Phòng thí nghiệm An toàn Thông tin InsecLab
Trường Đại học Công nghệ Thông tin (UIT)

BÁO CÁO THỰC HÀNH

Môn học: Cơ chế hoạt động của mã độc

Tên chủ đề: Ôn tập ngôn ngữ assembly và chèn mã vào tập tin PE

GVHD: Ngô Đức Hoàng Sơn

Nhóm: 12

1. THÔNG TIN CHUNG:

(Liệt kê tất cả các thành viên trong nhóm)

Lớp: NT230.O21.ANTT.1

STT	Họ và tên	MSSV	Email
1	Nguyễn Triệu Thiên Bảo	21520155	21520155@gm.uit.edu.vn
2	Trần Lê Minh Ngọc	21521195	21521195@gm.uit.edu.vn
3	Huỳnh Minh Khuê	21522240	21522240@gm.uit.edu.vn

2. NỘI DUNG THỰC HIỆN:[1]

STT	Nội dung	Tình trạng	Trang
1	Yêu cầu 1	100%	2 - 3
2	Yêu cầu 2	100%	3 - 5
3	Yêu cầu 3	100%	...
4	Yêu cầu 4	100%	
5	Yêu cầu 5	100%	
Điểm tự đánh giá			**?/10**

Phần bên dưới của báo cáo này là tài liệu báo cáo chi tiết của nhóm thực hiện.

[1] Ghi nội dung công việc, các kịch bản trong bài Thực hành

BÁO CÁO CHI TIẾT

Bài tập 1: Viết một đoạn chương trình tìm số nhỏ nhất trong 3 số (1 chữ số) a, b, c cho trước.

Link bài làm: http://tpcg.io/_XHQ8O8

Bước 1: Khai báo giá trị cho các vùng nhớ num1, num2, num3 lần lượt là các giá trị "1", "2", "3" và mỗi vùng nhớ được cấp phát kích thước 1 byte.

```
7  segment .data
8   num3 db '3'
9   num2 db '2'
10  num1 db '1'
```

Bước 2: So sánh num1 và num2.

- Lấy giá trị của num1 gán cho thanh ghi eax, sau đó so sánh thanh ghi eax với giá trị của num2. Nếu giá trị của num1 lớn hơn thì nhảy đến hàm check_third_number để thực thi, ngược lại, gán giá trị của num2 cho thanh ghi eax.

```
16 _start: ;tell linker entry point
17  mov eax, [num1] ; compare 1st num and 2nd num
18  cmp eax, [num2]
19  jl check_third_number
20  mov eax, [num2]
```

Bước 3: So sánh num1 hoặc num2 với num3.

- So sánh giá trị đã được lưu trong thanh ghi eax trước đó với giá trị của num3. Nếu giá trị trong thanh ghi eax lớn hơn giá trị num3 thì nhảy đến hàm exit, ngược lại thì gán giá trị num3 cho thanh ghi eax.

```
22 check_third_number:
23  cmp eax, [num3]
24  jl exit
25  mov eax, [num3]
26
```

Bước 4: In kết quả ra màn hình

- Gán giá trị trong thanh ghi eax cho vùng nhớ mà con trỏ res trỏ tới.

- Tiến hành ghi dữ liệu vào luồng output, thiết lập các giá trị cần thiết cho việc gọi hệ thống "write". Đặt mã syscall "SYS_WRITE" (SYS_WRITE đã được định nghĩa) cho hệ thống "write" vào thanh ghi eax. Đặt ebx là mã của stdout, ecx là con trỏ trỏ tới vùng nhớ muốn ghi vào stdout (là vùng nhớ res có chứa giá trị cần in ra màn hình), edx là độ dài dữ liệu muốn in ra màn hình (trường hợp này là 1 byte). Sau đó dùng lời gọi hệ thống "int 0x80" để thực hiện hành động ghi vào stdout (sau khi thực hiện bước này sẽ in ra màn hình kết quả mong muốn).

-Cuối cùng thoát khỏi chương trình bằng cách gọi hệ thống "exit". Đặt mã syscall (SYS_EXIT) cho hệ thống "exit" vào thanh ghi eax. Đặt ebx về 0 (bằng phép xor 2 thanh ghi giống nhau) và dùng lời gọi hệ thống "int 0x80" để kết thúc chương trình.

```
27 exit:
28  mov [res], eax
29
30  mov eax, SYS_WRITE
31  mov ebx, STDOUT
32  mov ecx, res
33  mov edx, 1
34  int 0x80
35
36  mov eax, SYS_EXIT
37  xor ebx, ebx
38  int 0x80
```

Kết quả

```
1
```

Bài tập 2: Viết chương trình chuyển đổi một số (number) 123 thành chuỗi '123' Sau đó thực hiện in ra màn hình số 123.

Link bài làm: http://tpcg.io/_ED2GEM

Cấp phát cho vùng nhớ x kích thước 1 byte với giá trị 123, vùng nhớ msgX có kích thước tương tự với chuỗi "x = ".

```
7  section .data
8  x db 123
9  msgX db "x = "
```

Hiển thị chuỗi msgX:

```
18  mov ecx, msgX
19  mov edx, 4
20  call _printString
```

Gán địa chỉ chuỗi msgX vào thanh ghi ecx (địa chỉ này được sử dụng để in chuỗi ra màn hình), gán độ dài chuỗi msgX vào thanh ghi edx (4 ký tự). Tiếp theo gọi đến hàm _printString để in chuỗi msgX.

Hàm _printString:

```
29  _printString:
30   push eax
31   push ebx
32
33   mov eax,SYS_WRITE
34   mov ebx,STDOUT
35   int 0x80
36   pop ebx
37   pop eax
38   ret
```

Đẩy giá trị 2 thanh ghi eax, ebx vào stack để lưu trữ. Gán mã syscall (SYS_WRITE) cho hệ thống "write" vào thanh ghi eax, gán mã STDOUT vào thanh ghi ebx. Sau đó dùng lời gọi hệ thống "int 0x80" để ghi ra stdout (sau bước này, chuỗi msgX được in ra màn hình). Cuối cùng khôi phục giá trị 2 thanh ghi eax, ebx và trở lại nơi gọi hàm.

Hiển thị giá trị biến x dưới dạng số thập phân:

```
21  mov eax, 0
22  mov al, byte[x]
23  call _printDec
```

Gán 0 vào thanh ghi eax để chuẩn bị in giá trị biến x, sau đó gán giá trị biến x vào byte thấp nhất của thanh ghi eax (al chứa phần bit thấp của eax) rồi gọi hàm _printDec để in giá trị biến x dưới dạng số thập phân.

Hàm _printDec:

```
62  _printDec:
63  ;;; saves all the registers so that they are not changed by
        the function
64  section .bss
65  .decstr resb 10
66  .ct1 resd 1 ; to keep track of the size of the string
67  section .text
68   pushad ; save all registers
69   mov dword[.ct1],0 ; assume initially 0
70   mov edi,.decstr ; edi points to decstring
71   add edi,9 ; moved to the last element of string
72   xor edx,edx ; clear edx for 64-bit division
73  .whileNotZero:
74   mov ebx,10 ; get ready to divide by 10
75   div ebx ; divide by 10
76   add edx,'0' ; converts to ascii char
77   mov byte[edi],dl ; put it in sring
78   dec edi ; mov to next char in string
79   inc dword[.ct1] ; increment char counter
80   xor edx,edx ; clear edx
81   cmp eax,0 ; is remainder of division 0?
82   jne .whileNotZero ; no, keep on looping
83   inc edi ; conversion, finish, bring edi
84   mov ecx, edi ; back to beg of string. make ecx
85   mov edx, [.ct1] ; point to it, and edx gets # chars
86   mov eax, SYS_WRITE ; and print!
87   mov ebx, STDOUT
88   int 0x80
89   popad ;
90   ret
```

- Đầu tiên lưu trữ giá trị của tất cả các thanh ghi vào trong stack. Tiếp theo khởi tạo biến và con trỏ, gán giá trị 0 cho biến ".ct1" (số lượng ký tự trong chuỗi=0), đặt con trỏ edi trỏ đến địa chỉ chuỗi decstr (là nơi lưu kết quả việc chuyển đổi), di chuyển con trỏ edi tới phần tử cuối cùng của chuỗi.
- Tiếp theo đổi giá trị thanh ghi edx về 0 (bằng phép xor) để làm nơi lưu số dư sau phép chia 10. Gán giá trị 10 cho ebx, thực hiện phép chia cho ebx, kết quả được lưu trong thanh ghi eax, phần dư được lưu trong edx. Lấy phần dư chuyển đổi thành ký tự ASCII tương ứng, đặt ký tự này vào vị trí thích hợp trong chuỗi. Tiếp tục di chuyển tới vị trí tiếp theo tvà tăng số lượng ký tự trong chuỗi lên 1, so sánh xem phần dư có bằng 0 không, nếu

bằng 0, đẩy con trỏ edi tới vị trí tiếp theo trong chuỗi, ngược lại thì tiếp tục vòng lặp ".whileNotZero".
- Sau khi chia xong, đặt con trỏ ecx về vị trí đầu tiên trong chuỗi, gán số lượng ký tự cho thanh ghi edx.
- Đặt các giá trị cần thiết cho hệ thống "write" và dùng lời gọi hệ thống "int 0x80" để in chuỗi đã chuyển đổi ra stdout.
- Cuối cùng khôi phục thanh ghi và thoát khỏi hàm, trở về nơi gọi hàm.

Kết quả

Bài tập 3:
Link bài làm: http://tpcg.io/_2SD1CB

Chúng ta kết hợp thuật toán tìm kiếm số nhỏ nhất trong 3 số của bài 1 và cách hiển thị một số thành dạng chuỗi ra màn hình của bài 2 để thực hiện yêu cầu này.
Khai báo các number, hàm compare_third_num và đưa chúng vào các thanh ghi để tiến hành so sánh như bài 1.

```
_start:
;;; find smallest
    number
mov ecx, [num1]
mov ebx, [num2]
cmp ecx, ebx
jl .compare_third_num
mov ecx, ebx

.compare_third_num:
mov ebx, [num3]
cmp ecx, ebx
jl .exit
mov ecx, ebx
```

Sau khi so sánh xong, hiện tại số nhỏ nhất được lưu trữ trong thanh ghi ecx. Chuyển giá trị của thanh ghi ecx sang biến smallest và hiển thị dưới dạng một từ (word) có kích thước 32 bit (4 byte).

```
.exit:
;;; assign smallest
    number to x
mov dword[smallest],
    ecx
```

Sau đó hiển thị số nhỏ nhất ra màn hình như bài 2.

```
;;; display x
mov ecx, msgX
mov edx, len
call _printString
mov eax, 0
mov eax,
    dword[smallest]
call _printDec
```

.....

Thử so sánh 3 số (nhiều hơn 1 chữ số) gồm 725, 2235 và 131

```
section .data
smallest dd 0
msgX db "The smallest
    digit is "
len equ $- msgX
num1 dd 725
num2 dd 2235
num3 dd 131
```

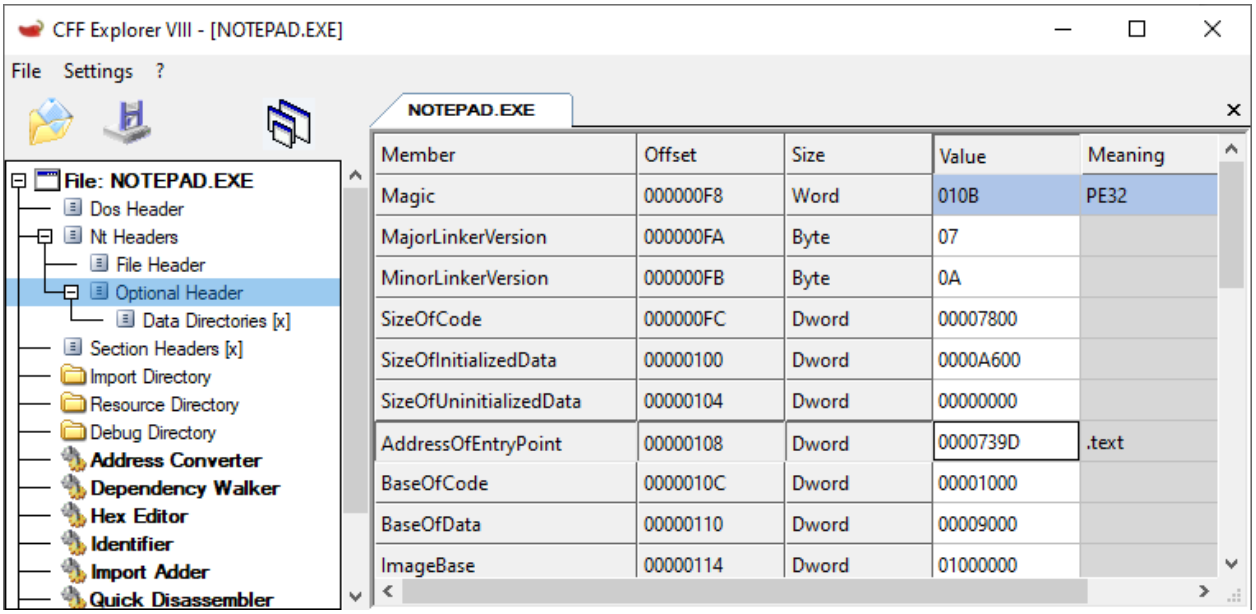
Kết quả hiển thị đúng chính xác số nhỏ nhất là 131

```
The smallest digit is 131
```

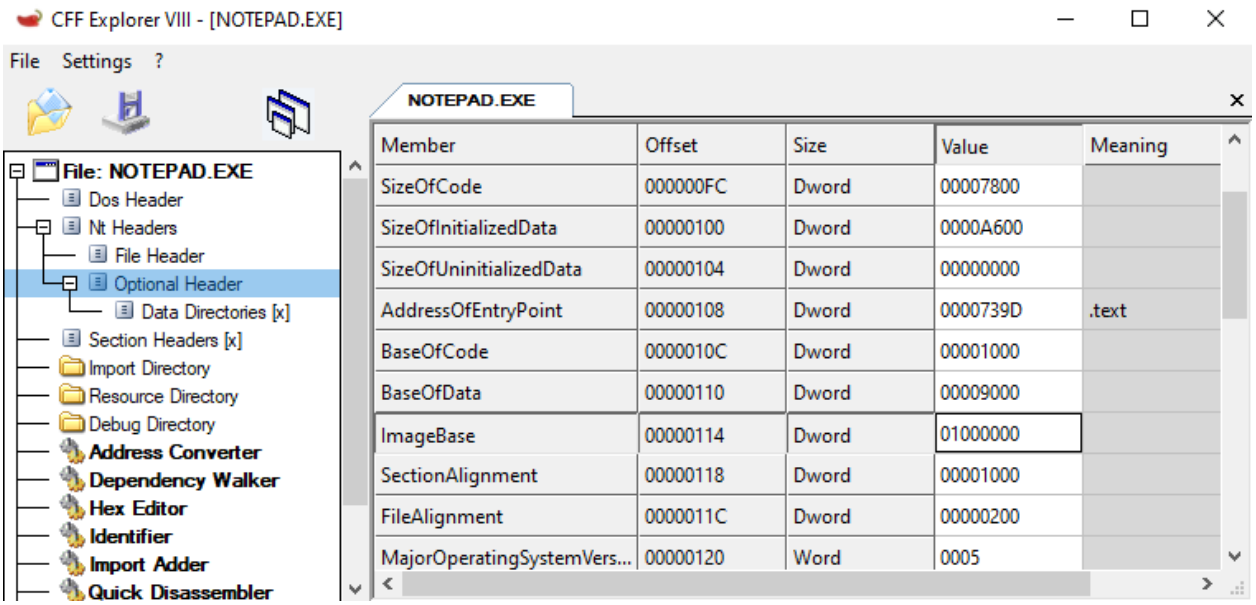
Bài tập 4:

Mở CFF Explorer để đọc nội dung của Notepad.exe. Nhìn vào các nội dung được hiển thị.

Chọn Optional Header, tìm AddressOfEntryPoint trong bảng bên phải chứa giá trị 0x0000739D.

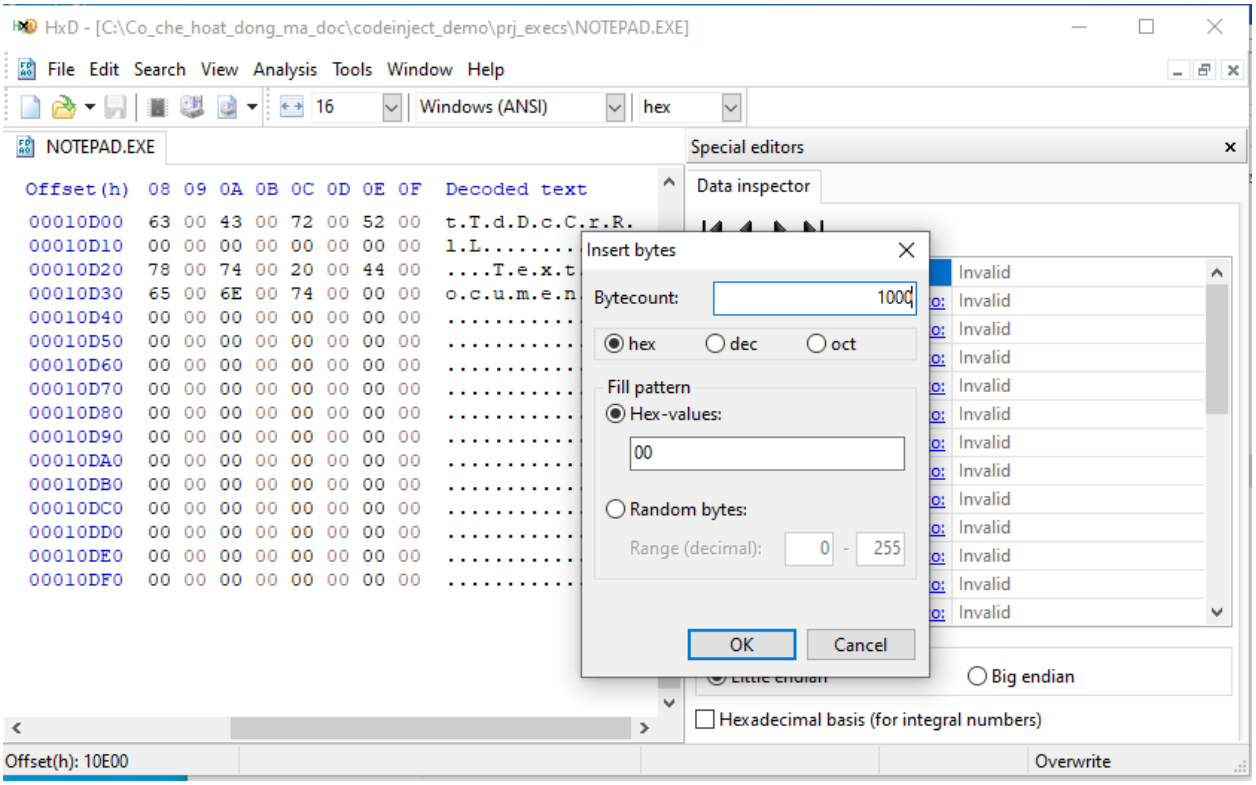


Cũng trong bảng bên phải, tìm ImageBase chứa giá trị 0x01000000.



1. Tạo vùng nhớ trong tập tin PE

Sử dụng HxD để mở Notepad.exe. Đặt trỏ chuột vào cuối file, chọn Edit -> Insert Bytes, nhập giá trị 0x1000 và nhấn OK.



Ta có một chương trình hiển thị MessageBox như sau:

```
#include <windows.h>
int main(int argc, char * argv[])
{
 MessageBox(NULL, L"Code injected", L"Info", MB_OK);
 return 0;
}
```

Biên dịch chương trình dưới chế độ Release, Not Using Precompiled Headers. Sử dụng IDA Pro để mở file PE và xem mã hợp ngữ của chương trình vừa biên dịch.

```
.text:00401000 sub_401000      proc near               ; CODE XREF: start-8D↓p
.text:00401000                 push    0               ; uType
.text:00401002                 push    offset Caption  ; "Info"
.text:00401007                 push    offset Text     ; "21520155 - 21521195 - 21522240"
.text:0040100C                 push    0               ; hWnd
.text:0040100E                 call    ds:MessageBoxW
.text:00401014                 xor     eax, eax
.text:00401016                 retn
```

Và mã hex của Caption và Text

```
00402100  49 00 6E 00 66 00 6F 00  00 00 00 00 32 00 31 00  I.n.f.o.....2.1.
00402110  35 00 32 00 30 00 31 00  35 00 35 00 20 00 2D 00  5.2.0.1.5.5.·.-.
00402120  20 00 32 00 31 00 35 00  32 00 31 00 31 00 39 00  ·.2.1.5.2.1.1.9.
00402130  35 00 20 00 2D 00 20 00  32 00 31 00 35 00 32 00  5.·.-.·.2.1.5.2.
00402140  32 00 32 00 34 00 30 00  00 00 00 00 00 00 00 00  2.2.4.0.........
```

2. Về cơ bản, chương trình gồm 5 dòng lệnh (sử dụng chức năng hexview để xem mã hex của từng lệnh).

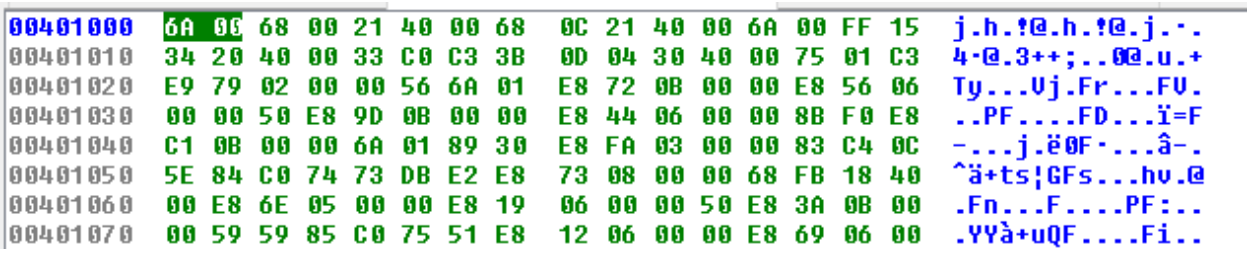
push 0 ; 6a 00

push Caption ; 68 X

push Text ; 68 Y

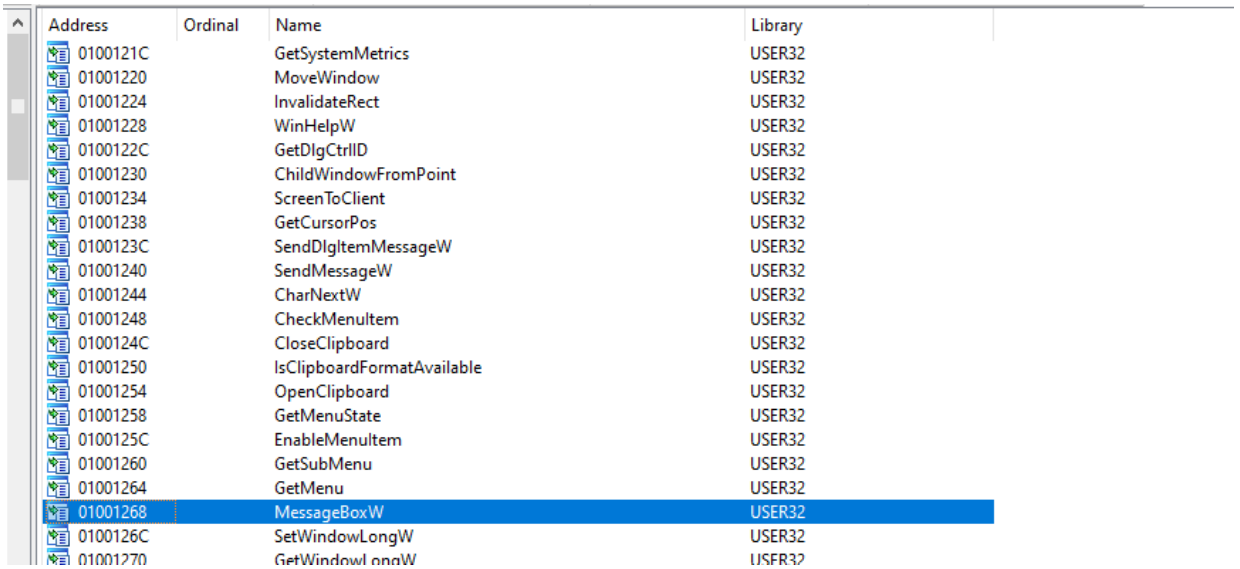
push 0 ; 6a 00

call [MessageBoxW] ; ff15 Z



Để chèn đoạn code này vào Notepad.exe, ta phải đi tìm các giá trị (X, Y, Z) phù hợp.

3. Giá trị Z chính là địa chỉ của hàm MessageBoxW được import từ thư viện USER32.dll. Trong IDA Pro, mở Notepad.exe, chọn View -> Open Subviews -> Imports và ta thấy địa chỉ của hàm MessageBoxW chính là 01001268



Ta chọn các địa chỉ sau cho việc lưu trữ:

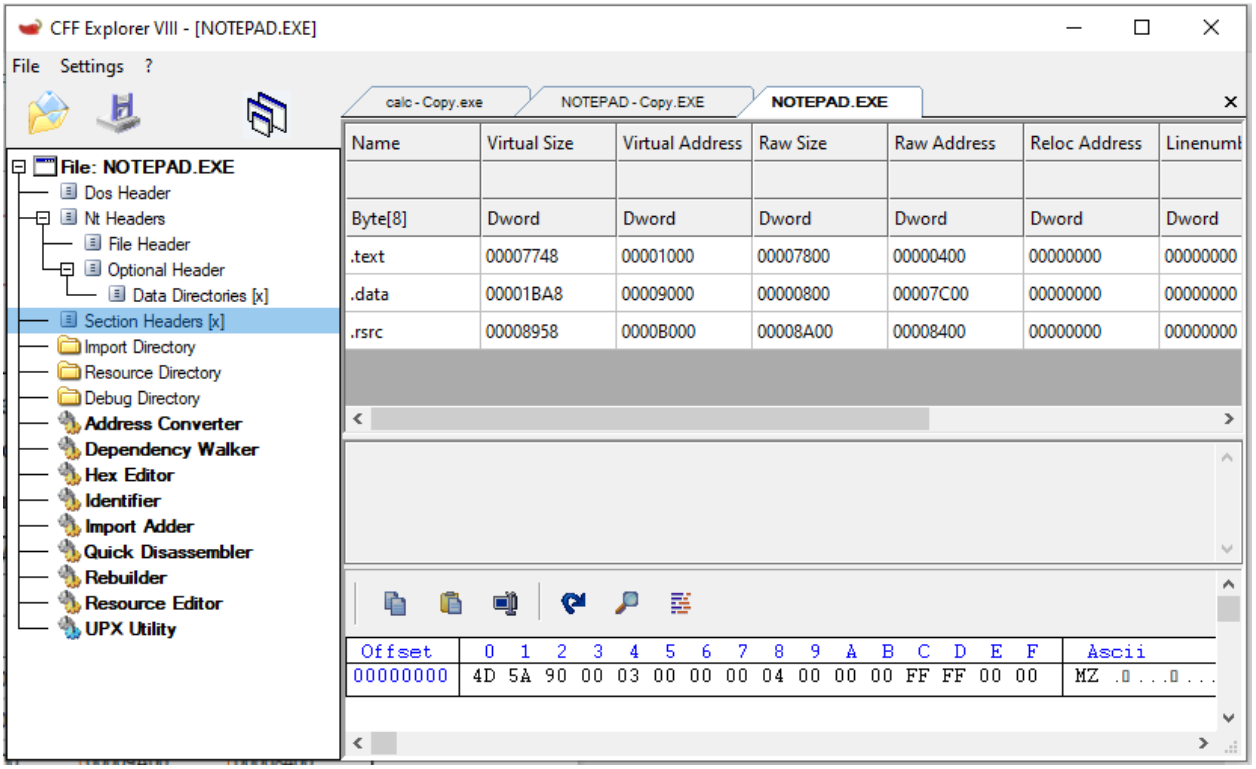
- 0x00011000 cho lưu trữ hợp ngữ
- 0x00011040 cho lưu trữ Caption
- 0x00011080 cho lưu trữText

Ta sẽ sử dụng công thức sau để tính X, Y:

Offset = RA – Section RA = VA – Section VA

Trong đó X là VA của Caption và Y là VA của Text, RA là địa chỉ vật lý của các phần cần lưu trữ

Và các số Section RA, Section VA là các giá trị trong Section headers của .rsrc



Cụ thể trong trường hợp này:

- Section RA = 00008400

- Section VA = 0000B000

Vậy từ đây ta có các phép tính như sau:

X = VA của Caption = RA của Caption – Section RA + Section VA

= 0x00011040 – 0x00008400 + 0x0000B000

= 0x00013C40

Y = VA của Text = RA của Text – Section RA + Section VA

= 0x00011080 – 0x00008400 + 0x0000B000

= 0x00013C80

Cộng thêm ImageBase, suy ra X = 0x01013C40. Tương tự , Y = 0x01013C80.

Tiếp theo ta tính địa chỉ thực thi mới của đoạn code, sử dụng công thức như trên

New_entry_point = 0x00011000 0x00008400 + 0x0000B000 = 0x00013C00

Ta cần chương trình tiếp tục thực thi sau khi chạy đoạn code trên ta càn chèn dòng lệnh quay về AddressOfEntryPoint cũ ngay sau đoạn code ở bước 2. jmp relative_VA

Relative_VA có thể tính theo công thức sau:

old_entry_point = jmp_instruction_VA + 5 + relative_VA (2)

Néu đặt lệnh jmp sau 5 câu lệnh ở bước tìm hợp ngữ thì jmp_instruction_VA = 0x01013C00 + 0x14 = 0x01013C14. (Vì 0x14 là số byte của 5 câu lệnh)

Vậy relative_VA = old_entry_point - jmp_instruction_VA – 0x5

= 0x0000739D - 0x01013C14 – 0x5

= 0xFFFF3784

Đến đây, ta đã có một đoạn mã hợp ngữ hoàn chỉnh để chèn vào Notepad.exe. Các địa chỉ được biểu diễn theo thứ tự little endian (x86).

push 0 ; 6a 00

push Caption ; 68 403C0101

push Text ; 68 803C0101

push 0 ; 6a00

call [MessageBoxW] ; ff15 68120001

jmp Origianl_Entry_Point ; e9 8437FFFF

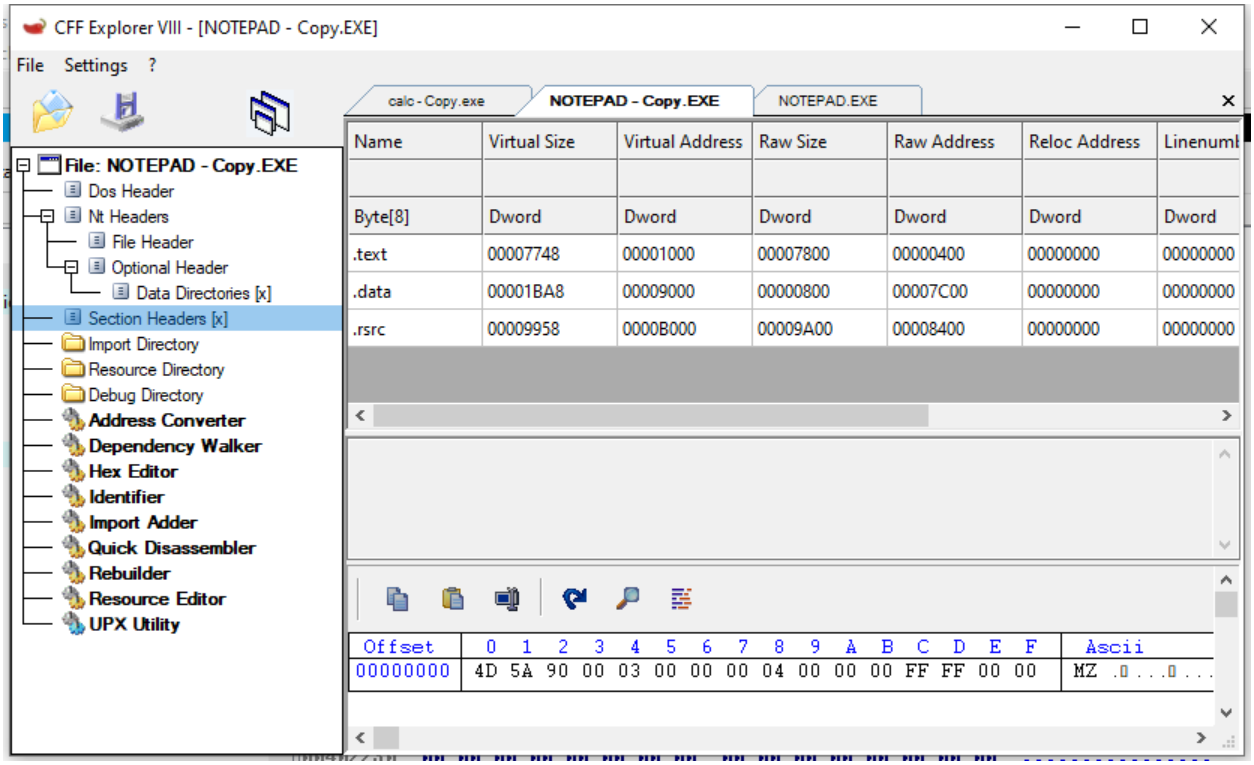
Sử dụng HxD để chèn đoạn mã cùng với giá trị Caption và Text vào Notepad.exe. Lưu lại file.

```
00011000  6A 00 68 80 3C 01 01 68 40 3C 01 01 6A 00 FF 15  j.h€<..h@<..j.ÿ.
00011010  68 12 00 01 E9 84 37 FF FF 00 00 00 00 00 00 00  h...é„7ÿÿ.......
00011020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00011030  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00011040  32 00 31 00 35 00 32 00 30 00 31 00 35 00 35 00  2.1.5.2.0.1.5.5.
00011050  20 00 2D 00 20 00 32 00 31 00 35 00 32 00 31 00   .-. .2.1.5.2.1.
00011060  31 00 39 00 35 00 20 00 2D 00 20 00 32 00 31 00  1.9.5. .-. .2.1.
00011070  35 00 32 00 32 00 32 00 34 00 30 00 00 00 00 00  5.2.2.2.4.0.....
00011080  49 00 6E 00 66 00 6F 00 00 00 00 00 00 00 00 00  I.n.f.o.........
00011090  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

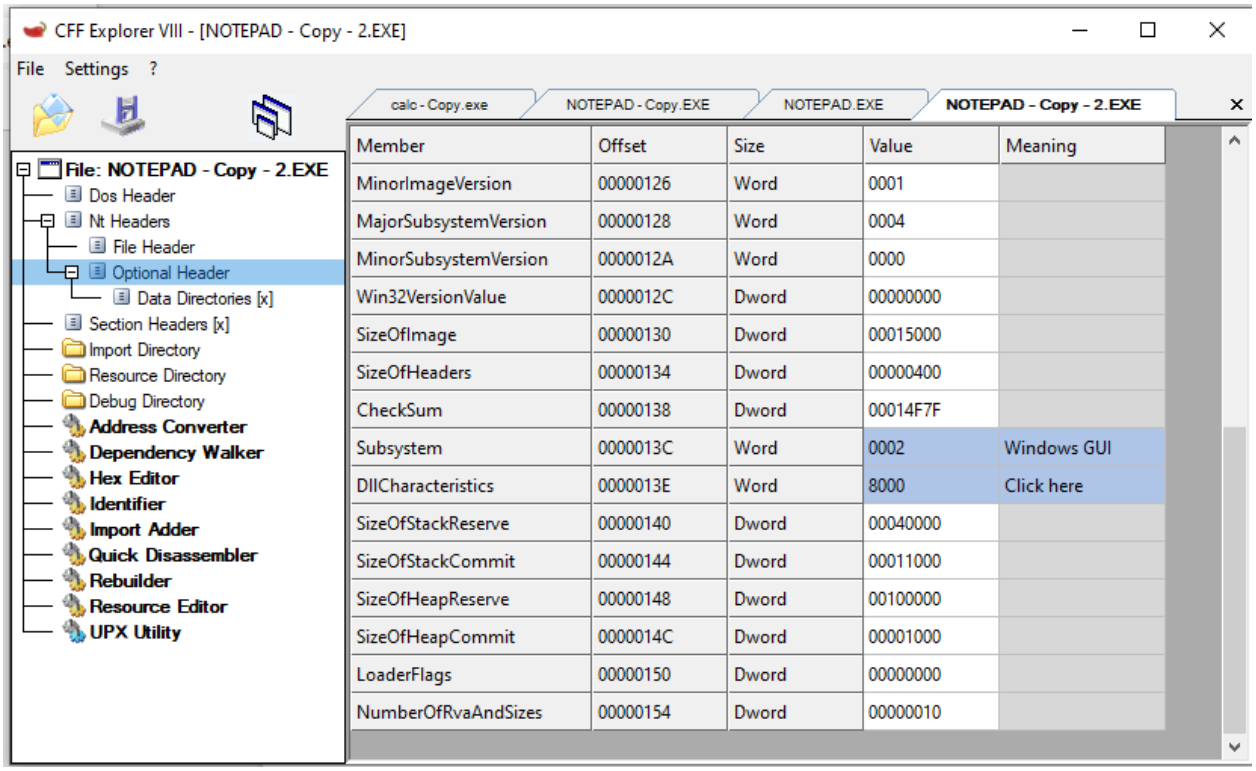
Cuối cùng là hiệu chỉnh các tham số trong PE header

Sử dụng CFF Explorer để thay đổi các giá trị sau:

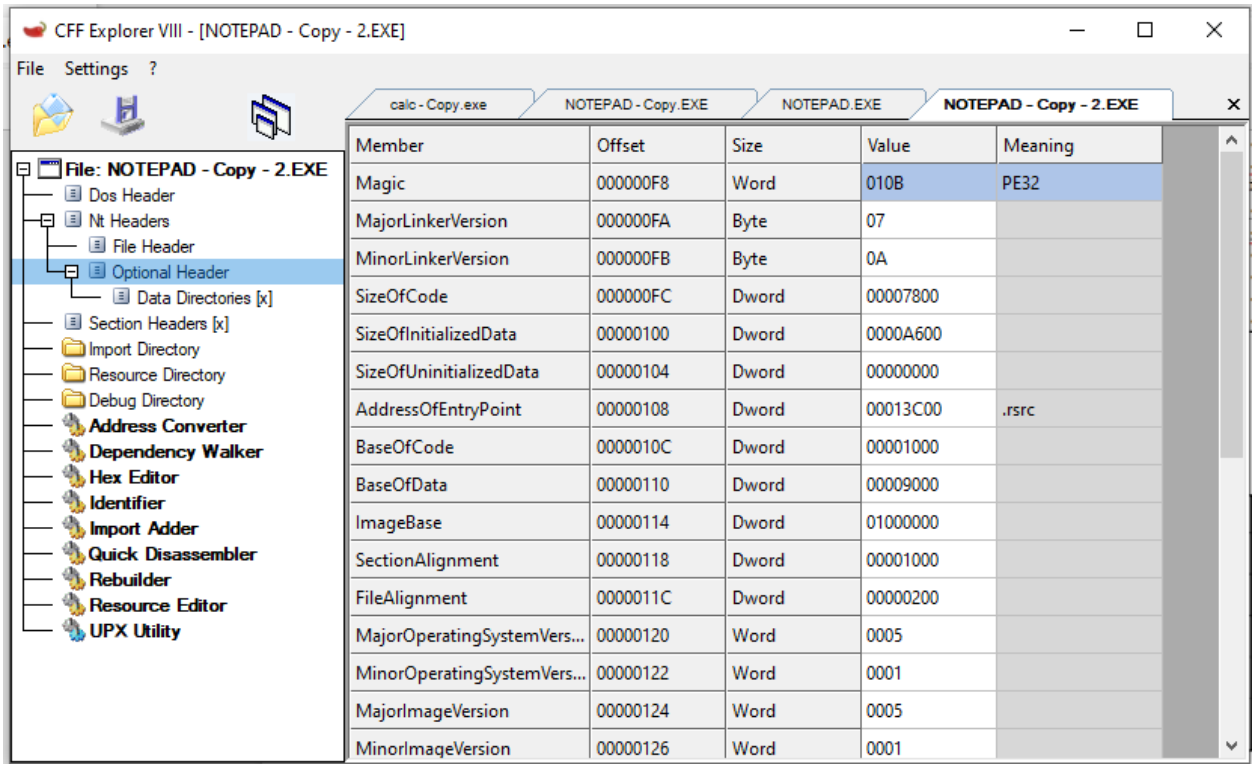
• Trong Section Headers, thay đổi .rsrc Section Header.



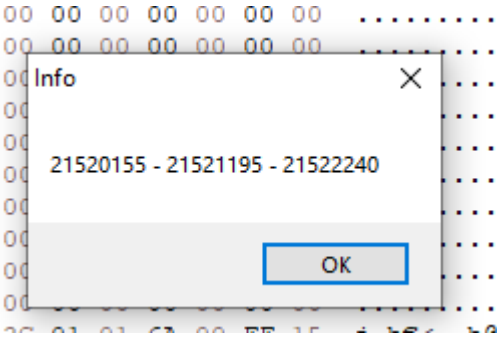
- Trong Optional Headers, tăng SizeOfImage le n 0x1000.



- Trong Optional Headers, chỉnh sửa AddressOfEntryPoint thành 0x00013C00.



Thực thi file Notepad.exe, một cửa sổ xuất hiện như sau:



Bài tập 5:

Injected code cho Notepad:

Do thao tác trên Notepad.exe nên ta có thể sử dụng lại các thông tin đã tìm hiểu ở bài 4:

- AddressOfEntryPoint: 0x0000739D.

- ImageBase: 0x01000000

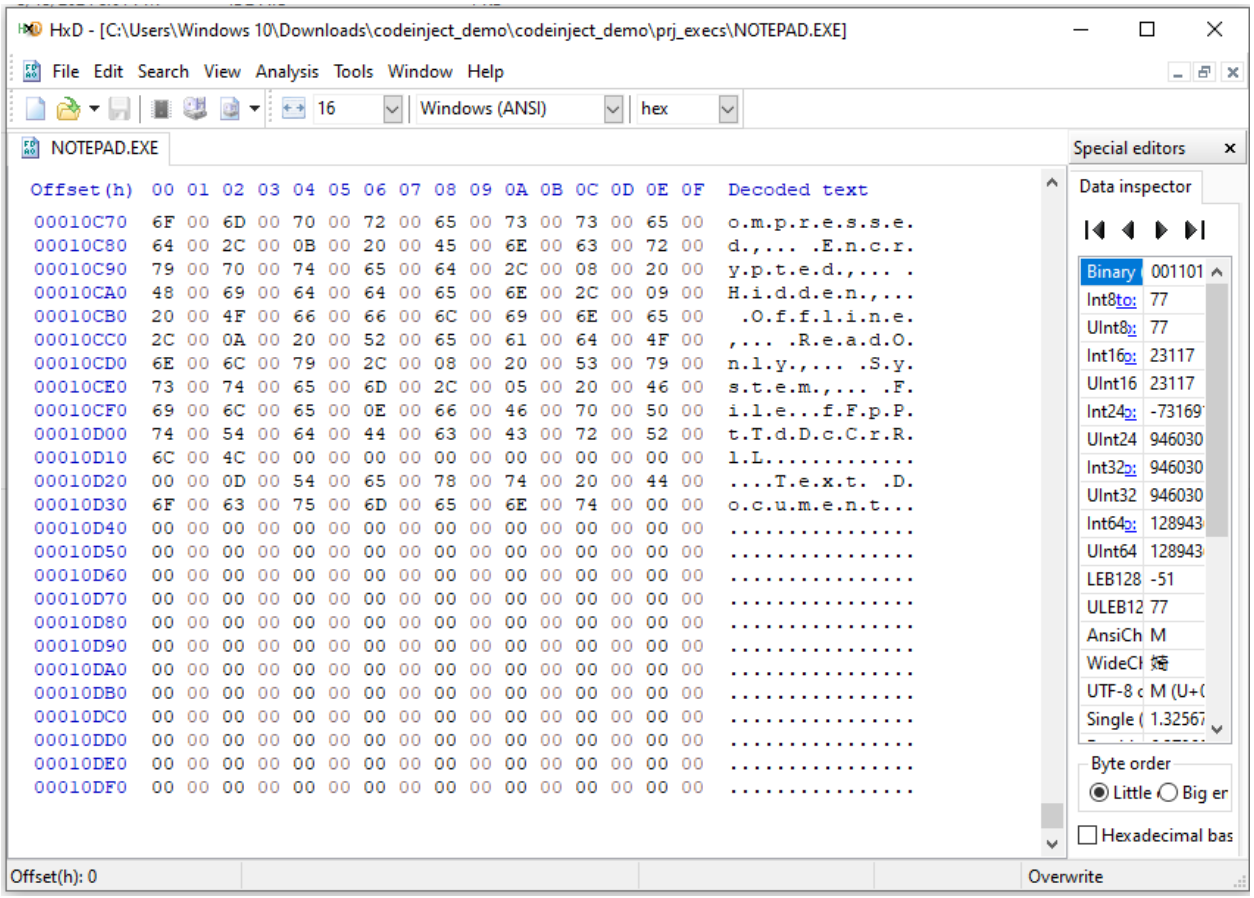
- Đoạn hợp ngữ và mã hex tương ứng:

```
.text:00401000 sub_401000      proc near               ; CODE XREF: start-8D↓p
.text:00401000                 push    0               ; uType
.text:00401002                 push    offset Caption  ; "Info"
.text:00401007                 push    offset Text     ; "Code injected"
.text:0040100C                 push    0               ; hWnd
.text:0040100E                 call    ds:MessageBoxW
.text:00401014                 xor     eax, eax
.text:00401016                 retn
.text:00401016 sub_401000      endp
```

```
00401000  6A 00 68 00 21 40 00 68  0C 21 40 00 6A 00 FF 15  j.h.!@.h.!@.j.ÿ.
00401010  34 20 40 00 33 C0 C3 3B  0D 04 30 40 00 75 01 C3  4 @.3ÀÃ;..0@.u.Ã
00401020  E9 79 02 00 00 56 6A 01  E8 72 0B 00 00 E8 56 06  éy...Vj.èr...èV.
00401030  00 00 50 E8 9D 0B 00 00  E8 44 06 00 00 8B F0 E8  ..Pè....èD...ïðè
00401040  C1 0B 00 00 6A 01 89 30  E8 FA 03 00 00 83 C4 0C  Á...j.ë0èú...âÄ.
00401050  5E 84 C0 74 73 DB E2 E8  73 08 00 00 68 FB 18 40  ^ä+ts¦GFs...hv.@
00401060  00 E8 6E 05 00 00 E8 19  06 00 00 50 E8 3A 0B 00  .Fn...F....PF:..
00401070  00 59 59 85 C0 75 51 E8  12 06 00 00 E8 69 06 00  .YYà+uQF....Fi..
00401080  00 85 C0 74 0B 68 81 16  40 00 E8 16 0B 00 00 59  .à+t.h..@.F....Y
```

- Địa chỉ của MessageBoxW: 0x01001268
- Section RA = 0x00008400
- Section VA = 0x0000B000

Nhận xét thấy ở cuối file có 1 khoảng trống, ta sẽ lợi dụng khoảng trống này để chèn mã vào mà không làm tăng kích thước file



Ta chọn các địa chỉ sau cho việc lưu trữ:

- 0x00010D50 cho lưu trữ hợp ngữ
- 0x00010D90 cho lưu trữ Caption
- 0x00010DB0 cho lưu trữ Text

Tương tự với bài trước ta cần tìm X, Y, Z trong đoạn hợp mã để chèn và thực thi chức năng thành công vào file Notepad.exe

push 0 ; 6a 00

push Caption ; 68 X

push Text ; 68 Y

push 0 ; 6a 00

call [MessageBoxW] ; ff15 Z

Sử dụng công thức Offset = RA – Section RA = VA – Section VA, ta tính được X, Y, Z như sau:

X = VA của Caption = RA của Caption – Section RA + Section VA

= 0x00010D90 – 0x00008400 + 0x0000B000

= 0x00013990

Y = VA của Text = RA của Text – Section RA + Section VA

= 0x00010DB0 – 0x00008400 + 0x0000B000

= 0x000139B0

Cộng thêm ImageBase, suy ra X = 0x01013990. Tương tự , Y = 0x010139B0.

Z = New_entry_point = 0x00010D50 - 0x00008400 + 0x0000B000 = 0x00013950

Thêm ImageBase thì new_entry_point = 0x01013950

Ta cần chương trình tiếp tục thực thi sau khi chạy đoạn code trên ta càn chèn dòng lệnh quay về AddressOfEntryPoint cũ ngay sau đoạn code mã độc: jmp relative_VA

Relative_VA có thể tính theo công thức sau:

old_entry_point = jmp_instruction_VA + 5 + relative_VA (2)

Néu đặt lệnh jmp sau 5 câu lệnh ở bước tìm hợp ngữ thì jmp_instruction_VA = 0x01013950 + 0x14 = 0x01013964. (Vì 0x14 là số byte của 5 câu lệnh)

Vậy relative_VA = old_entry_point - jmp_instruction_VA – 0x5

= 0x0100739D - 0x01013964 – 0x5

= 0xFFFF3A4

Đến đây, ta đã có một đoạn mã hợp ngữ hoàn chỉnh để chèn vào Notepad.exe. Các địa chỉ được biểu diễn theo thứ tự little endian (x86).

push 0 ; 6a 00

push Caption ; 68 90390101

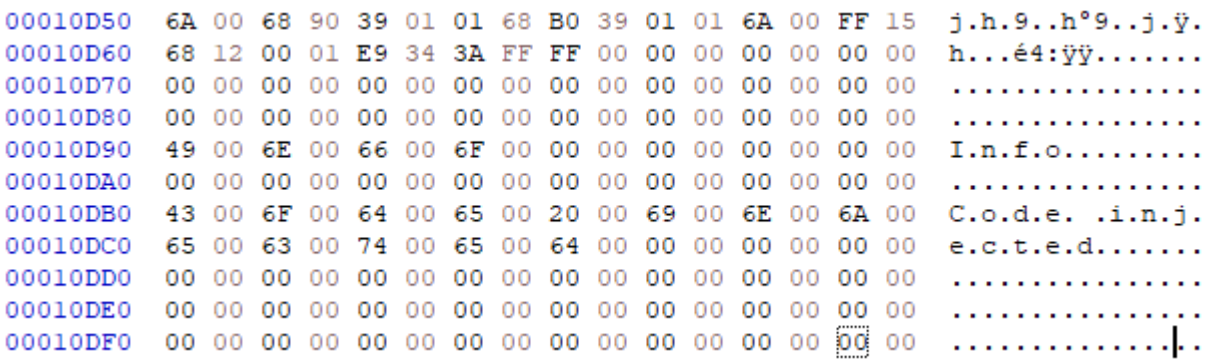
push Text ; 68 B0390101

push 0 ; 6a00

call [MessageBoxW] ; ff15 68120001

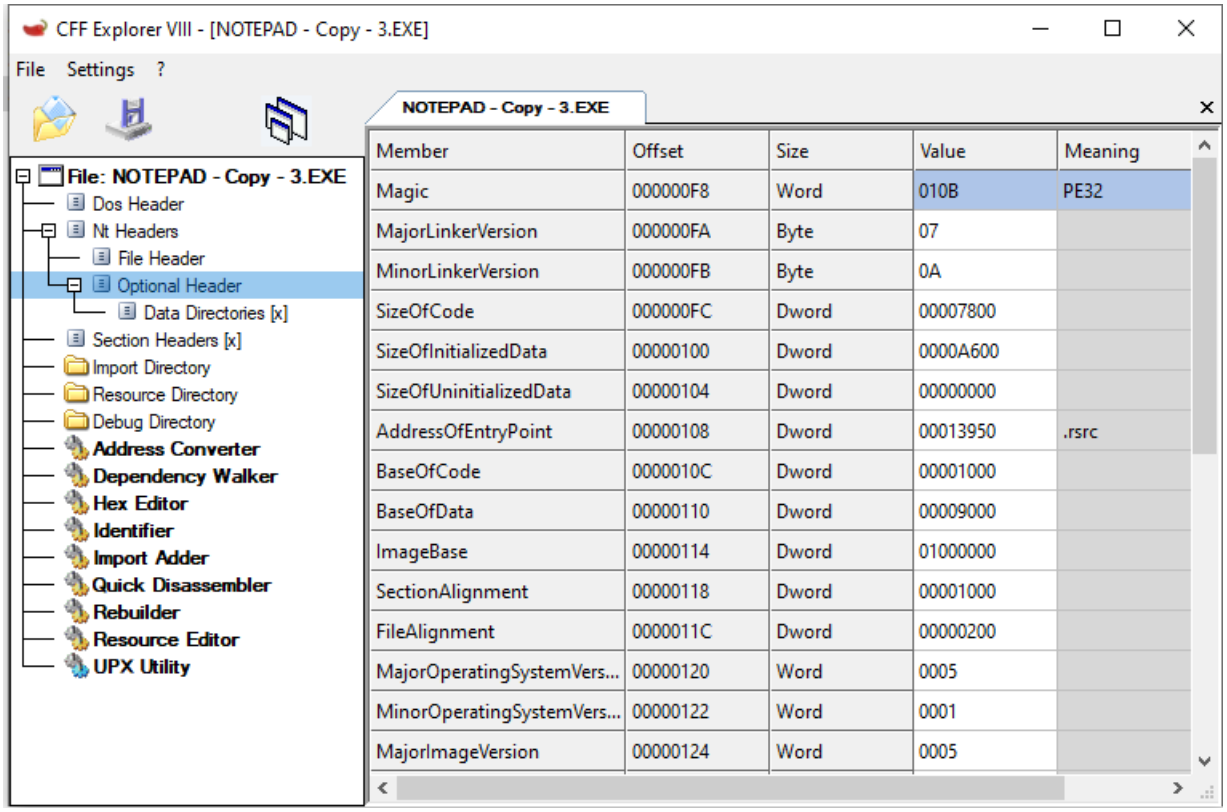
jmp Origianl_Entry_Point ; e9 A4F3FFFF

Sử dụng HxD để chèn đoạn mã cùng với giá trị Caption và Text vào Notepad.exe. Lưu lại file.



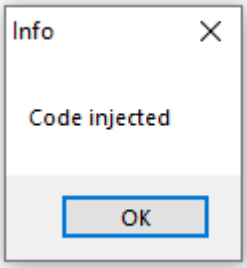
Cuối cùng là hiệu chỉnh các tham số trong PE header

Sử dụng CFF Explorer để thay đổi các giá trị của AddressOfEntryPoint thành địa chỉ mới là 0x00013950 (Địa chỉ new_entry_point)



Lưu lại thay đổi

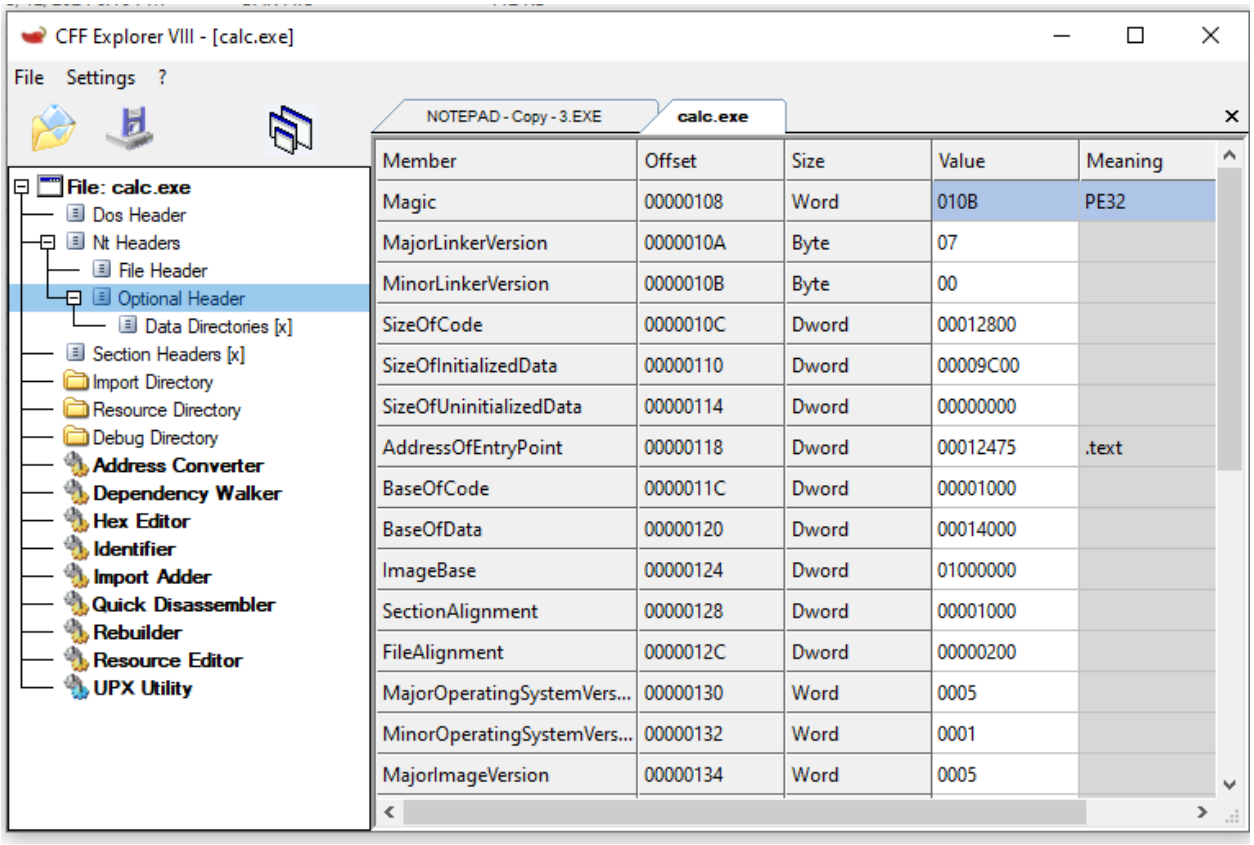
Thực thi file Notepad.exe, một cửa sổ xuất hiện như sau:



Injected code cho Notepad:

Mở CFF Explorer để đọc nội dung của calc.exe. Nhìn vào các nội dung được hiển thị.

Chọn Optional Header, tìm AddressOfEntryPoint trong bảng bên phải chứa giá trị 0x00012475 và cũng trong bảng bên phải, tìm ImageBase chứa giá trị 0x01000000.



Ta có một chương trình hiển thị MessageBox như sau:

```
#include <windows.h>

int main(int argc, char * argv[])

{

 MessageBox(NULL, L"Code injected", L"Info", MB_OK);

 return 0;

}
```

Biên dịch chương trình dưới chế độ Release, Not Using Precompiled Headers. Sử dụng IDA Pro để mở file PE và xem mã hợp ngữ của chương trình vừa biên dịch.

```
.text:00401000 sub_401000      proc near               ; CODE XREF: start-8D↓p
.text:00401000                 push    0               ; uType
.text:00401002                 push    offset Caption  ; "Info"
.text:00401007                 push    offset Text     ; "Code injected"
.text:0040100C                 push    0               ; hWnd
.text:0040100E                 call    ds:MessageBoxW
.text:00401014                 xor     eax, eax
.text:00401016                 retn
.text:00401016 sub_401000      endp
```

Và mã hex của Caption và Text

```
00402100  49 00 6E 00 66 00 6F 00  00 00 00 00 43 00 6F 00  I.n.f.o.....C.o.
00402110  64 00 65 00 20 00 69 00  6E 00 6A 00 65 00 63 00  d.e.·.i.n.j.e.c.
00402120  74 00 65 00 64 00 00 00  00 00 00 00 00 00 00 00  t.e.d...........
00402130  7A 02 F0 65 00 00 00 00  02 00 00 00 67 00 00 00  z.=e........g...
00402140  9C 22 00 00 9C 14 00 00  00 00 00 00 7A 02 F0 65  £"..£.......z.=e
00402150  00 00 00 00 0C 00 00 00  14 00 00 00 04 23 00 00  .............#..
```

Tương tự với bài trước ta cần tìm X, Y, Z trong đoạn hợp mã để chèn và thực thi chức năng thành công vào file Notepad.exe

push 0 ; 6a 00

push Caption ; 68 X

push Text ; 68 Y

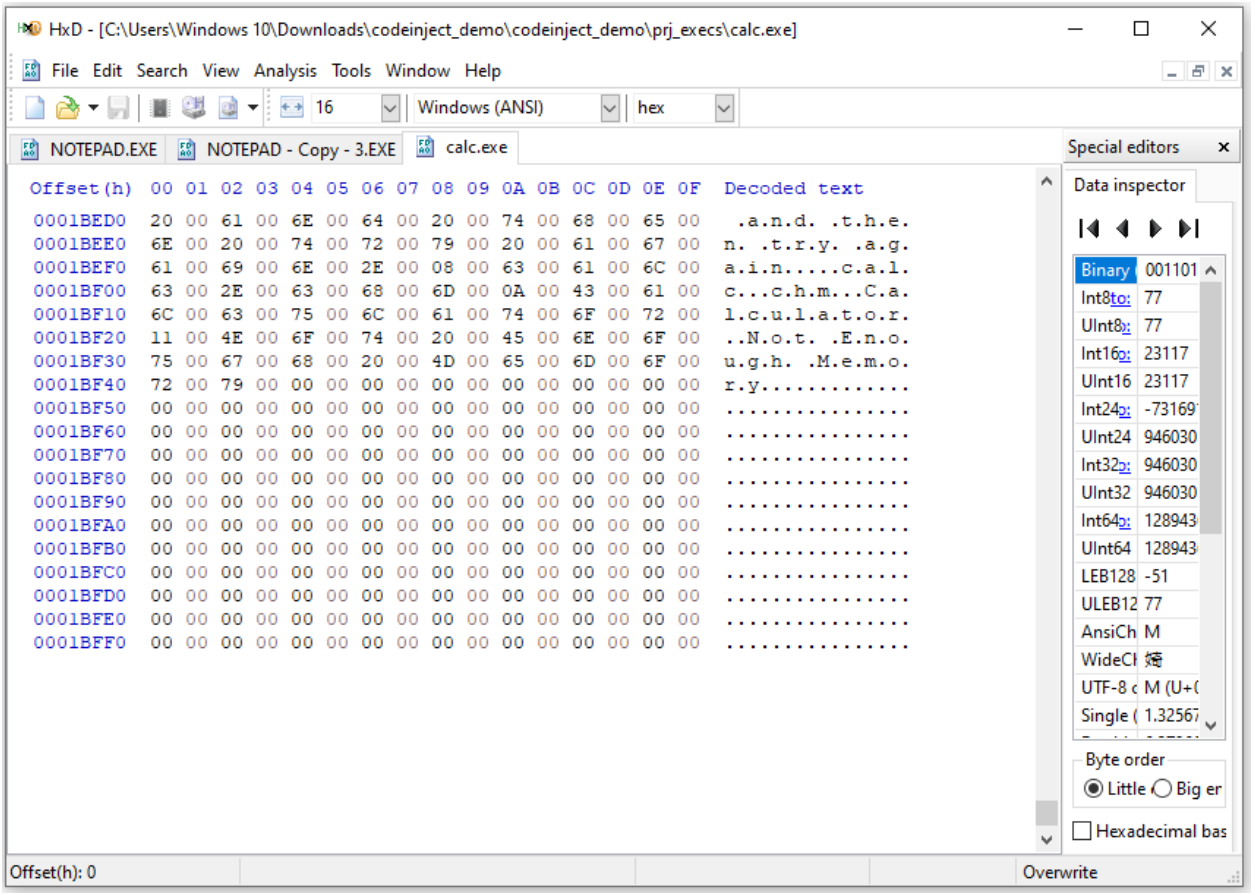
push 0 ; 6a 00

call [MessageBoxW] ; ff15 Z

Giá trị Z chính là địa chỉ của hàm MessageBoxW được import từ thư viện USER32.dll. Trong IDA Pro, mở calc.exe, chọn View -> Open Subviews -> Imports và ta thấy địa chỉ của hàm MessageBoxW chính là 0x010011A8

Address	Ordinal	Name	Library
01001178		DestroyWindow	USER32
0100117C		SetMenu	USER32
01001180		GetWindowRect	USER32
01001184		SystemParametersInfoW	USER32
01001188		DispatchMessageW	USER32
0100118C		TranslateMessage	USER32
01001190		TranslateAcceleratorW	USER32
01001194		IsChild	USER32
01001198		IsDialogMessageW	USER32
0100119C		GetMessageW	USER32
010011A0		LoadAcceleratorsW	USER32
010011A4		CreateWindowExW	USER32
010011A8		MessageBoxW	USER32
010011AC		LoadStringW	USER32
010011B0		SetProcessDefaultLayout	USER32
010011B4		GetProcessDefaultLayout	USER32
010011BC		__CxxFrameHandler	msvcrt
010011C0		_CxxThrowException	msvcrt
010011C4		wcstoul	msvcrt
010011C8		toupper	msvcrt
010011CC		wcschr	msvcrt
010011D0		memmove	msvcrt
010011D4		wcslen	msvcrt
010011D8		_wcsrev	msvcrt

Nhận xét thấy ở cuối file có 1 khoảng trống, ta sẽ lợi dụng khoảng trống này để chèn mã vào mà không làm tăng kích thước file



Ta chọn các địa chỉ sau cho việc lưu trữ:

- 0x0001BF60 cho lưu trữ hợp ngữ

- 0x0001BFA0 cho lưu trữ Caption

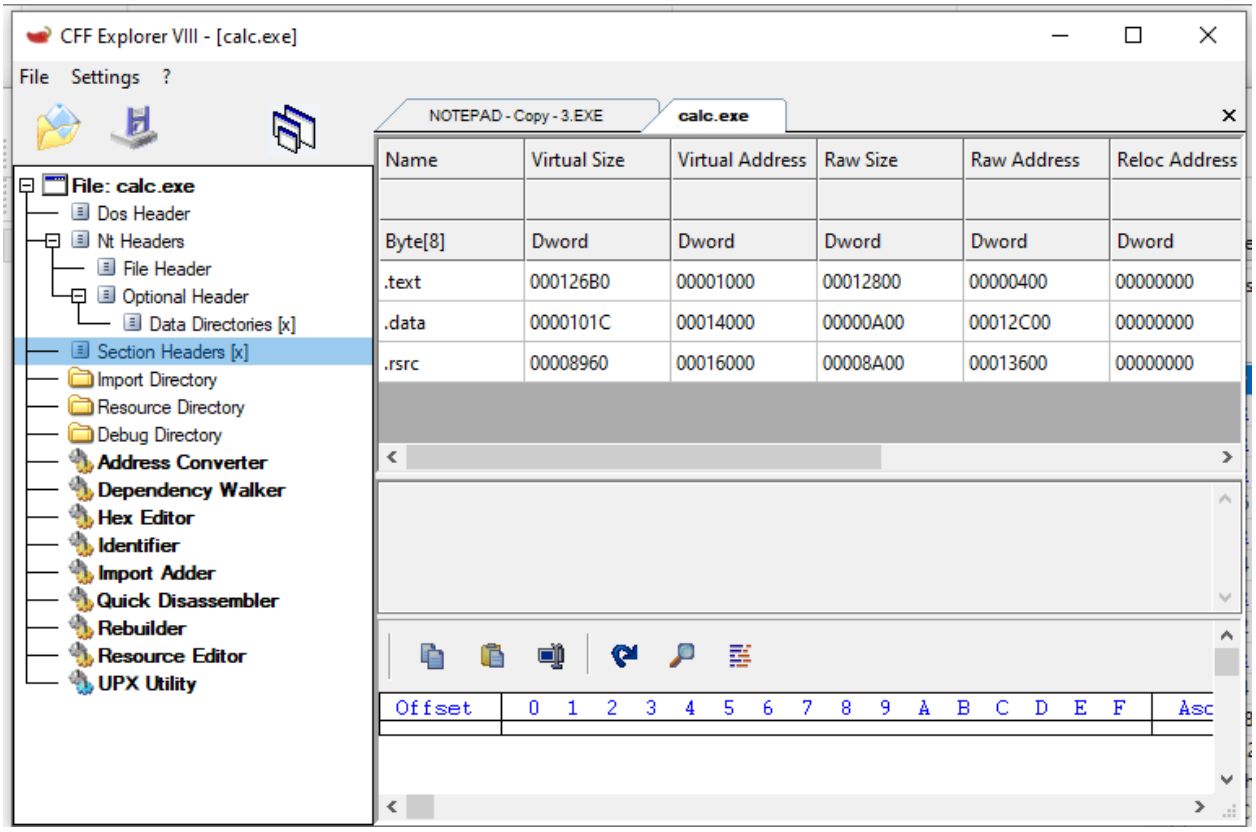
- 0x0001BFC0 cho lưu trữText

Ta sẽ sử dụng công thức sau để tính X, Y:

Offset = RA – Section RA = VA – Section VA

Trong đó X là VA của Caption và Y là VA của Text, RA là địa chỉ vật lý của các phần cần lưu trữ

Và các số Section RA, Section VA là các giá trị trong Section headers của .rsrc



Cụ thể trong trường hợp này:

- Section RA = 0x00013600

- Section VA = 0x00016000

Vậy từ đây ta có các phép tính như sau:

X = VA của Caption = RA của Caption – Section RA + Section VA

= 0x0001BFA0 – 0x00013600 + 0x00016000

= 0x0001E9A0

Y = VA của Text = RA của Text – Section RA + Section VA

= 0x0001BFC0 – 0x00013600 + 0x00016000

= 0x0001E9C0

Cộng thêm ImageBase, suy ra X = 0x0101E9A0. Tương tự , Y = 0x0101E9C0.

Tiếp theo ta tính địa chỉ thực thi mới của đoạn code, sử dụng công thức như trên

Z = new_entry_point = 0x0001BF60 - 0x00013600 + 0x00016000 = 0x0001E960

Thêm ImageBase thì new_entry_point = 0x0101E960

Ta cần chương trình tiếp tục thực thi sau khi chạy đoạn code trên ta càn chèn dòng lệnh quay về AddressOfEntryPoint cũ ngay sau đoạn code mã độc: jmp relative_VA

Relative_VA có thể tính theo công thức sau:

old_entry_point = jmp_instruction_VA + 5 + relative_VA (2)

Néu đặt lệnh jmp sau 5 câu lệnh ở bước tìm hợp ngữ thì jmp_instruction_VA = 0x0101E960 + 0x14 = 0x0101E974. (Vì 0x14 là số byte của 5 câu lệnh)

Vậy relative_VA = old_entry_point - jmp_instruction_VA – 0x5

= 0x010012475 - 0x0101E974 – 0x5

= 0xFFFF3AFC

Đến đây, ta đã có một đoạn mã hợp ngữ hoàn chỉnh để chèn vào Notepad.exe. Các địa chỉ được biểu diễn theo thứ tự little endian (x86).

push 0 ; 6a 00

push Caption ; 68 A0E90101

push Text ; 68 C0E90101

push 0 ; 6a 00

call [MessageBoxW] ; ff15 A8110001

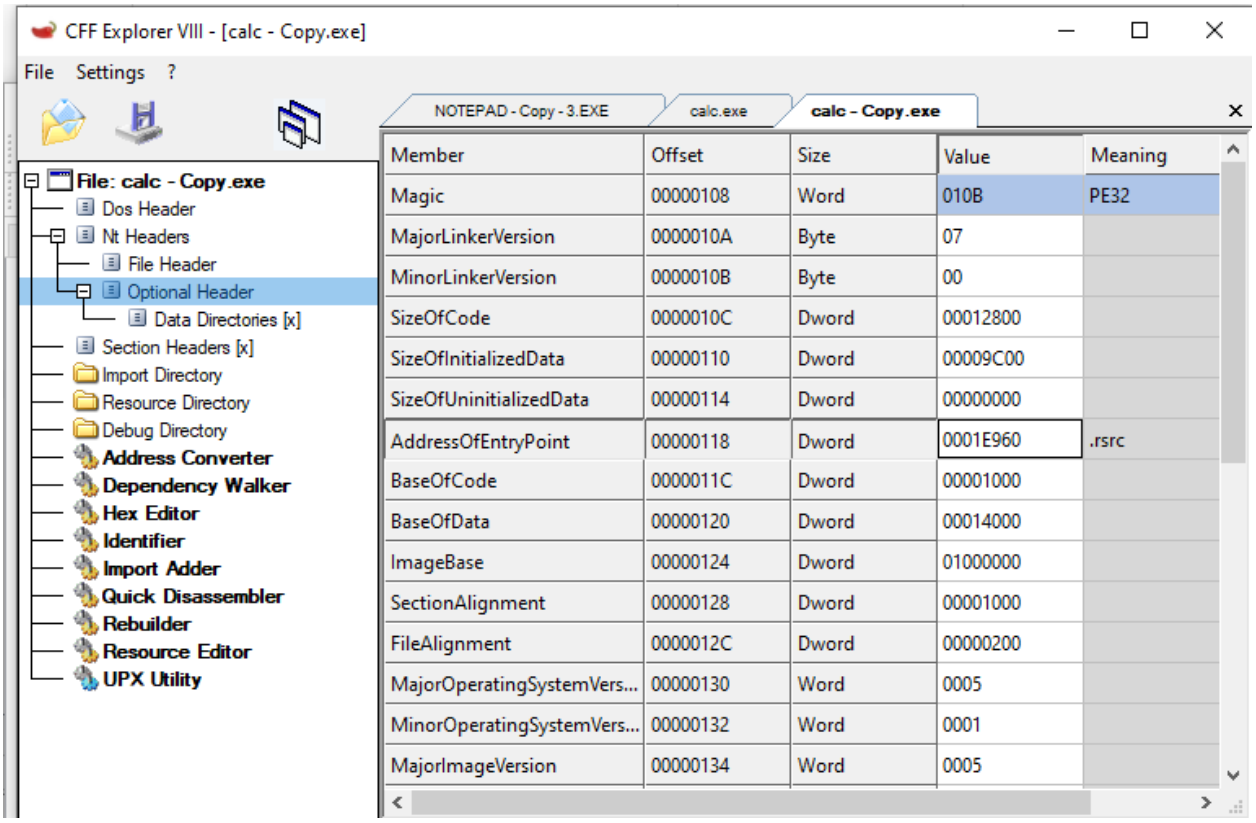
jmp Origianl_Entry_Point ; e9 FC3AFFFF

Sử dụng HxD để chèn đoạn mã cùng với giá trị Caption và Text vào Notepad.exe. Lưu lại file.

```
0001BF60  6A 00 68 A0 E9 01 01 68 C0 E9 01 01 6A 00 FF 15  j.h é..hÀé..j.ÿ.
0001BF70  A8 11 00 01 E9 FC 3A FF FF 00 00 00 00 00 00 00  ¨...éü:ÿÿ.......
0001BF80  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0001BF90  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0001BFA0  49 00 6E 00 66 00 6F 00 00 00 00 00 00 00 00 00  I.n.f.o.........
0001BFB0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0001BFC0  43 00 6F 00 64 00 65 00 20 00 69 00 6E 00 6A 00  C.o.d.e. .i.n.j.
0001BFD0  65 00 63 00 74 00 65 00 64 00 00 00 00 00 00 00  e.c.t.e.d.......
0001BFE0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

Cuối cùng là hiệu chỉnh các tham số trong PE header

Sử dụng CFF Explorer để thay đổi các giá trị của AddressOfEntryPoint thành địa chỉ mới là 0x0001E960 (Địa chỉ new_entry_point)



Lưu lại thay đổi. Thực thi file Calc.exe, một cửa sổ xuất hiện như sau: