

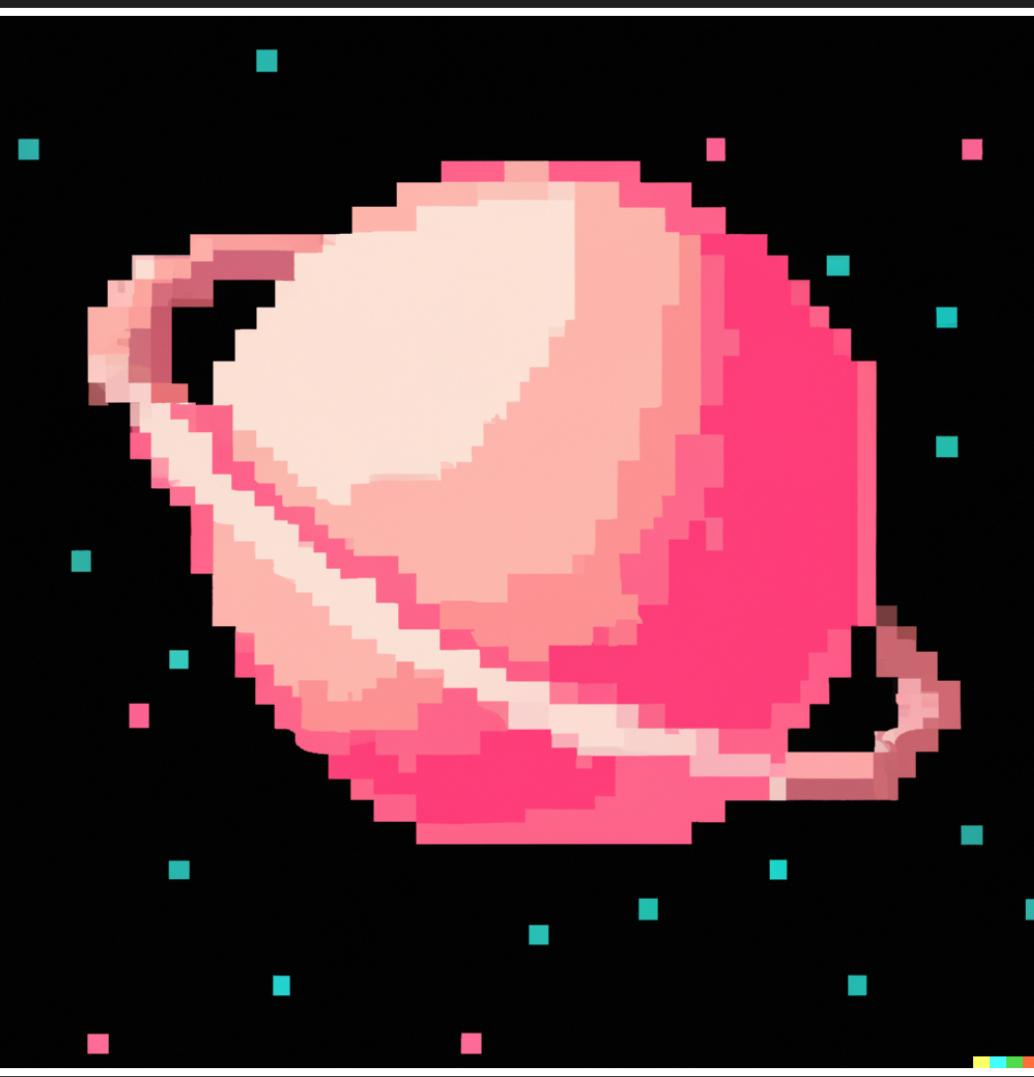
# Reverse #2

Turn It Up to Next Level

TwinkleStar03 @ 2023/11/

# // whoami

- TwinkleStar03 / 星星
  - RE/Pwn @ XxTSJxX, \${CyStick}
  - Member of UNDEFINED Conclave



# // Outline

- IDA Recap
- More Structures
- Symbolic Execution Engine
- Obfuscation
- Exception
- Packer
- Anti-Debug
- Anti-Disassemble

# IDA Recap

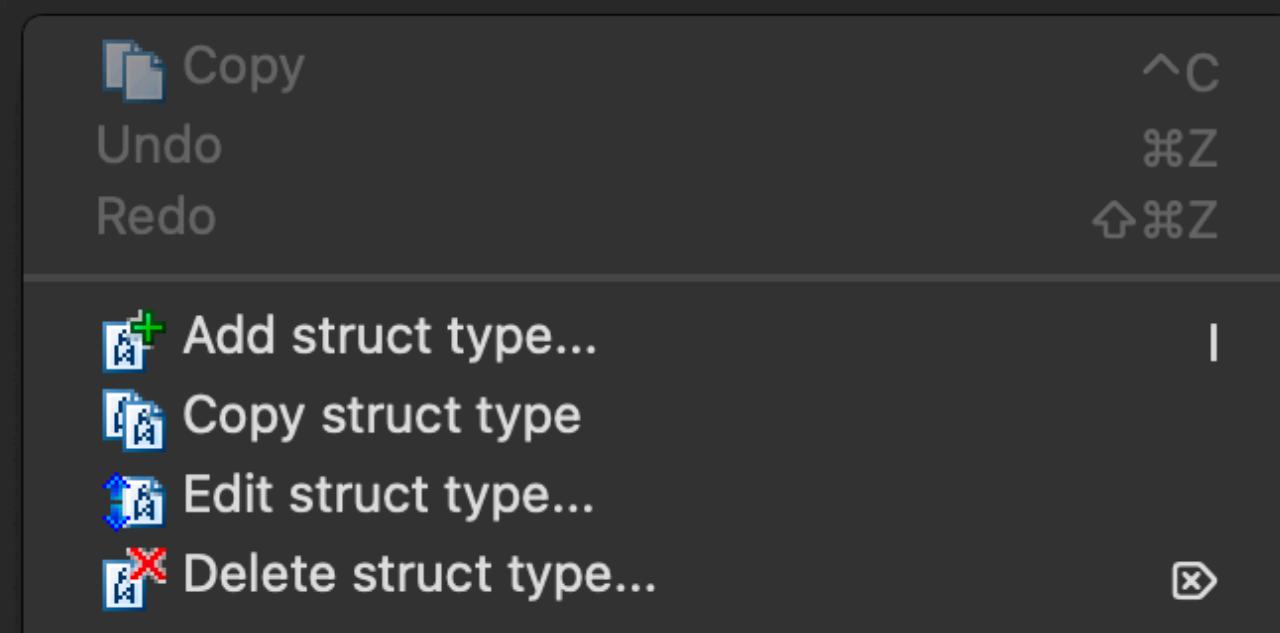


# IDA - Data Structure

- Create New Struct
- Structure
  - 用 IDA 的定義方式來寫 Structure
- LocalType
  - 可以用 C-Struct 的方式來寫

# // IDA - Data Structure

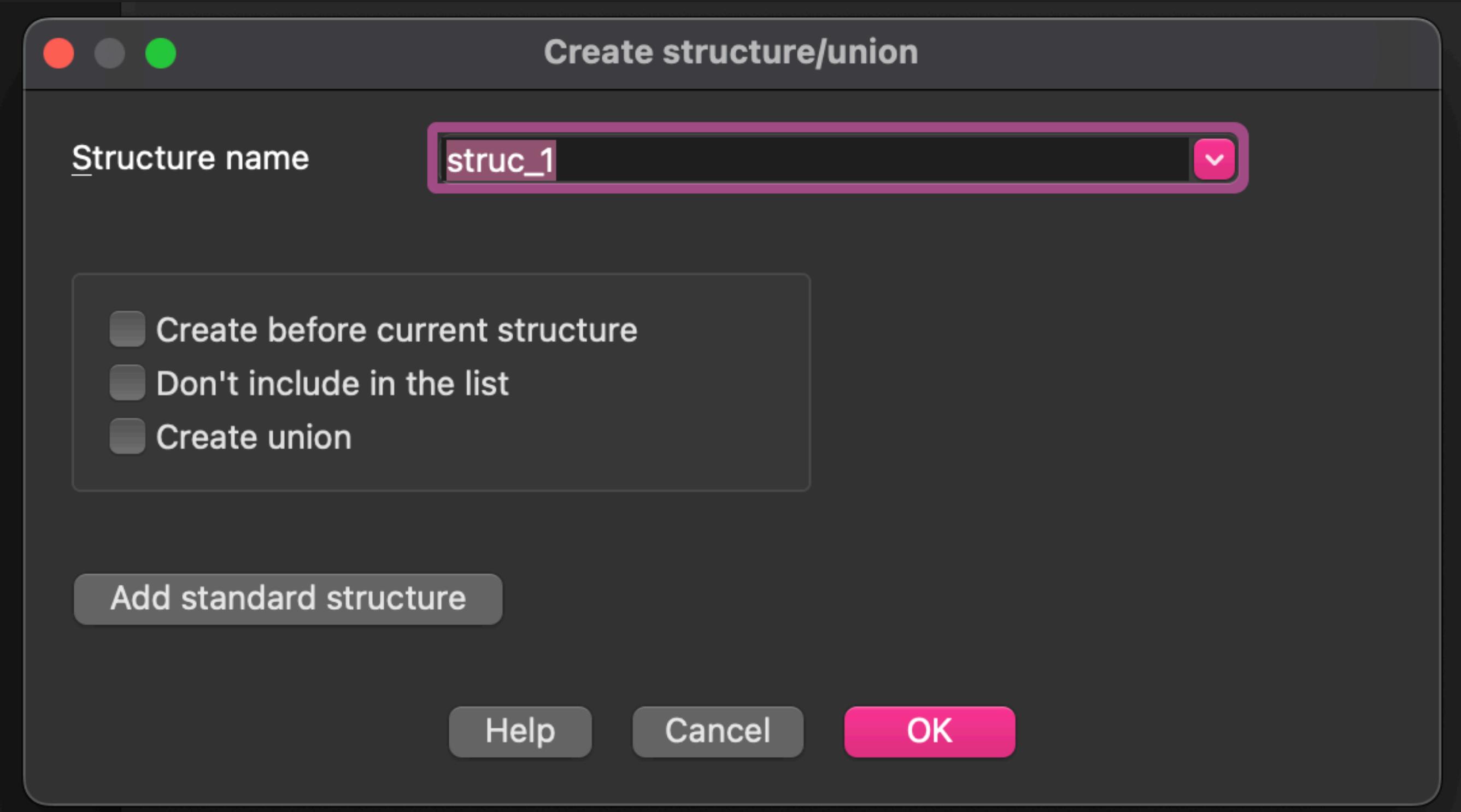
Name	
Elf64_Sym	<code>00000000 ; Ins/Del : create/delete structure</code>
Elf64_Rela	<code>00000000 ; D/A/* : create structure member (data/ascii/array)</code>
Elf64_Dyn	<code>00000000 ; N : rename structure or structure member</code>
Elf64_Verneed	<code>00000000 ; U : delete structure member</code>
Elf64_Vernaux	<code>&gt; 00000000 ; [00000018 BYTES. COLLAPSED STRUCT Elf64_Sym. PRESS CTRL-NUMPAD+ TO EXPAND]</code>
	<code>&gt; 00000000 ; [00000018 BYTES. COLLAPSED STRUCT Elf64_Rela. PRESS CTRL-NUMPAD+ TO EXPAND]</code>
	<code>&gt; 00000000 ; [00000010 BYTES. COLLAPSED STRUCT Elf64_Dyn. PRESS CTRL-NUMPAD+ TO EXPAND]</code>
	<code>&gt; 00000000 ; [00000010 BYTES. COLLAPSED STRUCT Elf64_Verneed. PRESS CTRL-NUMPAD+ TO EXPAND]</code>
	<code>&gt; 00000000 ; [00000010 BYTES. COLLAPSED STRUCT Elf64_Vernaux. PRESS CTRL-NUMPAD+ TO EXPAND]</code>



A context menu is displayed over the collapsed structure entries. The menu includes:

- Copy (represented by a clipboard icon)
- Undo (represented by a left arrow icon)
- Redo (represented by a right arrow icon)
- ^C (Copy hotkey)
- ⌘Z (Undo hotkey)
- ⇧⌘Z (Redo hotkey)
- Add struct type... (represented by a plus icon)
- Copy struct type (represented by a clipboard icon)
- Edit struct type... (represented by a pencil icon)
- Delete struct type... (represented by a trash bin icon)

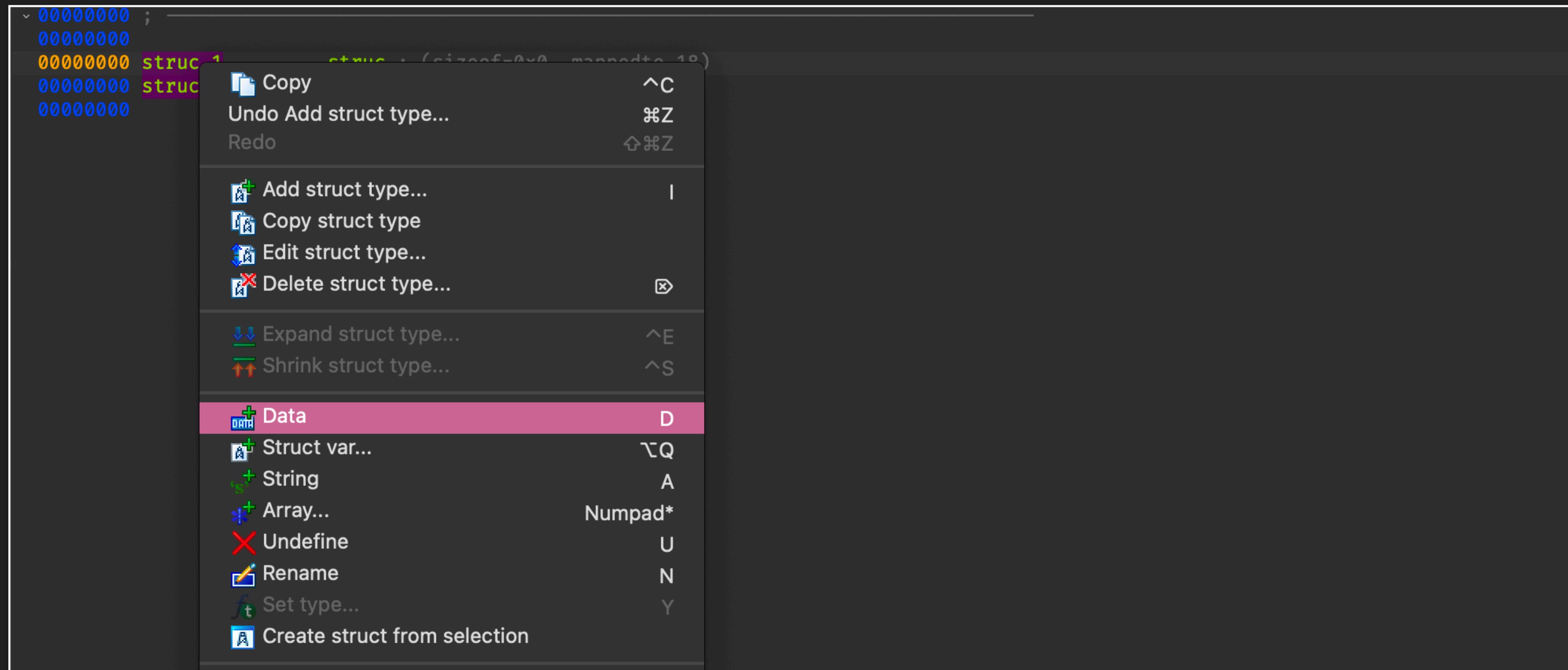
# // IDA - Data Structure



# // IDA - Data Structure

```
v |00000000 ; ——————  
00000000  
00000000 struc ; (sizeof=0x0, mappedto_18)  
00000000 struc_1  
00000000 ends  
00000000
```

# // IDA - Data Structure



# // IDA - Data Structure

- Create New Struct
- Structure
  - 用 IDA 的定義方式來寫 Structure
- LocalType
  - 可以用 C-Struct 的方式來寫

# // IDA - Data Structure

Ordinal	Name	Size	Sync	Description
1	Elf64_Sym	00000018	Auto	struct __attribute__((aligned(8))) {unsigned __int32 st_name;unsigned __int8 st_info;unsigned __int8...
2	Elf64_Rela	00000018	Auto	struct {unsigned __int64 r_offset;unsigned __int64 r_info;__int64 r_addend;}
3	Elf64_Dyn	00000010	Auto	struct {unsigned __int64 d_tag;unsigned __int64 d_un;}
4	Elf64_Verneed	00000010	Auto	struct __attribute__((aligned(4))) {unsigned __int16 vn_version;unsigned __int16 vn_cnt;unsigned __i...
5	Elf64_Vernaux	00000010	Auto	struct __attribute__((aligned(4))) {unsigned __int32 vna_hash;unsigned __int16 vna_flags;unsigned __...
6	_m64	00000008		union __attribute__((aligned(8))) {unsigned __int64 m64_u64;float m64_f32[2];__int8 m64_i8[8];__int1...
7	_m128	00000010		union __attribute__((aligned(16))) {float m128_f32[4];unsigned __int64 m128_u64[2];__int8 m128_i8[16...]
8	_m128d	00000010		struct {double m128d_f64[2];}
9	_m128i	00000010		union __attribute__((aligned(16))) {__int8 m128i_i8[16];__int16 m128i_i16[8];__int32 m128i_i32[4];__...
10	_m256	00000020		union __attribute__((aligned(32))) {float m256_f32[8];}
11	_m256d	00000020		union __attribute__((aligned(32))) {double m256d_f64[4];}
12	_m256i	00000020		union __attribute__((aligned(32))) {__int8 m256i_i8[32];__int16 m256i_i16[16];__int32 m256i_i32[8];__...
13	_m512	00000040		union __attribute__((aligned(64))) {float m512_f32[16];}
14	_m512d	00000040		union __attribute__((aligned(64))) {double m512d_f64[8];}
15	_m512i	00000040		union __attribute__((aligned(64))) {__int8 m512i_i8[64];__int16 m512i_i16[32];__int32 m512i_i32[16];__...
16	_va_list_tag	00000018		struct {unsigned int gp_offset;unsigned int fp_offset;void *overflow_arg_area;void *reg_save_area;}
17	gcc_va_list	00000018		typedef __va_list_tag[1]

Insert... ^I

Delete ^D

Edit... ^E

Copy ⌘C

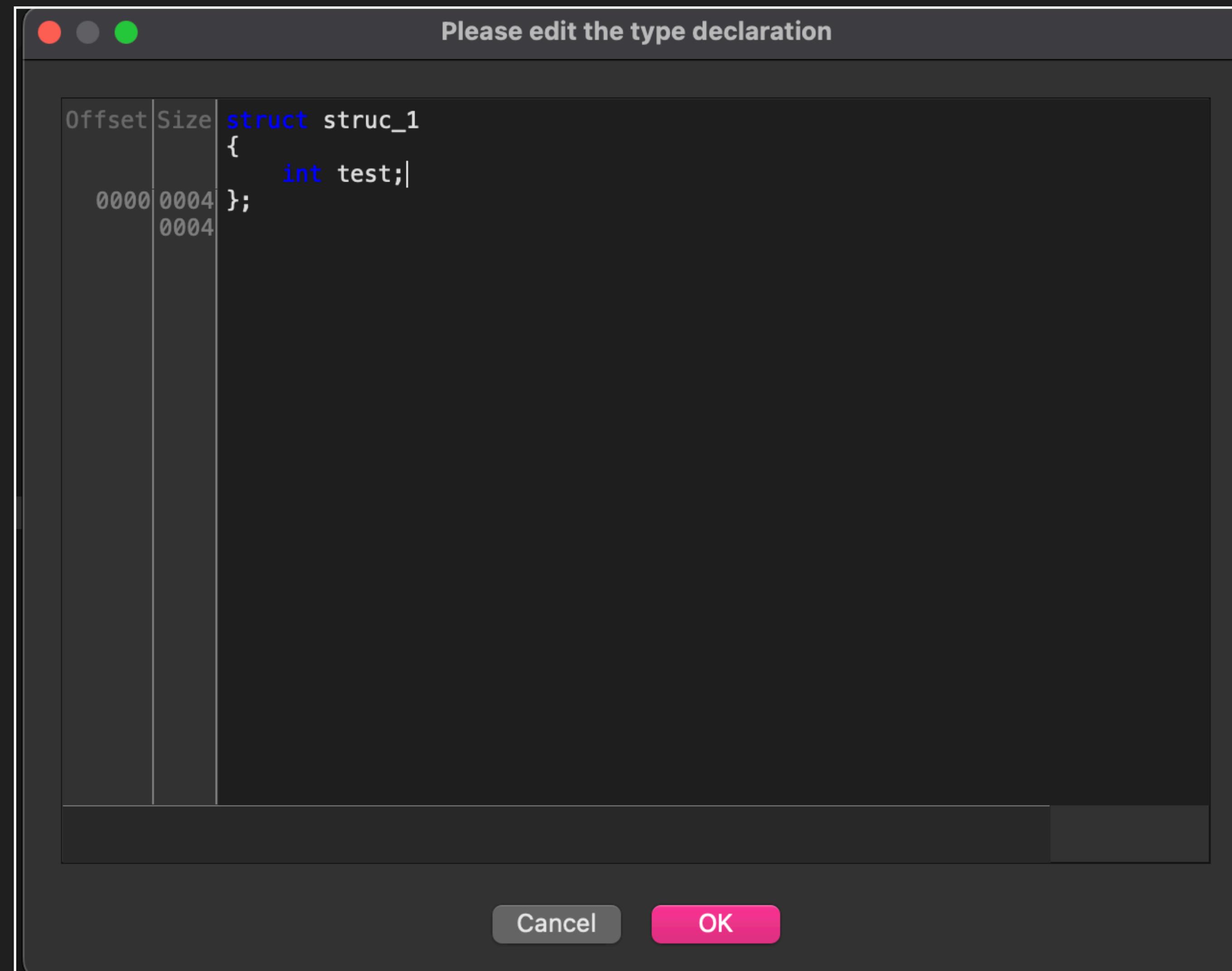
Copy all ^⇧I

Quick filter ^F

Modify filters... ^⇧F



# IDA - Data Structure





# IDA - Disassembly

- 可以框選起來做操作
- P - Create Function
- C - Mark as Code
- U - Undefine

# // IDA - Disassembly

- 可以框選起來做操作
- P - Create Function
  - 將一塊 Code 區域標記成 Function
  - 通常是將紅色區域標成 Function
- C - Mark as Code
- U - Undefine

# // IDA - Disassembly

- 可以框選起來做操作
- P - Create Function
- C - Mark as Code
  - 將 IDA 認不出來的部分當成 Code
- U - Undefine



# IDA - Disassembly

- 可以框選起來做操作
- P - Create Function
- C - Mark as Code
- U - Undefine
  - 將一塊區域標記成 Raw Bytes

# DEMO

## IDA



# DEMO-Note

- Structure
  - IDA Structure
  - LocalTypes
- Disassembly
  - Undefine
  - Mark Code
  - Make Function

# Inheritance

# // Class Layout

- 沒有用到 Virtual 的 Class 實作會長啥樣？

# 沒有繼承的 Class

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

```
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("A 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

# 有繼承的 Class

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

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

# Entity Constructor

```
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

```
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  
  
    lea    rax, 0x1_SDB8  
    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
```

你有沒有發現有東西不一樣？

# 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]   Deference *Vtable
mov      rdx, [rax]
lea     rax, [rbp+var_20]
mov      rdi, rax
call    rdx
```

# // VTable Invoke

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

# // 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

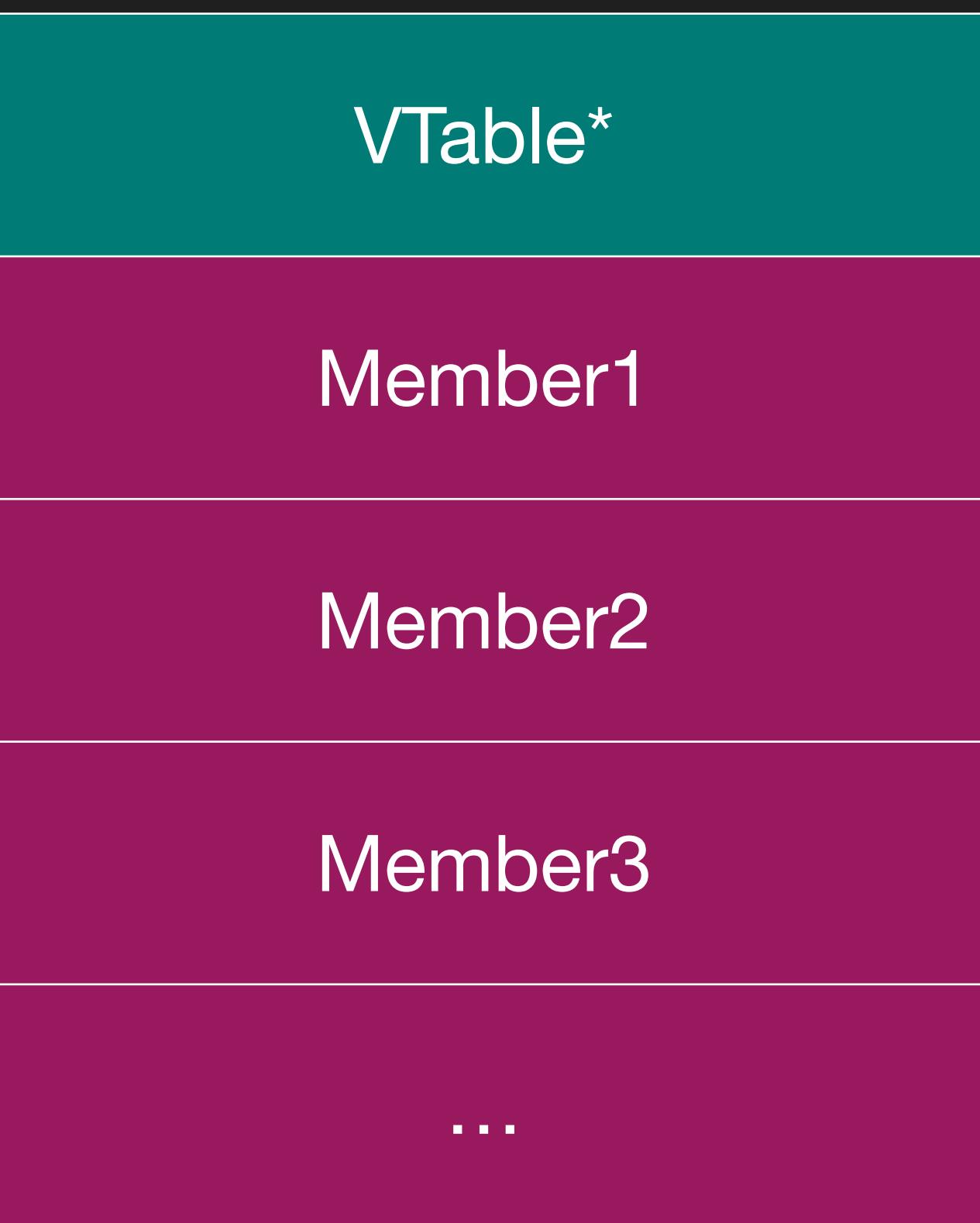
```
mov    rdi, rax      ; this
call   _ZN3CatC2Ev   ; Cat::Cat(void)
mov    rax, [rbp+var_20]
mov    rdx, [rax]
lea    rax, [rbp+var_20]
mov    rdi, rax
call   rdx
```

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

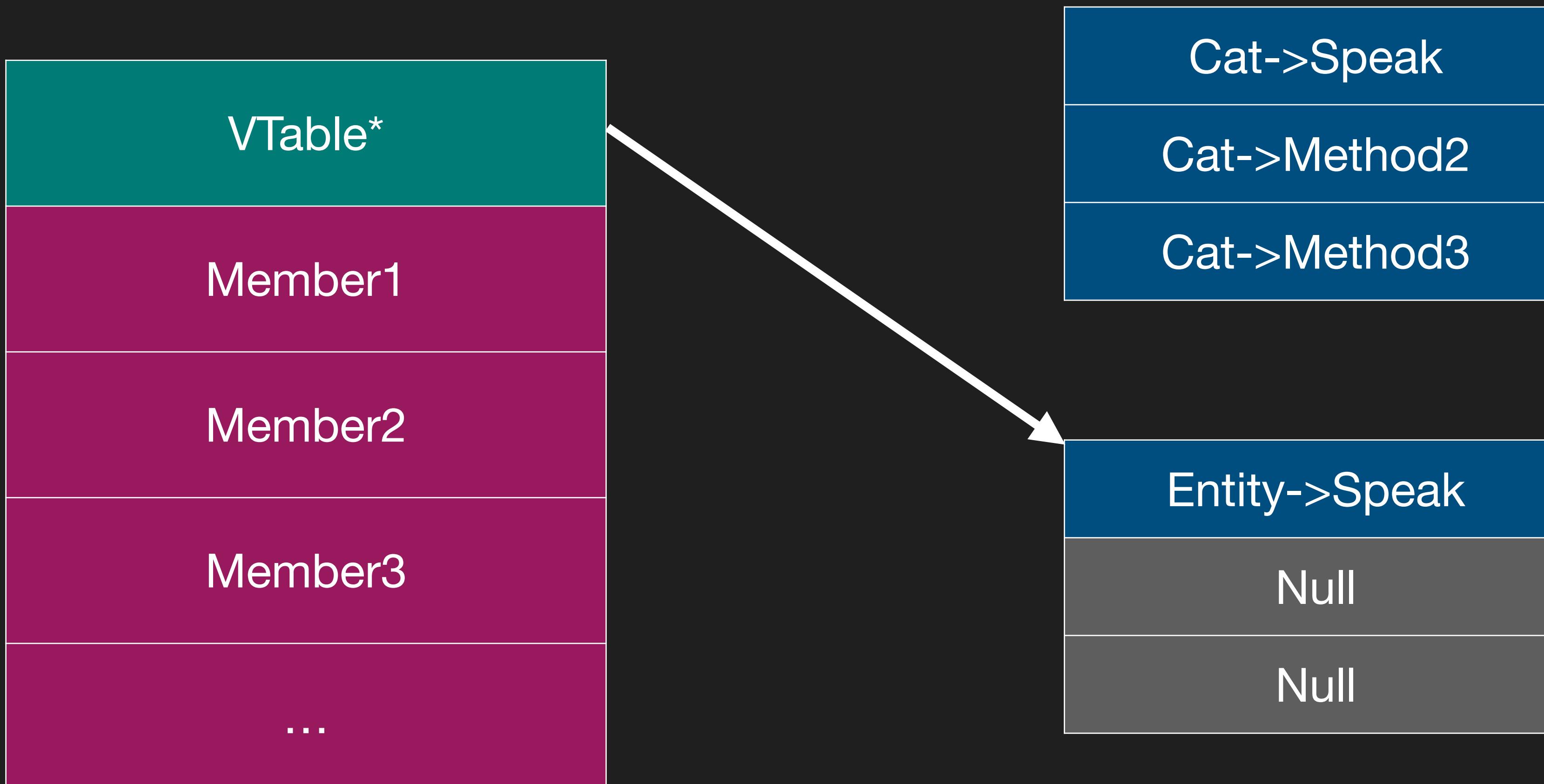
# // Impact

- 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

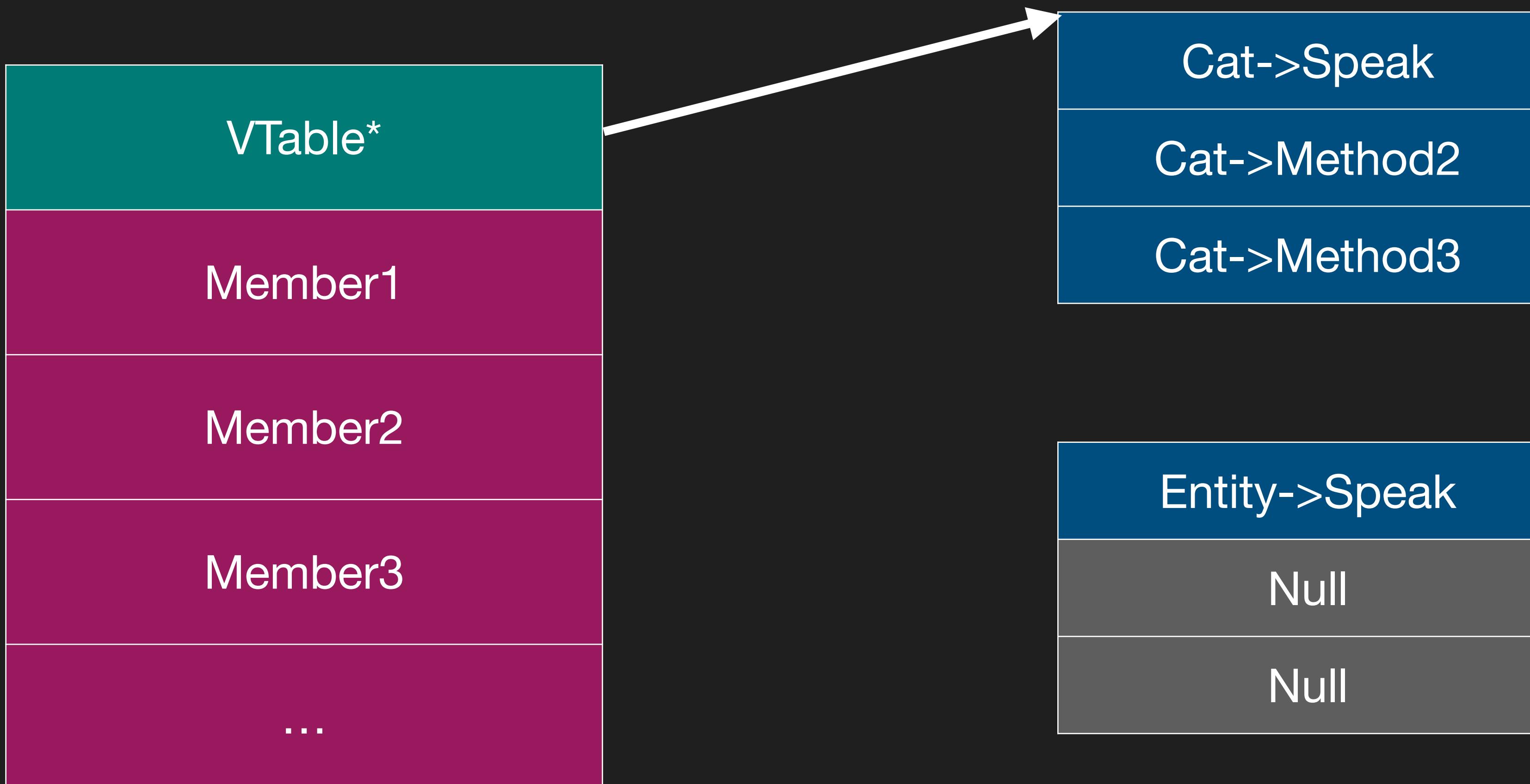


# // Class-Layout





# Class-Layout



# // Class-Layout

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

Entity->Speak
Null
Null

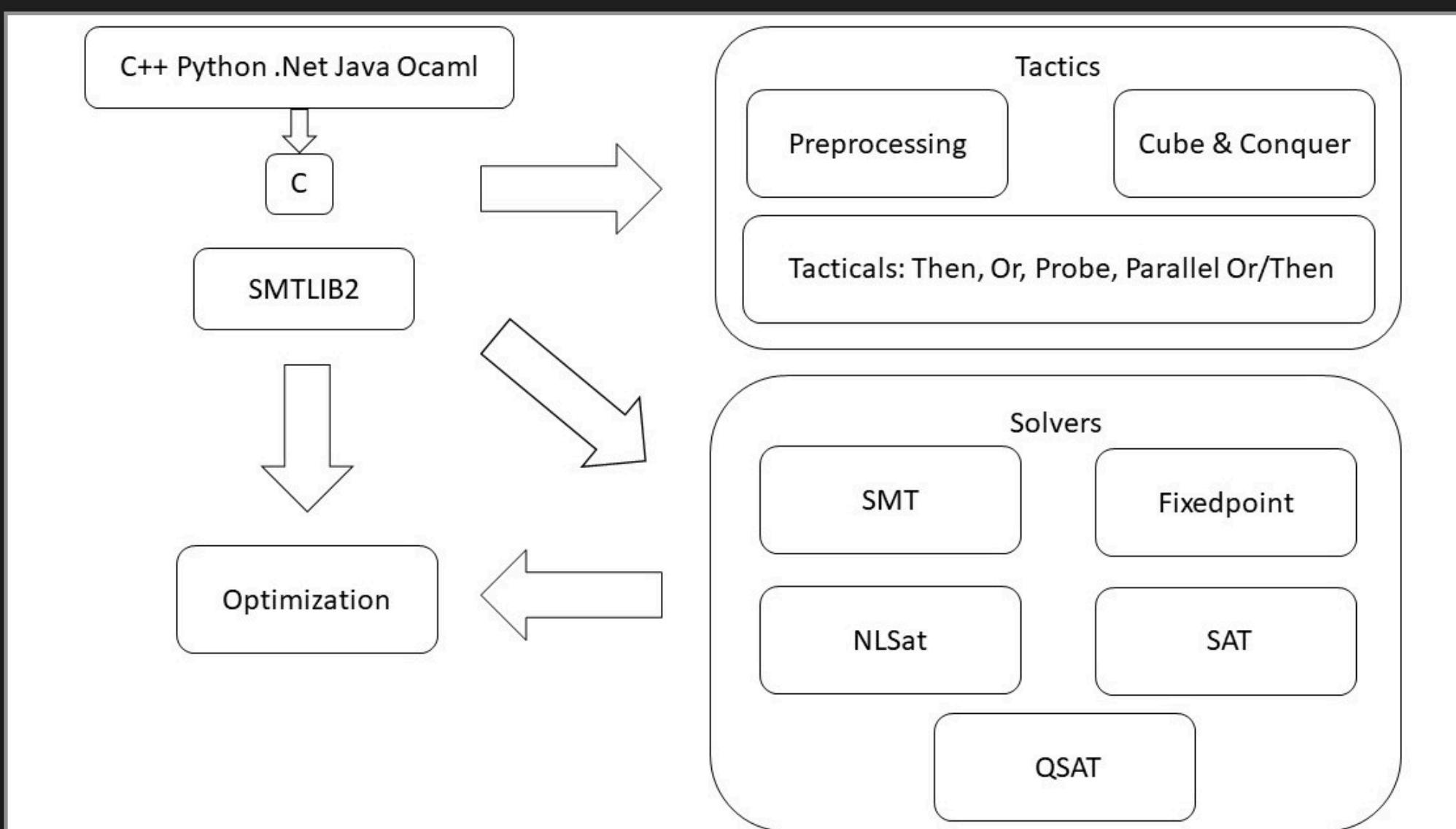
# DEMO

## Non-Virtual/VTable

# Symbolic Execution

# // Z3

- Z3 是一個 Solver 可以用來幫忙解 SAT 問題
  - 有 Python Binding 可以用





- 把他當個黑盒子用，給他 Constraints 他會想辦法找出解答
  - 但不是萬能的，符號太多或是系統太複雜會死掉
- 定義各種符號，與對應的型別
  - BitVec
  - Integer
  - ...



- 完成定義後，往 Solver 送入與符號相關的限制式
- 讓 Solver 嘗試找出一組合理的解
  - 等待奇蹟發生

# // Z3: Example

```
from z3 import *

s = Solver()

# Define a bitvec with 8bits
bv = BitVec('bv', 8)
s.add(bv + 0x20 == 0x30)

# Check if constraints are satisfiable
if s.check() == sat:
    # Print model
    print(s.model())
    # Extract Value of a symbol
    print(s.model()[bv].as_long())
```

# Lab scramble

# // Symbolic Execution?

- 以 angr 為主
- SimulationManager 通過 Strategy 走訪程式邏輯
  - 可以下參數，像是想走到的位置、不想碰到的位置
- Program State
  - 代表了當前程式的狀態
  - 可以設定記憶體狀態、各種預先定義好的符號
- SimulationManager aka simgr
  - 吃 State 當參數並模擬執行

# // Symbolic Execution?

- Claripy
  - 走訪過程收集各種限制式蓋出一顆 AST
  - 通過 Claripy 解開限制式找到 Solution

# // Symbolic Execution?

- Claripy 是 angr 的 Solver Engine
  - 會根據探索到的東西長出一顆 AST
    - 轉換後丟去 Solver Backend 跑
  - 目前 angr 會讓 claripy 套在 z3 上跑
    - 可以用 z3 以外的 Solver Backend
      - BackendVSA 之類的





# Resources

- <https://docs.angr.io/en/latest/appendix/cheatsheet.html>

# Lab

super\_angry

# Obfuscation

# // Obfuscation

- 通過 Obfuscator 將程式碼扭成麻花
  - 讓程式碼便的艱澀難懂
  - 編譯器優化就有點扭成麻花了
- 參雜垃圾 Code
- 一些有趣的例子
  - 在 Control-Flow Graph 上面畫畫的 REPsych, Artfuscator
  - 把 Control-Flow 變成一直線的 Movfuscator

# // Obfuscation

- 一些讓程式碼變噁爛的技巧
- 將 Constant 變成某種操作的產物
- 將 if 這種 branch 攤平
  - 用一個 loop 加 switch 來做分支攤平
- 將程式碼小片段變成某種 State Machine
  - 執行時 State Machine 會將程式碼照正常順序執行
  - 但觀察者頭會痛死

# // Obfuscation

- 一些讓程式碼變噁爛的技巧
  - 寫爛 Code

# // Obfuscation

- 一些將程式碼片段搞消失的方法
- 通過各種加載手段來映射片段的程式碼
  - VirtualAlloc, VirtualProtect, mmap, mprotect
- 目的是讓逆向者不能第一時間就看懂所有東西
  - 把程式切成小份到處亂丟，要用的時候再解包出來 map 到記憶體上

# Anti-Reverse

# // Anti-Revese

- 各種讓逆向工程師難過的技巧
  - Anti-Debug
  - Anti-Disassembler
  - Packing
  - Obfuscation
- 增加逆向的時間成本
  - 當這個成本遠高於收穫時，逆向工程師就放棄了

# // Anti-Revese

- 並不是全能的，往往都有對應的解法
  - 要花時間去認識並理解原理，進而 Bypass
  - 一兩個堆在一起還好，但一群堆在一起就不想看了
- 所以可以看到什麼 Anti Anti-Debug 之類的
  - 例如 Scylla Hide

# Packer

# // Packer

- 俗稱：殼；動詞：加殼、解殼 
- Packer 通常會將改變的 Executable 的模樣
  - 通常會有變化的地 方: Header, Segment, Code
- 上殼的目的
  - 減少 Executable 的大小
  - 混淆化 (Obfuscation)

# // Packer

- 有很多種殼 
- 加密殼
  - 將程式內容物加密
  - Themida, VMProtect, ASProtect
- 壓縮殼
  - 縮小執行檔
  - UPX, ASPack

# // Packer

- 有很多種殼 
- VM 殼
  - 將程式變成另外一種客製化的 Bytecode 並跑在 VM 裡面
  - VMProtect, Themida

# // 脫殼

- 脫殼方式
  - 找工具
    - 如果殼已經被破解並釋出工具的話，用工具最快
  - 人工脫殼
    - 看懂殼的邏輯並寫工具脫殼
  - 動態脫殼
    - 將程式跑起來，等待程式邏輯被解密後 Dump Memory

# Lab

## Unpackme

# Anti-Debug



# Anti-Debug

- Anti-Debug 是屬於避免自身 Process 被 Debug 的技巧
- 經常透過各種方式來檢查自己是不是被 Debug
  - 也有避免自己被 Attach 的技巧
  - 檢查方法從呼叫 Kernel API 到特別的 Process 結構都有

# Timer

# // Timer

- 時間？時間怎麼用來 Anti-Debug ?
  - 還真的可以
  - 比較時間差來測試自己是不是被 Debug
  - 在程式的各種地方插入檢查時間與製造 Delay
    - 可能是一個 Global Object 、也可能是寫檔案
    - 數據可能跟程式行為有關，不能冒然 Patch
  - 沒有穩定的解法，不過時間本身就不穩定
    - 程式不能做到很精確的測時

# // Timer

- 一些規避方法
- 做 Patch 直接把 Timer, Sleep 相關的東西全都 NOP 掉
- Hook 時間函數做 Speed Hack 讓時間變快
  - 可以用 CheatEngine 做 SpeedHack
  - 但要注意 Debugger 不能共存這件事情

# Anti-Attach

# // Anti-Attach

- 在 Windows 下，ntdll 有一個函數可以用來做到 Anti-Attach
  - DbgUiRemoteBreakin
  - Hook 這 Function 可以在 RemoteDbg 掛上的時候執行想要的 Code
    - 直接把程式關掉

# Win32 APIs

# // Anti-Debug: Win32 API

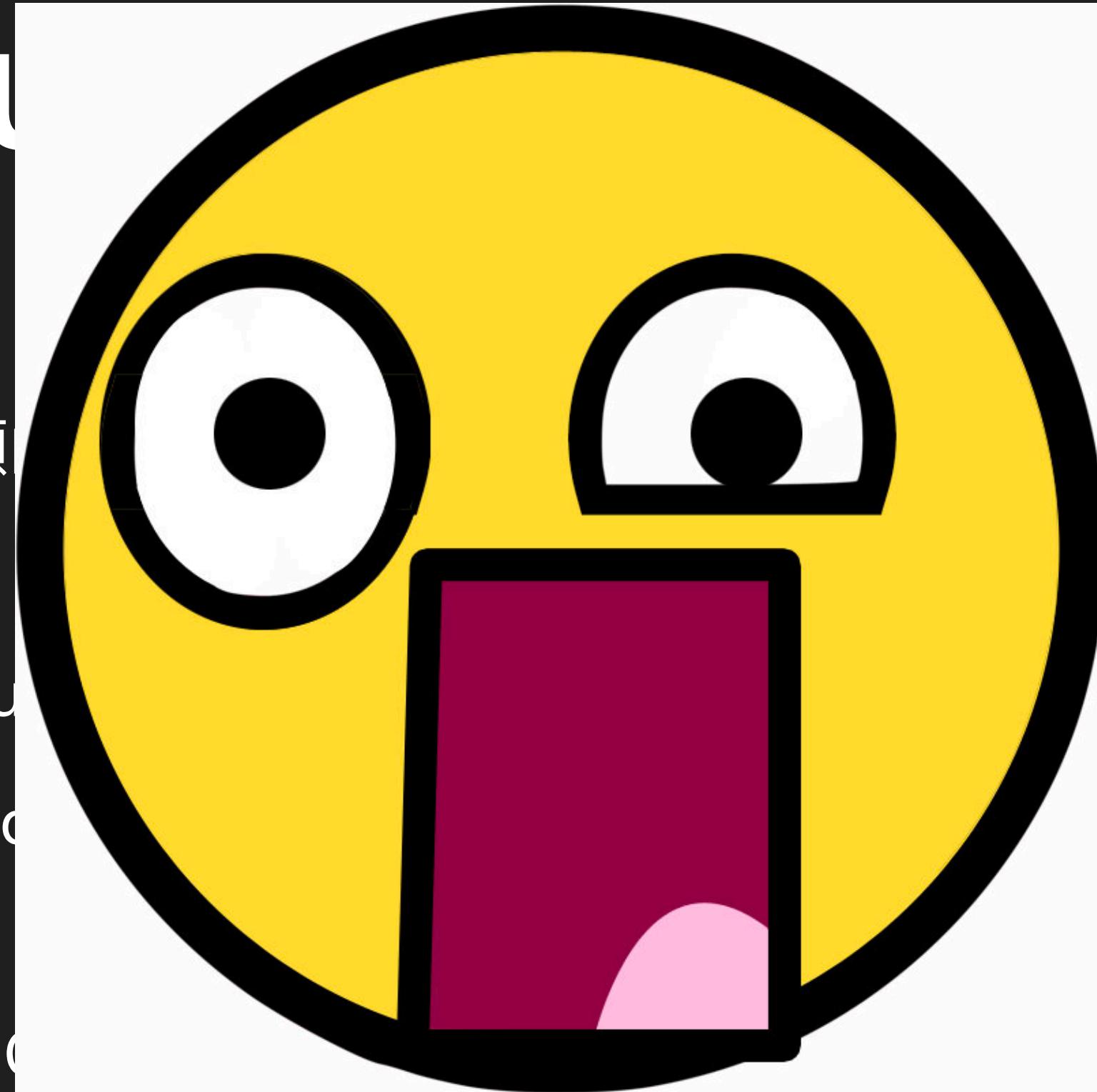
- IsDebuggerPresent()
  - 有一種變形是不通過 Win32 API 而是通過 PEB 的 Flag
  - 本質上就是這個 API 在做的事情
- CheckRemoteDebuggerPresent()
- RtlQueryProcessHeapInformation()
- RtlQueryProcessDebugInformation()
- NtQuerySystemInformation()

# // Anti-Debug: Win32 API

- NtQueryInformationProcess()
  - 這個 API 可以 Query 很多種類的 Process Information
  - ProcessDebugPort
    - 如果 Port != 0 就是被 Debug
    - MSDN: A nonzero value indicates that the process is being run under the control of a ring 3 debugger
  - ProcessDebugFlags
    - 有一個 Undocumented 的 Class: ProcessDebugFlags
  - ProcessDebugObjectHandle
    - 被 Debug 時，Kernel 內會有一個 debug object 被生成
    - 一樣可以被 Undocumented 的 Class 查詢
    - ProcessDebugObjectHandle (0x1e)

# // Anti-Debug

- NtQueryInformationProcess()
  - 這個 API 可以 Query 很多種類型的資訊
- ProcessDebugPort
  - 如果 Port != 0 就是被 Debugger 附上
  - MSDN: A nonzero value indicates the process is under the control of a ring 3 debugger
- ProcessDebugFlags
  - 有一個 Undocumented 的參數



有沒有搞錯，為啥這麼多 Undocumented 的東西  
還都用來針對逆向工程師

- ProcessDebugObjectHandle (UXIE)

# // Anti-Debug: Win32 API

- NtQueryInformationProcess
  - 這個 API 可以查詢到很多資訊
  - ProcessDebugInformation
  - 如果 ProcessDebugInformation = 1
  - MSDN 沒有說明
  - ProcessDebugInformation
  - 有一個位元是 1
  - ProcessDebugInformation
  - 被 Debugged
  - 一樣可以被 Undocumented 的 Class 查詢
  - ProcessDebugObjectHandle (0x1e)



因為是微軟啊

ion

cess is

gFlags

被生成

# // Anti-Debug: Win32 API

- 如何逃逸？
- Hook 掉這些 Function 就好，讓回傳值跟沒被 Debug 的數值一樣

# // Anti-Debug: Win32 API

- 如何逃逸？
- Hook 掉這些 Function 就好，讓回傳值跟沒被 Debug 的數值一樣
- 嘛？

# // Anti-Debug: Win32 API

- 如何逃逸？
- Hook 掉這些 Function 就好，讓回傳值跟沒被 Debug 的數值一樣
- 嘛？
  - Hook 也是可以被偵測的
  - 商業 Anti-Debug 各家的實作不太一樣，隨便 Hook 也會有機會被發現

# DEMO

Scylla Hide

# Exception Handler

# // Exception Handler

- Debugger 通常都會接住 Tracee 的 Exception
  - 可以用來檢查 Debugger 是否存在
  - 刻意製造 Exception 觀察是否接到自己彈出的 Exception

# // Exception Handler

- Windows 上的 Exception Handling 機制是 SEH
  - Structured Exception Handling
  - 跟 C++ 的 Exception 是兩回事
- 本質上是一個鏈狀結構體，會不斷 Match Exception
  - 如果沒有 Match 就往下一個 Block 移動繼續做 Matching
- 可以自己註冊 Exception Handler 上去 SEH

# // Exception Handler

- 只有在 Windows 上才寫得出來的東西

```
__try
{
    TestExceptions();
}

__except(EXCEPTION_EXECUTE_HANDLER)
{
    printf("Executing SEH __except block\n");
}
```

•

# // Anti Exception Handler

- 看懂程式流程在哪裡被接走
  - 做 Patch
  - 手動繞過
  - 直接改 Register

# Anti-Disassemble



# Anti-Disassemble

- 讓 Disassembler 壞掉的小技巧
- Disassembler 的運作機制
  - 線性掃描
  - Control-Flow 分析

# // Anti-Disassemble

- 針對線性掃描
- 因為指令集密度很高，如果在程式中製造 Offset...
  - 可能會解出看起來對的程式碼
  - 但行為可能根本不一樣

# // Anti-Disassemble

- 針對 Control-flow Based Disassembler
- 因為這種 Disassembler 會根據 Control-Flow 做追蹤
  - 如果利用假的 jmp 指令來跳躍
  - 可以使分析頭跳到奇怪的地方，然後就解壞了

# // Anti Anti-Disassembler

- 怎麼辦...東西壞光光了...
- 如果你都動態解析，頂多 Debugger 跳針一下，但就會繼續跑下去
- 手動修好，這種技巧只會影響到靜態分析
  - 把解析壞掉的部分 `Undefine` 掉
  - 找到對的 `Code` 開始點再標記回去

# Thanks for attention!