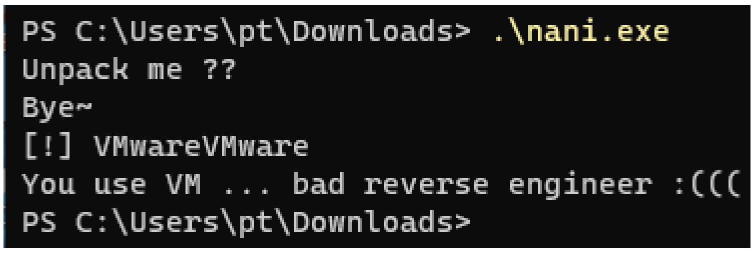
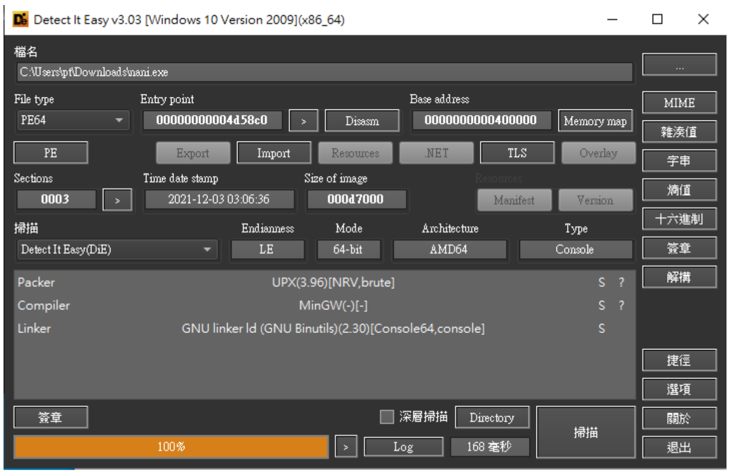
nani

- 直接執行:



```
PS C:\Users\pt\Downloads> .\nani.exe
Unpack me ??
Bye~
[!] VMwareVMware
You use VM ... bad reverse engineer :(((
PS C:\Users\pt\Downloads>
```

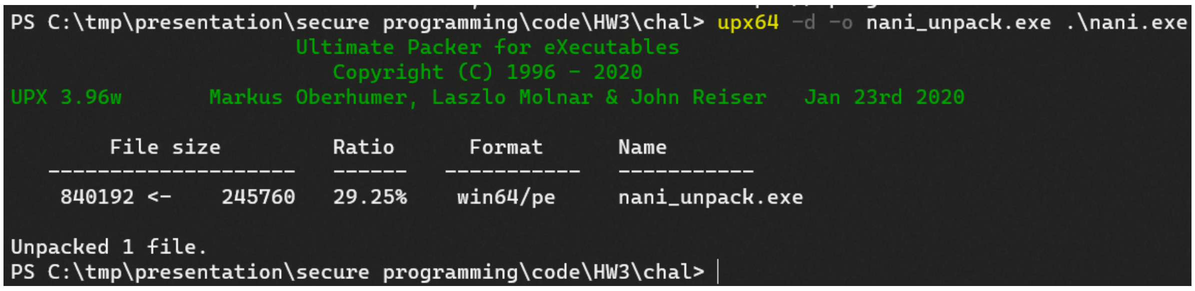
- 以 UPX 3.96 加殼:



Detect It Easy v3.03 [Windows 10 Version 2009](x86_64)

```
檔名: C:\Users\pt\Downloads\nani.exe
File type: PE64    Entry point: 00000000004d58c0    Base address: 0000000000400000
Sections: 0003    Time date stamp: 2021-12-03 03:06:36    Size of image: 000d7000
掃描: Detect It Easy(DiE)    Endianness: LE    Mode: 64-bit    Architecture: AMD64    Type: Console

Packer      UPX(3.96)[NRV,brute]                                S ?
Compiler    MinGW(-)[-]                                         S ?
Linker      GNU linker ld (GNU Binutils)(2.30)[Console64,console]  S

100%    168 毫秒
```

- 以 UPX 3.96 脫殼:
    - `upx.exe -d -o nani_unpack.exe nani.exe`



```
PS C:\tmp\presentation\secure programming\code\HW3\chal> upx64 -d -o nani_unpack.exe .\nani.exe
                       Ultimate Packer for eXecutables
                          Copyright (C) 1996 - 2020
UPX 3.96w       Markus Oberhumer, Laszlo Molnar & John Reiser   Jan 23rd 2020

        File size         Ratio      Format      Name
   --------------------   ------   -----------   -----------
    840192 <-    245760   29.25%    win64/pe     nani_unpack.exe

Unpacked 1 file.
PS C:\tmp\presentation\secure programming\code\HW3\chal>
```

- `main`:

```
push    rbp
mov     rbp, rsp
sub     rsp, 30h
call    sub_40B260
nop
nop
nop
nop
nop
nop
nop
mov     [rbp+var_8], rax
nop
nop
```

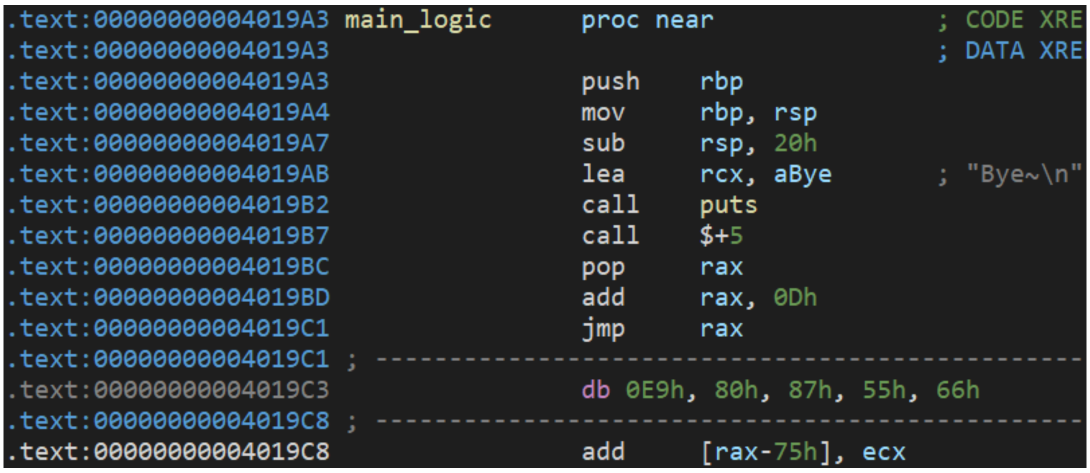
```
nop
nop
nop
nop
nop
mov     [rbp+var_8], rax
nop
nop
nop
nop
nop
nop
nop
mov     [rbp+var_8], rax
lea     rcx, str_unpackme ; "Unpack me "
call    puts
call    main_logic
mov     eax, 0
add     rsp, 30h
pop     rbp
retn
```

- `sub_40B260` 函數不為 user 寫作的, 可以實際編一個測試程式出來看, 會有相同的函數, 以下為實驗用的程式, 以 mingw 編譯:

  ```
  // test.cpp
  #include <stdio.h>

  int main()
  {
      printf("Test\n");
  }
  ```

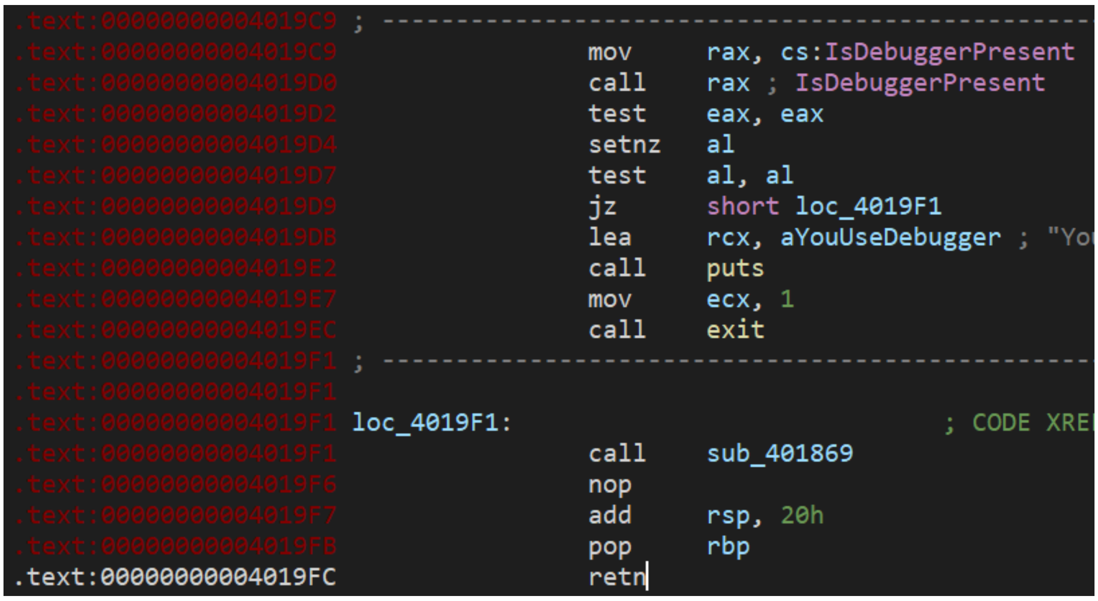
  ```
  # Makefile
  # use mingw32-make.exe to make

  CFLAGS = -Wl,--dynamicbase -s -static

  all:
      @echo "Compiling..."
      g++ $(CFLAGS) -o test.exe test.cpp
  ```

- `main_logic`:
  - 邏輯如下:



```
.text:00000000004019A3 main_logic      proc near               ; CODE XRE
.text:00000000004019A3                                         ; DATA XRE
.text:00000000004019A3                 push    rbp
.text:00000000004019A4                 mov     rbp, rsp
.text:00000000004019A7                 sub     rsp, 20h
.text:00000000004019AB                 lea     rcx, aBye       ; "Bye~\n"
.text:00000000004019B2                 call    puts
.text:00000000004019B7                 call    $+5
.text:00000000004019BC                 pop     rax
.text:00000000004019BD                 add     rax, 0Dh
.text:00000000004019C1                 jmp     rax
.text:00000000004019C1 ; ---------------------------------------------
.text:00000000004019C3                 db 0E9h, 80h, 87h, 55h, 66h
.text:00000000004019C8 ; ---------------------------------------------
.text:00000000004019C8                 add     [rax-75h], ecx
```

- `call $+5` 呼叫 0x4019bc, 下一條指令作為 return address 0x4019bc 被推進 stack, 將其 pop 至 rax, 並且加 0xd 後跳過去
- 因此位址是 0x4019bc + 0xd = 0x4019c9
- 將 0x4019c8 undefine (按 `u`), 並在 0x4019c9 定義成 code (按 `c`):



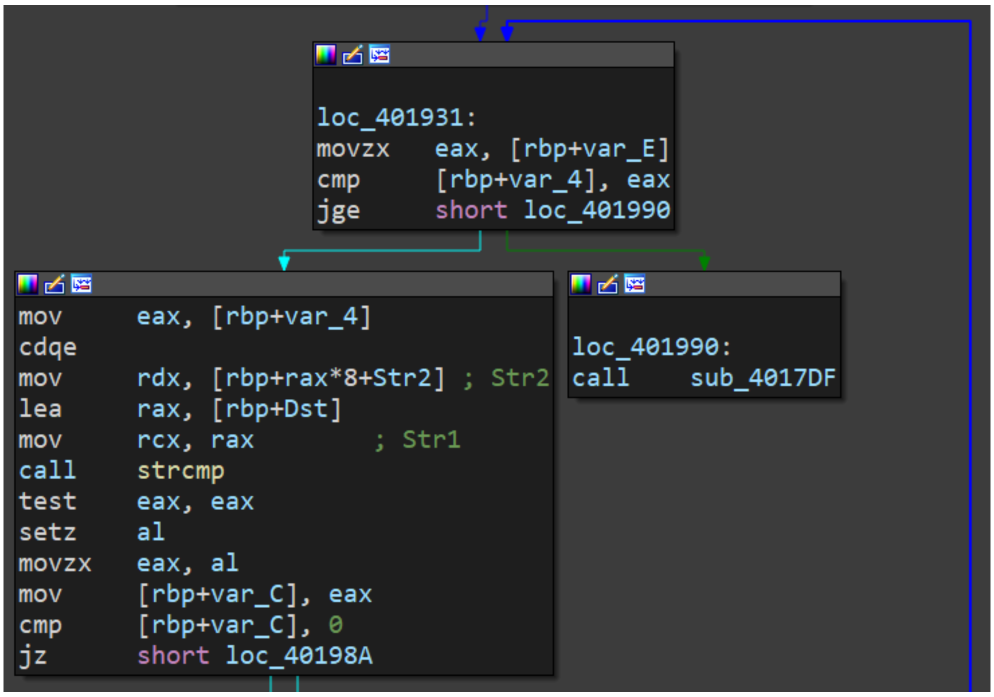
```
.text:00000000004019C9 ; ---------------------------------------------
.text:00000000004019C9                 mov     rax, cs:IsDebuggerPresent
.text:00000000004019D0                 call    rax ; IsDebuggerPresent
.text:00000000004019D2                 test    eax, eax
.text:00000000004019D4                 setnz   al
.text:00000000004019D7                 test    al, al
.text:00000000004019D9                 jz      short loc_4019F1
.text:00000000004019DB                 lea     rcx, aYouUseDebugger ; "Yo
.text:00000000004019E2                 call    puts
.text:00000000004019E7                 mov     ecx, 1
.text:00000000004019EC                 call    exit
.text:00000000004019F1 ; ---------------------------------------------
.text:00000000004019F1
.text:00000000004019F1 loc_4019F1:                             ; CODE XRE
.text:00000000004019F1                 call    sub_401869
.text:00000000004019F6                 nop
.text:00000000004019F7                 add     rsp, 20h
.text:00000000004019FB                 pop     rbp
.text:00000000004019FC                 retn
```

- 呼叫了 `IsDebuggerPresent` 偵測是否正在被 debug 中, 接著呼叫 `sub_401869`

- `sub_401869`:
  - 有一段以 cpuid 指令偵測是否在 VM 的程式碼:

```
mov     rax, 40000000h
cpuid
mov     [rbp-1Ch], ebx
mov     [rbp-18h], ecx
mov     [rbp-14h], edx
```

  - 若沒有偵測到 VM 特徵字串, 則呼叫 `sub_4017df`:

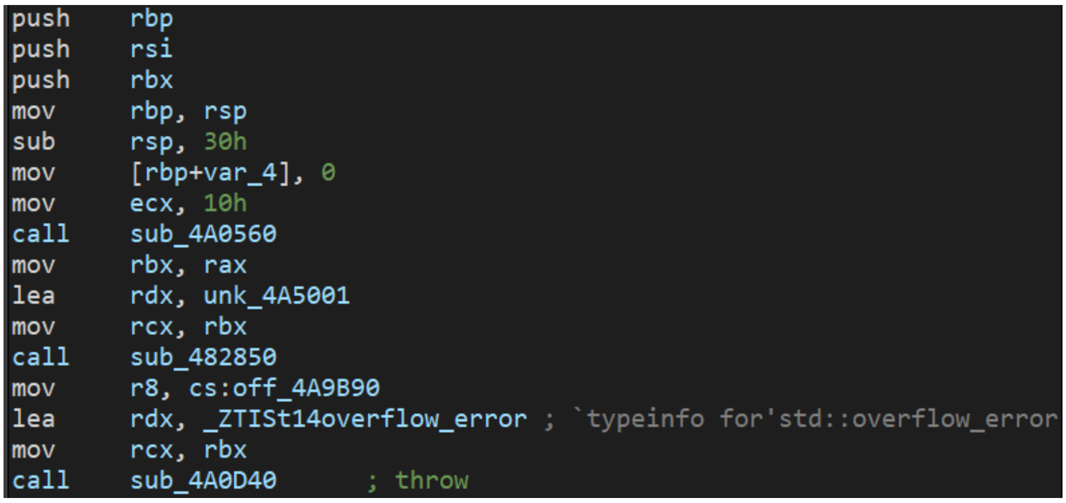


```
loc_401931:
movzx   eax, [rbp+var_E]
cmp     [rbp+var_4], eax
jge     short loc_401990
```

```
mov     eax, [rbp+var_4]
cdqe
mov     rdx, [rbp+rax*8+Str2] ; Str2
lea     rax, [rbp+Dst]
mov     rcx, rax        ; Str1
call    strcmp
test    eax, eax
setz    al
movzx   eax, al
mov     [rbp+var_C], eax
cmp     [rbp+var_C], 0
jz      short loc_40198A
```

```
loc_401990:
call    sub_4017DF
```

- `sub_4017df`:
  - 這邊 throw 了錯誤:

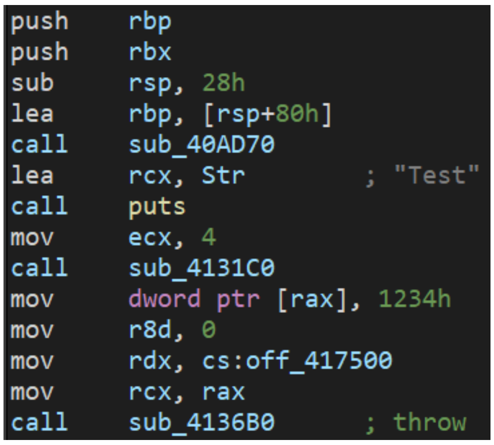


```
push    rbp
push    rsi
push    rbx
mov     rbp, rsp
sub     rsp, 30h
mov     [rbp+var_4], 0
mov     ecx, 10h
call    sub_4A0560
mov     rbx, rax
lea     rdx, unk_4A5001
mov     rcx, rbx
call    sub_482850
mov     r8, cs:off_4A9B90
lea     rdx, _ZTISt14overflow_error ; `typeinfo for'std::overflow_error
mov     rcx, rbx
call    sub_4A0D40      ; throw
```

  - 可以實際寫一個程式對照, 以下為實驗程式碼以及相關截圖:

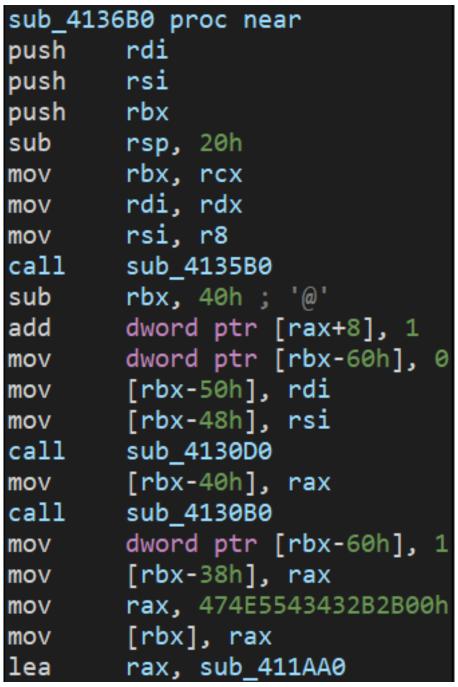
```cpp
// test.cpp
#include <stdio.h>

int main()
{
    try {
        printf("Test\n");
        throw 0x1234;
    } catch (...) {
        printf("GG\n");
    }
}
```

```
# Makefile
# use mingw32-make.exe to make

CFLAGS = -Wl,--dynamicbase -s -static

all:
    @echo "Compiling..."
    g++ $(CFLAGS) -o test.exe test.cpp
```



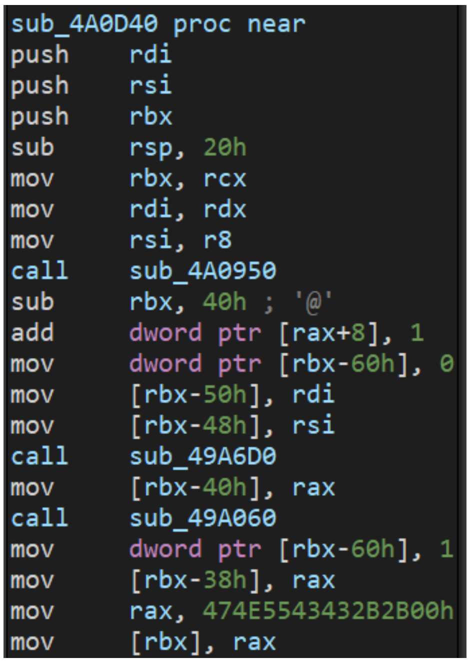
```
push    rbp
push    rbx
sub     rsp, 28h
lea     rbp, [rsp+80h]
call    sub_40AD70
lea     rcx, Str          ; "Test"
call    puts
mov     ecx, 4
call    sub_4131C0
mov     dword ptr [rax], 1234h
mov     r8d, 0
mov     rdx, cs:off_417500
mov     rcx, rax
call    sub_4136B0        ; throw
```

- 可以看到在 `puts("Test")` 後, 呼叫了 `sub_4131c0`, 並往其返回的 pointer 指向的位址放 0x1234, 之後呼叫 `sub_4136b0`
- 可以簡易判斷是 `sub_4131c0` 為 throw 出的 object 創出空間, 而 `sub_4136b0` 實際執行 throw 的行為, 查看他的程式碼:



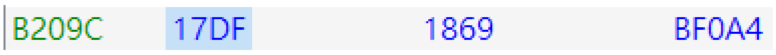
```
sub_4136B0 proc near
push    rdi
push    rsi
push    rbx
sub     rsp, 20h
mov     rbx, rcx
mov     rdi, rdx
mov     rsi, r8
call    sub_4135B0
sub     rbx, 40h ; '@'
add     dword ptr [rax+8], 1
mov     dword ptr [rbx-60h], 0
mov     [rbx-50h], rdi
mov     [rbx-48h], rsi
call    sub_4130D0
mov     [rbx-40h], rax
call    sub_4130B0
mov     dword ptr [rbx-60h], 1
mov     [rbx-38h], rax
mov     rax, 474E5543432B2B00h
mov     [rbx], rax
lea     rax, sub_411AA0
```

- 可以看到 `sub_4136b0` 內部有一個特別的 const `474E5543432B2B00h`
- 回頭比對 `nani.exe` 中懷疑是 throw 的函數內部:



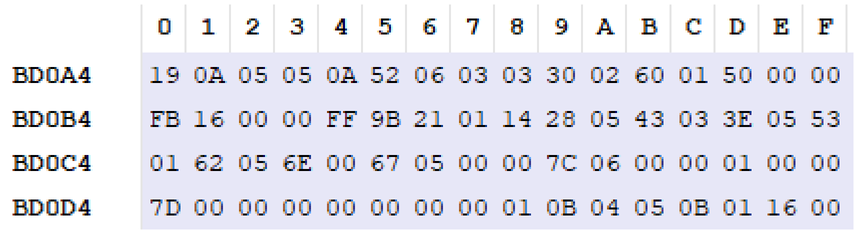
```
sub_4A0D40 proc near
push    rdi
push    rsi
push    rbx
sub     rsp, 20h
mov     rbx, rcx
mov     rdi, rdx
mov     rsi, r8
call    sub_4A0950
sub     rbx, 40h ; '@'
add     dword ptr [rax+8], 1
mov     dword ptr [rbx-60h], 0
mov     [rbx-50h], rdi
mov     [rbx-48h], rsi
call    sub_49A6D0
mov     [rbx-40h], rax
call    sub_49A060
mov     dword ptr [rbx-60h], 1
mov     [rbx-38h], rax
mov     rax, 474E5543432B2B00h
mov     [rbx], rax
```

        - 兩函數相同, 因此能確定 `nani.exe` 的 `sub_4a0d40` 為 throw
    - 查看呼叫了 throw 的 `sub_4017df` 在 Exception Directory 是否有 unwind info (欄位分別為 Offset, BeginAddress, EndAddress, UnwindInfoAddress):



B209C	17DF	1869	BF0A4

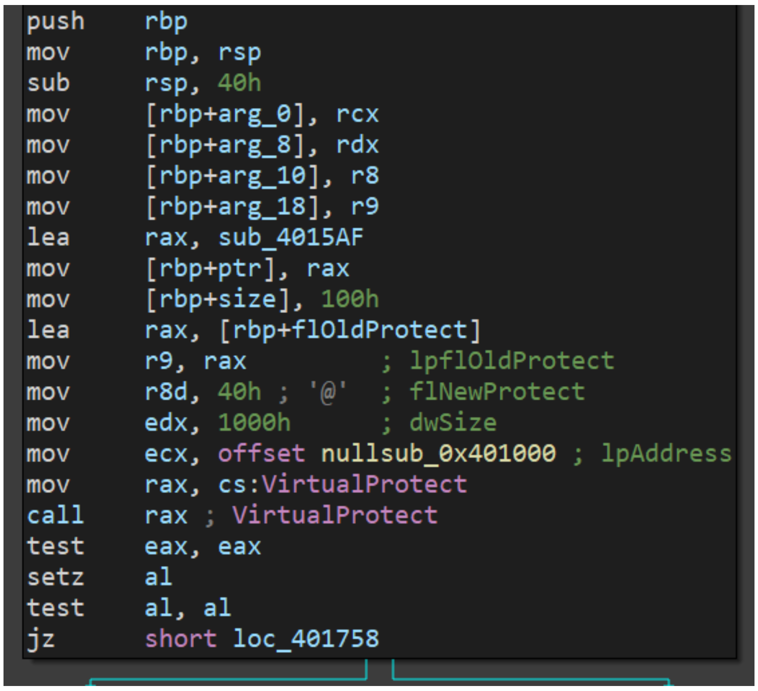
    - 查看 RVA 0xBF0A4 內容 (對應 Raw offset 為 0xBD0A4):



	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
BD0A4	19	0A	05	05	0A	52	06	03	03	30	02	60	01	50	00	00
BD0B4	FB	16	00	00	FF	9B	21	01	14	28	05	43	03	3E	05	53
BD0C4	01	62	05	6E	00	67	05	00	00	7C	06	00	00	01	00	00
BD0D4	7D	00	00	00	00	00	00	00	01	0B	04	05	0B	01	16	00

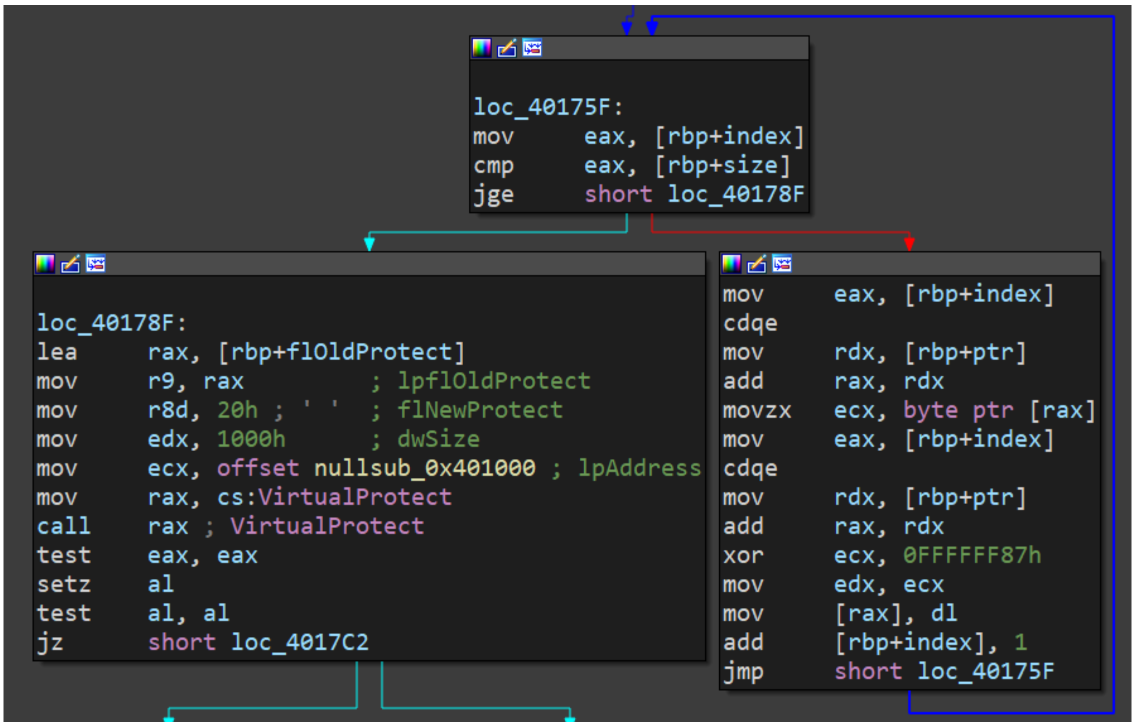
        - 0xBD0A4 + 2: CountOfCodes 為 0x5
        - 0xBD0A4 + 4: 為 UnwindCode 陣列, 大小為 0x2 * 0x5 = 0xa, 因此一直到 0xBD0A4 + 0xd 皆為 UnwindCode 陣列範圍
        - 0xBD0A4 + 0x10: 由於有 MoreUnwindCode 會使 ExceptionHandler 對齊 4 倍數的 address, 因此 ExceptionHandler 是從此開始, 為 0xfb160000, 以 little endian 來讀, 為 0x000016fb, 加上 ImageBase 就能得到 handler 位址為 0x4016fb

- `0x4016fb` exception handler:
    - 首先呼叫 `VirtualProtect`, 將 0x401000 開始的 0x1000 address space 改成 RWX, 如下圖:



```
push    rbp
mov     rbp, rsp
sub     rsp, 40h
mov     [rbp+arg_0], rcx
mov     [rbp+arg_8], rdx
mov     [rbp+arg_10], r8
mov     [rbp+arg_18], r9
lea     rax, sub_4015AF
mov     [rbp+ptr], rax
mov     [rbp+size], 100h
lea     rax, [rbp+flOldProtect]
mov     r9, rax         ; lpflOldProtect
mov     r8d, 40h ; '@'  ; flNewProtect
mov     edx, 1000h      ; dwSize
mov     ecx, offset nullsub_0x401000 ; lpAddress
mov     rax, cs:VirtualProtect
call    rax ; VirtualProtect
test    eax, eax
setz    al
test    al, al
jz      short loc_401758
```

- 注意上圖另外初始化幾個變數:
  - `ptr`: `sub_4015af`
  - `size`: 0x100
- 再來將 `ptr` 指向的位址內容 xor 0x87, 共 patch `size` 個 bytes, patch 後將 address space 改回原本的權限:

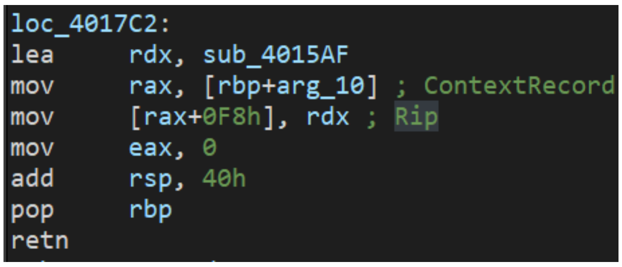


```
loc_40175F:
mov     eax, [rbp+index]
cmp     eax, [rbp+size]
jge     short loc_40178F
```

```
loc_40178F:
lea     rax, [rbp+flOldProtect]
mov     r9, rax         ; lpflOldProtect
mov     r8d, 20h ; ' '  ; flNewProtect
mov     edx, 1000h      ; dwSize
mov     ecx, offset nullsub_0x401000 ; lpAddress
mov     rax, cs:VirtualProtect
call    rax ; VirtualProtect
test    eax, eax
setz    al
test    al, al
jz      short loc_4017C2
```

```
mov     eax, [rbp+index]
cdqe
mov     rdx, [rbp+ptr]
add     rax, rdx
movzx   ecx, byte ptr [rax]
mov     eax, [rbp+index]
cdqe
mov     rdx, [rbp+ptr]
add     rax, rdx
xor     ecx, 0FFFFFF87h
mov     edx, ecx
mov     [rax], dl
add     [rbp+index], 1
jmp     short loc_40175F
```

- 最終將第三個參數 `ContextRecord` 的 `Rip` 改成 `sub_4015af`:



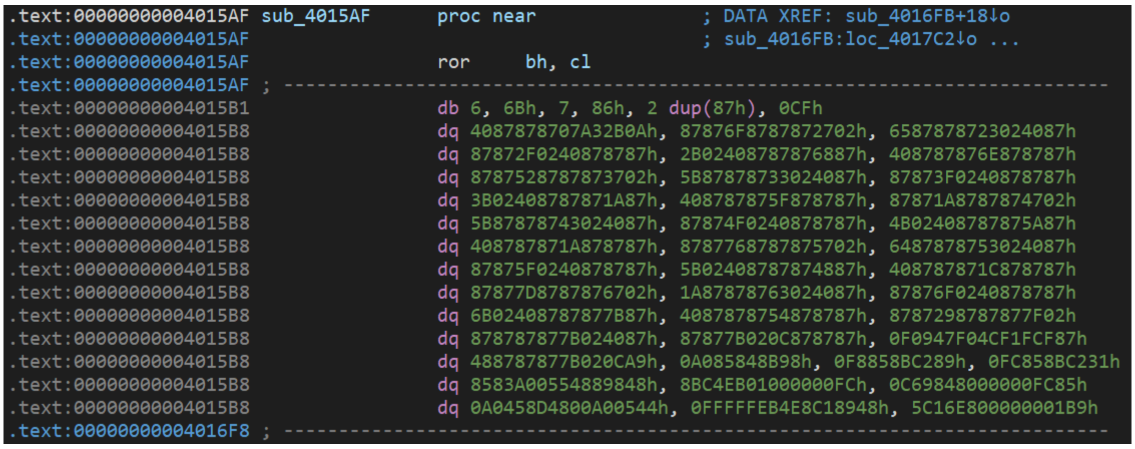
```
loc_4017C2:
lea     rdx, sub_4015AF
mov     rax, [rbp+arg_10] ; ContextRecord
mov     [rax+0F8h], rdx ; Rip
mov     eax, 0
add     rsp, 40h
pop     rbp
retn
```

- `struct _CONTEXT` 可以參考本連結, offset 0xf8 位址為 Rip

  - 統整一下, 此 handler 作用是把 `sub_4015af` patch 0x100 bytes, patch 方式為 xor 0x87, 並且跳過去執行
- `sub_4015af`:
  - 在 IDA 中直接 patch 他並且分析, 運行以下腳本:

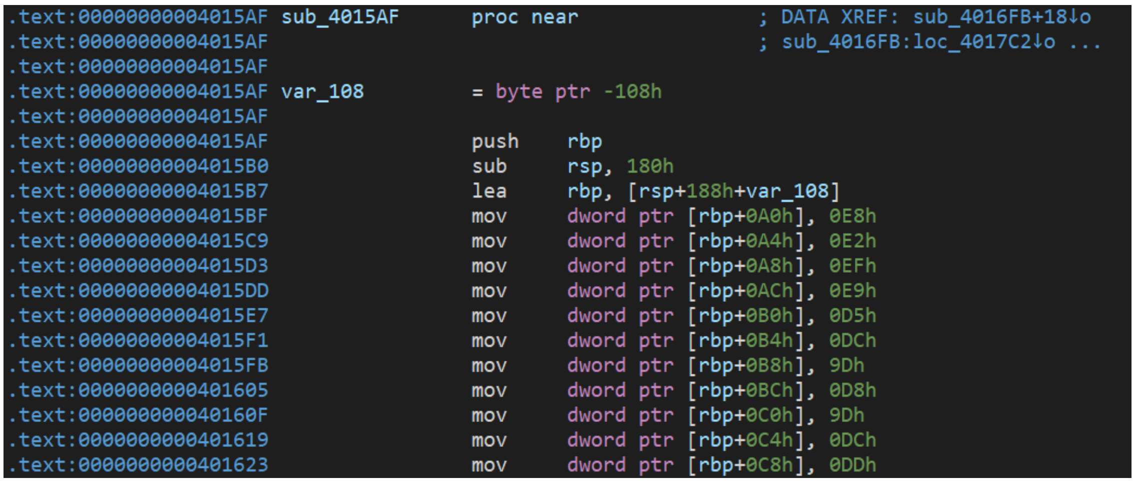
```python
# Run in IDA python

start_addr = 0x4015af
size       = 0x100

for i in range(size):
    b = get_bytes(start_addr + i, 1, False)
    PatchByte(start_addr + i, ord(b) ^ 0x87)
```

  - Before patch:



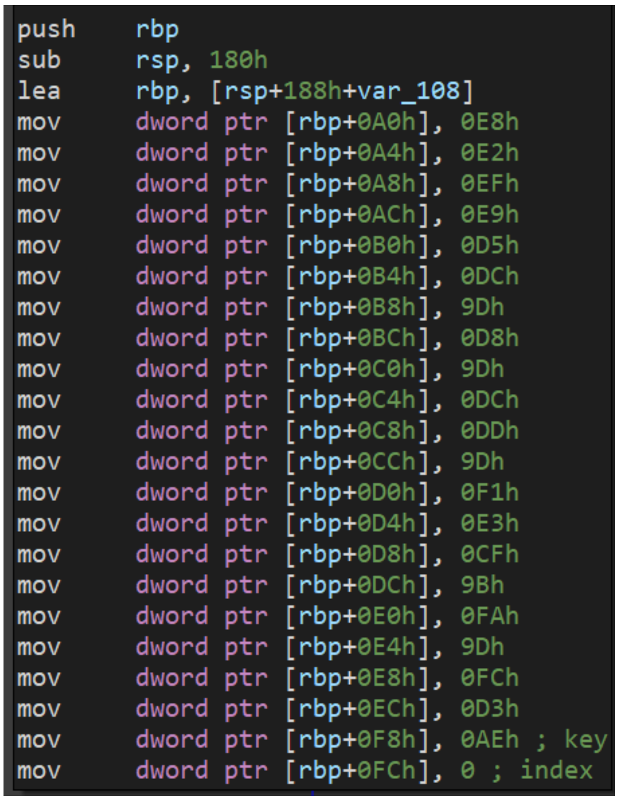
```
.text:00000000004015AF sub_4015AF      proc near               ; DATA XREF: sub_4016FB+18↓o
.text:00000000004015AF                                         ; sub_4016FB:loc_4017C2↓o ...
.text:00000000004015AF                 ror     bh, cl
.text:00000000004015AF ; ---------------------------------------------------------------------------
.text:00000000004015B1                 db 6, 6Bh, 7, 86h, 2 dup(87h), 0CFh
.text:00000000004015B8                 dq 4087878707A32B0Ah, 87876F8787872702h, 6587878723024087h
.text:00000000004015B8                 dq 87872F0240878787h, 2B02408787876887h, 408787876E878787h
.text:00000000004015B8                 dq 8787528787873702h, 5B87878733024087h, 87873F0240878787h
.text:00000000004015B8                 dq 3B02408787871A87h, 408787875F878787h, 87871A8787874702h
.text:00000000004015B8                 dq 5B87878743024087h, 87874F0240878787h, 4B02408787875A87h
.text:00000000004015B8                 dq 408787871A878787h, 8787768787875702h, 6487878753024087h
.text:00000000004015B8                 dq 87875F0240878787h, 5B02408787874887h, 408787871C878787h
.text:00000000004015B8                 dq 87877D8787876702h, 1A87878763024087h, 87876F0240878787h
.text:00000000004015B8                 dq 6B02408787877B87h, 4087878754878787h, 878729878787 7F02h
.text:00000000004015B8                 dq 878787877B024087h, 87877B020C878787h, 0F0947F04CF1FCF87h
.text:00000000004015B8                 dq 488787877B020CA9h, 0A085848B98h, 0F8858BC289h, 0FC858BC231h
.text:00000000004015B8                 dq 8583A00554889848h, 8BC4EB01000000FCh, 0C69848000000FC85h
.text:00000000004015B8                 dq 0A0458D4800A00544h, 0FFFFFEB4E8C18948h, 5C16E800000001B9h
.text:00000000004016F8 ; ---------------------------------------------------------------------------
```

  - After:



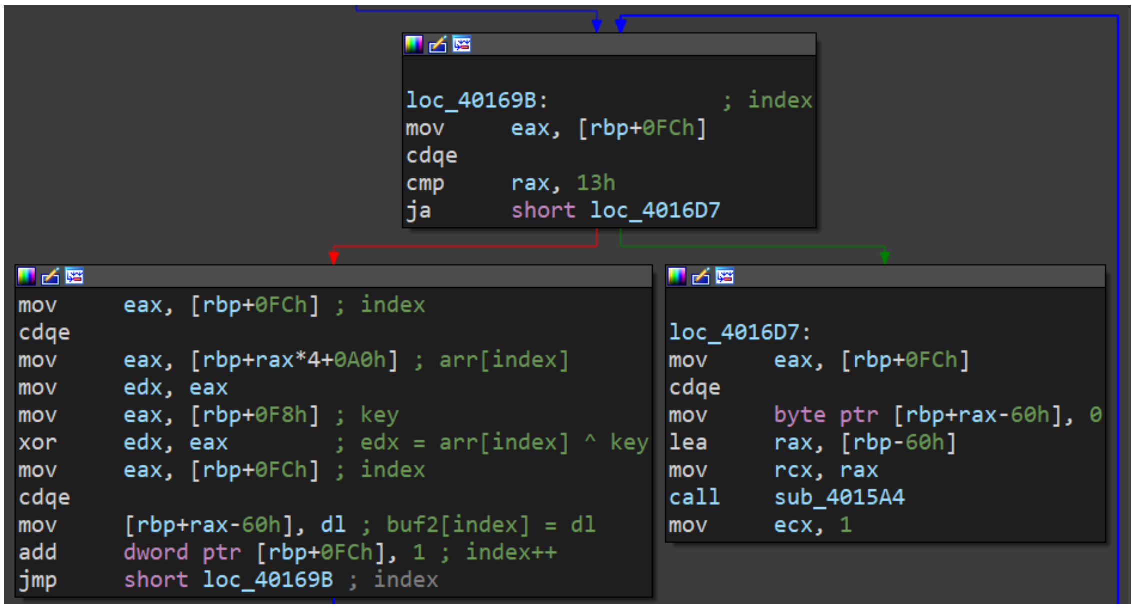
```
.text:00000000004015AF sub_4015AF      proc near               ; DATA XREF: sub_4016FB+18↓o
.text:00000000004015AF                                         ; sub_4016FB:loc_4017C2↓o ...
.text:00000000004015AF
.text:00000000004015AF var_108         = byte ptr -108h
.text:00000000004015AF
.text:00000000004015AF                 push    rbp
.text:00000000004015B0                 sub     rsp, 180h
.text:00000000004015B7                 lea     rbp, [rsp+188h+var_108]
.text:00000000004015BF                 mov     dword ptr [rbp+0A0h], 0E8h
.text:00000000004015C9                 mov     dword ptr [rbp+0A4h], 0E2h
.text:00000000004015D3                 mov     dword ptr [rbp+0A8h], 0EFh
.text:00000000004015DD                 mov     dword ptr [rbp+0ACh], 0E9h
.text:00000000004015E7                 mov     dword ptr [rbp+0B0h], 0D5h
.text:00000000004015F1                 mov     dword ptr [rbp+0B4h], 0DCh
.text:00000000004015FB                 mov     dword ptr [rbp+0B8h], 9Dh
.text:0000000000401605                 mov     dword ptr [rbp+0BCh], 0D8h
.text:000000000040160F                 mov     dword ptr [rbp+0C0h], 9Dh
.text:0000000000401619                 mov     dword ptr [rbp+0C4h], 0DCh
.text:0000000000401623                 mov     dword ptr [rbp+0C8h], 0DDh
```

  - 首先初始化陣列:



```
push    rbp
sub     rsp, 180h
lea     rbp, [rsp+188h+var_108]
mov     dword ptr [rbp+0A0h], 0E8h
mov     dword ptr [rbp+0A4h], 0E2h
mov     dword ptr [rbp+0A8h], 0EFh
mov     dword ptr [rbp+0ACh], 0E9h
mov     dword ptr [rbp+0B0h], 0D5h
mov     dword ptr [rbp+0B4h], 0DCh
mov     dword ptr [rbp+0B8h], 9Dh
mov     dword ptr [rbp+0BCh], 0D8h
mov     dword ptr [rbp+0C0h], 9Dh
mov     dword ptr [rbp+0C4h], 0DCh
mov     dword ptr [rbp+0C8h], 0DDh
mov     dword ptr [rbp+0CCh], 9Dh
mov     dword ptr [rbp+0D0h], 0F1h
mov     dword ptr [rbp+0D4h], 0E3h
mov     dword ptr [rbp+0D8h], 0CFh
mov     dword ptr [rbp+0DCh], 9Bh
mov     dword ptr [rbp+0E0h], 0FAh
mov     dword ptr [rbp+0E4h], 9Dh
mov     dword ptr [rbp+0E8h], 0FCh
mov     dword ptr [rbp+0ECh], 0D3h
mov     dword ptr [rbp+0F8h], 0AEh ; key
mov     dword ptr [rbp+0FCh], 0 ; index
```

- 解密陣列:



```
loc_40169B:                     ; index
mov     eax, [rbp+0FCh]
cdqe
cmp     rax, 13h
ja      short loc_4016D7
```

```
mov     eax, [rbp+0FCh] ; index
cdqe
mov     eax, [rbp+rax*4+0A0h] ; arr[index]
mov     edx, eax
mov     eax, [rbp+0F8h] ; key
xor     edx, eax        ; edx = arr[index] ^ key
mov     eax, [rbp+0FCh] ; index
cdqe
mov     [rbp+rax-60h], dl ; buf2[index] = dl
add     dword ptr [rbp+0FCh], 1 ; index++
jmp     short loc_40169B ; index
```

```
loc_4016D7:
mov     eax, [rbp+0FCh]
cdqe
mov     byte ptr [rbp+rax-60h], 0
lea     rax, [rbp-60h]
mov     rcx, rax
call    sub_4015A4
mov     ecx, 1
```

- 等校腳本如下:

```python
#!/usr/bin/env python3

arr = [0xe8, 0xe2, 0xef, 0xe9, 0xd5, 0xdc, 0x9d, 0xd8, 0x9d, 0xdc, 0xdd,
0x9d, 0xf1, 0xe3, 0xcf, 0x9b, 0xfa, 0x9d, 0xfc, 0xd3]
key = 0xae

print(bytes([x ^ key for x in arr]))
```

`b'FLAG{r3v3rs3_Ma5T3R}'`