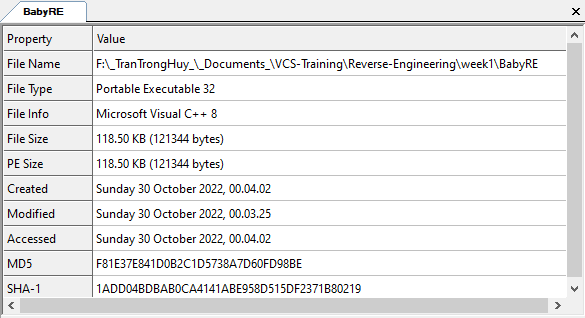
**Week1 – Easy Crackme, Keygenme**

1. **BabyRE**

Check thông tin file:



Disassembly main function, comment code and rename function:

----------------------------------------------------------------------------------------

int \_\_cdecl main(int argc, const char \*\*argv, const char \*\*envp)

{

unsigned int i; // esi

int j; // edi

char \*v5; // esi

int v6; // ecx

\_\_int128 v8; // [esp+0h] [ebp-28h] BYREF

int v9; // [esp+10h] [ebp-18h]

int v10; // [esp+14h] [ebp-14h]

\_\_int16 v11; // [esp+18h] [ebp-10h]

char v12[8]; // [esp+1Ch] [ebp-Ch] BYREF

v8 = xmmword\_AFB6E0; // 'gunmifun|iaQpndb'

v9 = 1182494304; // 'F{n'

v10 = 1919444577; // 'rhfa'

v11 = 1111; // '\x04W'

// v8 -> v11 = 25 char

\_printf("Guest your flag. The flag will be of the form flag{[a-zA-Z0-9]+}\n");

\_printf("Enter your key:");

for ( i = 0; i < 5; ++i )

\_scanf("%hhd", &v12[i]); // '%hhd' input 1 byte | length v12 = 5

for ( j = 0; j < 26; j += 2 )

{

v5 = (char \*)&v8 + j; // pointer v5 = v8 index even

\*v5 ^= v12[j % 5u]; // encrypt XOR with v12 index even

// xor [esi], al ; al = key[j % 5]

v6 = j;

v5[1] ^= v12[v6 - 5 \* ((unsigned int)&v5[1 - (\_DWORD)&v8] / 5) + 1]; // pointer v5[1] = v8 index odd

// xor [esi+1], al ; al = key[(j+1)%5]

}

// str [0] [1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [...]

// key [0] [1] [2] [3] [4] [0] [1] [2] [3] [4] [0] [...]

\_printf("flag is : %s\n; // lea eax, [ebp-28h] ; v8

// push eax

// push offset aFlagIsS ; "flag is : %s\n"

// call \_\_printf

return 0;

}

----------------------------------------------------------------------------------------

Ta kiểm tra giá trị v8, v9, v10, v11 là chuỗi str[] = “**bdnpQai|nufimnug`n{FafhrW**”. Khi chạy file chương trình khởi tạo và lưu giá trị chuỗi str[] lên stack, sau đó cho nhập (key) gồm 5 chữ số và dùng (key) XOR str[] => flag (key xor lặp lại 5 giá trị số đến khi hết chuỗi str[]). Vì vậy nhập đúng key của tác giả sẽ lấy được flag.

Debug quá trình XOR key:

esi = &str[j]

-----------------------------------

xor [esi], al ; al = key[j%5]

…

xor [esi+1], al ; al = key[(j+1)%5]

…

str [0] [1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [...]

key [0] [1] [2] [3] [4] [0] [1] [2] [3] [4] [0] [...]



Ta có form flag là ‘flag{’ từ đó XOR với str[] để lấy đúng key của tác giả.

* Tìm đúng key bằng cách XOR lại 5 ký tự đầu
* Key = str[0, 1, 2, 3, 4] XOR “flag{”

----------------------------------------------------------------------------------------------------

C:\Users\Admin>python

Python 3.10.2 (tags/v3.10.2:a58ebcc, Jan 17 2022, 14:12:15) [MSC v.1929 64 bit (AMD64)] on win32

Type "help", "copyright", "credits" or "license" for more information.

>>> str = "bdnpQ"

>>> flag = "flag{"

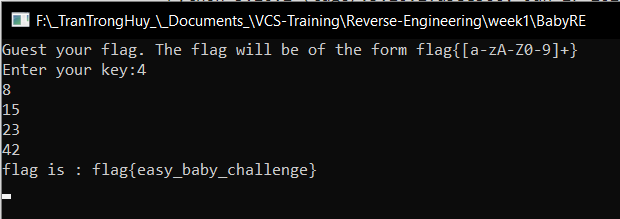
>>> for i in range(5):

... print(ord(str[i])^ord(flag[i]))

...

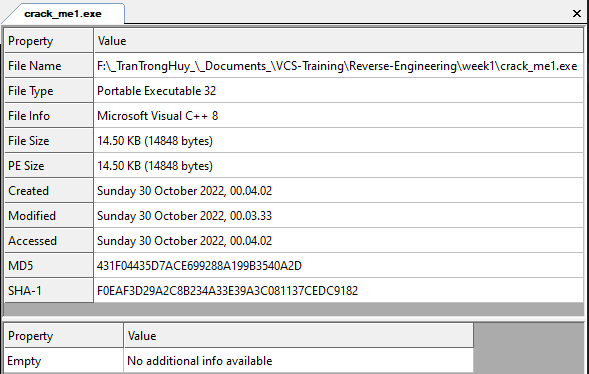
4, 8, 15, 23, 42

* Key = [4, 8, 15, 23, 42]



**Flag: flag{easy\_baby\_challenge}**

1. **Crack\_me1.exe**



Sau khi run chương trình thì ta thấy cũng giống như bài BabyRE, nếu chúng ta nhập đúng password mà chương trình yêu cầu thì chương trình sẽ in flag cho chúng ta (flow chương trình cơ bản là vậy ☺) .

Phân tích code

----------------------------------------------------------------------------------------

int \_\_cdecl main(int argc, const char \*\*argv, const char \*\*envp)

{

char v4; // [esp+0h] [ebp-2CCh]

char v5; // [esp+0h] [ebp-2CCh]

char v6; // [esp+0h] [ebp-2CCh]

char Format[160]; // [esp+8h] [ebp-2C4h] BYREF

char v8; // [esp+A8h] [ebp-224h] BYREF

char v9[299]; // [esp+A9h] [ebp-223h] BYREF

char v10[200]; // [esp+1D4h] [ebp-F8h] BYREF

void \*v11; // [esp+29Ch] [ebp-30h]

int v12; // [esp+2A0h] [ebp-2Ch]

int v13; // [esp+2A4h] [ebp-28h]

char \*v14; // [esp+2A8h] [ebp-24h]

int v15; // [esp+2ACh] [ebp-20h]

char \*v16; // [esp+2B0h] [ebp-1Ch]

int v17; // [esp+2B4h] [ebp-18h]

int v18; // [esp+2B8h] [ebp-14h]

char \*v19; // [esp+2BCh] [ebp-10h]

char \*v20; // [esp+2C0h] [ebp-Ch]

int i; // [esp+2C8h] [ebp-4h]

v8 = 0;

memset(v9, 0, sizeof(v9));

v12 = 335;

memset(v10, 0, sizeof(v10));

strcpy(

Format,

"Do you remember the good old days?! I don't know how about you,but I don't. Please help me to recover my memory, it'"

"s password protected, and that's sad :(\n");

v11 = &unk\_404BE8;

\_printf(Format, v4);

\_printf("Enter password: ", v5);

\_scanf("%300[^\n]s", (char)&v8);

v20 = &v8;

v16 = v9;

v20 += strlen(v20);

v15 = ++v20 - v9;

v17 = v20 - v9;

v19 = &v8;

v14 = v9;

v19 += strlen(v19);

v13 = ++v19 - v9;

if ( (unsigned int)(v19 - v9) >= 294 ) // strlen(v8)

{

if ( (unsigned \_\_int8)check\_Password(&v8) )

{

v18 = v17 / 3;

for ( i = 0; i < v17; ++i )

v10[i % v18] ^= v9[i - 1];

for ( i = 0; i < v12; ++i )

enc\_flag[i] ^= v10[i % v18];

\_printf("\n\nCongratulation! Here is your memo :> \n\n", v6);

\_printf("%s", (char)enc\_flag);

}

else

{

\_printf("\nInvalid password\n", v6);

}

getchar();

getchar();

return 0;

}

else

{

\_printf("oh, no!", v6);

return 0;

}

}

----------------------------------------------------------------------------------------

Hàm main cho chúng ta nhập vào password rồi kiểm tra độ dài chuỗi password, tiếp theo là chuyển qua hàm check\_Password để xử lý, nếu đúng pass thì thực hiện giải mã trên chuỗi enc\_flag và in flag.

----------------------------------------------------------------------------------------

char \_\_cdecl check\_Password(const char \*a1)

{

int i; // [esp+14h] [ebp-10h]

\_DWORD \*v3; // [esp+18h] [ebp-Ch]

v3 = &unk\_404BE8;

if ( (int)strlen(a1) < 55 )

return 0;

for ( i = 0; i < 122; ++i )

{

if ( !(unsigned \_\_int8)check\_Char(\*v3, &a1[v3[1]], v3 + 2) )

return 0;

v3 += 3;

}

return 1;

}

----------------------------------------------------------------------------------------

Hàm check\_Password chuyển tiếp tới hàm check\_Char để kiểm tra password.

----------------------------------------------------------------------------------------

char \_\_cdecl check\_Char(int a1, int a2, int a3)

{

char v4[4]; // [esp+0h] [ebp-20h] BYREF

LPVOID inited; // [esp+4h] [ebp-1Ch]

LPVOID v6; // [esp+8h] [ebp-18h]

SIZE\_T v7; // [esp+Ch] [ebp-14h]

SIZE\_T v8; // [esp+10h] [ebp-10h]

LPVOID lpAddress; // [esp+14h] [ebp-Ch]

SIZE\_T dwSize; // [esp+18h] [ebp-8h]

char v11; // [esp+1Fh] [ebp-1h]

v8 = 221;

inited = Init\_func((int)&unk\_E44288, 0xDDu, 5);// Create function to check char in password

v7 = 278;

v6 = Init\_func((int)&unk\_E44170, 0x116u, 6);

if ( !inited || !v6 )

return 0;

switch ( a1 )

{

case 1:

dwSize = 97;

lpAddress = Init\_func((int)&unk\_E44B80, 0x61u, 1);

break;

case 2:

dwSize = 142;

lpAddress = Init\_func((int)&unk\_E44AF0, 0x8Eu, 2);

break;

case 3:

dwSize = 1685;

lpAddress = Init\_func((int)&unk\_E44458, 0x695u, 3);

break;

case 4:

dwSize = 235;

lpAddress = Init\_func((int)&unk\_E44368, 0xEBu, 4);

break;

default:

return 0;

}

if ( !lpAddress )

return 0;

v11 = ((int (\_\_cdecl \*)(int, int, char \*))lpAddress)(a2, a3, v4);

VirtualFree(lpAddress, dwSize, 0x8000u);

VirtualFree(inited, v8, 0x8000u);

VirtualFree(v6, v7, 0x8000u);

return v11;

}

----------------------------------------------------------------------------------------

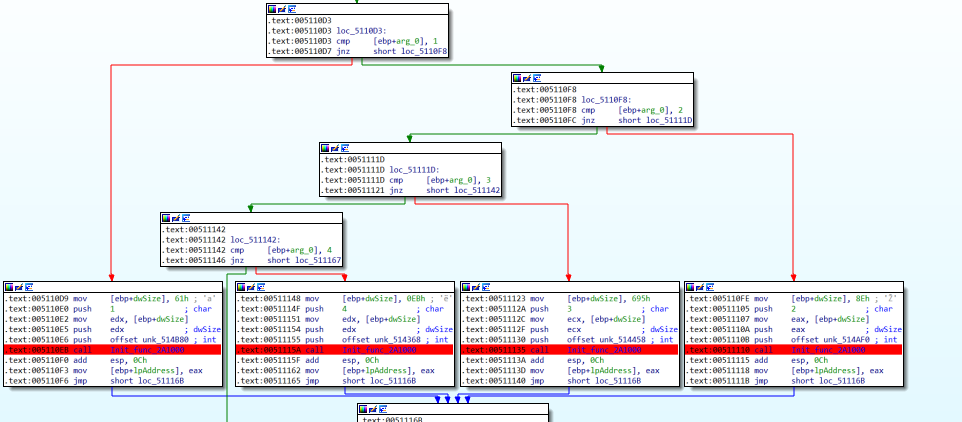
Hàm check\_Char gọi các hàm được khởi tạo trong quá trình chạy chương trình và xử lý dữ liệu là chuỗi đầu vào được khởi tạo ở hàm check\_pass v3 = &unk\_404BE8; Sau khi phân tích thì data mà con trỏ v3 trỏ tới gồm một chuỗi dữ liệu với mỗi thành phần gồm ba dữ liệu chính.

+ inst: cái này là case trong switch của hàm check\_Char tương ứng với kiểu sẽ chọn để xử lý với dữ liệu ở đó

+ input\_index: cái này giống với vị trí của dữ liệu sau khi được xử lý sẽ điền vào (là index của chuỗi password ta cần tìm).

+ cons: là data chưa được xử lý.

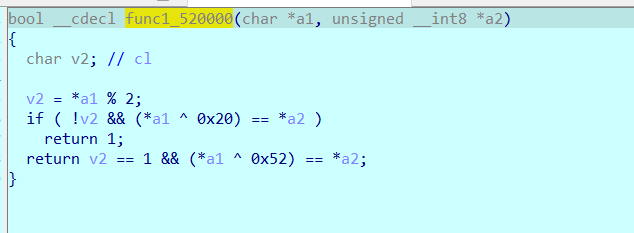
Sau một hồi debug, ta tìm được đoạn mã resolve các hàm encrypt sẽ gọi đến trong check\_Char, nó gồm 4 trường hợp xử lý.



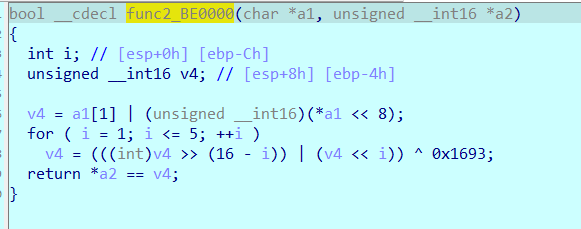
Để đọc giả mã của những hàm được gọi đến trong bốn trường hợp ta phải debug chương trình rồi mới đọc được các hàm được khởi tạo trong quá trình chạy.

Debug run qua dòng đặt breakpoint thì địa chỉ hàm encrypt được resolve lưu trong eax, jmp đến địa chỉ đó rồi ấn C (make Code) để convert từ shellcode sang assembly rồi chuột phải chọn Create Funtion => khi đó ta có thể F5 để đọc mã giả code.

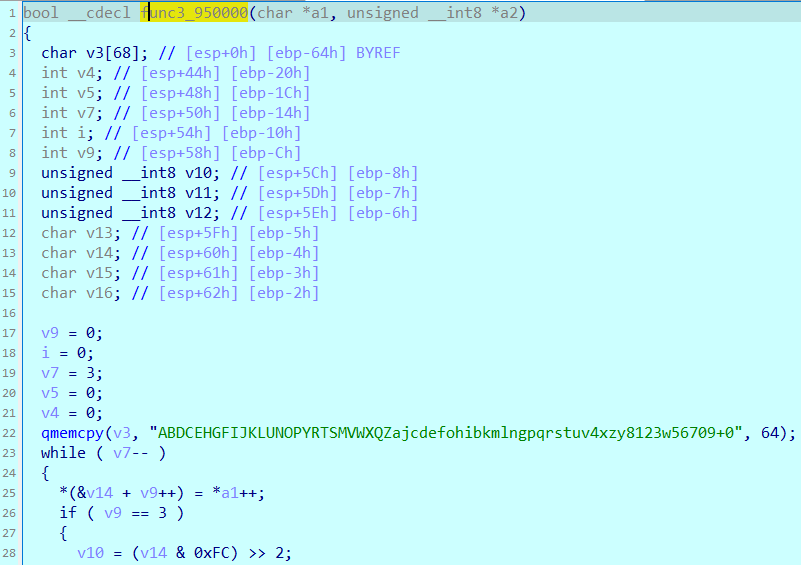
Func 1:

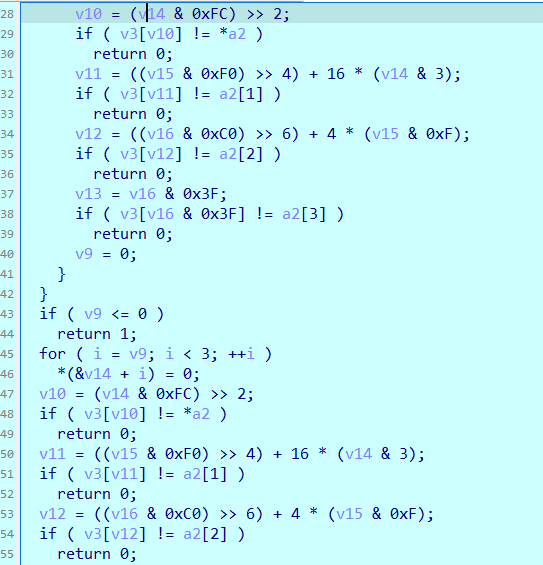


Func 2:

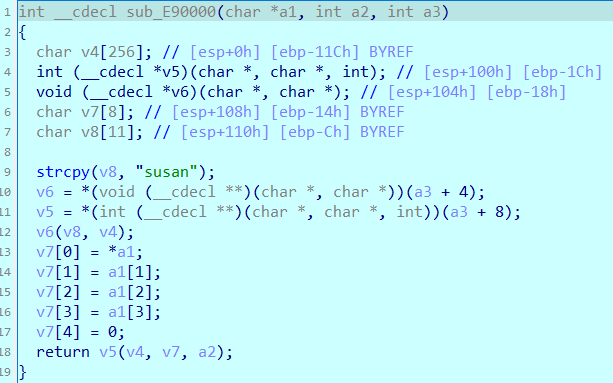


Func 3:





Func 4:



Hàm func01:

Kiểm tra kí tự a[i] tại vị trí idx bằng cách XOR với 0x20 nếu chẵn hoặc 0x52 nếu lẻ sau đó so sánh với BYTE trong data.

Hàm func02:

Kiểm tra kí tự a[i] và a[i+1] tại vị trí idx bằng cách dịch bit nhiều lần và xor với 0x1693 sau đó so sánh với WORD trong data.

Hàm func03:

Kiểm tra kí tự a[i], a[i+1], a[i+2] tại vị trí idx bằng phép dịch bit (res) sau đó so sánh với charset[res] với từng BYTE trong data.

Hàm func04 (gồm 2 hàm nhỏ init\_array và encrypt\_array):

Kiểm tra kí tự a[i], a[i+1], a[i+2], a[i+3] tại vị trí idx bằng array tạo ra từ init\_array(bằng cách dùng const char "susan") sau đó xor bằng hàm encrypt\_array để kiểm tra với từng BYTE trong data.

* Ta sẽ viết một hàm crack\_me1.py để reverse lại chức năng của 4 func trên của chương trình lấy password và dữ liệu vào ta sẽ đọc lấy từ địa chỉ v3 = &unk\_A54BE8.
* Run đoạn mã python trong IDA sau để đọc lấy data từ con trỏ v3 được khởi tạo từ chương trình trong hàm check Pass\_word.

----------------------------------------------------------------------------------------------------

data = []

addr = 0xA54BE8 # v3

index = 0

for i in range(366//3): # 0x00A5519F – 0x00A54BE8 = (1463 / 4) = 365 => 366 // 3

t1 = idc.get\_wide\_dword(addr+index+0\*4)

t2 = idc.get\_wide\_dword(addr+index+1\*4)

t3 = idc.get\_wide\_dword(addr+index+2\*4)

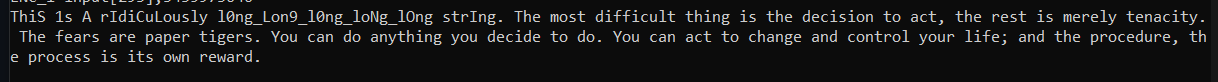
data.append((t1,t2,t3))

index+=3\*4

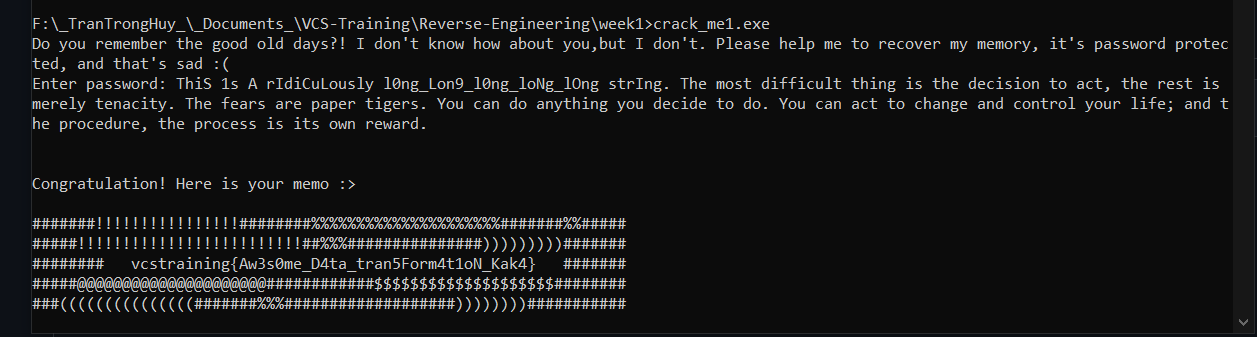
print(data)

----------------------------------------------------------------------------------------------------

Sau khi run crack\_me1.py ta lấy được password:



Lấy flag:



**#######!!!!!!!!!!!!!!!!########%%%%%%%%%%%%%%%%%%%%%#######%%#####**

**#####!!!!!!!!!!!!!!!!!!!!!!!!!##%%%###############)))))))))#######**

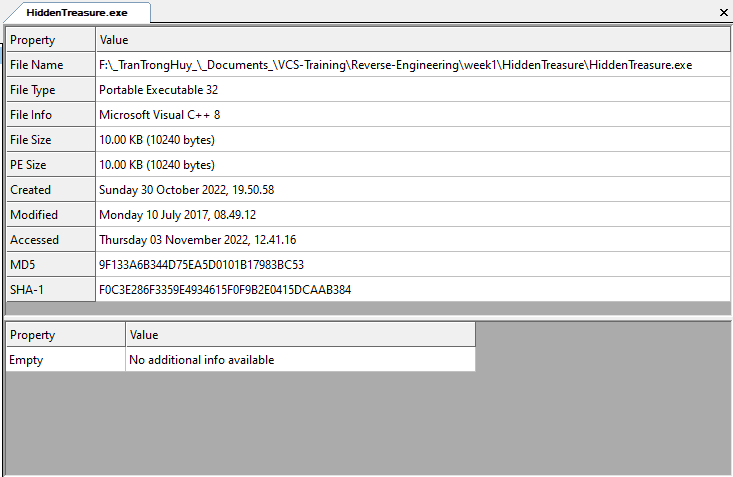
**######## vcstraining{Aw3s0me\_D4ta\_tran5Form4t1oN\_Kak4} #######**

**#####@@@@@@@@@@@@@@@@@@@@@############$$$$$$$$$$$$$$$$$$$$########**

**###(((((((((((((((#######%%%###################))))))))###########**

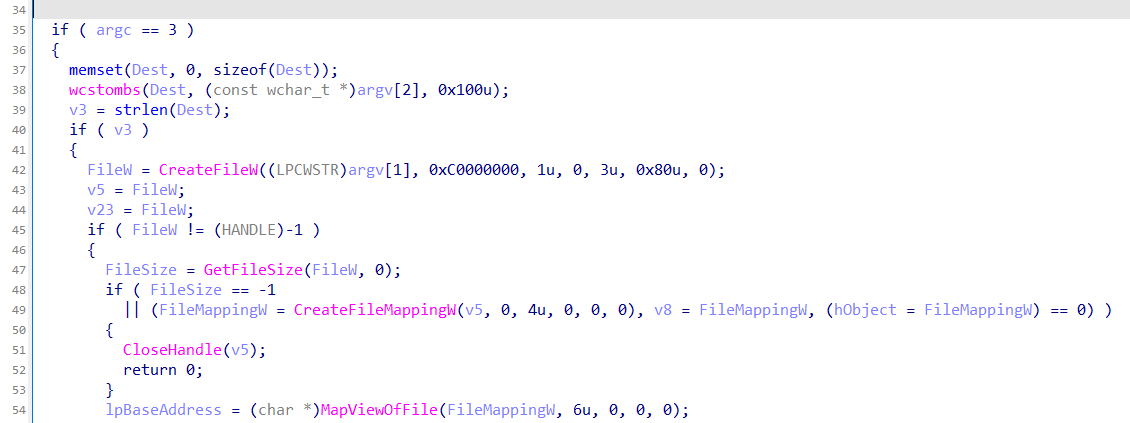
1. **HiddenTreasure**

Check thông tin file:

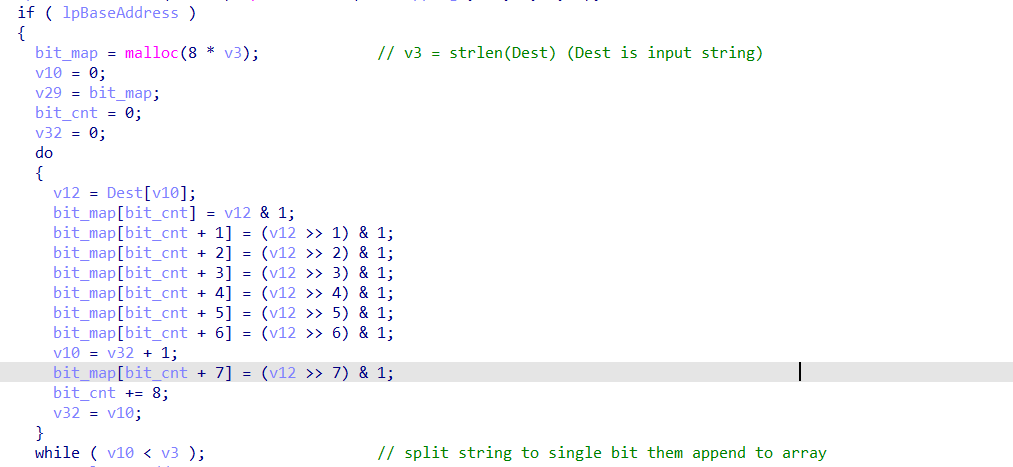
****

Đọc và phân tích code của chương trình, ta thấy chương trình sẽ lấy input từ hai đối số là **ảnh (.bmp)** và **chuỗi string** để chèn thông tin và giấu vào ảnh. Ta sẽ chỉ tập trung quan tâm đến những đoạn code nhằm thực thi các tác vụ trên để phân tích chương trình.

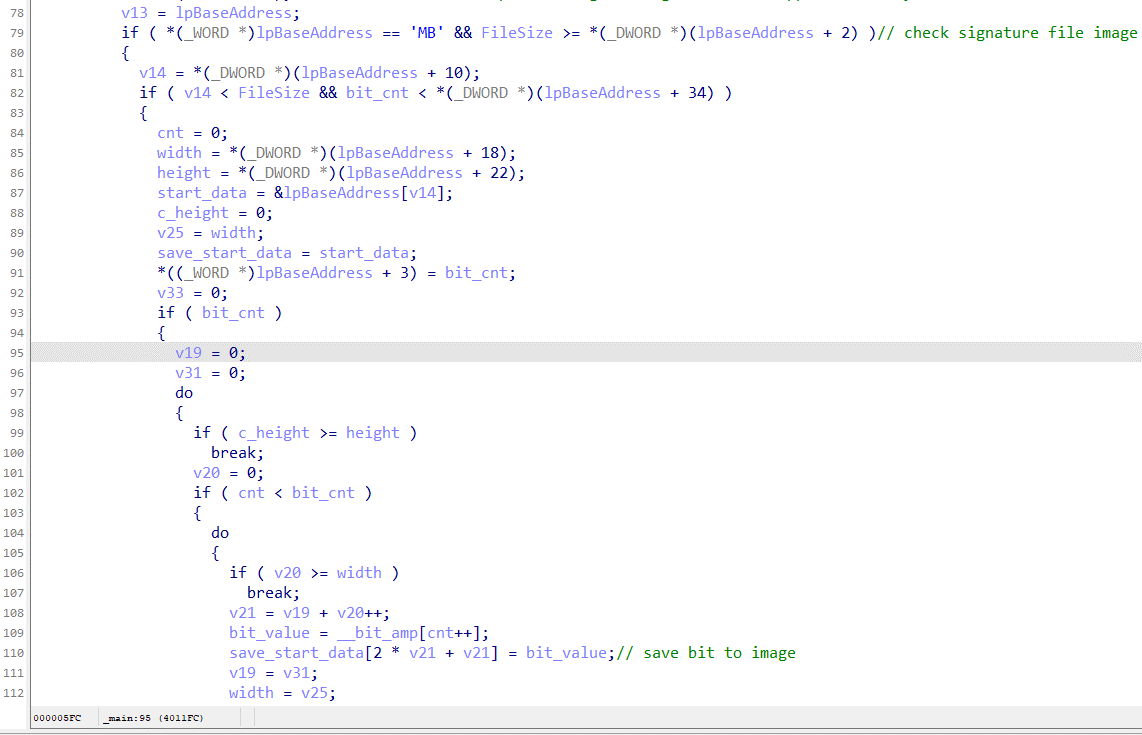
Đầu tiên là phần tạo mảng và đọc thông tin, nó lấy hai biến đó từ hai đối số của chương trình là **file ảnh(argv1)** và **string(argv2).**

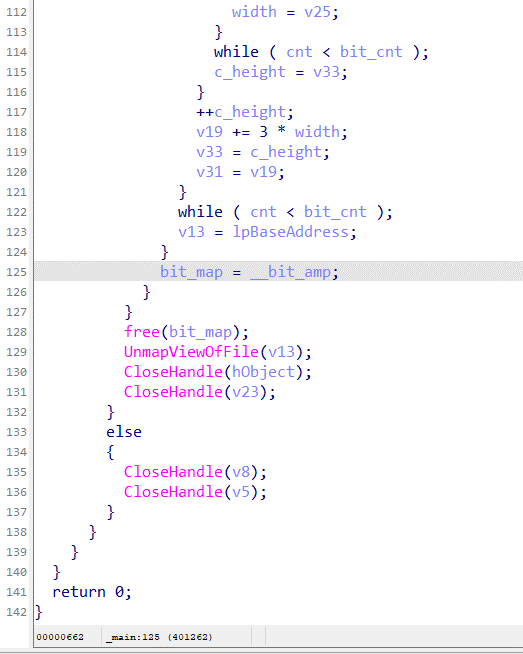
****

Sau đó chương trình tách các bit của input string và lưu vào mảng:

****

Đoạn code tiếp theo thì check signature của file ảnh .bmp rồi bắt đầu thực thi việc giấu thông tin vào ảnh.

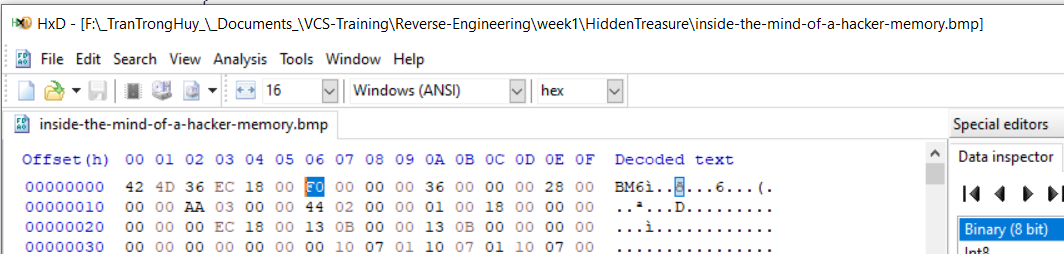




Chương trình kiểm tra header file và size có phù hợp để tiến hành chèn chuỗi vào được không, ta nhìn vào dòng có comment **`save bit to image`.** Đơn giản thì nó sẽ chèn các bit từ bit\_map và chèn chúng vào phần data của ảnh.

* Ta có thể lấy được flag từ tấm ảnh đã cho đi kèm theo file.

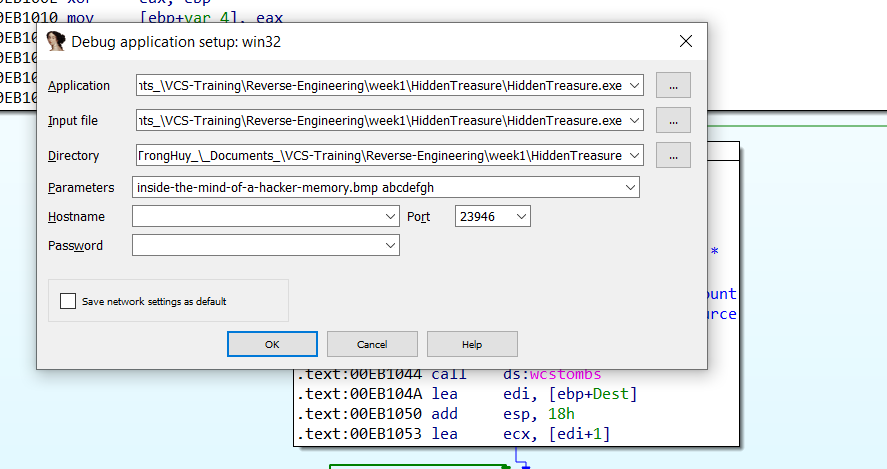
Dùng HxD để kiểm tra size của flag (ở vị trí word thứ 4 tính từ đầu file).



* Flag có 0xf0 (240) / 8 = 30 byte.

Ta tiến hành đọc data từ file để lấy được bit\_arr (đặt break point và dùng python để đọc data).

1. Thiết lập đối số khi debug.



1. Thay đổi giá trị của start\_data\_addr trong script thành giá trị lưu trong eax.

----------------------------------------------------------------------------------------------------

from idaapi import \*

bit\_arr = “”

hex\_arr = []

for i in range(0,240):

bit\_arr += str(ord(get\_bytes(**start\_data\_addr**+2\*i+i, 1).decode('utf-8')))

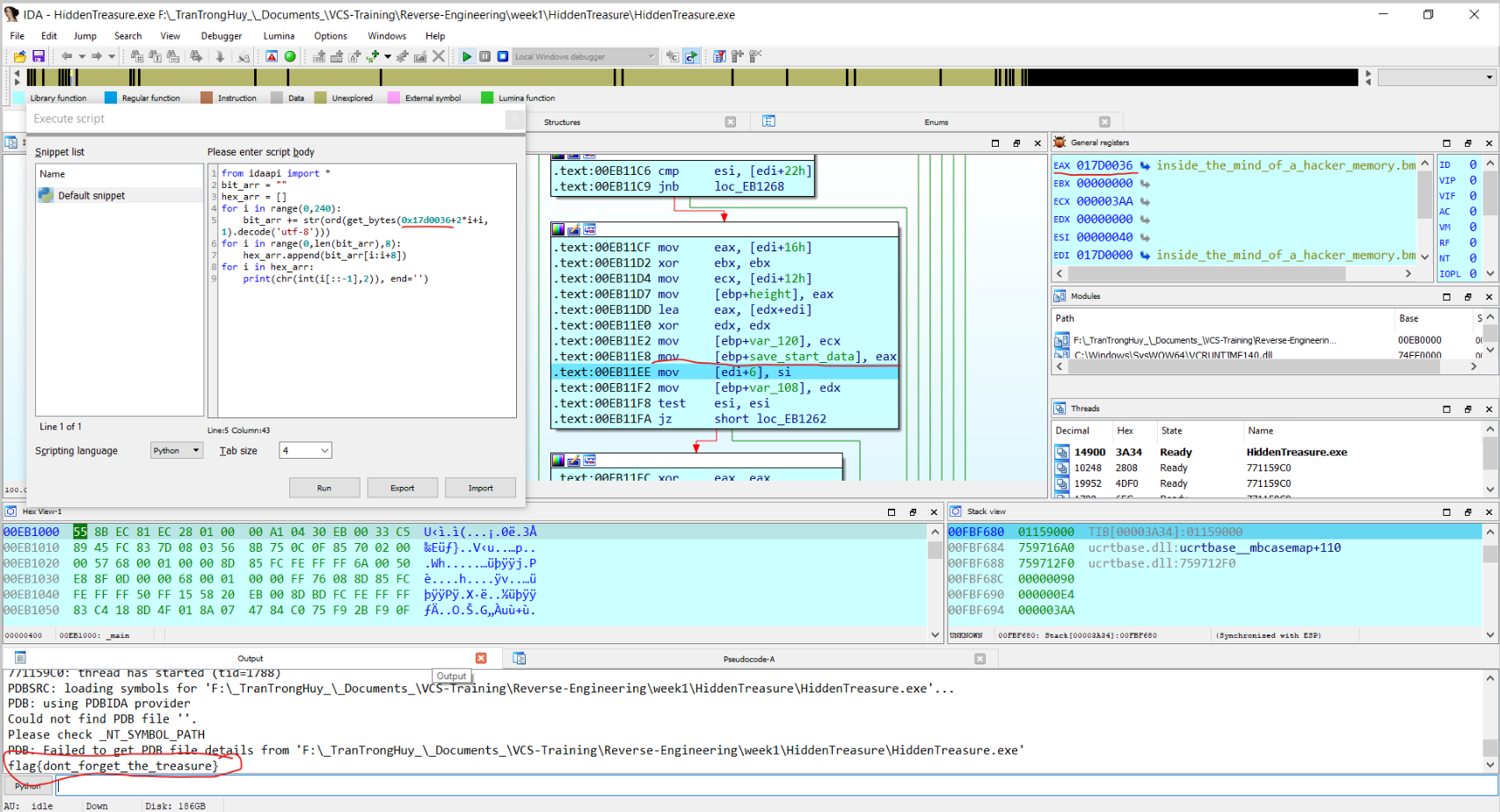
for i in range(0,len(bit\_arr),8):

hex\_arr.append(bit\_arr[i:i+8])

for i in hex\_arr:

print(chr(int(i[::-1],2)), end='')

----------------------------------------------------------------------------------------------------



**flag{dont\_forget\_the\_treasure}**