Exception

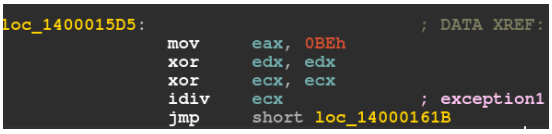
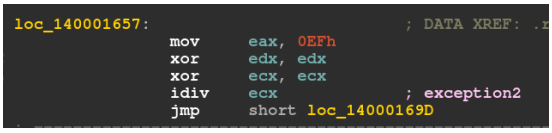
這題是要瞭解Windows的SEH(Structured Exception Handling)機制，使用時需要include Windows.h，來使用他的API

SCOPE_RECORD

這是用來處理Exception的一個struct，有4個資料: Begin, End, Handler, JumpTarget來描述exception的具體範圍與處理方式。

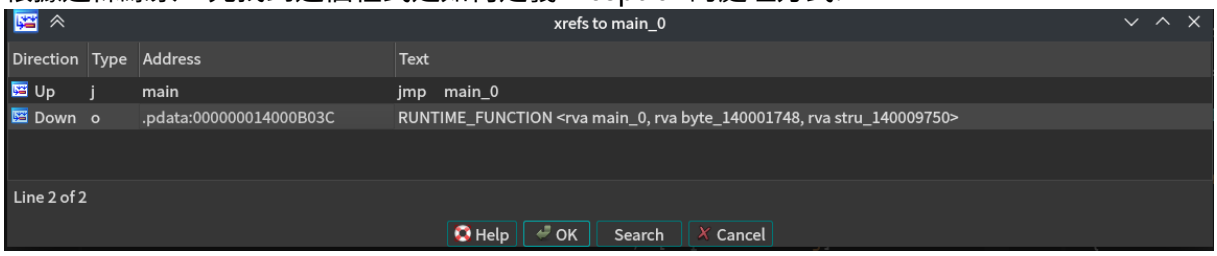
Discovery

首先觀察main，可以看到照著流程執行的話，會出現divde-by-zero的exception(2次):

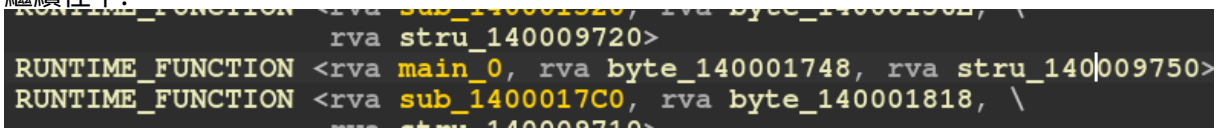
1400015D5	140001657
	

(需要注意的是這邊將eax設定爲0xbe與0xef，稍後會用到)

根據這條線索，先找到這個程式是如何定義Exception的處理方式:



繼續往下:



就可以找到定義的位置

```
stru_140009750  UNWIND_INFO_HDR <9, 0Eh, 4, 25h>
                                        ; DATA XREF: .pdata:000000014000B03C↓o
                UNWIND_CODE <0Eh, 23h>  ; UWOP_SET_FPREG
                UNWIND_CODE <9, 1>      ; UWOP_ALLOC_LARGE
                dw 1Eh
                UNWIND_CODE <2, 50h>    ; UWOP_PUSH_NONVOL
                dd rva sub_140001400
                dd 3
                db 0D5h
                db  15h
                db    0
                db    0
                db 0E2h
                db  15h
                db    0
                db    0
                db  70h ; p
                db  61h ; a
                db    0
                db    0
                db 0E2h
```

接下來定義structure，以便檢視資訊:

```
0000000
0000000 SCOPE_RECORD    struc ; (sizeof=0x10, mappedto_43)
0000000                                       ; XREF: .rdata
0000000 begin           dd ?                  ; offset rva
0000004 end             dd ?                  ; offset rva
0000008 handler         dd ?                  ; offset rva
000000C jumpTarget      dd ?                  ; offset rva
0000010 SCOPE_RECORD    ends
0000010
```

定義好之後就可以轉成我們想要的資訊:

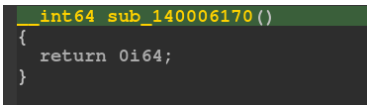
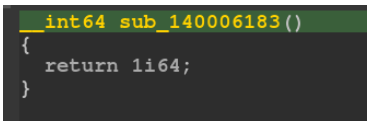
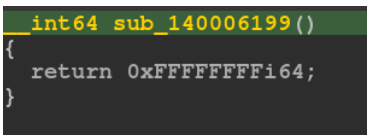
```
                align 10h
stru_140009750  UNWIND_INFO_HDR <9, 0Eh, 4, 25h>
                                        ; DATA XREF: .pdata:000000014000B03C↓o
                UNWIND_CODE <0Eh, 23h>  ; UWOP_SET_FPREG
                UNWIND_CODE <9, 1>      ; UWOP_ALLOC_LARGE
                dw 1Eh
                UNWIND_CODE <2, 50h>    ; UWOP_PUSH_NONVOL
                dd rva sub_140001400
                dd 3
                SCOPE_RECORD <rva loc_1400015D5, rva loc_1400015E2, rva sub_140006170,\
                              rva loc_1400015E2>
                SCOPE_RECORD <rva loc_1400015D5, rva loc_14000161D, rva sub_140006183,\
                              rva loc_14000161D>
                SCOPE_RECORD <rva loc_140001657, rva loc_140001664, rva sub_140006199,\
                              rva loc_140001664>
                db    0
```

其中，exception發生時並不會直接跳到target執行，而是先進入圖中sub_140001400執行，看看他做了什麼:

```
EXCEPTION_DISPOSITION __fastcall sub_140001400(
        struct _EXCEPTION_RECORD *a1,
        void *a2,
        struct _CONTEXT *a3,
        struct _DISPATCHER_CONTEXT *a4)
{
  byte_14000A8C8 = a3->Rax;
  a3->Rip += 63i64;
  return _C_specific_handler(a1, a2, a3, a4);
}
```

這個function將當前rax的值存入14000A8C8中，並增加rip後執行正常的handler。

而下面三個handler，每個都點進去，看看會做什麼事情:

140006170	140006183	140006199
		
回傳0，不做動	回傳1，執行handler	回傳-1，繼續往下執行

Exceptions

接下來，觀察兩個exception發生時分別會被哪些handler處理，
第一個exception位置在1400015DE，因此在前兩個handler範圍內，但第一個handler不做動，因此直接到第二個執行，jump到14000161D執行:

```
loc_14000161D:                          ; DATA XREF: .rdata:00000
                mov     dword ptr [rbp+74h], 0
                jmp     short loc_14000162E
; ---------------------------------------------------------------------------

loc_140001626:                          ; CODE XREF: main_0+C4↓j
                mov     eax, [rbp+74h]
                inc     eax
                mov     [rbp+74h], eax

loc_14000162E:                          ; CODE XREF: main_0+94↑j
                cmp     dword ptr [rbp+74h], 26h ; '&'
                jge     short loc_140001656
                movsxd  rax, dword ptr [rbp+74h]
                movzx   ecx, cs:byte_14000A8C8
                add     ecx, [rbp+74h]
                movzx   ecx, cl
                movzx   eax, byte ptr [rbp+rax+0]
                xor     eax, ecx
                movsxd  rcx, dword ptr [rbp+74h]
                mov     [rbp+rcx+0], al
                jmp     short loc_140001626
```

根據我的理解，這些組合語言可以轉換成下面的pseudo code:

```
for i in 0..26:
    ecx = 0xbe
    ecx += i
    ecx &= 255
    eax = input[i]
    eax ^= ecx
    input[i] = eax & 255
```

而第二個exception觸發後會從excpetion發生的位置繼續執行下去，也就是從140001662開始:

```
                mov     [rbp+0D0h+var_54], 0
                jmp     short loc_1400016B0
; ---------------------------------------------------------------------------

loc_1400016A8:                          ; CODE XREF: main_0+1
                mov     eax, [rbp+0D0h+var_54]
                inc     eax
                mov     [rbp+0D0h+var_54], eax

loc_1400016B0:                          ; CODE XREF: main_0+1
                cmp     [rbp+0D0h+var_54], 26h ; '&'
                jge     short loc_1400016D8
                movsxd  rax, [rbp+0D0h+var_54]
                movzx   ecx, cs:byte_14000A8C8
                add     ecx, [rbp+0D0h+var_54]
                movzx   ecx, cl
                movzx   eax, [rbp+rax+0D0h+flag]
                add     eax, ecx
                movsxd  rcx, [rbp+0D0h+var_54]
                mov     [rbp+rcx+0D0h+flag], al
                jmp     short loc_1400016A8
; ---------------------------------------------------------------------------
```

跟上面的差不多，可以轉換成下面的pseudo code:

```
for i in 0..26:
    ecx = 0xef
    ecx += i
    ecx &= 255
    eax = input[i]
    eax += ecx
    input[i] = eax & 255
```

Solve

有了上面的資訊，就可以依照流程，寫出下面的C程式碼:

```
#include <stdio.h>

const char ECX1 = 0xBE;
const char ECX2 = 0xEF;

int main() {
    char data[] = {0xe7, 0xe3, 0x72, 0x78, 0xac, 0x90, 0x90, 0x7c, 0x90,
```

```
                    0xac, 0xb1, 0xa6, 0xa4, 0x9e, 0xa7, 0xa2, 0xac, 0x90,
                    0xb9, 0xb2, 0xbf, 0xbb, 0xbd, 0xb6, 0xab, 0x90, 0xba,
                    0xb4, 0x90, 0xbf, 0xc0, 0xc0, 0xc4, 0xca, 0x95, 0xed,
                    0xc0, 0xb2, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                    0x00, 0x00, 0x00, 0xff, 0xff, 0xff, 0xff};
    for (int i = 0; i < 0x26; i++) {
        data[i] -= (ECX2 + i);
    }
    for (int i = 0; i < 0x26; i++) {
        data[i] ^= (ECX1 + i);
    }
    printf("%s\n", data);
}
```

Result

```
$ gcc sol.c -o sol && ./sol
FLAG{__C_specific_handler_is_hooked:0}
```