

2014

[Cracking with OllyDbg]

Based on OllyDbg tuts of Ricardo Narvaja (CrackLatinos Team)



www.reaonline.net

kienmanowar



30/11/2014

Mục Lục

I. Giới thiệu chung	2
II. Phân tích và xử lý target	2
1. Tổng quan	2
2. Thực hành	3
2.1. Tìm Opcode	3
2.2. Sử dụng tính năng tìm kiếm OEP của OllyDBG	7
2.3. Sử dụng BPM on access với OllyDBG đã patch để tìm OEP	10
2.4. PUSHAD Method	15
2.5. Phương pháp Exceptions	20
2.6. Phương pháp hàm API được sử dụng bởi Packer	25
2.7. Phương pháp hàm API được gọi đầu tiên bởi chương trình	31
III. Kết luận	33

I. Giới thiệu chung

Trong phần trước, tôi đã cùng các bạn tìm hiểu một số kiến thức tổng quan về packer, về layout của PE file sau khi bị pack, tóm lược các bước cơ bản khi thực thi một chương trình bị pack, và quan trọng nhất là chúng ta đã hiểu rõ hơn thuật ngữ OEP - là nơi dòng lệnh đầu tiên của file gốc được thực thi. Thông thường, 99% là OEP sẽ nằm ở section đầu tiên (đó là section .text (code), bên dưới PE header khi quan sát trong cửa sổ Memory map của OllyDBG), tuy nhiên vẫn có những trường hợp ngoại lệ là OEP không nằm tại section đầu tiên. Nhưng về bản chất thì cuối cùng OEP vẫn phải nằm đâu đó thuộc **Executable Code Section**, có tên là .text (Microsoft) hoặc là CODE (Borland). Trong phần 26 này tôi sẽ giới thiệu tới các bạn một số phương pháp cơ bản, phổ biến, thường áp dụng khi tìm OEP.

Now let's go..... 😁

II. Phân tích và xử lý target

1. Tổng quan

Như các bạn đã biết, các trình Packer thực hiện việc nén ứng dụng của chúng ta và chèn thêm một đoạn code - thường gọi là unpacking stub, có chứa giải thuật để giải nén (decompress). **EP** (EntryPoint) của ứng dụng sẽ bị thay đổi để trỏ tới địa chỉ bắt đầu đoạn stub và sau khi stub thực hiện xong công việc của nó, hướng thực hiện của chương trình sẽ nhảy về **Original Entrypoint** (OEP) để tiếp tục việc thực thi chương trình đã được unpack của chúng ta. Và khi tới được OEP, quan sát code/bộ nhớ (memory), ta thấy nội dung sẽ tương tự như file gốc ban đầu (vì lúc đó packer đã hoàn tất quá trình unpack file), điều này cho phép chúng ta có thể thực hiện việc kết xuất (dump) và tạo unpacked file gần giống với file gốc nhất để từ đó thực hiện việc tái tạo (rebuild) file. Mục đích của việc rebuild là làm sao để file sau khi unpack có thể thực thi được bình thường như khi ta thực thi file gốc.

Thông thường, các bước cơ bản để thực hiện unpack như sau:


1. Tìm OEP
2. Kết xuất file (Dump)
3. Sửa lại IAT (Repair IAT), fix sections
4. Kiểm tra file xem còn các cơ chế Anti hoặc ngăn chặn không cho thực thi không và tiến hành chỉnh sửa. Đảm bảo file sau unpack thực thi bình thường.

Trên đây được xem là những bước phổ biến, và có thể tùy biến tùy thuộc vào từng loại packer. Trong phần này, chúng ta sẽ tập trung vào các phương pháp/cách thức để tìm được OEP, thông qua việc thực hành với vài trình packer khác nhau. Phải nhấn mạnh rằng với unpacking không gì khác ngoài việc chúng ta phải chịu khó, kiên nhẫn, thử đi thử lại các cách, thông thường các phương pháp hoạt động tốt, tuy nhiên sẽ có lúc ta gặp các trình packers "dị" hơn thì các phương pháp mà ta hay áp dụng không còn phù hợp nữa, lúc đó nó đòi hỏi một số kỹ thuật lắt léo mà chỉ có bằng việc thực hành thật nhiều chúng ta mới tích lũy được cho mình những kinh nghiệm quý giá.

2. Thực hành

Để minh họa một cách cụ thể, chi tiết hơn, tôi sẽ sử dụng file Cruehead Crackme đã bị pack trong phần trước để giải thích cho các phương pháp sẽ được sử dụng dưới đây. Bên cạnh đó, chúng ta cũng sẽ thực hành trên các trình packer khác nữa để có cái nhìn tổng quan, đa chiều hơn.

2.1. Tìm Opcode

Cách này đơn giản và hơi thô, chỉ có thể áp dụng với các trình packer đơn giản, không thể áp dụng nó cho các trình packer khó nhằn như Asprotect v...v... . Cách này cũng được dùng để tìm opcodes của `LONG JMP (0xE9)` hoặc `LONG CALL (0xE8)`. Ý tưởng của phương pháp rất đơn giản: các trình packer sau khi thực hiện pack file thì khả năng nó cần có một lệnh nhảy hoặc một lời gọi hàm (call) tới section đầu tiên, mục đích là để tới được OEP. Với phương pháp này, ta hi vọng các lệnh `JUMP` hoặc `CALL` sẽ xuất hiện ngay đầu tiên trong quá trình tìm kiếm hoặc trình packer sẽ không có các cơ chế tự động chỉnh sửa.

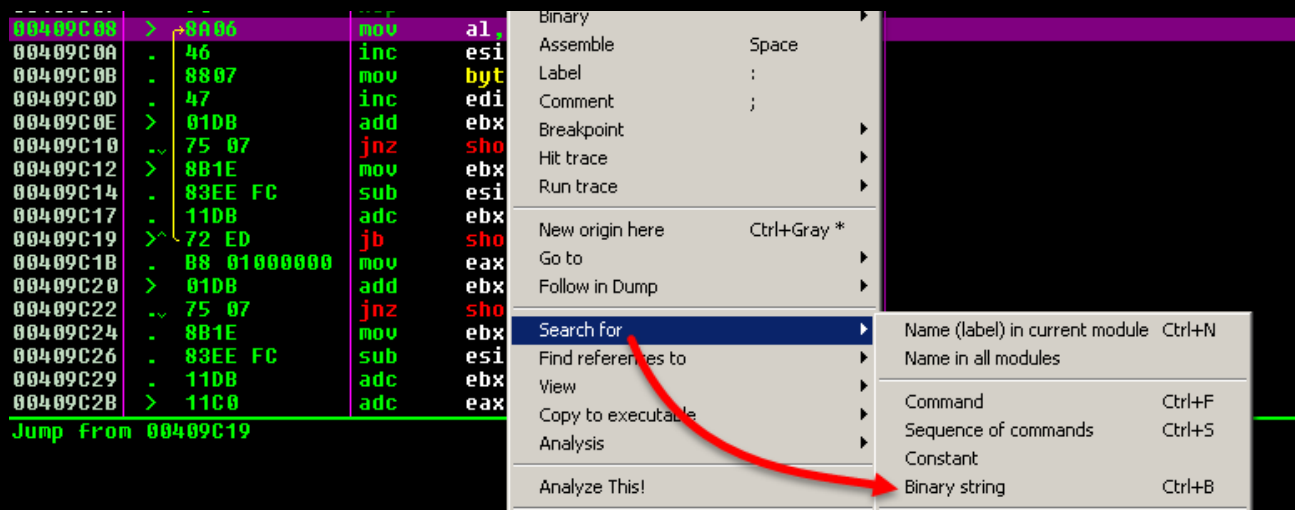
Vì vậy, với phương pháp này ta sẽ thực hiện như sau, load crackme đã bị pack ở phần trước vào OllyDBG:

```

00409BF0 $ 60 pushad
00409BF1 > BE 00904000 mov esi, 00409000
00409BF6 - 8DBE 0080FFFF lea edi, dword ptr [esi+0xFFFF8000]
00409BFC - 57 push edi
00409BFD - 83CD FF or ebp, 0xFFFFFFFF
00409C00 ~ EB 10 jmp short 00409C12
00409C02 90 nop
00409C03 90 nop
00409C04 90 nop
00409C05 90 nop
00409C06 90 nop
00409C07 90 nop
00409C08 > 8A06 mov al, byte ptr [esi]
00409C0A - 46 inc esi
00409C0B - 8807 mov byte ptr [edi], al
00409C0D - 47 inc edi
00409C0E > 01DB add ebx, ebx

```

Chuột phải và chọn **Search for > Binary string (Ctrl + B)**:



Nhập thông tin opcode cần tìm kiếm, tương tự như hình minh họa:

Nhấn OK để thực hiện tìm kiếm. Mục tiêu của chúng ta là tìm lệnh nhảy nhảy tới section đầu tiên, do đó nếu kết quả không đúng như mong đợi, nhấn **Ctrl+L** để tiếp tục tìm tiếp. Sau khi nhấn OK, ta có được kết quả đầu tiên như sau:



Yup 🤪, đây chính là lệnh nhảy mà ta cần tìm. Đặt một BP tại đây, nhấn **F9** để thực thi, sau đó nhấn **F7** để trace qua lệnh nhảy, ta sẽ tới được OEP:

Address	Disassembly	Comment
00401000	6A 00	push 0x0
00401002	FF 04 00 00	call 00401506
00401007	A3 02 04 00	mov dword ptr [0x4020CA], eax
0040100C	6A 00	push 0x0
0040100E	68 F4 20 00	mov eax, 20F4
00401013	E8 A6 04 00	call 004014BE
00401018	0BC0	mov eax, 0
0040101A	74 01	jump short 0040101D
0040101C	C3	retn
0040101D	C7 05 64 20 00 00	mov dword ptr [0x402064], 0x4003
00401027	C7 05 68 20 00 00	mov dword ptr [0x402068], 00401128
00401031	C7 05 6C 20 00 00	mov dword ptr [0x40206C], 0x0

Ngoài cách tìm kiếm lệnh nhảy như trên, đôi khi ta xem code của các trình packer sẽ thấy có các lệnh kiểu như **CALL EAX**, **CALL EBX**, **JMP EAX**. Rất nhiều trình packer thường sử dụng các thanh ghi nhằm mục đích để che dấu việc nhảy tới OEP. Với các trường hợp này, cách tối ưu nhất là thực hiện tìm kiếm tất cả các lệnh bằng cách nhấn chuột phải, chọn **Search for > All commands**, sau đó đặt BP lên tất cả các lệnh tìm được, ví dụ:

Address	Disassembly	Comment
00409C08	> 8A 06	mov al, byte ptr [esi]
00409C0A	. 46	inc esi
00409C0B	. 88 07	mov byte ptr [edi], al
00409C0D	. 47	inc edi
00409C0E	> 01 DB	add ebx, 1
00409C10	~ 75 07	jnz short 00409C19
00409C12	> 8B 1E	mov ebx, esi
00409C14	. 83 EE FC	sub esi, 5
00409C17	. 11 DB	adc ebx, esi
00409C19	> 72 ED	jb short 00409C12
00409C1B	. B8 01 00 00 00	mov eax, 1
00409C20	> 01 DB	add ebx, 1
00409C22	~ 75 07	jnz short 00409C19
00409C24	. 8B 1E	mov ebx, esi
00409C26	. 83 EE FC	sub esi, 5
00409C29	. 11 DB	adc ebx, esi
00409C2B	> 11 C0	adc eax, ebx

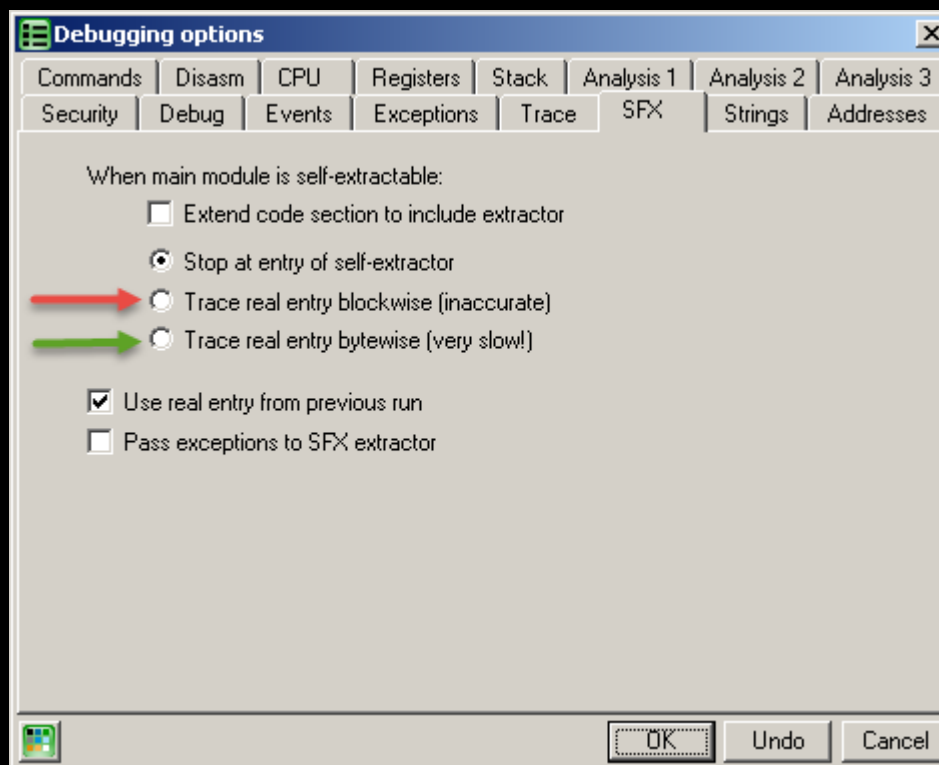
Với trường hợp crackme của chúng ta đang sử dụng làm ví dụ, khi thực hiện tìm kiếm sẽ không có kết quả nào trả về. Tuy nhiên, giả sử nếu như có nhiều kết quả trả về, ta sẽ đặt BP lên toàn bộ các lệnh đã tìm được. Cho thực thi và hi vọng có khả năng break tại BP đã đặt, lúc đó ta sẽ tiến hành kiểm tra xem giá trị của thanh ghi **EAX** lúc break là

gì, nếu ta thấy giá trị của thanh ghi trong lệnh **CALL** hoặc **JMP** thuộc sections đầu tiên, nhấn **F7** để tìm tới OEP.

Như đã nói, cách thức thực hiện tìm kiếm kiểu này khá thô và tay to 🤔, thường không được áp dụng rộng rãi, bởi hầu hết các trình packer bây giờ đều có khả năng tự động chỉnh sửa, hoặc áp dụng các phương pháp/cơ chế đặc biệt để che dấu việc nhảy tới OEP, sẽ không dễ dàng để tìm kiếm được. Tóm lại, đây là một cách đơn giản và có thể áp dụng đối với một số trình packer cũ hoặc bad packer 😞.

2.2. Sử dụng tính năng tìm kiếm OEP của OllyDBG

Load crackme Cruehead đã bị pack vào OllyDBG, chọn **Options > Debugging options** hoặc nhấn phím tắt **Alt+O**, chuyển tới tab **SFX**:



Tại tab **SFX**, ta thấy có hai tùy chọn có thể được áp dụng để tìm OEP. Với tùy chọn được trỏ bởi mũi tên màu đỏ sẽ cho phép thực hiện tìm kiếm nhanh nhất, tùy chọn được trỏ bởi mũi tên màu xanh sẽ thực hiện tìm kiếm chậm hơn, nhưng đôi khi lại cho kết quả tốt hơn. Ta thử lựa chọn tùy chọn được trỏ bởi mũi tên màu đỏ xem thế nào.

Sau khi chọn xong, khởi động lại OllyDBG và quan sát, tuy nhiên ta thấy dường như không có tác dụng gì cả, có vẻ nó không hoạt động 🤔. OllyDBG vẫn dừng lại tại lệnh **00409BF0 > \$ 60 pushad**, chứ không phải thực hiện quá trình tìm kiếm OEP như

chúng ta mong muốn. Để tìm hiểu kĩ hơn tác dụng của SFX, ta mở file help của OllyDBG và tìm kiếm thông tin liên quan đến SFX, ta có được thông tin như sau:

Self-extracting (SFX) files

Self-extracting file consists of extracting routine and packed original program. When troubleshooting SFX, you usually want to skip extractor and stop on the entry point of original program ("real entry"). OllyDbg contains several functions that facilitate this task.

Usually extractor loads to address that is outside the executable section of the original program. In this case OllyDbg recognizes file as SFX.

When [SFX options](#) request tracing of real entry, OllyDbg sets memory breakpoint on the whole code section. Initially this is empty or contains compressed data. When program attempts to execute some command within protected area which is neither **RET** nor **JMP**, OllyDbg reports real entry. This is how bytewise extraction works.

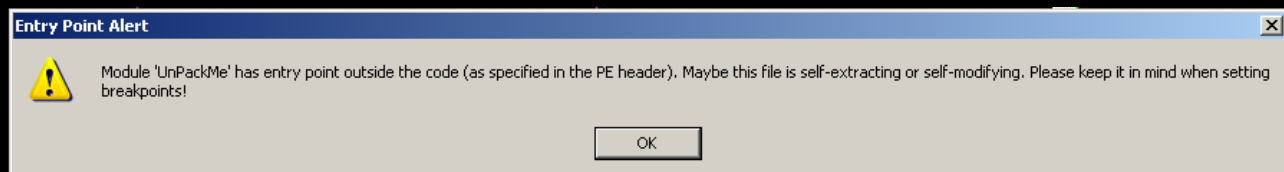
This method is very slow. There is another, much faster method. Each time exception on data read occurs, OllyDbg enables reading from this 4-K memory block and disables previous read window. On each data write exception it enables writing to this block and disables previous write window. When program executes command within remaining protected area, OllyDbg reports real entry. However, when real entry is inside read or write window, its location will be reported incorrectly.

You can correct entry position. Select new entry and from Disassembler popup menu choose 'Breakpoint[[Set real SFX entry here](#)']'. If corresponding SFX option is enabled, next time OllyDbg skips extractor quickly and reliably.

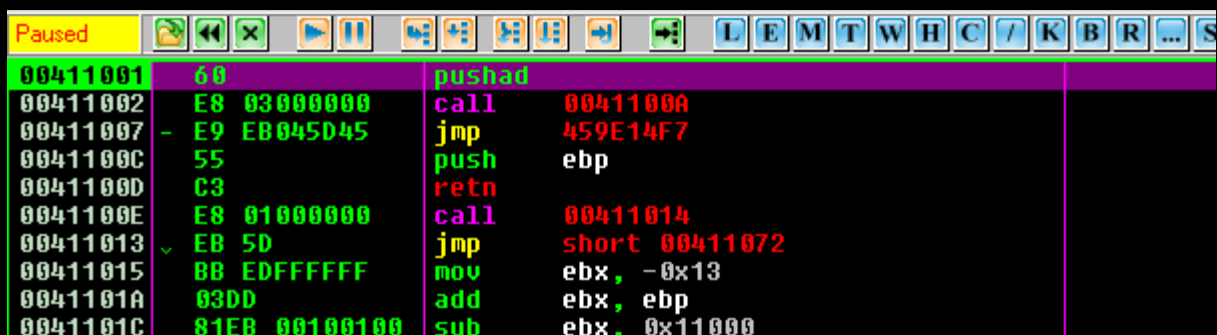
Notice that OllyDbg usually fails to trace extracting routine that implements protection or anti-debugging techniques.

Căn cứ trên thông tin về SFX mà help file của OllyDBG đã cung cấp, ta thấy phương pháp này chỉ hoạt động khi OllyDBG phát hiện/nhận biết được Entry Point nằm ngoài section code, như trong hầu hết các chương trình bị pack. Nhưng cụ thể, đối với trường hợp này, chúng ta thấy OllyDBG không đưa ra bất kỳ cảnh báo nào về việc Entry Point nằm ngoài section code cả, bản chất vấn đề ở đây là UPX thực hiện thay đổi code section để chạy unpacker, do vậy EP vẫn nằm cùng luôn với section code. Chính vì vậy, OllyDBG không phát hiện được đây là file bị pack, suy ra phương pháp này không thể áp dụng được trong ví dụ này.

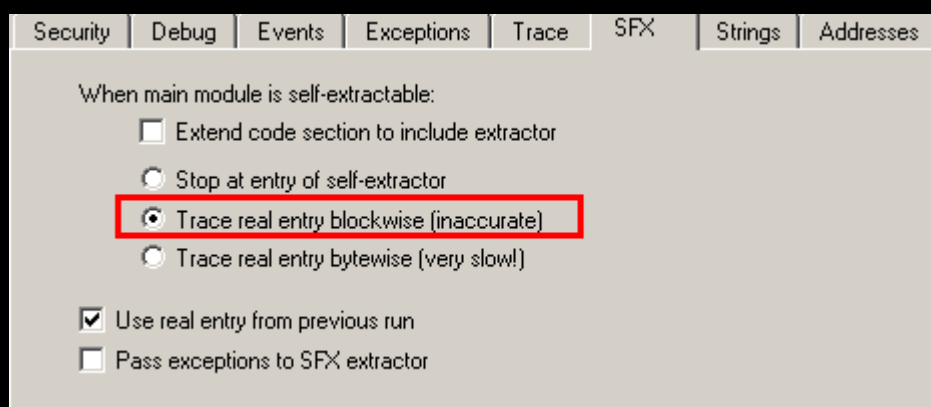
Tuy nhiên, để minh họa cho phương pháp này, ta sẽ sử dụng một file Unpackme khác được cung cấp kèm theo bài viết là: **UnPackMe_ASPack2.12**. Unpackme này được pack bằng ASPack, một trình packer cũng nổi tiếng không kém gì UPX. Trước khi load crackme này vào OllyDBG, ta thiết lập tùy chọn tại tab **SFX** về như ban đầu. Sau đó, tiến hành load file vào OllyDBG. Ta nhận được thông báo sau:



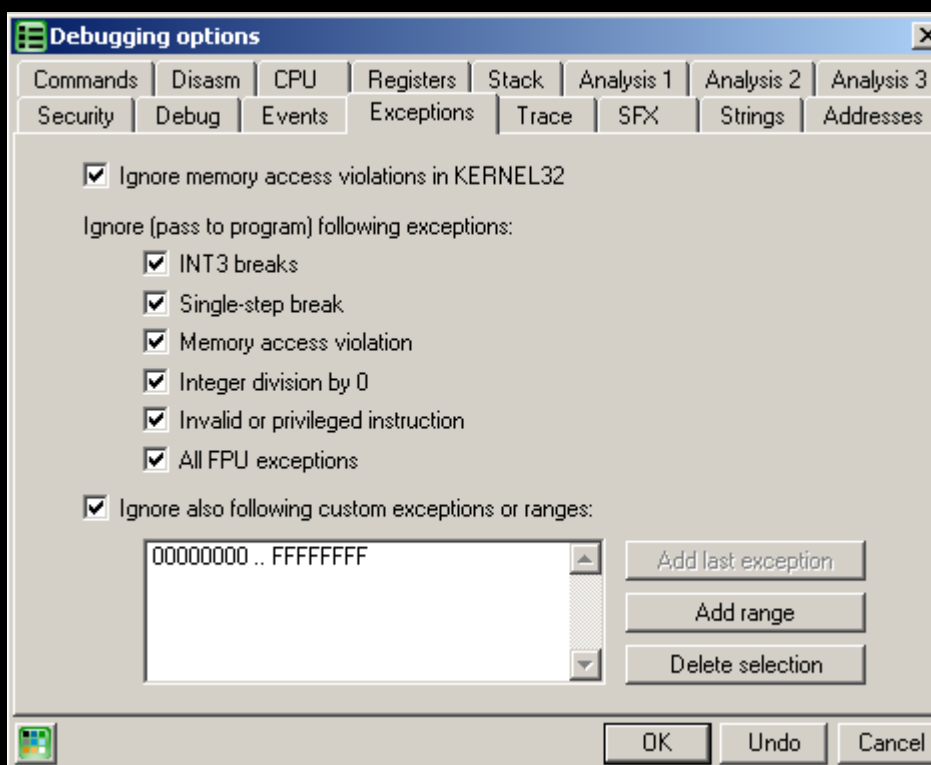
Nhấn OK, ta sẽ dừng lại tại EP của Unpackme:



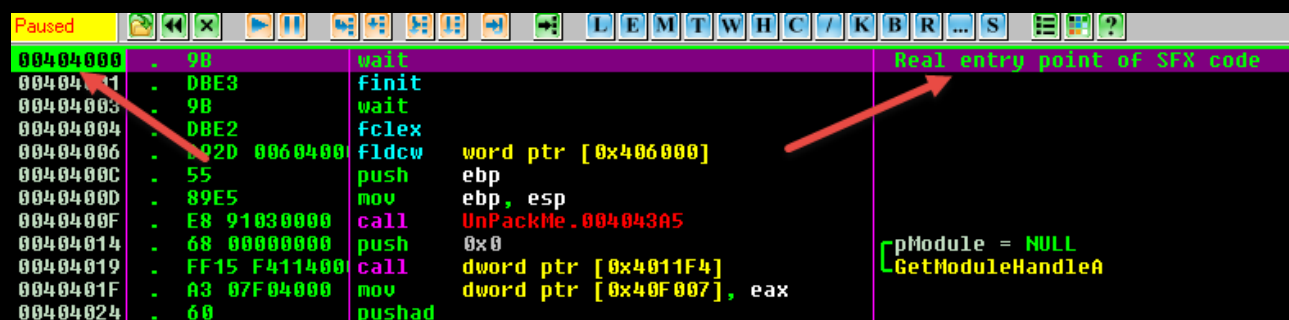
Thiết lập lại tùy chọn trong tab **SFX** như hình:



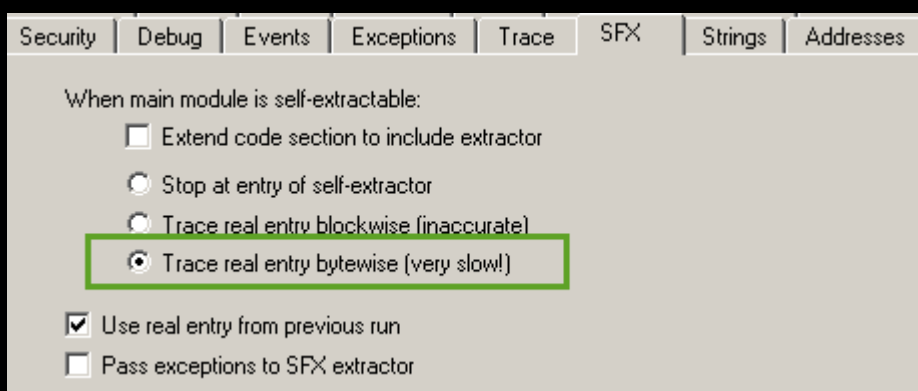
Chuyển qua tab Exceptions, lựa chọn tương tự như hình:



Thiết lập xong ta khởi động lại OllyDBG (**Ctrl+F2**), quan sát một lúc ta sẽ thấy OllyDBG đưa ta đến địa chỉ **00404000**, kèm theo dòng chú thích **"Real entry point of SFX code"** (mặc dù địa chỉ này không nằm trong section đầu tiên như chúng ta đã từng nói, tuy nhiên Unpackme này là một trường hợp đặc biệt, nó nằm ở section thứ 3, bản chất vẫn là section code).



Như vậy, ta thấy rằng với crackme này thì phương pháp sử dụng SFX để tìm OEP hoạt động khá tốt. Tiếp theo, ta thử lựa chọn tùy chọn thứ 2 trong tab SFX xem nó hoạt động thế nào (hi vọng trả về cùng một kết quả 😊):



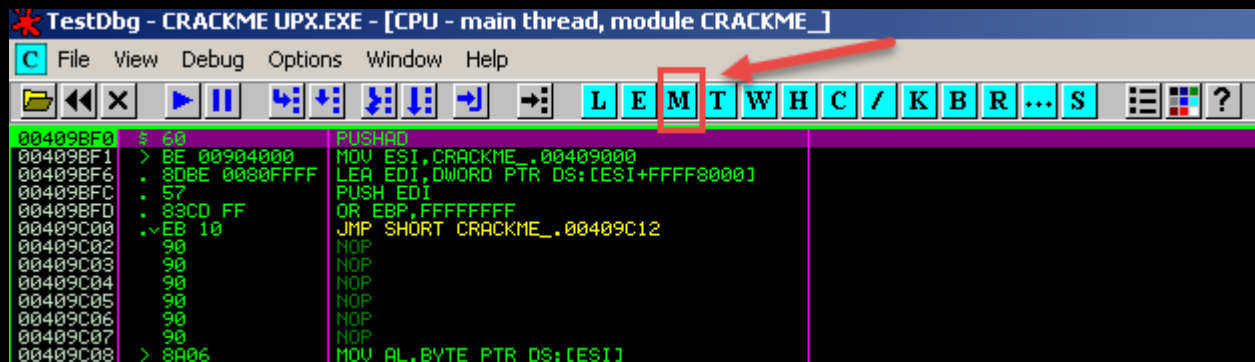
Sau khi thiết lập xong, khởi động lại OllyDBG (**Ctrl+F2**). Đợi một chút, ta cũng dừng lại tại địa chỉ giống hệt như trên và đó chính là OEP cần tìm.

Như vậy, phần thực hành với phương pháp này đã xong. Các bạn nhớ sau khi thực hiện xong phương pháp này thì phải thiết lập tab SFX trở về tùy chọn ban đầu, nếu không thì OllyDBG sẽ không dừng lại tại EP như bình thường nữa mà luôn luôn thực hiện việc tìm kiếm OEP 🤖.

2.3. Sử dụng BPM on access với OllyDBG đã patch để tìm OEP

Bản OllyDBG dùng trong ví dụ này được cung cấp bởi lão Ricardo. Tại sao lại có bản patch này? Đơn giản là với bản OllyDBG này, khi ta đặt một **Memory breakpoint on access** thì OllyDBG chỉ dừng lại khi thực thi (Execute), mà không dừng lại khi có sự đọc (Read) hoặc ghi (Write) vào vùng nhớ mà ta đã thiết lập breakpoint, đó chính là ý tưởng được áp dụng để phục vụ việc tìm OEP. Do vậy, khi ta đặt một BPM tại section code và cho thực thi trong OllyDBG, OllyDBG sẽ bỏ qua mọi thao tác Read/Write và chỉ dừng lại tại địa chỉ (dòng code) được thực thi đầu tiên tại Section này, và khả năng địa chỉ đó chính là OEP cần tìm 🤖.

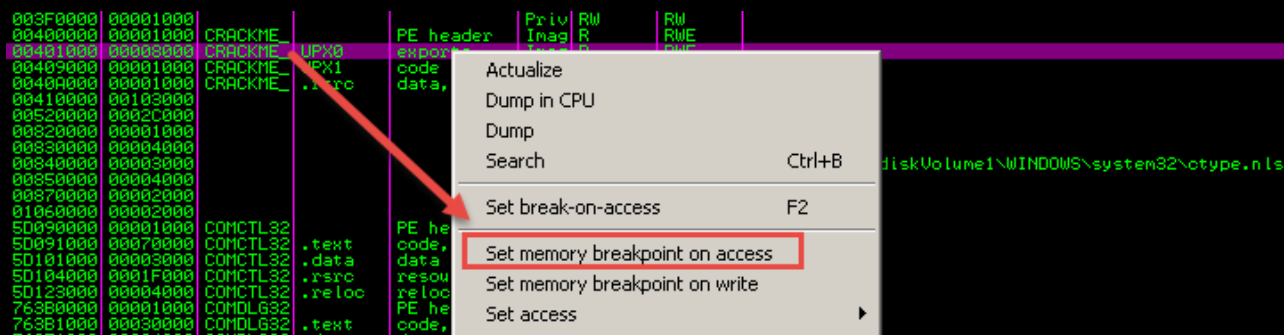
Load file crackme Cruehead đã pack bằng UPX vào bản OllyDBG mà lão Ricardo cung cấp. Chuyển tới cửa sổ **Memory map** bằng cách nhấn **Alt+M** hoặc nhấn vào biểu tượng chữ **M**:



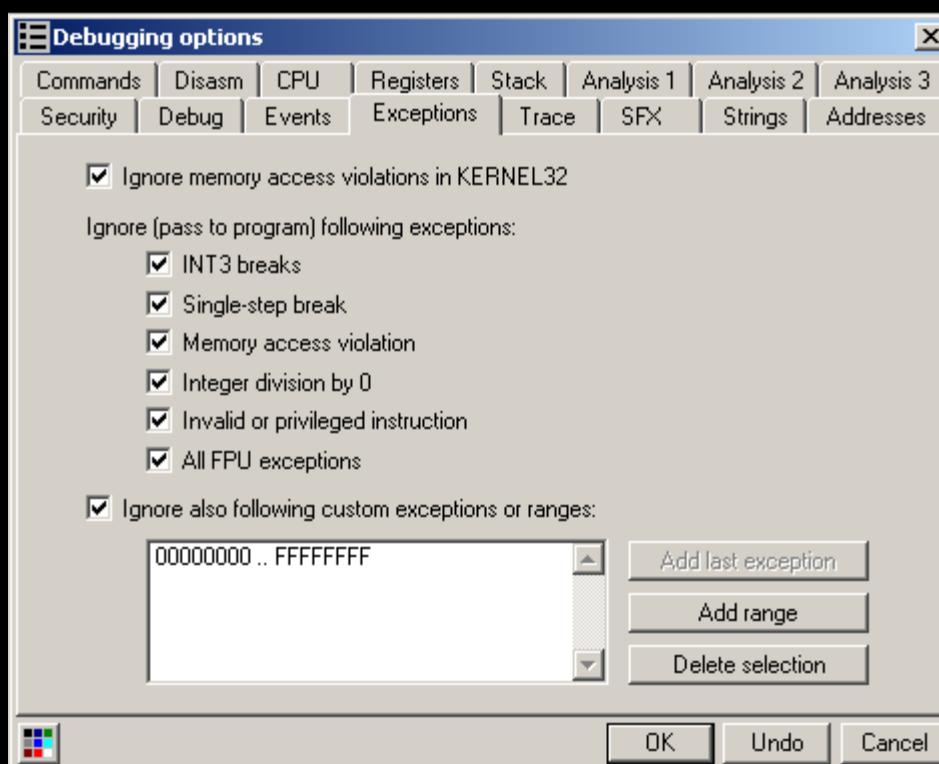
Tại cửa sổ Memory map ta thấy được các sections của file bị pack:

003F0000	00001000								
00400000	00001000	CRACKME_	PE header	Priv	RW				
00401000	00003000	CRACKME_	UPX0	Imports	Imag	R		RWE	
00409000	00001000	CRACKME_	UPX1	code	Imag	R		RWE	
0040A000	00001000	CRACKME_	.rsrc	data, import	Imag	R		RWE	
00410000	00103000	CRACKME_		Map	R			R	

Tại section đang được highlight chính là section đầu tiên, đây chính là nơi được dùng để giải nén và giải mã các byte gốc. Thực hiện đặt một BPM tại section này:



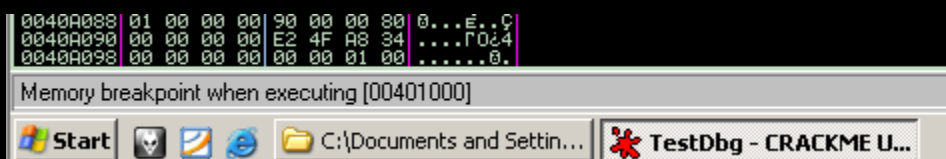
Để đảm bảo quá trình thực thi trong OllyDBG không bị dính exceptions, ta thiết lập các tùy chọn tại tab **Exceptions** như sau:



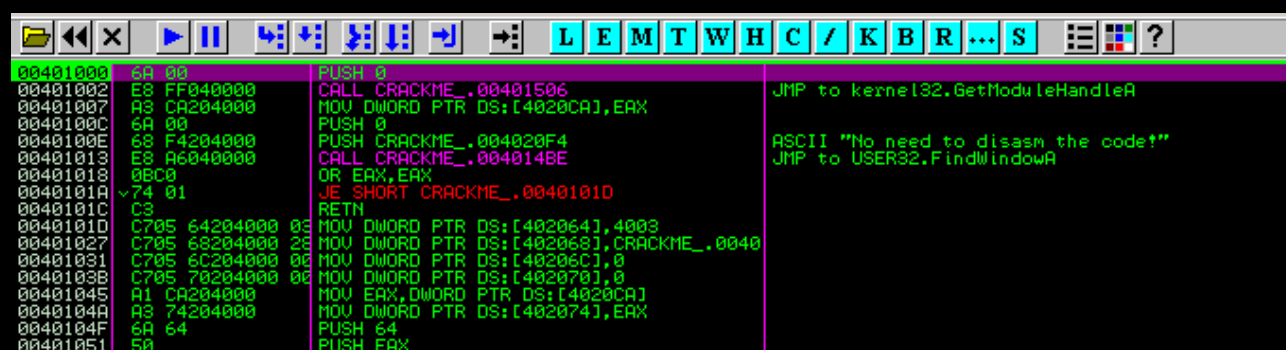
Sau khi thiết lập xong, nhấn **OK** để quay trở lại màn hình code của OllyDBG. Tại đây, nhấn **F9** để thực thi crackme, tại Status bar ta sẽ thấy thông tin sau:

```
0040A080 00 00 00 00 00 00 01 00 .....0.
0040A088 01 00 00 00 90 00 00 80 0...é...C
0040A090 00 00 00 00 E2 4F A8 34 ....Γ0z4
0040A098 00 00 00 00 00 00 01 00 .....0.
Tracing SFX: write=00401000
```

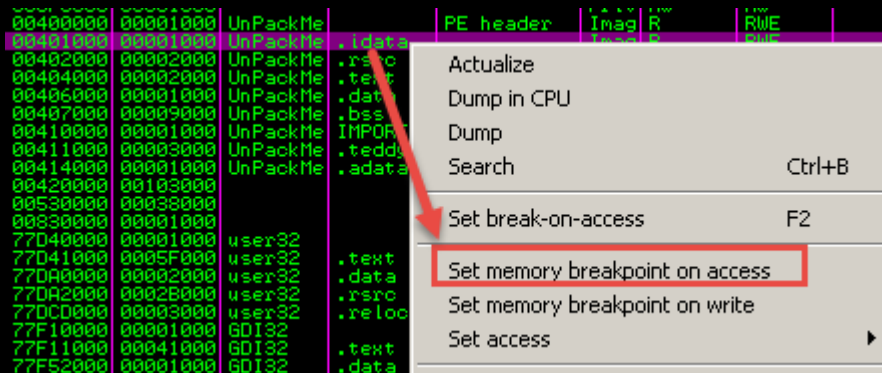
Sau khoảng hơn chục giây (tùy thuộc vào từng packer, thời gian này có thể là vài giây hoặc lên tới vài phút), OllyDBG sẽ break tại BPM đã thiết lập:



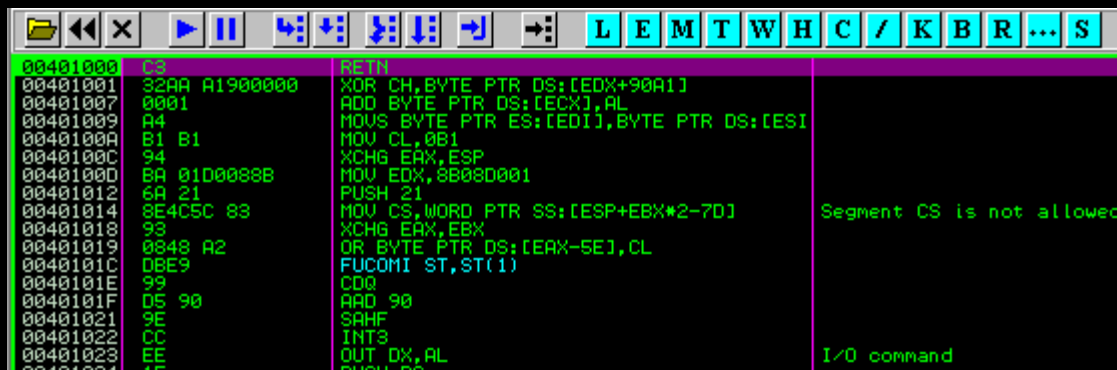
Tại cửa sổ code ta sẽ thấy đang dừng tại OEP:



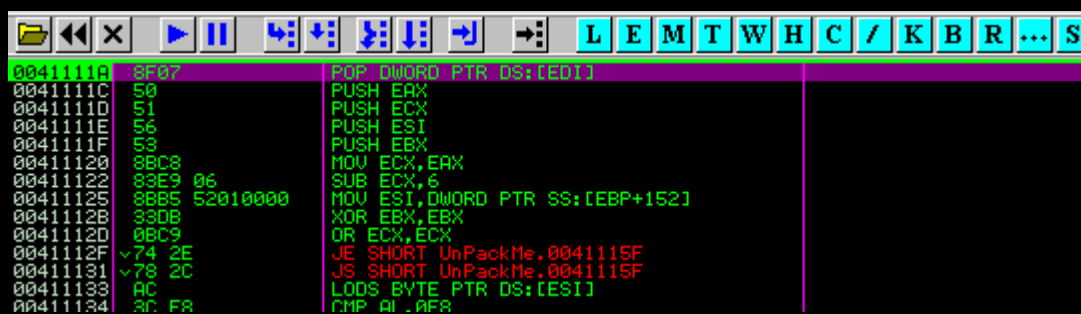
Tiếp tục thực hành phương pháp này với file unpackme được pack bằng ASPack (**UnPackMe_ASPack2.12.exe**). Load file vào OllyDBG, chuyển tới cửa sổ **Memory map**, lựa chọn section đầu tiên để thiết lập BPM:



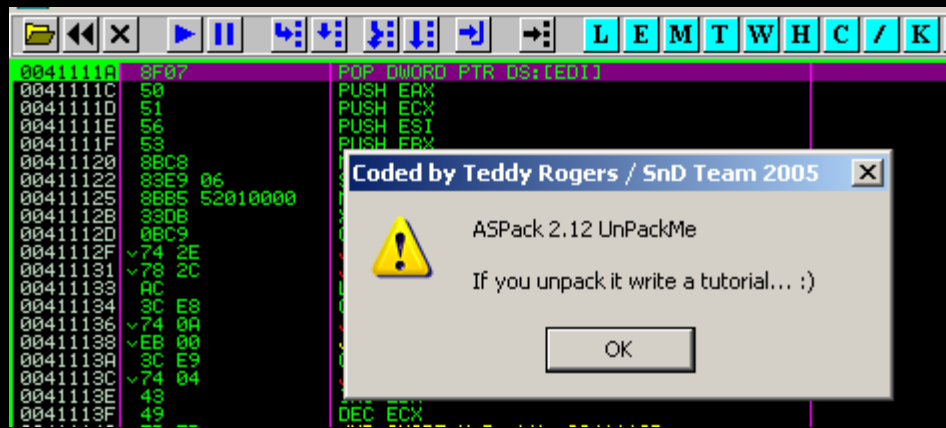
Chuyển về cửa sổ code, nhấn **F9** để run, OllyDBG sẽ break và dừng lại tại đây:



Đây là dòng code đầu tiên được thực thi, xem thử điều gì xảy ra nếu như ta nhấn **F7** để trace:



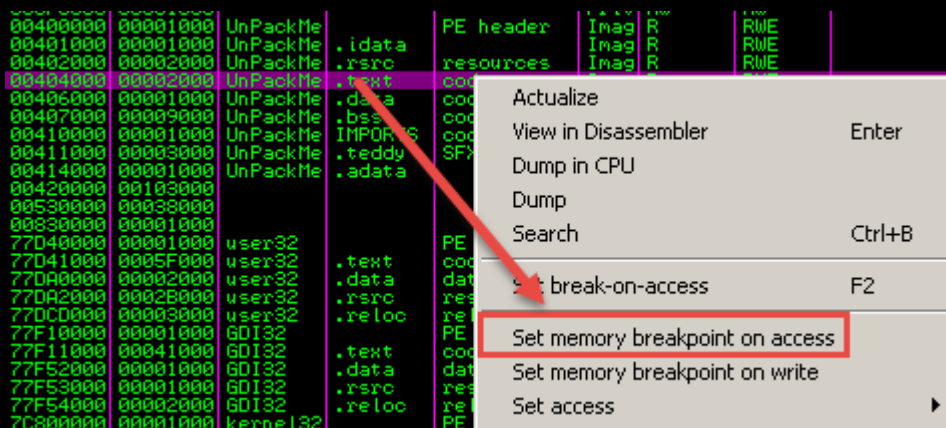
Trace qua lệnh **RETN** ta được đưa tới địa chỉ **0x004111A**, địa chỉ này nếu như ta quan sát tại cửa sổ **Memory map** thì nó vẫn thuộc section mà trình packer thêm vào. Tiếp tục nhấn **F9** để run một lần nữa, ta thấy unpackme sẽ thực thi hoàn toàn mà không break nữa. Như vậy có thể thấy rằng, khi ta tìm kiếm OEP với OllyDBG trong trường hợp packer đã thay đổi và không còn unpack vào section đầu tiên nữa thì để tìm OEP chỉ còn mỗi cách là tiếp tục đặt BPM on access lên các section khác để xác định phạm vi.




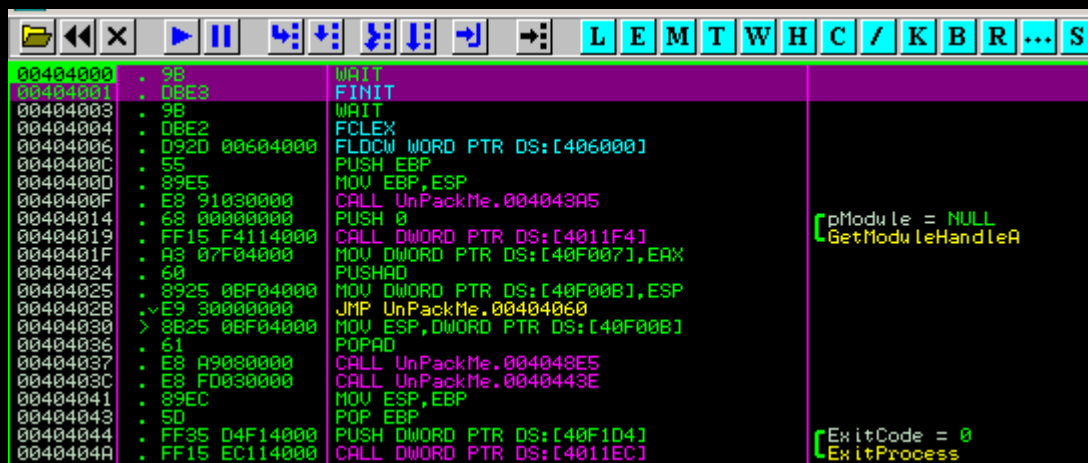
Tiếp tục thử đặt BPM on access tại section thứ hai:

00400000	00001000	UnPackMe		PE header	Inag	R	RWE
00401000	00001000	UnPackMe	.idata		Inag	R	RWE
00402000	00002000	UnPackMe	.rsrc	resources	Inag	R	RWE
00404000	00002000	UnPackMe	.text	code	Inag	R	RWE
00406000	00001000	UnPackMe	.data	code,data	Inag	R	RWE
00407000	00009000	UnPackMe	.bss	code	Inag	R	RWE

Kết quả cũng tương tự như section đầu tiên. Tiếp tục với section thứ 3:



Nhấn **F9** để thực thi, OllyDBG sẽ break và dừng lại tại OEP  :



2.4. PUSHAD Method

Nhiều trình packer sau khi load vào OllyDBG ta thấy lệnh đầu tiên được thực thi là lệnh **PUSHAD**. Lệnh này nhằm mục đích bảo toàn các thanh ghi, nó sẽ thực hiện lưu toàn bộ giá trị khởi tạo của tất cả các thanh ghi vào stack, sau đó thực hiện unpack routine, cuối cùng trước khi nhảy tới OEP, packer sẽ thực hiện việc khôi phục lại giá trị của các thanh ghi đã lưu trên stack bằng lệnh **POPAD**. Cặp lệnh **PUSHAD** và **POPAD** thường đi cùng với nhau. Nếu như ta thấy ở đâu đó trong mã của chương trình xuất hiện lệnh **PUSHAD** thì có nghĩa là đâu đó ở bên dưới chắc chắn sẽ có câu lệnh **POPAD**. Đây là phương pháp được áp dụng thành công cho khá nhiều trình packer.

Load UPX Crackme vào OllyDBG và quan sát:

Address	Disassembly	Comment
00409BF0	pushad	
00409BF1	mov esi, CRACKME_.00409000	
00409BF6	lea edi, dword ptr [esi+0xFFFF8000]	
00409BFC	push edi	
00409BFD	or ebp, 0xFFFFFFFF	
00409C00	jmp short CRACKME_.00409C12	

Ta thấy lệnh đầu tiên là **PUSHAD** (đôi khi lệnh này có thể nằm bên dưới, sau một vài lệnh khác). Trước khi thử thực hiện lệnh này ta ghi lại thông tin giá trị ban đầu của các thanh ghi tại cửa sổ Registers (có thể khác trên máy các bạn):

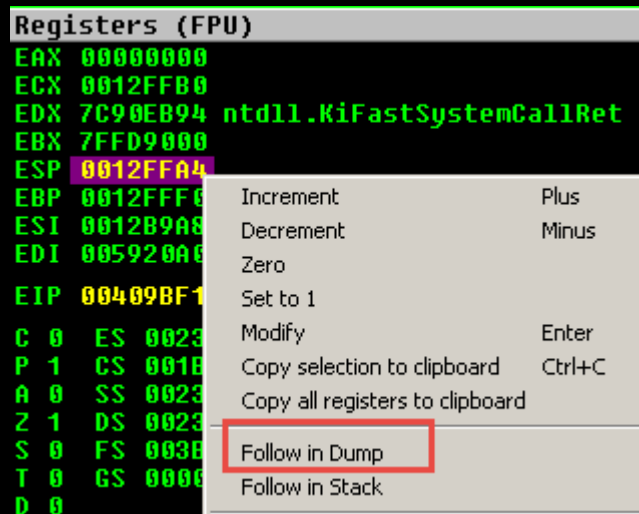
Register	Value	Comment
EAX	00000000	
ECX	0012FFB0	
EDX	7C90EB94	ntdll.KiFastSystemCallRet
EBX	7FFD9000	
ESP	0012FFC4	
EBP	0012FFF0	
ESI	0012B9A8	
EDI	005920A0	
EIP	00409BF0	CRACKME_.<ModuleEntryPoint>

Thực hiện trace qua lệnh **PUSHAD** và quan sát trên cửa sổ Stack:

Address	Value	Comment
0012FFA4	005920A0	EDI
0012FFA8	0012B9A8	
0012FFAC	0012FFF0	
0012FFB0	0012FFC4	
0012FFB4	7FFD9000	
0012FFB8	7C90EB94	ntdll.KiFastSystemCallRet
0012FFBC	0012FFB0	
0012FFC0	00000000	EAX
0012FFC4	7C816D4F	RETURN t
0012FFC8	005920A0	

Như các bạn thấy tại cửa sổ Stack, giá trị ban đầu của các thanh ghi đã được lưu lại. Do vậy, trước khi nhảy tới OEP thì chắc chắn packer phải thực hiện việc khôi phục lại các giá trị đã lưu này. Căn cứ vào đó, ta có thể tiến hành đặt một Hardware BPX On Access trong OllyDBG nhằm break lại khi packer thực hiện lệnh khôi phục giá trị, và khi break ta sẽ dừng lại tại lệnh nhảy tới OEP.

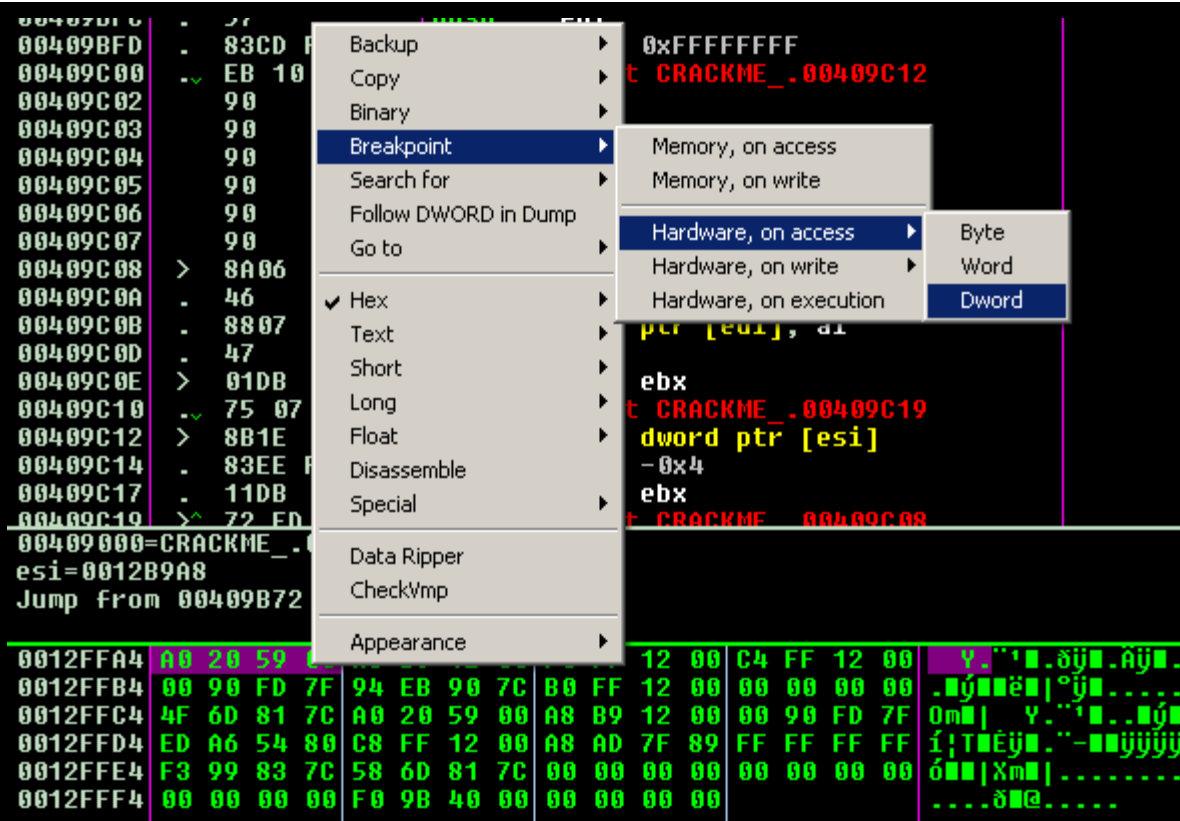
Lựa chọn thanh ghi ESP, nhấn chuột phải và chọn **Follow in Dump**:



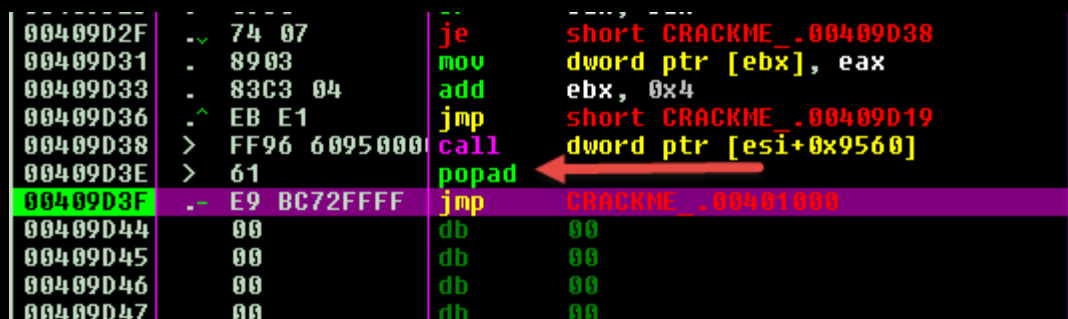
Quan sát tại cửa sổ Dump ta có được như sau:

0012FFA4	A0 20 59 00	A8 B9 12 00	F0 FF 12 00	C4 FF 12 00	Y.''.δyü.Äyü.
0012FFB4	00 90 FD 7F	94 EB 90 7C	B0 FF 12 00	00 00 00 00	.üüüü °yü.....
0012FFC4	4F 6D 81 7C	A0 20 59 00	A8 B9 12 00	00 90 FD 7F	0m Y.''.üüüü
0012FFD4	ED A6 54 80	C8 FF 12 00	A8 AD 7F 89	FF FF FF FF	í:TËyü.~--üüüüü
0012FFE4	F3 99 83 7C	58 6D 81 7C	00 00 00 00	00 00 00 00	óüü Xm
0012FFF4	00 00 00 00	F0 9B 40 00	00 00 00 00	δü@.....

Tại đây ta thấy được giá trị được lưu tại `ESP` chính là giá trị của thanh ghi `EDI`. Đánh dấu 4 bytes đầu tiên và lựa chọn đặt BP như sau:



Sau khi đặt BP xong, nhấn **F9** để Run, ta sẽ dừng lại tại đây:

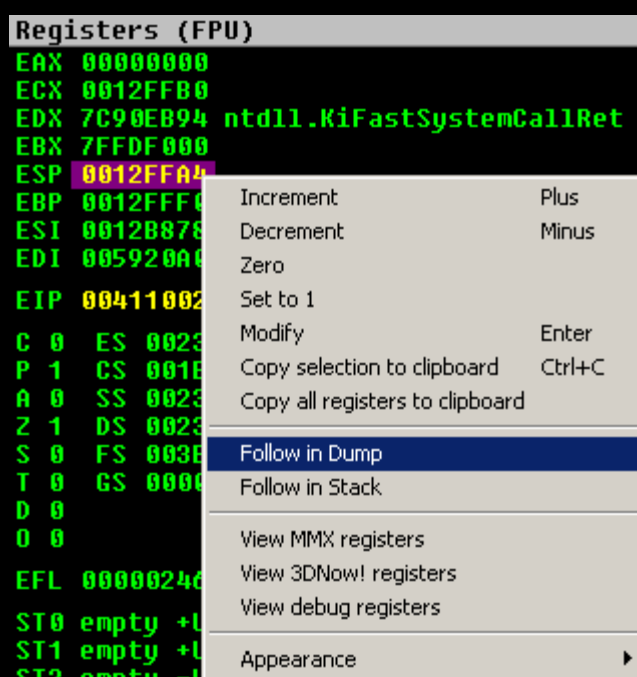


Theo dấu của mũi tên, ta có thể thấy lệnh **POPAD** được sử dụng để khôi phục lại giá trị của các thanh ghi từ Stack. Lệnh mà OllyDBG đang break tại, đó chính là lệnh nhảy tới OEP.

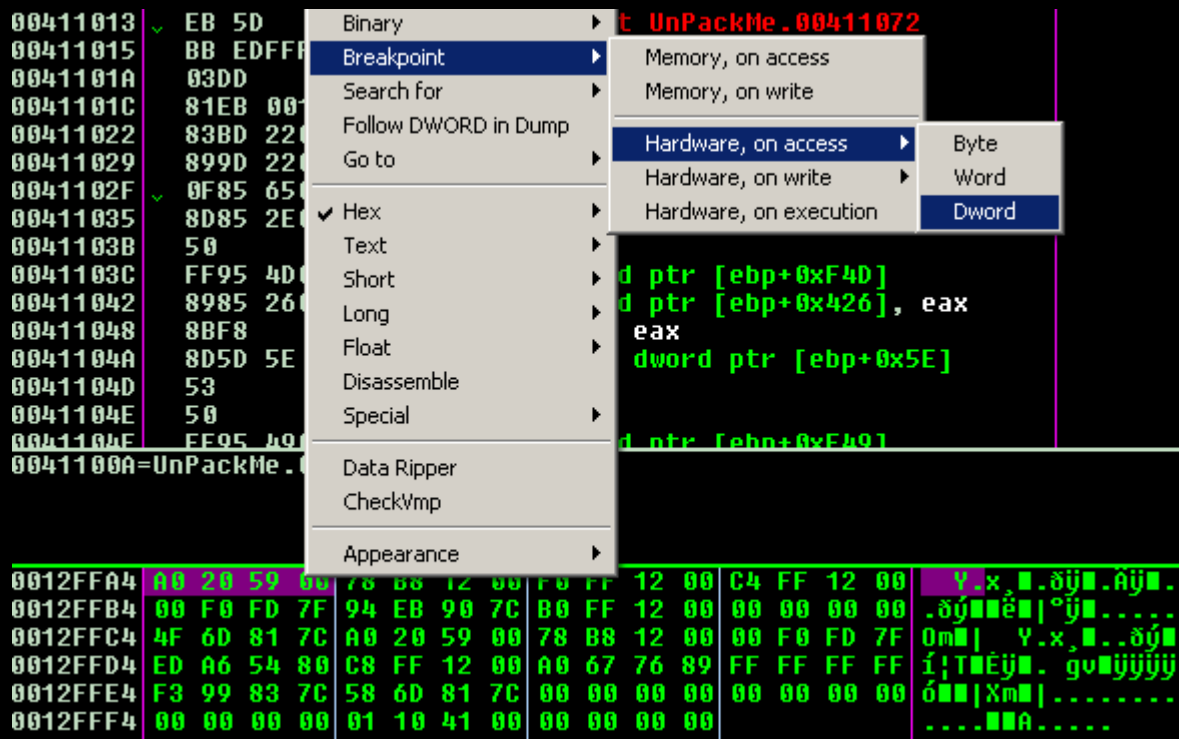
Tiếp tục áp dụng phương pháp này với Unpackme được pack bằng ASPack:

Address	Disassembly
00411001	60 pushad
00411002	E8 03000000 call UnPackMe.0041100A
00411007	- E9 EB045D45 jmp 459E14F7
0041100C	55 push ebp
0041100D	C3 retn
0041100E	E8 01000000 call UnPackMe.00411014
00411013	✓ EB 5D jmp short UnPackMe.00411072
00411015	BB EDFFFFFF mov ebx, -0x13
0041101A	0300 add ebx, ebp
0041101C	81EB 00100100 sub ebx, 0x11000
00411022	83BD 22040000 cmp dword ptr [ebp+0x422], 0x0

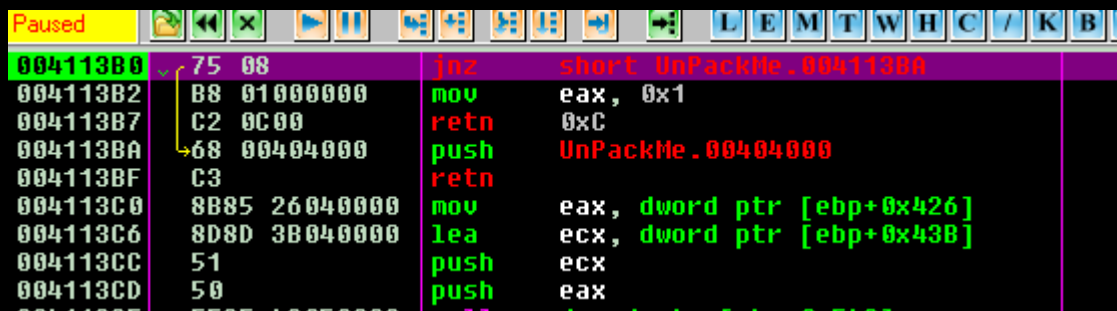
Trace qua lệnh **PUSHAD**, sau đó chọn thanh ghi **ESP**, chuột phải chọn **Follow in Dump**:



Tại cửa sổ Dump tiến hành đặt BP như sau:



Sau khi đặt BP xong nhấn **F9** để Run, OllyDBG sẽ break tại đây:



Ta đang dừng tại địa chỉ **0x004113B0**. Nhiều bạn sẽ thắc mắc là tại sao lại không thấy có lệnh nhảy nào tới OEP tương tự như ví dụ trước. Khả khả nếu bạn tinh ý, bạn sẽ thấy có cặp lệnh **PUSH** và **RETN** ở dưới, cặp lệnh này về bản chất sẽ tương tự như một lệnh nhảy. Để cụ thể hơn ta thực hiện trace qua lệnh **PUSH 404000** và lệnh **RETN**, OllyDBG sẽ đưa ta đến đây:

Address	Disassembly	Comment
00404000	wait	Real entry point
00404001	finit	
00404003	wait	
00404004	fclex	
00404006	fldcw word ptr [0x406000]	
0040400C	push ebp	
0040400D	mov ebp, esp	
0040400F	call UnPackMe.004043A5	
00404014	push 0x0	pModule = NULL
00404019	call dword ptr [0x4011F4]	GetModuleHandleA
0040401F	mov dword ptr [0x40F007], eax	
00404024	pushad	

Đó chính là OEP mà chúng ta cần tìm!

2.5. Phương pháp Exceptions

Trong trường hợp nếu ta gặp một trình packer tạo ra rất nhiều các exceptions trong quá trình unpack, vậy phải sử dụng phương pháp nào? Trong phần ví dụ này, tôi sẽ sử dụng một Unpackme là BitArts mà lão Ricardo cung cấp để demo.

Load Unpackme vào, sử dụng các plugins cần thiết để tránh việc Anti-OllyDBG. Cấu hình tab Exceptions tương tự như đã làm ở các ví dụ trên. Sau đó, cho thực thi unpackme trong OllyDBG cho tới khi nó run hoàn toàn như sau:

Address	Disassembly
0046B000	jmp short bitarts.0046B017
0046B002	add eax, dword ptr [eax]
0046B004	add byte ptr [eax], al
0046B006	push

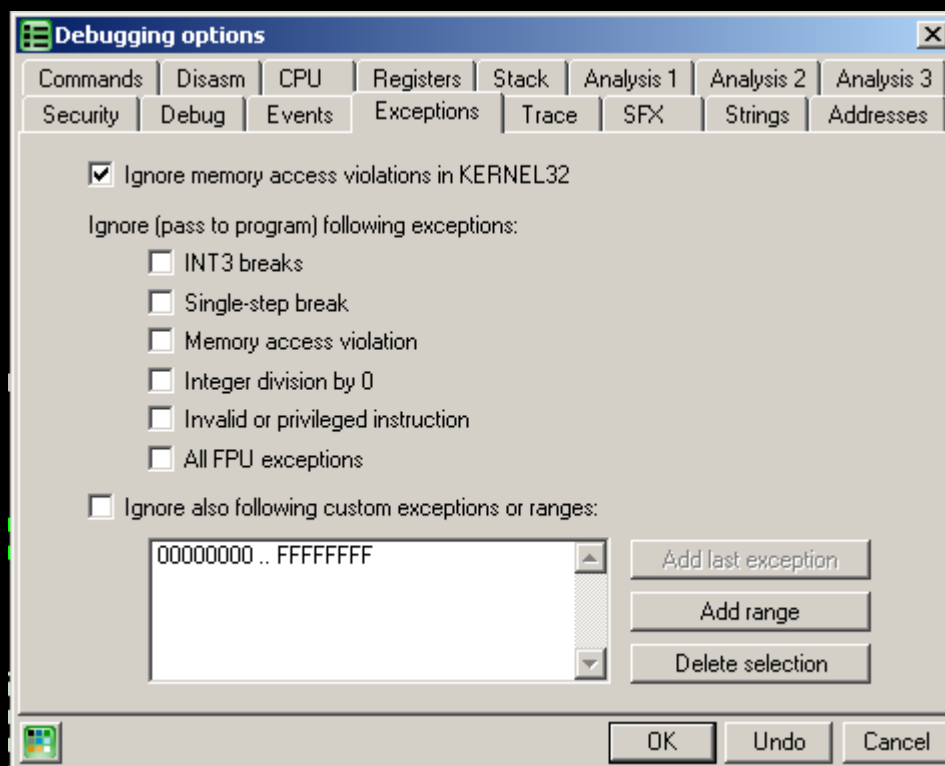
OK sau khi Unpackme đã run hoàn toàn, ta sẽ soi xem thông tin trong cửa sổ Log của OllyDBG có những gì. Nhấn **Alt + L** hoặc bấm vào biểu tượng **L** trên menu để tới cửa sổ **Log**, quan sát ta thấy như sau:

```

0046B000 Program entry point
>> Status: EP break deleted
0046F3F0 Access violation when reading [00000000]
7C810856 New thread with ID 0000055C created
0046EF14 Access violation when reading [00000000]
0046ECA1 Integer division by zero
00B2008E INT3 command at 00B2008E
0046EF14 Access violation when reading [00000000]
0046ECA1 Integer division by zero
00B4008E INT3 command at 00B4008E
0046EF14 Access violation when reading [00000000]
0046ECA1 Integer division by zero
00B6008E INT3 command at 00B6008E
0046EF14 Access violation when reading [00000000]
0046ECA1 Integer division by zero
00B8008E INT3 command at 00B8008E
0046EF14 Access violation when reading [00000000]
0046ECA1 Integer division by zero
0046E88F Thread 0000055C terminated, exit code 46FE79 (4652665.)
0046E88F INT3 command at bitarts_0046E88F
773D0000 Module C:\WINDOWS\WinSxS\x86_Microsoft.Windows.Common-Controls_6595b641
732E0000 Module C:\WINDOWS\system32\RICHED32.DLL

```

Ta chú ý tới exception cuối cùng đã xảy ra, địa chỉ **0x0046E88F** không thuộc vùng section đầu tiên. OK tạm thời có thông tin như vậy, khởi động lại OllyDBG và thiết lập tại các tùy chọn tại tab Exceptions tương tự như dưới đây:



Sau khi thiết lập xong quay trở lại màn hình code, nhấn **F9** để run. Mỗi lần OllyDBG break, ta nhấn **Shift + F9** để bypass:

Address	Disassembly	Comment
0046F3F0	8B00	mov eax, dword ptr [eax]
0046F3F2	64:8F05 000000	pop dword ptr fs:[0]
0046F3F9	83C4 04	add esp, 0x4
0046F3FC	C3	retn
0046F3FD	4D	dec ebp
0046F3FE	65:73 73	jnb short bitarts_.0046F474
0046F401	61	popad
0046F402	67:65:42	inc edx
0046F405	6F	outs dx, dword ptr es:[edi]

Superfluous prefix
Superfluous prefix
I/O command

Cho tới khi ta tới được exception cuối cùng, trong trường hợp này của ta là 46E88F.

Address	Disassembly	Comment
0046E88E	C3	retn
0046E88F	CC	int3
0046E890	E8 28000000	call bitarts_.0046E8B0
0046E895	44	inc esp
0046E896	12C3	adc al, b1
0046E898	0321	add esp, dword ptr [ecx]
0046E89A	64:8F05 000000	pop dword ptr fs:[0]
0046E8A1	E8 17000000	call bitarts_.0046E8BD
0046E8A6	DF12	fist word ptr [edx]
0046E8A8	C150 FF 83	rcl dword ptr [eax-0x1], 0x83
0046E8AC	C408	les ecx, fword ptr [eax]
0046E8AE	E8 00000000	call bitarts_.0046E8B0

Shift constant out of range 1..31
Modification of segment register

Nếu tiếp tục nhấn **Shift+F9** thì Unpackme sẽ thực thi. Vậy đây chính là exception cuối cùng được sinh ra bởi trình packer trước khi file được thực thi hoàn toàn.

Tại đây, ta có thể có tiến hành đặt một BPM On Access tại section code, và sẽ có rất nhiều bạn thắc mắc tại sao không đặt BP ngay từ lúc đầu, câu trả lời là bởi vì có nhiều trình packer có thể detect được BPM nếu như ta thiết lập ngay từ lúc đầu. OK, chuyển qua cửa sổ Memory, đặt BPM tại section code như sau:

Address	Disassembly	Section	PE header	Priv	RWE	RWE
003F0000	00001000	bitarts_				
00400000	00001000	bitarts_				
00401000	0000A000	bitarts_	.text	code	Imag	RWE
00408000	0000C000	bitarts_	.rdata			
00457000	00009000	bitarts_	.data			
00460000	00003000	bitarts_	.idata			
00463000	00008000	bitarts_	.rsrc			
0046B000	00004800	bitarts_	.edata			
004C0000	00002000					
00580000	00002000					
00590000	00103000					
006A0000	00005A00					
009A0000	00001000					
00A20000	00001000					
00B20000	00001000					
00B30000	00001000					
00B40000	00001000					
00B50000	00001000					
00B60000	00001000					
00B70000	00001000					

Actualize

View in Disassembler

Dump in CPU

Dump

Search

Set break-on-access

Set memory breakpoint on access

Set memory breakpoint on write

Set access

Copy to clipboard

Sort by

Appearance

Sau khi đặt BP xong, nhấn **Shift + F9**:

Address	Disassembly	Comment
004271B0	55	push ebp
004271B1	8BEC	mov ebp, esp
004271B3	6A FF	push -0x1
004271B5	68 600E4500	push bitarts_.00450E60
004271B8	68 C8924200	push bitarts_.004292C8
004271BF	64:A1 000000	mov eax, dword ptr fs:[0]
004271C5	50	push eax
004271C6	64:8925 0000	mov dword ptr fs:[0], esp
004271CD	83C4 A8	add esp, -0x58
004271D0	53	push ebx
004271D1	56	push esi
004271D2	57	push edi
004271D3	8965 E8	mov dword ptr [ebp-0x18], esp
004271D6	FF15 DC0A460	call dword ptr [0x460ADC]

SE handler installation

kernel32.GetVersion

Bùm, chúng ta đã dừng lại tại OEP 🤔.

Ngoài cách trên, ta cũng có thể áp dụng phương pháp sử dụng **SFX** để tìm OEP. Kết quả tìm kiếm cho ra kết quả tương tự như trên và khá nhanh:

Address	Disassembly	Comment
004271B0	55	push ebp
004271B1	8BEC	mov ebp, esp
004271B3	6A FF	push -0x1
004271B5	68 600E4500	push bitarts_.00450E60
004271B8	68 C8924200	push bitarts_.004292C8
004271BF	64:A1 000000	mov eax, dword ptr fs:[0]
004271C5	50	push eax
004271C6	64:8925 0000	mov dword ptr fs:[0], esp
004271CD	83C4 A8	add esp, -0x58
004271D0	53	push ebx

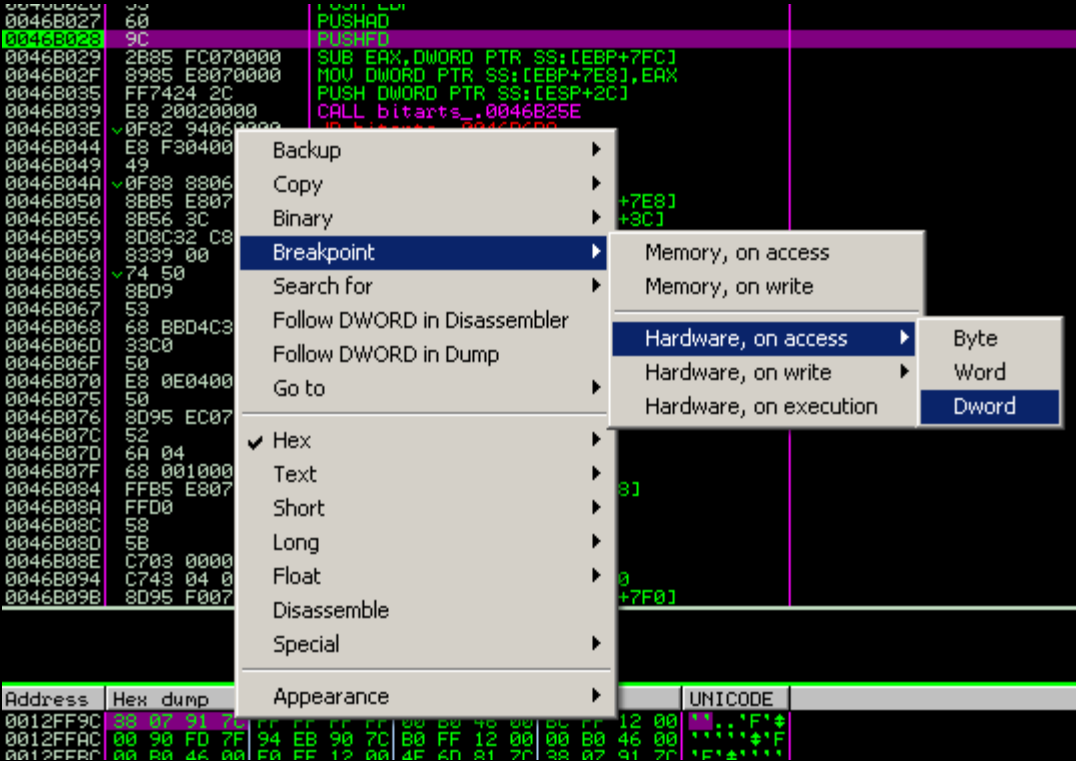
Real entry point of SFX code

SE handler installation

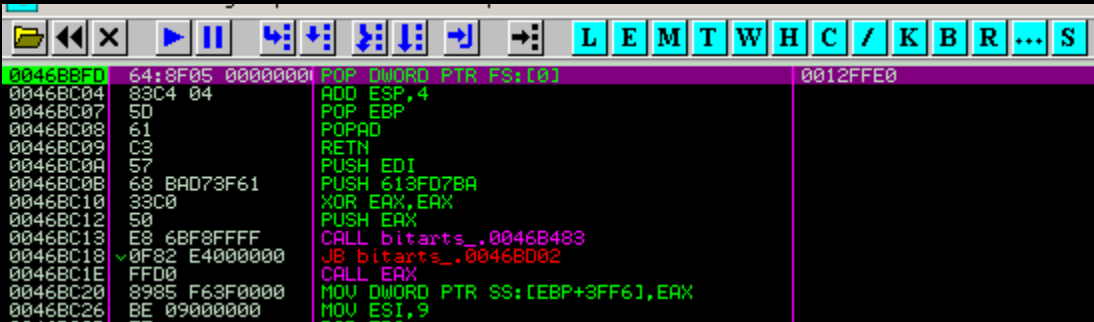
Thậm chí, ngay cả phương pháp **PUSHAD** cũng hoạt động rất hiệu quả. Ta thử load lại Unpackme và áp dụng phương pháp này:

Address	Disassembly	Comment
0046B000	EB 15	JMP SHORT bitarts_.0046B017
0046B002	0300	ADD EAX, DWORD PTR DS:[EAX]
0046B004	0000	ADD BYTE PTR DS:[EAX], AL
0046B006	06	PUSH ES
0046B007	0000	ADD BYTE PTR DS:[EAX], AL
0046B009	0000	ADD BYTE PTR DS:[EAX], AL
0046B00B	0000	ADD BYTE PTR DS:[EAX], AL
0046B00D	0000	ADD BYTE PTR DS:[EAX], AL
0046B00F	0000	ADD BYTE PTR DS:[EAX], AL
0046B011	0068 00	ADD BYTE PTR DS:[EAX], CH
0046B014	0000	ADD BYTE PTR DS:[EAX], AL
0046B016	0055 E8	ADD BYTE PTR SS:[EBP-18], DL
0046B019	0000	ADD BYTE PTR DS:[EAX], AL
0046B01B	0000	ADD BYTE PTR DS:[EAX], AL
0046B01D	5D	POP EBP
0046B01E	81ED 10000000	SUB EBP, 10
0046B024	8BC5	MOV EAX, EBP
0046B026	55	PUSH EBP
0046B027	60	PUSHAD
0046B028	9C	PUSHFD
0046B029	2B85 FC070000	SUB EAX, DWORD PTR SS:[EBP+7FC]
0046B02F	8985 E8070000	MOV DWORD PTR SS:[EBP+7E8], EAX

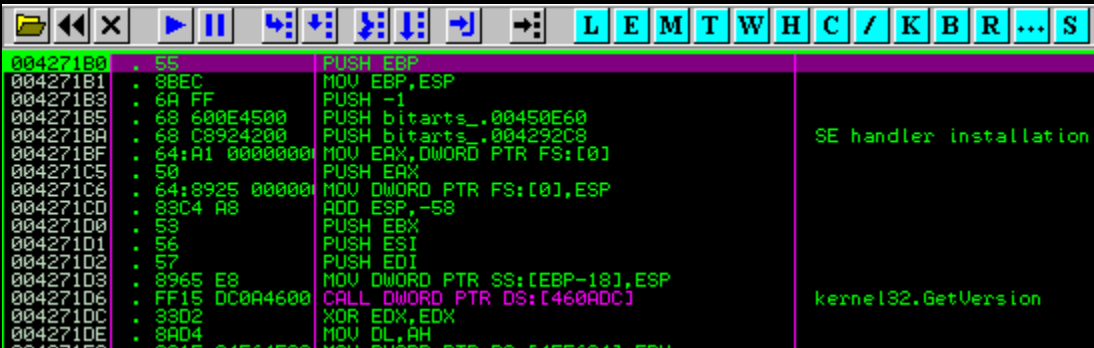
Trace qua lệnh **PUSHAD** tại địa chỉ **0x0046B027**. Lựa chọn thanh ghi **ESP** và Follow in Dump. Tiến hành đặt BP:



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



Thực hiện trace code và trace qua lệnh **RETN**, ta sẽ tới được OEP.

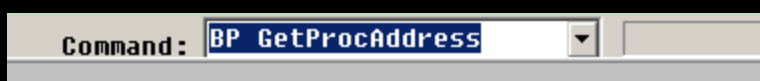


2.6. Phương pháp hàm API được sử dụng bởi Packer

Khởi động lại Unpackme BitArts trong OllyDBG, cấu hình exceptions như các ví dụ trước và tìm kiếm thông tin về các hàm API thông dụng nhất mà các trình packer hay dùng (tùy thuộc vào từng trình packer) là `GetProcAddress`, `LoadLibrary` hoặc `ExitThread` v.v.,. Trong phần này chúng ta sẽ thử sử dụng hàm `GetProcAddress`. Tại CommandBar ta tìm thông tin về địa chỉ hàm `GetProcAddress`:



Tiến hành đặt một BP tại hàm này:



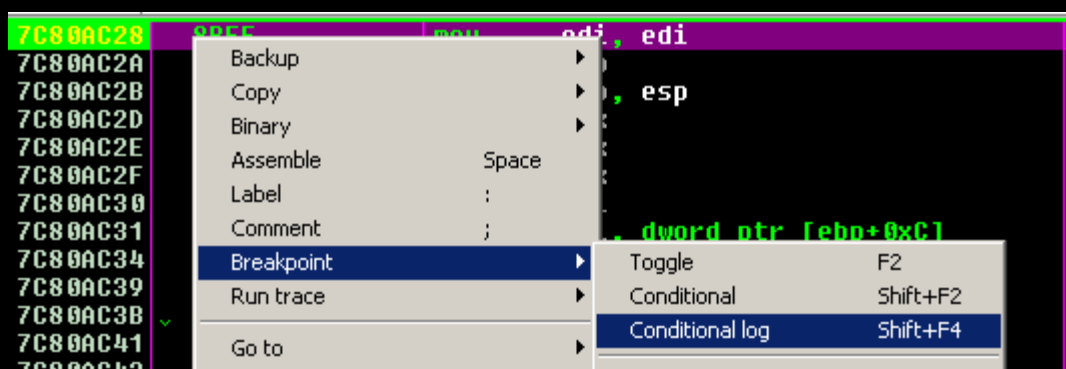
Kiểm tra lại BP vừa đặt tại cửa sổ BreakPoint:

PhantOm - [Br3akp0ints]			
File Vi3w D3bug Pluginz Options Window Help			
Paused [Icons]			
Address	Module	Active	Disassembly
7C80AC28	kerne132	Always	mov edi, edi

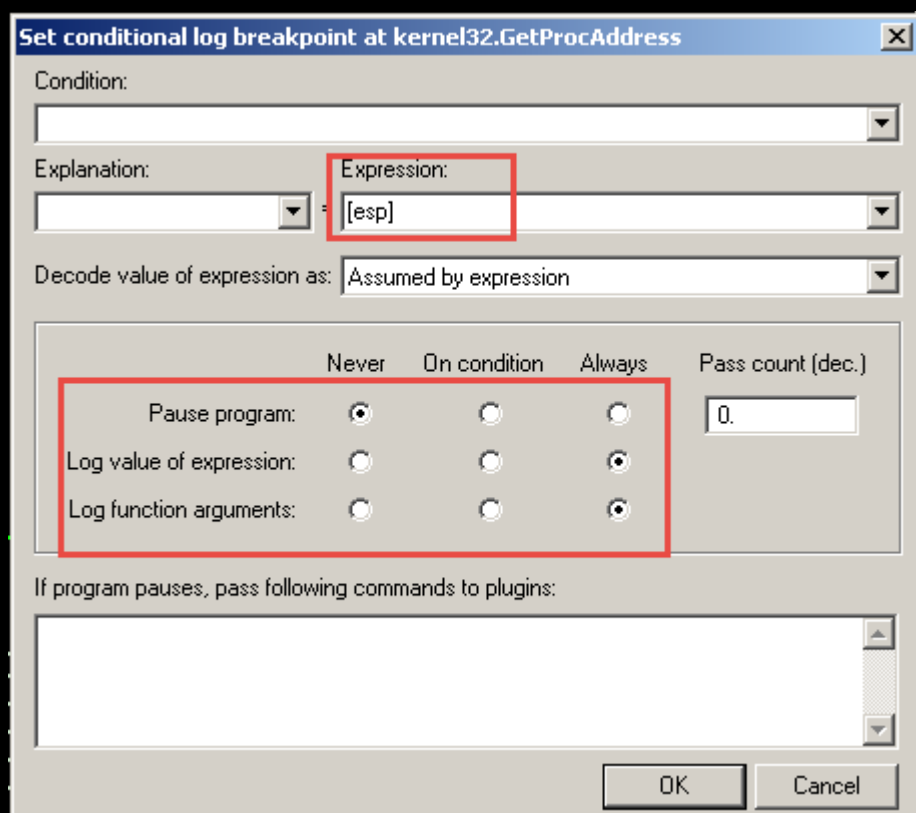
Nhấn **F9** để run, ta sẽ break tại hàm `GetProcAddress`:

Paused [Icons]			
7C80AC28	8BFF	mov	edi, edi
7C80AC2A	55	push	ebp
7C80AC2B	8BEC	mov	ebp, esp
7C80AC2D	51	push	ecx
7C80AC2E	51	push	ecx
7C80AC2F	53	push	ebx
7C80AC30	57	push	edi
7C80AC31	8B7D 0C	mov	edi, dword ptr [ebp+0xC]

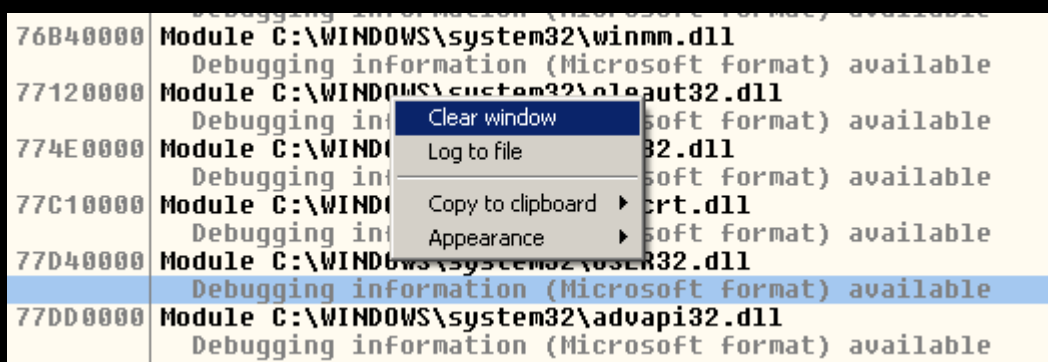
Tại đây ta thay đổi BP sang kiểu BP Conditional Log (không break, chỉ lưu Log). Chuột phải tại địa chỉ của hàm, chọn **Breakpoint > Conditional log**:



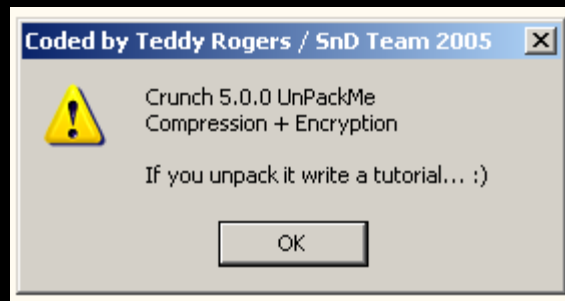
Đặt điều kiện của BP như sau:



Về conditional BP tôi đã có hẳn một phần viết về nó, bạn nào không nhớ thì đọc lại nhé. Với việc thiết lập như trên ta sẽ để cho Unpackme thực thi một cách bình thường, không dừng, nhưng sẽ luôn lưu lại log về lỗi cũng như log về giá trị của thanh ghi `[esp]`, mà cụ thể là giá trị trả về, mục đích để biết hàm API được gọi từ những đâu. Sau khi thiết lập xong, chuyển qua cửa sổ log và xóa hết các thông tin hiện có tại cửa sổ này:



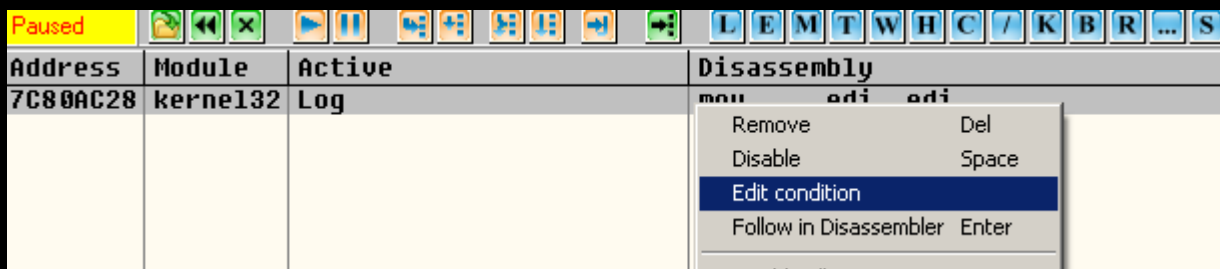
Sau đó, nhấn **F9** để thực thi unpackme trong OllyDBG cho tới khi file run hoàn toàn như sau:

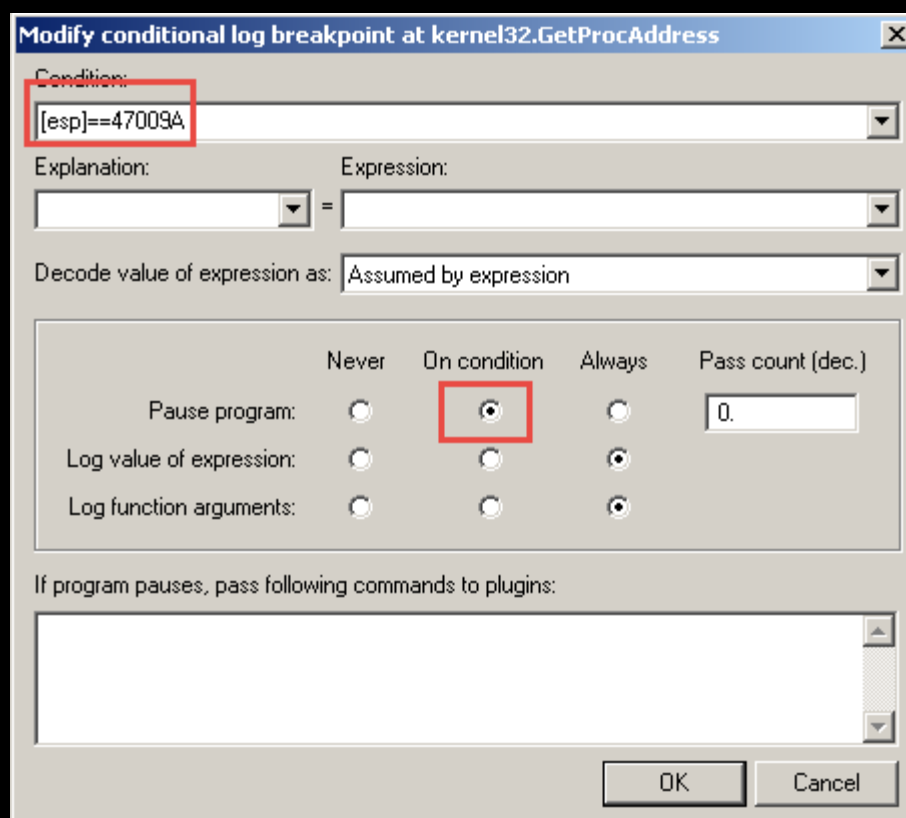


Tại cửa sổ Log lúc này ta sẽ thấy lời gọi cuối cùng tới hàm API `GetProcAddress` trước khi `Unpackme` run hoàn toàn sẽ phát sinh từ vùng code thuộc section đầu tiên (`[esp] = 00428C2B` thuộc section `.text`, nơi chứa OEP) và lời gọi hàm xuất phát từ địa chỉ `00428C25: CALL to GetProcAddress from bitarts .00428C25:`

```
7C80AC28 CALL to GetProcAddress
7C80AC28 COND: 0047009A
7C80AC28 CALL to GetProcAddress
7C80AC28 COND: 0047009A
7C80AC28 CALL to GetProcAddress
00B8008E INT3 command at 00B8008E
0046EF14 Access violation when reading [00000000]
0046ECA1 Integer division by zero
Thread 0000020C terminated, exit code 46FE79 (4652665.)
0046E88F INT3 command at bitarts_.0046E88F
7C80AC28 COND: 00428C2B
7C80AC28 CALL to GetProcAddress from bitarts_.00428C25
      hModule = 7C800000 (kernel32)
      ProcNameOrOrdinal = "IsProcessorFeaturePresent"
773D0000 Module C:\WINDOWS\WinSxS\x86_Microsoft.Windows.Common-Controls_6595b64144ccf1df_6.0.2600.5512_x-ww_317c0319-aab4-4bfb-bc6e-4b5e46ad6d2d_x-ww.mscommonctrl.dll
      Debugging information (Microsoft format) available
732E0000 Module C:\WINDOWS\system32\RICHED32.DLL
      Debugging information (Microsoft format) available
```

Trước **COND: 00428C2B** là **COND: 0047009A**, vậy ý tưởng đặt ra là ta sẽ break khi **[esp] == 47009A**, trước khi Unpackme thực thi hoàn toàn. Restart lại OllyDBG và sửa lại điều kiện như sau:





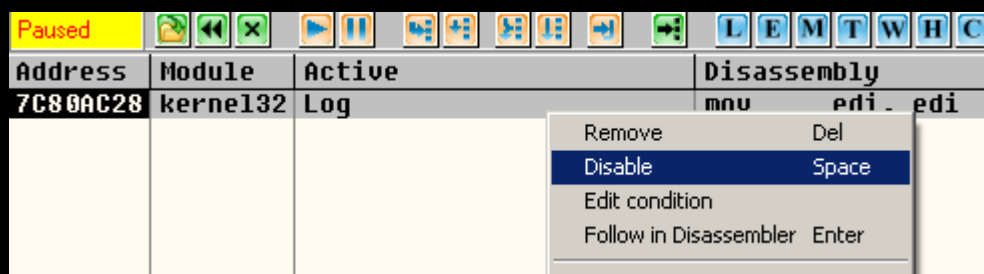
Sau khi thiết lập lại xong, nhấn **F9** để Run, ta sẽ dừng lại tại đây:

```

00B1FF54  0047009A  CALL to GetProcAddress from bitarts_.00470098
00B1FF58  76B40000  hModule = 76B40000 (winmm)
00B1FF5C  00460F2E  ProcNameOrOrdinal = "PlaySoundA"
00B1FF60  00460000  bitarts_.00460000
00B1FF64  0046B483  bitarts_.0046B483
00B1FF68  00400000  bitarts_.00400000
00B1FF6C  76B40000  winmm.76B40000

```

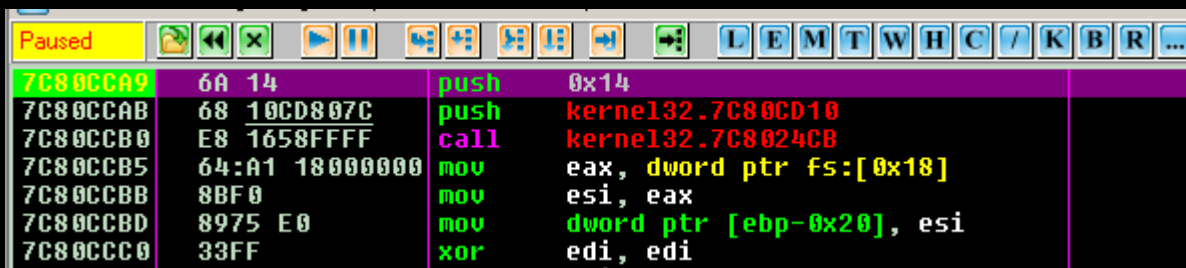
Chuyển qua cửa sổ BreakPoint, disable BP đã đặt:



Để tìm tới được OEP, ta tiến hành đặt một BPM On Access tại section chứa code. Chuyển qua cửa sổ Memory (**Alt + M**), chọn section .text và đặt BP như sau:

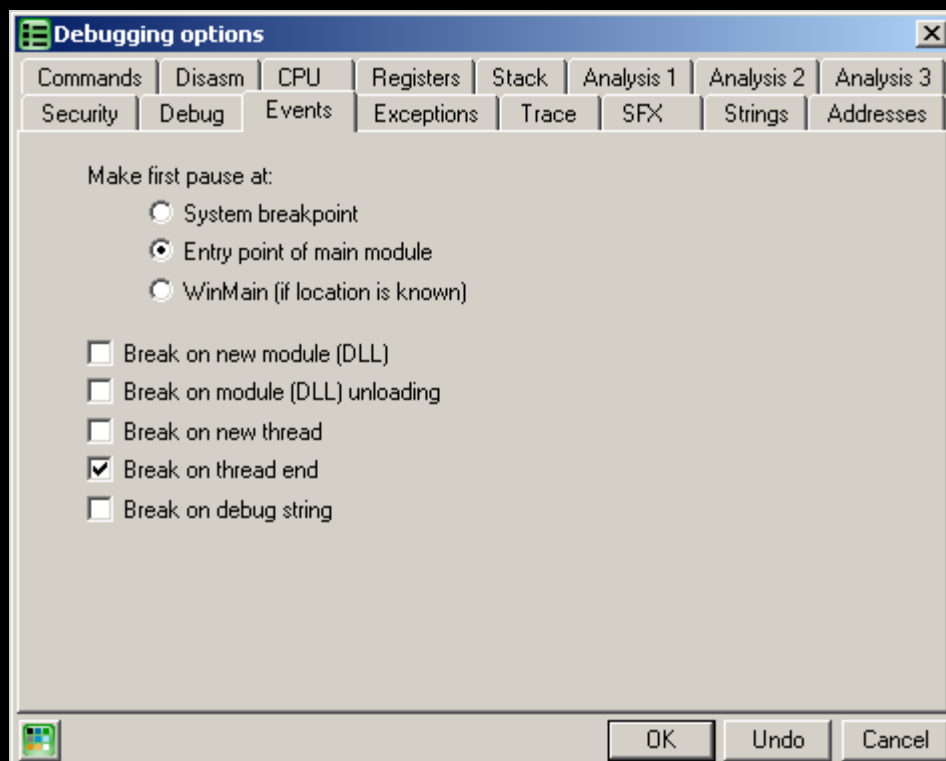


Nhấn F9 để Run, OllyDBG sẽ break tại hàm `ExitThread`:



Remove BP, chuyển qua cửa sổ Memory, đặt BPM on access tại section .text tương tự như trên. Sau đó nhấn **Shift + F9** ta cũng sẽ tới được OEP.

Cách thứ hai: Cấu hình lại OllyDBG như sau



Cấu hình xong, load lại Unpackme. Nhấn **F9** để run, ta sẽ break tại đây:

Address	Disassembly
7C90EB94	retn
7C90EB95	lea esp, dword ptr [esp]
7C90EB9C	lea esp, dword ptr [esp]
7C90EBA0	nop
7C90EBA1	nop
7C90EBA2	nop
7C90EBA3	nop

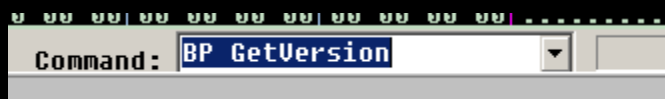
Tại đây, chuyển qua cửa sổ Memory, ta cũng đặt BPM on access tại section .text tương tự như trên. Sau đó nhấn **Shift + F9** ta sẽ tới OEP:

Address	Disassembly	Comment
004271B0	push ebp	
004271B1	mov ebp, esp	
004271B3	push -0x1	
004271B5	push bitarts_.00450E60	
004271BA	push bitarts_.004292C8	SE handler installation
004271BF	mov eax, dword ptr fs:[0]	
004271C5	push eax	
004271C6	mov dword ptr fs:[0], esp	
004271CD	add esp, -0x58	
004271D0	push ebx	
004271D1	push esi	
004271D2	push edi	
004271D3	mov dword ptr [ebp-0x18], esp	
004271D6	call dword ptr [0x460ADC]	kernel32.GetVersion
004271DC	xor edx, edx	

2.7. Phương pháp hàm API được gọi đầu tiên bởi chương trình

Mục tiêu của phương pháp này là đặt BP trực tiếp lên một hàm API được nghi ngờ là sẽ thực thi đầu tiên khi chương trình chạy. Thông thường, một chương trình khởi động nó sẽ gọi tới các hàm API như `GetVersion`, `GetModuleHandleA`, quan sát và làm việc với nhiều file bị pack thì các bạn sẽ thấy các hàm API được gọi đầu tiên sẽ không nhiều, trong trường hợp các file chúng ta đang sử dụng làm ví dụ thì BitArts gọi hàm `GetVersion` đầu tiên, Cruehead gọi hàm `GetModuleHandleA` đầu tiên. Để minh họa cho phương pháp này ta sẽ thực hành với hàm `GetVersion`.

Load BitArts vào OllyDBG, đặt BP như sau:



Đặt BP xong, nhấn F9 để Run:

Address	Disassembly
7C8114AB	mov eax, dword ptr fs:[0x18]
7C8114B1	mov ecx, dword ptr [eax+0x30]
7C8114B4	mov eax, dword ptr [ecx+0xB0]
7C8114BA	movzx edx, word ptr [ecx+0xAC]


```

0012FF48 004271DC RETURN to bitarts_.004271DC from kernel32.GetVersion
0012FF4C 006A0118
0012FF50 0012B9A8
0012FF54 004688F5 bitarts_.004688F5
0012FF58 01050104

```

Phải chắc chắn rằng lệnh call tới API được gọi bởi chương trình mà ta đang debug, hoặc từ section đầu tiên (thường là từ section code). Quan sát trên cửa sổ Stack ta thấy được địa chỉ trả về sau lời gọi hàm `GetVersion`, và địa chỉ trả về này thuộc section `.text` (code). Ta đi tới địa chỉ này như sau:

```

0012FF48 004271DC RETURN to bitarts_.004271DC from kernel32.GetVersion
0012FF4C 006A0118
0012FF50 0012B9A8

```

```

004271B0 . 55      push    ebp
004271B1 . 8BEC    mov     ebp, esp
004271B3 . 6AFF    push    -0x1
004271B5 . 68 60E45000 push    bitarts_.00450E60
004271B8 . 68 C8924200 push    bitarts_.004292C8
004271BF . 64:A1 000000 mov     eax, dword ptr fs:[0]
004271C5 . 50      push    eax
004271C6 . 64:8925 0000 mov     dword ptr fs:[0], esp
004271CD . 83C4 A8  add     esp, -0x58
004271D0 . 53      push    ebx
004271D1 . 56      push    esi
004271D2 . 57      push    edi
004271D3 . 8965 E8  mov     dword ptr [ebp-0x18], esp
004271D6 . FF15 DC0A460 call    dword ptr [0x460ADC]
004271DC . 33D2    xor     edx, edx
004271DE . 8AD4    mov     dl, ah
004271E0 . 8915 34E6450 mov     dword ptr [0x45E634], edx
004271E6 . 8BC8    mov     ecx, eax

```

SE handler installation

kernel32.GetVersion

Quan sát trên hình ta sẽ thấy được OEP (giống như đã tìm thông qua các phương pháp trên), trong trường hợp Packer áp dụng cơ chế detect khi ta đặt BP tại `GetVersion`, ta có thể chuyển hướng sang đặt BP tại lệnh `RET` của hàm API để lách 🤪.

III. Kết luận

Trên đây là những ví dụ về một số phương pháp được dùng khá phổ biến khi tìm OEP, sẽ có nhiều phương pháp khác nữa tùy theo kinh nghiệm của từng người. Khi bạn đã có được một nền tảng vững chắc thì phương pháp làm thế nào là do bạn, nó không còn phụ thuộc cố định vào một cách thức/phương pháp nào hết.

Ngoài ra, lão Ricardo có đính kèm một **UnPackMe_tElock0.98.exe** để phục vụ cho các bạn thực hành tìm OEP. Unpackme này theo giới thiệu sẽ có một vài trick mà bạn chưa thấy ở các phương pháp trên. Các bạn thử xem sao nhé! Và tất nhiên, phần tiếp theo của bài viết sẽ được đặt pass chính là địa chỉ OEP mà các bạn tìm được ở **UnPackMe_tElock0.98.exe**. Toàn bộ phần 26 đến đây là kết thúc, cảm ơn các bạn đã dành thời gian để theo dõi. Hẹn gặp lại ở bạn ở các phần tiếp theo với các chủ đề liên quan đến việc Dump file và Repair IATs 🤪.

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, HQQCrker, the_Lighthouse, Merc, Hoadongnoi, Nini ... all REA's members, TQN, HacNho, RongChauA, Deux, tlandn, light.phoenix, dqtn, ARTEAM ... all my friend, and YOU.

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

iamidiot, WhyNotBar, trickyboy, dzungltn, takada, hurt_heart, haule_nth, hytkl, moth, XIANUA, nhc1987, 0xdie, Unregistered!, akira, mrangle 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