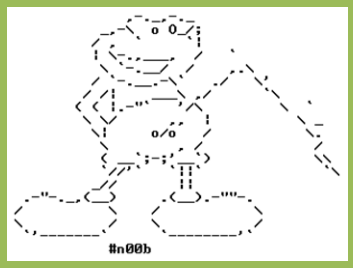
2018

# [Cracking with OllyDbg]

*Based on OllyDbg tuts of Ricardo Narvaja (CrackLatinos Team)*

www.reaonline.net

kienmanowar




21/01/2018

# Mục Lục

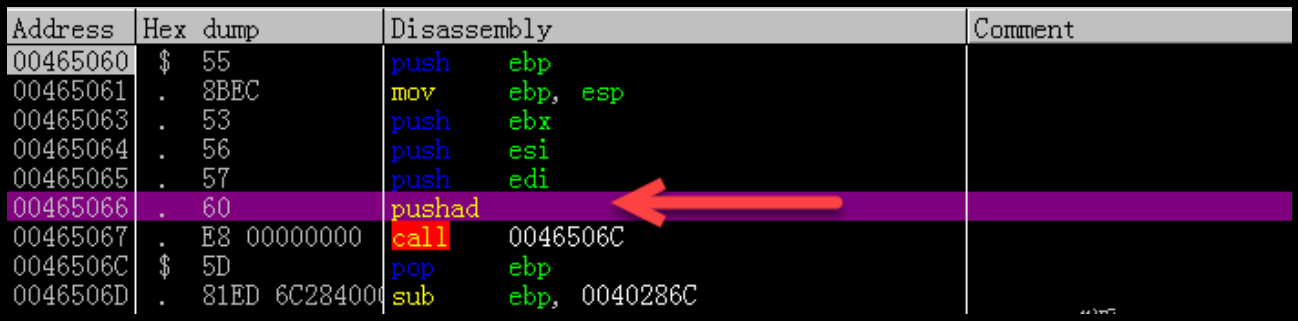
# I. Giới thiệu chung

Ở các phần trước, tôi đã hướng dẫn chi tiết quá trình unpack một unpackme được packed bởi tElock có áp dụng kĩ thuật chuyển hướng IAT (IAT redirected). Trong các phần tới đây, nếu có thời gian tôi sẽ tiếp tục nâng dần độ khó của packer lên. Với phần này, tôi sẽ dành thời gian để thực hành với một unpackme khác là **UnPackMe_YodasCrypter1.3.e.exe**.

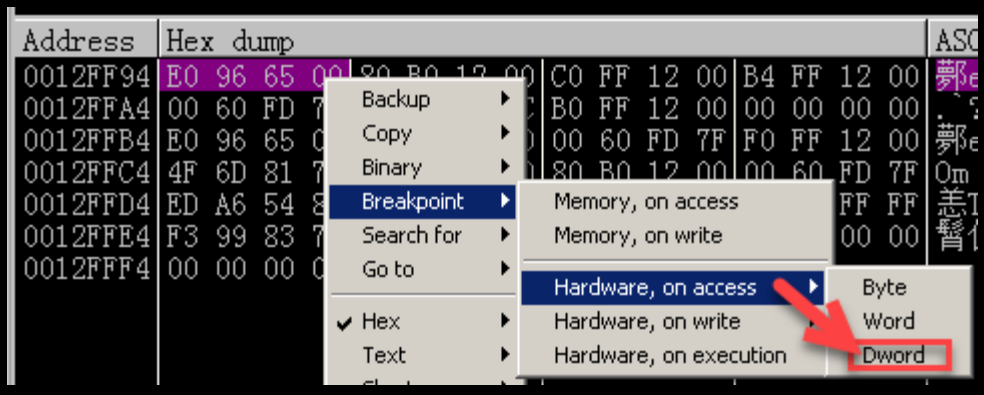
Để đảm bảo cho quá trình làm việc với các packers, OllyDbg cần được trang bị các plugins cần thiết để tránh bị phát hiện bởi các cơ chế anti-debug. Các plugins thì các bạn có thể tìm đọc trong các phần tôi viết về Anti-Debug hoặc tìm hiểu thêm thông qua các trang khác như tuts4you.com, v.v...

# II. Phân tích và xử lý target

Load unpackme vào OllyDbg, ta dừng lại tại Entry Point:




Quan sát các lệnh bên dưới, chúng ta thấy có lệnh PUSHAD. Thử áp dụng phương pháp PUSHAD xem có được không? Tiến hành trace code và trace qua lệnh PUSHAD bằng **F8/F7**. Sau đó chuyển qua cửa sổ Registers, chọn thanh ghi ESP, nhấn chuột phải và chọn **Follow in dump**. Đánh dấu 4 bytes đầu tiên và đặt một breakpoint: **Hardware, on access > Dword**.



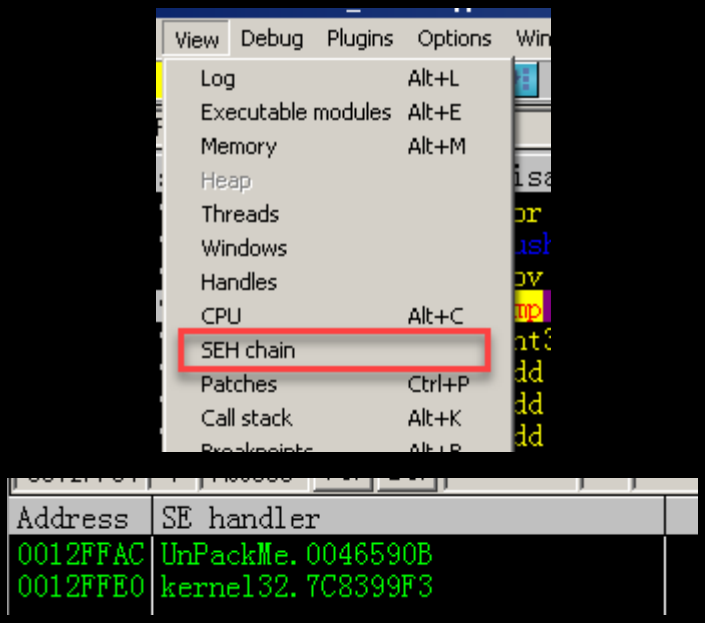

Sau khi đặt xong bp, nhấn **F9** để run, ta sẽ dừng tại đây:

```
Address   Hex dump   Disassembly                          Comment
00465976  50         push    eax                          UnPackMe.0046590B
00465977  33C0       xor     eax, eax
00465979  64:FF30    push    dword ptr fs:[eax]
0046597C  64:8920    mov     dword ptr fs:[eax], esp
0046597F  EB 01      jmp     short 00465982
00465981  CC         int3
00465982  0000       add     byte ptr [eax], al
00465984  0000       add     byte ptr [eax], al
```

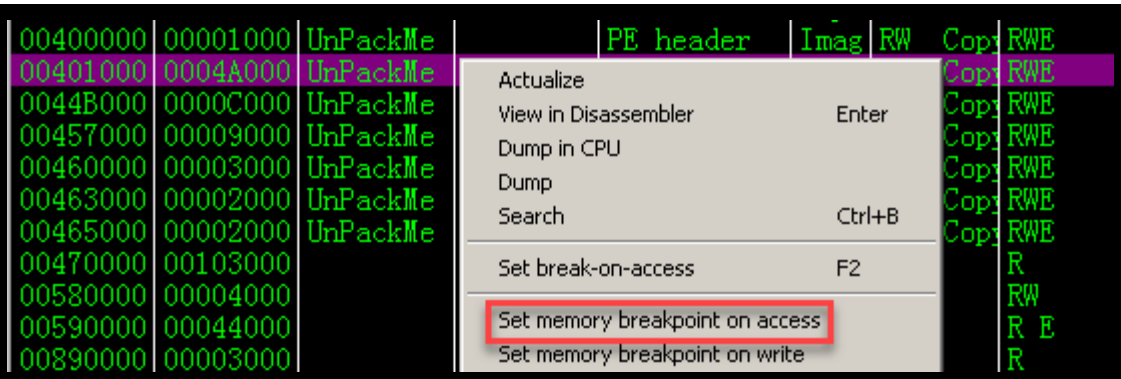
Nhìn vào đoạn code tại đây có thể thấy nó tạo ra một xử lý ngoại lệ (exception handler) và sau đó bằng lệnh JMP để nhảy đến một vùng code chứa toàn các bytes 0 bên dưới nhằm gây ra lỗi. Do vậy, nếu chúng ta trace code và dừng tại lệnh JMP.

```
Address   Hex dump   Disassembly                          Comment
00465977  33C0       xor     eax, eax
00465979  64:FF30    push    dword ptr fs:[eax]
0046597C  64:8920    mov     dword ptr fs:[eax], esp
0046597F  EB 01      jmp     short 00465982
00465981  CC         int3
00465982  0000       add     byte ptr [eax], al
00465984  0000       add     byte ptr [eax], al
00465986  0000       add     byte ptr [eax], al
```

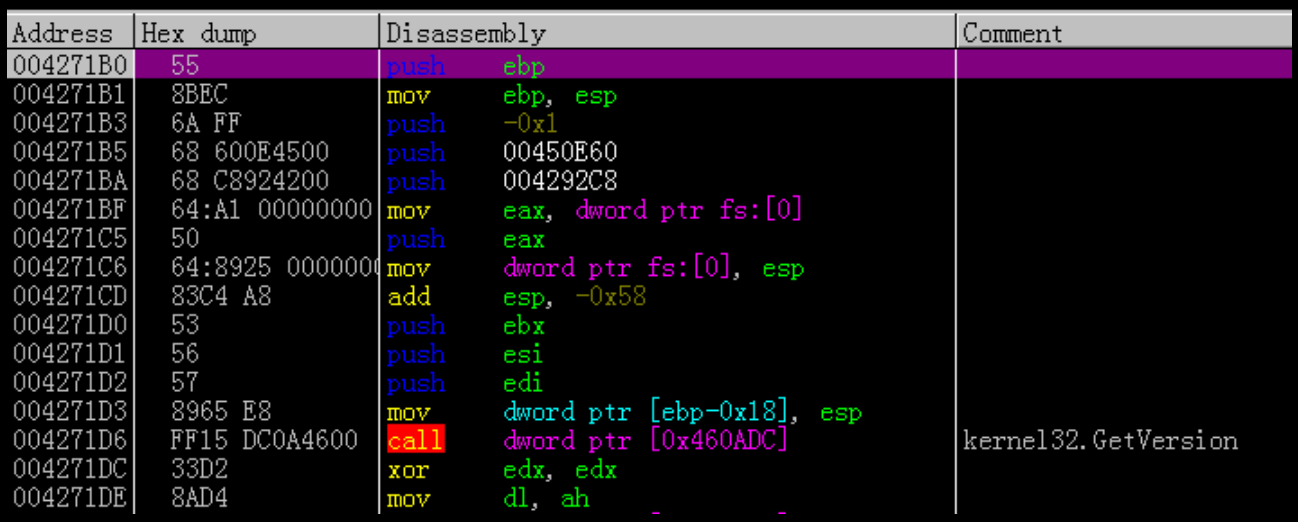
Tại đó, ta thấy lệnh nhảy này sẽ nhảy tới vùng bytes 0, gây ra exception và exception này sẽ được xử lý bởi exception handler đã thiết lập. Để xem danh các SEH, chọn **View > SEH chain**:




Mục tiêu chính của chúng ta là tới OEP, do đó để không làm phức tạp thêm, tôi chuyển qua cửa sổ **Memory** và đặt một **Memory breakpoint on access** tại section đầu tiên:



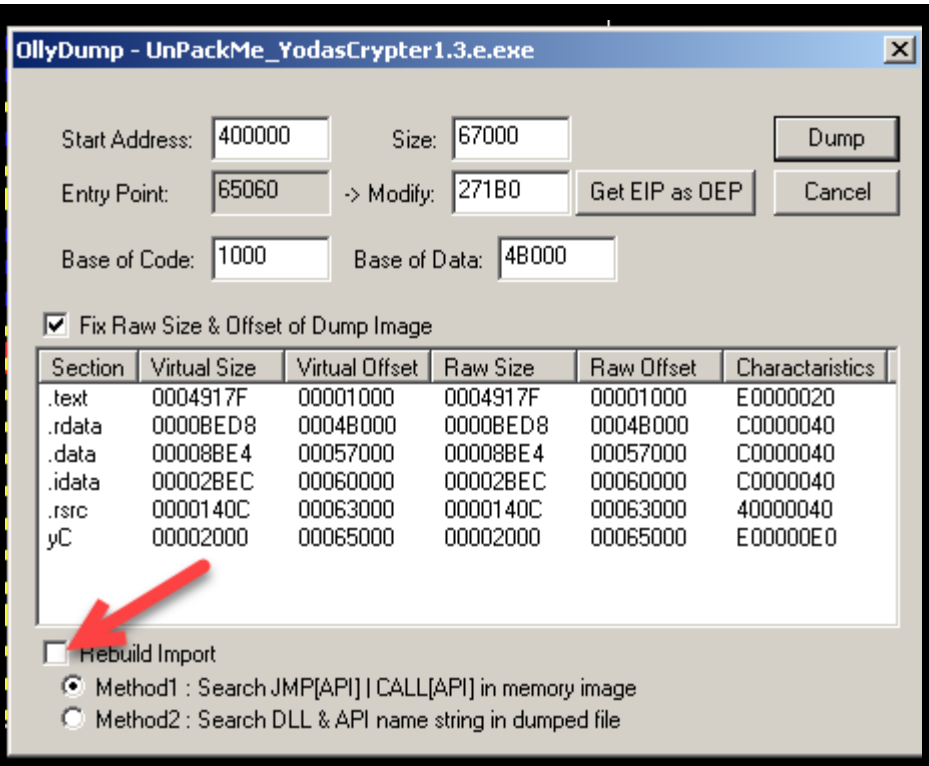

Sau đó, nhấn **F9** để vượt qua exception, ta sẽ đến **OEP**:



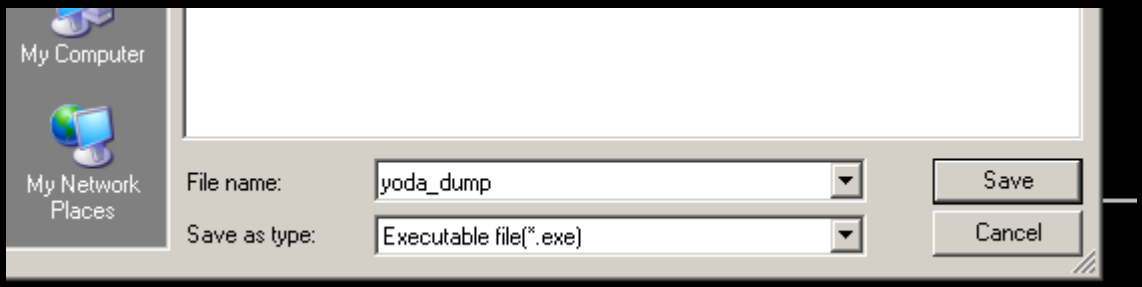

Vẫn là địa chỉ OEP `4271B0` quen thuộc như các phần trước, đơn giản là vì được packed cho cùng một crackme nhằm tiện cho việc thực hành  . Quan sát code bên dưới ta thấy xuất hiện lời gọi tới hàm API `GetVersion()` (*hàm này nếu ở tElock thì đã bị redirect*).

*Ngoài lề: Dành cho những bạn muốn đào sâu hơn một chút thì có thể thấy rằng trong exception handler tại 0046590B, code xử lý ở đó thực hiện thao tác để thay đổi EIP và tại đó địa chỉ của exception là 465982 được ghi đè bằng giá trị OEP hay 4271B0, để trở về trực tiếp từ exception tới OEP.*

Sau khi tới được OEP, tiến hành dump thử bằng OllyDump:




Nhớ bỏ tùy trọn **Rebuild Import** để tránh việc OllyDump tự sửa IAT. Nhấn nút Dump và lưu lại với tên file tùy ý:

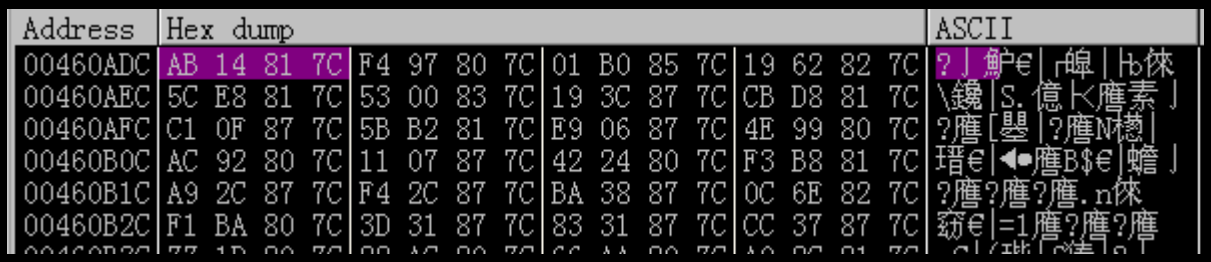



Tất nhiên, khi chúng ta chạy file **yoda_dump.exe** thì sẽ nhận được thông báo lỗi (*còn nếu chạy được thì có thể bạn là người may mắn hơn tôi*  ). Nếu quan sát kĩ ta sẽ thấy rằng IAT cũng đã bị chuyển hướng, có nghĩa là nó không thể thực thi được bình thường khi cố gắng truy cập những địa chỉ không tồn tại trong file dump.

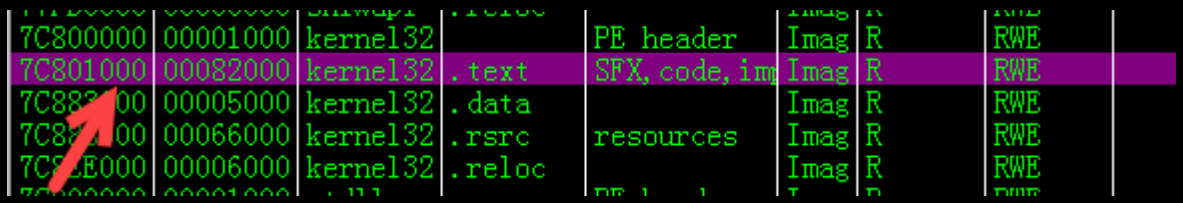
Tiến hành phân tích IAT bằng cách tìm một lệnh call đến một hàm API bất kỳ. Ở đây, tôi chọn luôn lệnh call tới API GetVersion() vốn đã quen thuộc ở các phần trước:

```
004271D1  56            push    esi
004271D2  57            push    edi
004271D3  8965 E8       mov     dword ptr [ebp-0x18], esp
004271D6  FF15 DC0A4600 call    dword ptr [0x460ADC]        kernel32.GetVersion
004271DC  33D2          xor     edx, edx
```

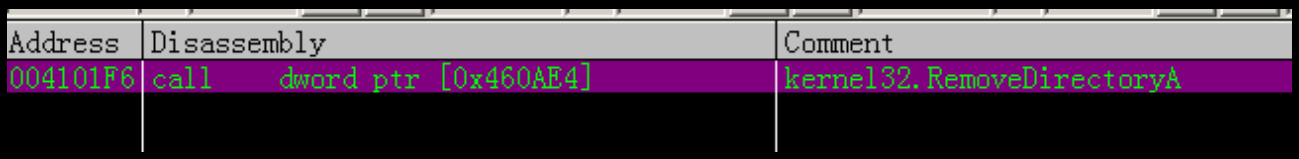
Follow in Dump tại địa chỉ bộ nhớ `0x460ADC`:



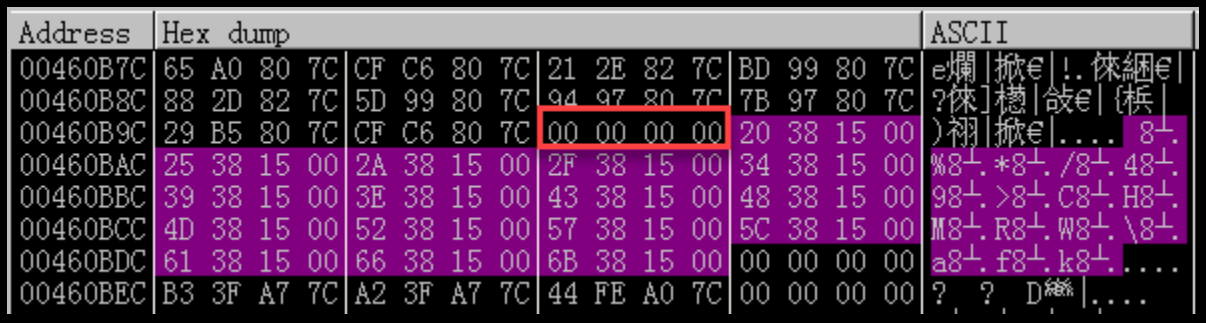
OK, các giá trị này của IAT là địa chỉ chính xác của các hàm API bởi vì tất cả đều có định dạng chung là `7C8XXXXX`. Nếu ta kiểm tra tại cửa sổ **Memory** thì những địa chỉ này nằm trong phần section code của thư viện `kernel32.dll`. Do vậy, ta khẳng định các IAT entry này là chuẩn.



Một cách khác để kiểm chứng, tìm kiếm lệnh tham chiếu tới bất kỳ một giá trị ngẫu nhiên bằng cách nhấn chuột phải và chọn **Find references (Ctrl+R)**.



Tiếp tục cuộn chuột xuống dưới để tìm dấu hiệu phân tách giữa các import:

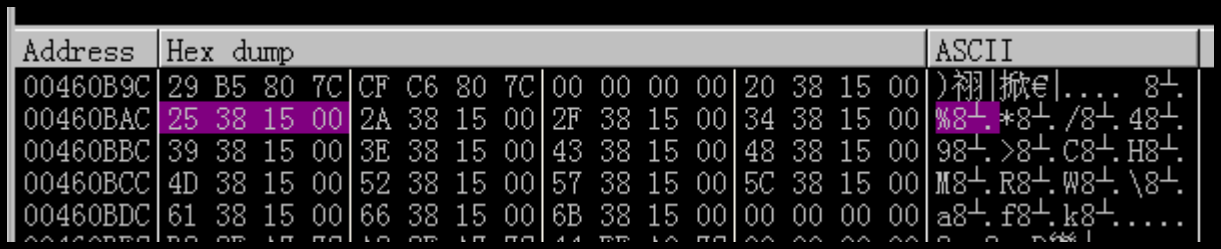


Tôi thấy rằng nhóm IAT tiếp theo được chuyển tới một vùng nhớ có định dạng `15XXXX`. Ta xem thử ở đó có thông tin gì:

Theo như trên hình, ta thấy đây là một section không thuộc dll nào, do vậy chắc chắn section này đã được tạo ra bởi packer. Khởi động lại OllyDbg để kiểm tra section này:

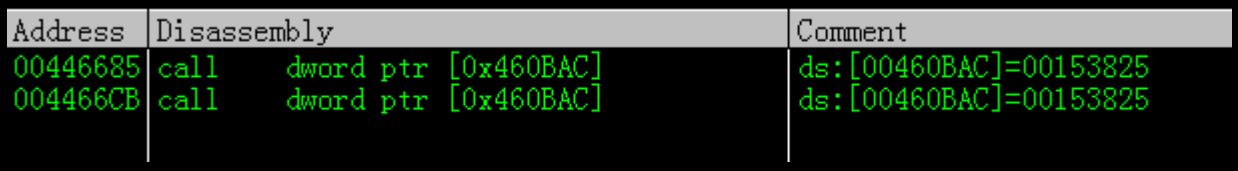
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 00140000 | 00001000 | | | | Priv | RW | RW |
| 00150000 | 00004000 | | | | Priv | RW | RW |
| 00250000 | 00006000 | | | | Priv | RW | RW |
| 00260000 | 00003000 | | | | Map | RW | RW |

Chúng ta thấy rằng ban đầu section này được tạo ra bởi hệ thống và nó có kích thước là 4000 bytes (trên máy tôi), nhưng sau đó nó được sử dụng bởi packer và packer đã mở rộng section này lên 28000 bytes (trên máy tôi) để sử dụng cho mục đích riêng của mình. Vì vậy, có thể đây là những IAT entry đã bị chuyển hướng bởi packer tới section đó. Ta kiểm tra thử một entry để xem nó hoạt động như thế nào. Chọn 4 bytes tại địa chỉ 0x460BAC:



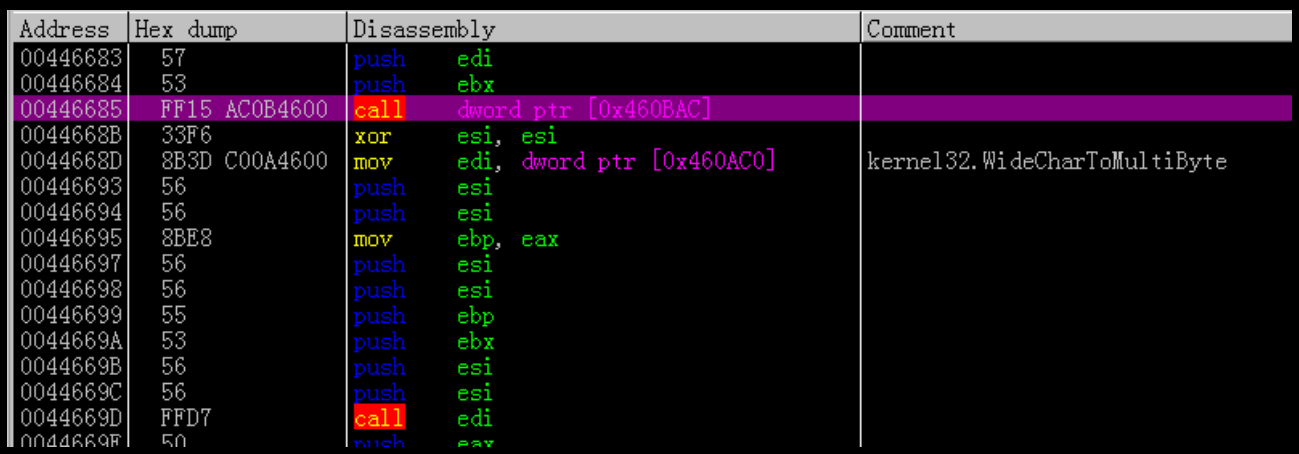
| Address | Hex dump | | | | | | | | | | | | ASCII |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00460B9C | 29 | B5 | 80 | 7C | CF | C6 | 80 | 7C | 00 | 00 | 00 | 00 | 20 | 38 | 15 | 00 | )袈\|掀€\|.... 8⊥. |
| 00460BAC | 25 | 38 | 15 | 00 | 2A | 38 | 15 | 00 | 2F | 38 | 15 | 00 | 34 | 38 | 15 | 00 | %8⊥.*8⊥./8⊥.48⊥. |
| 00460BBC | 39 | 38 | 15 | 00 | 3E | 38 | 15 | 00 | 43 | 38 | 15 | 00 | 48 | 38 | 15 | 00 | 98⊥.>8⊥.C8⊥.H8⊥. |
| 00460BCC | 4D | 38 | 15 | 00 | 52 | 38 | 15 | 00 | 57 | 38 | 15 | 00 | 5C | 38 | 15 | 00 | M8⊥.R8⊥.W8⊥.\8⊥. |
| 00460BDC | 61 | 38 | 15 | 00 | 66 | 38 | 15 | 00 | 6B | 38 | 15 | 00 | 00 | 00 | 00 | 00 | a8⊥.f8⊥.k8⊥..... |

Nhấn chuột phải tại 4 bytes đã chọn và chọn **Find references**. Kết quả như sau:



| Address | Disassembly | Comment |
|---|---|---|
| 00446685 | call dword ptr [0x460BAC] | ds:[00460BAC]=00153825 |
| 004466CB | call dword ptr [0x460BAC] | ds:[00460BAC]=00153825 |

Ta thấy trên hình xuất hiện hai lệnh Call nhận giá trị tại địa chỉ ta đã chọn. Tới lệnh call đầu tiên bằng cách nhấp đúp vào nó, ta sẽ tới vùng code sau:



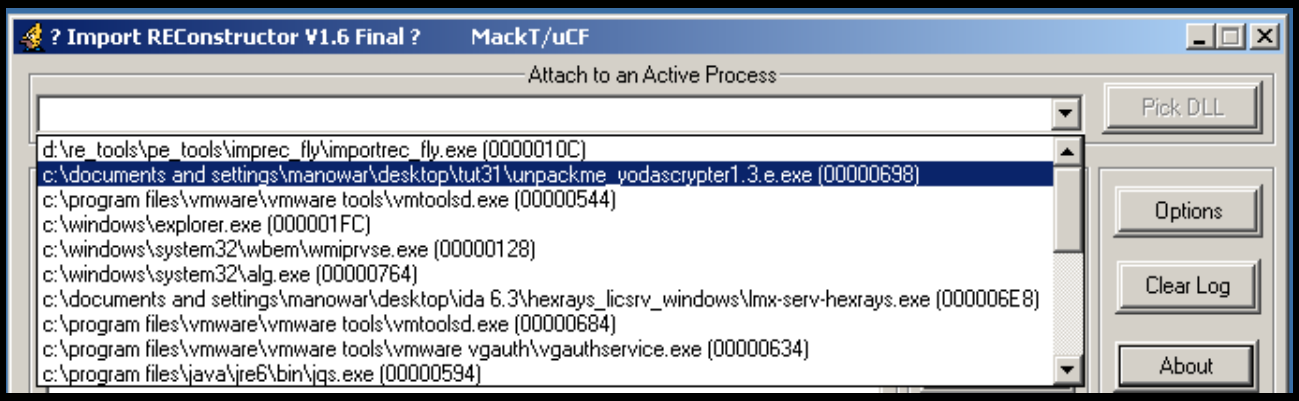
| Address | Hex dump | Disassembly | Comment |
|---|---|---|---|
| 00446683 | 57 | push edi | |
| 00446684 | 53 | push ebx | |
| 00446685 | FF15 AC0B4600 | call dword ptr [0x460BAC] | |
| 0044668B | 33F6 | xor esi, esi | |
| 0044668D | 8B3D C00A4600 | mov edi, dword ptr [0x460AC0] | kernel32.WideCharToMultiByte |
| 00446693 | 56 | push esi | |
| 00446694 | 56 | push esi | |
| 00446695 | 8BE8 | mov ebp, eax | |
| 00446697 | 56 | push esi | |
| 00446698 | 56 | push esi | |
| 00446699 | 55 | push ebp | |
| 0044669A | 53 | push ebx | |
| 0044669B | 56 | push esi | |
| 0044669C | 56 | push esi | |
| 0044669D | FFD7 | call edi | |
| 0044669F | 50 | push eax | |

Ta đang dừng lại tại lệnh CALL, nhấn chuột phải và chọn **Follow (Enter)** để xem code của lệnh Call này sẽ làm những gì trong section được chuyển hướng.

```
Address   Hex dump          Disassembly                          Comment
00153825  - E9 1114FD76     jmp     oleaut32.SysStringLen
0015382A  - E9 656DFF76     jmp     oleaut32.OleCreateFontIndirect
0015382F  - E9 2513FD76     jmp     oleaut32.SysAllocStringLen
00153834  - E9 4916FD76     jmp     oleaut32.SafeArrayGetDim
00153839  - E9 5A9CFF76     jmp     oleaut32.SafeArrayGetElemsize
0015383E  - E9 5818FD76     jmp     oleaut32.SafeArrayGetLBound
00153843  - E9 0718FD76     jmp     oleaut32.SafeArrayGetUBound
00153848  - E9 C317FD76     jmp     oleaut32.SafeArrayAccessData
0015384D  - E9 ED17FD76     jmp     oleaut32.SafeArrayUnaccessData
00153852  - E9 822EFD76     jmp     oleaut32.VariantChangeType
00153857  - E9 F40FFD76     jmp     oleaut32.SysFreeString
0015385C  - E9 F413FD76     jmp     oleaut32.SysAllocStringByteLen
00153861  - E9 5C13FD76     jmp     oleaut32.SysAllocString
00153866  - E9 2A9AFF76     jmp     oleaut32.VariantCopy
0015386B  - E9 10250377     jmp     oleaut32.OleLoadPicture
00153870    0000            add     byte ptr [eax], al
```

Ta thấy rằng, tại đây là một lệnh nhảy trực tiếp tới hàm API `SysStringLen()` thuộc `oelaut32.dll`. Dưới đó cũng là các lệnh nhảy trực tiếp tới các hàm API tương ứng như trên hình. Nếu ta follow tại từng lệnh này ta sẽ tới được các hàm API mà unpackme sử dụng.

Như vậy, quá trình thực hiện chuyển hướng dường như là rất đơn giản. Liệu rằng nếu dùng ImpREC với các tính năng trace code thì nó có fix file dump mà không cần phải tìm ra magic jump hay không? Mở ImpREC và attach process của unpackme:

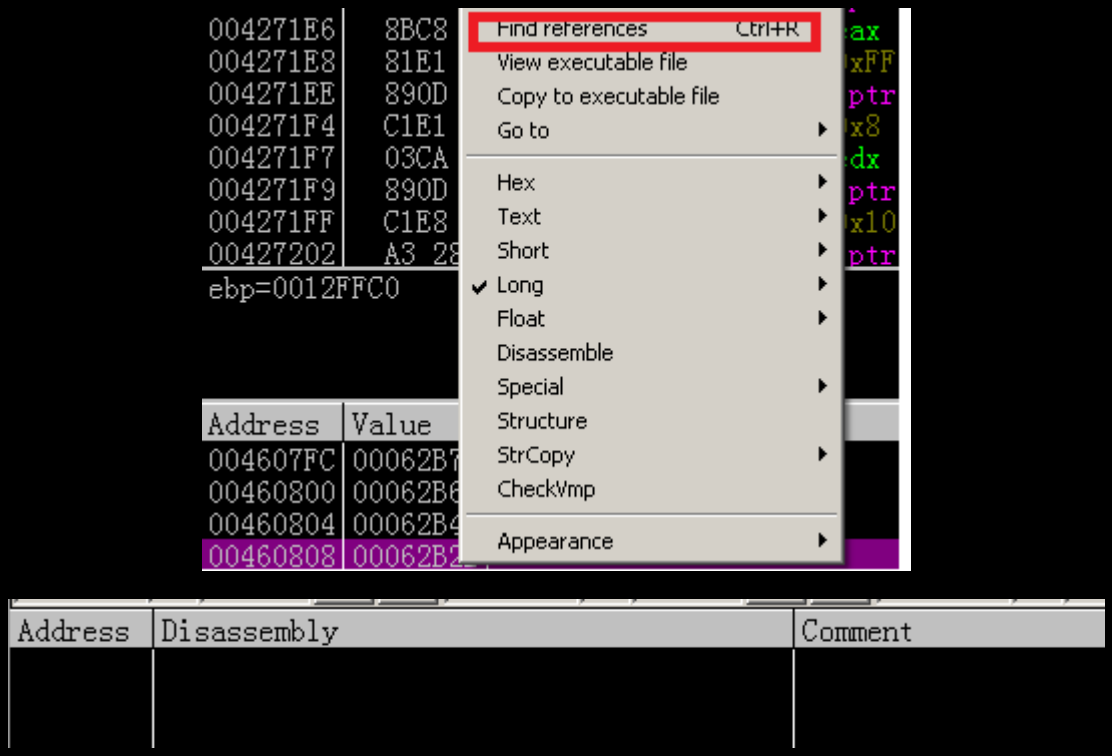



Đừng quên để ImpREC có thể fix được file ta phải cung cấp cho nó 3 thông tin là `OEP, IAT START và IAT Size`. Ta đã có được OEP là `4271B0`, chuyển đổi sang RVA là `271B0`. Giờ tìm IAT Start, cách tìm tôi sẽ không nhắc lại nữa.

Khi quan sát tại cửa sổ Dump, ta nhận thấy các IAT entry tại đây: một là địa chỉ của hàm API cụ thể, hai là được chuyển hướng tới section `15xxxxx`. Do đó, tìm ngược lên trên theo dấu hiệu này cho tới khi thấy có các giá trị khác:

| Address | Value | Comment |
|---|---|---|
| 004607FC | 00062B78 | |
| 00460800 | 00062B60 | |
| 00460804 | 00062B4C | |
| 00460808 | 00062B2E | |
| 0046080C | 00000000 | |
| 00460810 | 80000008 | |
| 00460814 | 00000000 | |
| 00460818 | 001537B2 | |
| 0046081C | 001537B7 | |
| 00460820 | 001537BC | |
| 00460824 | 001537C1 | |
| 00460828 | 001537C6 | |

Chọn một địa chỉ bất kỳ và thực hiện tìm kiếm, kết quả sẽ không thấy có lệnh Call nào:




Do đó, IAT Start của chúng ta sẽ là `460818`, sau khi trừ đi giá trị **ImageBase**, ta có giá trị `RVA = 60818`.

Thực hiện tương tự để tìm IAT End:

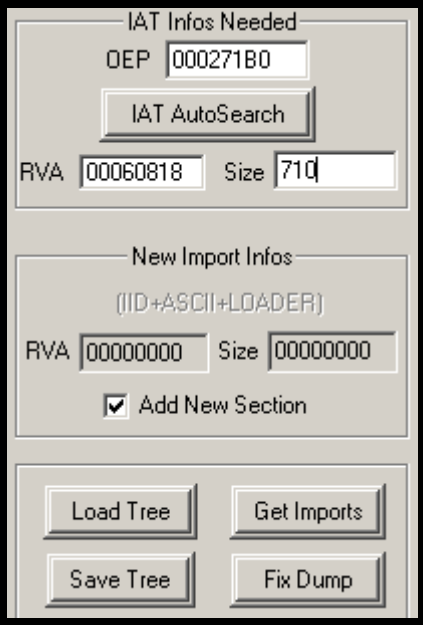
| Address | Value | Comment |
|---|---|---|
| 00460F10 | 0015380C | |
| 00460F14 | 00153811 | |
| 00460F18 | 00153816 | |
| 00460F1C | 0015381B | |
| 00460F20 | 00000000 | |
| 00460F24 | 001537CB | |
| 00460F28 | 00000000 | |
| 00460F2C | 6C50000B | |
| 00460F30 | 6F537961 | |
| 00460F34 | 41646E75 | |

Thông tin về IAT Start và IAT End đã có, giờ tính IAT Size theo công thức: `Size = IAT End – IAT Start = 460f28 – 460818 = 710`

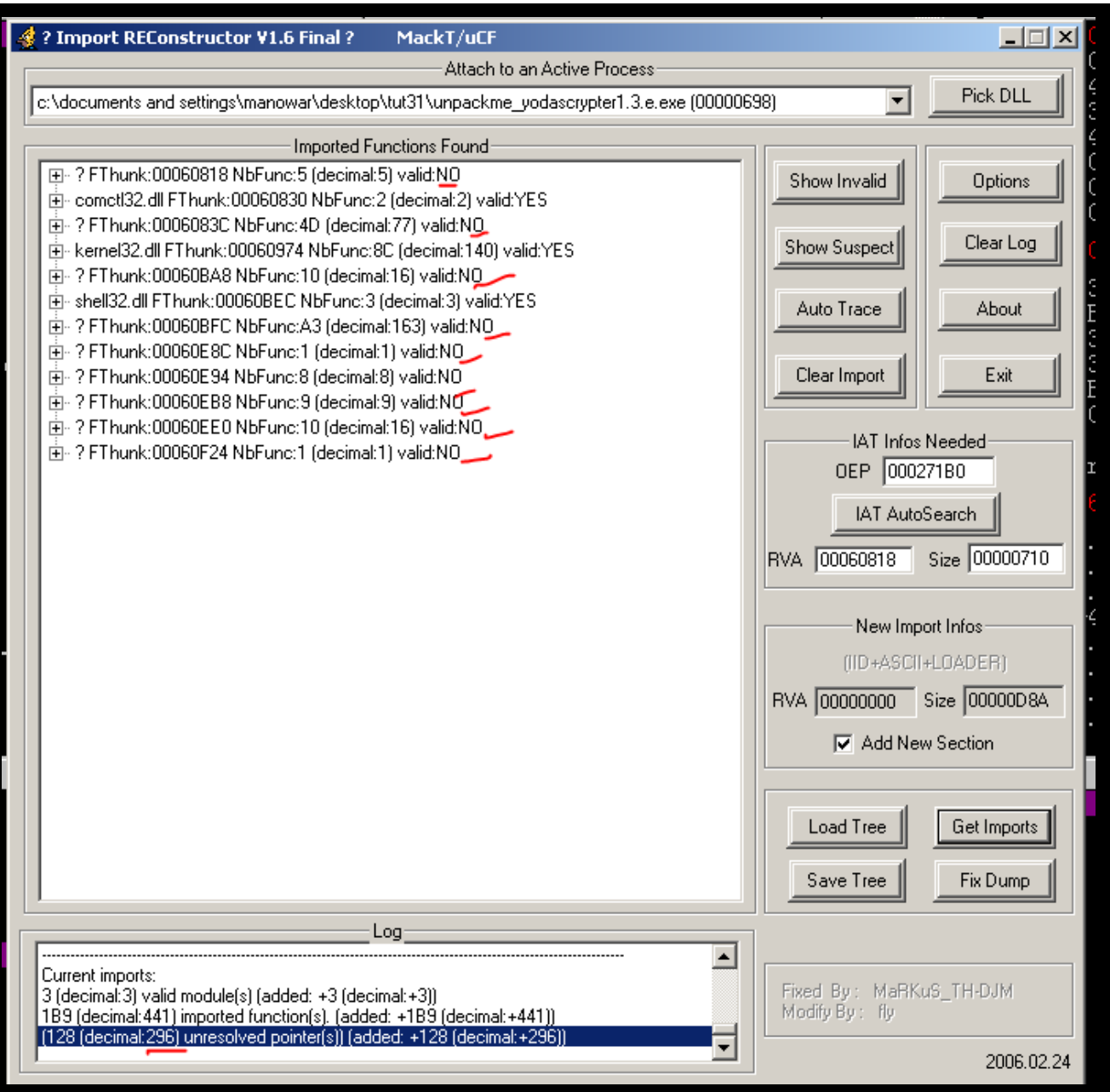

Tổng hợp lại ta có:

- OEP= 271B0
- RVA = 60818
- SIZE = 710

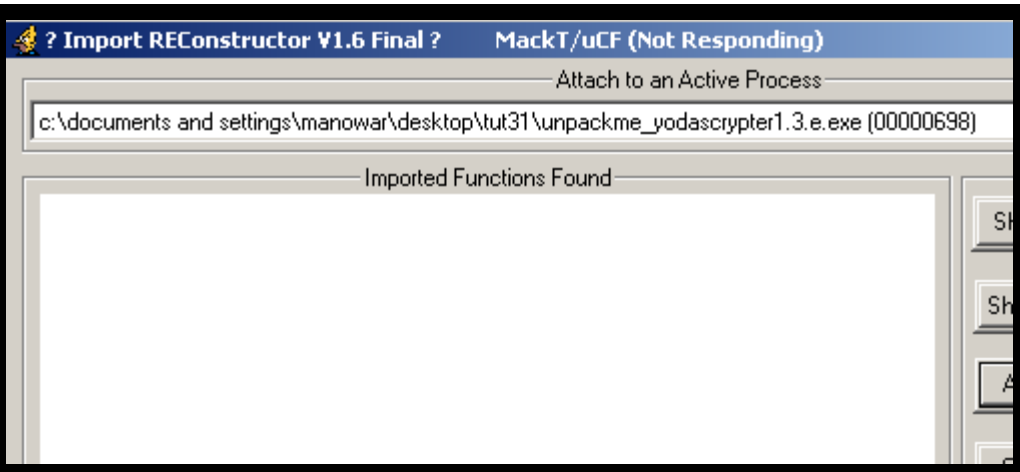
Điền các giá trị trên vào phần **IAT Infos Needed:**



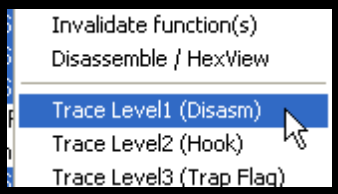

Sau đó nhấn **Get Imports**:

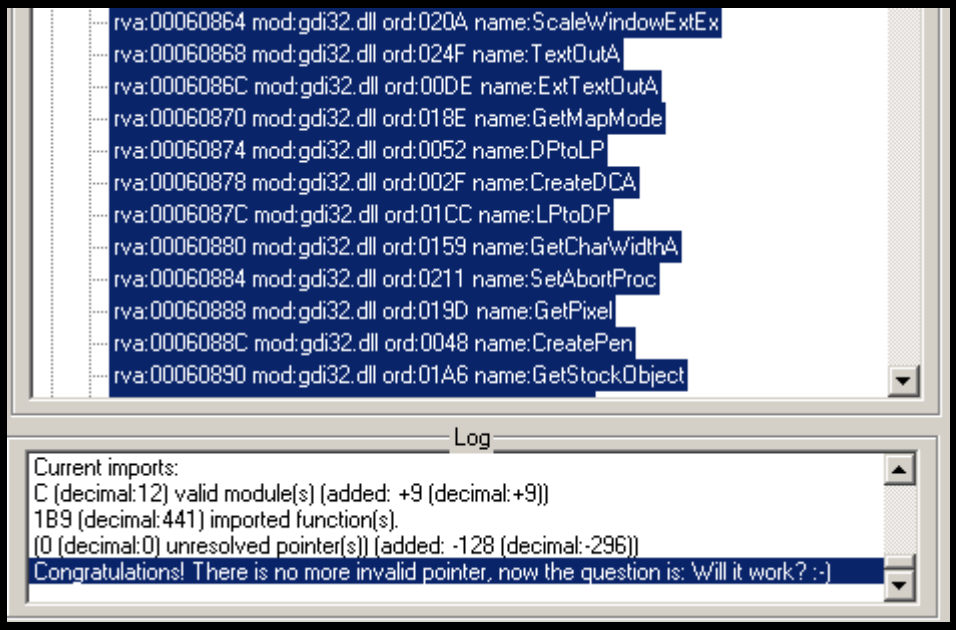



Ta thấy rằng có 296 entry chưa xử lý được. Thử dùng mấy tính năng Trace của ImpREC xem có làm được gì không? Đầu tiên, thử với **Auto Trace** thì ImpREC treo cứng luôn:

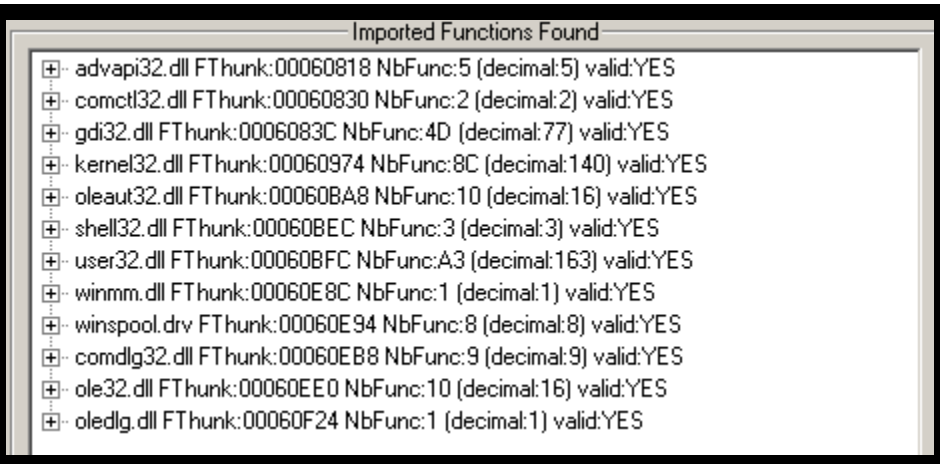



Khởi động lại ImpREC, nhấn Show Invalids, chuột phải và chọn **Trace Level 1**:

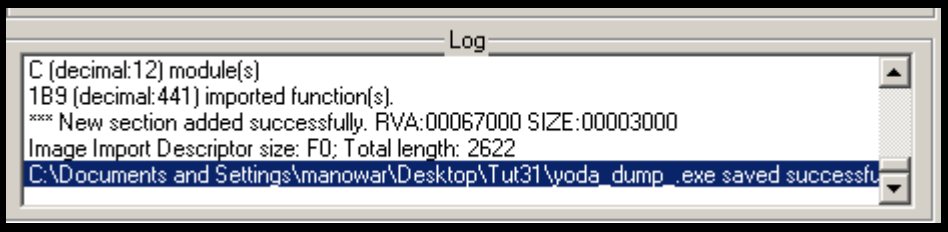


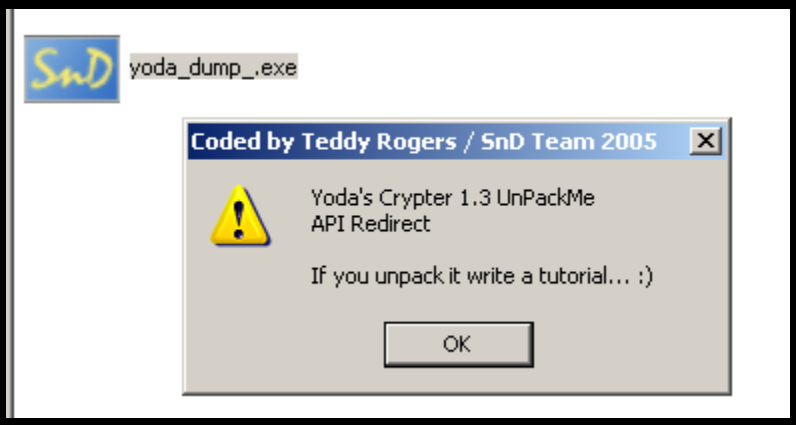

ImpREC thông báo đã lấy được toàn bộ bảng IAT và không còn bất kỳ một invalid entry nào nữa. Tin tưởng được không, lolz? Nhấn Show Invalid một lần nữa (*hơi thừa, nhưng để chắc chắn*), ta thấy toàn bộ đều là **YES**.




Quá đẹp! Giờ nhấn Fix Dump để ImpREC tiến hành fix file **yoda_dump.exe**:

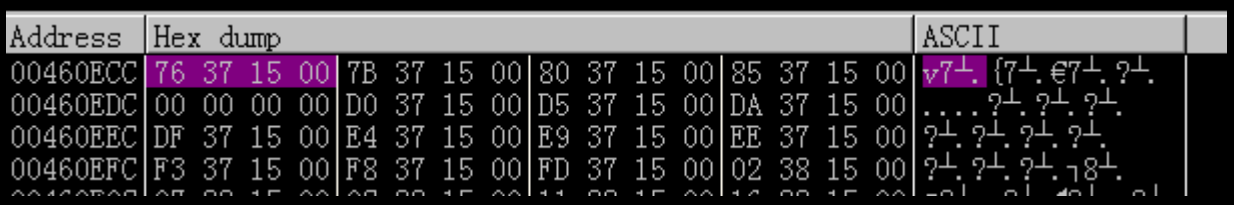



ImpREC thực hiện quá trình fix file và tạo một file mới là **yoda_dump_.exe**. Chạy thử file đã fix:

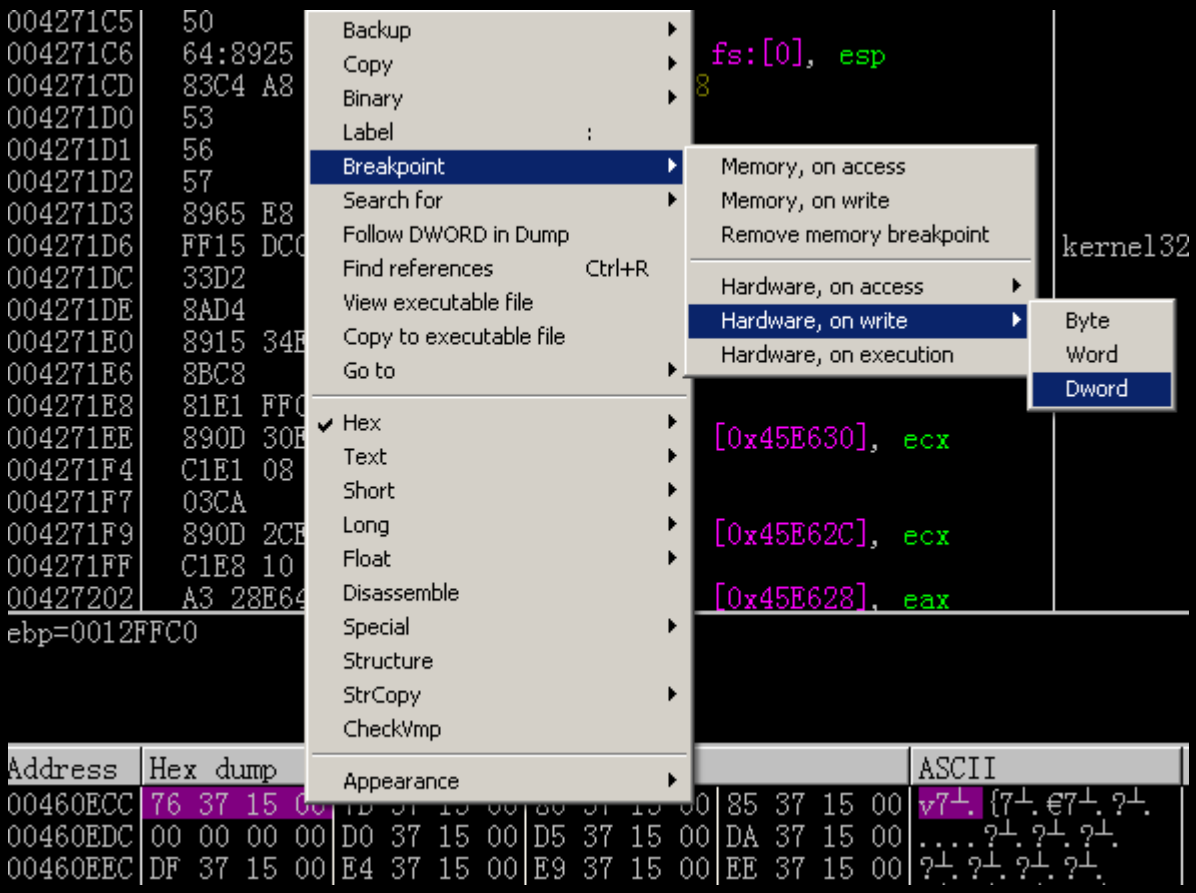



Hehe với unpackme này tính năng Trace của IMP REC đã làm việc quá tốt, giúp ta tiết kiệm rất nhiều thời gian. Ngoài việc áp dụng cách như trên, tôi vẫn đặt câu hỏi liệu packer này có sử dụng **magic jump** hay không?

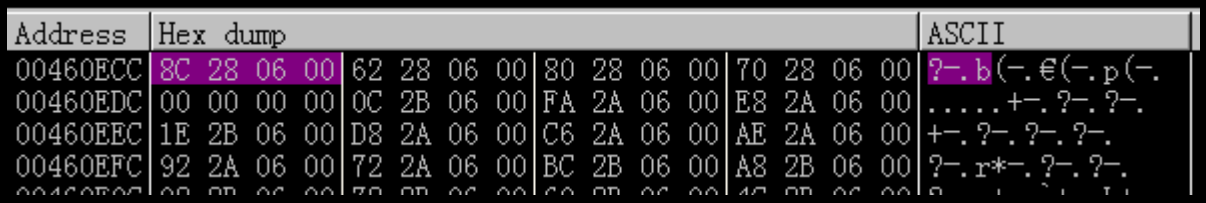
Tiến hành thực hiện từ đầu và tới OEP, tìm kiếm một bad entry trong IAT:



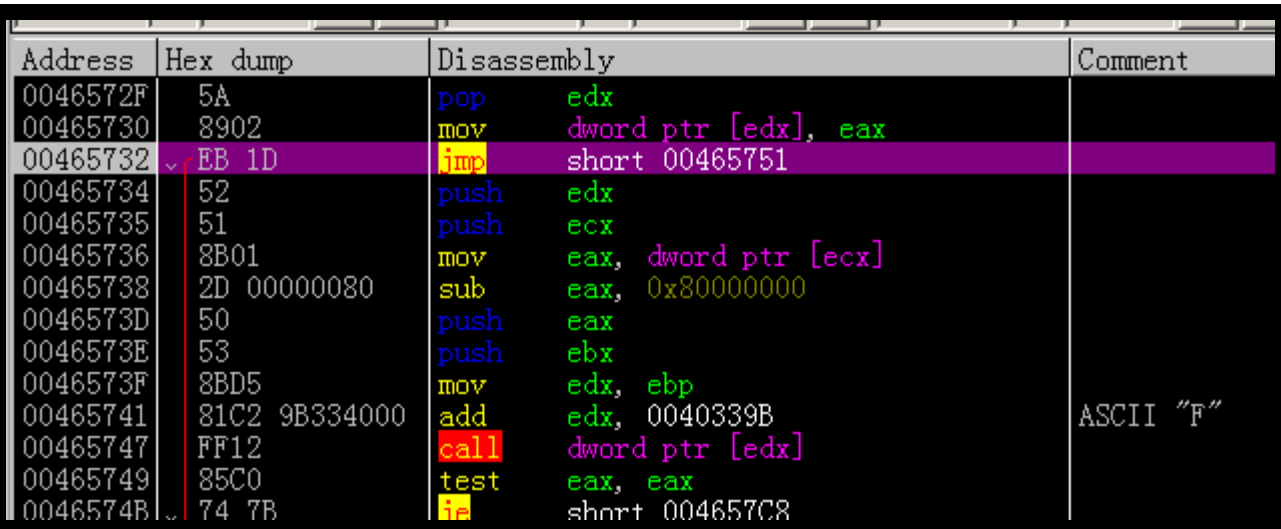
Đặt một Breakpoint: **Hardware, on write > Dword** tại 4 bytes đã được đánh dấu ở trên trước khi khởi động lại OllyDbg:



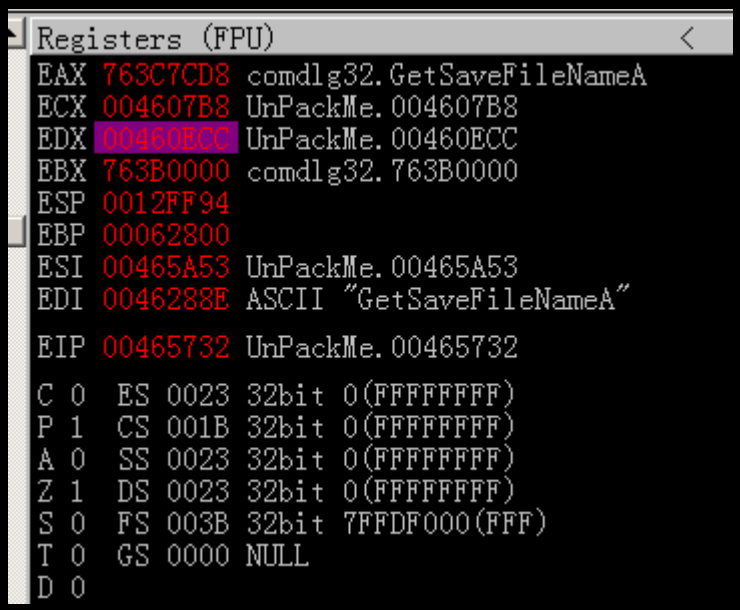

Sau đó khởi động lại OllyDbg, tại cửa số Dump, nhấn **Ctrl+G** tới địa chỉ `460ECC` để kiểm tra xem giá trị ban đầu của nó là gì:

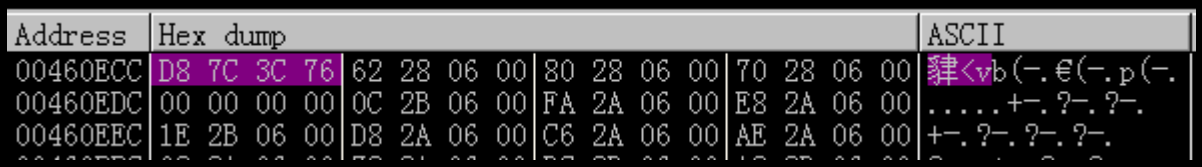



Nhấn **F9** để RUN, ta dừng lại tại đây:




Như trên hình, ta thấy có lệnh tại `00465730 8902 mov dword ptr [edx], eax`. Quan sát cửa sổ Registers, thanh ghi **EDX** đang lưu địa chỉ `00460ECC`, thanh ghi EAX đang lưu địa chỉ của một hàm API. Đó là lý do vì sao OllyDbg dừng lại tại vùng code này:




Ta thấy địa chỉ của hàm API được lưu vào vùng nhớ trỏ bởi EDX, điều đó có nghĩa là trong trường hợp này, lần đầu tiên sẽ lưu địa chỉ chính xác của hàm API và sau đó sửa đổi nó bằng giá trị khác (tôi hay gọi là bad entry). Tiếp tục theo dõi để xem khi nào điều này xảy ra.

```
Address   Hex dump          Disassembly                          Comment
00465791    81C1 79344000   add     ecx, 00403479
00465797    8D39            lea     edi, dword ptr [ecx]
00465799    3E:8B77 04      mov     esi, dword ptr [edi+0x4]
0046579D    8932            mov     dword ptr [edx], esi
0046579F    2BC6            sub     eax, esi
004657A1    83E8 05         sub     eax, 0x5
004657A4    C606 E9         mov     byte ptr [esi], 0xE9
```

Tại vùng code như trên hình, quan sát cửa số Register ta thấy **ESI** đang lưu giá trị xấu và sẽ ghi đè lên địa chỉ của hàm API đã được lưu trước đó:

```
Registers (FPU)                                    <
EAX 763C7CD8 comdlg32.GetSaveFileNameA
ECX 00465C79 UnPackMe.00465C79
EDX 00460ECC UnPackMe.00460ECC
EBX 763B0000 comdlg32.763B0000
ESP 0012FF88 ASCII "SZF"
EBP 00062800
ESI 00153776
EDI 00465C79 UnPackMe.00465C79

EIP 0046579F UnPackMe.0046579F

C 0  ES 0023 32bit 0(FFFFFFFF)
P 0  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
```

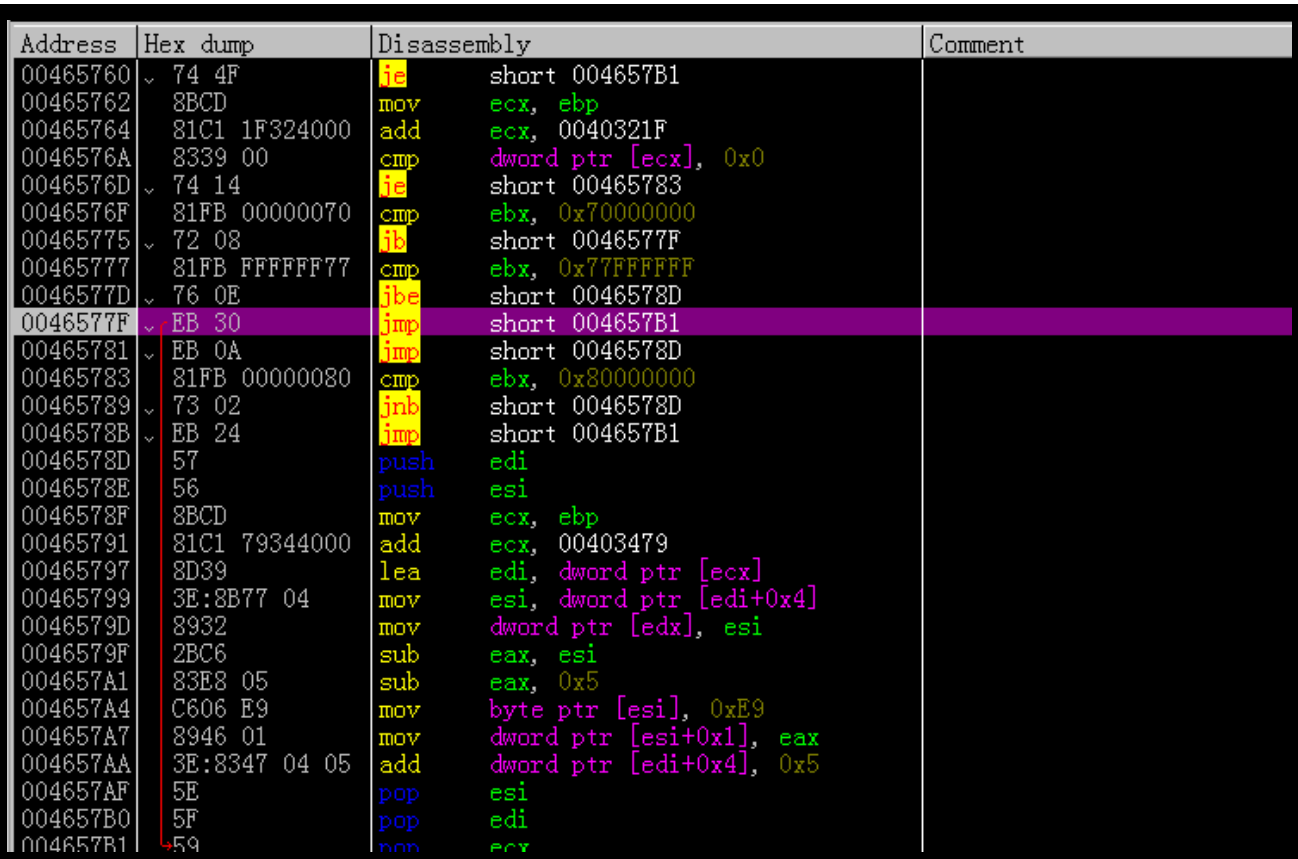
OK nếu tiếp tục thì unpackme sẽ thực thi một cách bình thường. Giờ ta tiến hành thực hiện lại công việc trên nhưng với một hàm API chuẩn. Tôi vẫn lựa chọn hàm API là `GetVersion()`. Follow in Dump tại `0x460ADC` và đặt một HW BP on write tại đó. Sau đó, khởi động lại OllyDbg, nhấn **F9** để run ta sẽ dừng tại đây:

```
Address   Hex dump          Disassembly                          Comment
00465729  E9 59020000       jmp     00465987
0046572E  61                popad
0046572F  5A                pop     edx
00465730  8902              mov     dword ptr [edx], eax
00465732  EB 1D             jmp     short 00465751
00465734  52                push    edx
00465735  51                push    ecx
00465736  8B01              mov     eax, dword ptr [ecx]
00465738  2D 00000080       sub     eax, 0x80000000
0046573D  50                push    eax
0046573E  53                push    ebx
0046573F  8BD5              mov     edx, ebp
00465741  81C2 9B334000     add     edx, 0040339B                ASCII "F"
00465747  FF12              call    dword ptr [edx]
00465749  85C0              test    eax, eax
0046574B  74 7B             je      short 004657C8
0046574D  59                pop     ecx
0046574E  5A                pop     edx
0046574F  8902              mov     dword ptr [edx], eax
00465751  51                push    ecx
00465752  8BCD              mov     ecx, ebp
```

Tương tự, quan sát tại cửa sổ Registers ta thấy gì nào:

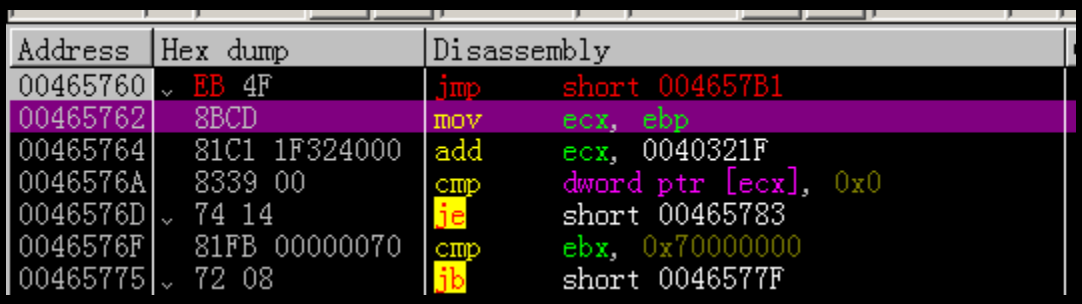
```
Registers (FPU)
EAX 7C8114AB kernel32.GetVersion
ECX 004603C8 UnPackMe.004603C8
EDX 00460ADC UnPackMe.00460ADC
EBX 7C800000 kernel32.7C800000
ESP 0012FF94
EBP 00062800
ESI 00465A2F UnPackMe.00465A2F
EDI 004612A2 ASCII "GetVersion"

EIP 00465732 UnPackMe.00465732

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
```

Thanh ghi EAX đang có địa chỉ của hàm API GetVersion(). Tiếp tục trace xuống một chút ta tới đây:

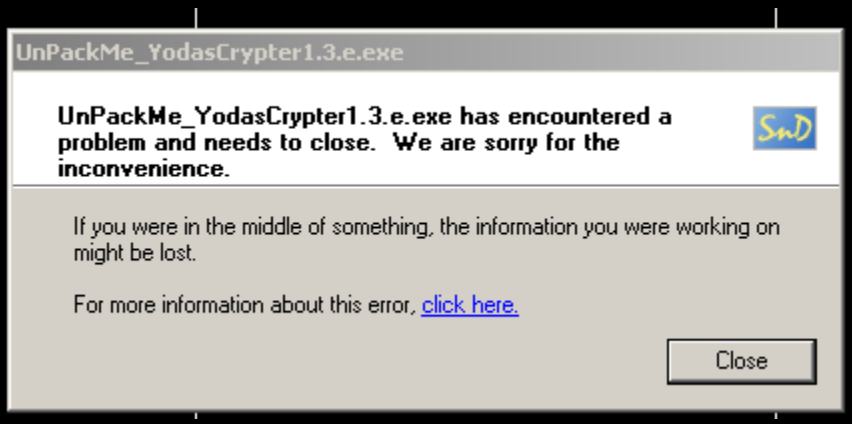



Ta dừng ở lệnh JMP tại địa chỉ `0046577F`. Lệnh JMP này sẽ nhảy tránh qua vùng code ở bên dưới (đánh dấu bằng mũi tên đỏ), mà vùng code bên dưới lại chính là nơi thực hiện thay đổi địa chỉ API chuẩn bằng giá trị xấu (bad entry). Vì vậy, mục tiêu là tìm cách sao cho đối với bad entry thì ta cũng cho nhảy để tránh vùng code bên dưới luôn. Trước lệnh JMP ta thấy có rất nhiều nhảy có điều kiện, lựa chọn một lệnh nhảy phù hợp, đó chính là **magic jump** cần tìm.

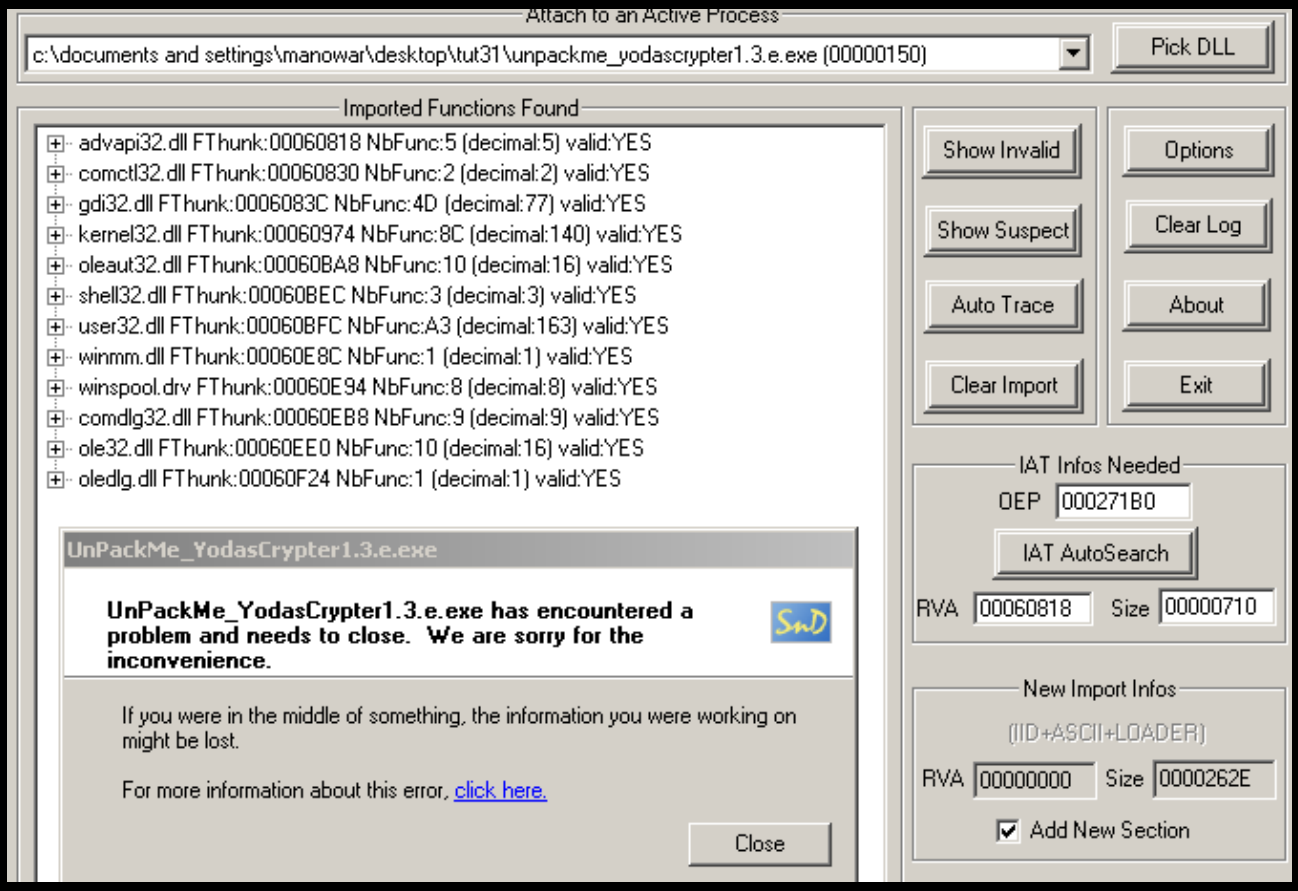
Sau khi quan sát, tôi lựa chọn lệnh nhảy tại địa chỉ `00465760 /74 4F je short 004657B1`. Đặt một BP HW on execution tại địa chỉ này. Sau đó, khởi động lại OllyDbg, nhấn **F9** ta sẽ dừng lại tại lệnh trên. Patch lệnh này thành lệnh `JMP` luôn để cho dù có thế nào thì ta cũng bypass luôn chỗ ghi đè bad entry lên địa chỉ API chuẩn:



Sau đó bỏ HW bp tại đây đi và nhấn **F9** để run. Tuy nhiên, unpackme sau khi ta patch như trên sẽ bị crash khi run trong OllyDbg:

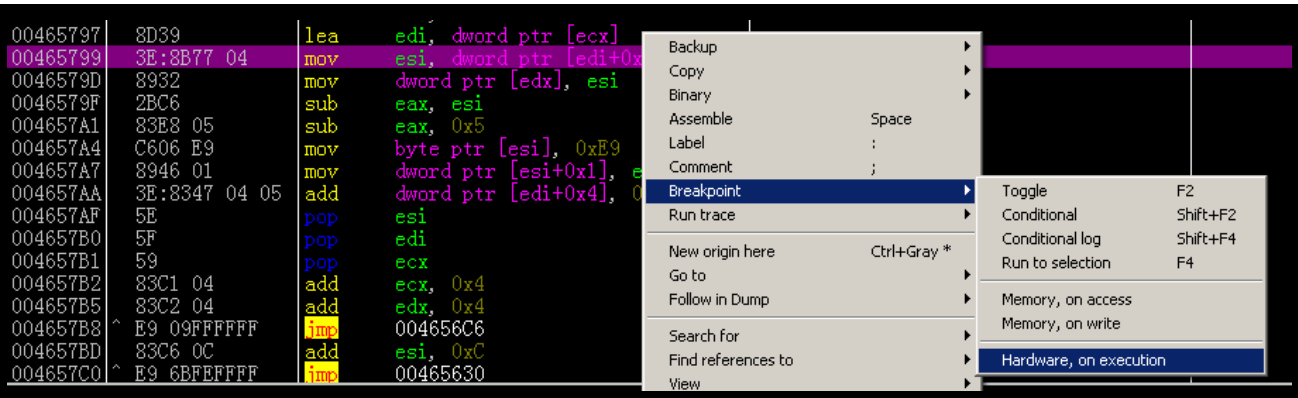



Kệ nó thôi, giữ nguyên như thế. Mở ImpREC, điền các thông tin cần thiết và nhấn **Get Imports**, ta sẽ có được bảng IAT hoàn chỉnh dùng để fix dump:

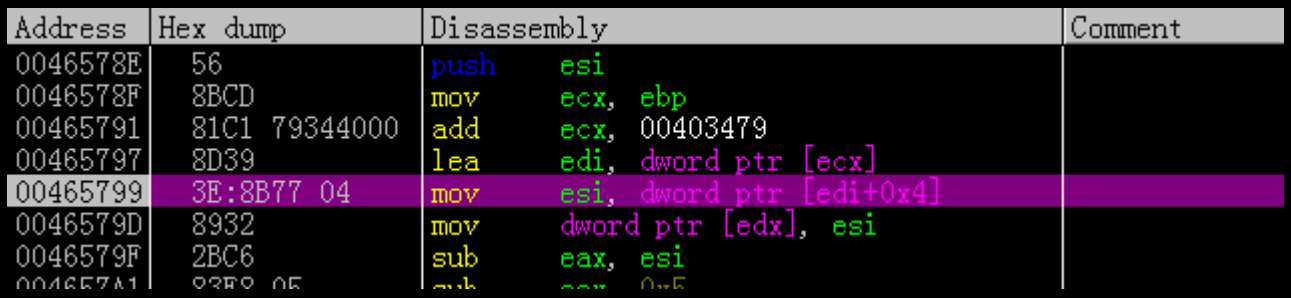



Với IAT như trên thì ta hoàn toàn có thể fix dump được rồi.

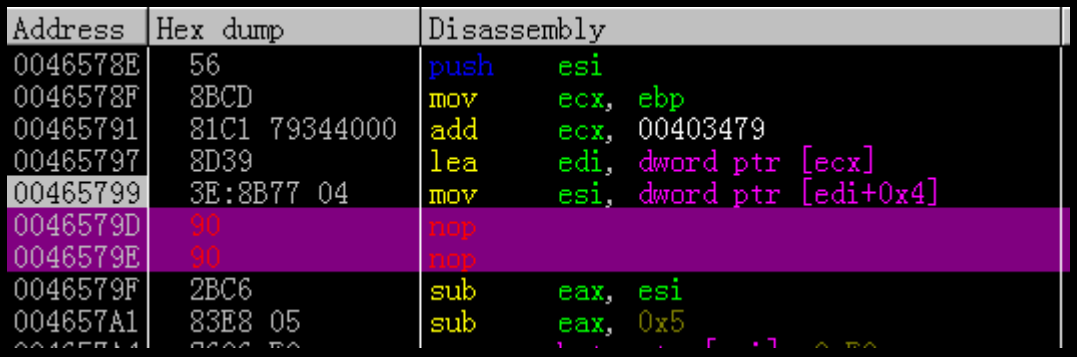
Ngoài cách fix bằng **magic jump** ở trên, ta có thể thực hiện một cách khác nữa như sau. Tôi tìm tới lệnh trước lệnh thực hiện việc lưu bad entry và đặt một HBP on execution tại đây:



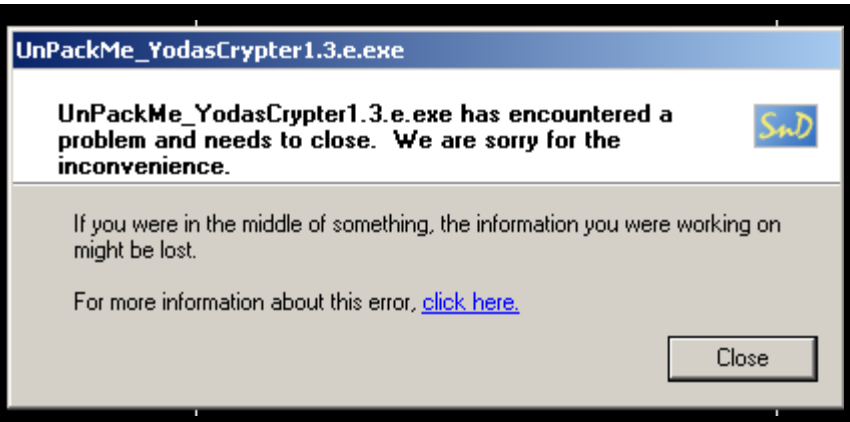

Sau đó, khởi động lại OllyDbg và thực thi bằng **F9** cho tới khi dừng sự thực thi tại câu lệnh này:



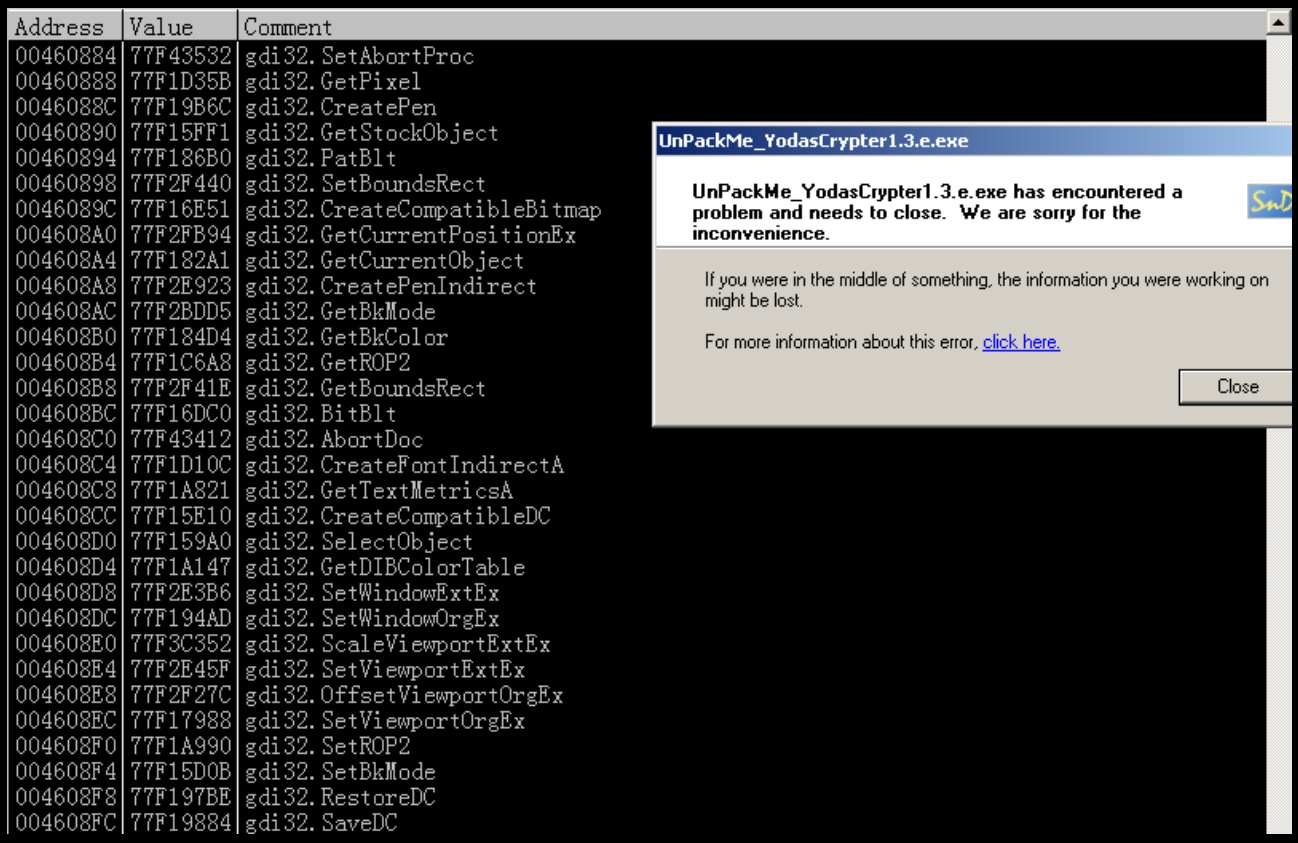

Như đã biết, câu lệnh bên dưới `0046579D 8932  mov dword ptr [edx], esi` chính là lệnh thực hiện lưu giá trị bad entry. Giờ tôi không muốn cho lưu nữa bằng cách là `nop` luôn lênh này:




Sau khi patch như trên, bỏ HW bp đi và đến OEP xem thế nào. Cũng bị crash như phương pháp **magic jump** ở trên:




Oh! Damn it… vậy là trình packer này có áp dụng phương pháp nào đó để phát hiện việc ta can thiệp patch chương trình. Nhưng mà crash thì cũng kệ thôi vì mục tiêu cuối cùng là đã có được bảng IAT full rồi:



| Address | Value | Comment |
|---|---|---|
| 00460884 | 77F43532 | gdi32.SetAbortProc |
| 00460888 | 77F1D35B | gdi32.GetPixel |
| 0046088C | 77F19B6C | gdi32.CreatePen |
| 00460890 | 77F15FF1 | gdi32.GetStockObject |
| 00460894 | 77F186B0 | gdi32.PatBlt |
| 00460898 | 77F2F440 | gdi32.SetBoundsRect |
| 0046089C | 77F16E51 | gdi32.CreateCompatibleBitmap |
| 004608A0 | 77F2FB94 | gdi32.GetCurrentPositionEx |
| 004608A4 | 77F182A1 | gdi32.GetCurrentObject |
| 004608A8 | 77F2E923 | gdi32.CreatePenIndirect |
| 004608AC | 77F2BDD5 | gdi32.GetBkMode |
| 004608B0 | 77F184D4 | gdi32.GetBkColor |
| 004608B4 | 77F1C6A8 | gdi32.GetROP2 |
| 004608B8 | 77F2F41E | gdi32.GetBoundsRect |
| 004608BC | 77F16DC0 | gdi32.BitBlt |
| 004608C0 | 77F43412 | gdi32.AbortDoc |
| 004608C4 | 77F1D10C | gdi32.CreateFontIndirectA |
| 004608C8 | 77F1A821 | gdi32.GetTextMetricsA |
| 004608CC | 77F15E10 | gdi32.CreateCompatibleDC |
| 004608D0 | 77F159A0 | gdi32.SelectObject |
| 004608D4 | 77F1A147 | gdi32.GetDIBColorTable |
| 004608D8 | 77F2E3B6 | gdi32.SetWindowExtEx |
| 004608DC | 77F194AD | gdi32.SetWindowOrgEx |
| 004608E0 | 77F3C352 | gdi32.ScaleViewportExtEx |
| 004608E4 | 77F2E45F | gdi32.SetViewportExtEx |
| 004608E8 | 77F2F27C | gdi32.OffsetViewportOrgEx |
| 004608EC | 77F17988 | gdi32.SetViewportOrgEx |
| 004608F0 | 77F1A990 | gdi32.SetROP2 |
| 004608F4 | 77F15D0B | gdi32.SetBkMode |
| 004608F8 | 77F197BE | gdi32.RestoreDC |
| 004608FC | 77F19884 | gdi32.SaveDC |



Với bảng IAT hoàn chỉnh không lỗi như trên, tôi có thể mở một Ollydbg khác, load unpackme vào và tới OEP mà không thực hiện việc chỉnh sửa gì, sau đó tôi quay trở lại OllyDbg đã có đầy đủ bảng IAT, thực hiện sao chép và dán đúng bảng IAT này sang OllyDbg mới bằng tính năng **Bynary Copy** and **Binary Paste**, rất đơn giản .. bạn có thể

tự thực hiện. Sau đó, từ OllyDbg mới tiến hành Dump file và dùng ImpREC để Get Imports và fix dump một cách bình thường.

## III. Kết luận

Toàn bộ phần 32 đến đây là kết thúc! Tôi có đính kèm thêm một unpackme **unpackme- FSG 1.31 - dulek.exe** như là một bài tập để các bạn thực hành. Nó rất đơn giản  . Tôi hy vọng các bạn có thể unpack được nó mà không gặp trở ngại nào, và OEP các bạn tìm được sẽ là pass để giải nén cho phần tiếp theo.

Cảm ơn các bạn đã dành thời gian để theo dõi. Hẹn gặp lại các bạn ở phần tiếp theo!

***PS: Tài liệu này chỉ mang tính tham khảo, tác giả không chịu trách nhiệm nếu người đọc sử dụng nó vào bất kì mục đích nào.***

**Best Regards**

***_[Kienmanowar]_***

**--++--==[ Greatz Thanks To ]==--++--**

My family, Computer_Angel, Moonbaby, Zombie_Deathman, Littleboy, Benina, QHQCrker, the_Lighthouse, Merc, Hoadongnoi, Nini ... all REA's members, TQN, HacNho, RongChauA, Deux, tlandn, light.phoenix, dqtln, ARTEAM ... all my friend, and YOU.

**--++--==[ Thanks To ]==--++--**

iamidiot, WhyNotBar, trickyboy, dzungltvn, takada, hurt_heart, haule_nth, hytkl, moth, XIANUA, nhc1987, 0xdie, Unregistered!, akira, mranglex v...v.. các bạn đã đóng góp rất nhiều cho REA. Hi vọng các bạn sẽ tiếp tục phát huy ☺

I want to thank **Teddy Roggers** for his great site, Reversing.be folks(especially **haggar**), Arteam folks(**Shub-Nigurrath, MaDMAn_H3rCuL3s**) and all folks on crackmes.de, thank to all members of **unpack.cn** (especially **fly** and **linhanshi**). Great

thanks to **lena151** (I like your tutorials). And finally, thanks to **RICARDO NARVAJA** and all members on **CRACKSLATINOS.**

>>>> If you have any suggestions, comments or corrections, email me:
**kienbigmummy[at]gmail.com**