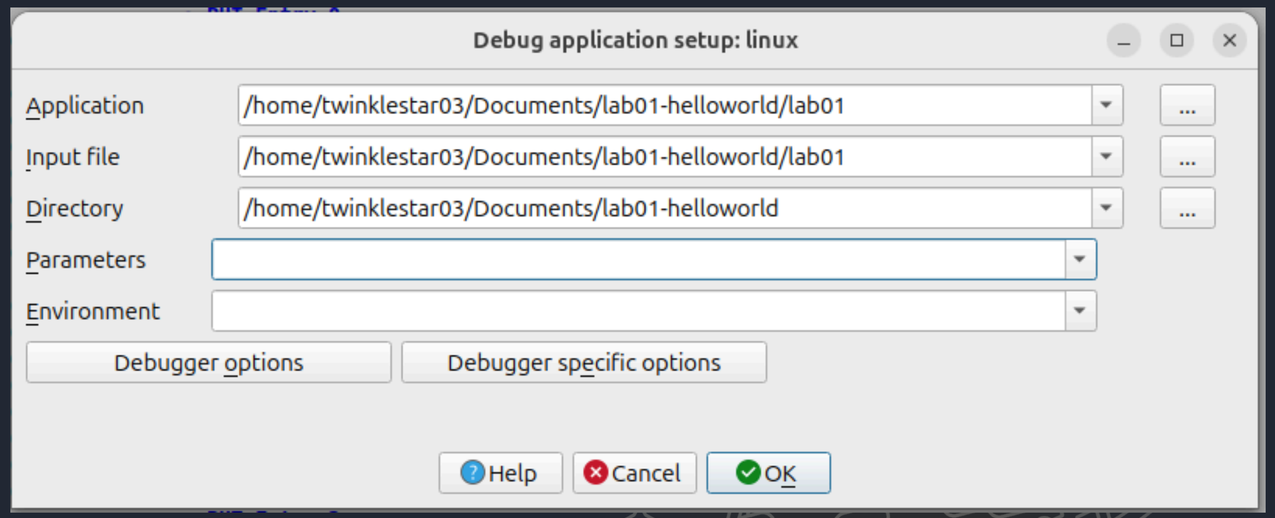
Local Debugger

- 啟動方式
  - 先進入 Process options 調整參數



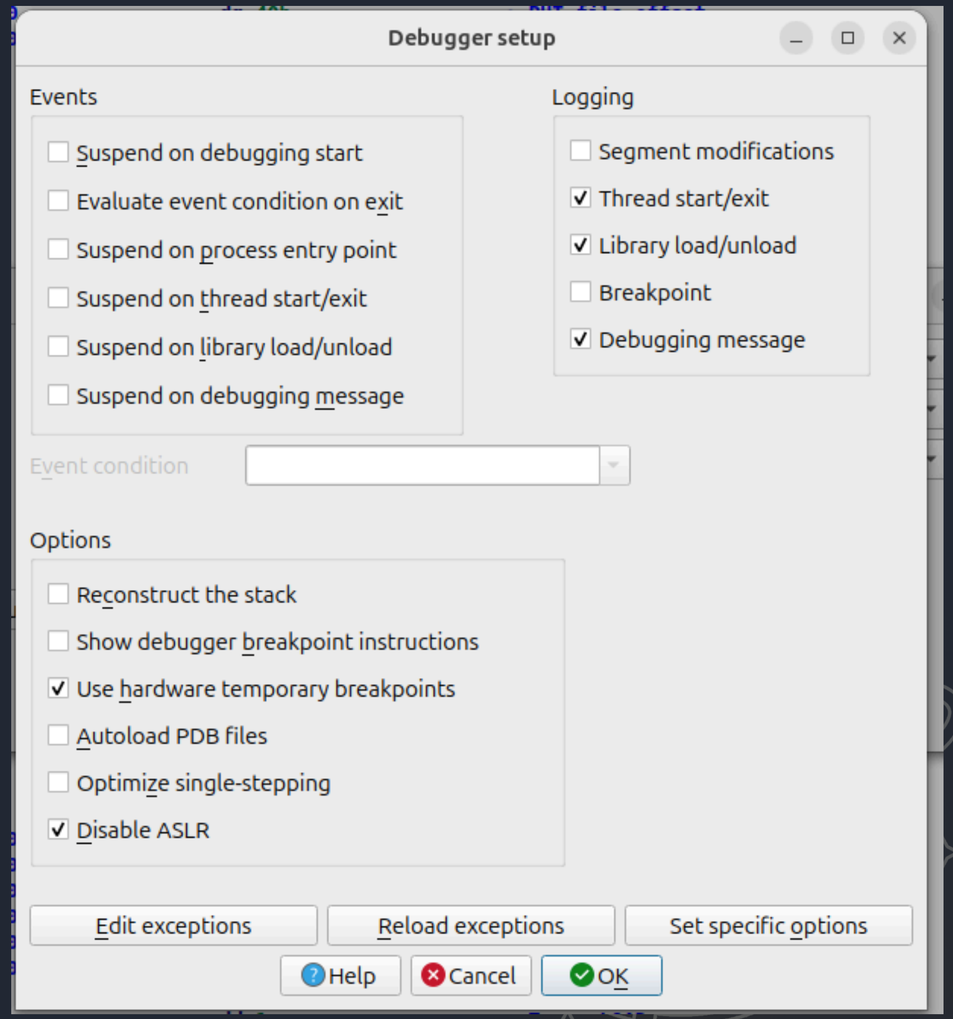
Local Debugger

- 指定執行檔位置
- 調整要輸入給程式的參數
- 環境變數



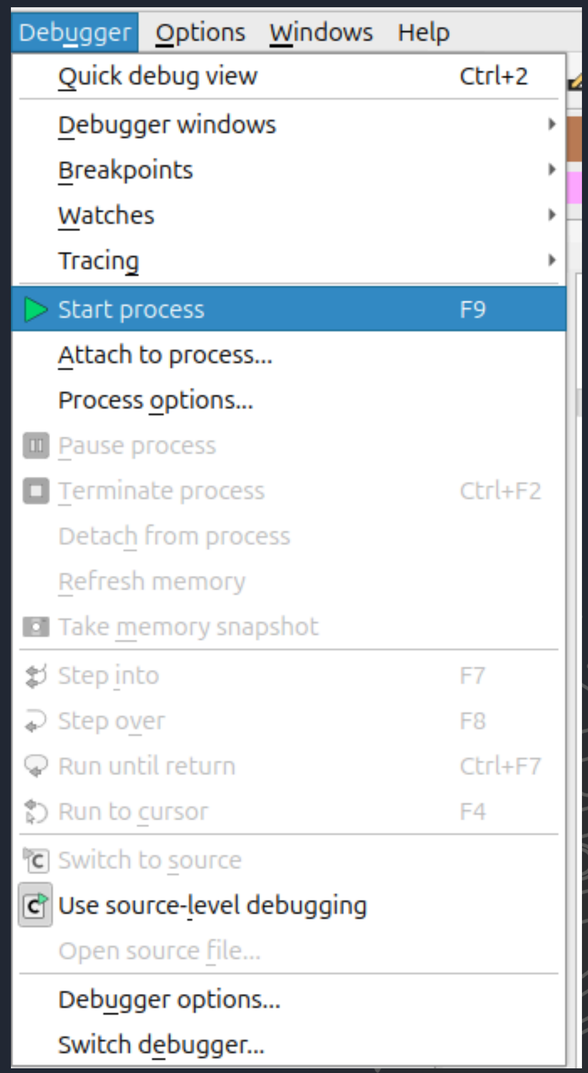
Local Debugger

- Debugger options
  - 設定要不要在 start 後中斷
  - Exception 抓取規則
  - ASLR 開關



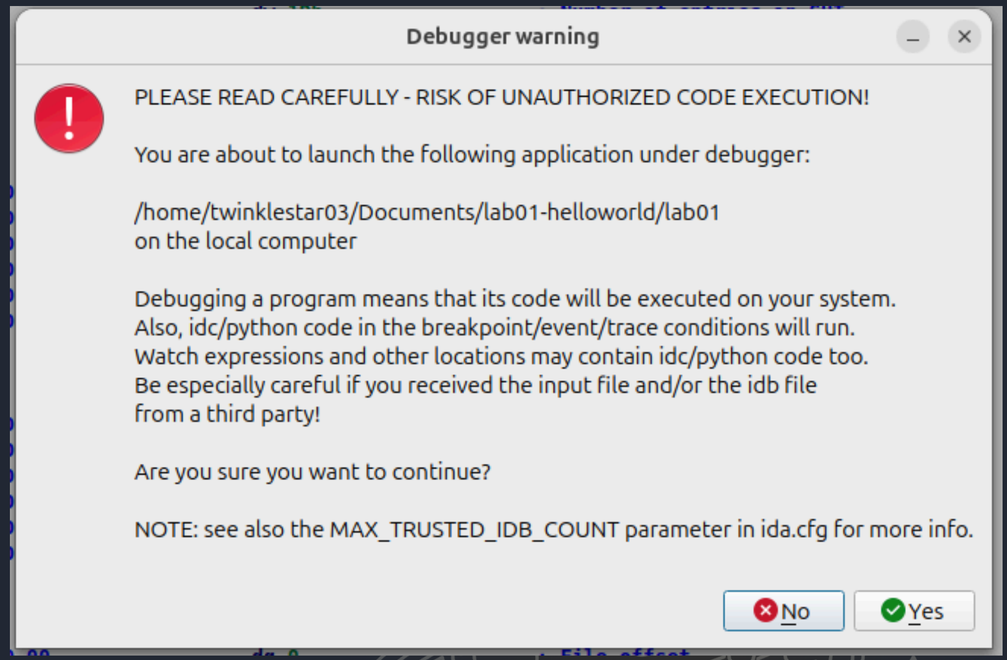
Local Debugger

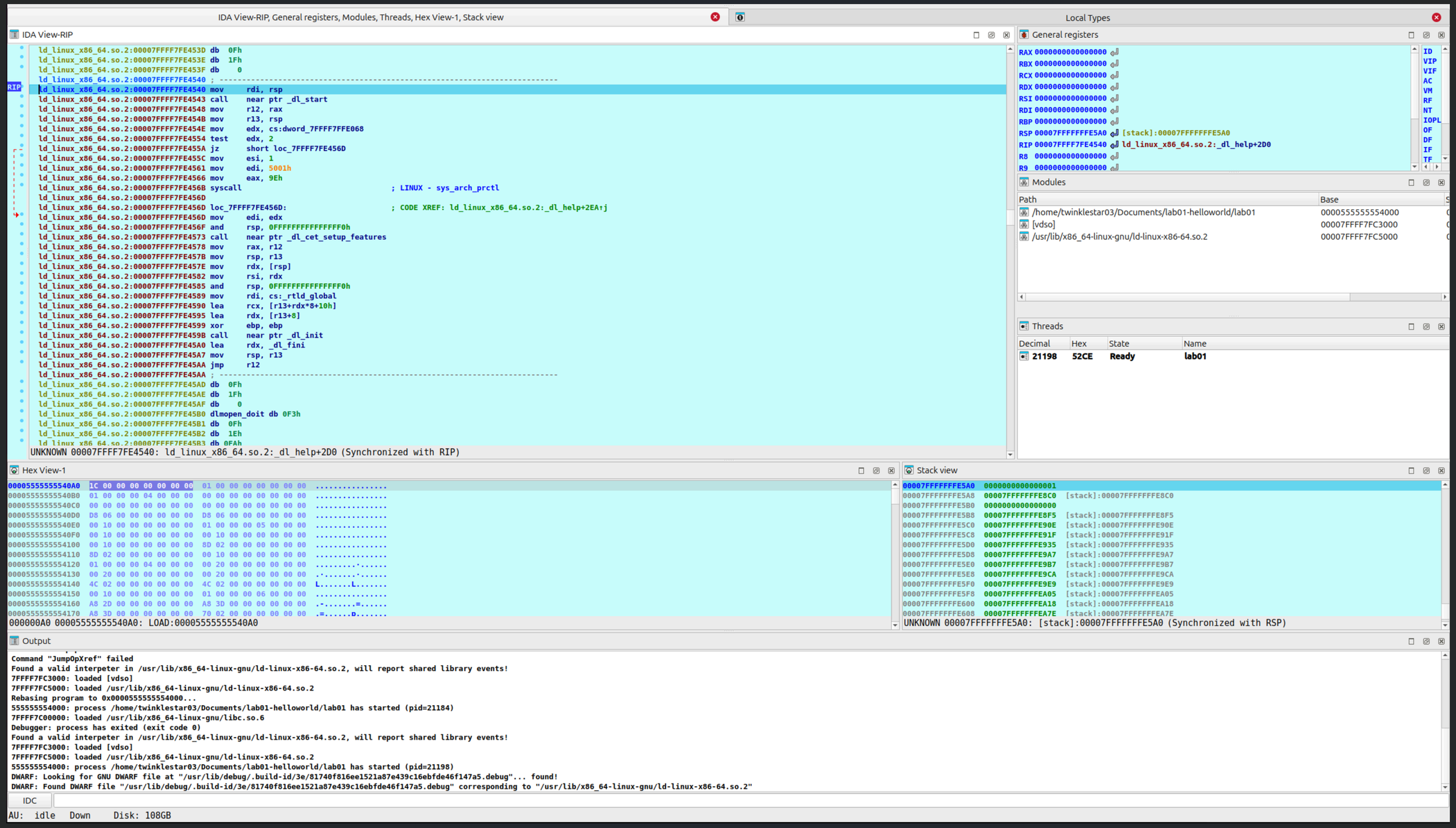
- 調整完畢後 [F9] 啟動 Process 並開始 Debug
  - 也可以從選單中用 Attach 的方式

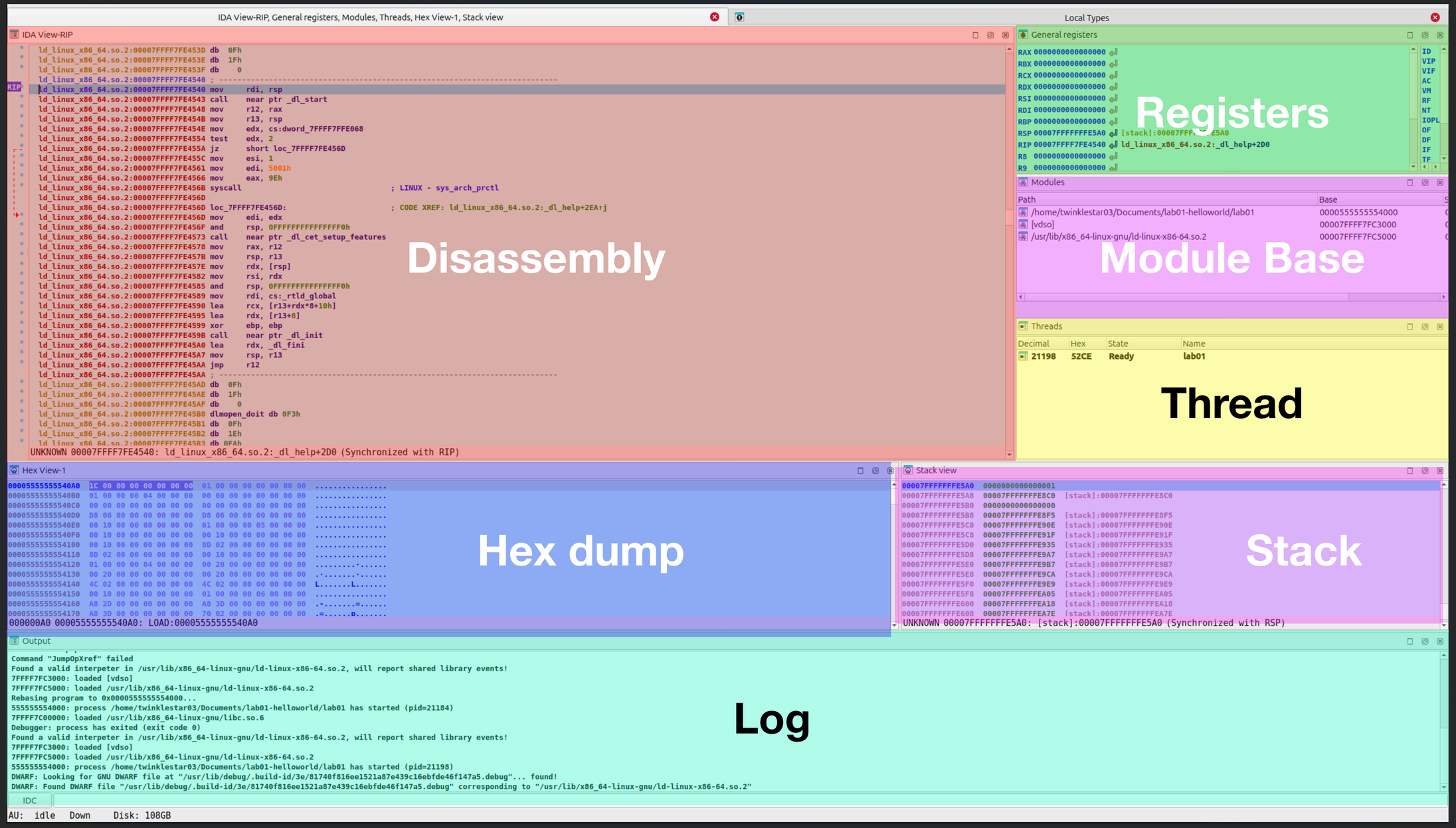


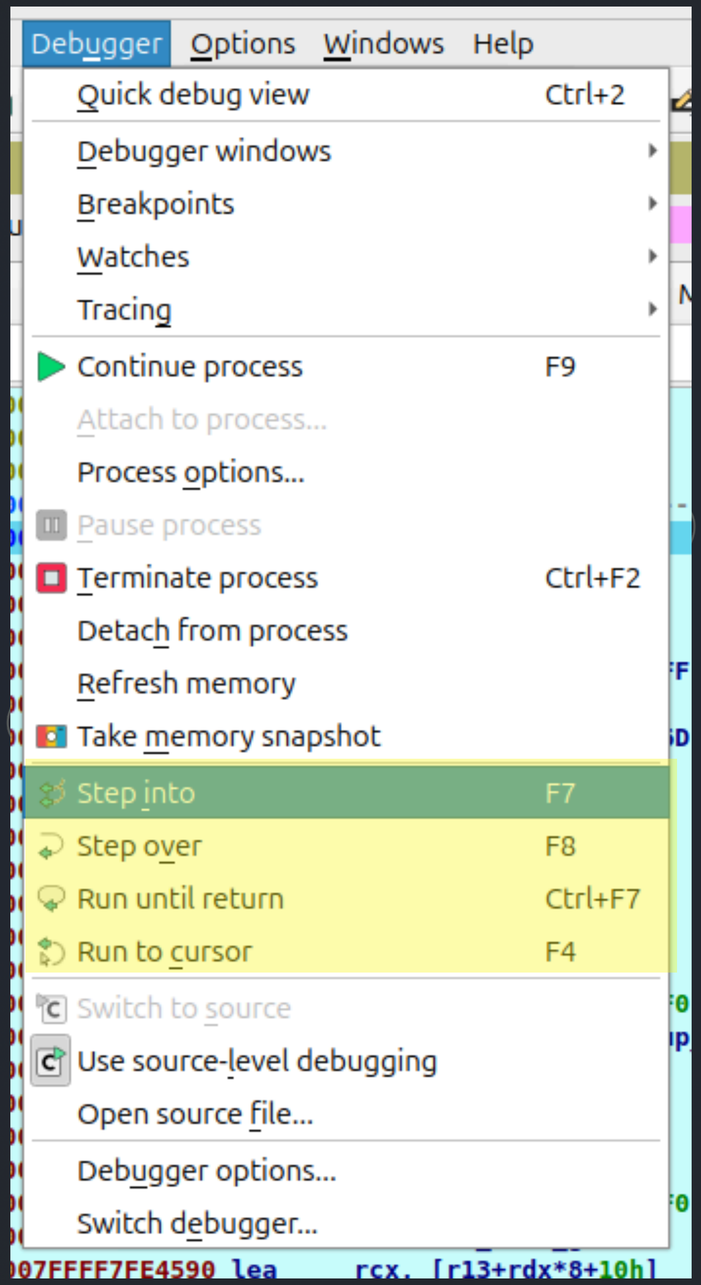
Local Debugger

- 注意！
  - Debug 的時候等同於直接執行
  - 不要執行來路不明的程式



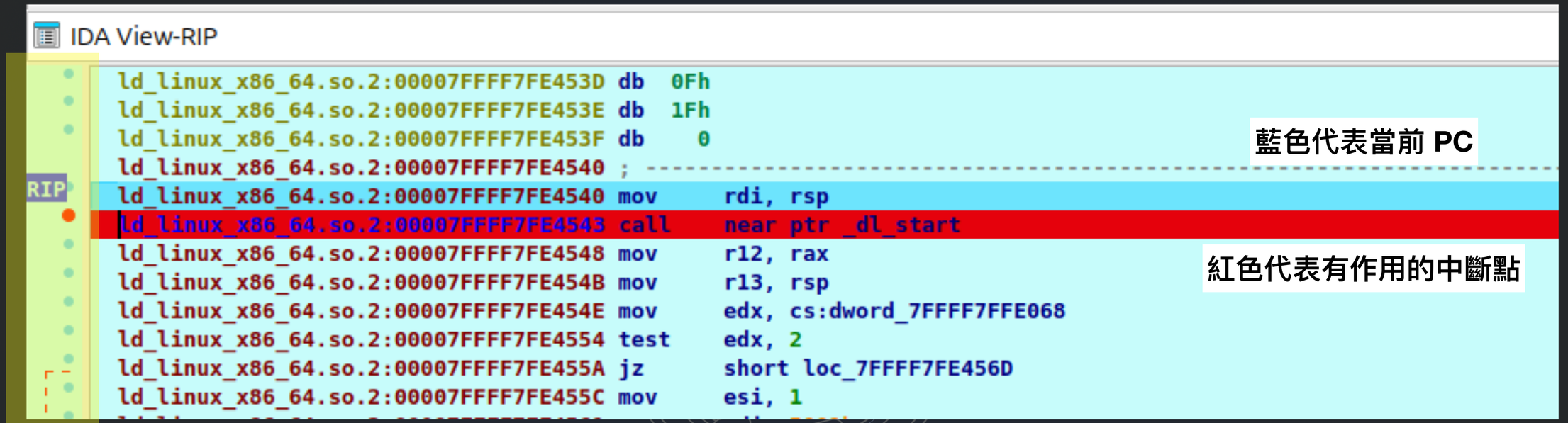

IDA View-RIP, General registers, Modules, Threads, Hex View-1, Stack view
Local Types
IDA View-RIP
ld_linux_x86_64.so.2:00007FFFF7FE453D db 0Fh
ld_linux_x86_64.so.2:00007FFFF7FE453E db 1Fh
ld_linux_x86_64.so.2:00007FFFF7FE453F db 0
ld_linux_x86_64.so.2:00007FFFF7FE4540 ; ---
RIP
ld_linux_x86_64.so.2:00007FFFF7FE4540 mov rdi, rsp
ld_linux_x86_64.so.2:00007FFFF7FE4543 call near ptr _dl_start
ld_linux_x86_64.so.2:00007FFFF7FE4548 mov r12, rax
ld_linux_x86_64.so.2:00007FFFF7FE454B mov r13, rsp
ld_linux_x86_64.so.2:00007FFFF7FE454E mov edx, cs:dword_7FFFF7FFE068
ld_linux_x86_64.so.2:00007FFFF7FE4554 test edx, 2
ld_linux_x86_64.so.2:00007FFFF7FE455A jz short loc_7FFFF7FE456D
ld_linux_x86_64.so.2:00007FFFF7FE455C mov esi, 1
ld_linux_x86_64.so.2:00007FFFF7FE4561 mov edi, 5001h
ld_linux_x86_64.so.2:00007FFFF7FE4566 mov eax, 9Eh
ld_linux_x86_64.so.2:00007FFFF7FE456B syscall ; LINUX - sys_arch_prctl
ld_linux_x86_64.so.2:00007FFFF7FE456D
ld_linux_x86_64.so.2:00007FFFF7FE456D loc_7FFFF7FE456D: ; CODE XREF: ld_linux_x86_64.so.2:_dl_help+2EA↑j
ld_linux_x86_64.so.2:00007FFFF7FE456D mov edi, edx
ld_linux_x86_64.so.2:00007FFFF7FE456F and rsp, 0FFFFFFFFFFFFFFF0h
ld_linux_x86_64.so.2:00007FFFF7FE4573 call near ptr _dl_cet_setup_features
ld_linux_x86_64.so.2:00007FFFF7FE4578 mov rax, r12
ld_linux_x86_64.so.2:00007FFFF7FE457B mov rsp, r13
ld_linux_x86_64.so.2:00007FFFF7FE457E mov rdx, [rsp]
ld_linux_x86_64.so.2:00007FFFF7FE4582 mov rsi, rdx
ld_linux_x86_64.so.2:00007FFFF7FE4585 and rsp, 0FFFFFFFFFFFFFFF0h
ld_linux_x86_64.so.2:00007FFFF7FE4589 mov rdi, cs:_rtld_global
ld_linux_x86_64.so.2:00007FFFF7FE4590 lea rcx, [r13+rdx*8+10h]
ld_linux_x86_64.so.2:00007FFFF7FE4595 lea rdx, [r13+8]
ld_linux_x86_64.so.2:00007FFFF7FE4599 xor ebp, ebp
ld_linux_x86_64.so.2:00007FFFF7FE459B call near ptr _dl_init
ld_linux_x86_64.so.2:00007FFFF7FE45A0 lea rdx, _dl_fini
ld_linux_x86_64.so.2:00007FFFF7FE45A7 mov rsp, r13
ld_linux_x86_64.so.2:00007FFFF7FE45AA jmp r12
ld_linux_x86_64.so.2:00007FFFF7FE45AA ; ---
ld_linux_x86_64.so.2:00007FFFF7FE45AD db 0Fh
ld_linux_x86_64.so.2:00007FFFF7FE45AE db 1Fh
ld_linux_x86_64.so.2:00007FFFF7FE45AF db 0
ld_linux_x86_64.so.2:00007FFFF7FE45B0 dlmopen_doit db 0F3h
ld_linux_x86_64.so.2:00007FFFF7FE45B1 db 0Fh
ld_linux_x86_64.so.2:00007FFFF7FE45B2 db 1Eh
ld_linux_x86_64.so.2:00007FFFF7FE45B3 db 0FAh
UNKNOWN 00007FFFF7FE4540: ld_linux_x86_64.so.2:_dl_help+2D0 (Synchronized with RIP)
General registers
RAX 0000000000000000
RBX 0000000000000000
RCX 0000000000000000
RDX 0000000000000000
RSI 0000000000000000
RDI 0000000000000000
RBP 0000000000000000
RSP 00007FFFFFFFE5A0 [stack]:00007FFFFFFFE5A0
RIP 00007FFFF7FE4540 ld_linux_x86_64.so.2:_dl_help+2D0
R8 0000000000000000
R9 0000000000000000
ID
VIP
VIF
AC
VM
RF
NT
IOPL
OF
DF
IF
TF
Modules
Path
Base
/home/twinklestar03/Documents/lab01-helloworld/lab01
0000555555554000
[vdso]
00007FFFF7FC3000
/usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
00007FFFF7FC5000
Threads
Decimal
Hex
State
Name
21198
52CE
Ready
lab01
Hex View-1
00005555555540A0 1C 00 00 00 00 00 00 00 01 00 00 00 00 00 00 00 ....
00005555555540B0 01 00 00 00 04 00 00 00 00 00 00 00 00 00 00 00 ....
00005555555540C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
00005555555540D0 D8 06 00 00 00 00 00 00 D8 06 00 00 00 00 00 00 ....
00005555555540E0 00 10 00 00 00 00 00 00 01 00 00 00 05 00 00 00 ....
00005555555540F0 00 10 00 00 00 00 00 00 00 10 00 00 00 00 00 00 ....
0000555555554100 00 10 00 00 00 00 00 00 8D 02 00 00 00 00 00 00 ....
0000555555554110 8D 02 00 00 00 00 00 00 00 10 00 00 00 00 00 00 ....
0000555555554120 01 00 00 00 04 00 00 00 00 20 00 00 00 00 00 00 ....
0000555555554130 00 20 00 00 00 00 00 00 00 20 00 00 00 00 00 00 ....
0000555555554140 4C 02 00 00 00 00 00 00 4C 02 00 00 00 00 00 00 ....
0000555555554150 00 10 00 00 00 00 00 00 01 00 00 00 06 00 00 00 ....
0000555555554160 A8 2D 00 00 00 00 00 00 A8 3D 00 00 00 00 00 00 ....
0000555555554170 A8 3D 00 00 00 00 00 00 70 02 00 00 00 00 00 00 ....
000000A0 0000555555554A0: LOAD:0000555555554A0
Stack view
00007FFFFFFFE5A0 0000000000000001
00007FFFFFFFE5A8 00007FFFFFFFE8C0 [stack]:00007FFFFFFFE8C0
00007FFFFFFFE5B0 0000000000000000
00007FFFFFFFE5B8 00007FFFFFFFE8F5 [stack]:00007FFFFFFFE8F5
00007FFFFFFFE5C0 00007FFFFFFFE90E [stack]:00007FFFFFFFE90E
00007FFFFFFFE5C8 00007FFFFFFFE91F [stack]:00007FFFFFFFE91F
00007FFFFFFFE5D0 00007FFFFFFFE935 [stack]:00007FFFFFFFE935
00007FFFFFFFE5D8 00007FFFFFFFE9A7 [stack]:00007FFFFFFFE9A7
00007FFFFFFFE5E0 00007FFFFFFFE9B7 [stack]:00007FFFFFFFE9B7
00007FFFFFFFE5E8 00007FFFFFFFE9CA [stack]:00007FFFFFFFE9CA
00007FFFFFFFE5F0 00007FFFFFFFE9E9 [stack]:00007FFFFFFFE9E9
00007FFFFFFFE5F8 00007FFFFFFFEA05 [stack]:00007FFFFFFFEA05
00007FFFFFFFE600 00007FFFFFFFEA18 [stack]:00007FFFFFFFEA18
00007FFFFFFFE608 00007FFFFFFFEA7E [stack]:00007FFFFFFFEA7E
UNKNOWN 00007FFFFFFFE5A0: [stack]:00007FFFFFFFE5A0 (Synchronized with RSP)
Output
Command "JumpOpXref" failed
Found a valid interpeter in /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2, will report shared library events!
7FFFF7FC3000: loaded [vdso]
7FFFF7FC5000: loaded /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
Rebasing program to 0x0000555555554000...
555555554000: process /home/twinklestar03/Documents/lab01-helloworld/lab01 has started (pid=21184)
7FFFF7C00000: loaded /usr/lib/x86_64-linux-gnu/libc.so.6
Debugger: process has exited (exit code 0)
Found a valid interpeter in /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2, will report shared library events!
7FFFF7FC3000: loaded [vdso]
7FFFF7FC5000: loaded /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
555555554000: process /home/twinklestar03/Documents/lab01-helloworld/lab01 has started (pid=21198)
DWARF: Looking for GNU DWARF file at "/usr/lib/debug/.build-id/3e/81740f816ee1521a87e439c16ebfde46f147a5.debug"... found!
DWARF: Found DWARF file "/usr/lib/debug/.build-id/3e/81740f816ee1521a87e439c16ebfde46f147a5.debug" corresponding to "/usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2"
IDC
AU: idle Down Disk: 108GB

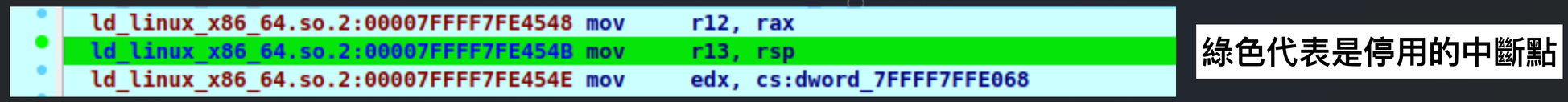

IDA View-RIP, General registers, Modules, Threads, Hex View-1, Stack view
Local Types
IDA View-RIP
ld_linux_x86_64.so.2:00007FFFF7FE453D db 0Fh
ld_linux_x86_64.so.2:00007FFFF7FE453E db 1Fh
ld_linux_x86_64.so.2:00007FFFF7FE453F db 0
ld_linux_x86_64.so.2:00007FFFF7FE4540 ; ---
RIP
ld_linux_x86_64.so.2:00007FFFF7FE4540 mov rdi, rsp
ld_linux_x86_64.so.2:00007FFFF7FE4543 call near ptr _dl_start
ld_linux_x86_64.so.2:00007FFFF7FE4548 mov r12, rax
ld_linux_x86_64.so.2:00007FFFF7FE454B mov r13, rsp
ld_linux_x86_64.so.2:00007FFFF7FE454E mov edx, cs:dword_7FFFF7FFE068
ld_linux_x86_64.so.2:00007FFFF7FE4554 test edx, 2
ld_linux_x86_64.so.2:00007FFFF7FE455A jz short loc_7FFFF7FE456D
ld_linux_x86_64.so.2:00007FFFF7FE455C mov esi, 1
ld_linux_x86_64.so.2:00007FFFF7FE4561 mov edi, 5001h
ld_linux_x86_64.so.2:00007FFFF7FE4566 mov eax, 9Eh
ld_linux_x86_64.so.2:00007FFFF7FE456B syscall ; LINUX - sys_arch_prctl
ld_linux_x86_64.so.2:00007FFFF7FE456D
ld_linux_x86_64.so.2:00007FFFF7FE456D loc_7FFFF7FE456D: ; CODE XREF: ld_linux_x86_64.so.2:_dl_help+2EA↑j
ld_linux_x86_64.so.2:00007FFFF7FE456D mov edi, edx
ld_linux_x86_64.so.2:00007FFFF7FE456F and rsp, 0FFFFFFFFFFFFFFF0h
ld_linux_x86_64.so.2:00007FFFF7FE4573 call near ptr _dl_cet_setup_features
ld_linux_x86_64.so.2:00007FFFF7FE4578 mov rax, r12
ld_linux_x86_64.so.2:00007FFFF7FE457B mov rsp, r13
ld_linux_x86_64.so.2:00007FFFF7FE457E mov rdx, [rsp]
ld_linux_x86_64.so.2:00007FFFF7FE4582 mov rsi, rdx
ld_linux_x86_64.so.2:00007FFFF7FE4585 and rsp, 0FFFFFFFFFFFFFFF0h
ld_linux_x86_64.so.2:00007FFFF7FE4589 mov rdi, cs:_rtld_global
ld_linux_x86_64.so.2:00007FFFF7FE4590 lea rcx, [r13+rdx*8+10h]
ld_linux_x86_64.so.2:00007FFFF7FE4595 lea rdx, [r13+8]
ld_linux_x86_64.so.2:00007FFFF7FE4599 xor ebp, ebp
ld_linux_x86_64.so.2:00007FFFF7FE459B call near ptr _dl_init
ld_linux_x86_64.so.2:00007FFFF7FE45A0 lea rdx, _dl_fini
ld_linux_x86_64.so.2:00007FFFF7FE45A7 mov rsp, r13
ld_linux_x86_64.so.2:00007FFFF7FE45AA jmp r12
ld_linux_x86_64.so.2:00007FFFF7FE45AA ; ---
ld_linux_x86_64.so.2:00007FFFF7FE45AD db 0Fh
ld_linux_x86_64.so.2:00007FFFF7FE45AE db 1Fh
ld_linux_x86_64.so.2:00007FFFF7FE45AF db 0
ld_linux_x86_64.so.2:00007FFFF7FE45B0 dlmopen_doit db 0F3h
ld_linux_x86_64.so.2:00007FFFF7FE45B1 db 0Fh
ld_linux_x86_64.so.2:00007FFFF7FE45B2 db 1Eh
ld_linux_x86_64.so.2:00007FFFF7FE45B3 db 0FAh
UNKNOWN 00007FFFF7FE4540: ld_linux_x86_64.so.2:_dl_help+2D0 (Synchronized with RIP)
Disassembly
General registers
RAX 0000000000000000
RBX 0000000000000000
RCX 0000000000000000
RDX 0000000000000000
RSI 0000000000000000
RDI 0000000000000000
RBP 0000000000000000
RSP 00007FFFFFFFE5A0 [stack]:00007FFF
RIP 00007FFFF7FE4540 ld_linux_x86_64.so.2:_dl_help+2D0
R8 0000000000000000
R9 0000000000000000
ID
VIP
VIF
AC
VM
RF
NT
IOPL
OF
DF
IF
TF
Registers
Modules
Path
Base
/home/twinklestar03/Documents/lab01-helloworld/lab01
0000555555554000
[vdso]
00007FFFF7FC3000
/usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
00007FFFF7FC5000
Module Base
Threads
Decimal
Hex
State
Name
21198
52CE
Ready
lab01
Thread
Hex View-1
0000555555554000 1C 00 00 00 00 00 00 00 01 00 00 00 00 00 00 00 ....
0000555555554010 01 00 00 00 04 00 00 00 00 00 00 00 00 00 00 00 ....
0000555555554020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
0000555555554030 D8 06 00 00 00 00 00 00 D8 06 00 00 00 00 00 00 ....
0000555555554040 00 10 00 00 00 00 00 00 01 00 00 00 05 00 00 00 ....
0000555555554050 00 10 00 00 00 00 00 00 00 10 00 00 00 00 00 00 ....
0000555555554100 00 10 00 00 00 00 00 00 8D 02 00 00 00 00 00 00 ....
0000555555554110 8D 02 00 00 00 00 00 00 00 10 00 00 00 00 00 00 ....
0000555555554120 01 00 00 00 04 00 00 00 00 20 00 00 00 00 00 00 ....
0000555555554130 00 20 00 00 00 00 00 00 00 20 00 00 00 00 00 00 . ...
0000555555554140 4C 02 00 00 00 00 00 00 4C 02 00 00 00 00 00 00 L.......L.......
0000555555554150 00 10 00 00 00 00 00 00 01 00 00 00 06 00 00 00 ....
0000555555554160 A8 2D 00 00 00 00 00 00 A8 3D 00 00 00 00 00 00 .-.......=......
0000555555554170 A8 3D 00 00 00 00 00 00 70 02 00 00 00 00 00 00 .=......p.......
000000A0 0000555555554540A0: LOAD:0000555555554540A0
Hex dump
Stack view
00007FFFFFFFE5A0 0000000000000001
00007FFFFFFFE5A8 00007FFFFFFFE8C0 [stack]:00007FFFFFFFE8C0
00007FFFFFFFE5B0 0000000000000000
00007FFFFFFFE5B8 00007FFFFFFFE8F5 [stack]:00007FFFFFFFE8F5
00007FFFFFFFE5C0 00007FFFFFFFE90E [stack]:00007FFFFFFFE90E
00007FFFFFFFE5C8 00007FFFFFFFE91F [stack]:00007FFFFFFFE91F
00007FFFFFFFE5D0 00007FFFFFFFE935 [stack]:00007FFFFFFFE935
00007FFFFFFFE5D8 00007FFFFFFFE9A7 [stack]:00007FFFFFFFE9A7
00007FFFFFFFE5E0 00007FFFFFFFE9B7 [stack]:00007FFFFFFFE9B7
00007FFFFFFFE5E8 00007FFFFFFFE9CA [stack]:00007FFFFFFFE9CA
00007FFFFFFFE5F0 00007FFFFFFFE9E9 [stack]:00007FFFFFFFE9E9
00007FFFFFFFE5F8 00007FFFFFFFEA05 [stack]:00007FFFFFFFEA05
00007FFFFFFFE600 00007FFFFFFFEA18 [stack]:00007FFFFFFFEA18
00007FFFFFFFE608 00007FFFFFFFEA7E [stack]:00007FFFFFFFEA7E
UNKNOWN 00007FFFFFFFE5A0: [stack]:00007FFFFFFFE5A0 (Synchronized with RSP)
Stack
Output
Command "JumpOpXref" failed
Found a valid interpeter in /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2, will report shared library events!
7FFFF7FC3000: loaded [vdso]
7FFFF7FC5000: loaded /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
Rebasing program to 0x0000555555554000...
555555554000: process /home/twinklestar03/Documents/lab01-helloworld/lab01 has started (pid=21184)
7FFFF7C00000: loaded /usr/lib/x86_64-linux-gnu/libc.so.6
Debugger: process has exited (exit code 0)
Found a valid interpeter in /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2, will report shared library events!
7FFFF7FC3000: loaded [vdso]
7FFFF7FC5000: loaded /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
555555554000: process /home/twinklestar03/Documents/lab01-helloworld/lab01 has started (pid=21198)
DWARF: Looking for GNU DWARF file at "/usr/lib/debug/.build-id/3e/81740f816ee1521a87e439c16ebfde46f147a5.debug"... found!
DWARF: Found DWARF file "/usr/lib/debug/.build-id/3e/81740f816ee1521a87e439c16ebfde46f147a5.debug" corresponding to "/usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2"
Log
IDC
AU: idle Down Disk: 108GB


Debugger
Options
Windows
Help
Quick debug view Ctrl+2
Debugger windows
Breakpoints
Watches
Tracing
Continue process F9
Attach to process...
Process options...
Pause process
Terminate process Ctrl+F2
Detach from process
Refresh memory
Take memory snapshot
Step into F7
Step over F8
Run until return Ctrl+F7
Run to cursor F4
Switch to source
Use source-level debugging
Open source file...
Debugger options...
Switch debugger...
007FFFF7FE4590 lea rcx, [r13+rdx*8+10h]

執行功能在選單中

下斷點按這個藍色點


IDA View-RIP
ld_linux_x86_64.so.2:00007FFFF7FE453D db 0Fh
ld_linux_x86_64.so.2:00007FFFF7FE453E db 1Fh
ld_linux_x86_64.so.2:00007FFFF7FE453F db 0
ld_linux_x86_64.so.2:00007FFFF7FE4540 ; ----------
RIP
ld_linux_x86_64.so.2:00007FFFF7FE4540 mov rdi, rsp
ld_linux_x86_64.so.2:00007FFFF7FE4543 call near ptr _dl_start
ld_linux_x86_64.so.2:00007FFFF7FE4548 mov r12, rax
ld_linux_x86_64.so.2:00007FFFF7FE454B mov r13, rsp
ld_linux_x86_64.so.2:00007FFFF7FE454E mov edx, cs:dword_7FFFF7FFE068
ld_linux_x86_64.so.2:00007FFFF7FE4554 test edx, 2
ld_linux_x86_64.so.2:00007FFFF7FE455A jz short loc_7FFFF7FE456D
ld_linux_x86_64.so.2:00007FFFF7FE455C mov esi, 1
藍色代表當前 PC
紅色代表有作用的中斷點


ld_linux_x86_64.so.2:00007FFFF7FE4548 mov r12, rax
ld_linux_x86_64.so.2:00007FFFF7FE454B mov r13, rsp
ld_linux_x86_64.so.2:00007FFFF7FE454E mov edx, cs:dword_7FFFF7FFE068
綠色代表是停用的中斷點

DEMO

Binary Patching

Binary Patching

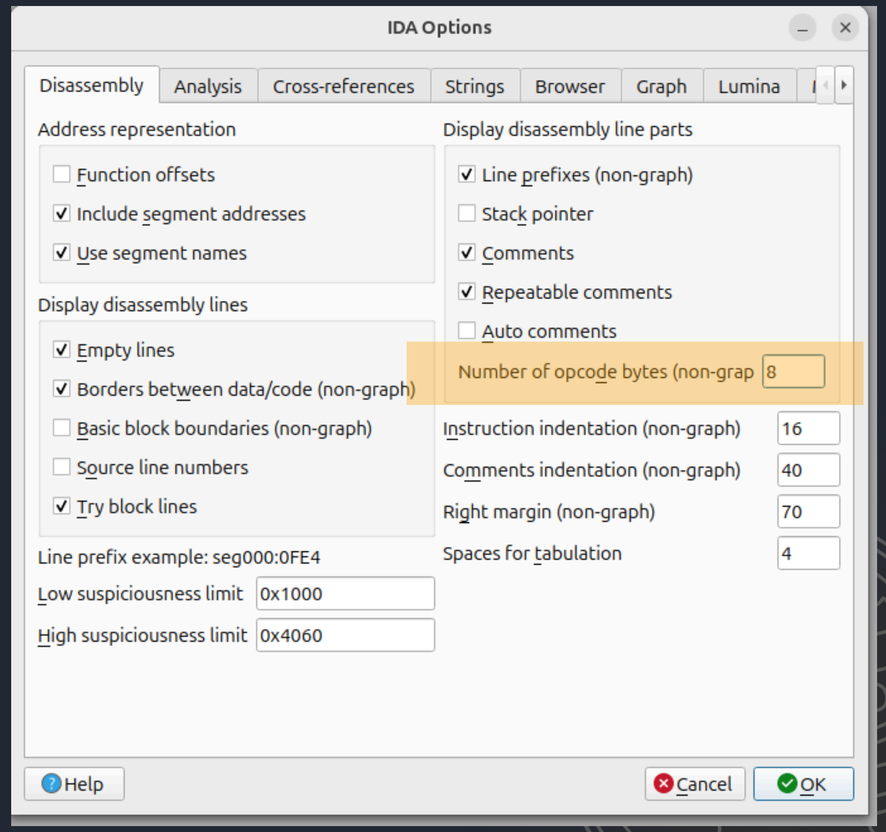
- 有時候 executable 不是執行我們想要的行為
  - 驗證資訊的方式、傳送的資料...
- 我們通過 Patch 去修改 opcode 去改變 executable 的行為
  - 例子：把 BNE 變成 BE，改變 if 的判斷
  - Note: BNE (Branch Not Equal), BE (Branch Equal)

Binary Patching

- 通常在 IDA 裡面做 patch 都是做比較少量的
  - 不會大幅度更動 executable 的大小
- 如果要大規模 patch 那就會變成 Binary Rewrite
  - 用別的工具會比較好做

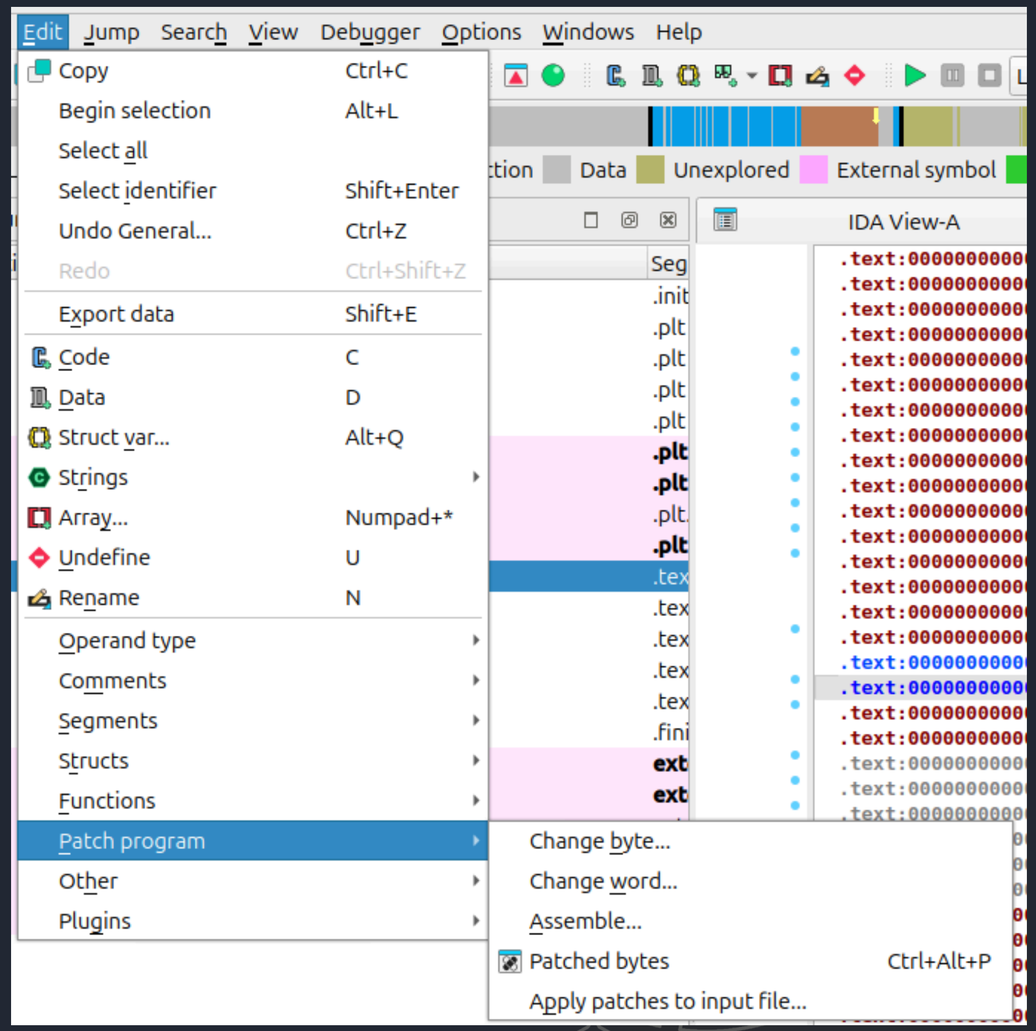
Binary Patching

- 建議把 “Number of opcode” 調高
- 方便觀察 opcode



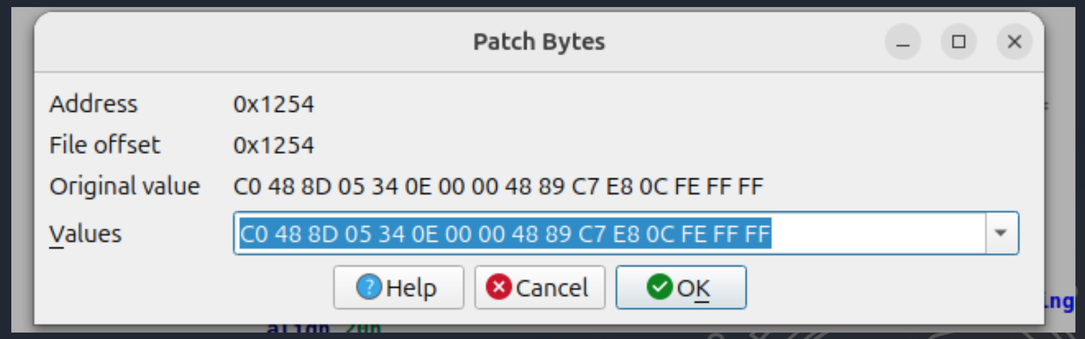
Binary Patching

- IDA 做 Patch 的選項
  - Edit > Patch Program



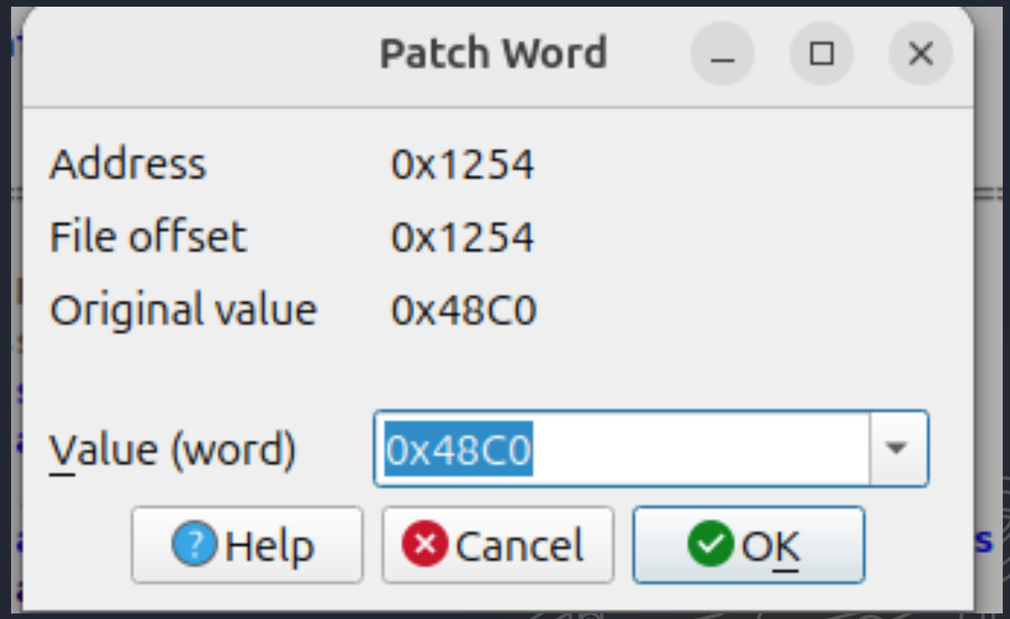
Binary Patching

- Change Byte



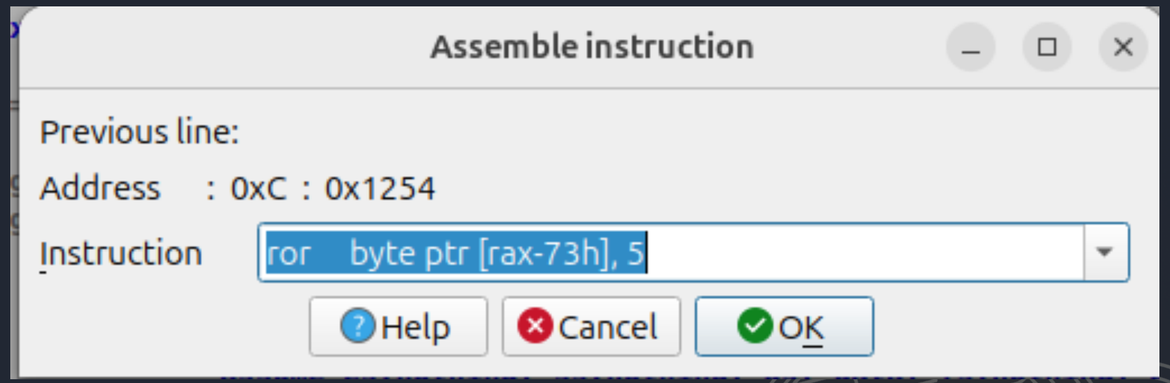
Binary Patching

- Change Word
  - 數字的形式而不是 Bytes



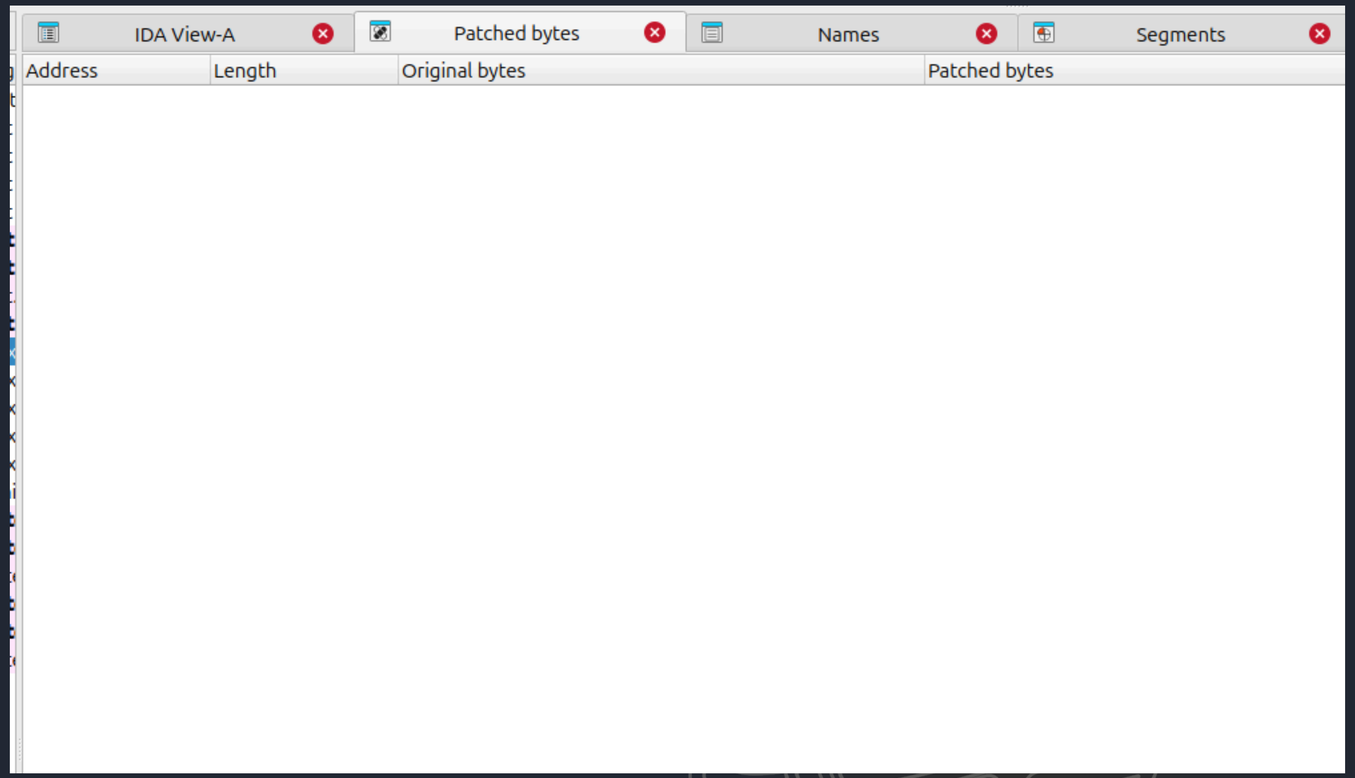
Binary Patching

- Assemble
- 直接寫 Assembly



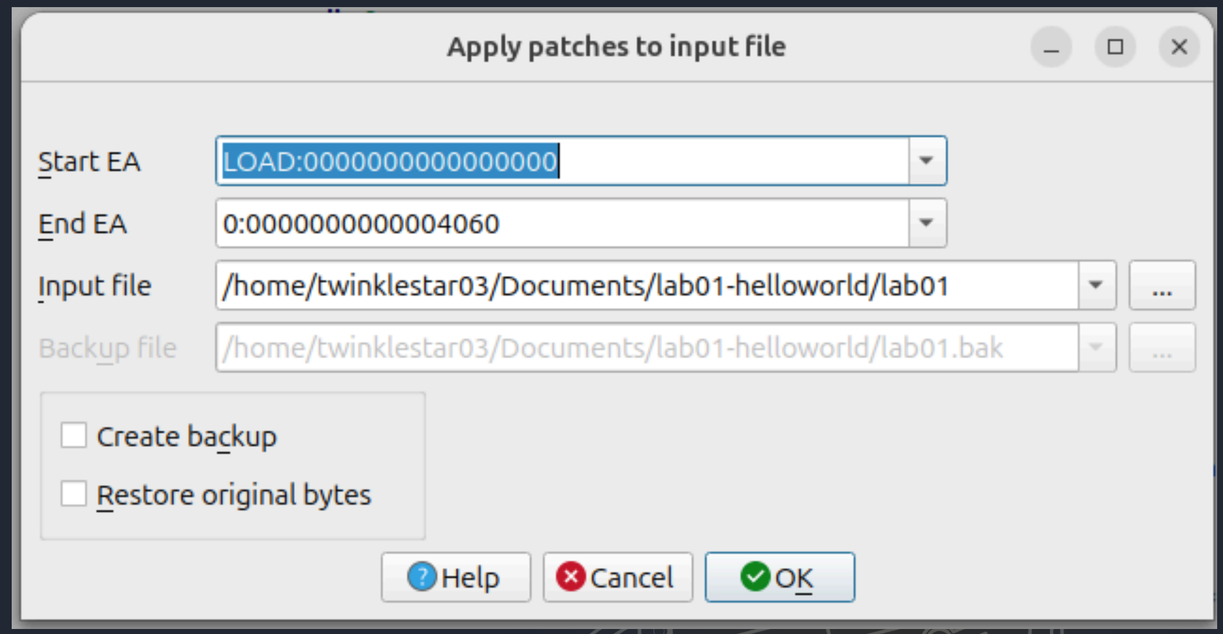
Binary Patching

- Patched bytes
- 用來看已修改資訊的 Subview



Binary Patching

- Apply patches
  - 將更動寫回檔案
- 建議將 “Create backup” 打勾
  - 會建立一個原檔案備份



Lab: Patch & Debug

Lab: Patch #1

- 目標
  - lab01.idb 將 main 中怪異的 JMP Patch 成 NOP (0x90)
  - 進而將 decompile 修好
    - Signature: EB FF C0 > 90 90 90

Lab: patchme

- 目標
  - 將 main 的判斷是 patch 成導向 win()
  - CMP 本身其實是在做減法，所以後面的 JNZ 就是在判斷結果是不是 0
    - JNZ (Jump Not Zero)
  - 你會需要的 Instruction
    - JZ (Jump Zero)

Structure Recovery

Data & Pointer

Type

- 在記憶體內存放了許多資料
  - 存放資料是有規則可循的
- 理解成每個型態都有自己的大小，程式會根據大小去堆放這些資料

Type

- C 語言的預設 type: short, int, long long int …
  - 上述關鍵字會因平台不同而導致大小也不同
  - Example: x64 底下的 int 是 4 byte，但 bc45 上的 int 是 2 byte
- 更好的做法，使用帶著長度資訊的 type
  - [u]int[8/16/32/64]_t

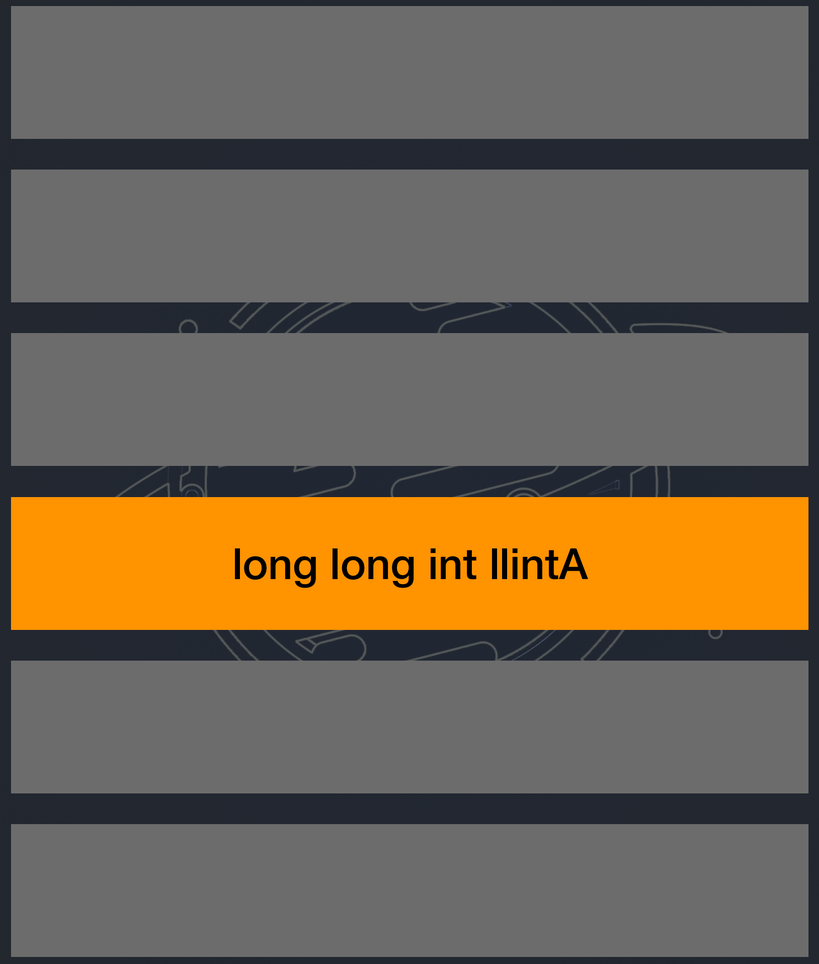
Alignment

- 記憶體對齊
- 資料開始的位址必須為 x 的倍數
- 什麼情況下會遇到 Alignment?
  - 資料型態的長度不一時，field 之間會留下空間
  - 雖然碎片化程度變高，但不用浪費空間紀錄每個 field 的 offset

一塊記憶體空間

存放一個 long long int

long long int 是 8 bytes



接著存放一個 int

一個 int 是 4 bytes

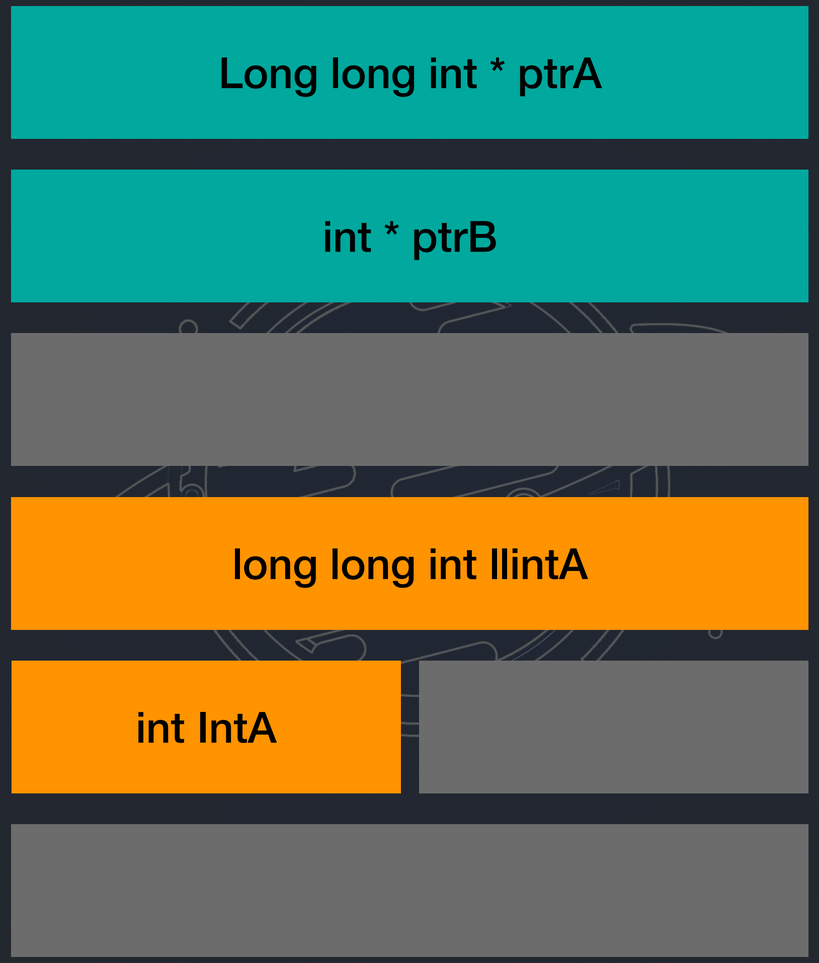
如果這時候有一個 char 想要塞到 llintA 前呢？

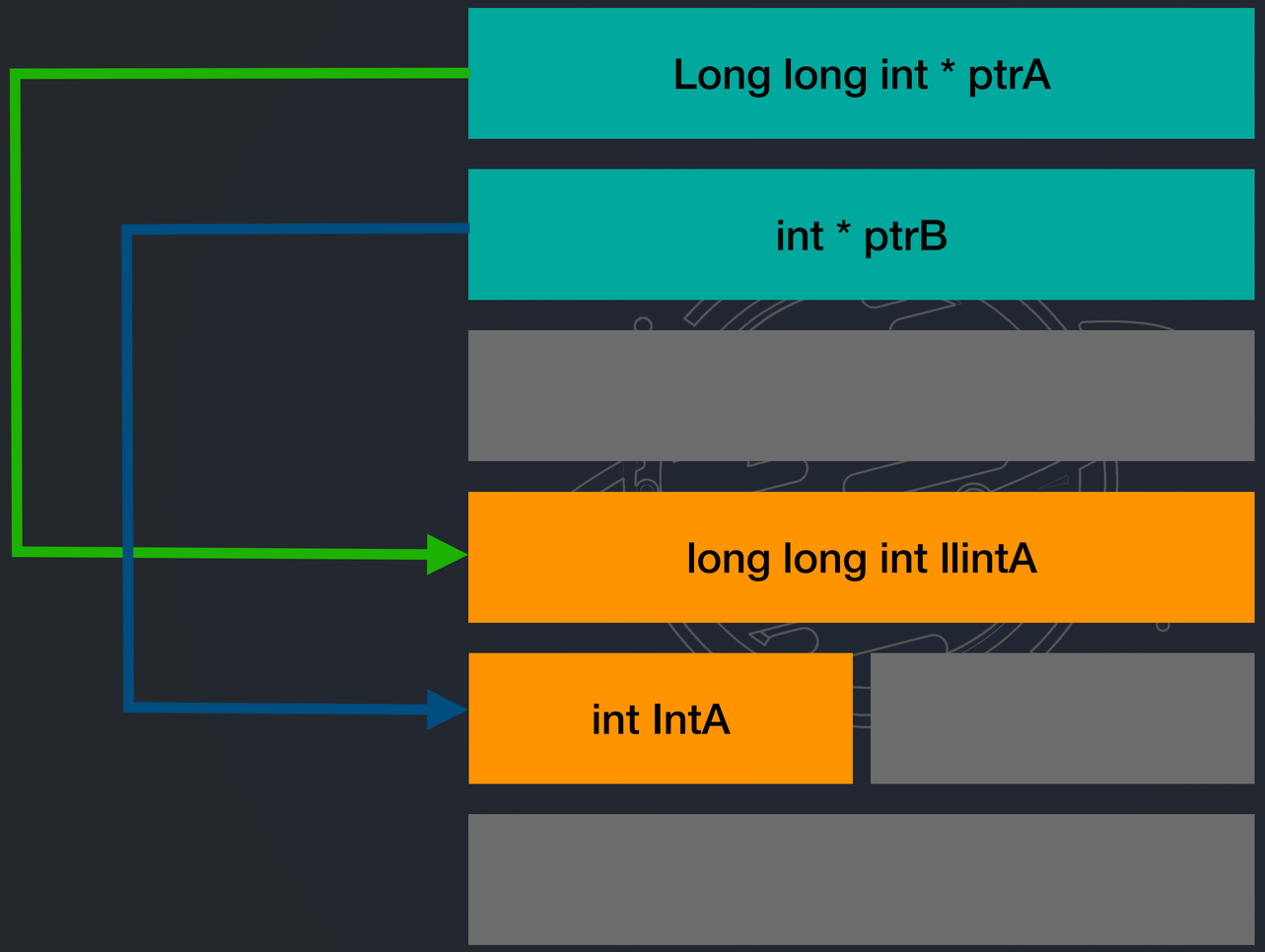
Alignment!

Pointer

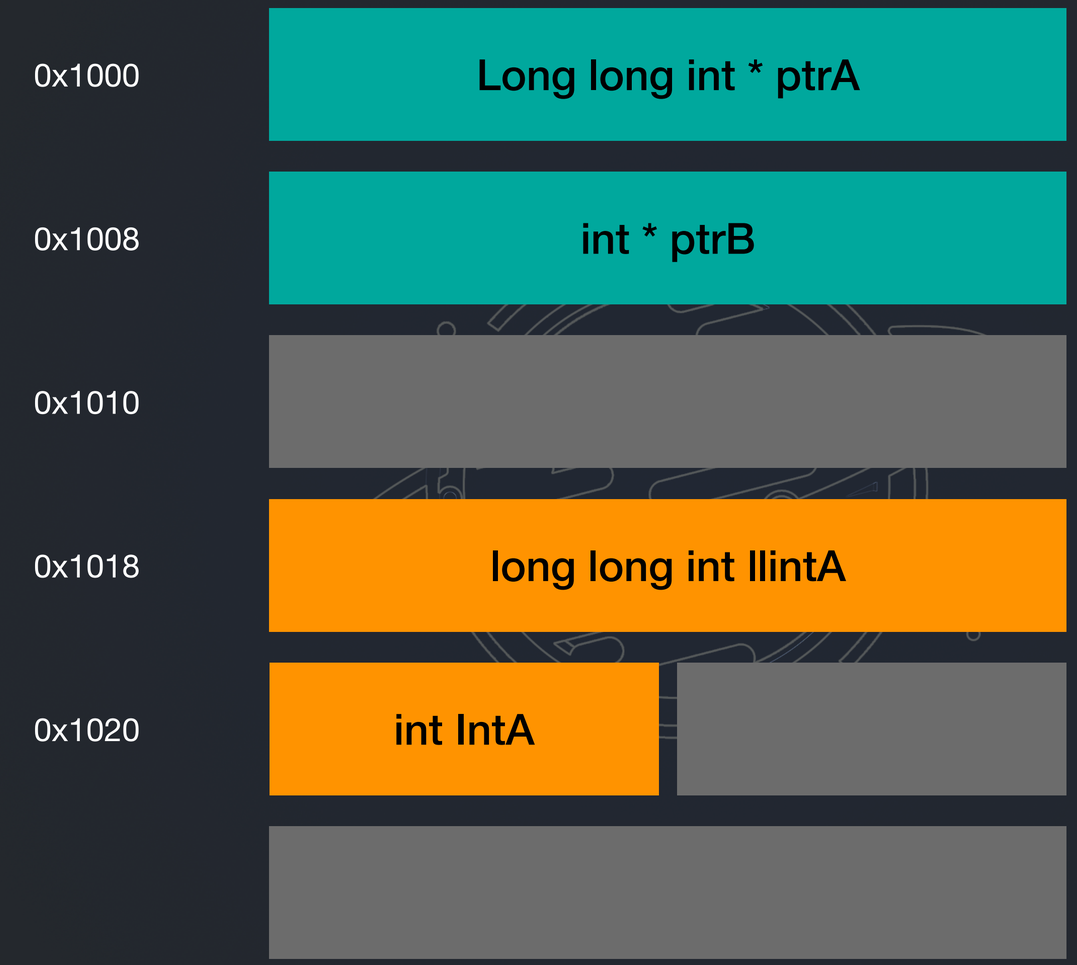
- Pointer 是指向一個記憶體空間的型態
  - 指標的大小無論型別都一樣大
  - 根據型別的不同，可以指向不同型別的資料

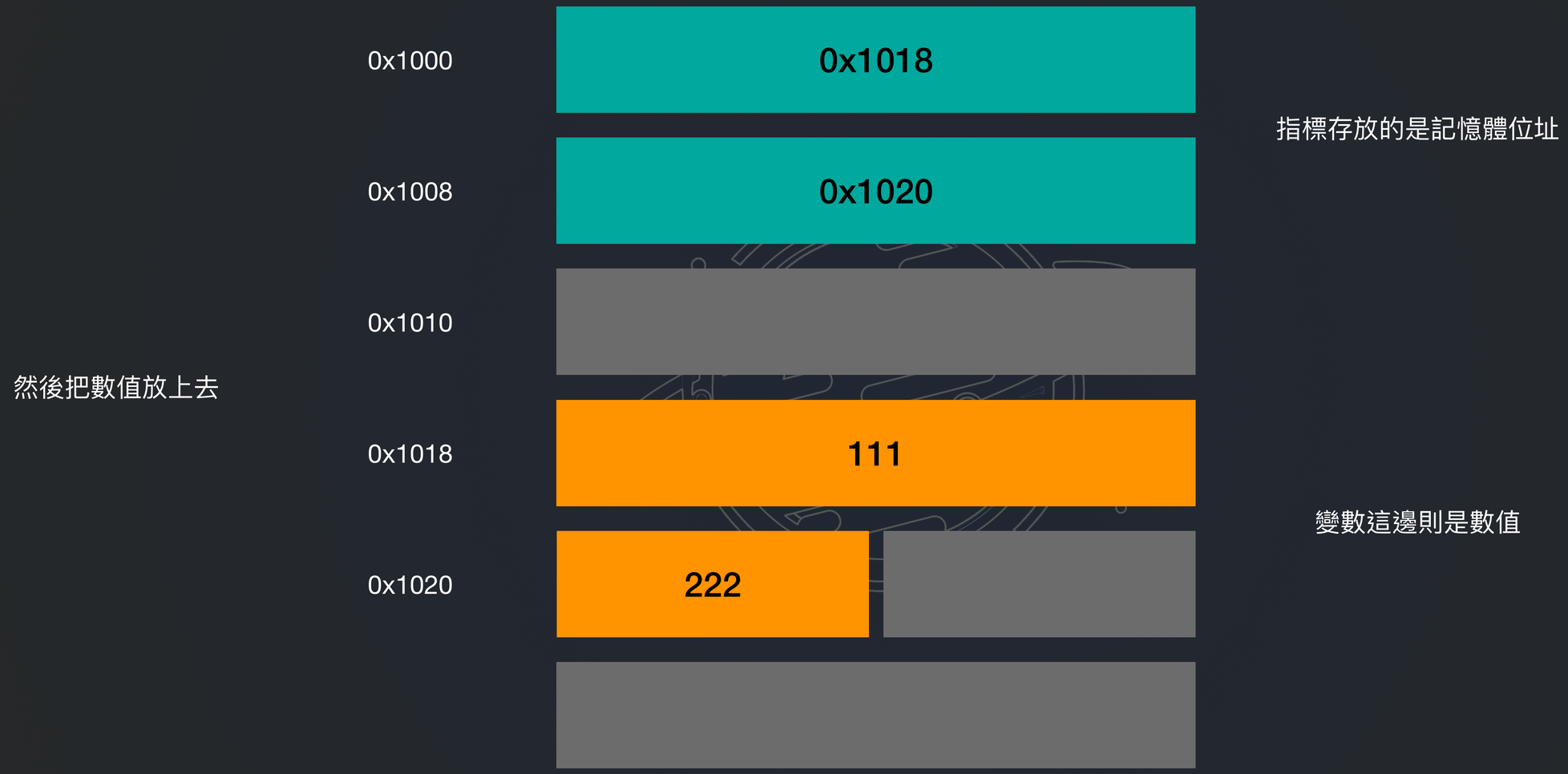
有兩指標分別指向前面創造出的兩個變數

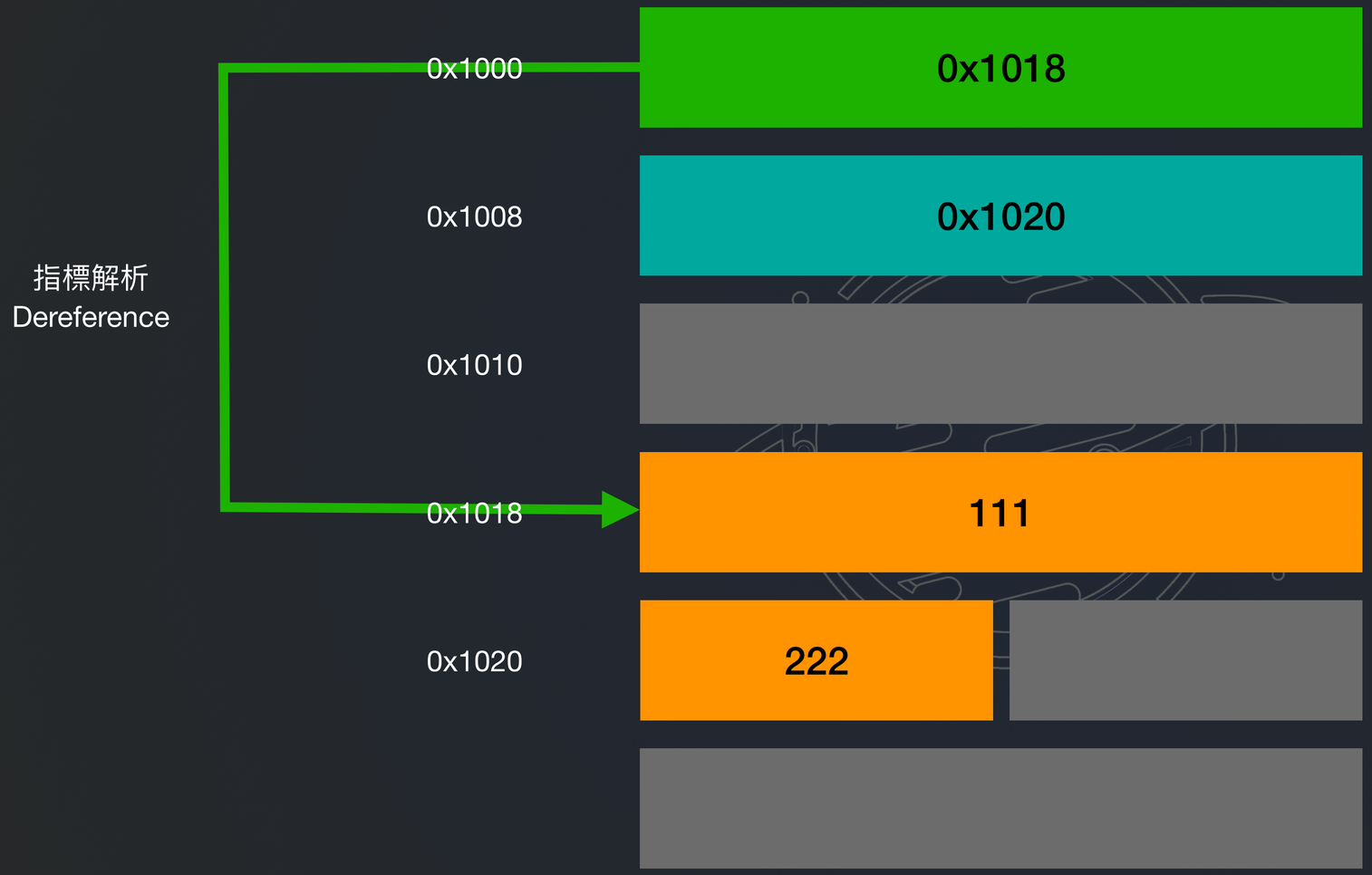


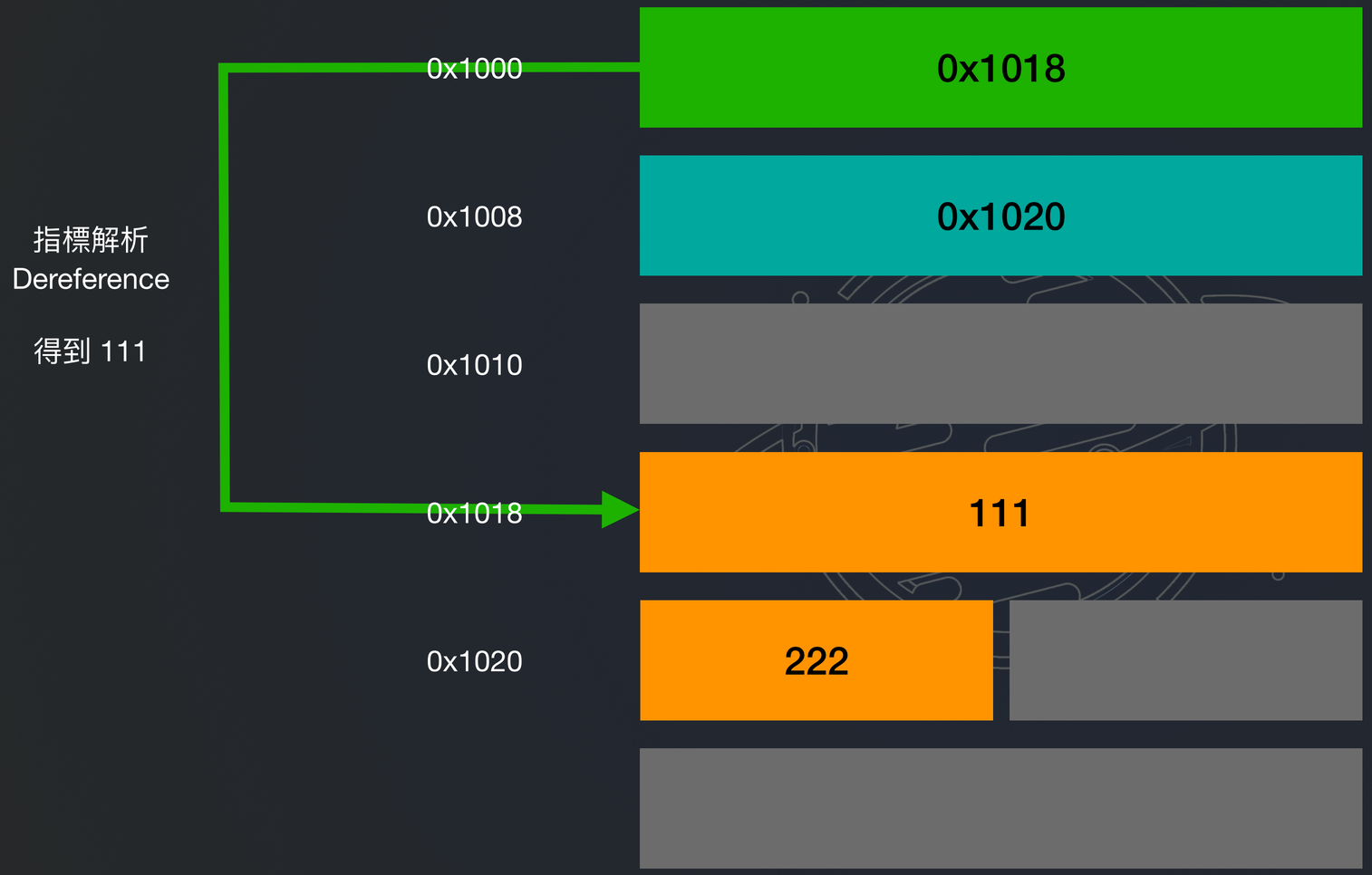

Long long int * ptrA
int * ptrB
long long int llintA
int IntA

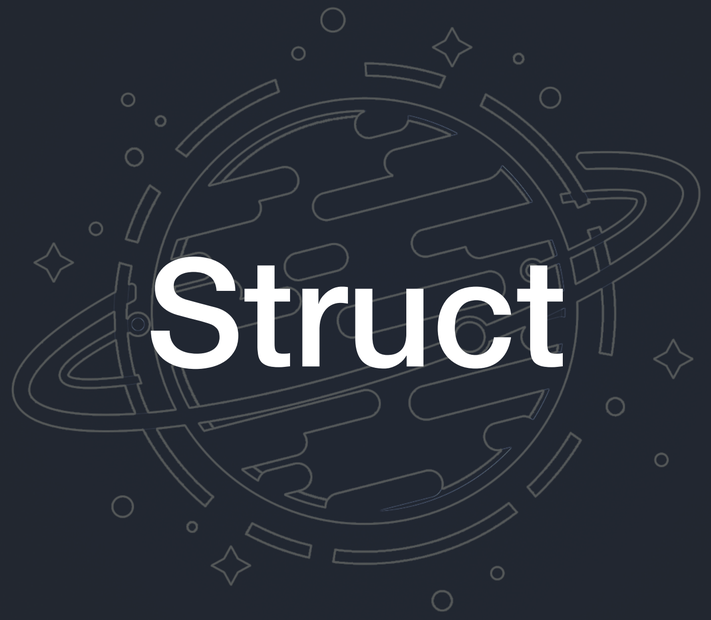
現在放入 “記憶體位址”的概念




0x1000
0x1018
指標存放的是記憶體位址
0x1008
0x1020
0x1010
然後把數值放上去
0x1018
111
變數這邊則是數值
0x1020
222


0x1000
0x1018
0x1008
0x1020
指標解析
Dereference
0x1010
0x1018
111
0x1020
222


指標解析
Dereference
得到 111
0x1000
0x1008
0x1010
0x1018
0x1020
0x1018
0x1020
111
222



Struct

Struct Recovery

- Structure 的結構重點
  - 連續的記憶體空間
  - 相同的 Offset 們重複出現
  - 用指標將資料飛來飛去

Struct Recovery

- 如何從 Offset 回推回去 Source Code 勒
  - IDA Auto Create Struct
  - 人工自己看

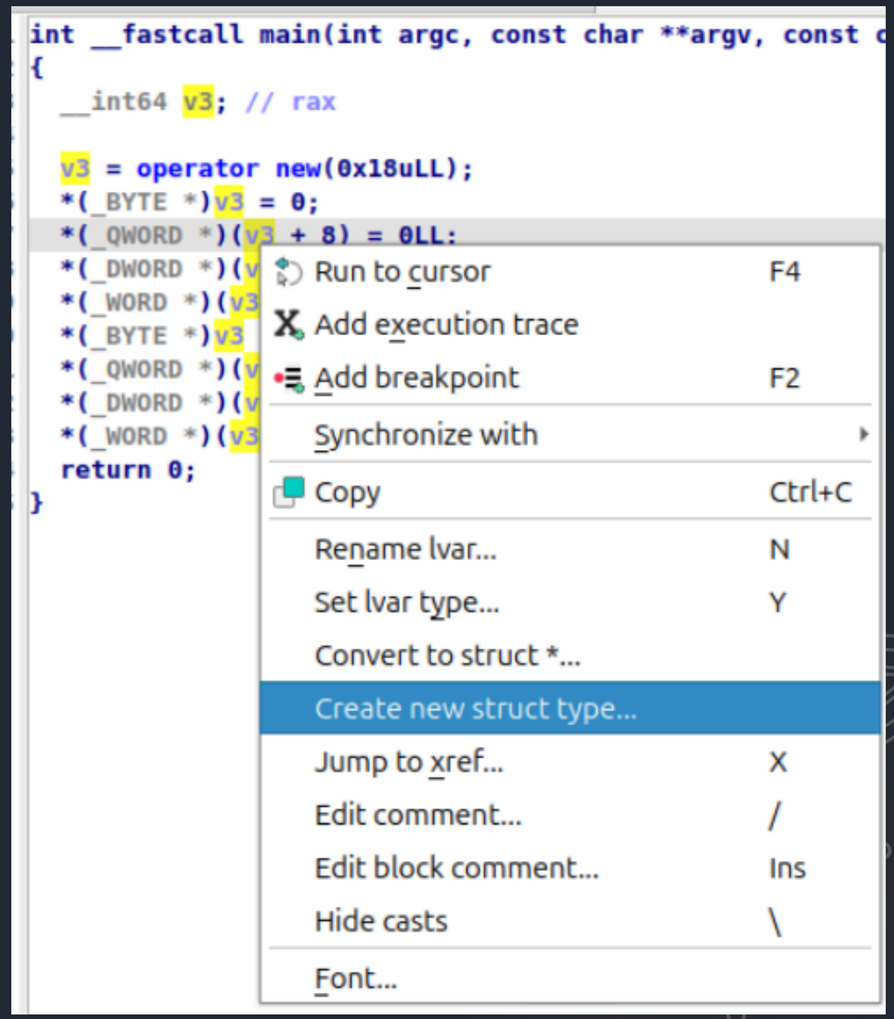
Struct Recovery

- Pointer to structure 的長相

```
*(_BYTE *)v3 = 0;
*(_QWORD *)(v3 + 8) = 0LL;
*(_DWORD *)(v3 + 16) = 0;
*(_WORD *)(v3 + 20) = 0;
*(_BYTE *)v3 = 71;
*(_QWORD *)(v3 + 8) = 0xDC11352BEEFLL;
*(_DWORD *)(v3 + 16) = 4919;
*(_WORD *)(v3 + 20) = -8531;
return 0;
```

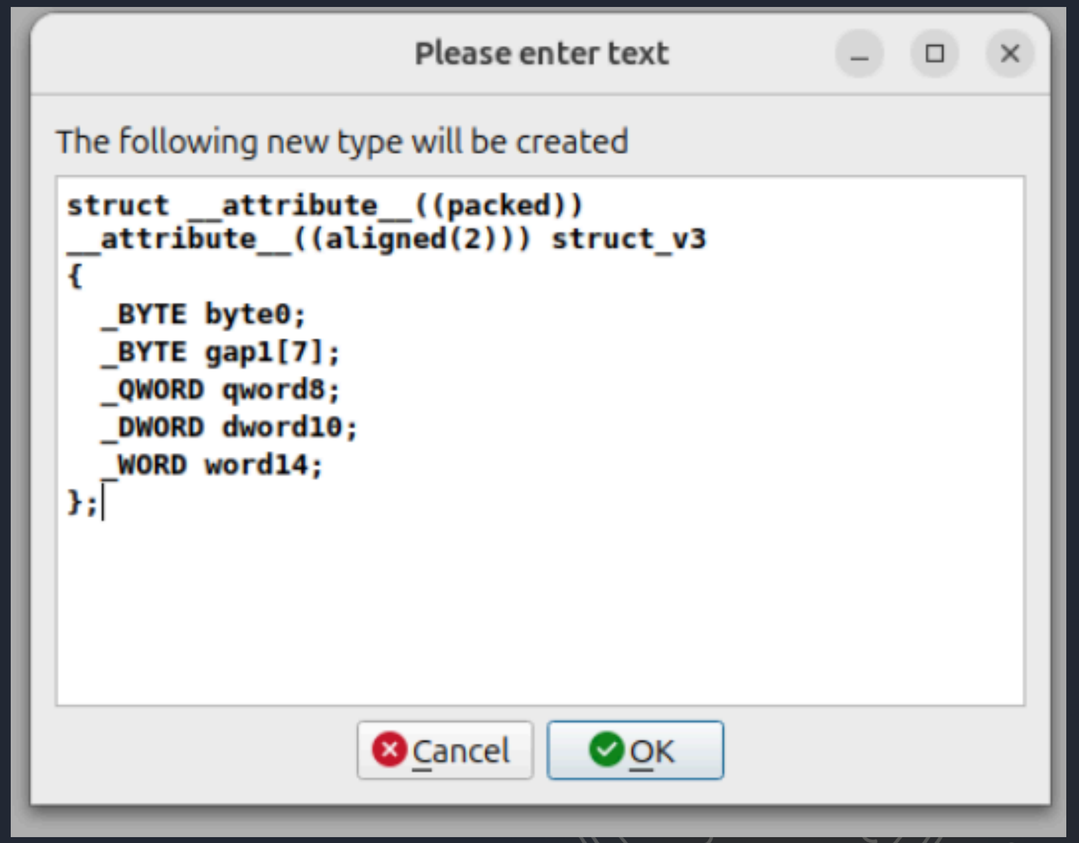
Struct Recovery

- 在 Decompile view 上選擇 pointer
- 右鍵 Create new struct type



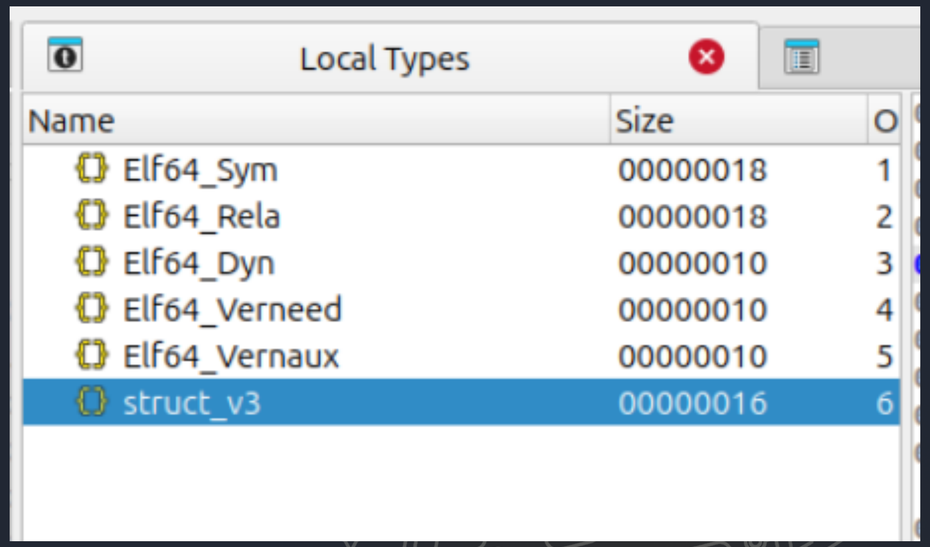
Struct Recovery

- IDA 會幫你猜測每個 Field 的 Type
  - 不保證正確
- 通常是個好的起始點



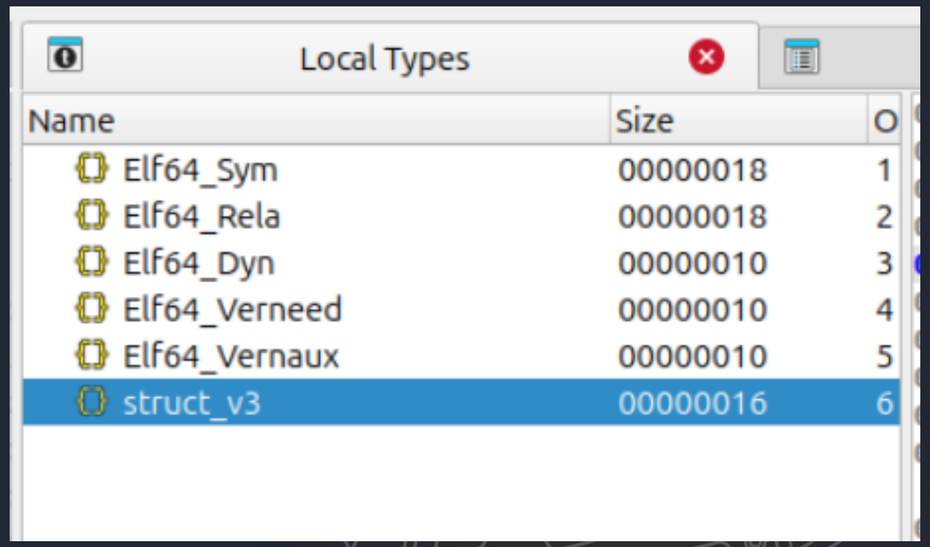
Local Types

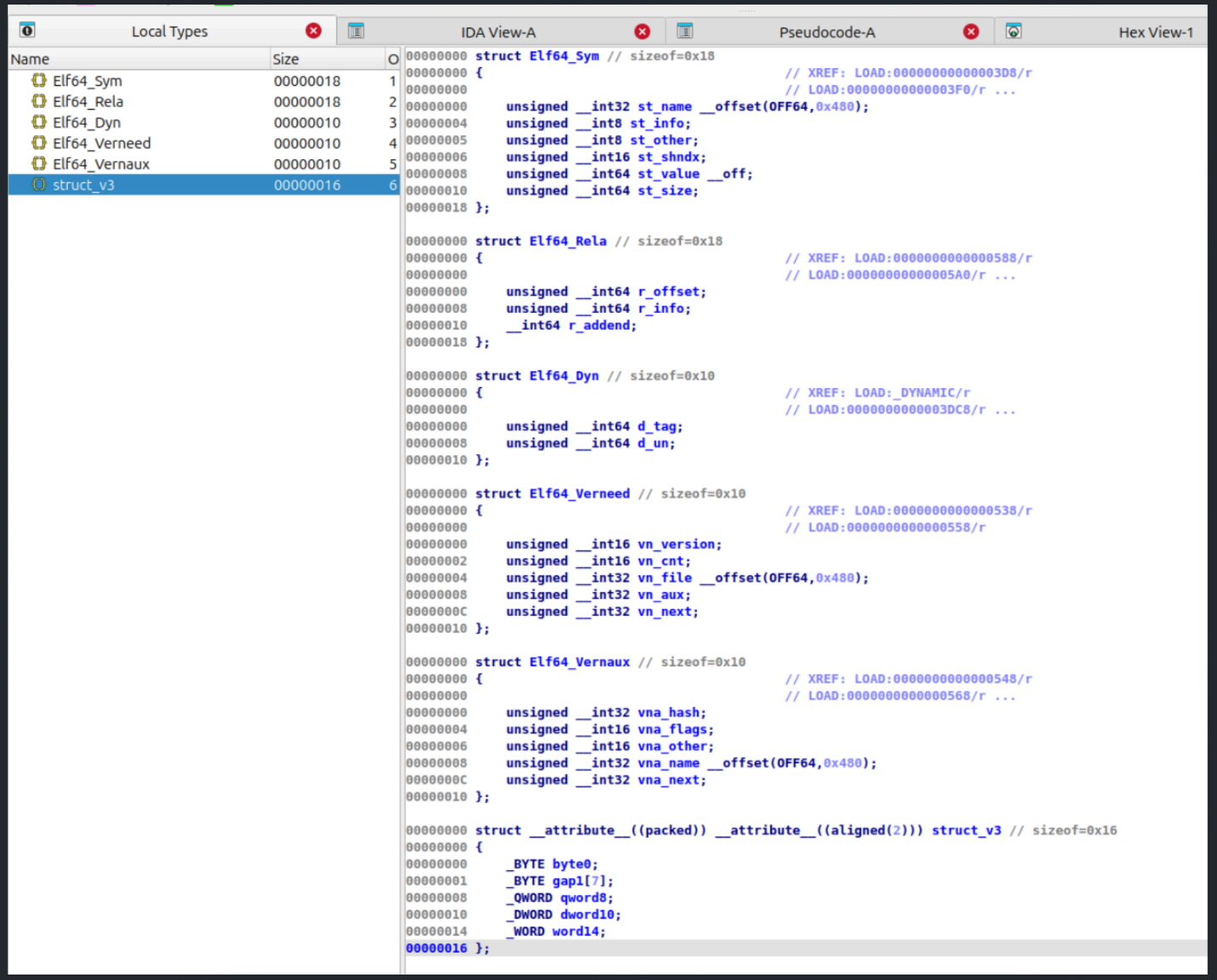
- 有一個 Subview 叫做 Local Types
  - 存放著 IDA 找到或是使用者定義的 Type



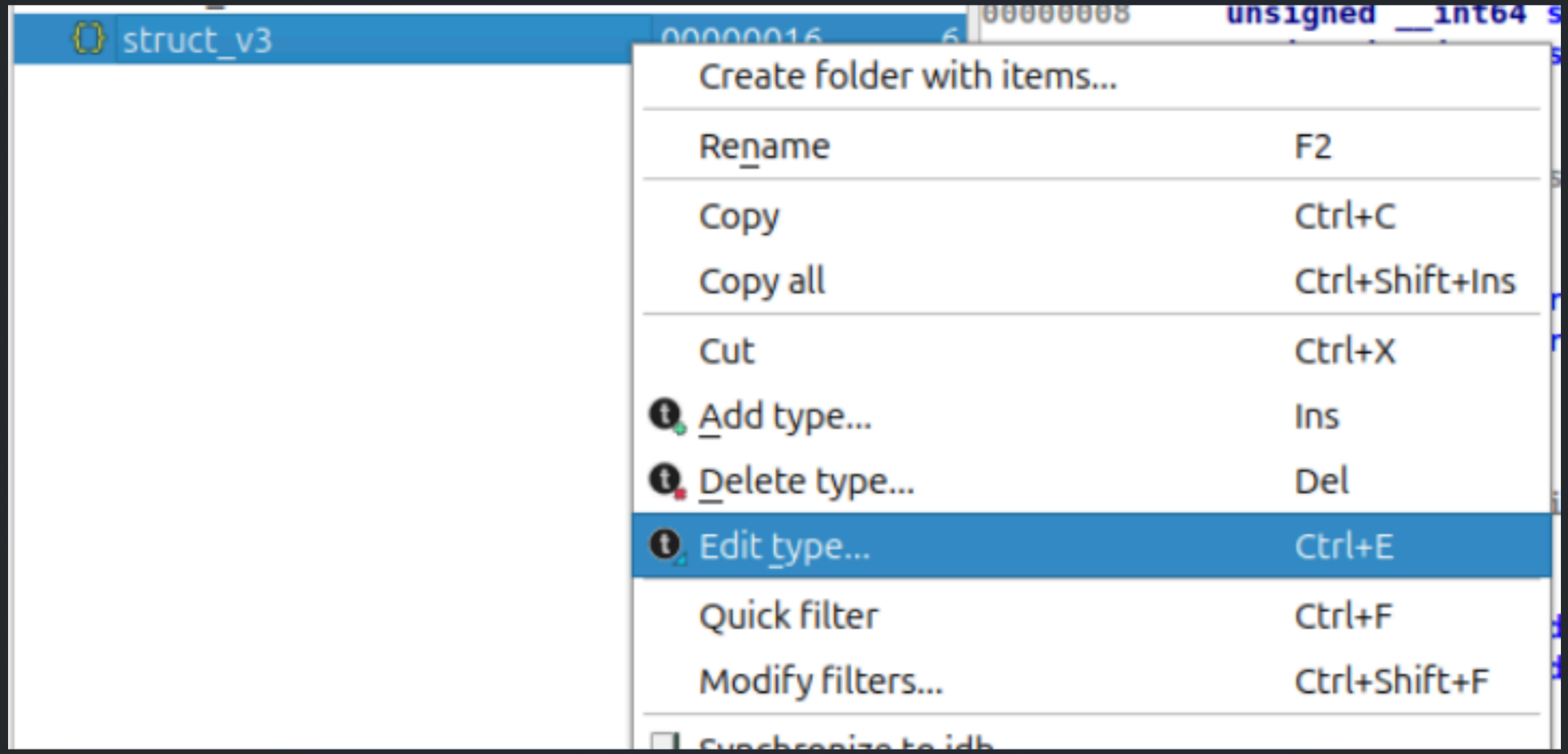
Local Types

- 有一個 Subview 叫做 Local Types
  - 存放著 IDA 找到或是使用者定義的 Type
- 8.3 以前是 Local Types + Structures
  - 現在合併成一個 View 了

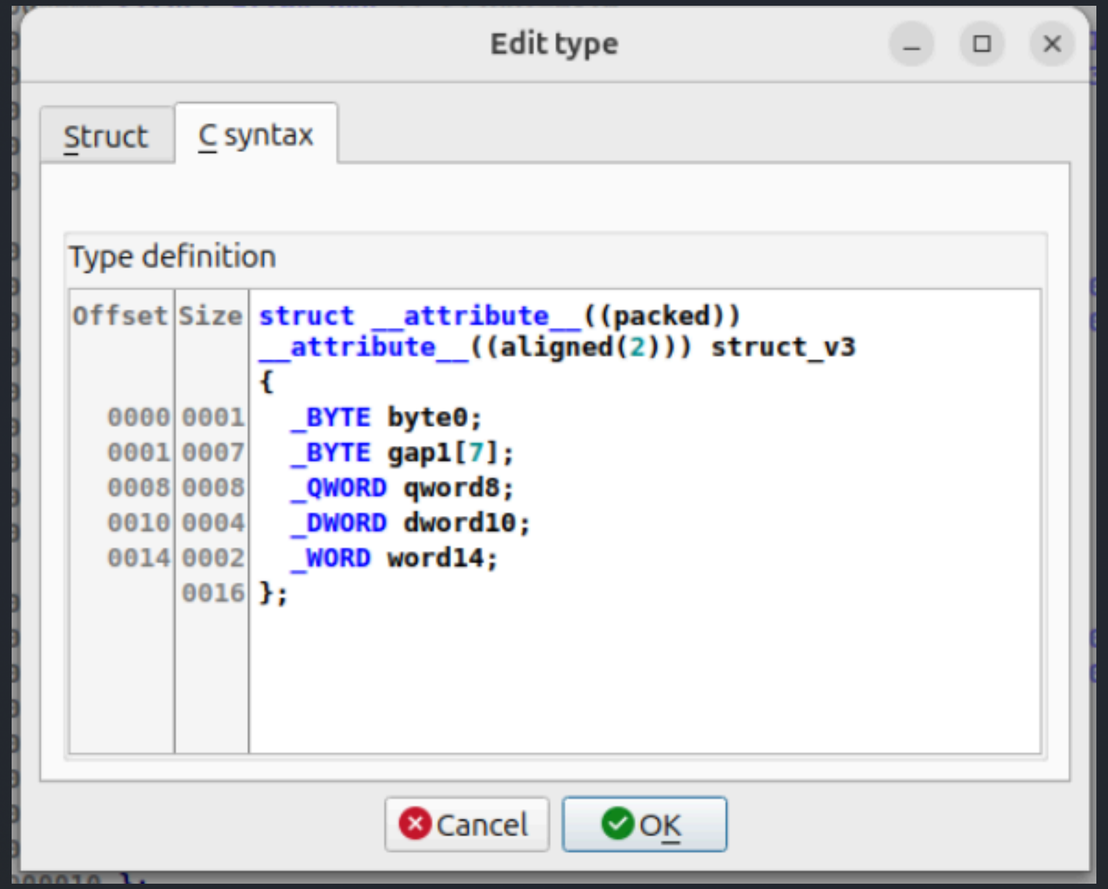



Local Types
IDA View-A
Pseudocode-A
Hex View-1
Name
Size
O
Elf64_Sym 00000018 1
Elf64_Rela 00000018 2
Elf64_Dyn 00000010 3
Elf64_Verneed 00000010 4
Elf64_Vernaux 00000010 5
struct_v3 00000016 6
00000000 struct Elf64_Sym // sizeof=0x18
00000000 { // XREF: LOAD:00000000000003D8/r
00000000 // LOAD:00000000000003F0/r ...
00000000 unsigned __int32 st_name __offset(OFF64,0x480);
00000004 unsigned __int8 st_info;
00000005 unsigned __int8 st_other;
00000006 unsigned __int16 st_shndx;
00000008 unsigned __int64 st_value __off;
00000010 unsigned __int64 st_size;
00000018 };
00000000 struct Elf64_Rela // sizeof=0x18
00000000 { // XREF: LOAD:0000000000000588/r
00000000 // LOAD:00000000000005A0/r ...
00000000 unsigned __int64 r_offset;
00000008 unsigned __int64 r_info;
00000010 __int64 r_addend;
00000018 };
00000000 struct Elf64_Dyn // sizeof=0x10
00000000 { // XREF: LOAD:_DYNAMIC/r
00000000 // LOAD:00000000000003DC8/r ...
00000000 unsigned __int64 d_tag;
00000008 unsigned __int64 d_un;
00000010 };
00000000 struct Elf64_Verneed // sizeof=0x10
00000000 { // XREF: LOAD:0000000000000538/r
00000000 // LOAD:0000000000000558/r
00000000 unsigned __int16 vn_version;
00000002 unsigned __int16 vn_cnt;
00000004 unsigned __int32 vn_file __offset(OFF64,0x480);
00000008 unsigned __int32 vn_aux;
0000000C unsigned __int32 vn_next;
00000010 };
00000000 struct Elf64_Vernaux // sizeof=0x10
00000000 { // XREF: LOAD:0000000000000548/r
00000000 // LOAD:0000000000000568/r ...
00000000 unsigned __int32 vna_hash;
00000004 unsigned __int16 vna_flags;
00000006 unsigned __int16 vna_other;
00000008 unsigned __int32 vna_name __offset(OFF64,0x480);
0000000C unsigned __int32 vna_next;
00000010 };
00000000 struct __attribute__((packed)) __attribute__((aligned(2))) struct_v3 // sizeof=0x16
00000000 {
00000000 _BYTE byte0;
00000001 _BYTE gap1[7];
00000008 _QWORD qword8;
00000010 _DWORD dword10;
00000014 _WORD word14;
00000016 };

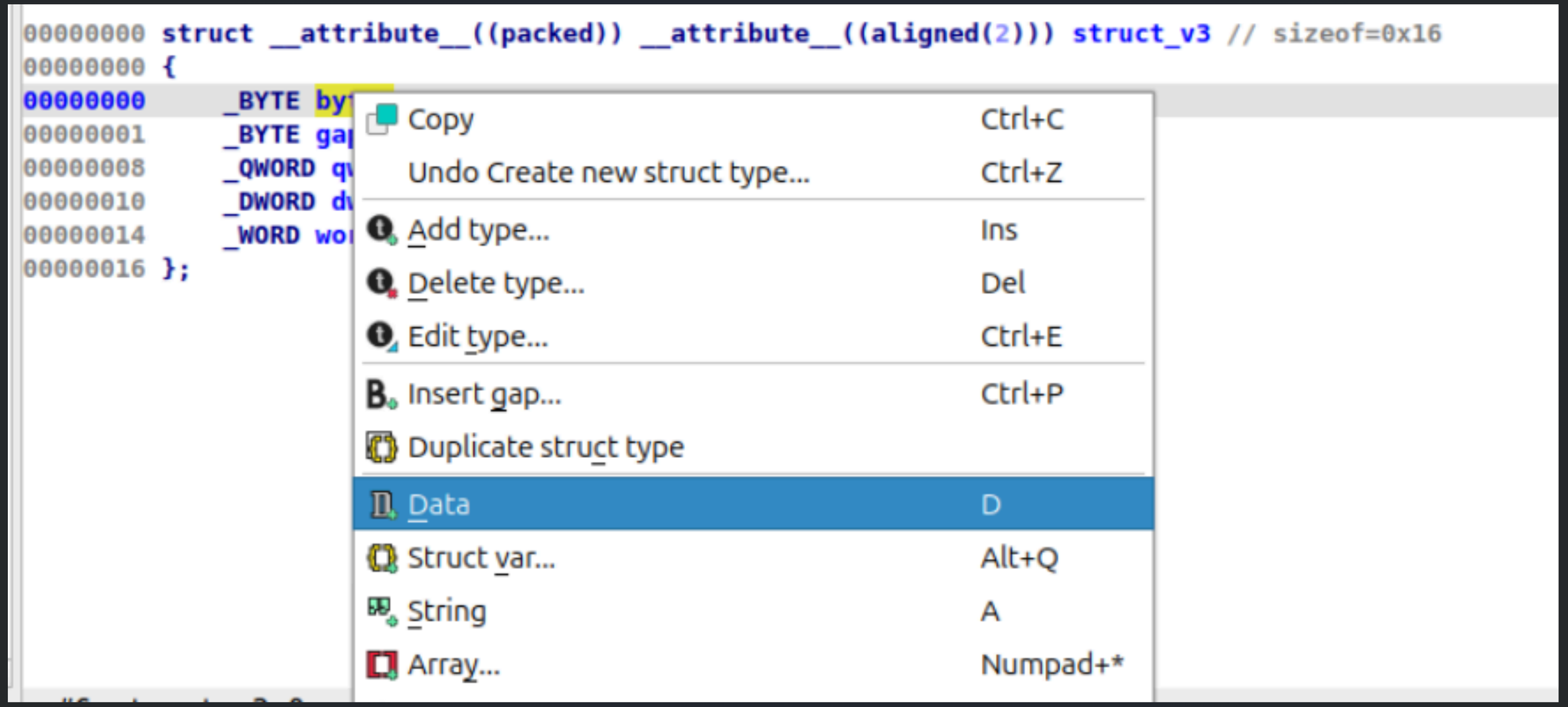
右鍵編輯 Type


struct_v3
00000008 unsigned __int64
Create folder with items...
Rename F2
Copy Ctrl+C
Copy all Ctrl+Shift+Ins
Cut Ctrl+X
Add type... Ins
Delete type... Del
Edit type... Ctrl+E
Quick filter Ctrl+F
Modify filters... Ctrl+Shift+F

方法 1: 跟寫 C 的 Struct 一樣



方法 2: 像是先前在 Disassembly View 一樣的編輯方式



Struct Recovery - Manual

- 人工閱讀的重點
  - 資料位址、資料大小、存取的方式
- 舉個例子：
  - 位址：[rbp-20h]、大小：dword、存取方式：add/sub
  - 那 [rbp-20h] 可能是一個 uint32_t 或是 int32_t

PAhole

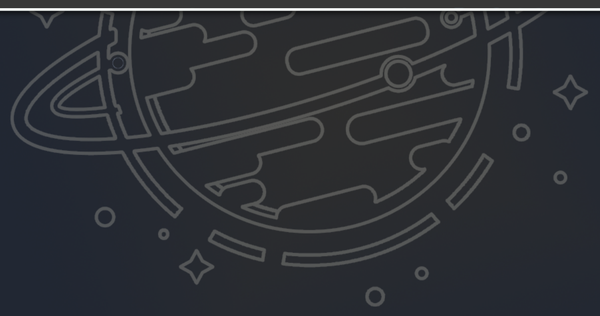
Poke A hole

- 戳洞洞
- 幫忙計算 memory alignment 對結構影響的工具
- 需要 executable 本身帶有 debug symbol
  - 預設編譯是不會有這麼詳細的 debug information 的
  - gcc/g++ 編譯時加上 -g

Poke A hole

```c
struct mystruct_t {
    char a;
    uint64_t b;
    uint32_t c;
    short d;
};
```

```
struct mystruct_t {
        char                       a;                    /*     0     1 */

        /* XXX 7 bytes hole, try to pack */

        uint64_t                   b;                    /*     8     8 */
        uint32_t                   c;                    /*    16     4 */
        short int                  d;                    /*    20     2 */

        /* size: 24, cachelines: 1, members: 4 */
        /* sum members: 15, holes: 1, sum holes: 7 */
        /* padding: 2 */
        /* last cacheline: 24 bytes */
};
```



Inheritance

Class Layout

- C++ 有一個 Class 的概念
  - Class 可以實作 OOP 的設計
- Class Method 如果想要做成有繼承關係的話需要 virtual
  - Virtual 對 Class 資料結構產生了什麼影響呢？

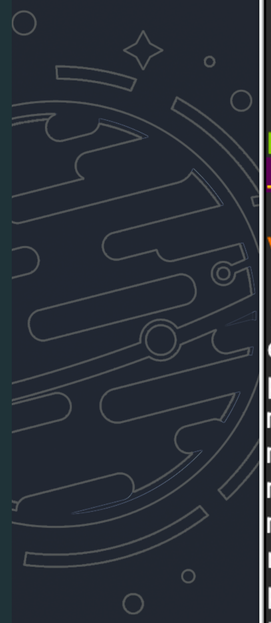
沒有繼承的 Class

```cpp
class Entity {
    int age;
public:
    Entity() {
        this->age = 0;
    }
    void Speak() {
        printf("An Entity can speak!\n");
    }
};
```

```cpp
class Cat {
    int age;
public:
    Cat() {
        this->age = 0;
    }
    void Speak() {
        printf("An Cat can speak!\n");
    }
};
```

Entity Constructor

```
class Entity {
    int age;
public:
    Entity() {
        this->age = 0;
    }
    void Speak() {
        printf("An Entity can speak!\n");
    }
};
```

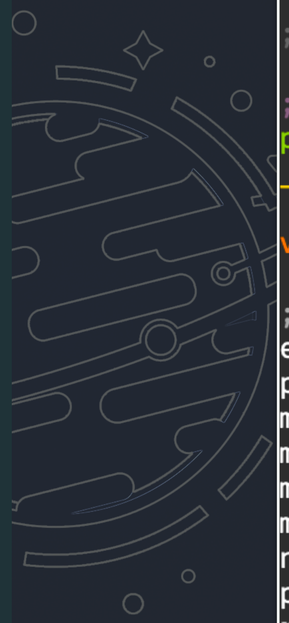


```
; Attributes: bp-based frame

; void __fastcall Entity::Entity(Entity *this)
public _ZN6EntityC2Ev ; weak
_ZN6EntityC2Ev proc near

var_8= qword ptr -8

; __unwind {
endbr64
push    rbp
mov     rbp, rsp
mov     [rbp+var_8], rdi
mov     rax, [rbp+var_8]
mov     dword ptr [rax], 0
nop
pop     rbp
retn
; } // starts at 11D0
_ZN6EntityC2Ev endp
```

this->Age = 0

Cat Constructor

```
class Cat {
    int age;
public:
    Cat() {
        this->age = 0;
    }
    void Speak() {
        printf("An Cat can speak!\n");
    }
};
```



```
; Attributes: bp-based frame

; Cat *__fastcall Cat::Cat(Cat *__hidden this)
public _ZN3CatC2Ev ; weak
_ZN3CatC2Ev proc near

var_8= qword ptr -8

; __unwind {
endbr64
push    rbp
mov     rbp, rsp
mov     [rbp+var_8], rdi
mov     rax, [rbp+var_8]
mov     dword ptr [rax], 0
nop
pop     rbp
retn
; } // starts at 120C
_ZN3CatC2Ev endp
```

this->Age = 0

Invoke Member

```
mov     rdi, rax          ; this
call    _ZN6EntityC2Ev    ; Entity::Entity(void)
lea     rax, [rbp+var_10]
mov     rdi, rax          ; this
call    _ZN6Entity5SpeakEv ; Entity::Speak(void)
lea     rax, [rbp+var_C]
mov     rdi, rax          ; this
call    _ZN3CatC2Ev       ; Cat::Cat(void)
lea     rax, [rbp+var_C]
mov     rdi, rax          ; this
call    _ZN3Cat5SpeakEv   ; Cat::Speak(void)
```

沒有什麼特別的，就是個正常呼叫

Class Layout

- 有 Virtual 的 Class 資料結構是什麼模樣？

有繼承的 Class

```cpp
class Entity {
    int age;
public:
    Entity() {
        this->age = 0;
    }
    virtual void Speak() {
        printf("An Entity can speak!\n");
    }
};
```

```cpp
class Cat: public Entity {
    int age;
public:
    Cat() {
        this->age = 0;
    }
    virtual void Speak() {
        printf("An Cat can speak!\n");
    }
};
```

Entity Constructor

```cpp
class Entity {
    int age;
public:
    Entity() {
        this->age = 0;
    }
    virtual void Speak() {
        printf("An Entity can speak!\n");
    }
};
```

```
; Attributes: bp-based frame

; Entity *__fastcall Entity::Entity(Entity *__hidden this)
public _ZN6EntityC2Ev ; weak
_ZN6EntityC2Ev proc near

var_8= qword ptr -8

; __unwind {
endbr64
push    rbp
mov     rbp, rsp
mov     [rbp+var_8], rdi
lea     rdx, off_3D80
mov     rax, [rbp+var_8]
mov     [rax], rdx
mov     rax, [rbp+var_8]
mov     dword ptr [rax+8], 0
nop
pop     rbp
retn
; } // starts at 11D4
_ZN6EntityC2Ev endp
```

Entity Constructor

```cpp
class Entity {
    int age;
public:
    Entity() {
        this->age = 0;
    }
    virtual void Speak() {
        printf("An Entity can speak!\n");
    }
};
```

你有沒有發現有東西不一樣？

```
; Attributes: bp-based frame

; Entity *__fastcall Entity::Entity(Entity *__hidden this)
public _ZN6EntityC2Ev ; weak
_ZN6EntityC2Ev proc near

var_8= qword ptr -8

; __unwind {
endbr64
                                        i
lea     rdx, off_3D80
mov     rax, [rbp+var_8]
mov     [rax], rdx
mov     rax, [rbp+var_8]
mov     dword ptr [rax+8], 0
nop
pop     rbp
retn
; } // starts at 11D4
_ZN6EntityC2Ev endp
```

Entity Constructor

```
; Attributes: bp-based frame

; void __fastcall Entity::Entity(Entity *this)
public _ZN6EntityC2Ev ; weak
_ZN6EntityC2Ev proc near

var_8= qword ptr -8

; __unwind {
endbr64
push    rbp
mov     rbp, rsp
mov     [rbp+var_8], rdi
mov     rax, [rbp+var_8]
mov     dword ptr [rax], 0
nop
pop     rbp
retn
; } // starts at 11D0
_ZN6EntityC2Ev endp
```

```
; Attributes: bp-based frame

; Entity *__fastcall Entity::Entity(Entity *__hidden this)
public _ZN6EntityC2Ev ; weak
_ZN6EntityC2Ev proc near

var_8= qword ptr -8

; __unwind {
endbr64
push    rbp
mov     rbp, rsp
mov     [rbp+var_8], rdi
lea     rdx, off_3D80
mov     rax, [rbp+var_8]
mov     [rax], rdx
mov     rax, [rbp+var_8]
mov     dword ptr [rax+8], 0
nop
pop     rbp
retn
; } // starts at 11D4
_ZN6EntityC2Ev endp
```

Entity Constructor

```
; Attributes: bp-based frame

; void __fastcall Entity::Entity(Entity *this)
public _ZN6EntityC2Ev ; weak
_ZN6EntityC2Ev proc near

var_8= qword ptr -8

; __unwind {
endbr64
push    rbp
mov     rbp, rsp
mov     [rbp+var_8], rdi
mov     rax, [rbp+var_8]
mov     dword ptr [rax], 0
nop
pop     rbp
retn
; } // starts at 11D0
_ZN6EntityC2Ev endp
```

```
; Attributes: bp-based frame

; Entity *__fastcall Entity::Entity(Entity *__hidden this)
public _ZN6EntityC2Ev ; weak
_ZN6EntityC2Ev proc near

var_8= qword ptr -8

; __unwind {
endbr64
push    rbp
mov     rbp, rsp
mov     [rbp+var_8], rdi
lea     rdx, off_3D80
mov     rax, [rbp+var_8]
mov     [rax], rdx
mov     rax, [rbp+var_8]
mov     dword ptr [rax+8], 0
nop
pop     rbp
retn
; } // starts at 11D4
_ZN6EntityC2Ev endp
```

Entity Constructor

```
; Attributes: bp-based frame

; void __fastcall Entity::Entity(Entity *this)
public _ZN6EntityC2Ev ; weak
_ZN6EntityC2Ev proc near

var_8= qword ptr -8

; __unwind {
endbr64
push    rbp
mov     rbp, rsp
mov     [rbp+var_8], rdi
mov     rax, [rbp+var_8]
mov     dword ptr [rax], 0
nop
pop     rbp
retn
; } // starts at 11D0
_ZN6EntityC2Ev endp
```

VTable

```
; Attributes: bp-based frame

; Entity *__fastcall Entity::Entity(Entity *__hidden this)
public _ZN6EntityC2Ev ; weak
_ZN6EntityC2Ev proc near

var_8= qword ptr -8

; __unwind {
endbr64
push    rbp
mov     rbp, rsp
mov     [rbp+var_8], rdi
lea     rdx, off_3D80
mov     rax, [rbp+var_8]
mov     [rax], rdx
mov     rax, [rbp+var_8]
mov     dword ptr [rax+8], 0
nop
pop     rbp
retn
; } // starts at 11D4
_ZN6EntityC2Ev endp
```

VTable

- 在這邊把 Cat Cast 成 Entity，以 Entity 呼叫會用 Cat->Speak
  - 通過 VTable 來讓繼承物件的 Method 可以被正確呼叫

```
int main() {
    Entity entity = Entity();
    entity.Speak();

    Cat cat = Cat();
    dynamic_cast<Entity*>(&cat)->Speak();
}
```

VTable Invoke

```
mov     rdi, rax            ; this
call    _ZN3CatC2Ev         ; Cat::Cat(void)
mov     rax, [rbp+var_20]
mov     rdx, [rax]
lea     rax, [rbp+var_20]
mov     rdi, rax
call    rdx
```

Deference *Vtable

VTable Invoke

```
mov     rdi, rax            ; this
call    _ZN3CatC2Ev         ; Cat::Cat(void)
mov     rax, [rbp+var_20]
mov     rdx, [rax]
lea     rax, [rbp+var_20]
mov     rdi, rax
call    rdx
```

- Index to Function Pointer

VTable Invoke

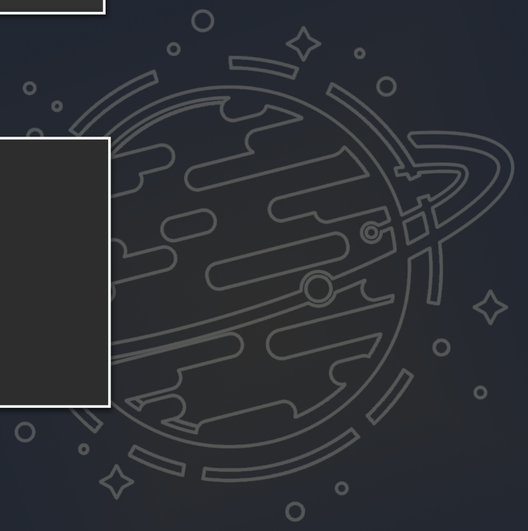
```
mov     rdi, rax          ; this
call    _ZN3CatC2Ev       ; Cat::Cat(void)
mov     rax, [rbp+var_20]
mov     rdx, [rax]
lea     rax, [rbp+var_20]
mov     rdi, rax
call    rdx
```

Call

VTable Invoke vs Normal Invoke

```
mov     rdi, rax          ; this
call    _ZN3CatC2Ev       ; Cat::Cat(void)
mov     rax, [rbp+var_20]
mov     rdx, [rax]
lea     rax, [rbp+var_20]
mov     rdi, rax
call    rdx
```

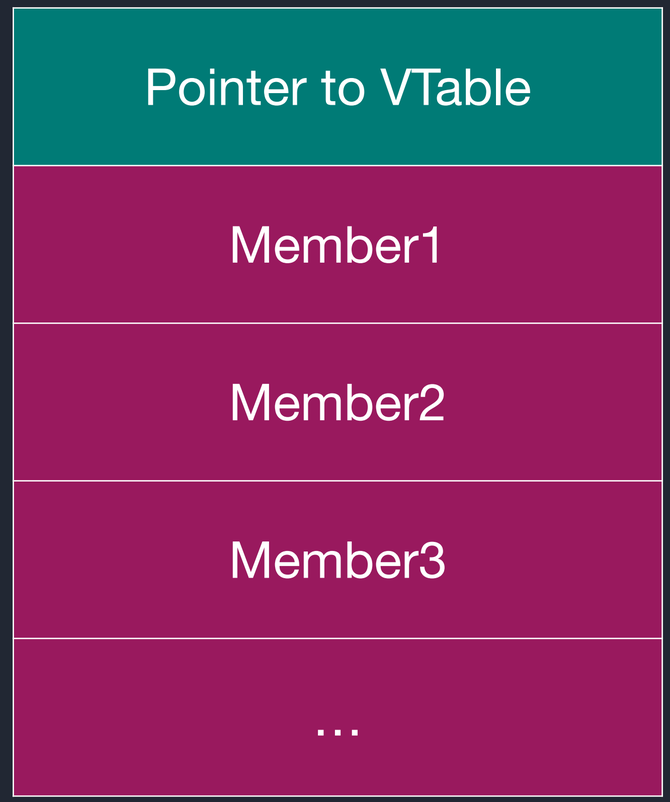
```
mov     rdi, rax          ; this
call    _ZN3CatC2Ev       ; Cat::Cat(void)
lea     rax, [rbp+var_C]
mov     rdi, rax          ; this
call    _ZN3Cat5SpeakEv   ; Cat::Speak(void)
```



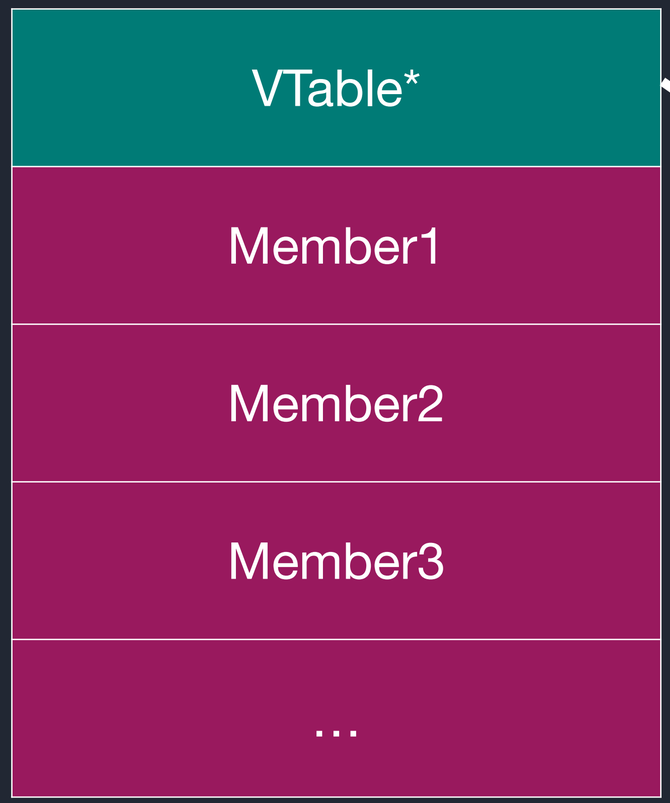
Impact

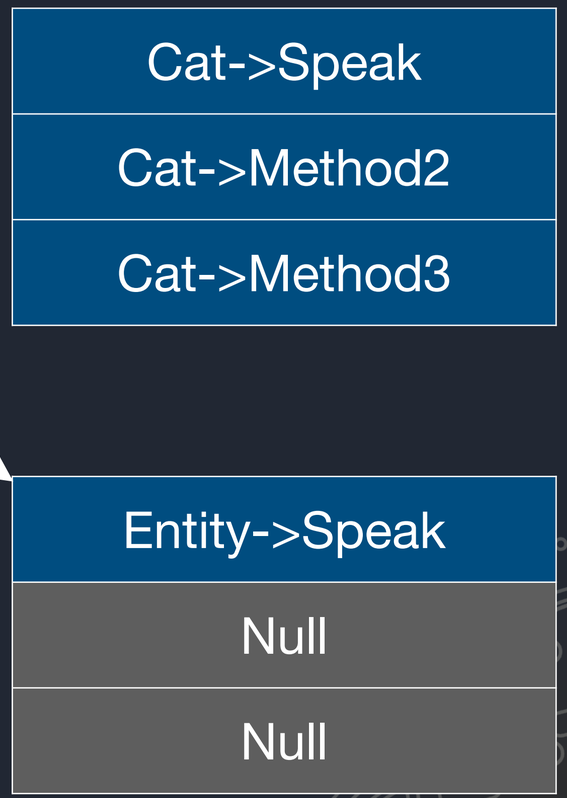
- 因為不是直接跳去某個位置，而是從結構體取出一個位置才跳轉
- 動態分析變的麻煩
  - 因為不好抓 Control-Flow
- 靜態分析也變麻煩 (Stripped Binary 沒 Symbol 時)
  - 繼承關係麻煩時，很難還原結構
  - Method 數量很多時，很難還原結構

Class-Layout

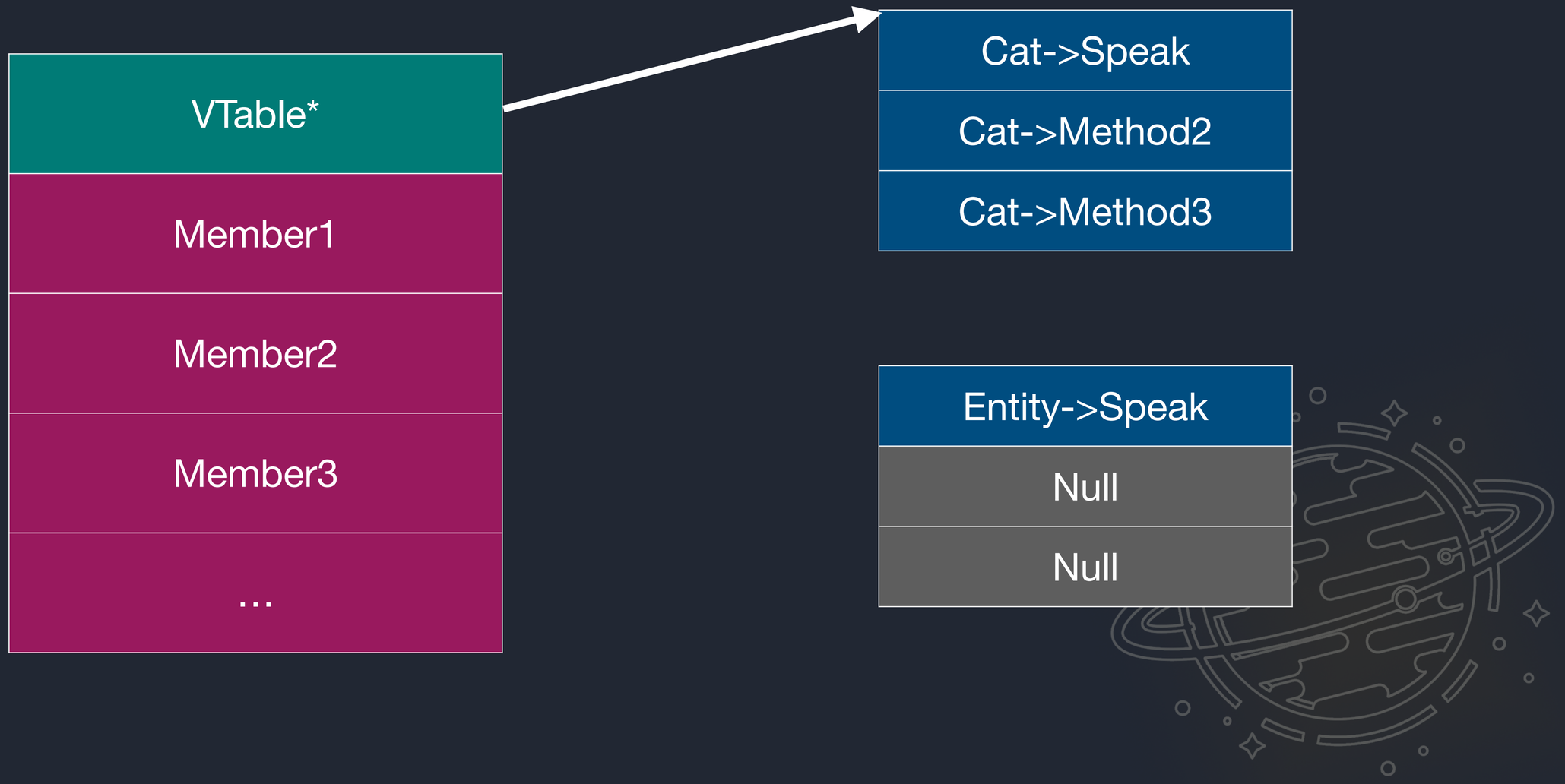


Class-Layout





Class-Layout



Class-Layout

Cat->Speak
Cat->Method2
Cat->Method3

Entity->Speak
Null
Null

Takeaway

Takeaway

- 學會 IDAFree 的基礎操作
- 學會如何使用 IDA Local debugger 與 Binary Patching
- 認識基本的資料結構與還原方式
  - VTable


Q & A

Thanks for attention ✨