

# 2017

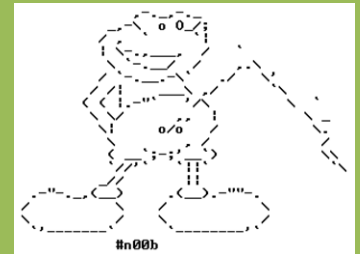
## [Cracking with OllyDbg]

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



[www.reasonline.net](http://www.reasonline.net)

kienmanowar



16/12/2017

## Mục Lục

I. Giới thiệu chung.....	2
II. Phân tích và xử lý target.....	2
1. Sử dụng tính năng trace code của OllyDbg.....	3
2. Sử dụng plugin viết riêng cho từng packer .....	24
3. Tìm magic jump .....	25
3.1. Theo dõi các lệnh nhảy của bad entry .....	31
3.2. Theo dõi các lệnh nhảy của good entry .....	34
III. Kết luận .....	42

## I. Giới thiệu chung

Ở phần trước, thông qua [UnPackMe\\_tElock0.98.exe](#), tôi đã giới thiệu với các bạn về kĩ thuật **IAT Redirection**, một kĩ thuật rất hay gặp ở các packers/protectors. Trong phần 31 này, tôi sẽ áp dụng một số phương pháp fix IAT, để làm sao khi ImpREC thực hiện Get Imports thì thông tin về hàm API thu được sẽ đầy đủ nhất phục vụ việc fix dump. Đảm bảo cho file sau khi fix chạy mượt mà, không lỗi.

Cũng tương tự như phần trình bày các phương pháp làm thế nào để tới được OEP, ở phần này tôi cũng sẽ áp dụng một số phương pháp tổng quát nhất, để sau đó, khi chúng ta gặp các trình packers khác, ta sẽ tùy biến các phương pháp này hoặc nghiên cứu một cách thức hoàn toàn mới nhằm phù hợp với tính huống thực tế mà ta đang gặp phải, có thể chưa được đề cập đến trong bài viết này.

## II. Phân tích và xử lý target

Để tiện theo dõi, tôi đưa lại các thông tin về OEP, IAT Start, IAT Size đã tìm được ở phần trước như sau:

- OEP= 0x271B0
- RVA (IAT Start) = 0x60818
- IAT Size= 0x710

Trong phần tiếp theo này, ta sẽ cùng nhau xem xét một số phương pháp khả thi nhằm khôi phục lại các entry của IAT đã bị chuyển hướng (redirected). Tuy nhiên, vấn đề sẽ không chỉ dừng lại ở riêng bài viết này, bởi khi tới được OEP, các trình packer khác nhau có thể áp dụng các phương pháp bảo vệ khác nhau. Do đó, khi gặp các trình packer khác thì ta phải tìm các phương pháp để tới được OEP cũng như cách để khôi phục lại bảng IAT một cách đầy đủ nhất.

Các phương pháp mà tôi sắp trình bày bên dưới đây xuất phát từ những ý tưởng chung, nên nó có thể áp dụng được với packer này mà không áp dụng được với packer khác, điều này là hoàn toàn bình thường. Vì vậy, ta cần thích ứng để điều chỉnh cách làm cho phù hợp theo từng trường hợp. Lời khuyên tốt nhất là luôn luôn tìm hiểu và thực hành nhiều hơn, liên tục thay đổi và thử nghiệm để đúc rút được những kinh nghiệm riêng.

Ba phương pháp mà tôi trình bày trong phần này bao gồm:

- Sử dụng tính năng trace code của OllyDbg.
- Sử dụng plugin viết riêng cho từng packer.

- Tìm tử huyệt – magic jump.

## 1. Sử dụng tính năng trace code của OllyDbg

```

004271B0  55          push     ebp
004271B1  8BEC        mov     ebp, esp
004271B3  6A FF       push     -1
004271B5  68 600E4500 push     UnPackMe.00450E60
004271BA  68 C8924200 push     UnPackMe.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, -58
004271D0  53          push     ebx
004271D1  56          push     esi
004271D2  57          push     edi
004271D3  8965 E8     mov     dword ptr [ebp-18], esp
004271D6  FF15 DC0A460 call    near dword ptr [460ADC]
004271DC  33D2        xor     edx, edx
  
```

OK, thực hiện lại các bước trong phần trước, ta đang dừng tại OEP của unpackme như trên hình. Tại đây, ta đã phân tích có các entries của bảng IAT đã bị chuyển hướng tới một số section được tạo ra bởi chính trình packer tElock trong quá trình nó thực hiện unpack chương trình.

Address	Hex dump	ASCII
0046080C	00 00 00 00 00 00 00 80 00 00 00 00 F0 6B DD 77	.....8kYw
0046081C	1B 76 DD 77 F4 EA DD 77 E7 EB DD 77 83 78 DD 77	+vYwôeYwçeYw xYw
0046082C	00 00 00 00 00 DD 15 0B 5D 2E BD 09 5D 00 00 00 00	...Yr].%]....
0046083C	00 00 A0 00 11 00 A0 00 22 00 A0 00 33 00 A0 00	...3...
0046084C	41 00 A0 00 50 00 A0 00 5F 00 A0 00 7F 00 A0 00	A...P...I...
0046085C	8D 00 A0 00 B0 00 A0 00 C1 00 A0 00 D2 00 A0 00	I...A...O...
0046086C	E4 00 A0 00 F4 00 A0 00 05 01 A0 00 24 01 A0 00	ă...ô...lr...\$r...
0046087C	3E 01 A0 00 4F 01 A0 00 60 01 A0 00 71 01 A0 00	>r...Or...qr...
0046088C	7F 01 A0 00 8E 01 A0 00 9D 01 A0 00 BD 01 A0 00	lr...lr...kr...
0046089C	CB 01 A0 00 EE 01 A0 00 FF 01 A0 00 10 02 A0 00	Er...ir...ÿr...hr...
004608AC	22 02 A0 00 32 02 A0 00 43 02 A0 00 62 02 A0 00	"...2...C...B...
004608BC	7C 02 A0 00 8D 02 A0 00 9E 02 A0 00 AF 02 A0 00	h...h...h...r...
004608CC	BD 02 A0 00 CC 02 A0 00 DB 02 A0 00 FB 02 A0 00	%...î...ô...ô...
004608DC	09 03 A0 00 2C 03 A0 00 3D 03 A0 00 4E 03 A0 00	...l...l...=...N...
004608EC	60 03 A0 00 70 03 A0 00 81 03 A0 00 A0 03 A0 00	...p...l...l...
004608FC	BA 03 A0 00 CB 03 A0 00 DC 03 A0 00 ED 03 A0 00	ê...É...Û...î...
0046090C	FB 03 A0 00 0A 04 A0 00 19 04 A0 00 39 04 A0 00	û...j...P...9...
0046091C	47 04 A0 00 6A 04 A0 00 7B 04 A0 00 8C 04 A0 00	G...j...{...P...
0046092C	9E 04 A0 00 AE 04 A0 00 BF 04 A0 00 DE 04 A0 00	P...@...@...P...
0046093C	F8 04 A0 00 09 05 A0 00 1A 05 A0 00 2B 05 A0 00	ø...l...+...l...
0046094C	39 05 A0 00 48 05 A0 00 57 05 A0 00 77 05 A0 00	9l...Hl...Wl...wl...
0046095C	85 05 A0 00 A8 05 A0 00 B9 05 A0 00 CA 05 A0 00	ll...l...l...Êl...
0046096C	DC 05 A0 00 EC 05 A0 00 00 00 9E 00 11 00 9E 00	Û...l...l...l...
0046097C	22 00 9E 00 33 00 9E 00 41 00 9E 00 50 00 9E 00	"...3...A...P...
0046098C	5F 00 9E 00 7F 00 9E 00 8D 00 9E 00 B0 00 9E 00	...l...l...l...l...
0046099C	C1 00 9E 00 D2 00 9E 00 E4 00 9E 00 F4 00 9E 00	Ă...l...ô...l...ô...
004609AC	05 01 9E 00 24 01 9E 00 3E 01 9E 00 4F 01 9E 00	lr...\$r...>r...Or...

Như quan sát các giá trị của bảng IAT tại cửa sổ Dump trên hình, sẽ thấy có các vùng nhỏ mà trình packer chuyển hướng tới để từ đó khôi phục lại IAT gốc (ví dụ, như tại

máy tôi là các vùng nhớ bắt đầu bằng `0x00A0xxxx`; `0x009Exxxx...`). Do vậy, ta cần phải biết các phương pháp để khôi phục lại các entry đã bị điều hướng này, đồng nghĩa với việc ta phải biết làm thế nào để tìm ra hàm API tương ứng với từng entry.

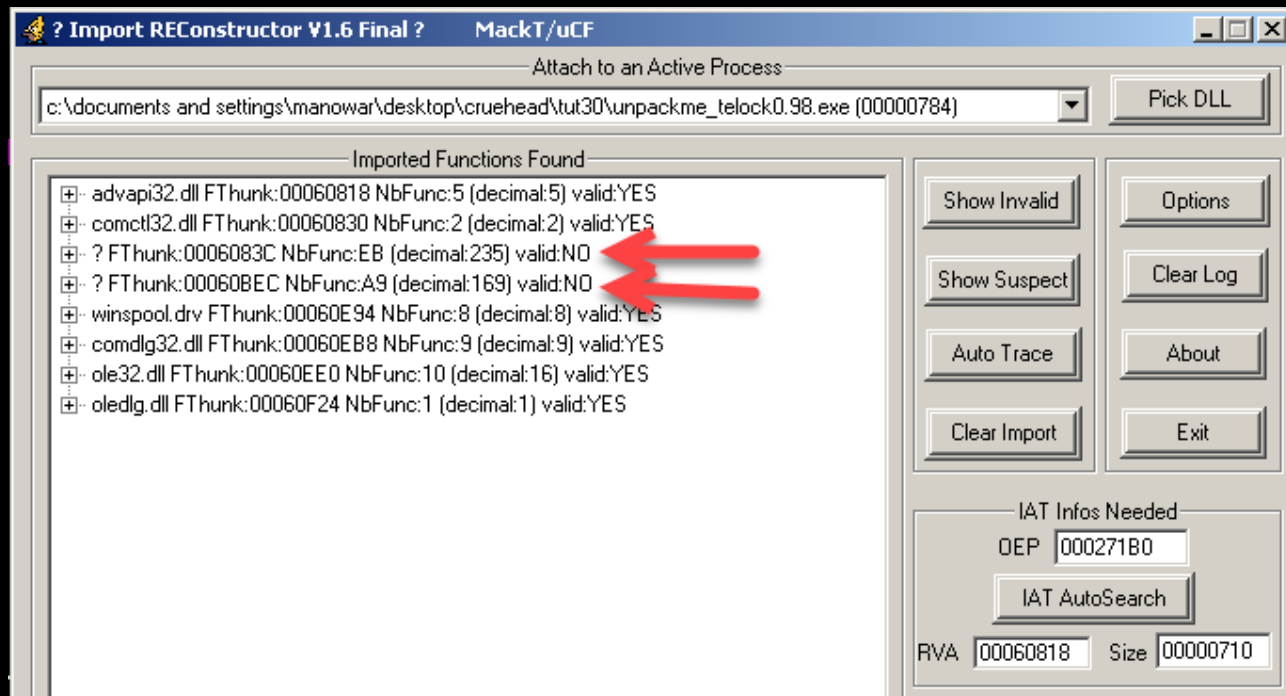
Tôi sẽ lấy luôn địa chỉ tại lệnh call đầu tiên mà đã thực hiện tìm kiếm theo cách thủ công ở phần trước, vì tại đó tôi đã biết chắc chắn là nó sẽ gọi tới hàm API `GetVersion()`.

004271D3	8965 E8	mov	dword ptr [ebp-18],esp	
004271D6	FF15 DC0A460	call	near dword ptr [460ADC]	
004271DC	33D2	xor	edx,edx	
004271DE	8AD4	mov	dl,ah	
004271E0	8915 34E6450	mov	dword ptr [45E634],edx	
004271E6	8BC8	mov	ecx,eax	
004271E8	81E1 FF00000	and	ecx,0FF	
004271EE	890D 30E6450	mov	dword ptr [45E630],ecx	
004271F4	C1E1 08	shl	ecx,8	
004271F7	03CA	add	ecx,edx	
004271F9	890D 2CE6450	mov	dword ptr [45E62C],ecx	
004271FF	C1E8 10	shr	eax,10	
00427202	A3 28E64500	mov	dword ptr [45E628],eax	
00427207	E8 94210000	call	UnPackMe.004293A0	
0042720C	85C0	test	eax,eax	
0042720E	75 0A	jnz	short UnPackMe.0042721A	
00427210	6A 1C	push	1C	
00427212	E8 49010000	call	UnPackMe.00427360	
00427217	83C4 04	add	esp,4	
0042721A	E8 D12F0000	call	UnPackMe.0042A1F0	
0042721F	85C0	test	eax,eax	

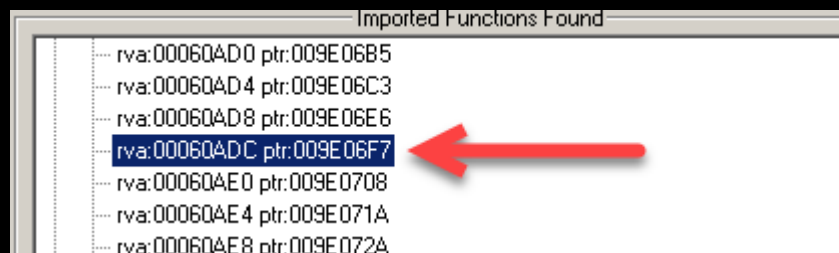
ds:[00460ADC]=009E06F7

Address	Value	Comment
00460ADC	009E06F7	
00460AE0	009E0708	
00460AE4	009E071A	

Tại lệnh Call, địa chỉ `0x460ADC` đang lưu giá trị (trên máy tính của tôi) là `0x9E06F7`. Cùng với đó tại ImpREC, sau khi điền tất cả các giá trị liên quan bao gồm OEP, RVA và SIZE của bảng IAT, ta sẽ có toàn bộ IAT kèm theo đó là cả các giá trị invalid IATs.



Tại ImpREC, tôi địa chỉ RVA là **0x60ADC** (Giá trị RVA: 0x60ADC tương ứng với mục IAT tại 0x460ADC (VA)) ta cũng sẽ thấy giá trị tương ứng như đã thấy ở OllyDbg:

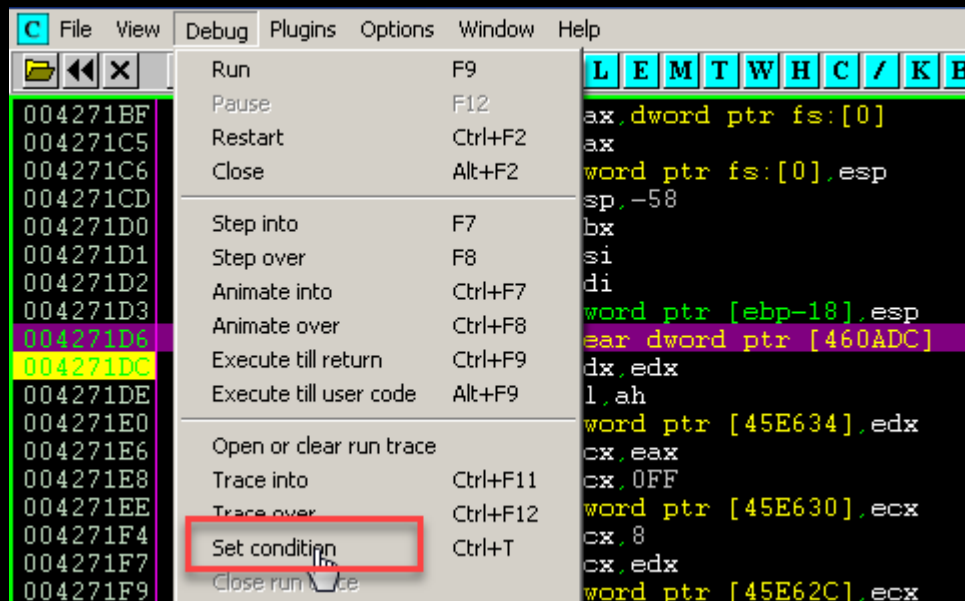


Quay lại OllyDbg, theo như phần trước, bằng cách trace code thủ công ta sẽ tới được hàm API cần tìm. Tuy nhiên, nếu làm như thế cho từng lệnh Call thì sẽ rất nản. Bản thân OllyDbg đã tích hợp sẵn tính năng trace code, ta sẽ áp dụng tính năng này để thực hiện nhanh hơn. Với packer tElock, ta thấy chỉ qua có 5 hoặc 6 dòng code là đã tìm ra và tới được hàm API, tuy nhiên có những trình packers sẽ cho chúng ta đi lòng vòng trước khi đến được đích cuối cùng. Chính vì vậy, sử dụng tính năng trace code tự động của OllyDbg sẽ giúp ích rất nhiều.

Trước khi sử dụng tính năng trace code, tôi đặt một BP tại lệnh bên dưới lệnh call tới hàm API. Mục đích để đề phòng trường hợp làm sai hoặc quá trình trace code không có điểm dừng. Và cơ bản, đây sẽ là lệnh tiếp theo được thực thi sau khi trở về từ lời gọi hàm API.

004271D6	FF15 DC0A460	call	near dword ptr [460ADC]
004271DC	33D2	xor	edx,edx
004271DE	8AD4	mov	dl,ah
004271E0	8915 34E6450	mov	dword ptr [45E634],edx

Sau đó, tại lệnh `004271D6 . FF15 DC0A4600 call near dword ptr [460ADC]`, tôi sẽ thiết lập điều kiện để trace code bằng cách chọn: **Debug > Set Condition** (hoặc chuột phải và chọn **Run trace > Set condition**).



Tại đây, ta thấy có rất nhiều các lựa chọn cho phép thiết lập để OllyDbg dừng quá trình trace code khi thỏa mãn điều kiện:

Condition to pause run trace

Pause run trace when any checked condition is met:

- ☐ EIP is in range 00000000 ... 00000000
- ☐ EIP is outside the range 00000000 ... 00000000
- ☐ Condition is TRUE
- ☐ Command is suspicious or possibly invalid
- ☐ Command count is 0. (actual 0. ) Reset
- ☐ Command is one of

In command, R8, R32, RA, RB and CONST match any register or constant

OK Cancel

Tại màn hình **"Condition to pause run trace"** trên, ta sẽ thấy một số tùy chọn để cấu hình và có thể được điều chỉnh để phù hợp theo từng trường hợp. Với tùy chọn **"EIP is in range"**, có nghĩa là nếu ta cấu hình giá trị tại đây, OllyDbg sẽ dừng lại khi EIP nằm trong phạm vi vùng giá trị mà chúng ta thiết lập ở phía bên phải. Ví dụ như sau:

Condition to pause run trace

Pause run trace when any checked condition is met:

- ☒ EIP is in range 00401000 ... 00500000
- ☐ EIP is outside the range 00000000 ... 00000000
- ☐ Condition is TRUE
- ☐ Command is suspicious or possibly invalid
- ☐ Command count is 0. (actual 0. ) Reset
- ☐ Command is one of

In command, R8, R32, RA, RB and CONST match any register or constant

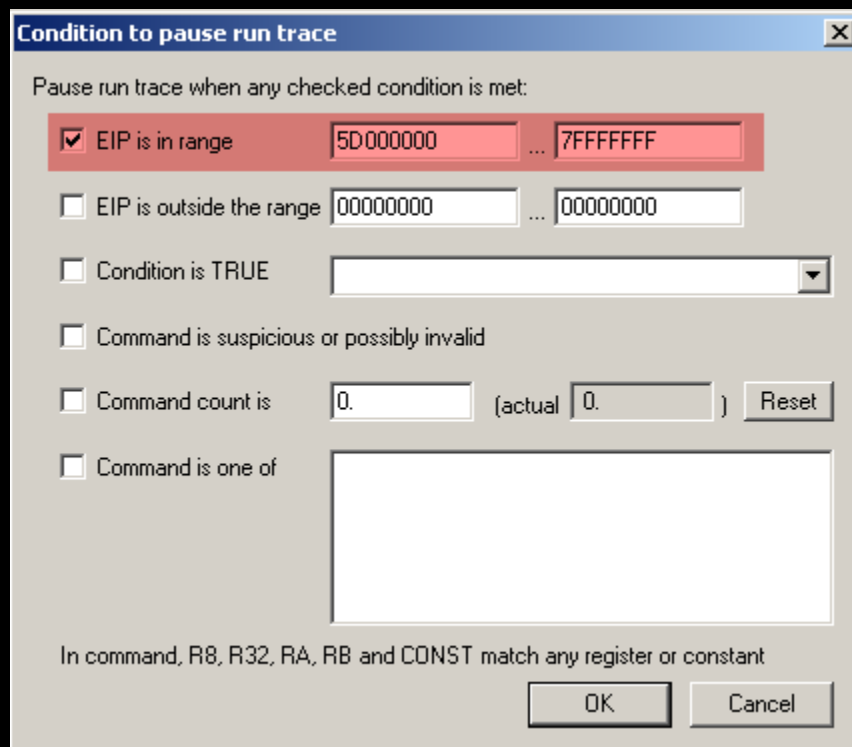
OK Cancel



Trong ví dụ trên hình, OllyDbg sẽ trace code cho đến khi ta thấy EIP là một giá trị nằm giữa khoảng từ 401000 đến 500000. Đây chỉ là một ví dụ minh họa để các bạn hiểu thôi, nó sẽ không có ý nghĩa trong trường hợp này, vì ở đây chúng ta muốn nó dừng lại tại hàm API. Để tìm khoảng giá trị mà ta muốn OllyDbg dừng lại trong trường hợp của packer tElock, tôi chuyển qua cửa sổ **Memory** và quan sát các sections tại cửa sổ này:

00400000	00001000	UnPackMe		PE header	Imag	RW	RWE
00401000	0004A000	UnPackMe	.teddy	code	Imag	RW	RWE
0044B000	0000C000	UnPackMe	.teddy	data	Imag	RW	RWE
00457000	00009000	UnPackMe	.teddy		Imag	RW	RWE
00460000	00003000	UnPackMe	.teddy		Imag	RW	RWE
00463000	00002000	UnPackMe	.rsrc	resources	Imag	RW	RWE
00465000	00004000	UnPackMe	.teddy	SFX,imports	Imag	RW	RWE
00470000	00002000				Map	R E	R E
00530000	00002000				Map	R E	R E
00540000	00103000				Map	R	R
00650000	00035000				Map	R E	R E
00960000	00001000				Priv	RW	RW
009E0000	00002000				Priv	RW	RW
009F0000	00002000				Priv	RW	RW
00A00000	00001000				Priv	RW	RW
00A10000	00004000				Priv	RW	RW
00A20000	00003000				Priv	RW	RW
00A30000	00002000				Map	R	R
00A40000	00001000				Priv	RW	RW
01220000	00002000				Map	R	R
5D090000	00001000	COMCTL32		PE header	Imag	R	RWE
5D091000	00070000	COMCTL32	.text	code,imports	Imag	R	RWE
5D101000	00003000	COMCTL32	.data	data	Imag	R	RWE
5D104000	0001F000	COMCTL32	.rsrc	resources	Imag	R	RWE
5D123000	00004000	COMCTL32	.reloc	relocations	Imag	R	RWE
629C0000	00001000	LPK		PE header	Imag	R	RWE
629C1000	00005000	LPK	.text	code,imports	Imag	R	RWE
629C6000	00001000	LPK	.data	data	Imag	R	RWE
629C7000	00001000	LPK	.rsrc	resources	Imag	R	RWE
629C8000	00001000	LPK	.reloc	relocations	Imag	R	RWE
73000000	00001000	WINSPOOL		PE header	Imag	R	RWE
73001000	00020000	WINSPOOL	.text	code,imports	Imag	R	RWE
73021000	00002000	WINSPOOL	.data	data	Imag	R	RWE
73023000	00001000	WINSPOOL	.rsrc	resources	Imag	R	RWE
73024000	00002000	WINSPOOL	.reloc	relocations	Imag	R	RWE
74D30000	00001000	oledlg		PE header	Imag	R	RWE
74D31000	00011000	oledlg	.text	code,imports	Imag	R	RWE
74D42000	00002000	oledlg	.data	data	Imag	R	RWE
74D44000	0000B000	oledlg	.rsrc	resources	Imag	R	RWE
74D4F000	00001000	oledlg	.reloc	relocations	Imag	R	RWE
74D90000	00001000	USP10		PE header	Imag	R	RWE
74D91000	00044000	USP10	.text	code,imports	Imag	R	RWE
74DD5000	00010000	USP10	.data	data	Imag	R	RWE
74DE5000	00002000	USP10	Shared		Imag	R	RWE
74DE7000	00012000	USP10	.rsrc	resources	Imag	R	RWE
74DF9000	00002000	USP10	.reloc	relocations	Imag	R	RWE
76390000	00001000	IMM32		PE header	Imag	R	RWE
76391000	00015000	IMM32	.text	code,imports	Imag	R	RWE
763A6000	00001000	IMM32	.data	data	Imag	R	RWE
763A7000	00005000	IMM32	.rsrc	resources	Imag	R	RWE
763AC000	00001000	IMM32	.reloc	relocations	Imag	R	RWE
763B0000	00001000	comdlg32		PE header	Imag	R	RWE
763B1000	00030000	comdlg32	.text	code,imports	Imag	R	RWE
763E1000	00004000	comdlg32	.data	data	Imag	R	RWE
763E5000	00011000	comdlg32	.rsrc	resources	Imag	R	RWE
763F6000	00003000	comdlg32	.reloc	relocations	Imag	R	RWE
76B40000	00001000	WINMM		PE header	Imag	R	RWE
76B41000	0001F000	WINMM	.text	code,imports	Imag	R	RWE
76B60000	00002000	WINMM	.data	data	Imag	R	RWE
76B62000	00009000	WINMM	.rsrc	resources	Imag	R	RWE
76B6B000	00002000	WINMM	.reloc	relocations	Imag	R	RWE
77120000	00001000	OLEAUT32		PE header	Imag	R	RWE

Tại cửa sổ **Memory**, vùng được tôi đánh dấu màu xanh như trên hình chính là vùng mà chúng ta cần quan tâm để thiết lập điều kiện dừng cho OllyDbg khi tiến hành trace code, hay nói nôm na đó là nơi mà DLL đầu tiên được load lên. Ở đây, tôi không muốn dừng tại các sections của unpackme, hoặc tại các sections được tạo ra bởi packer vì đó là các đoạn mã trung gian, cái tôi cần là dừng lại tại các DLLs. Do đó, tôi lựa chọn tùy chọn đầu tiên (**EIP is in range**) và cấu hình giá trị của EIP nằm trong khoảng từ 5D000000 tới 7FFFFFFF – là địa chỉ cao nhất chắc chắn để dừng lại ở một số DLL (Cần làm rõ một chút ở đây, không quá quan trọng khi ta không chọn địa chỉ chính xác của DLL đầu tiên, vì nếu để ý các bạn sẽ thấy rằng không có code nằm giữa khoảng 5D000000 và địa chỉ bắt đầu của DLL đầu tiên tại 5D090000, vì vậy quá trình trace code sẽ không dừng lại, nhưng để tránh đưa con số chính xác tôi đã làm tròn).



Với việc thiết lập các giá trị như vậy, OllyDbg sẽ thực hiện quá trình debug unpackme bằng cách trace từng dòng lệnh và ứng mỗi dòng sẽ kiểm tra xem điều kiện có được đáp ứng, đó là, giá trị EIP thuộc section của một số DLL hay không, và nếu đúng như vậy nó sẽ dừng lại.

Trước khi cho OllyDbg trace code, tôi đặt thêm một số BP tại địa chỉ **004271D6**.  
**FF15 DC0A4600 call near dword ptr [460ADC]**

004271B0	55	push	ebp		
004271B1	8BEC	mov	ebp,esp		
004271B3	6A FF	push	-1		
004271B5	68 600E4500	push	UnPackMe.00450E60		
004271BA	68 C8924200	push	UnPackMe.004292C8		SE
004271BF	64:A1 00000000	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,-58		
004271D0	53	push	ebx		
004271D1	56	push	esi		
004271D2	57	push	edi		
004271D3	8965 E8	mov	dword ptr [ebp-18],esp		
004271D6	FF15 DC0A460	call	near dword ptr [460ADC]		
004271DC	33D2	xor	edx,edx		

Ngoài ra, tôi **Follow in Dump** tại địa chỉ 00460ADC, chọn 4 bytes tại đây và đặt một Memory bp là **Memory, on access**. Việc đặt BP này khi Run, OllyDbg sẽ dừng lại khi có truy cập tại vùng nhớ này.

Address	Hex dump	ASCII
00460ADC	F7 06 9E 00 08 07 9E 00 1A 07 9E 00 2A 07 9E 00	---
00460AEC	3B 07 00 00 0E 00 85 07 9E 00	...Z
00460AFC	96 07 00 00 0E 00 C4 07 9E 00	...S
00460B0C	D3 07 00 00 0E 00 24 08 9E 00	...ó
00460B1C	35 08 00 00 0E 00 68 08 9E 00	...P
00460B2C	79 08 00 00 0E 00 C3 08 9E 00	...P
00460B3C	D4 08 00 00 0E 00 00 00 00 00	....
00460B4C	11 09 00 00 0E 00 00 00 00 00	....
00460B5C	73 09 00 00 0E 00 00 00 00 00	....
00460B6C	B7 09 00 00 0E 00 00 00 00 00	....
00460B7C	12 0A 00 00 0E 00 00 00 00 00	....
00460B8C	4E 0A 00 00 0E 00 00 00 00 00	....

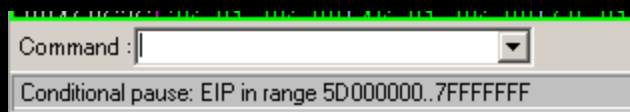
Sau khi đặt thêm các BP xong, tiến hành cho OllyDbg trace code bằng cách chọn **Debug > Trace into** hoặc nhấn phím tắt là **Ctrl+F11**:

Debug	Plugins	Options	Window	Help
Run		F9		
Pause		F12		
Restart		Ctrl+F2		
Close		Alt+F2		
Step into		F7		
Step over		F8		
Animate into		Ctrl+F7		
Animate over		Ctrl+F8		
Execute till return		Ctrl+F9		
Execute till user code		Alt+F9		
Open or clear run trace				
Trace into		Ctrl+F11		
Trace over		Ctrl+F12		
Set condition		Ctrl+T		
Close run trace				

**Tại sao lại sử dụng Trace into mà không phải là Trace over?** Lý do rất quan trọng là khi dùng Trace into thì OllyDbg sẽ chui vào lệnh Call và sẽ trace từng lệnh một, còn nếu chọn Trace over nó sẽ không chui vào các lệnh call, dẫn đến quá trình thực hiện có thể thất bại và không có được kết quả như mong muốn. Kết quả sau khi thực hiện ta dừng lại tại đây trong OllyDbg:

7C8114AB	64:A1 18000000	mov	eax,dword ptr fs:[18]
7C8114B1	8B48 30	mov	ecx,dword ptr [eax+30]
7C8114B4	8B81 B0000000	mov	eax,dword ptr [ecx+B0]
7C8114BA	0FB791 AC000000	movzx	edx,word ptr [ecx+AC]
7C8114C1	83F0 FE	xor	eax,FFFFFFFFE
7C8114C4	C1E0 0E	shl	eax,0E
7C8114C7	0BC2	or	eax,edx
7C8114C9	C1E0 08	shl	eax,8
7C8114CC	0B81 A8000000	or	eax,dword ptr [ecx+A8]
7C8114D2	C1E0 08	shl	eax,8
7C8114D5	0B81 A4000000	or	eax,dword ptr [ecx+A4]
7C8114DB	C3	retn	
7C8114DC	54	push	esp
7C8114DD	004D 00	add	byte ptr [ebp],cl
7C8114E0	50	push	eax

Quan sát thanh trạng thái của OllyDbg, ta thấy OllyDbg đã dừng lại theo điều kiện mà chúng ta đã thiết lập:



Giá trị của thanh ghi EIP lúc này nằm trong phạm vi **5D000000-7FFFFFFF**, đó là điều nằm trong mong đợi của tôi. Giờ là lúc để kiểm tra nhằm chắc chắn nó đã không dừng lại vì bất kỳ lý do gì nào khác.

Để kiểm tra ta cần nhớ lại khi thực hiện một lời gọi hàm. Như đã biết, khi quá trình thực thi một hàm API nào đó hoàn tất, nó sẽ quay trở về code chính của chương trình tại câu lệnh ngay sau lệnh CALL tới hàm API. Nhưng vì sẽ có những trình packers có thể gây nhiễu/nhầm lẫn bằng cách đầu tiên nó gọi tới một hàm API, và sau đó lại gọi đến một hàm API thứ hai. Do vậy hàm API thực sự cần tìm chính là hàm cuối cùng được gọi, và tại đó ta cần quan sát địa chỉ trở về của nó, theo đó nó sẽ trở về chương trình tại câu lệnh ngay bên dưới lệnh Call tới hàm API.

Tất nhiên trong trường hợp đơn giản này, ta thấy thông tin địa chỉ trở về nằm trên đỉnh của Stack chính là lệnh tiếp theo sau lệnh CALL (004271DC . 33D2 xor edx,edx):

```

| FST 4020  Cond 1 0 0 0  Err 0 0 1 0 0 0 0 0  (EQ)
0012FF48 004271DC RETURN to UnPackMe.004271DC from 009E06F7
0012FF4C 7C910738 ntdll.7C910738
0012FF50 FFFFFFFF
0012FF54 7FFD8000
0012FF58 0012FFE0
0012FF5C 004667DA RETURN to UnPackMe.004667DA from UnPackMe.004667E0
0012FF60 00000000

```

Chuột phải tại đây và chọn **Follow in Disassembler** để kiểm tra:

```

| FST 4020  Cond 1 0 0 0  Err 0 0
0012FF48 004271DC RETURN to UnPackMe.004271DC from 009E06F7
0012FF4C 7C910738 ntdll.7C910738
0012FF50 FFFFFFFF
0012FF54 7FFD8000
0012FF58 0012FFE0
0012FF5C 004667DA RETURN to UnPackMe
0012FF60 00000000
0012FF64 00000000
0012FF68 0012FBD0
0012FF6C 0012FF80
0012FF70 00000000
0012FF74 7C9037D8 ntdll.7C9037D8
0012FF78 004667AB RETURN to UnPackMe
0012FF7C 00466872 RETURN to UnPackMe
0012FF80 00000000
0012FF84 00000000
0012FF88 0005995E
0012FF8C 0012FFA0
0012FF90 00000000
0012FF94 004271B0 UnPackMe.004271B0
0012FF98 004666FF UnPackMe.004666FF
0012FF9C FFED6674
0012FFA0 0046673F UnPackMe.0046673F

```

Address  
Show ASCII dump  
Show UNICODE dump  
Lock stack  
Copy to clipboard Ctrl+C  
Modify  
Edit Ctrl+E  
Push DWORD  
Pop DWORD  
Search for address  
Search for binary string Ctrl+B  
Go to EBP  
Go to expression Ctrl+G  
**Follow in Disassembler Enter**  
Follow in Dump  
Appearance

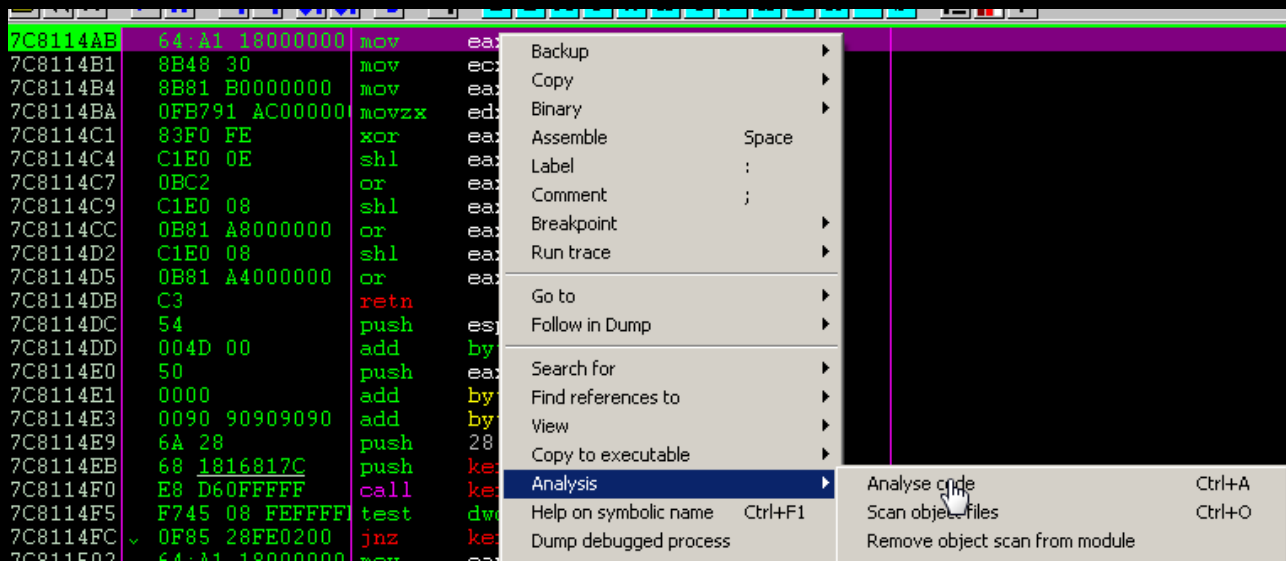
```

004271D3 8965 E8 mov dword ptr [ebp-18],esp
004271D6 FF15 DC0A460 call near dword ptr [460ADC]
004271DC 33D2 xor edx,edx
004271DE 8AD4 mov dl,ah

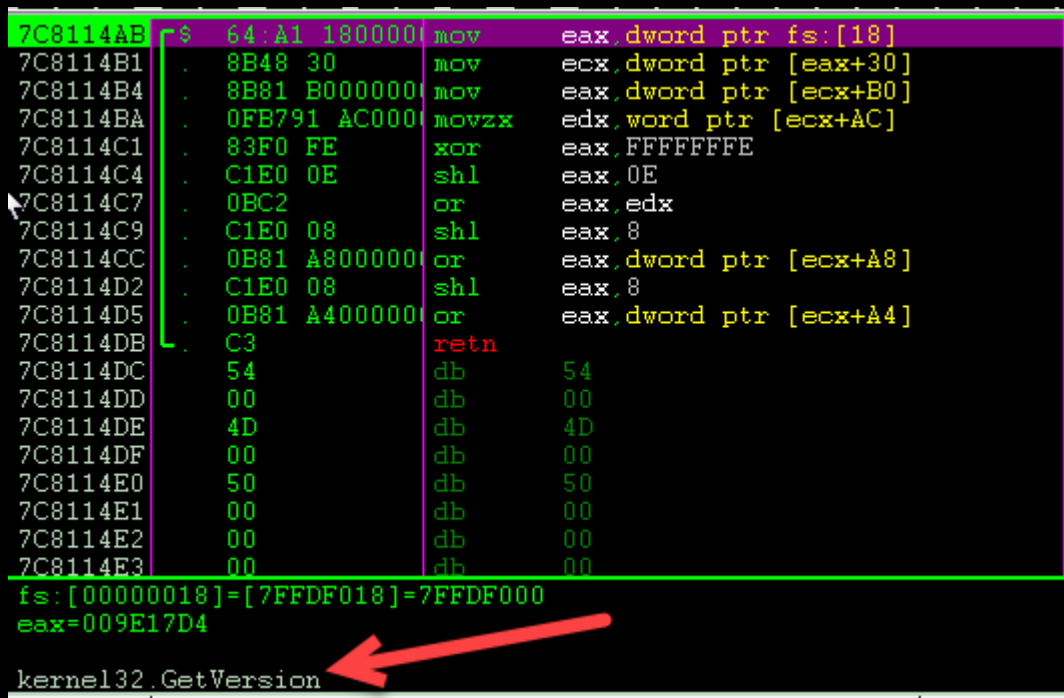
```

Như đã thấy trên hình, ta quay lại địa chỉ trở về là **0x4271DC**.

Ok, giờ ta đang dừng lại tại địa chỉ (7C8114AB > 64:A1 18000000 mov eax,dword ptr fs:[18]) trên máy của tôi, do EIP đã thỏa mãn điều kiện dừng đã thiết lập. Bây giờ điều tôi cần là thông tin về hàm API, tuy nhiên hiện tại trong OllyDbg chưa cung cấp thông tin một cách rõ ràng. Cách đầu tiên là tiến hành phân tích code tại chính DLL này. Tại OllyDbg nhấn chuột phải và chọn **Analysis > Analyse code (Ctrl+A)**:



Kết quả sau khi Analyse code, tại cửa sổ Tip ta thấy xuất hiện thông tin hàm API:



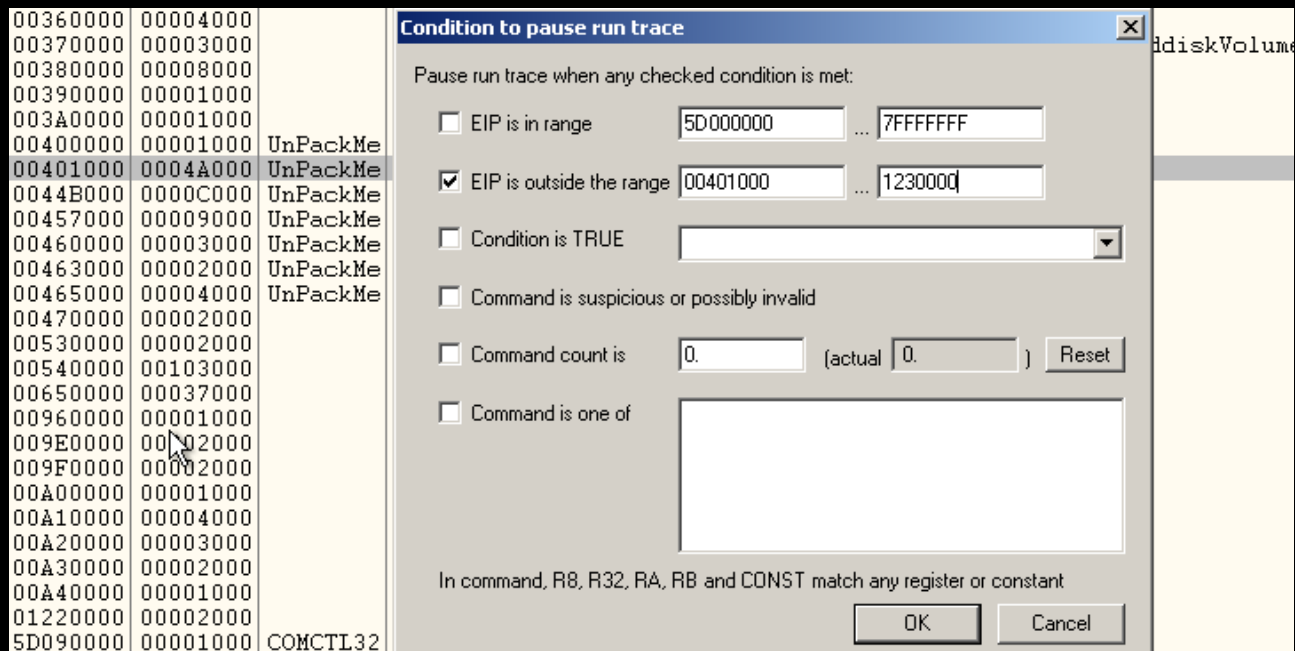
Thông tin của cửa sổ Tip cũng trùng khớp với tên của hàm API đang được lưu tại thanh ghi EIP ở cửa sổ **Registers**:

```

Registers (FPU)
EAX 009E17D4
ECX 0012FFB0
EDX 7C90EB94 ntdll.KiFastSystemCallRet
EBX 7FFD8000
ESP 0012FF48
EBP 0012FFC0
ESI FFFFFFFF
EDI 7C910738 ntdll.7C910738
EIP 7C8114AB kernel32.GetVersion
C 0 ES 0023 32bit 0(FFFFFFFF)
P 1 CS 001B 32bit 0(FFFFFFFF)
A 0 SS 0023 32bit 0(FFFFFFFF)

```

Ngoài thiết lập ở trên, một cách thiết lập khác tại cửa sổ **"Condition to pause run trace"** có thể giúp chúng ta tối được hàm API như sau:



Thiết lập này ngược lại với cách thiết lập trước, nó sẽ kiểm tra giá trị của thanh ghi EIP nằm ngoài phạm vi các sections của unpackme, và các sections nằm trước DLL đầu tiên. Khi tiến hành thực hiện trace ta sẽ có kết quả tương tự:



7C8114AB 64 A1 180000 mov eax,dword ptr fs:[18]

7C8114B1 8B48 30 mov ecx,dword ptr [eax+30]

7C8114B4 8B81 B000000 mov eax,dword ptr [ecx+B0]

7C8114BA 0FB791 AC000 movzx edx,word ptr [ecx+AC]

7C8114C1 83F0 FE xor eax,FFFFFFFF

7C8114C4 C1E0 0E shl eax,0E

7C8114C7 0BC2 or eax,edx

7C8114C9 C1E0 08 shl eax,8

7C8114CC 0B81 A800000 or eax,dword ptr [ecx+A8]

7C8114D2 C1E0 08 shl eax,8

7C8114D5 0B81 A400000 or eax,dword ptr [ecx+A4]

7C8114DB C3 ret

7C8114DC 54 db 54

7C8114DD 00 db 00

7C8114DE 4D db 4D

7C8114DF 00 db 00

7C8114E0 50 db 50

7C8114E1 00 db 00

7C8114E2 00 db 00

7C8114E3 00 db 00

7C8114E4 90 nop

fs:[00000018]=[7FFDF018]=7FFDF000  
eax=009E17D4

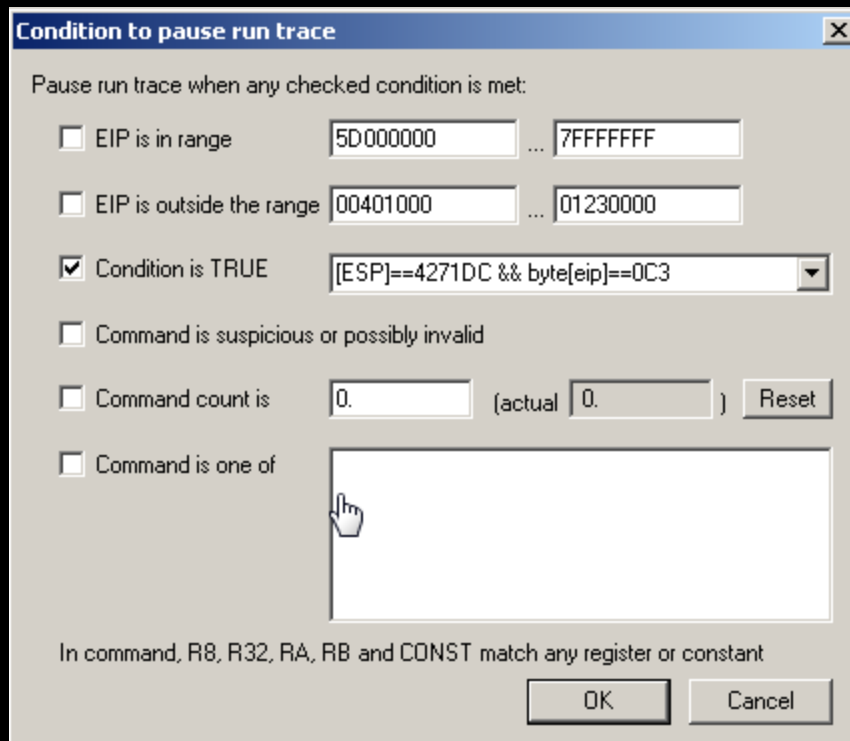
kernel32.GetVersion

Address	Hex dump	AS
00460ADC	F7 06 9E 00 08 07 9E 00 1A 07 9E 00 2A 07 9E 00	0012FF
00460AEC	3B 07 9E 00 5A 07 9E 00 74 07 9E 00 85 07 9E 00	0012FF!
00460AFC	96 07 9E 00 A7 07 9E 00 B5 07 9E 00 C4 07 9E 00	0012FF!

Command :

Conditional pause: EIP outside the range 00401000..01230000

Một thiết lập khác nữa cũng cho kết quả giống như hai phương pháp trên:



Trong một số trường hợp, các trình packer có thể tạo ra các sections nằm lẫn lộn giữa các DLL, với cách thiết lập kết hợp như trên có thể được áp dụng để dừng lại khi tìm thấy một lệnh **RET (0xC3)** và tại đỉnh của Stack chứa giá trị **0x4271DC**, là địa chỉ trở về sau khi thực hiện xong hàm API. Mục tiêu của phương pháp này nhằm dừng lại trước khi trở về lệnh bên dưới lời gọi hàm API, và thường sẽ có hiệu quả hơn đối với các trình packer đã emulate (mô phỏng/giả lập) các câu lệnh đầu tiên của hàm API và nhảy đến lệnh thứ ba hoặc thứ tư của hàm. Thông thường, ta biết lệnh **RET** của một hàm sẽ không bị can thiệp. Giờ khởi động lại OllyDbg, tới OEP, thiết lập lại các BP, cấu hình điều kiện dừng như trên hình và tiến hành cho OllyDbg trace để kiểm tra.

Dấu **&&** hàm ý phải thỏa mãn hai điều kiện cùng lúc mới dừng lại. Trong trường hợp muốn dừng khi thỏa một trong hai điều kiện thì sử dụng **||**. Do vậy, với biểu thức: **[ESP]==4271dc && byte [EIP]==0C3**, đồng nghĩa là cả hai điều kiện phải được đáp ứng cùng một lúc thì mới dừng:

- **[ESP]==4271dc** (điều kiện này là tại đỉnh của Stack chứa địa trị trở về sau lời gọi hàm API).
- **byte [EIP]==0C3** (nội dung của thanh ghi EIP chứa giá trị C3 hay tương đương với lệnh RET)

Kết quả sau khi trace code, quan sát tại OllyDbg ta thấy sẽ dừng tại lệnh **RET** của API.

7C8114AB	64:A1 180000	mov	eax,dword ptr fs:[18]	
7C8114B1	8B48 30	mov	ecx,dword ptr [eax+30]	
7C8114B4	8B81 B0000000	mov	eax,dword ptr [ecx+B0]	
7C8114BA	0FB791 AC0000	movzx	edx,word ptr [ecx+AC]	
7C8114C1	83F0 FE	xor	eax,FFFFFFFFE	
7C8114C4	C1E0 0E	shl	eax,0E	
7C8114C7	0BC2	or	eax,edx	
7C8114C9	C1E0 08	shl	eax,8	
7C8114CC	0B81 A8000000	or	eax,dword ptr [ecx+A8]	
7C8114D2	C1E0 08	shl	eax,8	
7C8114D5	0B81 A4000000	or	eax,dword ptr [ecx+A4]	
7C8114DE	C3	ret		
7C8114DC	54	db	54	CHAR
7C8114DD	00	db	00	
7C8114DE	4D	db	4D	CHAR
7C8114DF	00	db	00	

Return to 004271DC (UnPackMe.004271DC)

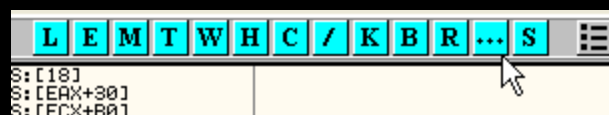
kernel32.GetVersion+30

Command:

Conditional pause: [ESP]==4271DC && byte [eip]==0C3

Với kết quả như trên hình, khi cả hai điều kiện cùng được thỏa mãn thì quá trình trace code sẽ dừng lại. Bằng cách này, khi dừng lại tại lệnh **RET**, rất khó để biết tên của hàm API mà ta đang thực thi, bởi ta không đứng tại điểm bắt đầu của hàm. Điều tương tự xảy ra khi mà trình packer mô phỏng hai hay ba lệnh đầu tiên của API và ta đang đứng tại dòng lệnh thứ 4 hoặc 5 của hàm. Ở đây, OllyDbg không hiển thị cho chúng ta tên hàm, tuy nhiên có một số cách để tìm ra.

Cách thứ nhất là tìm trong danh sách những lệnh mà OllyDbg đã trace, để có được danh sách này nhấn vào nút có ba chấm như trong hình:



Back	Thread	Module	Address	Command	Modified registers
19.	Main	UnPackMe	004271D6	call near dword ptr [460ADC]	
18.	Main		009E06F7	test esp,esp	
17.	Main		009E06F9	jns short 009E06FE	
16.	Main		009E06FE	inc eax	EAX=0012FFE1
15.	Main		009E06FF	mov eax,9E17D3	EAX=009E17D3
14.	Main		009E0704	inc eax	EAX=009E17D4
13.	Main		009E0705	push dword ptr [eax]	
12.	Main		009E0707	retn	
11.	Main	kernel32	7C8114AB	mov eax,dword ptr fs:[18]	EAX=7FFDF000
10.	Main	kernel32	7C8114B1	mov ecx,dword ptr [eax+30]	ECX=7FFD5000
9.	Main	kernel32	7C8114B4	mov eax,dword ptr [ecx+B0]	EAX=00000002
8.	Main	kernel32	7C8114BA	movzx edx,word ptr [ecx+AC]	EDX=00000A28
7.	Main	kernel32	7C8114C1	xor eax,FFFFFFFF	EAX=FFFFFFFF
6.	Main	kernel32	7C8114C4	shl eax,0E	EAX=FFFFF000
5.	Main	kernel32	7C8114C7	or eax,edx	EAX=FFFFF0A28
4.	Main	kernel32	7C8114C9	shl eax,8	EAX=FF0A2800
3.	Main	kernel32	7C8114CC	or eax,dword ptr [ecx+A8]	EAX=FF0A2801
2.	Main	kernel32	7C8114D2	shl eax,8	EAX=0A280100
1.	Main	kernel32	7C8114D5	or eax,dword ptr [ecx+A4]	EAX=0A280105
0.		kernel32	7C8114DB	retn	

Ta thấy rõ ràng là lệnh đầu tiên mà ta trace trong DLL chính là dòng mà tôi đánh dấu bởi mũi tên trên hình, tuy nhiên nó không cung cấp cho chúng ta tên của hàm API, do đó tiến hành phân tích đoạn code và sau đó kích đúp vào dòng được đánh dấu mũi tên như trên hình.

The screenshot shows the OllyDbg interface. The assembly window displays the following instructions:

```

7C8114AB 8B41 10000B mov     eax,dword ptr fs:[18]
7C8114B1 8B48 30     mov     ecx,dword ptr [eax+30]
7C8114B4 8B81 B00000 mov     eax,dword ptr [ecx+B0]
7C8114BA 0FB791 AC000 movzx   edx,word ptr [ecx+AC]
7C8114C1 83F0 FE     xor     eax,FFFFFFFF
7C8114C4 C1E0 0E     shl     eax,0E
7C8114C7 0BC2       or      eax,edx
7C8114C9 C1E0 08     shl     eax,8
7C8114CC 0B81 A80000 or      eax,dword ptr [ecx+A8]
7C8114D2 C1E0 08     shl     eax,8
7C8114D5 0B81 A40000 or      eax,dword ptr [ecx+A4]
7C8114DE C3         retn
7C8114DC 54         db      54
7C8114DD 00         db      00
7C8114DE 4D         db      4D
7C8114DF 00         db      00

```

The registers window on the right shows the following values:

```

EAX 009E17D4
ECX 0012FFB0
EDX 7C90EB94 ntdll.KiFastSystemCallRet
EBX 7FFD5000
ESP 0012FF48
EBP 0012FFC0
ESI FFFFFFFF
EDI 7C910738 ntdll.7C910738
EIP 7C8114AB kernel32.GetVersion


```

A red arrow points to the 'EIP' field, which is highlighted in yellow and shows '7C8114AB kernel32.GetVersion'. Another red arrow points to the instruction 'mov eax, dword ptr fs:[18]' in the assembly window.

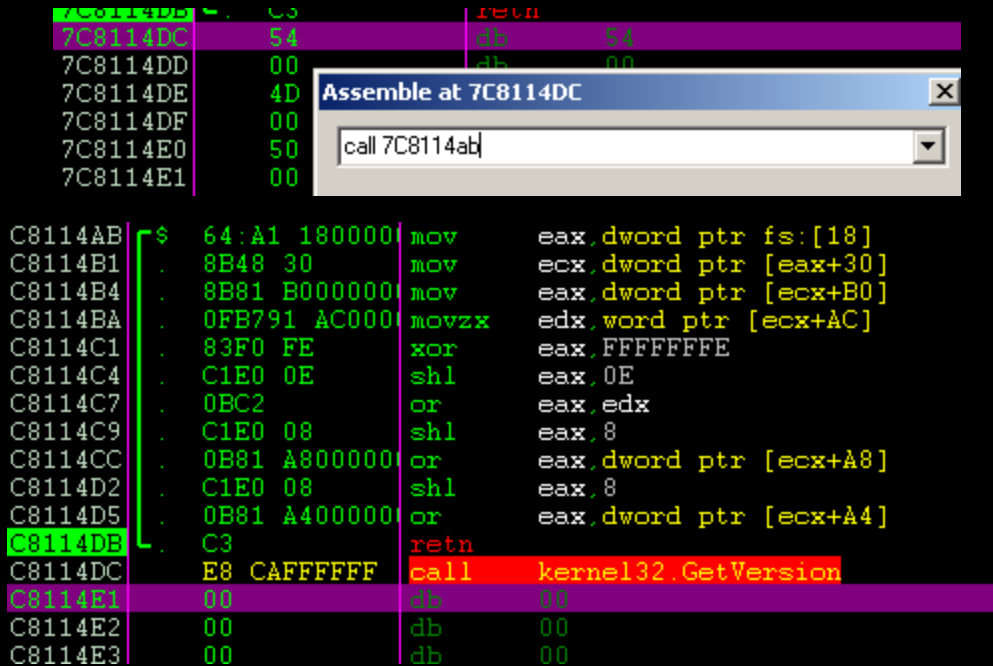
Ta thấy rằng OllyDbg sẽ cung cấp lại cho ta thông tin về các giá trị (vùng nhớ; thanh ghi...) tại thời điểm đó, đó là lý do vì sao các cửa sổ Registers hay Tip lại có màu tím trên máy tôi (hay màu xám trên máy các bạn), bởi vì các giá trị đó phải là giá trị hiện thời. Thông tin mà OllyDbg hiển thị sẽ cho ta cảm giác như thể là đang thực thi câu lệnh đó, nhờ vậy ta có thể thấy tên của hàm API được gọi.

Một cách làm khác cũng sẽ cho kết quả tương tự, nếu sau khi trace xong, ta nhấn phím tắt trừ ("="), OllyDbg sẽ quay trở lại các lệnh đã thực hiện trước đó và hiển thị thông tin và các giá trị lưu giữ khi trace qua từng từng lệnh.

Tuy nhiên, có trường hợp mà các trình packer mô phỏng lại hai hay ba lệnh đầu tiên của API, sau khi thực hiện trace code, ta dừng lại tại lệnh thứ tư hoặc thứ năm. Lúc đó trình tracer sẽ không đi qua lệnh đầu tiên vì chúng đã được giả lập bởi packer, do vậy

đôi khi ta phải tinh mắt một chút, tuy vậy có một phương pháp tự chế mà sẽ luôn có được kết quả như sau hehe .

Tìm bất kì dòng nào trống bên dưới nhấn phím cách, viết một lệnh CALL đến địa chỉ mà ta nghi ngờ là nơi bắt đầu của hàm API, ví dụ trong trường hợp này: **CALL 7C8114AB**.



The screenshot shows the OllyDbg assembly window with the following code:

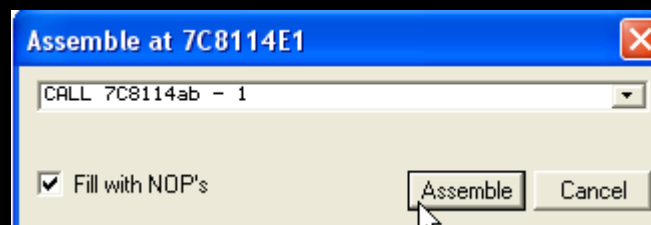
Address	Disassembly	Comment
7C8114DC	54	db 54
7C8114DD	00	db 00
7C8114DE	4D	db 4D
7C8114DF	00	db 00
7C8114E0	50	db 50
7C8114E1	00	db 00

The 'Assemble at 7C8114DC' dialog box is open, showing the command 'call 7C8114ab'.

The assembly window also shows the following code:

Address	Disassembly	Comment
C8114AB	64:A1 180000	mov eax,dword ptr fs:[18]
C8114B1	8B48 30	mov ecx,dword ptr [eax+30]
C8114B4	8B81 B00000	mov eax,dword ptr [ecx+B0]
C8114BA	0FB791 AC00	movzx edx,word ptr [ecx+AC]
C8114C1	83F0 FE	xor eax,FFFFFFFE
C8114C4	C1E0 0E	shl eax,0E
C8114C7	0BC2	or eax,edx
C8114C9	C1E0 08	shl eax,8
C8114CC	0B81 A80000	or eax,dword ptr [ecx+A8]
C8114D2	C1E0 08	shl eax,8
C8114D5	0B81 A40000	or eax,dword ptr [ecx+A4]
C8114DE	C3	retn
C8114DC	E8 CAFFFFFF	call kernel32.GetVersion
C8114E1	00	db 00
C8114E2	00	db 00
C8114E3	00	db 00

Sau khi nhập lệnh và nhấn **Assemble**, nếu đó là địa chỉ đầu của hàm API, OllyDbg sẽ hiển thị cho ta tên của hàm như trên hình, nhưng nếu đó không phải là địa chỉ đầu tiên, ta có thể lặp lại với lệnh như sau:



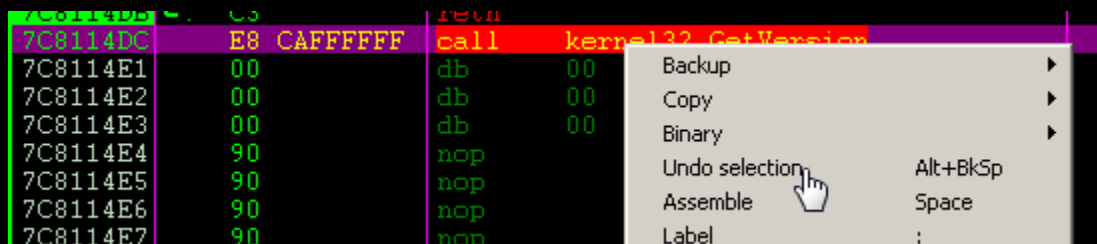
Cứ làm như vậy trong vùng code đang phân tích, cho đến khi OllyDbg hiển thị được tên của hàm. Tuy nhiên, thường thì sẽ rất dễ dàng để nhận ra đâu là địa chỉ bắt đầu của hàm, dấu hiệu nhận biết chính là dấu ngoặc vuông mà OllyDbg hiển thị khi tiến hành phân tích code (**Ctrl+A**). Như trong hình, ta thấy được khung với địa chỉ của hàm API, với dấu **\$**.

```

C81141B1 64:A1 180000 mov     eax,dword ptr fs:[18]
C81141B2 8B48 30      mov     ecx,dword ptr [eax+30]
C81141B4 8B81 B000000 mov     eax,dword ptr [ecx+B0]
C81141BA 0FB791 AC000 movzx   edx,word ptr [ecx+AC]
C81141C1 83F0 FE      xor     eax,FFFFFFF
C81141C4 C1E0 0E      shl     eax,0E
C81141C7 0BC2         or      eax,edx
C81141C9 C1E0 08      shl     eax,8
C81141CC 0B81 A800000 or      eax,dword ptr [ecx+A8]
C81141D2 C1E0 08      shl     eax,8
C81141D5 0B81 A400000 or      eax,dword ptr [ecx+A4]
C81141DB C3          retn

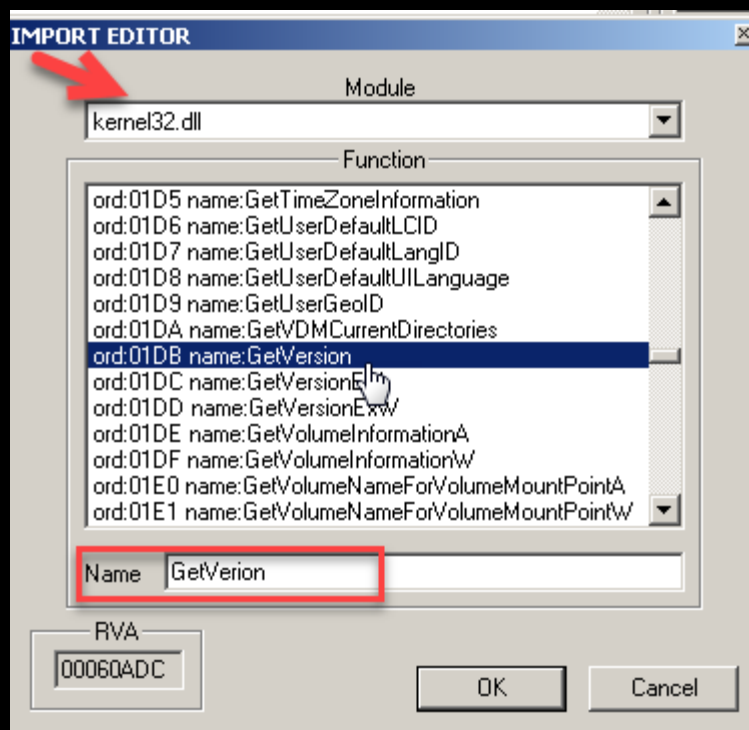
```

Sau khi tìm ra được tên hàm, để khôi phục lại các bytes gốc ban đầu tại địa chỉ mà ta đã chen lệnh ASM, lựa chọn lệnh đã chen, nhấn chuột phải và chọn **Undo selection**:

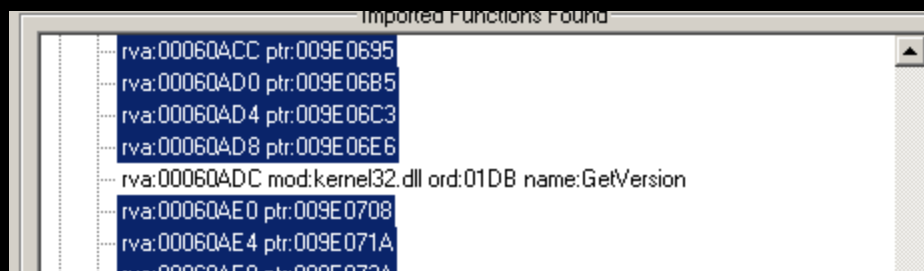


Với cách làm như trên, ta đã biết một cách thủ công để tìm ra tên của hàm API.

Sau khi đã tìm được tên hàm cần tìm, ta chuyển qua ImpREC, nhấp đúp vào giá trị cần sửa và chọn đúng tên hàm đã tìm được. Tương tự như hình:



Sau khi sửa xong, tôi nhấn **Show Invalid**, ta thấy có được kết quả chính xác như mong muốn:

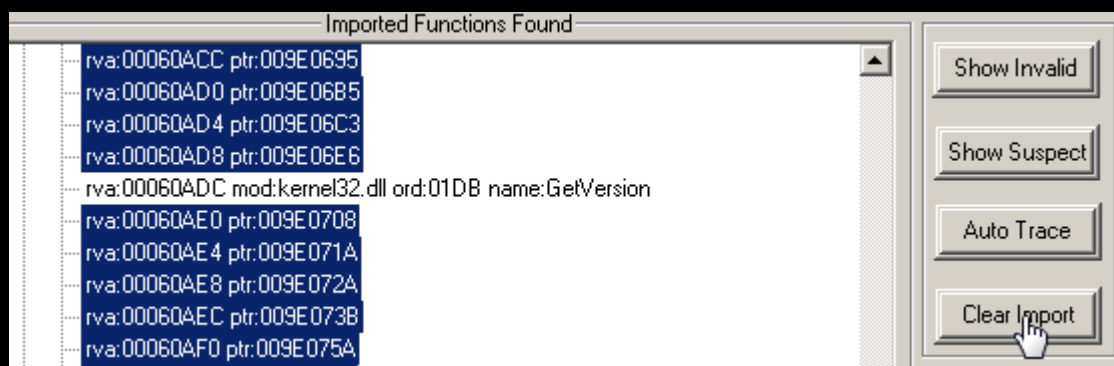


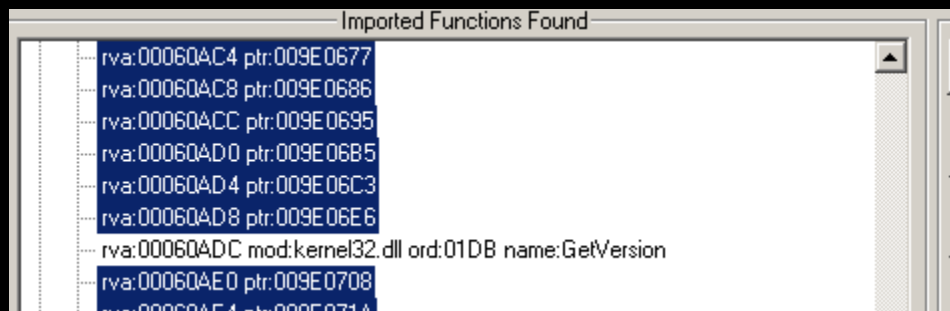
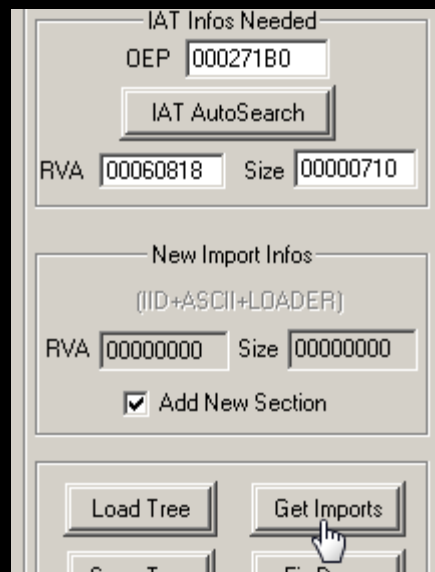
Một cách khác để sửa lại entry này là trong OllyDbg, ta ghi đè các bytes tại 460ADC bằng địa chỉ chính xác của api mà ta tìm được (địa chỉ API trên máy của tôi là 7C8114AB). Ví dụ:

Address	Hex dump	ASCII
00460ADC	F7 06 9E 00 08 07 9E 00 1A 07 9E 00 2A 07 9E 00	[-] [0] [→] [**] [.]
00460AEC	3B 07 9E 00 5A 07 9E 00 74 07 9E 00 85 07 9E 00	[-] [Z] [t] [.] [.]
00460AFC	96 07 9E 00 A7 07 9E 00 B5 07 9E 00 C4 07 9E 00	[-] [S] [μ] [À] [.]
00460B0C	D3 07 9E 00 F3 07 9E 00 01 08 9E 00 24 08 9E 00	Ó [-] [-] [r] [-] [s] [-]

Address	Hex dump	ASCII
00460ADC	AB 14 81 7C 08 07 9E 00 1A 07 9E 00 2A 07 9E 00	[-] [0] [→] [**] [.]
00460AEC	3B 07 9E 00 5A 07 9E 00 74 07 9E 00 85 07 9E 00	[-] [Z] [t] [.] [.]
00460AFC	96 07 9E 00 A7 07 9E 00 B5 07 9E 00 C4 07 9E 00	[-] [S] [μ] [À] [.]
00460B0C	D3 07 9E 00 F3 07 9E 00 01 08 9E 00 24 08 9E 00	Ó [-] [-] [r] [-] [s] [-]

Với cách sửa như trên, quay trở lại ImpREC, tiến hành làm sạch toàn bộ các Imports bằng cách nhấn nút **Clear Import**, sau đó nhấn lại nút **Get Imports**, ta sẽ thấy rằng entry vừa fix được xem là một entry hợp lệ.



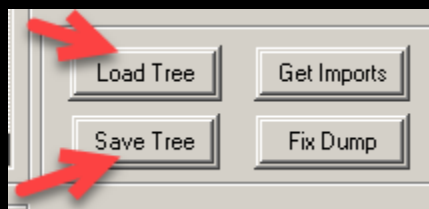


Như các bạn thấy từ đầu đến giờ, toàn bộ quá trình thực hiện trên ta mới chỉ làm cho một hàm. Nếu tiếp tục làm lần lượt tương tự cho từng địa chỉ bị redirect, sửa từng hàm một, và sau đó thực hiện việc fix dump, đương nhiên là file sau khi fix sẽ chạy. Vấn đề nằm ở chỗ khi có quá nhiều địa chỉ bị redirect, nếu làm tay như vậy thì vô cùng oải và nhanh nản, dù đây vẫn là một cách tốt vừa để hiểu cũng như để xác minh lại các điểm entry nghi ngờ.

Trong quá trình thực hiện sửa bằng tay thủ công như trên, phải có một cách nào đó giúp lưu lại được toàn bộ thông tin liên quan tới OEP, IAT table, RVA, Size.. nếu không

khi có lỗi xảy ra, làm lại từ đầu là nản luôn 🤯. Rất may là công cụ IMP REC cho phép ta lưu lại các thông tin này thông qua nút **Save Tree**. Nếu trong quá trình thực hiện bị gián đoạn, ta có thể quay lại OEP của chương trình, mở IMP REC tiến hành tải lại các thông tin liên quan đã lưu như OEP, RVA ... thông qua nút **Load Tree**.



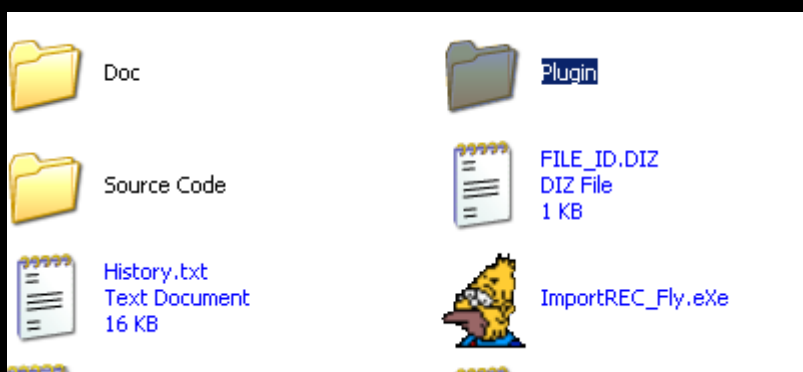


## 2. Sử dụng plugin viết riêng cho từng packer

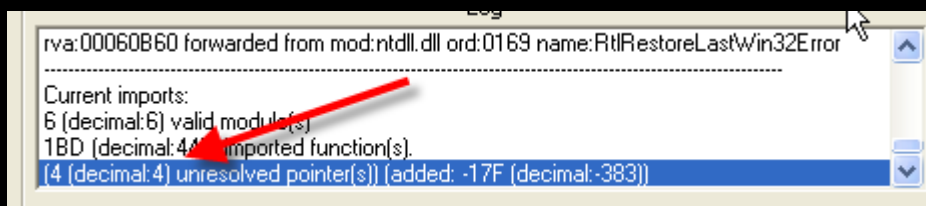
Bên cạnh cách làm manual ở trên, một cách khác có thể sử dụng để sửa các IAT bị chuyển hướng đó là dùng plugin của ImpREC được viết riêng để fix tElock.



Plugin này được tôi gửi kèm theo bài viết, các bạn chỉ việc chép vào thư mục Plugin của ImpREC là có thể sử dụng bình thường.



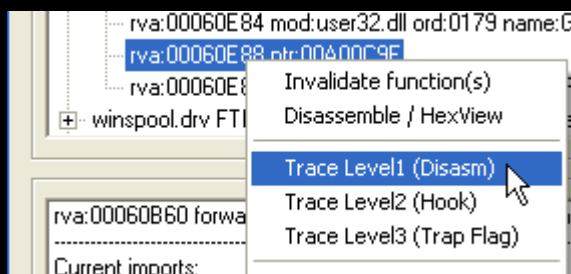
Sau khi chép vào thư mục Plugin, ta phải khởi động lại ImpREC và nạp lại các thông tin đã lưu, sau đó nhấn **Show Invalid** để liệt kê những entry chưa được fix. Sau đó, nhấn chuột phải và chọn **Plugin Tracers > tElock**. Plugin này sẽ tiến hành việc tìm và sửa lại bảng IAT như quá trình chúng ta đã thực hiện các thao tác bằng tay ở trên. Khi plugin này chạy xong, ImpREC sẽ thông báo kết quả thông qua Log, theo đó chỉ còn lại 4 chỗ mà plugin này chưa sửa được:



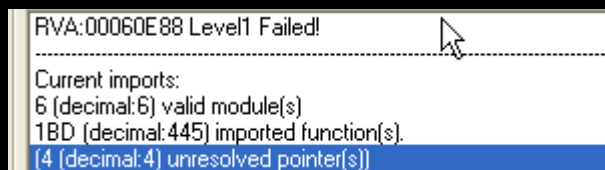
Với 4 RVA chưa fix được ở trên, lúc này ta có thể trace bằng tay bằng phương pháp mà đã làm ở phần đầu để xác minh các RVA đó có đúng là API hay không. Phương pháp

sử dụng plugin có điểm hạn chế là chúng ta sẽ phải phụ thuộc vào plugin đó, và chỉ áp dụng với một Packer cụ thể, không thể áp dụng rộng rãi cho các trình packer khác.

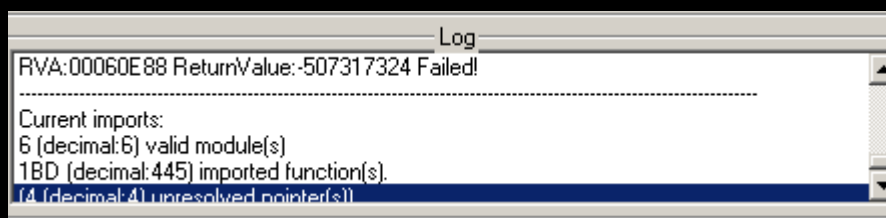
Bên cạnh việc sử dụng plugin, ImpREC còn có thêm các tính năng hỗ trợ khác, ta có thể áp dụng thử để xem nó có hoạt động không, nhưng trước khi thử tôi khuyên bạn nên lưu lại bản IAT đã sửa tính đến thời điểm này, bởi vì phần lớn khi sử dụng các chức năng như hình dưới có khả năng sẽ làm crash hoặc treo ImpREC.



Sau khi đã lưu lại toàn bộ thông tin, ta sẽ thử áp dụng các tính năng trên xem kết quả thế nào. Ví dụ, ta chọn một địa chỉ RVA invalid như trên hình, nhấn chuột phải tại đây sẽ thấy ImpREC cung cấp cho chúng ta ba cấp độ Trace, trước tiên thử với **Trace Level 1 (Disasm)** xem thế nào:



Ta nhận được thông báo **Level1 Failed!** Tiếp tục thử nốt với hai level còn lại, ta thấy ImpREC thông báo như bên dưới (có thể ImpREC sẽ bị treo khi sử dụng hai level này).



Mặc dù, với trường hợp cụ thể này nó không hoạt động nhưng các phương pháp này vẫn có thể áp dụng cho các trường hợp khác, tuy nhiên chúng thường gây crash hoặc làm treo ImpREC, và thường được chỉ định áp dụng cho các trình packer đơn giản.

### 3. Tìm magic jump

Phương pháp cuối cùng tôi đề cập trong bài viết này là Magic Jmp (phương pháp này nghe nói được truyền nội bộ trong nhóm CracksLatinos), có thể áp dụng cho rất nhiều trình packer khác nhau.

Ý tưởng của phương pháp này rất đơn giản, đó là dựa trên việc đi tìm ra thời điểm mà trình packer thực hiện lưu các giá trị trong bảng IAT, sau đó tiến hành quan sát và so sánh điều gì xảy ra khi nó lưu một bad entry với điều gì xảy ra khi nó lưu một good entry.

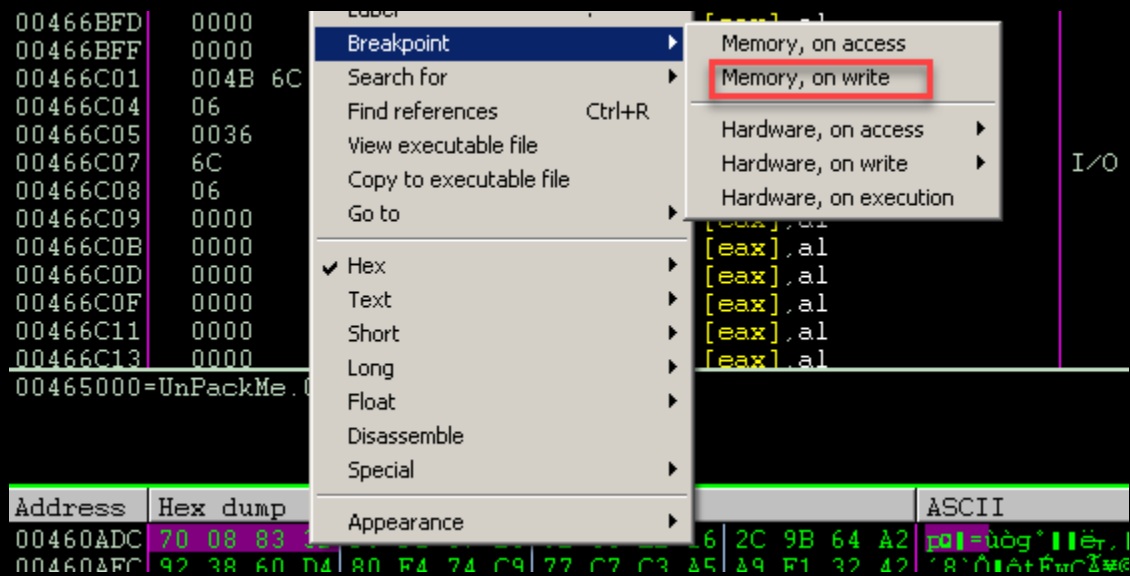
Để thực hiện phương pháp này tôi sẽ lấy ví dụ với hàm API **GetVersion()** mà như ta đã biết đây là một redirected IAT hay bad entry. Khởi động lại crackme, tìm giá trị hiện thời đang lưu tại địa **0x460ADC** trước khi tiến hành tìm OEP, ta có như sau:

00466C13	0000	add	byte ptr [eax], al	
00465000=UnPackMe.00465000				
Address	Hex dump	ASCII		
00460ADC	70 08 83 3D F9 F2 67 B0 9B 85 EB 16 2C 9B 64 A2	001=00g*  êr, d0		
00460AEC	92 38 60 D4 80 F4 74 C9 77 C7 C3 A5 A9 F1 32 42	'8`Ô ôtEwCÃ#0ñ2B		
00460AFC	89 DC B4 0B 7A 7C C9 6E CA 89 9E AA F2 A2 89 81	Û'rz ÊnE  â00		
00460B0C	2B 99 F9 53 60 A2 69 A7 F1 3B 2F C9 7A BF 0D F4	+14S`B1\$3-Ê7J A		

Quan sát tại cửa sổ Dump ta thấy được giá trị ban đầu lưu tại 460ADC Như vậy, rõ ràng là tại một nơi nào đó giá trị ở đây sẽ bị thay thế bằng một giá trị khác mà như ta đã biết chính là redirected entry, ví dụ trên máy tôi là **9F06F7**. Từ đây, ta hiểu rằng tại đâu đó trong code của mình trình packer sẽ ghi đè lên giá trị ban đầu này bằng địa chỉ redirected kia.

Address	Hex dump	ASCII		
00460ADC	F7 06 9E 00 08 07 9E 00 1A 07 9E 00 2A 07 9E 00	== 0 .→ .* .		
00460AEC	3B 07 9E 00 5A 07 9E 00 74 07 9E 00 85 07 9E 00	:0 .Z0 .t0 . 0 .		
00460AFC	96 07 9E 00 A7 07 9E 00 B5 07 9E 00 C4 07 9E 00	0 .S0 .p0 .Ă0 .		
00460B0C	D3 07 9E 00 F3 07 9E 00 01 08 9E 00 24 08 9E 00	Ó0 .ó0 .p0 . \$0 .		
00460B1C	35 08 9E 00 46 08 9E 00 58 08 9E 00 68 08 9E 00	\$0 .F0 .X0 .ha .		
00460B2C	79 08 9E 00 98 08 9E 00 B2 08 9E 00 C3 08 9E 00	va   na  2na  2na		

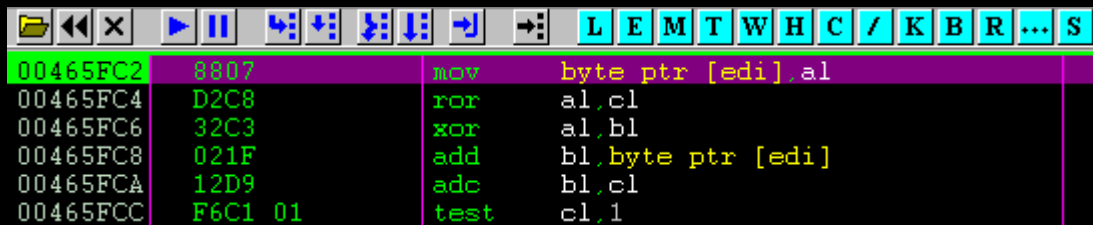
Để tìm được đoạn code nào thực hiện ghi giá trị vào 460ADC, thông thường sẽ đặt một breakpoint là **Hardware, on write**. Tuy nhiên, nhiều trường hợp trình packer (cụ thể ở packer) có thể phát hiện việc đặt Hardware BP, do đó ta sẽ đặt một Breakpoint, **Memory, on write** như sau:



Đặt xong bp, nhấn **F9** để RUN, lần đầu tiên ta break lại ở đây:



Dễ dàng nhận thấy đây không phải là nơi cần tìm vì khi trace code bằng **F8**, ta thấy nó không lưu giá trị mong muốn, vì vậy tiếp tục nhấn **F9** để RUN một lần nữa sẽ dừng lại tại đây:



Tương tự như trên, đây cũng không phải là ta nơi cần tìm. Nhấn **F9** để tiếp tục RUN, tới đây:



Vẫn không phải là vùng code cần quan tâm, nhấn **F9**, dừng lại tại đây:

```

00466128  F3:A4      rep     movs byte ptr es:[edi], byte ptr [esi]
0046612A  5E         pop     esi
0046612B  ^ EB 8E     jmp     short UnPackMe.004660BB
0046612D  02D2      add     dl, dl
0046612F  75 05     jnz     short UnPackMe.00466136
00466131  0A1C      mov     dl, byte ptr [esi]

```

Quan sát các giá trị tại `ds:[esi]` và `es:[edi]`, ta cũng thấy đây vẫn chưa phải là nơi cần tìm. Vì sau khi thực hiện lệnh REP MOVSB trên, ta có giá trị sau lưu ở 00460ADC như sau:

Address	Hex dump	ASCII
00460ADC	A0 12 06 00 96 12 06 00 82 12 06 00 6E 12 06 00	[-. [-. [-. n[-.
00460AEC	60 12 06 00 54 12 06 00 3A 12 06 00 2A 12 06 00	`[-. T[-. :[-. *[-.
00460AFC	1A 12 06 00 02 12 06 00 EE 11 06 00 D8 11 06 00	-[-. i[-. i[-. @[-.
00460B0C	C8 11 06 00 B4 11 06 00 AC 11 06 00 8E 11 06 00	E[-. '[-. '[-. l[-.
00460B1C	70 11 06 00 52 11 06 00 36 11 06 00 24 11 06 00	p[-. R[-. 6[-. \$[-.

Tiếp tục nhấn **F9** (RUN), tới đoạn code sau:

```

004664E5  8908      mov     dword ptr [eax], ecx
004664E7  58        pop     eax
004664E8  83C0 09   add     eax, 9
004664EB  0185 56D44000 add     dword ptr [ebp+40D456], eax
004664F1  ✓ EB 08     jmp     short UnPackMe.004664FB
004664F3  838D 52D44000 or      dword ptr [ebp+40D452], 0FFFFFFF
004664FA  61        popad
004664FB  03BD 4ED34000 add     edi, dword ptr [ebp+40D34E]
00466501  85DB      test    ebx, ebx
00466503  ✓ 0F84 C7000000 je      UnPackMe.004665D0

```

Ngó qua cửa sổ Registers, ta thấy thanh ghi ECX đang chứa giá trị **009E06F7**:

Registers (FPU)		
EAX	00460ADC	UnPackMe.00460ADC
ECX	009E06F7	
EDX	00400000	UnPackMe.00400000
EBX	000612A0	
ESP	0012FFA0	
EBP	0005995E	
ESI	00460014	UnPackMe.00460014
EDI	009E166C	
EIP	004664E5	UnPackMe.004664E5
C 0	ES 0023 32bit	0(FFFFFFFF)
P 0	CS 001B 32bit	0(FFFFFFFF)

**ECX = 9F06F7** (chính là địa chỉ mà API của ta bị chuyển hướng tới) và sẽ được lưu vào vùng nhớ trỏ bởi EAX hay 460ADC, vậy đây đúng là vùng code mà chúng ta cần tìm. Khi nhấn **F8**, ta thấy 460ADC lưu giá trị của thanh ghi ECX. Đây mới là bước đầu tiên chúng ta

phải làm khi áp dụng phương pháp này, bây giờ đến phần khó nhất là đi tìm tử huyệt – magic jump.

Address	Hex dump
00460ADC	F7 06 9E 00 96 12 06 00 82 12 06 00 6E 12 06
00460AEC	60 12 06 00 54 12 06 00 3A 12 06 00 2A 12 06
00460AFC	1A 12 06 00 02 12 06 00 EE 11 06 00 D8 11 06
00460B0C	C8 11 06 00 B4 11 06 00 AC 11 06 00 8E 11 06

Nhấn F8 để trace code, tới lệnh nhảy **JE** tại **0046651A** / **0F84 93000000** je **UnPackMe.004665B3**, quan sát cửa sổ **Registers**, ta thấy thanh ghi **EBX** lúc này trỏ tới tên hàm API **GetVersion()**:

Registers (FPU)

EAX	00000011
ECX	009E06F7
EDX	00400000 UnPackMe.00400000
EBX	004612A2 ASCII "GetVersion"
ESP	0012FFA0
EBP	0005995E
ESI	00460014 UnPackMe.00460014
EDI	009E17D4
EIP	0046651A UnPackMe.0046651A
C 0	ES 0023 32bit 0(FFFFFFFF)

Trace tiếp xuống dưới sẽ thấy lời gọi tới hàm API **GetProcAddress()** để tìm địa chỉ của hàm API **GetVersion()** trên máy của tôi.

Registers (FPU)

EAX	00000011
ECX	009E06F7
EDX	00400000 UnPackMe.00400000
EBX	004612A2 ASCII "GetVersion"
ESP	0012FF98
EBP	0005995E
ESI	00460014 UnPackMe.00460014
EDI	009E17D4
FTP	00466574 UnPackMe.00466574

```

0012FF94 0046657A CALL to GetProcAddress from UnPackMe.00466574
0012FF98 7C800000 hModule = 7C800000 (kernel32)
0012FF9C 004612A2 ProcNameOrOrdinal = "GetVersion"
0012FFA0 00000000

```

Thực hiện lệnh call này, quan sát tại cửa sổ **Registers**, thanh ghi **EAX** lúc này sẽ lưu địa chỉ của hàm API **GetVersion()** (trên máy tôi):

Registers (FPU)

EAX	7C8114AB kernel32.GetVersion
ECX	7C919AEB ntdll.7C919AEB
EDX	7C97C0D8 ntdll.7C97C0D8
EBX	004612A2 ASCII "GetVersion"
ESP	0012FFA0
EBP	0005995E
ESI	00460014 UnPackMe.00460014
EDI	009E17D4
EIP	0046657A UnPackMe.0046657A

Tiếp tục trace tiếp tới đây:

Address	Disassembly	Comment
0046657C	jnz short UnPackMe.004665B1	
0046657E	pop eax	
0046657F	stc	
00466580	jb UnPackMe.004662E7	
00466586	inc edi	
00466587	inc esp	
00466588	dec ecx	

Nhấn **Enter** để follow tới địa chỉ đích của lệnh nhảy này, dừng lại tại đây:

Address	Disassembly	Comment
004665B3	mov dword ptr [edi], eax	
004665B4	pop eax	
004665B5	dec eax	
004665B7	je short UnPackMe.004665C4	
004665B8	inc eax	
004665B9	clc	
004665BD	mov word ptr [ebx-2], ax	
004665BE	mov byte ptr [ebx], al	
004665BF	inc ebx	
004665C0	cmp byte ptr [ebx], al	

Registers (FPU)

EAX	7C8114AB	kernel32.GetVersion
ECX	7C919AEB	ntdll.7C919AEB
EDX	7C97C0D8	ntdll.7C97C0D8
EBX	004612A2	ASCII "GetVersion"
ESP	0012FFA0	
EBP	0005995E	
ESI	00460014	UnPackMe.00460014
EDI	009E17D4	

Để ý thì thấy rằng, lệnh trên sẽ thực hiện lưu địa chỉ của hàm API nhưng lại ở một vùng nhớ khác chứ không phải thuộc IAT, vùng nhớ này trở bởi thanh ghi EDI như trên hình. Nếu follow tới địa chỉ 009E06F7, và quan sát code tại đây, ta sẽ thấy code ở đây sẽ nhảy tới địa chỉ của hàm API **GetVersion()** (như các bạn đã thấy khi trace bằng tay ở bài trước):

Address	Disassembly	Comment
009E06F7	test esp, esp	
009E06F9	jns short 009E06FE	
009E06FB	setno byte ptr [edx+40]	
009E06FD	mov eax, 7E177E00	
009E06FE	inc eax	
009E06FF	push dword ptr [eax]	
009E0700	ret	
009E0708	jmp short 009E070C	
009E070A	int 20	
009E070C	add eax, 7E177E1D	

Registers (FPU)

EAX	7C8114AB	kernel32.GetVersion
ECX	7C919AEB	ntdll.7C919AEB
EDX	7C97C0D8	ntdll.7C97C0D8
EBX	004612A2	ASCII "GetVersion"
ESP	0012FFA0	
EBP	0005995E	
ESI	00460014	UnPackMe.00460014
EDI	009E17D4	

OK, **F8** để trace qua khúc lưu địa chỉ API, ta tới đây:

Address	Disassembly	Comment
004665B1	mov dword ptr [edi], eax	
004665B3	pop eax	
004665B4	dec eax	
004665B5	je short UnPackMe.004665C4	
004665B7	inc eax	
004665B8	clc	
004665B9	mov word ptr [ebx-2], ax	
004665BD	mov byte ptr [ebx], al	
004665BE	inc ebx	
004665BF	inc ebx	
004665C0	cmp byte ptr [ebx], al	
004665C2	jnz short UnPackMe.004665BD	
004665C4	add dword ptr [ebp+40D34E], 4	
004665CB	jmp UnPackMe.00466538A	
004665D0	add esi, 14	

Như vậy, lệnh JMP ở trên sẽ nhảy ngược trở lại chỗ bắt đầu toàn bộ quá trình thực hiện, để ý trước khi tới lệnh nhảy này, đoạn code trên nó sử dụng thanh ghi al để xóa toàn bộ vùng nhớ chứa tên hàm API và địa chỉ API tại thanh ghi EAX cũng đã bị xóa. Nhấn **Enter** để follow tới địa chỉ của lệnh nhảy này:

0046639A	8B95 62D34000	mov	edx,dword ptr [ebp+40D362]
00466390	8B06	mov	eax,dword ptr [esi]
00466392	85C0	test	eax, eax
00466394	75 0B	jnz	short UnPackMe.004663A1
00466396	8B46 10	mov	eax,dword ptr [esi+10]
00466399	85C0	test	eax, eax
0046639B	0F84 46FFFFFF	je	UnPackMe.004662E7
004663A1	03C2	add	eax,edx
004663A3	0385 4ED34000	add	eax,dword ptr [ebp+40D34E]
004663A9	8B18	mov	ebx,dword ptr [eax]
004663AB	F7C3 00000080	test	ebx,80000000
004663B1	74 06	je	short UnPackMe.004663B9

Tại code trên, rõ ràng thấy rằng giá trị của EAX ở dòng thứ hai (00466390 8B06 `mov eax,dword ptr [esi]`) sẽ được thay thế bằng một giá trị khác và chắc chắn sẽ lặp lại chu trình cho entry tiếp theo, như vậy là ta đã tiếp cận được rất gần mục tiêu rồi.

Như đã biết, sẽ có hàm API bị redirect và không. Do đó, ý tưởng để tìm magic jump là sẽ theo dõi đối với hàm API bị redirect thì các lệnh nhảy có điều kiện nào sẽ thực sự nhảy, tương tự cũng sẽ theo dõi như thế đối với hàm API không bị redirect. Từ đó thực hiện so sánh để thu hẹp phạm vi. Để thuận tiện trong việc so sánh, tôi thường lập một bảng nhỏ để theo dõi những gì xảy ra với các lệnh nhảy có điều kiện của cả good IAT và bad IAT entry.

### 3.1. Theo dõi các lệnh nhảy của bad entry

Đối với các **bad entry** trong ví dụ này (Tôi sẽ sử dụng ảnh minh họa của để lưu lại tất cả các bước nhảy khi tôi thực hiện quá trình trace code, cho đến khi tôi đến được nơi thực hiện lưu giá trị). Tôi có như sau:

00466390	8B06	mov	eax,dword ptr [esi]
00466392	85C0	test	eax, eax
00466394	75 0B	jnz	short UnPackMe.004663A1
00466396	8B46 10	mov	eax,dword ptr [esi+10]
00466399	85C0	test	eax, eax
0046639B	0F84 46FFFFFF	je	UnPackMe.004662E7
004663A1	03C2	add	eax,edx
004663A3	0385 4ED34000	add	eax,dword ptr [ebp+40D34E]
004663A9	8B18	mov	ebx,dword ptr [eax]



00466390	8B06	mov	eax,dword ptr [esi]
00466392	85C0	test	eax, eax
00466394	75 0B	jnz	short UnPackMe.004663A1
00466396	8B46 10	mov	eax,dword ptr [esi+10]
00466399	85C0	test	eax, eax
0046639B	0F84 46FFFFFF	je	UnPackMe.004662E7
004663A1	03C2	add	eax,edx
004663A3	0385 4ED34000	add	eax,dword ptr [ebp+40D34E]
004663A9	8B18	mov	ebx,dword ptr [eax]
004663AB	F7C3 00000080	test	ebx,80000000
004663B1	74 06	je	short UnPackMe.004663B9
004663B3	8120 00000080	and	dword ptr [eax],80000000
004663B9	8B7E 10	mov	edi,dword ptr [esi+10]
004663BC	03FA	add	edi,edx
004663BE	80A5 D6CC4000	and	byte ptr [ebp+40CCD6],0FF
004663C5	0F84 30010000	je	UnPackMe.004664FB
004663A1	03C2	add	eax,edx
004663A3	0385 4ED34000	add	eax,dword ptr [ebp+40D34E]
004663A9	8B18	mov	ebx,dword ptr [eax]
004663AB	F7C3 00000080	test	ebx,80000000
004663B1	74 06	je	short UnPackMe.004663B9
004663B3	8120 00000080	and	dword ptr [eax],80000000
004663B9	8B7E 10	mov	edi,dword ptr [esi+10]
004663BC	03FA	add	edi,edx
004663BE	80A5 D6CC4000	and	byte ptr [ebp+40CCD6],0FF
004663C5	0F84 30010000	je	UnPackMe.004664FB
004663CB	80A5 D7CC4000	and	byte ptr [ebp+40CCD7],0FF
004663D2	0F84 23010000	je	UnPackMe.004664FB
004663D8	89BD 5AD44000	mov	dword ptr [ebp+40D45A],edi
004663BC	03FA	add	edi,edx
004663BE	80A5 D6CC4000	and	byte ptr [ebp+40CCD6],0FF
004663C5	0F84 30010000	je	UnPackMe.004664FB
004663CB	80A5 D7CC4000	and	byte ptr [ebp+40CCD7],0FF
004663D2	0F84 23010000	je	UnPackMe.004664FB
004663D8	89BD 5AD44000	mov	dword ptr [ebp+40D45A],edi
004663DE	8B85 52D44000	mov	eax,dword ptr [ebp+40D452]
004663E4	40	inc	eax
004663DE	8B85 52D44000	mov	eax,dword ptr [ebp+40D452]
004663E4	40	inc	eax
004663E5	0F84 10010000	je	UnPackMe.004664FB
004663EB	48	dec	eax
004663EC	0F85 B2000000	jnz	UnPackMe.004664A4
004663F2	60	pushad	
004663D8	89BD 5AD44000	mov	dword ptr [ebp+40D45A],edi
004663DE	8B85 52D44000	mov	eax,dword ptr [ebp+40D452]
004663E4	40	inc	eax
004663E5	0F84 10010000	je	UnPackMe.004664FB
004663EB	48	dec	eax
004663EC	0F85 B2000000	jnz	UnPackMe.004664A4
004663F2	60	pushad	
004663F3	8BF7	mov	esi,edi
004663F5	2BC0	sub	eax, eax
004663F7	40	inc	eax

```

004664A4 8D85 14BB4000 lea     eax,dword ptr [ebp+40BB14]
004664AA FFB5 FBCA4000 push   dword ptr [ebp+40CAFB]
004664B0 0FB608        movzx   ecx,byte ptr [eax]
004664B3 FF0C24        dec     dword ptr [esp]
004664B6 7E 05        jle     short UnPackMe.004664BD
004664B8 40          inc     eax
004664B9 03C1        add     eax,ecx
004664BE EB F3        jmp     short UnPackMe.004664BD
004664BD 890C24        mov     dword ptr [esp],ecx

```

Hình trên một miniloop, sau khi trace qua ta tới đây:

```

00466501 85DB        test    ebx,ebx
00466503 0F84 C7000000 je      UnPackMe.004665D0
00466509 F7C3 00000080 test    ebx,80000000
0046650F 6A 00        push    0
00466511 75 0F        jnz     short UnPackMe.00466522
00466513 8D5C13 02    lea     ebx,dword ptr [ebx+edx+2]
00466517 803B 00        cmp     byte ptr [ebx],0
0046651A 0F84 93000000 je      UnPackMe.004665B3
00466520 EB 45        jmp     short UnPackMe.00466567
00466522 FF0424        inc     dword ptr [esp]
00466525 66 85DB      test    bx,bx

00466501 85DB        test    ebx,ebx
00466503 0F84 C7000000 je      UnPackMe.004665D0
00466509 F7C3 00000080 test    ebx,80000000
0046650F 6A 00        push    0
00466511 75 0F        jnz     short UnPackMe.00466522
00466513 8D5C13 02    lea     ebx,dword ptr [ebx+edx+2]
00466517 803B 00        cmp     byte ptr [ebx],0
0046651A 0F84 93000000 je      UnPackMe.004665B3
00466520 EB 45        jmp     short UnPackMe.00466567
00466522 FF0424        inc     dword ptr [esp]
00466525 66 85DB      test    bx,bx

00466511 75 0F        jnz     short UnPackMe.00466522
00466513 8D5C13 02    lea     ebx,dword ptr [ebx+edx+2]
00466517 803B 00        cmp     byte ptr [ebx],0
0046651A 0F84 93000000 je      UnPackMe.004665B3
00466520 EB 45        jmp     short UnPackMe.00466567
00466522 FF0424        inc     dword ptr [esp]
00466525 66 85DB      test    bx,bx

0046655C 8B041A        mov     eax,dword ptr [edx+ebx]
0046655F 0385 4AD34000 add     eax,dword ptr [ebp+40D34A]
00466565 EB 13        jmp     short UnPackMe.0046657A
00466567 81E3 FFFFFFFF and     ebx,7FFFFFFF
0046656D 53          push    ebx
0046656E FFB5 4AD34000 push   dword ptr [ebp+40D34A]
00466574 FF95 E0BA4000 call    near dword ptr [ebp+40BAE0] kernel32.GetProcAddress
0046657A 40          inc     eax

```

Trace tới lệnh gọi API **GetProcAddress()** để lấy địa chỉ API và sau đó nhảy nơi thực hiện lưu kết quả:

```

00466574 FF95 E0BA4000 call    near dword ptr [ebp+40BAE0]
0046657A 40          inc     eax
0046657B 48          dec     eax
0046657C 75 33        jnz     short UnPackMe.004665B1
0046657E 58          pop     eax
0046657F F9          stc
00466580 0F82 61FDFFFF jb      UnPackMe.004662E7

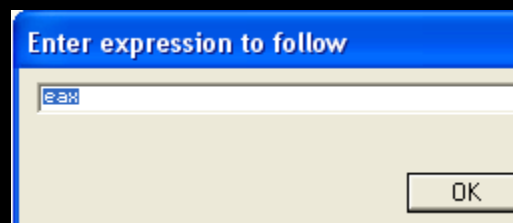
```



Giá trị tại thanh ghi EAX lúc này sẽ là địa chỉ của hàm API:

```
Registers (FPU)
EAX 77124C3B
ECX 7C919AEB ntdll.7C919AEB
EDX 7C97C0D8 ntdll.7C97C0D8
EBX 00000007
ESP 0012FFA0
EBP 0005995E
ESI 004600DC UnPackMe.004600DC
EDI 00460BAC UnPackMe.00460BAC
EIP 004665B1 UnPackMe.004665B1
```

Như trên hình, ta thấy rằng EAX lưu địa chỉ của một hàm API, tuy nhiên ta không biết chính xác tên hàm. Nhấn **Ctrl+G (Goto Expression)**, nhập EAX:



77124C3B	8BFF	mov	edi,edi
77124C3D	55	push	ebp
77124C3E	8BEC	mov	ebp,esp
77124C40	8B45 08	mov	eax,dword ptr [ebp+8]
77124C43	85C0	test	eax,ecx
77124C45	74 05	je	short OLEAUT32.77124C4C
77124C47	8B40 FC	mov	eax,dword ptr [eax-4]
77124C4A	D1E8	shr	eax,1
77124C4C	5D	pop	ebp
77124C4D	C2 0400	retn	4

Ta sẽ tới hàm API, quay trở về code của Crackme sẽ thấy tên hàm hiện ra như sau:

```
Registers (FPU)
EAX 77124C3B OLEAUT32.SysStringLen
ECX 7C919AEB ntdll.7C919AEB
EDX 7C97C0D8 ntdll.7C97C0D8
EBX 00000007
ESP 0012FFA0
EBP 0005995E
ESI 004600DC UnPackMe.004600DC
EDI 00460BAC UnPackMe.00460BAC
EIP 004665B1 UnPackMe.004665B1
C 0 ES 0023 32bit 0(FFFFFFFF)
D 0 CS 001B 32bit 0(FFFFFFFF)
```

Giờ chúng ta sẽ thực hiện lại công việc mà ta đã làm với bad API, và hy vọng là địa chỉ tiếp theo đây mà chúng ta thực hiện sẽ là một hàm API chuẩn, còn nếu không phải thì ta sẽ phải thực tiếp tục cho đến khi ta tìm được một hàm API chuẩn mới thôi.

Chúng ta sẽ bắt đầu từ đầu vòng lặp và tiến hành theo dõi toàn bộ các lệnh nhảy có điều kiện:

```

0046638A 8B95 62D34000 mov     edx,dword ptr [ebp+40D362]
00466390 8B06          mov     eax,dword ptr [esi]
00466392 85C0          test    eax,eax
00466394 75 0B          jnz     short UnPackMe.004663A1
00466396 8B46 10        mov     eax,dword ptr [esi+10]
00466399 85C0          test    eax,eax
0046639B 0F84 46FFFFFF je      UnPackMe.004662E7
004663A1 03C2          add     eax,edx

```

Lệnh nhảy trên tương tự như với bad API, do đó ta bỏ qua. Tiếp tục:

```

004663A9 8B18          mov     ebx,dword ptr [eax]
004663AB F7C3 00000080 test    ebx,80000000
004663B1 74 06          je      short UnPackMe.004663B9
004663B3 8120 00000080 and     dword ptr [eax],80000000
004663B9 8B7E 10        mov     edi,dword ptr [esi+10]
004663BC 03FA          add     edi,edx

```

Ở khúc liên quan tới bad API thì lệnh trên sẽ nhảy, tuy nhiên lệnh này có bước nhảy ngắn nên ta suy đoán nó không thể là magic jump, chúng ta hãy tiếp tục.

```

004663BC 03FA          add     esi,eax
004663BE 80A5 D6CC4000 and     byte ptr [ebp+40CCD6],0FF
004663C5 0F84 30010000 je      UnPackMe.004664FB
004663CB 80A5 D7CC4000 and     byte ptr [ebp+40CCD7],0FF
004663D2 0F84 23010000 je      UnPackMe.004664FB
004663D8 89BD 5AD44000 mov     dword ptr [ebp+40D45A],edi
004663DE 8B85 52D44000 mov     eax,dword ptr [ebp+40D452]
004663E4 40            inc     eax

```

Không nhảy, giống như đối với bad API, do đó bỏ qua và tiếp tục:

```

004663C5 0F84 30010000 je      UnPackMe.004664FB
004663CB 80A5 D7CC4000 and     byte ptr [ebp+40CCD7],0FF
004663D2 0F84 23010000 je      UnPackMe.004664FB
004663D8 89BD 5AD44000 mov     dword ptr [ebp+40D45A],edi
004663DE 8B85 52D44000 mov     eax,dword ptr [ebp+40D452]
004663E4 40            inc     eax
004663E5 0F84 10010000 je      UnPackMe.004664FB

```

Ở đây, ta thấy một sự khác biệt lớn, lệnh này sẽ nhảy, trong khi đối với bad API nó không nhảy và điểm quan trọng nữa là bước nhảy của lệnh này tương đối dài.

004663D2	0F84 23010000	je	UnPackMe.004664FB
004663D8	89BD 5AD44000	mov	dword ptr [ebp+40D45A],edi
004663DE	8B85 52D44000	mov	eax,dword ptr [ebp+40D452]
004663E4	40	inc	eax
004663E5	0F84 10010000	je	UnPackMe.004664FB
004663EB	48	dec	eax
004663EC	0F85 B2000000	jnz	UnPackMe.004664A4
004663F2	60	pushad	
004663F3	8BF7	mov	esi,edi
004663F5	2BC0	sub	eax,edx
004663F7	40	inc	eax
004663F8	833F 00	cmp	dword ptr [edi],0
004663FB	8D7F 04	lea	edi,dword ptr [edi+4]
004663FE	75 F7	jnz	short UnPackMe.004663F7
00466400	48	dec	eax
00466401	0F84 EC000000	je	UnPackMe.004664F3
00466407	8BD8	mov	ebx,edx
00466409	6BC0 31	imul	eax,edx,31
0046640C	6A 04	push	4
0046640E	68 00100000	push	1000
00466413	50	push	eax
00466414	6A 00	push	0
00466416	FF95 ECBA4000	call	near dword ptr [ebp+40BAEC]
0046641C	85C0	test	eax,edx
0046641E	0F84 CF000000	je	UnPackMe.004664F3
00466424	8BFE	mov	edi,esi
00466426	8BCB	mov	ecx,ebx
00466428	8BF8	mov	edi,edx
0046642A	8985 56D44000	mov	dword ptr [ebp+40D456],eax
00466430	8BCB	mov	ecx,ebx
00466432	6BDB 29	imul	ebx,ebx,29
00466435	03DF	add	ebx,edi
00466437	891C24	mov	dword ptr [esp],ebx
0046643A	B0 B8	mov	al,0B8
0046643C	6A 00	push	0
0046643E	50	push	eax
0046643F	53	push	ebx
00466440	0FB74424 08	movzx	eax,word ptr [esp+8]
00466445	50	push	eax
00466446	8D85 14BB4000	lea	eax,dword ptr [ebp+40BB14]
0046644C	0FB74424 08	movzx	ebx,word ptr [esp+8]
Jump is taken			
004664FB=UnPackMe.004664FB			

Ta thấy đây là một bước nhảy khá dài, vậy rất có thể đây chính là sự khác biệt quyết định giữa việc một bad API hay good API được lưu, hoặc nếu mọi thứ diễn ra như chúng ta tưởng tượng thì khả năng đây sẽ là lệnh ***magic jump*** - *một lệnh nhảy để quyết định việc lưu một API hợp lệ*, có nghĩa là để API hợp lệ được lưu thì lệnh nhảy này bắt buộc phải nhảy. Do đó, chỉ có một khả năng để làm điều đó là thay thế **JE** bằng lệnh **JMP**.

Tuy nhiên, lệnh nhảy này không phải xuất hiện ngay lúc đầu khi ta load unpackme, vậy nên chúng ta phải rất cẩn thận. Nếu chúng ta khởi động lại OllyDbg và tìm kiếm theo địa chỉ của lệnh nhảy này:

004663D2	E8 4278605E	call	SEA6DC19
004663D7	7E CF	jle	short UnPackMe.004663A8
004663D9	3095 64EA3501	xor	byte ptr [ebp+135EA64],dl

Với kết quả trên hình, ta thấy ban đầu nó là một đoạn code khác, như vậy trình packer sẽ tạo ra lệnh nhảy này sau đó. **Follow in Dump** tại địa chỉ này:

004663D2	E8 4278605E	call	SEA6DC19
004663D7	7E CF	jle	short UnPackMe.004663A8
004663D9	3095 64EA3501	xor	byte ptr [ebp+135EA64],dl
004663DF	E6 D3	out	0D3, eax
004663E1	9C	pushfd	
004663E2	BC 2F99553C	mov	esp, 2F99553C
004663E7	A2 9E8CDD6B	mov	byte ptr [esi],al
004663EC	82E9 27	sub	cl, 27
004663EF	9D	popfd	
004663F0	8751 A5	xchg	dword ptr [esi],eax
004663F3	9D	popfd	
004663F4	65 2F	das	
004663F6	4D	dec	ebp
004663F7	72 44	jb	short UnPackMe.004663A8
004663F9	56	push	esi

Giờ vấn đề là làm thế nào để dừng lại tại đó? Tôi sẽ đặt một **BPM On Write** bao gồm toàn bộ IAT để nó dừng lại khi nó lưu được giá trị đầu tiên. Như đã biết IAT bắt đầu tại 460818 và kết thúc tại 460f28, chọn toàn bộ khu vực này và đặt một **BPM On Write**.

00466412	53	push	edi
00466414	68 3F40	mov	eax, 3F4068
00466419	57	push	edi
0046641A	97	push	edi
0046641B	882A	mov	eax, 882A
0046641D	53	push	edi
0046641E	78 18	mov	eax, 1878
00466420	E1 DF	mov	ecx, DF E1

OK, tất cả các IAT đã được thiết lập với **BPM On Write**, bây giờ ta đến nơi mà nó lưu giá trị đầu tiên.

00465D91	AA	STOS BYTE PTR ES:[EDI]	
00465D92	69D2 A5B0CD4B	IMUL EDX,EDX,4BCDB0A5	
00465D98	F9	STC	
00465D99	72 02	JB SHORT 00465D9D	UnP
00465D9B	CD 20	INT 20	
00465D9D	D1C2	ROL EDX,1	
00465D9F	69DB 701FEE6A	IMUL EBX,EBX,6AEE1F70	
00465DA5	0000	0000_FBX_FBX	

Lệnh STOS được thực hiện rất nhiều lần vì nó gồm toàn bộ IAT mà ta đã đặt BPM, do vậy chúng ta hãy xem liệu vào thời điểm này thì lệnh nhảy được giả thiết là magic jump tại địa chỉ **0x4663D2** đã xuất hiện chưa. Tới địa chỉ này ta thấy:

004663D2	0F84 23010000	je	UnPackMe.004664FB
004663D8	89BD 5AD44000	mov	dword ptr [ebp+40D45A],edi
004663DE	8B85 52D44000	mov	eax,dword ptr [ebp+40D452]
004663E4	40	inc	eax
004663E5	0F84 10010000	je	UnPackMe.004664FB
004663EB	48	dec	eax
004663EC	0F85 B2000000	jnz	UnPackMe.004664A4

Ok, như trên hình ta thấy trình packer đã thay thế lệnh call ban đầu bằng lệnh nhảy rồi. Ta xem thử điều gì sẽ xảy ra nếu chúng ta thay đổi nó thành JMP, bỏ BPM và đến OEP.

004663D2	E9 24010000	jmp	UnPackMe.004664FB
004663D7	90	nop	
004663D8	89BD 5AD44000	mov	dword ptr [ebp+40D45A],edi
004663DE	8B85 52D44000	mov	eax,dword ptr [ebp+40D452]

Có hai khả năng sẽ xảy ra lúc này:

- Một là packer phát hiện ra việc ta patch lệnh nhảy trên và thực hiện không thành công như ta mong muốn.
- Hai là hoạt động đúng như ta muốn và đến được OEP.

Giờ chỉ còn cách là tới luôn thôi, tới OEP đặt 1 BP, sau đó nhấn **F9** để run:

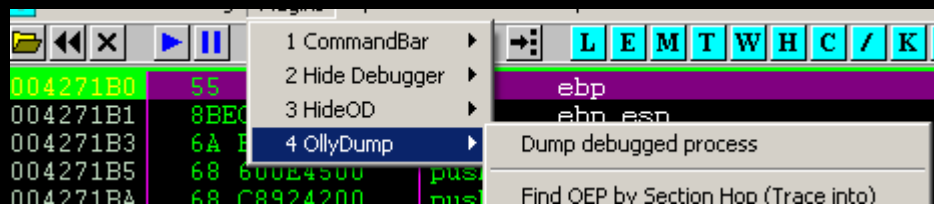
004271B0	55	push	ebp	
004271B1	8BEC	mov	ebp,esp	
004271B3	6A FF	push	-1	
004271B5	68 600E4500	push	UnPackMe.00450E60	
004271BA	68 C8924200	push	UnPackMe.004292C8	
004271BF	64:A1 00000000	mov	eax,dword ptr fs:[0]	
004271C5	50	push	eax	
004271C6	64:8925 000000	mov	dword ptr fs:[0],esp	
004271CD	83C4 A8	add	esp,-58	
004271D0	53	push	ebx	
004271D1	56	push	esi	
004271D2	57	push	edi	
004271D3	8965 E8	mov	dword ptr [ebp-18],esp	
004271D6	FF15 DC0A4600	call	near dword ptr [460ADC]	kernel32.GetVersion
004271DC	33D2	xor	edx,edx	
004271DE	8AD4	mov	dl,ah	
004271E0	8915 34E64500	mov	dword ptr [45E634],edx	

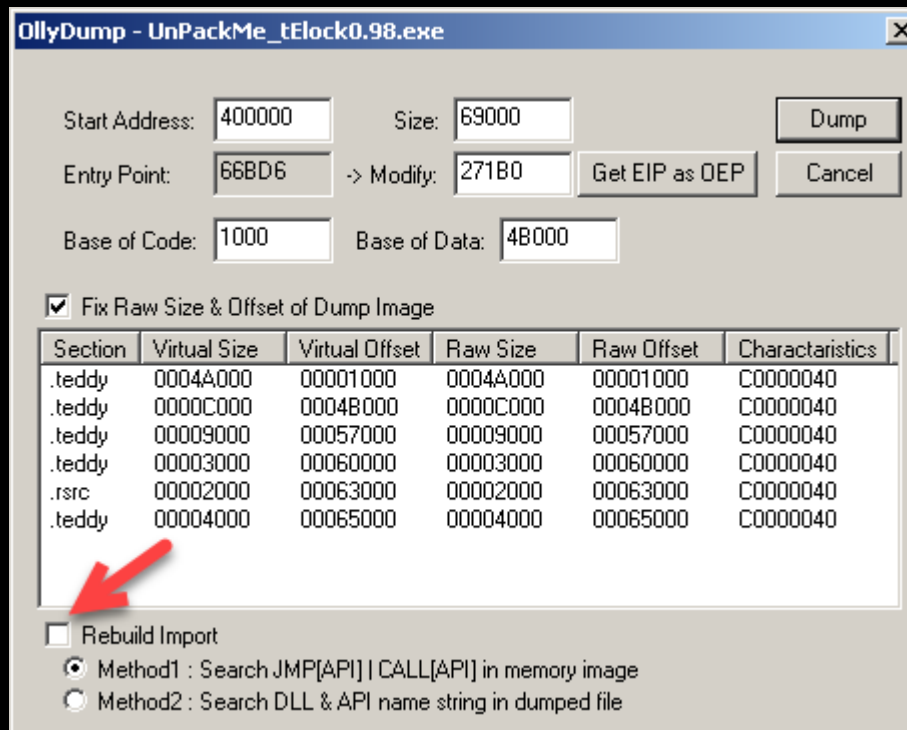


Ok, ta tới OEP một cách bình thường, lúc này quan sát tại lệnh call bên dưới, khi chưa fix lệnh nhảy ta sẽ thấy lời gọi tới hàm **GetVersion()** sẽ bị redirect, nhưng lúc này tên hàm đã hiển thị đúng luôn rồi. Chuyển tới cửa sổ dump và quan sát toàn bộ bảng IAT:

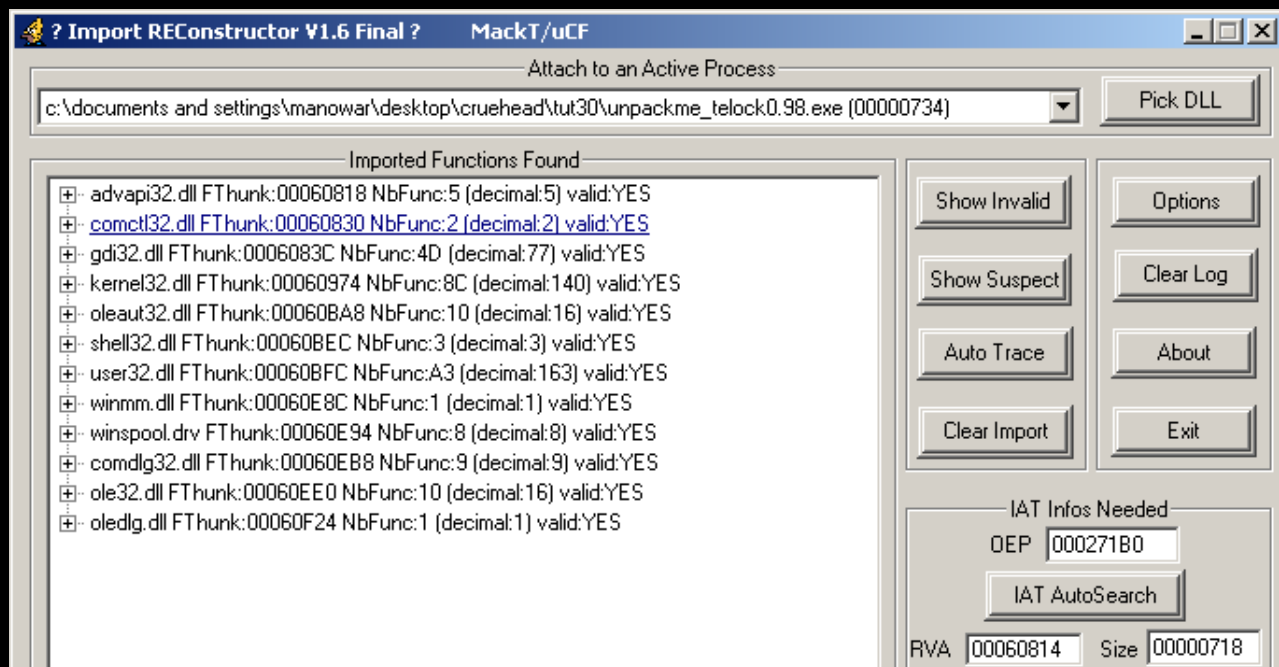
Address	Value	Comment
00460818	77DD6BF0	ADVAPI32.RegCloseKey
0046081C	77DD761B	ADVAPI32.RegOpenKeyExA
00460820	77DDEAF4	ADVAPI32.RegCreateKeyExA
00460824	77DDEBE7	ADVAPI32.RegSetValueExA
00460828	77DD7883	ADVAPI32.RegQueryValueExA
0046082C	00000000	
00460830	5D0B15DD	COMCTL32.InitCommonControls
00460834	5D09BD2E	COMCTL32.ImageList_Destroy
00460838	00000000	
0046083C	77F168E4	GDI32.GetClipBox
00460840	77F18665	GDI32.ExcludeClipRect
00460844	77F16899	GDI32.IntersectClipRect
00460848	77F19C60	GDI32.MoveToEx
0046084C	77F19D07	GDI32.LineTo
00460850	77F19921	GDI32.SetTextAlign
00460854	77F1D4AA	GDI32.SetPixelV
00460858	77F17B2D	GDI32.GetViewportExtEx
0046085C	77F17AB5	GDI32.GetWindowExtEx
00460860	77F44F47	GDI32.PtVisible
00460864	77F3C433	GDI32.ScaleWindowExtEx
00460868	77F1C449	GDI32.TextOutA
0046086C	77F19012	GDI32.ExtTextOutA
00460870	77F1AB59	GDI32.GetMapMode
00460874	77F1BFE7	GDI32.DPtoLP
00460878	77F1CE55	GDI32.CreateDCA
0046087C	77F18195	GDI32.LPtoDP
00460880	77F3003E	GDI32.GetCharWidthA
00460884	77F43532	GDI32.SetAbortProc
00460888	77F1D35B	GDI32.GetPixel
0046088C	77F19B6C	GDI32.CreatePen
00460890	77F15FF1	GDI32.GetStockObject
00460894	77F186B0	GDI32.PatBlt
00460898	77F2F440	GDI32.SetBoundsRect

Tất cả các hàm API đã được fix chính xác, hehe như vậy tôi đã có được bảng IAT hoàn chỉnh. Giờ các bạn biết các bước tiếp theo phải làm gì rồi:





Rebuild lại toàn bộ IAT bằng ImpREC:



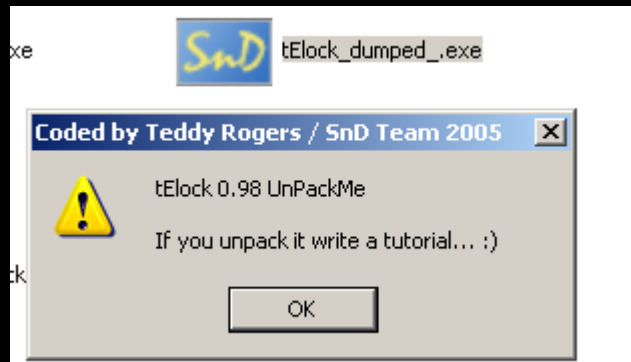
Toàn bộ IAT đã valid hết. Tiến hành Fix Dump:

```

Log
C (decimal:12) module(s)
1B9 (decimal:441) imported function(s).
*** New section added successfully. RVA:00069000 SIZE:00003000
Image Import Descriptor size: F0; Total length: 262E
C:\Documents and Settings\manowar\Desktop\CrueHead\Tut30\tElock_dumped_.exe

```

Chạy thử file sau khi đã fix:



Chạy quá hoàn hảo



### III. Kết luận

Toàn bộ phần 31 đến đây là kết thúc. Như vậy, qua bài viết này tôi đã cố gắng trình bày cho các bạn thấy một phương pháp để tìm ra magic jump, tuy nhiên sẽ khác nhau giữa các trình packer, nhưng bằng việc so sánh và tìm kiếm điểm khác biệt giữa việc trình packer thực hiện với một bad API và good API, ta sẽ luôn có thể xác định được vị trí của lệnh nhảy đó một cách dễ dàng. Chúng ta sẽ tiếp tục thực hành với nhiều trình packer hơn trong các 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 **haggarr**), 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**