

2017

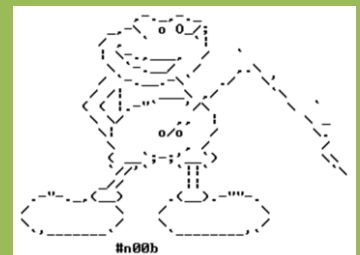
[Cracking with OllyDbg]

Based on OllyDbg tuts of Ricardo Narvaja (CrackLatinos Team)



www.reasonline.net

kienmanowar



22/05/2017

Mục Lục

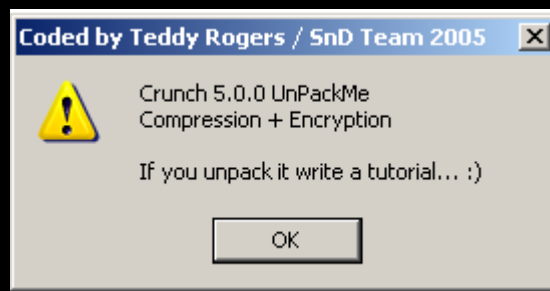
I. Giới thiệu chung.....	2
II. Phân tích và xử lý target.....	2
III. Kết luận	30

I. Giới thiệu chung

Vẫn tiếp tục với chủ đề liên quan tới Unpack, phần tiếp theo này sẽ nâng độ khó thêm một chút. Phần này sẽ thực hành với hai unpackme, một file được packed bởi *Crunch(5.0)*: **bitarts_evaluation.c.exe** và một được packed bởi *tElock(0.98)*: **UnPackMe_tElock0.98.exe**. Ngoài mục tiêu chính là unpack và chạy được file, phần này còn giới thiệu thêm về kĩ thuật chuyển hướng IAT hay còn được gọi với thuật ngữ là IAT Redirection. Kĩ thuật này được sử dụng trong nhiều trình packer/protector, cho phép hủy một phần hay toàn bộ IAT, nhưng lưu lại một pointer trở vào vùng code riêng cho từng hàm API đã được chuyển hướng. Hai unpackme này tôi đã gửi kèm trong phần 26.

II. Phân tích và xử lý target

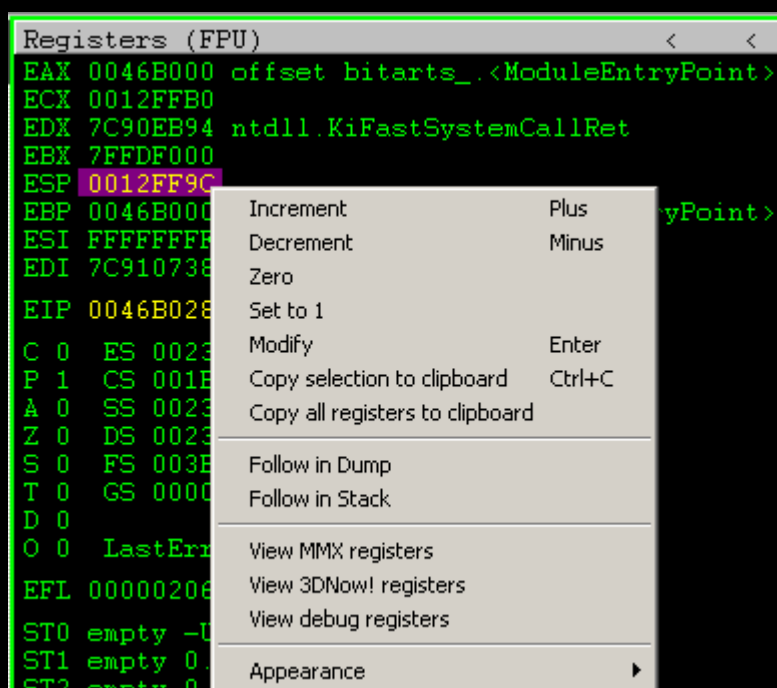
Chúng ta sẽ bắt đầu với file **bitarts_evaluation.c.exe** trước, chạy thử unpackme này xem thế nào:



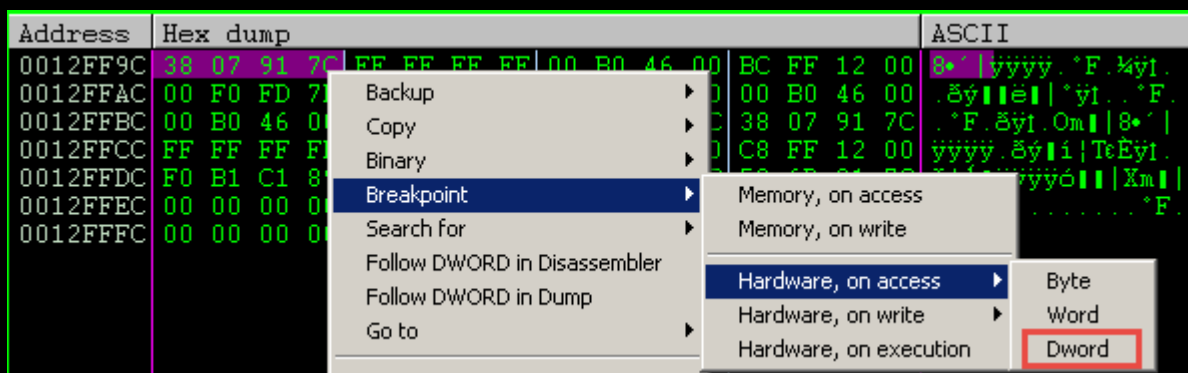
Theo thông tin thì Packer này có áp dụng **Compression + Encryption**. Chẳng biết nó thế nào, cứ load file này vào OllyDbg, ta dừng lại tại đây:

0046B000	EB 15	jmp	short bitarts 0046B017
0046B002	0300	add	eax,dword ptr [eax]
0046B004	0000	add	byte ptr [eax],al
0046B006	06	push	
0046B007	0000	add	byte ptr [eax],al
0046B009	0000	add	byte ptr [eax],al
0046B00B	0000	add	byte ptr [eax],al
0046B00D	0000	add	byte ptr [eax],al
0046B00F	0000	add	byte ptr [eax],al
0046B011	0068 00	add	byte ptr [eax],ch
0046B014	0000	add	byte ptr [eax],al
0046B016	0055 E8	add	byte ptr [ebp-18],dl
0046B019	0000	add	byte ptr [eax],al
0046B01B	0000	add	byte ptr [eax],al
0046B01D	5D	pop	ebp
0046B01E	81ED 1D000000	sub	ebp,1D
0046B024	8BC5	mov	eax,ebp
0046B026	55	push	ebp
0046B027	60	pushad	
0046B028	9C	pushfd	
0046B029	2B85 FC070000	sub	eax,dword ptr [ebp+7FC]
0046B02F	8985 E8070000	mov	dword ptr [ebp+7E8],eax
0046B035	FF7424 2C	push	dword ptr [ebp+2C]

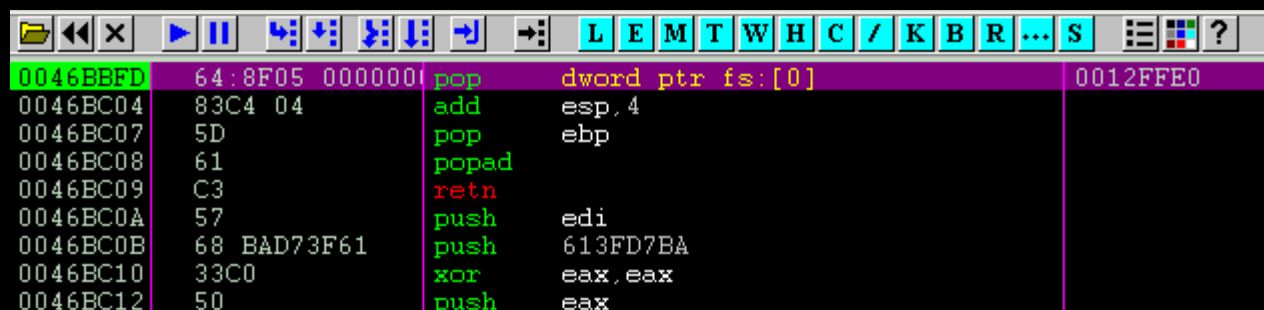
Quan sát một chút ta sẽ thấy có lệnh **PUSHAD** tại địa chỉ **0x46B027**, nhấn **F7/F8** để trace qua lệnh này. Sau đó chuyển qua cửa sổ **Registers**, chuột phải tại thanh ghi ESP và chọn **Follow in Dump**:



Chuyển qua cửa sổ Dump và đặt một **Breakpoint > Hardware, on access** như sau:



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



Nhấn **F7/F8** để trace qua lệnh `retn`, ta sẽ tới được OEP của unpackme:

004271B0	55	db	55	CHAR 'U'
004271B1	8B	db	8B	
004271B2	EC	db	EC	
004271B3	6A	db	6A	CHAR 'j'
004271B4	FF	db	FF	
004271B5	68	db	68	CHAR 'h'
004271B6	60	db	60	CHAR ''
004271B7	0E	db	0E	
004271B8	45	db	45	CHAR 'E'
004271B9	00	db	00	
004271BA	68	db	68	CHAR 'h'
004271BB	C8	db	C8	
004271BC	92	db	92	
004271BD	42	db	42	CHAR 'B'
004271BE	00	db	00	
004271BF	64	db	64	CHAR 'd'
004271C0	A1	db	A1	

Khi tới được OEP, lúc này code đã được bung hoàn toàn, do đó tại OllyDbg nhấn **Ctrl+A** để analyse code:

004271B0	55	push	ebp	
004271B1	8BEC	mov	ebp,esp	
004271B3	6A FF	push	-1	
004271B5	68 600E4500	push	bitarts_.00450E60	
004271BA	68 C8924200	push	bitarts_.004292C8	SE handler installation
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]	kernel32.GetVersion
004271DC	33D2	xor	edx,edx	
004271DE	8AD4	mov	dl,ah	
004271E0	8915 34E6450	mov	dword ptr [45E634],edx	
004271E6	8BC8	mov	ecx,ecx	
004271E8	81E1 FF00000	and	ecx,0FF	
004271EE	890D 30E6450	mov	dword ptr [45E630],ecx	

Kiểm tra tại cửa sổ **Memory Map**, ta thấy địa chỉ OEP ở trên nằm tại section **.text** (section đầu tiên sau PE header), đây là section chứa code thực thi của unpackme:

003E0000	00004000				Priv	Rw	Rw	
003F0000	00002000				Map	R	R	
00400000	00001000	bitarts_		PE header	Imag	R	RwE	
00401000	0004A000	bitarts_	.text	code	Imag	R	RwE	
0044B000	0000C000	bitarts_	.rdata		Imag	R	RwE	
00457000	00009000	bitarts_	.data	data	Imag	R	RwE	
00460000	00003000	bitarts_	.idata		Imag	R	RwE	
00463000	00008000	bitarts_	.rsrc	resources	Imag	R	RwE	
0046B000	00048000	bitarts_	.edata	SFX, imports	Imag	R	RwE	
004C0000	00009000				Map	R E	R E	

Quan sát code tại OEP, ta thấy hàm API đầu tiên được gọi là **GetVersion** (hàm này thuộc `kernel32.dll`):

					L E M T W H C / K B R ... S										
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]	kernel32.GetVersion									
004271DC	.	33D2		xor	edx, edx										
004271DE	.	8AD4		mov	dl, ah										
004271E0	.	8915	34E6450	mov	dword ptr [45E634], edx										

Tương tự như phần trước, ta nhận thấy rằng địa chỉ 0x460ADC là một phần của IAT, tại đó lưu trữ địa chỉ của hàm API tương ứng là **GetVersion**. Chuyển tới cửa sổ Dump để tiến hành xác minh các thông tin liên quan tới IAT:

Address	Hex dump																ASCII
00460ADC	AB	14	81	7C	F4	97	80	7C	01	B0	85	7C	19	62	82	7C	<[
00460AEC	5C	E8	81	7C	53	00	83	7C	19	3C	87	7C	CB	D8	81	7C	\è S.
00460AFC	C1	0F	87	7C	5B	B2	81	7C	E9	06	87	7C	4E	99	80	7C	À
00460B0C	AC	92	80	7C	11	07	87	7C	42	24	80	7C	F3	B8	81	7C	~'è
00460B1C	A9	2C	87	7C	F4	2C	87	7C	BA	38	87	7C	0C	6E	82	7C	©,
00460B2C	F1	BA	80	7C	3D	31	87	7C	83	31	87	7C	CC	37	87	7C	ñ² =
00460B3C	77	1D	80	7C	28	AC	80	7C	66	AA	80	7C	A9	2C	81	7C	wè (~è f àè ©,
00460B4C	ED	CB	81	7C	3D	0D	87	7C	19	90	83	7C	59	35	81	7C	iÈ = .
00460B5C	31	03	91	7C	40	03	91	7C	D7	EF	80	7C	2D	FF	80	7C	1' @
00460B6C	2F	FE	80	7C	51	28	81	7C	11	03	81	7C	B1	C7	80	7C	/
00460B7C	65	A0	80	7C	CF	C6	80	7C	21	2E	82	7C	BD	99	80	7C	e
00460B8C	88	2D	82	7C	5D	99	80	7C	94	97	80	7C	7B	97	80	7C	-
00460B9C	29	B5	80	7C	CF	C6	80	7C	00	00	00	00	C0	48	12	77) µè
00460BAC	3B	4C	12	77	94	A5	14	77	59	4B	12	77	82	4E	12	77	:
00460BBC	98	D4	14	77	9B	50	12	77	4F	50	12	77	10	50	12	77	
00460BCC	3F	50	12	77	D9	66	12	77	50	48	12	77	55	4C	12	77	?
00460BDC	C2	4B	12	77	95	D2	14	77	80	5D	18	77	00	00	00	00	À
00460BEC	B3	3F	A7	7C	A2	3F	A7	7C	44	FE	A0	7C	00	00	00	00	? ? S
00460BFC	A4	C9	D4	77	D5	B6	D8	77	BF	B5	D8	77	8A	C4	D4	77	µÈÖwÖlÖwÖµÖw
00460C0C	00	8E	D4	77	B0	EB	D6	77	ED	CD	D4	77	99	A2	D8	77	.
00460C1C	6E	B4	D4	77	98	EC	D6	77	BF	5E	D8	77	83	8E	D4	77	n 'Öw
00460C2C	BB	D7	D4	77	31	C5	D4	77	45	EA	D6	77	29	EC	D6	77	>>>Öw
00460C3C	66	C5	D4	77	03	8D	D4	77	EB	ED	D6	77	40	EC	D6	77	f
00460C4C	A1	E3	D4	77	46	BA	D8	77	DD	57	D8	77	61	5D	D6	77	j
00460C5C	DD	A1	D8	77	C5	B4	D4	77	B1	B4	D4	77	F9	8F	D4	77	Ý
00460C6C	19	00	D8	77	A8	67	D5	77	9F	74	D6	77	2B	EF	D7	77	.
00460C7C	D2	F7	D7	77	DE	D4	D4	77	15	D5	D4	77	5A	DC	D4	77	Ò - x w
00460C8C	D5	60	D6	77	D3	63	D6	77	C9	6C	D5	77	1E	DF	D4	77	Ö 'ÖwÖcÖwÖlÖwÖBÖw
00460C9C	41	FD	D5	77	2F	15	D6	77	54	05	D5	77	D7	B9	D4	77	AýÖw /
00460CAC	40	C6	D4	77	DC	E5	D4	77	A2	20	D5	77	D1	BD	D4	77	@ÈÖwÜàÖwÖc ÖwNÖw
00460CBC	13	CE	D4	77	CB	CD	D4	77	9F	CD	D4	77	38	04	D5	77	ÖwÈlÖw
00460CCC	2C	90	D4	77	42	01	D5	77	92	C5	D4	77	F3	BE	D4	77	.
00460CDC	FF	94	D4	77	CF	50	D6	77	4A	4D	D6	77	23	FE	D4	77	ý
00460CEC	5B	37	D5	77	E8	EE	D7	77	16	4F	D9	77	A4	52	D5	77	[7Öwèi x w ÖwÖRÖw
00460CFC	2E	F8	D6	77	79	C3	D4	77	6B	DF	D4	77	66	E6	D4	77	.
00460D0C	0B	19	D5	77	B2	02	D7	77	6E	ED	D4	77	FA	ED	D4	77	±
00460D1C	9F	F2	D6	77	3E	4E	D6	77	AE	C4	D4	77	A7	66	D5	77	
00460D2C	3C	EE	D4	77	4B	E3	D4	77	A2	EE	D4	77	10	C2	D4	77	<
00460D3C	E4	C6	D4	77	98	C2	D4	77	8E	C7	D4	77	00	8E	D4	77	àÈÖw
00460D4C	F1	F0	D6	77	D3	DE	D4	77	7C	94	D4	77	4D	72	D6	77	ñ
00460D5C	5A	35	D5	77	F7	EE	D4	77	71	DA	D6	77	E5	EE	D4	77	Z5Öw -
00460D6C	38	71	D6	77	2C	BF	D4	77	1B	71	D5	77	3C	FC	D4	77	8qÖw .
00460D7C	CE	E8	D4	77	CE	E8	D4	77	EF	01	D6	77	B7	F7	D4	77	ÎèÖwÎèÖwi .
00460D8C	BB	F6	D4	77	6F	F6	D4	77	32	E0	D4	77	80	F7	D4	77	>>
00460D9C	1A	8C	D4	77	06	8C	D4	77	BE	EA	D6	77	9A	4F	D6	77	-
00460DAC	A3	F7	D7	77	3F	36	D5	77	6C	71	D5	77	16	F1	D7	77	£ ÷ x w ? 6ÖwlqÖw
00460DBC	BC	F3	D7	77	6F	D1	D4	77	FD	CE	D4	77	CE	8B	D4	77	¼
00460DCC	BD	BC	D4	77	E9	93	D4	77	98	5C	D6	77	C6	F3	D6	77	¾
00460DDC	17	F1	D6	77	51	D0	D4	77	DE	A2	D4	77	0B	05	D8	77	{
00460DEC	AE	21	D5	77	A8	C6	D4	77	DB	B7	D4	77	85	F6	D7	77	©
00460DFC	D7	B5	D4	77	D4	F6	D7	77	BC	C6	D4	77	9D	B4	D4	77	x µÖwÖc x w ¼ÈÖw
00460E0C	0D	F5	D6	77	50	D4	D4	77	69	69	D8	77	21	BB	D8	77	.

Nhìn vào bảng này ta thấy khá dài và khó nhìn. Hiện tại, thông tin về OEP ta đã có, việc chính tiếp theo là xác định **IAT Start** và **IAT End** để tính ra được kích thước của IAT (IAT Size), phục vụ cho việc fix IAT sau này. Để dễ dàng trong việc xác định các giá

trị này, tại cửa sổ Dump sắp xếp lại theo kiểu **Long > Address**, ta có được thông tin về IAT như sau:

Address	Value	Comment
00460ADC	7C8114AB	kernel32.GetVersion
00460AE0	7C8097F4	kernel32.MulDiv
00460AE4	7C85B001	kernel32.RemoveDirectoryA
00460AE8	7C826219	kernel32.CreateDirectoryA
00460AEC	7C81E85C	kernel32.DeleteFileA
00460AF0	7C830053	kernel32.CopyFileA
00460AF4	7C873C19	kernel32.SetConsoleTextAttribute
00460AF8	7C81D8CB	kernel32.SetStdHandle
00460AFC	7C870FC1	kernel32.AllocConsole
00460B00	7C81B25B	kernel32.SetConsoleCtrlHandler
00460B04	7C8706E9	kernel32.GetConsoleTitleA
00460B08	7C80994E	kernel32.GetCurrentProcessId
00460B0C	7C8092AC	kernel32.GetTickCount
00460B10	7C870711	kernel32.SetConsoleTitleA
00460B14	7C802442	kernel32.Sleep
00460B18	7C81B8F3	kernel32.GetConsoleScreenBufferInfo
00460B1C	7C872CA9	kernel32.FillConsoleOutputCharacterA
00460B20	7C872CF4	kernel32.FillConsoleOutputAttribute
00460B24	7C8738BA	kernel32.SetConsoleCursorPosition
00460B28	7C826E0C	kernel32.GetDateFormatA
00460B2C	7C80BAF1	kernel32.SizeofResource
00460B30	7C87313D	kernel32.PeekConsoleInputA
00460B34	7C873183	kernel32.ReadConsoleInputA
00460B38	7C8737CC	kernel32.FlushConsoleInputBuffer
00460B3C	7C801D77	kernel32.LoadLibraryA
00460B40	7C80AC28	kernel32.GetProcAddress
00460B44	7C80AA66	kernel32.FreeLibrary
00460B48	7C812CA9	kernel32.GetStdHandle
00460B4C	7C81CBED	kernel32.WriteConsoleA
00460B50	7C870D3D	kernel32.FreeConsole
00460B54	7C839019	kernel32.FindNextFileA
00460B58	7C813559	kernel32.FindFirstFileA
00460B5C	7C910331	ntdll.RtlGetLastWin32Error
00460B60	7C910340	ntdll.RtlSetLastWin32Error
00460B64	7C80EFD7	kernel32.FindClose
00460B68	7C80FF2D	kernel32.GlobalAlloc
00460B6C	7C80FE2F	kernel32.GlobalFree
00460B70	7C812851	kernel32.GetVersionExA
00460B74	7C810311	kernel32.lstrcpynA
00460B78	7C80C7B1	kernel32.FindResourceA
00460B7C	7C80A065	kernel32.LoadResource
00460B80	7C80C6CF	kernel32.SetHandleCount
00460B84	7C822E21	kernel32.LocalUnlock
00460B88	7C8099BD	kernel32.LocalAlloc
00460B8C	7C822D88	kernel32.LocalLock
00460B90	7C80995D	kernel32.LocalFree
00460B94	7C809794	kernel32.InterlockedDecrement

Từ địa chỉ 0x460ADC, cuộn chuột lên trên để xác định IAT Start. Ta có thông tin IAT Start tại 00460818 77DD6BF0 advapi32.RegCloseKey như hình dưới đây:

Address	Value	Comment
004607D4	00062AE8	
004607D8	00062B1E	
004607DC	00062AD8	
004607E0	00062AC6	
004607E4	00062AAE	
004607E8	00062A92	
004607EC	00062A72	
004607F0	00062BBC	
004607F4	00062BA8	
004607F8	00062B92	
004607FC	00062B78	
00460800	00062B60	
00460804	00062B4C	
00460808	00062B2E	
0046080C	00000000	
00460810	80000008	
00460814	00000000	
00460818	77DD6BF0	advapi32.RegCloseKey
0046081C	77DD761B	advapi32.RegOpenKeyExA
00460820	77DDEAF4	advapi32.RegCreateKeyExA
00460824	77DDEBE7	advapi32.RegSetValueExA
00460828	77DD7883	advapi32.RegQueryValueExA
0046082C	00000000	

Tiếp theo cuộn chuột xuống dưới, ta xác định được IAT End tại 0x460F28 như hình dưới đây:

Address	Value	Comment
00460EFC	7751EA61	ole32.CreateILockBytesOnHGlobal
00460F00	7751EB91	ole32.StgCreateDocfileOnILockBytes
00460F04	7757A379	ole32.OleIsCurrentClipboard
00460F08	7757A529	ole32.OleFlushClipboard
00460F0C	7752431A	ole32.CoRevokeClassObject
00460F10	77552DA0	ole32.CoRegisterMessageFilter
00460F14	7752D1E0	ole32.CoFreeUnusedLibraries
00460F18	7753F356	ole32.CoGetClassObject
00460F1C	775DB375	ole32.StgOpenStorageOnILockBytes
00460F20	00000000	
00460F24	74D3F0F3	oledlg.OleUIBusyA
00460F28	00000000	
00460F2C	6C50000B	
00460F30	6F537961	
00460F34	41646E75	
00460F38	49570000	
00460F3C	2E4D4D4E	
00460F40	006C6C64	UNICODE "x&tCtrl+F6"
00460F44	654700FE	

Với hai giá trị có được, ta tính kích thước IAT như sau:

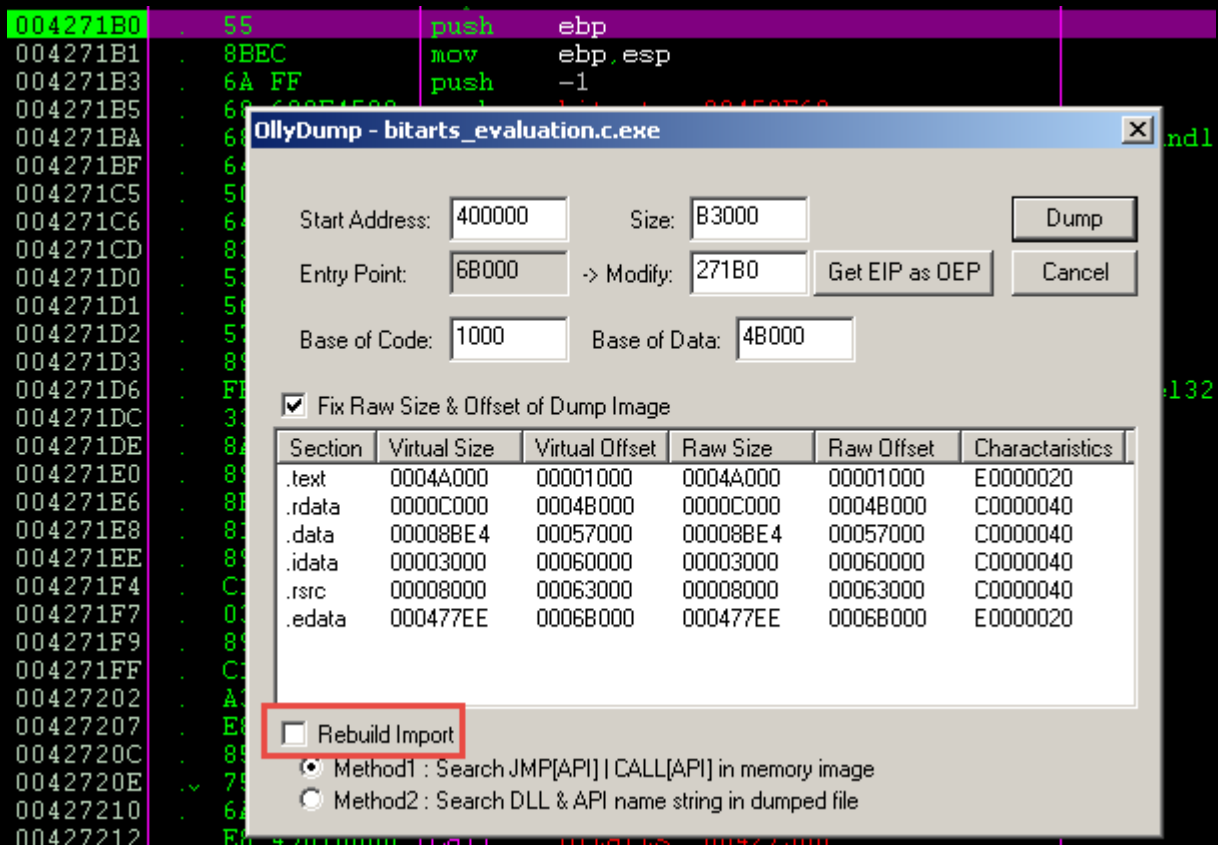
$$\text{IAT Size} = 0x460F28 - 0x460818 = 0x710$$



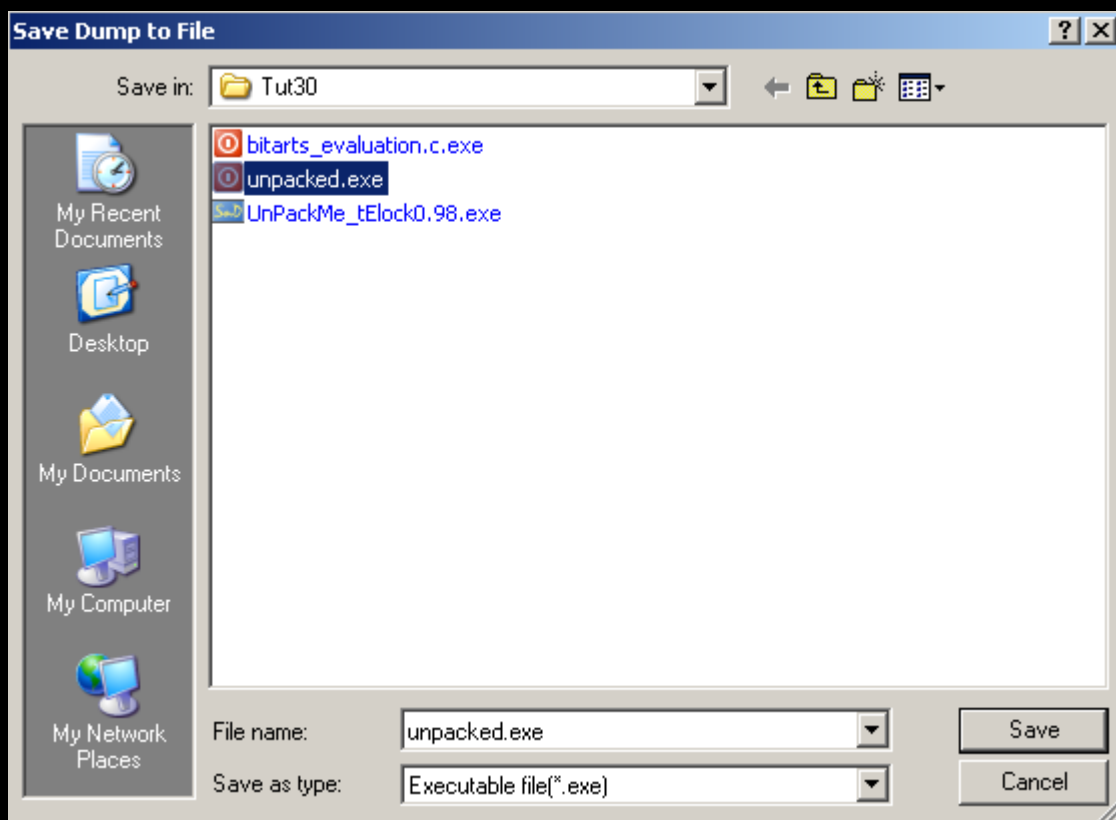
Như vậy, tổng kết lại các thông tin phục vụ cho việc fix lại file bằng ImpREC như sau:

- OEP= 0x271B0 (0x4271B0 - 0x400000)
- RVA hay IAT Start = 0x60818 (0x460818 - 0x400000)
- IAT Size= 0x710

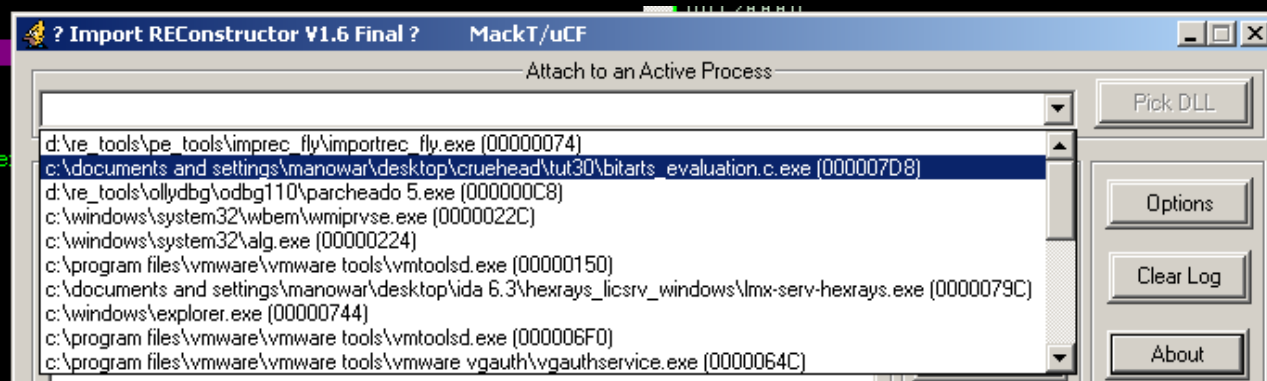
Để fix được file, trước tiên ta phải tiến hành Dump file tại OEP đã. Sử dụng **OllyDump** plugin đã giới thiệu ở phần trước (nhớ bỏ chọn **Rebuild Import**):



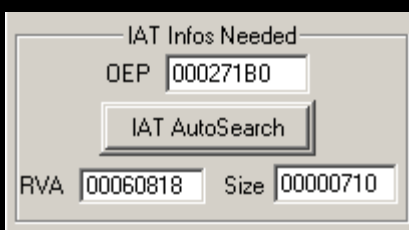
Sau đó, nhấn **Dump** và lưu lại với tên file là **unpacked.exe**:



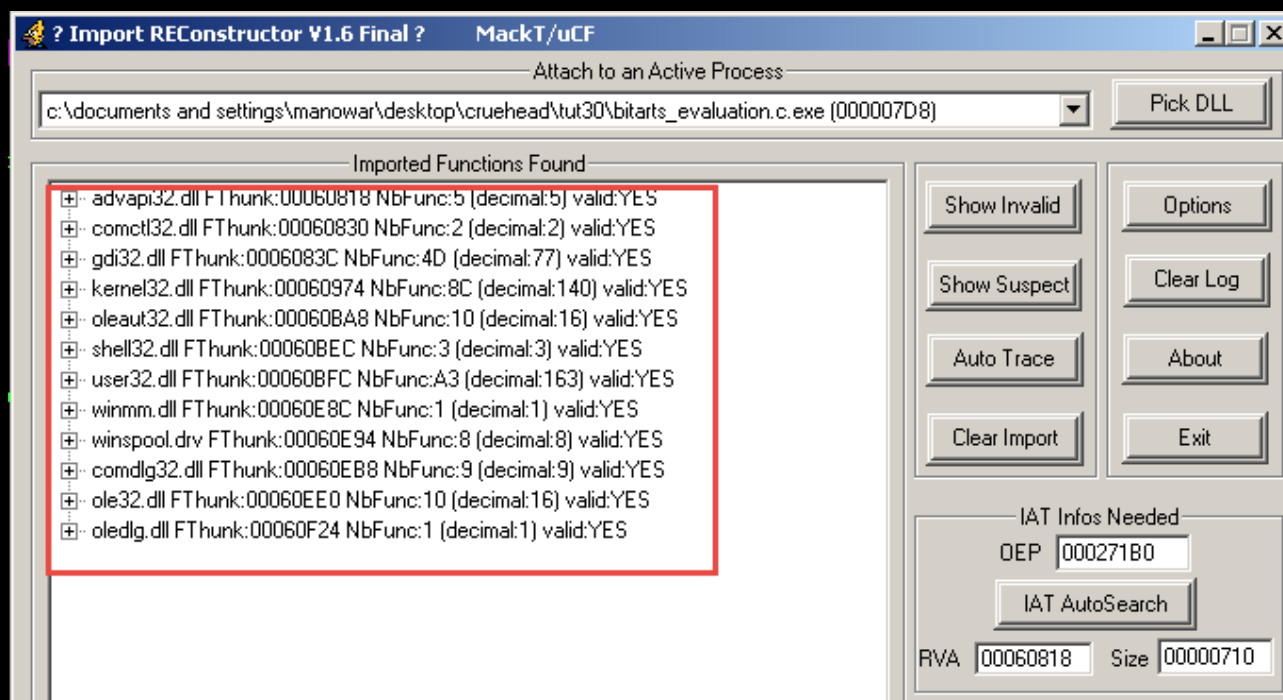
Giữ nguyên màn hình OllyDbg, mở ImpREC, lựa chọn Active Process như hình:



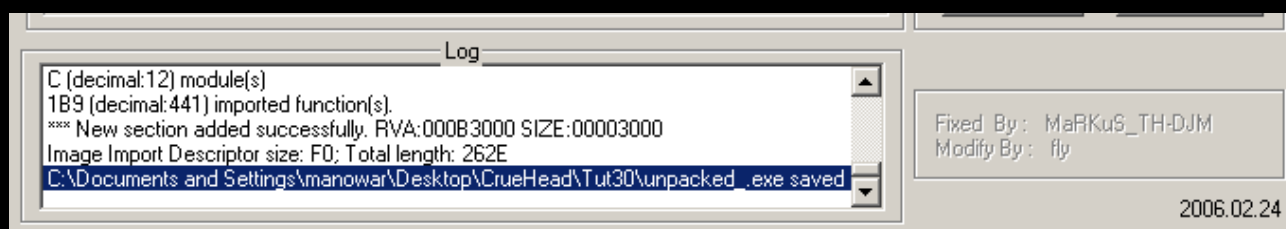
Điền các thông tin liên quan tại **IAT Infos Needed**:



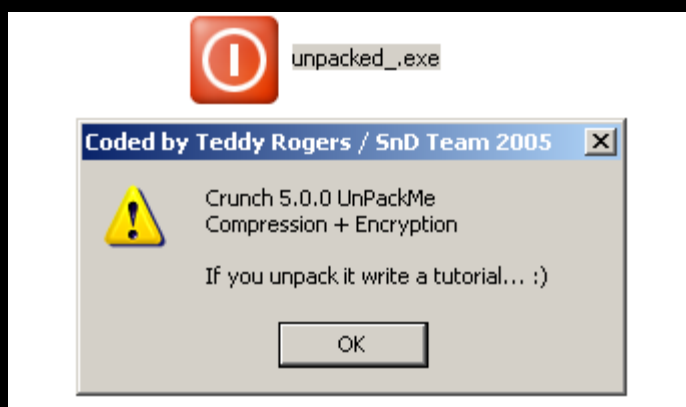
Sau đó nhấn **Get Imports** để lấy toàn bộ thông tin về Imported Functions:




Toàn bộ valid hết, quá đẹp . Bước cuối cùng là Fix Dump thôi:

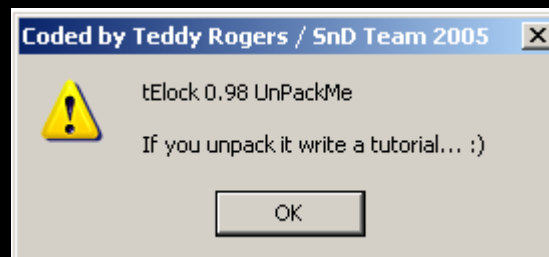


Chạy thử file **unpacked_.exe** mà ImpREC đã save lại ở bước trên, file run mượt mà không báo lỗi:



Như vậy là xong với unpackme được pack bằng Crunch. Chắc các bạn cảm thấy đơn giản quá!! .

Tiếp theo ta sẽ thực hành với unpackme được pack bởi **tElock 0.98**, packer này áp dụng kỹ thuật IAT Redirection để gây khó khăn trong quá trình thu thập thông tin IAT. Chạy thử unpackme này:



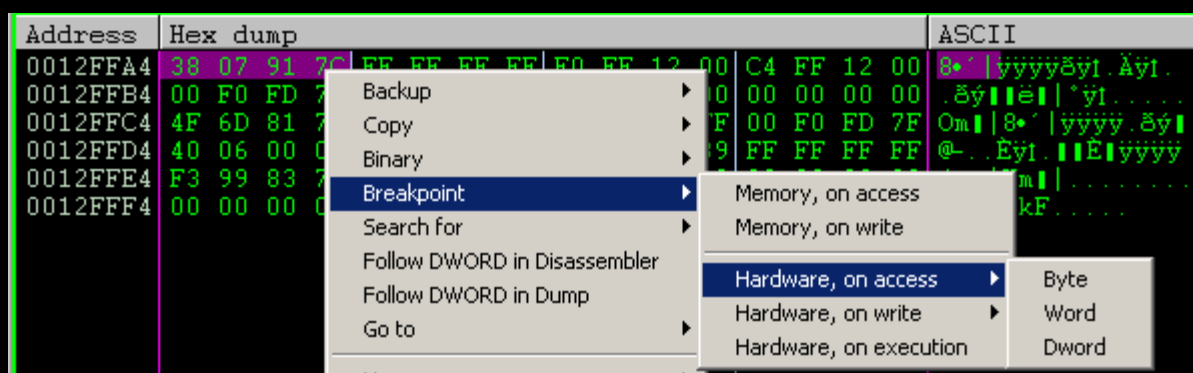
Không có thông tin gì đặc biệt được cung cấp kèm theo. Tiến hành load file vào OllyDbg, ta dừng lại tại đây:

Address	Disassembly	Comment
00466BD6	jmp	UnPackMe.00465000
00466BDB	add byte ptr [eax],al	
00466BDD	add byte ptr [eax+eax*2+6A],ah	
00466BE1	jnz short UnPackMe.00466C01	
00466BE3	ins byte ptr es:[edi],dx	I/O command
00466BE4	push	
00466BE5	add byte ptr [eax],al	
00466BE7	add byte ptr [eax],al	
00466BE9	add byte ptr [eax],al	
00466BEB	add byte ptr [eax],al	
00466BED	add byte ptr [esi],bh	
00466BEF	ins byte ptr es:[edi],dx	I/O command
00466BF0	push	

Quan sát code, nhận thấy không có lệnh **pushad** nào cả, thử trace code xem có thông tin gì không. Sau khi trace qua lệnh nhảy ta tới vùng code tại 0x465000, tại đây ta thấy có lệnh **pushad**.

Address	Disassembly	Comment
00465000	nop	
00465001	pushad	
00465002	call UnPackMe.00465009	
00465007	call 0047380C	
0046500C	add byte ptr [eax],al	
0046500E	pop esi	
0046500F	sub ecx,ecx	
00465011	pop eax	
00465012	je short UnPackMe.00465016	

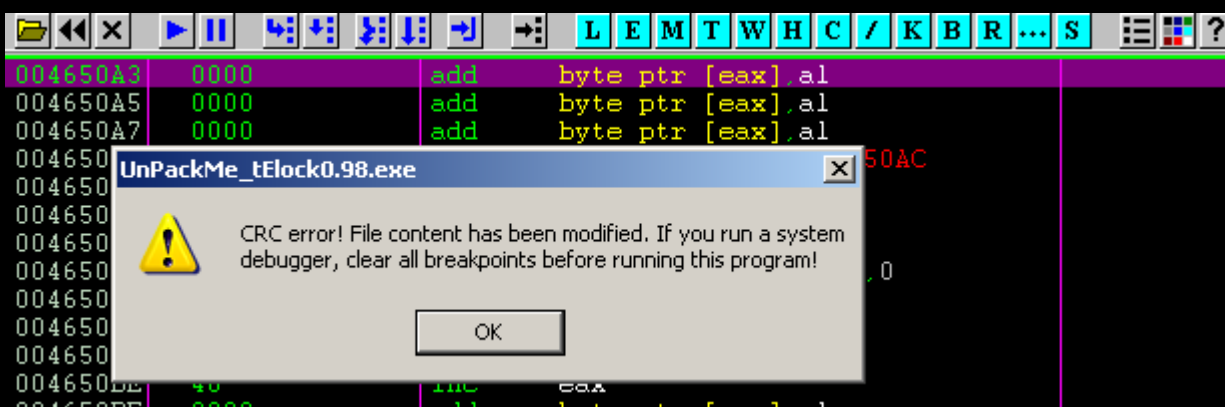
Trace qua lệnh **pushad**, chuột phải tại thanh ghi ESP và chọn **Follow in Dump**. Tại cửa sổ Dump thiết lập một **Breakpoint > Hardware, on access** như sau:



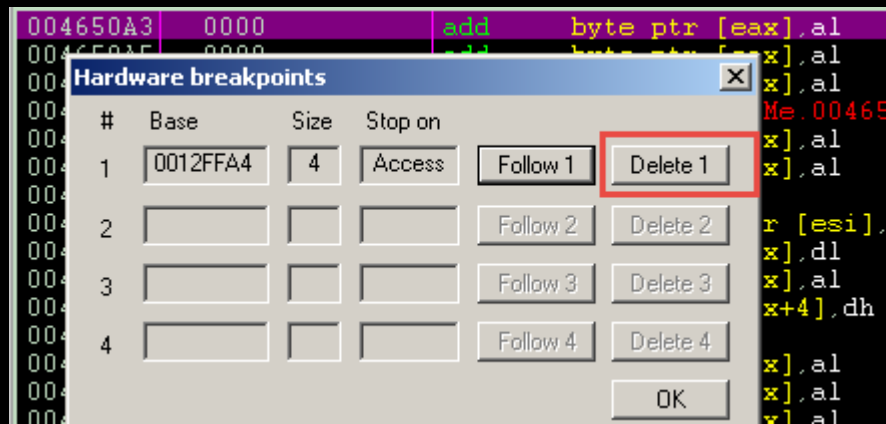
Sau khi đặt BP xong, nhấn **F9** để run, ta sẽ tới đây:

004650A3	90	nop	
004650A4	90	nop	
004650A5	33DB	xor	ebx,ebx
004650A7	F7F3	div	ebx
004650A9	64:67:8F06 000	pop	dword ptr fs:[0]
004650AF	83C4 04	add	esp,4
004650B2	66:BE 4746	mov	si,4647
004650B6	66:BF 4D4A	mov	di,4A4D
004650BA	8A85 99000000	mov	al,byte ptr [ebp+99]
004650C0	E9 9C000000	jmp	UnPackMe.00465161
004650C5	8B4424 04	mov	eax,dword ptr [esp+4]
004650C9	8B4C24 0C	mov	ecx,dword ptr [esp+C]
004650CD	FF81 B8000000	inc	dword ptr [ecx+B8]
004650D3	8B00	mov	eax,dword ptr [eax]
004650D5	3D 940000C0	cmp	eax,C0000094
004650DA	75 24	jnz	short UnPackMe.00465100
004650DC	FF81 B8000000	inc	dword ptr [ecx+B8]
004650E2	33C0	xor	eax, eax

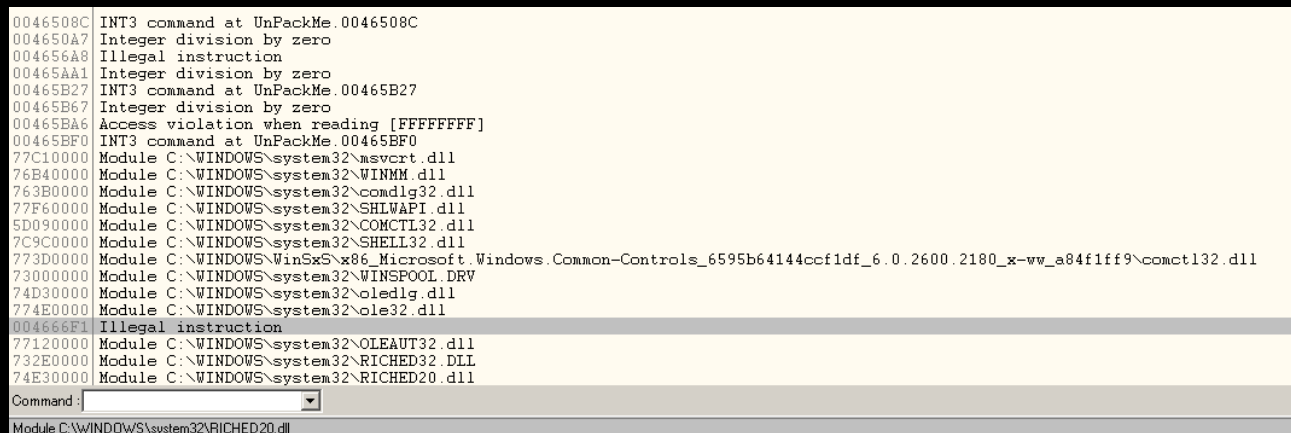
Quan sát code bên dưới, không thấy có lệnh `jmp` hay cặp lệnh `push & retn...` để chuyển hướng thực thi sang vùng code khác mà vẫn thuộc 0x465000. Như vậy, kết luận phương pháp `PUSHAD` không giúp ta đạt được mục đích và dường như dưới sự bảo vệ của `tElock`, `unpackme` có khả năng chống lại việc đặt Hardware Breakpoints, vì nếu bạn tiếp tục nhấn **F9** thì sẽ nhận được thông báo lỗi tương tự như sau:



Do vậy, thực hiện bỏ hardware breakpoint vừa đặt:



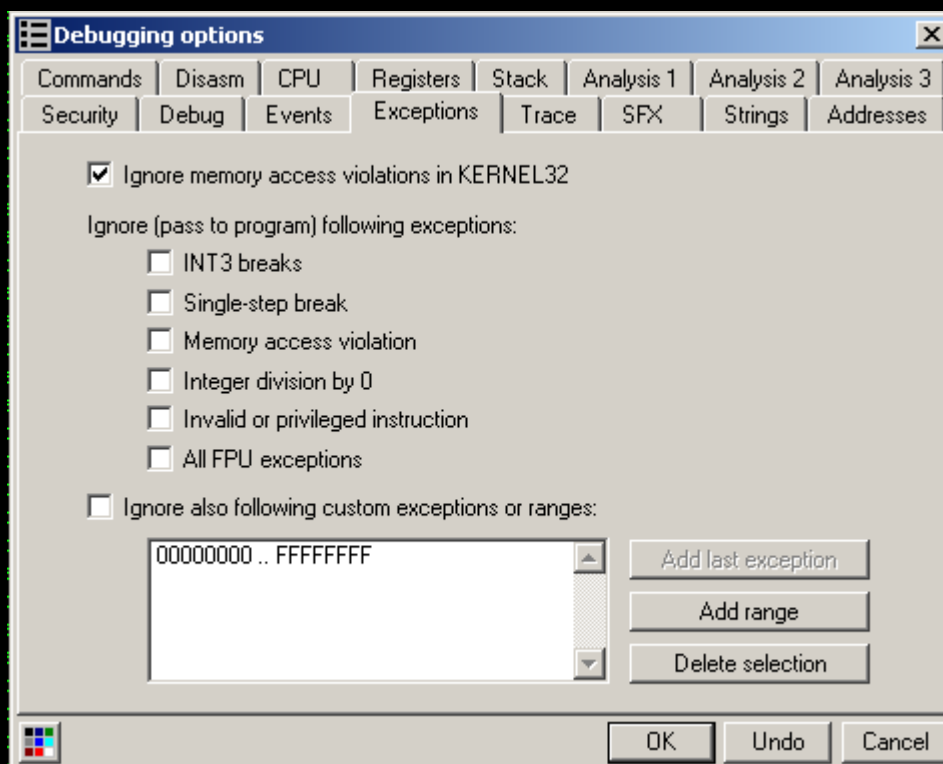
Sau khi xóa xong, khởi động lại OllyDbg. Như đã thấy, do phương pháp **PUSHAD** không thể áp dụng đối với target này nên ta sẽ thử áp dụng phương pháp Exceptions xem thế nào. Nhấn **F9** để run target và kiểm tra danh sách các exceptions đã xảy ra tại cửa sổ Log của OllyDbg, kết quả có được tương tự như hình dưới đây:



Ta chú ý tới exception cuối cùng đã xảy ra trước khi unpackme thực thi bình thường, đó là tại địa chỉ **004666F1 Message=Illegal instruction**. Địa chỉ này không thuộc vùng section đầu tiên mà thuộc section **.teddy**:

003A0000	00001000				Priv	RW	RW
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				Man	R E	R E

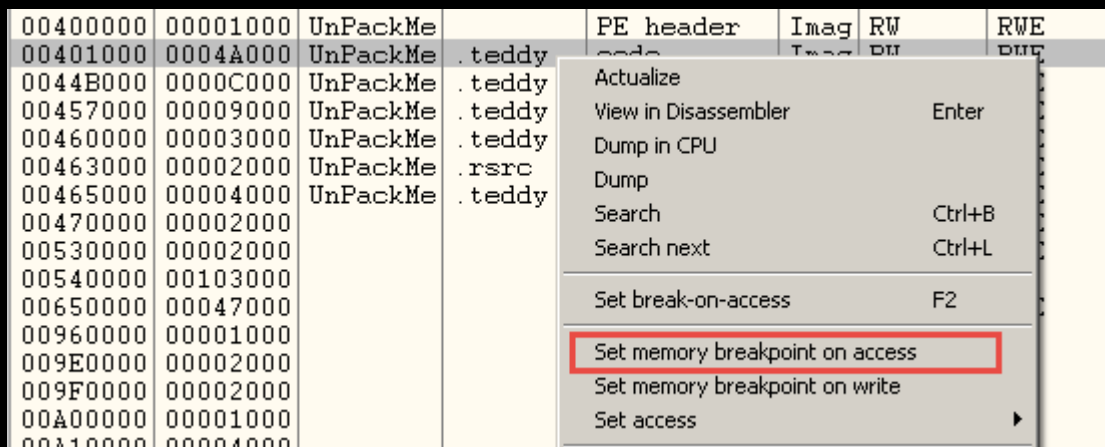
OK tạm thời có thông tin như vậy, khởi động lại OllyDBG và thiết lập lại các tùy chọn tại tab Exceptions tương tự như dưới đây:



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. Cho tới khi ta tới được exception cuối cùng mà ta biết được thông qua cửa sổ Log, trong trường hợp này là 0x4666F1:

004666F1	8DC0	lea	eax, eax	Illegal use of register
004666F3	EB 01	jmp	short UnPackMe.004666F6	
004666F5	EB 68	jmp	short UnPackMe.0046675F	
004666F7	33C0	xor	eax, eax	
004666F9	EB FE	jmp	short UnPackMe.004666F9	
004666FB	FFE4	jmp	near esp	
004666FD	CD 20	int	20	
004666FF	8B6424 08	mov	esp, dword ptr [esp+8]	

OK, như vậy ta đang dừng lại tại exception cuối cùng xảy ra trước khi file thực thi hoàn toàn. Tại đây, ta có thể có tiến hành đặt một **BPM On Access** tại section code của unpackme. Chuyển qua cửa sổ Memory, thực hiện đặt BPM tại section code như sau:



Sau khi đặt BP xong, nhấn **Shift + F9**, ta dừng lại tại đây:

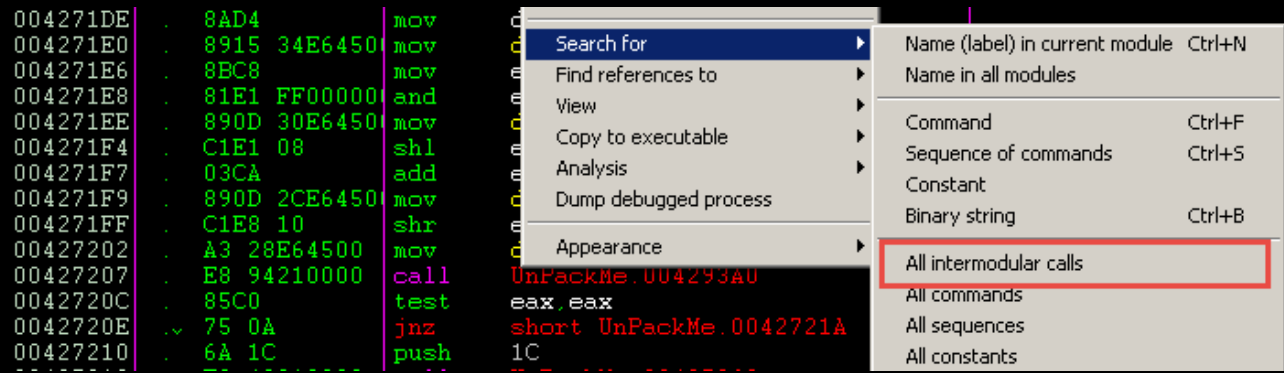
004667BA	F7F2	div	edx	
004667BC	EB E8	jmp	short UnPackMe.004667A6	
004667BE	EB 02	jmp	short UnPackMe.004667C2	
004667C0	FF20	jmp	near dword ptr [eax]	
004667C2	F5	cmc		
004667C3	33C3	xor	eax,ebx	
004667C5	2BC9	sub	ecx,ecx	
004667C7	64:8F01	pop	dword ptr fs:[ecx]	
004667C9	F8	---	---	

Tiếp tục nhấn **Shift + F9** thêm 2 lần nữa, ta sẽ tới được OEP của unpackme:

004271B0	55	push	ebp	
004271B1	8BEC	mov	ebp,esp	
004271B3	6A FF	push	-1	
004271B5	68 600E4500	push	UnPackMe.00450E60	
004271B7	68 C8924200	push	UnPackMe.004292C8	SE handler installation
004271B9	64:A1 000000	mov	eax,dword ptr fs:[0]	
004271BB	50	push	eax	
004271BD	64:8925 0000	mov	dword ptr fs:[0],esp	
004271BF	83C4 A8	add	esp,-58	
004271C1	53	push	ebx	
004271C3	56	push	esi	
004271C5	57	push	edi	
004271C7	8965 E8	mov	dword ptr [ebp-18],esp	
004271C9	FF15 DC0A460	call	near dword ptr [460ADC]	
004271CB	33D2	xor	edx,edx	
004271CD	8AD4	mov	dl,ah	
004271CF	8915 34E6450	mov	dword ptr [45E634],edx	

Như vậy, OEP tìm được là **0x4271B0**, giống OEP tìm được khi thực hiện unpack file unpackme được pack bằng Crunch ở trên. Tuy nhiên, quan sát kĩ ta thấy ở đây có điểm khác so với ví dụ trước, nếu ở unpackme trước ta biết được hàm API đầu tiên được gọi sau OEP là **GetVersion**, thì với unpackme này, tại code ta đang đứng ở OEP ta không thấy thông tin này được thể hiện.

Thử tìm các **Intermodular calls** xem có thêm thông tin nào khác không:



OllyDbg sẽ thực hiện tìm kiếm toàn bộ tất cả các intermodular calls, kết quả có được tương tự như hình dưới đây:

Address	Disassembly	Destination
00423E96	call near dword ptr [460B5C]	ds:[00460B5C]=009E0973
004249B6	call near dword ptr [4609F4]	ds:[004609F4]=009E027C
00425003	call near dword ptr [460B98]	ds:[00460B98]=009E0AA0
0042501D	call near edi	
004251B3	call near dword ptr [460B98]	ds:[00460B98]=009E0AA0
004251C7	call near dword ptr [460B94]	ds:[00460B94]=009E0A7D
0042520B	call near dword ptr [460B94]	ds:[00460B94]=009E0A7D
004252D4	call near dword ptr [460B94]	ds:[00460B94]=009E0A7D
004252FB	call near dword ptr [460978]	ds:[00460978]=009E0011
00425306	call near dword ptr [460974]	ds:[00460974]=009E0000
0042535E	call near dword ptr [4609B0]	ds:[004609B0]=009E0124
004259D1	call near dword ptr [460A54]	ds:[00460A54]=009E0447
004259E8	call near dword ptr [460A3C]	ds:[00460A3C]=009E03DC
004259F5	call near dword ptr [460A44]	ds:[00460A44]=009E03FB
00425B2B	call near dword ptr [460AF8]	ds:[00460AF8]=009E0785
00425B3F	call near dword ptr [460AF8]	ds:[00460AF8]=009E0785
00425B53	call near dword ptr [460AF8]	ds:[00460AF8]=009E0785
00425BF1	call near dword ptr [460AF8]	ds:[00460AF8]=009E0785
00425C9C	call near dword ptr [46097C]	ds:[0046097C]=009E0022
00425CA6	call near dword ptr [460B5C]	ds:[00460B5C]=009E0973
00425D71	call near dword ptr [460A54]	ds:[00460A54]=009E0447
00425D8B	call near dword ptr [460A3C]	ds:[00460A3C]=009E03DC
00425DBB	call near dword ptr [460A44]	ds:[00460A44]=009E03FB
00425E13	call near dword ptr [460B98]	ds:[00460B98]=009E0AA0
00425E27	call near dword ptr [460B94]	ds:[00460B94]=009E0A7D
00425E6B	call near dword ptr [460B94]	ds:[00460B94]=009E0A7D
00425F34	call near dword ptr [460B94]	ds:[00460B94]=009E0A7D
004271B0	push ebp	(Initial CPU selection)
004271D6	call near dword ptr [460ADC]	ds:[00460ADC]=009E06F7
0042723E	call near dword ptr [460984]	ds:[00460984]=009E0041
004272D5	call near dword ptr [460980]	ds:[00460980]=009E0033
004272F6	call near dword ptr [460B9C]	ds:[00460B9C]=009E0AB1
004274F7	call UnPackMe.00435CC0	
004277F3	call UnPackMe.00435CC0	
00427929	call near dword ptr [46098C]	ds:[0046098C]=009E005F
00427E07	call near dword ptr [460A84]	ds:[00460A84]=009E0539
00427E0E	call near dword ptr [460994]	ds:[00460994]=009E008D
00427E98	call near dword ptr [460990]	ds:[00460990]=009E007F
00428029	call near dword ptr [4609F8]	ds:[004609F8]=009E028D
004280BB	call near dword ptr [460A08]	ds:[00460A08]=009E02CC
004287BD	call near esi	
00428823	call near dword ptr [460A54]	ds:[00460A54]=009E0447
0042884D	call near dword ptr [460A3C]	ds:[00460A3C]=009E03DC
0042886E	call near dword ptr [460A44]	ds:[00460A44]=009E03FB
004288AE	call near dword ptr [460A3C]	ds:[00460A3C]=009E03DC
004288E0	call near dword ptr [460A3C]	ds:[00460A3C]=009E03DC
0042891E	call near dword ptr [460A44]	ds:[00460A44]=009E03FB

Với kết quả có được như trên, ta thử kiểm tra xem trong số các lệnh call này, có lời gọi nào tới hàm API hay không? Ta thấy có rất nhiều indirect calls thay vì là lệnh call tới địa chỉ một hàm API của một DLL như ta vẫn thường thấy, trong trường hợp tại máy của tôi (lưu ý: có thể khác ở máy của bạn) các vùng nhớ của các lệnh call đều đang trỏ tới các giá trị là 0x9Exxxx (lưu ý: còn có các giá trị khác mà tôi không đề cập hết ở đây).

Tiếp tục cuộn chuột xuống một chút, ta thấy trong kết quả thu được vẫn có xen lẫn một số lệnh call tới API:

Address	Disassembly	Destination
00435558	call near dword ptr [460B5C]	ds:[00460B5C]=009E0973
004356E8	call near ebx	
0043570C	call near dword ptr [4609EC]	ds:[004609EC]=009E0243
004357EC	call near dword ptr [460A08]	ds:[00460A08]=009E02CC
004358BB	call near edi	
00435953	call near dword ptr [4609EC]	ds:[004609EC]=009E0243
00435B5F	call near dword ptr [4609F0]	ds:[004609F0]=009E0262
00435D96	call near dword ptr [460B74]	ds:[00460B74]=009E09F0
00435DF1	call near dword ptr [460AB0]	ds:[00460AB0]=009E061C
00435E06	call near dword ptr [460AB4]	ds:[00460AB4]=009E0636
00435FA5	call UnPackMe.00435CDE	comdlg32.PrintDlgA
00436004	call near dword ptr [460AD0]	ds:[00460AD0]=009E06B5
0043602E	call near dword ptr [460AD0]	ds:[00460AD0]=009E06B5
0043606C	call near dword ptr [460AD0]	ds:[00460AD0]=009E06B5
004360AB	call near dword ptr [460AD0]	ds:[00460AD0]=009E06B5
004360D1	call UnPackMe.00435CDE	comdlg32.PrintDlgA
004360EB	call near edi	
0043611B	call near dword ptr [460878]	ds:[00460878]=00A00124
0043612D	call near esi	
0043623D	call near dword ptr [460B74]	ds:[00460B74]=009E09F0
004362EC	call near dword ptr [460CAC]	ds:[00460CAC]=009F0360
00436315	call near dword ptr [460CD4]	ds:[00460CD4]=009F0419
00436351	call UnPackMe.00435CEA	comdlg32.GetOpenFileNameA
00436358	call UnPackMe.00435CE4	comdlg32.GetSaveFileNameA
00436374	call near dword ptr [460DF4]	ds:[00460DF4]=009F09B7
00436382	call near dword ptr [460CB0]	ds:[00460CB0]=009F0370
004363E1	call near ebx	
00436407	call near dword ptr [460E78]	ds:[00460E78]=009F0C52
0043644E	call near dword ptr [460E78]	ds:[00460E78]=009F0C52
004366CA	call near dword ptr [460DFC]	ds:[00460DFC]=009F09F0
0043678C	call near edi	
004368C5	call near dword ptr [460D48]	ds:[00460D48]=009F0669
004368DB	call near dword ptr [460D48]	ds:[00460D48]=009F0669
004368F1	call near dword ptr [460D48]	ds:[00460D48]=009F0669
00436907	call near dword ptr [460D48]	ds:[00460D48]=009F0669
0043691D	call near dword ptr [460D48]	ds:[00460D48]=009F0669
00436933	call near dword ptr [460D48]	ds:[00460D48]=009F0669
00436A6A	call near dword ptr [460E78]	ds:[00460E78]=009F0C52
00436BAB	call near edi	
00436BD0	call UnPackMe.00435CFC	comdlg32.FindTextA
00436BD7	call UnPackMe.00435CF6	comdlg32.ReplaceTextA
00436CB1	call near dword ptr [460AA8]	ds:[00460AA8]=009E05EC
00436CC3	call near dword ptr [460AAC]	ds:[00460AAC]=009E05FD
0043799E	call near dword ptr [460D48]	ds:[00460D48]=009F0669
00437B2F	call near dword ptr [460D54]	ds:[00460D54]=009F069F

Những vùng khoanh đỏ là một số lệnh call trực tiếp, để chắc chắn xem đây có phải là một Indirect Jump hay không, ta thử kiểm tra một vài lệnh bất kỳ. Ví dụ tới lệnh call tại địa chỉ 0x435FA5:

```

00435FA5  E8 34FDFFFF  call  UnPackMe.00435CDE  LPrintDlgA
00435FAA  . 8BCE      mov  ecx,esi
00435FAC  . 8BF8      mov  edi,eax
00435FAE  . E8 6E4E0000 call  UnPackMe.0043AE21
00435FB3  . 85FF      test edi,edi
00435FB5  . 74 04     je   short UnPackMe.00435FBB
00435FB7  . 8BC7      mov  eax,edi
00435FB9  . 74 03     jmp  short UnPackMe.00435FBE
00435FBB  > 6A 02     push 2
00435FBD  . 58        pop  eax
00435FBE  > 5F        pop  edi
00435FBF  . 5E        pop  esi
00435FC0  . C3        retn

```

Nhấn **Enter** để Follow theo lệnh Call này:

```

00435CBF  90        nop
00435CC0  $- FF25 8809460 jmp  near dword ptr [460988]
00435CC6  $- FF25 000C460 jmp  near dword ptr [460C00]
00435CCC  $- FF25 040C460 jmp  near dword ptr [460C04]
00435CD2  $- FF25 D00E460 jmp  near dword ptr [460ED0]  comdlg32.ChooseFontA
00435CD8  $- FF25 D80E460 jmp  near dword ptr [460ED8]  comdlg32.ChooseColorA
00435CDE  $- FF25 D40E460 jmp  near dword ptr [460ED4]  comdlg32.PrintDlgA
00435CE4  $- FF25 CC0E460 jmp  near dword ptr [460ECC]  comdlg32.GetSaveFileNameA
00435CEA  $- FF25 C80E460 jmp  near dword ptr [460EC8]  comdlg32.GetOpenFileNameA
00435CF0  $- FF25 C40E460 jmp  near dword ptr [460EC4]  comdlg32.GetFileTitleA
00435CF6  $- FF25 C00E460 jmp  near dword ptr [460EC0]  comdlg32.ReplaceTextA
00435CFC  $- FF25 BC0E460 jmp  near dword ptr [460EBC]  comdlg32.FindTextA
00435D02  $- FF25 B80E460 jmp  near dword ptr [460EB8]  comdlg32.CommDlgExtendedError
00435D08  $- FF25 B00E460 jmp  near dword ptr [460EB0]  WINSPOOL.ClosePrinter
00435D0E  $- FF25 940E460 jmp  near dword ptr [460E94]  WINSPOOL.EndDocPrinter
00435D14  $- FF25 AC0E460 jmp  near dword ptr [460EAC]  WINSPOOL.StartPagePrinter
00435D1A  $- FF25 A80E460 jmp  near dword ptr [460EA8]  WINSPOOL.StartDocPrinterA
00435D20  $- FF25 A40E460 jmp  near dword ptr [460EA4]  WINSPOOL.OpenPrinterA
00435D26  $- FF25 A00E460 jmp  near dword ptr [460EA0]  WINSPOOL.EndPagePrinter
00435D2C  $- FF25 9C0E460 jmp  near dword ptr [460E9C]  WINSPOOL.WritePrinter
00435D32  $- FF25 980E460 jmp  near dword ptr [460E98]  WINSPOOL.DocumentPropertiesA
00435D38  $- FF25 240F460 jmp  near dword ptr [460F24]  oledlg.OleUIBusyA
00435D3E  CC        int3
00435D3F  CC        int3

```

Ta thấy có một số lệnh nhảy Indirect Jmp tới các hàm API, như vậy đây là các giá trị thuộc bảng IAT, đồng nghĩa với việc 0x460ED4 là một mục trong IAT. Follow in Dump tại địa chỉ này và xem dưới dạng **Long > Address**:

00460ECC	763C7CD8	comdlg32.GetSaveFileNameA
00460ED0	763CC289	comdlg32.ChooseFontA
00460ED4	763D46CD	comdlg32.PrintDlgA
00460ED8	763BEECE	comdlg32.ChooseColorA
00460EDC	00000000	
00460EE0	774F2068	ole32.CoTaskMemAlloc
00460EE4	775548A4	ole32.CLSIDFromString
00460EE8	775429DD	ole32.CLSIDFromProgID
00460EEC	774F204C	ole32.CoTaskMemFree
00460EF0	7752949B	ole32.OleInitialize
00460EF4	77529539	ole32.OleUninitialize
00460EF8	774F974A	ole32.CreateStreamOnHGlobal
00460EFC	7751EA61	ole32.CreateILockBytesOnHGlobal
00460F00	7751EB91	ole32.StgCreateDocfileOnILockBytes
00460F04	7757A379	ole32.OleIsCurrentClipboard
00460F08	7757A529	ole32.OleFlushClipboard
00460F0C	7752431A	ole32.CoRevokeClassObject
00460F10	77552DA0	ole32.CoRegisterMessageFilter
00460F14	7752D1E0	ole32.CoFreeUnusedLibraries
00460F18	7753F356	ole32.CoGetClassObject
00460F1C	775DB375	ole32.StgOpenStorageOnILockBytes
00460F20	00000000	
00460F24	74D3F0F3	oledlg.OleUIBusyA
00460F28	00000000	
00460F2C	00000000	
00460F30	00000000	
00460F34	00000000	

IAT End

Theo kết quả thu được, chúng ta có được thông tin của IAT End là **0x460F28**, hoàn toàn giống với kết quả đã làm ở ví dụ trước. Có thể khẳng định được chính xác đây là IAT End là bởi sau đó chỉ là các địa chỉ có giá trị **0x00**. Giờ ta thử cuộn chuột lên trên xem có tìm được thông tin về IAT Start không:

00460800	00002B00	
00460804	00062B4C	
00460808	00062B2E	
0046080C	00000000	
00460810	80000000	
00460814	00000000	
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	00A00000	
00460840	00A00011	
00460844	00A00022	
00460848	00A00033	
0046084C	00A00041	
00460850	00A00050	
00460854	00A0005F	
00460858	00A0007F	

Ồ... như vậy vẫn tìm được IAT Start như đã tìm ở ví dụ trên. Tuy nhiên, quan sát kỹ ta sẽ thấy vùng giữa IAT Start và IAT End lại không có đầy đủ thông tin về APIs như ở target trước.

Address	Value	Comment
00460E1C	009F0A7D	
00460E20	009F0AA0	
00460E24	009F0AB1	
00460E28	009F0AC2	
00460E2C	009F0AD4	
00460E30	009F0AE4	
00460E34	009F0AF5	
00460E38	009F0B14	
00460E3C	009F0B2E	
00460E40	009F0B3F	
00460E44	009F0B50	
00460E48	009F0B61	
00460E4C	009F0B6F	
00460E50	009F0B7E	
00460E54	009F0B8D	
00460E58	009F0BAD	
00460E5C	009F0BBB	
00460E60	009F0BDE	
00460E64	009F0BEF	
00460E68	009F0C00	
00460E6C	009F0C12	
00460E70	009F0C22	
00460E74	009F0C33	
00460E78	009F0C52	
00460E7C	009F0C6C	
00460E80	009F0C7D	
00460E84	009F0C8E	
00460E88	009F0C9F	
00460E8C	76B5A8F7	WINMM.PlaySoundA
00460E90	00000000	
00460E94	730074C8	WINSPOOL.EndDocPrinter
00460E98	73016673	WINSPOOL.DocumentPropertiesA
00460E9C	73007287	WINSPOOL.WritePrinter
00460EA0	73008043	WINSPOOL.EndPagePrinter
00460EA4	73013767	WINSPOOL.OpenPrinterA
00460EA8	730141FB	WINSPOOL.StartDocPrinterA
00460EAC	73008367	WINSPOOL.StartPagePrinter
00460EB0	73005390	WINSPOOL.ClosePrinter
00460EB4	00000000	
00460EB8	763C00CE	comdlg32.CommDlgExtendedError
00460EBC	763C867C	comdlg32.FindTextA
00460EC0	763C86B0	comdlg32.ReplaceTextA
00460EC4	763B2533	comdlg32.GetFileTitleA
00460EC8	763B311E	comdlg32.GetOpenFileNameA
00460ECC	763C7CD8	comdlg32.GetSaveFileNameA
00460ED0	763CC289	comdlg32.ChooseFontA
00460ED4	763D46CD	comdlg32.PrintDlgA
00460ED8	763BEECE	comdlg32.ChooseColorA

Ta nhận thấy, có sự thay đổi bắt đầu từ địa chỉ 0x460E88, từ địa chỉ này trở lên ta thấy đều chứa các giá trị 0x9Fxxxx, 0x9Exxxx, Hàm API phân tách sự khác biệt là

tại `00460E8C 76B5A8F7 WINMM.PlaySoundA`. Lựa chọn một địa chỉ bất kỳ ở trên, nhấn chuột phải và chọn **Find References**, ta có được kết quả:

Address	Disassembly	Comment
004038A6	call near dword ptr [460E48]	ds:[00460E48]=009F0B61
004047D0	call near dword ptr [460E48]	ds:[00460E48]=009F0B61
00404923	call near dword ptr [460E48]	ds:[00460E48]=009F0B61
00404AD9	call near dword ptr [460E48]	ds:[00460E48]=009F0B61
0041331B	call near dword ptr [460E48]	ds:[00460E48]=009F0B61
004171DE	call near dword ptr [460E48]	ds:[00460E48]=009F0B61
0041A61E	call near dword ptr [460E48]	ds:[00460E48]=009F0B61
00421045	call near dword ptr [460E48]	ds:[00460E48]=009F0B61
0043D912	call near dword ptr [460E48]	ds:[00460E48]=009F0B61
0043DCC0	call near dword ptr [460E48]	ds:[00460E48]=009F0B61
0043E9D6	call near dword ptr [460E48]	ds:[00460E48]=009F0B61
0043EA8A	call near dword ptr [460E48]	ds:[00460E48]=009F0B61
00441B0A	call near dword ptr [460E48]	ds:[00460E48]=009F0B61

Điều ta mong muốn là lệnh `Call` hay `JMP` tới địa chỉ một hàm API thì ở đây lại chuyển hướng tới một vùng nhớ khác, trong trường hợp của tôi là `0x9Fxxxx` (có thể khác nhau tùy máy). Đây chính là lý do tại sao lại được gọi là IAT Redirect, là vì khi quá trình unpack xảy ra, packer sẽ thực hiện ghi đè một số giá trị thuộc IAT bằng các giá trị khác nhằm trỏ tới vùng code riêng, trong trường hợp này ví dụ giống như ảnh trên.

Address=004038A6

Disassembly=call near dword ptr [460E48]

Comment=ds:[00460E48]=009F0B61

Thay vì phải lưu địa chỉ chính xác của các hàm API, packer sẽ thực hiện thay thế địa chỉ này bằng một địa chỉ của một section riêng biệt được tạo ra bởi packer lúc thực thi (runtime), khi chuyển tới section đó sẽ có quá trình tính toán/ chuyển đổi để làm sao tới được API một cách chính xác.

Để làm rõ thêm, tôi sẽ lấy ví dụ với lệnh `Call` gọi tới hàm API `GetVersion` bên dưới OEP:


```

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
004271DE  8AD4    mov     dl, ah
004271E0  8915 34E6450 mov     dword ptr [45E634], edx
004271E6  8BC8    mov     ecx, eax
004271E8  81E1 FF000000 and     ecx, 0FF

```

SE handler installation

Ta biết được đây là lệnh call tới hàm **GetVersion** là do lúc thực hiện unpack với unpackme được pack bằng Crunch ở trên. Nhấn Enter để Follow theo lệnh call này, nếu chuẩn chỉ thì ta sẽ tới hàm API, nhưng tuy nhiên địa chỉ hàm API giờ đây đã bị thay thế bằng một địa chỉ khác. Do vậy, ta sẽ tới đây:

```

009E06F7  85E4    test    esp, esp
009E06F9  79 03   jns     short 009E06FE
009E06FB  0F9142 40    setno   byte ptr [edx+40]
009E06FF  B8 D3179E00 mov     eax, 9E17D3
009E0704  40      inc     eax
009E0705  FF30    push    dword ptr [eax]
009E0707  C3      retn

```

Ta thấy lệnh call đưa ta tới địa chỉ 0x9E06F7 (có thể khác ở máy bạn), địa chỉ này nằm ở vùng nhớ khác không thuộc bất kỳ một sections nào của bản thân unpackme:

003A0000	00001000				Priv	RW	RW
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	0003B000				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

Các sections của unpackme được map vào bộ nhớ được tôi highlight ở trên, bên dưới là section không có tên, được tôi khoanh đỏ, đó là nơi mà code của unpackme nhảy tới.

Section này được tạo ra lúc runtime, do đó nếu ta khởi động lại Olly và quan sát cửa sổ Memory Map, ta sẽ thấy section này không tồn tại:

003A0000	00001000				Priv	RW	RW
00400000	00001000	UnPackMe		PE header	Imag	R	RWE
00401000	0004A000	UnPackMe	.teddy	code	Imag	R	RWE
0044B000	0000C000	UnPackMe	.teddy	data	Imag	R	RWE
00457000	00009000	UnPackMe	.teddy		Imag	R	RWE
00460000	00003000	UnPackMe	.teddy		Imag	R	RWE
00463000	00002000	UnPackMe	.rsrc	resources	Imag	R	RWE
00465000	00004000	UnPackMe	.teddy	SFX, imports	Imag	R	RWE
00470000	00002000				Map	R E	R E
00530000	00002000				Map	R E	R E
00540000	00103000				Map	R	R
00650000	0003B000				Map	R E	R E
629C0000	00001000	LPK		PE header	Imag	R	RWE
629C1000	00005000	LPK	.text	code, imports	Imag	R	RWE

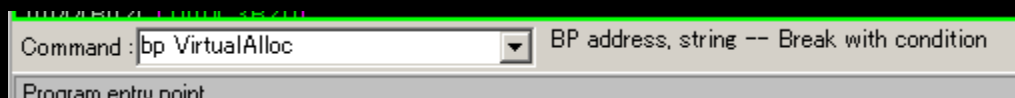
Tại đoạn code ở 0x9E06F7, như lập luận ở trên tại đây sẽ có quá trình tính toán/thay đổi để tới được hàm API. Đọc code chạy ta thấy giá trị của thanh ghi `eax` sau tính toán sẽ là 0x9E17D4. Lệnh `Push [eax] & Retn` sẽ nhảy tới địa chỉ được lưu tại 0x9E17D4. Tại cửa sổ Dump, chuyển tới địa chỉ này để xem giá trị được lưu là gì:

Address	Value	Comment
009E17D4	7C8114AB	kernel32.GetVersion
009E17D8	7C8097F4	kernel32.MulDiv
009E17DC	7C85B001	kernel32.RemoveDirectoryA
009E17E0	7C826219	kernel32.CreateDirectoryA
009E17E4	7C81E85C	kernel32.DeleteFileA

Như thấy trên hình, đó chính là địa chỉ của hàm API `GetVersion`.

Quay trở lại nội dung đang đề cập ở trước, ta thấy rằng section này được tạo ra bởi chương trình trong quá trình thực hiện unpacking. Bây giờ, ta cần phải xác định xem section này được tạo ra khi nào? Để làm được điều này ta đặt một BP tại API là `VirtualAlloc`, lý do lựa chọn API này là do nó chịu trách nhiệm trong việc tạo ra các virtual sections và truy cập chúng. Thông tin về hàm `VirtualAlloc` có thể xem thêm tại đây ([https://msdn.microsoft.com/en-us/library/windows/desktop/aa366887\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/aa366887(v=vs.85).aspx)).

Khởi động lại OllyDbg, tiến hành đặt bp như hình dưới đây:



Sau khi đặt BP xong, cấu hình lại phần Exceptions (check lại toàn bộ các tùy chọn), sau đó nhấn **F9** để run, nhưng bị Terminate luôn. Vậy khả năng có thêm cả cơ chế anti-BP nữa rồi, đổi lại cách đặt BP bằng cách đặt BP tại lệnh `RETN` của API `VirtualAlloc`:

Address	Disassembly	Comment
7C809A81	8BFF	mov edi,edi
7C809A83	55	push ebp
7C809A84	8BEC	mov ebp,esp
7C809A86	FF75 14	push dword ptr [ebp+14]
7C809A89	FF75 10	push dword ptr [ebp+10]
7C809A8C	FF75 0C	push dword ptr [ebp+C]
7C809A8F	FF75 08	push dword ptr [ebp+8]
7C809A92	6A FF	push -1
7C809A94	E8 09000000	call kernel32.VirtualAllocEx
7C809A99	5D	pop ebp
7C809A9A	C2 1000	ret 10
7C809A9D	90	nop
7C809A9E	90	nop

Sau khi đặt lại BP như trên, nhấn **F9** để run, OllyDbg break tại lệnh **Retn**:

Address	Disassembly	Comment
7C809A81	8BFF	mov edi,edi
7C809A83	55	push ebp
7C809A84	8BEC	mov ebp,esp
7C809A86	FF75 14	push dword ptr [ebp+14]
7C809A89	FF75 10	push dword ptr [ebp+10]
7C809A8C	FF75 0C	push dword ptr [ebp+C]
7C809A8F	FF75 08	push dword ptr [ebp+8]
7C809A92	6A FF	push -1
7C809A94	E8 09000000	call kernel32.VirtualAllocEx
7C809A99	5D	pop ebp
7C809A9A	C2 1000	ret 10
7C809A9D	90	nop
7C809A9E	90	nop
7C809A9F	90	nop
7C809AA0	90	nop
7C809AA1	90	nop
7C809AA2	6A 10	push 10
7C809AA3	55	push ebp

Register	Value
EAX	003B0000
ECX	7C809AE9 kernel32.7C809AE9
EDX	7C90EB94 ntdll.KiFastSystemCallRet
EBX	00028610
ESP	0012FF64
EBP	0005995E
ESI	00000004
EDI	0005995E
EIP	7C809A9A kernel32.7C809A9A
C 0	ES 0023 32bit 0(FFFFFFFF)
P 1	CS 001B 32bit 0(FFFFFFFF)
A 0	SS 0023 32bit 0(FFFFFFFF)
Z 1	DS 0023 32bit 0(FFFFFFFF)
S 0	FS 003B 32bit 7FDE0000(FFF)
T 0	GS 0000 NULL

Quan sát giá trị trả về tại thanh ghi EAX, ta thấy địa chỉ của section mới được tạo ra (trong trường hợp máy của tôi) là 0x3B0000. Tiếp tục nhấn **F9** và quan sát giá trị của thanh ghi EAX:

Register	Value
EAX	009E0000
ECX	7C809AE9 kernel32.7C809AE9
EDX	7C90EB94 ntdll.KiFastSystemCallRet
EBX	0000008C
ESP	0012FF70
EBP	0005995E
ESI	00460974 UnPackMe.00460974
EDI	00460BA8 UnPackMe.00460BA8
EIP	7C809A9A kernel32.7C809A9A

Nếu các bạn để ý kĩ khi thực hiện việc tìm kiếm toàn bộ Intermodular calls, thì ngoài các lệnh call trở tới vùng địa chỉ 0x9Exxxx, thì còn có các lệnh call trở tới vùng địa chỉ 0x9Fxxxx và 0xA0xxxx. Tiếp tục nhấn F9 ta sẽ có thông tin như sau:

Registers (MMX)	<	Registers (MMX)
EAX 009F0000		EAX 00A00000
ECX 7C809AE9 kernel32.7C809AE9		ECX 7C809AE9 kernel32.7C809AE9
EDX 7C90EB94 ntdll.KiFastSystemCallRet		EDX 7C90EB94 ntdll.KiFastSystemCallRet
EBX 000000A3		EBX 0000004D
ESP 0012FF70		ESP 0012FF70
EBP 0005995E		EBP 0005995E
ESI 00460BFC UnPackMe.00460BFC		ESI 0046083C UnPackMe.0046083C
EDI 00460E8C UnPackMe.00460E8C		EDI 00460974 UnPackMe.00460974
EIP 7C809A9A kernel32.7C809A9A		EIP 7C809A9A kernel32.7C809A9A

Chuyển tới cửa sổ Memory map, quan sát ta sẽ thấy được vùng nhớ mới được tạo ra:

00400000	00001000	UnPackMe		PE header	Priv 01001002	R	RWE
00401000	0004A000	UnPackMe	.teddy	code	Priv 01001002	R	RWE
0044B000	0000C000	UnPackMe	.teddy	data	Priv 01001002	R	RWE
00457000	00009000	UnPackMe	.teddy		Priv 01001002	R	RWE
00460000	00003000	UnPackMe	.teddy		Priv 01001002	R	RWE
00463000	00002000	UnPackMe	.rsrc	resources	Priv 01001002	R	RWE
00465000	00004000	UnPackMe	.teddy	SFX, imports	Priv 01001002	R	RWE
00470000	00002000				Map 00041020	R E	R E
00530000	00002000				Map 00041020	R E	R E
00540000	00103000				Map 00041002	R	R
00650000	00042000				Map 00041020	R E	R E
00960000	00001000				Priv 00021004	RW	RW
009E0000	00002000				Priv 00021004	RW	RW
009F0000	00002000				Priv 00021004	RW	RW
00A00000	00001000				Priv 00021004	RW	RW

Như trong hình, các vùng nhớ này được đánh dấu là Priv hay Private, hàm ý nó được tạo ra và sử dụng khi chương trình thực hiện quá trình unpack. Lúc này, quay trở lại màn hình CPU, bỏ BP đã đặt và tới OEP giống như cách đã thực hiện ở trên. Sau khi tới OEP, quay trở lại màn hình Memory Map để kiểm tra, ta thấy có thêm một số vùng nhớ nữa được tạo ra. Tuy nhiên, ta chỉ quan tâm những vùng nhớ đã đề cập ở trên do unpack có sử dụng tới chúng:

00400000	00001000	UnPackMe		PE header	Imag	01001004	RW	RWE
00401000	0004A000	UnPackMe	.teddy	code	Imag	01001004	RW	RWE
0044B000	0000C000	UnPackMe	.teddy	data	Imag	01001004	RW	RWE
00457000	00009000	UnPackMe	.teddy		Imag	01001004	RW	RWE
00460000	00003000	UnPackMe	.teddy		Imag	01001004	RW	RWE
00463000	00002000	UnPackMe	.rsrc	resources	Imag	01001004	RW	RWE
00465000	00004000	UnPackMe	.teddy	SFX,imports	Imag	01001004	RW	RWE
00470000	00002000				Map	00041020	R E	R E
00530000	00002000				Map	00041020	R E	R E
00540000	00103000				Map	00041002	R	R
00650000	00042000				Map	00041020	R E	R E
00960000	00001000				Priv	00021004	RW	RW
009E0000	00002000				Priv	00021004	RW	RW
009F0000	00002000				Priv	00021004	RW	RW
00A00000	00001000				Priv	00021004	RW	RW
00A10000	00004000				Priv	00021004	RW	RW
00A20000	00003000				Priv	00021004	RW	RW
00A30000	00002000				Map	00041002	R	R
00A40000	00001000				Priv	00021004	RW	RW
01220000	00002000				Map	00041002	R	R
5D090000	00001000	COMCTL32		PE header	Imag	01001002	R	RWE
5D091000	00070000	COMCTL32	.text	code,imports	Imag	01001002	R	RWE
5D101000	00003000	COMCTL32	.data	data	Imag	01001002	R	RWE
5D104000	0001F000	COMCTL32	.rsrc	resources	Imag	01001002	R	RWE
5D123000	00004000	COMCTL32	.reloc	relocations	Imag	01001002	R	RWE

OK, ta đang dừng tại OEP. Các vùng nhớ cần thiết phục vụ cho việc Redirect IAT cũng đã được unpackme tạo xong. Từ OEP, tiến hành trace code với **F7** để trace vào lệnh call đầu tiên, ta tới đây:

009E06F7	85E4	test	esp, esp
009E06F9	79 03	jns	short 009E06FE
009E06FB	0F9142 40	setno	byte ptr [edx+40]
009E06FF	B8 D3179E00	mov	eax, 9E17D3
009E0704	40	inc	eax
009E0705	FF30	push	dword ptr [eax]
009E0707	C3	retn	

Đọc code chạy như ở trên ta cũng đã biết được sẽ chuyển hướng tới API nào. Giờ ta trace bằng F7/F8 để rõ hơn:

009E06FB	0F9142 40	setno	byte ptr [edx+40]	
009E06FF	B8 D3179E00	mov	eax, 9E17D3	
009E0704	40	inc	eax	
009E0705	FF30	push	dword ptr [eax]	kernel32.GetVersion
009E0707	C3	retn		

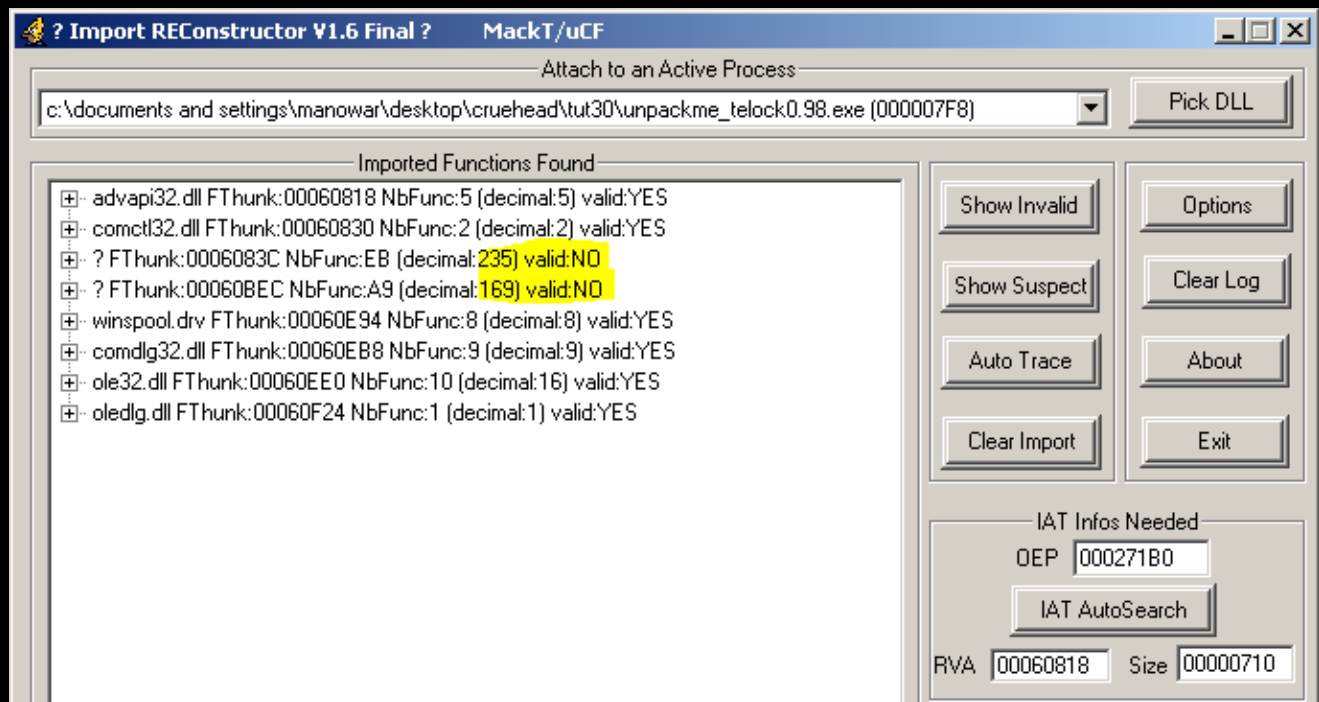
Chúng ta dừng lại tại lệnh **PUSH**, lệnh này đặt địa chỉ của hàm API **GetVersion** vào ngăn xếp, và sau đó thực hiện nhảy tới API này khi thực hiện lệnh **RET**, đây chính là lý do đoạn code này giống như bước trung gian để tới được hàm **GetVersion**. Có thể thấy, tElock đã thực hiện thay thế địa chỉ đầu của hàm API **GetVersion** bằng một địa chỉ trỏ đến một vùng nhớ riêng được tạo ra bởi chính tElock chứ không phải dll, nếu ta tiếp tục trace qua đoạn code trên, ta sẽ đến được hàm API một cách chính xác.

Chính bởi lý do này, khi ta xác định thông tin liên quan tới IAT Start và IAT End, ta không những nhận được thông tin địa chỉ của các hàm trong các DLL mà còn cả những địa chỉ thuộc IAT nhưng chúng lại tham chiếu và đưa ta tới những vùng code của packer tạo ra trong quá trình unpack code, vùng code đó sẽ đóng vai trò trung gian với nhiệm vụ chuyển tới chính xác hàm API.

Tương tự như với Target trước, đến thời điểm này ta đang có các thông tin về OEP, IAT Start, IAT Size như sau:

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

Giữ nguyên OllyDbg, mở ImpREC lên và load process của Unpackme vào. Điền các thông tin liên quan ở trên và thực hiện Get Imports:



Như chúng ta thấy ImpREC hoàn toàn phát hiện ra và lấy cả những địa chỉ được chuyển hướng và gán thông tin NO (tức là không tìm thấy hàm hợp lệ), số lượng API chưa nhận diện được còn rất nhiều, nhấn **Show Invalid** ta thấy:

rva:0006083C	ptr:00400000
rva:00060840	ptr:00400011
rva:00060844	ptr:00400022
rva:00060848	ptr:00400033
rva:0006084C	ptr:00400041
rva:00060850	ptr:00400050
rva:00060854	ptr:0040005F
rva:00060858	ptr:0040007F
rva:0006085C	ptr:0040008D
rva:00060860	ptr:00400080
rva:00060864	ptr:004000C1
rva:00060868	ptr:004000D2
rva:0006086C	ptr:004000E4
rva:00060870	ptr:004000F4
rva:00060874	ptr:00400105
rva:00060878	ptr:00400124
rva:0006087C	ptr:0040013E
rva:00060880	ptr:0040014F
rva:00060884	ptr:00400160
rva:00060888	ptr:00400171
rva:0006088C	ptr:0040017F
rva:00060890	ptr:0040018E
rva:00060894	ptr:0040019D
rva:00060898	ptr:004001BD
rva:0006089C	ptr:004001CB
rva:000608A0	ptr:004001EE
rva:000608A4	ptr:004001FF
rva:000608A8	ptr:00400210
rva:000608AC	ptr:00400222

Các thông tin trùng khớp khi ta làm việc với OllyDbg. ImpREC được hỗ trợ một plugin có thể trace và lấy lại toàn bộ thông tin của các hàm API, tuy nhiên tôi sẽ chưa đề cập tại bài viết này. Đương nhiên, nếu bạn tay to, bạn hoàn toàn có thể tự trace từng địa chỉ, tìm ra hàm API và fix lại hàm bằng tay tại ImpREC 😊.

III. Kết luận

Toàn bộ phần 30 đến đây là kết thúc do tôi nghĩ cũng khá dài rồi, phần này còn đang bỏ ngỏ đoạn fix lại IAT của target UnPackMe_tElock0.98.exe. Tôi sẽ cố gắng dành thời gian để viết về cách fix IAT trong phần tiếp theo, để làm sao khi ImpREC thực hiện Get Imports thì thông tin về hàm API thu được sẽ đầy đủ nhất để phục vụ việc fix dump, đảm bảo cho file chạy mượt mà, không lỗi.

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, dqtn, 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 Rogers** 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