

2013

[Cracking with OllyDbg]

Based on OllyDbg tuts of Ricardo Narvaja (CrackLatinos Team)



www.reaonline.net

kienmanowar



03/08/2013

Mục Lục

I. Giới thiệu chung.....	2
II. Phân tích và xử lý target.....	2
1. Thông tin sơ lược	2
2. Sử dụng OllyDBG đã fix kèm Plugins	4
3. Sử dụng OllyDBG nguyên bản không Plugins	14
III. Kết luận	28

I. Giới thiệu chung

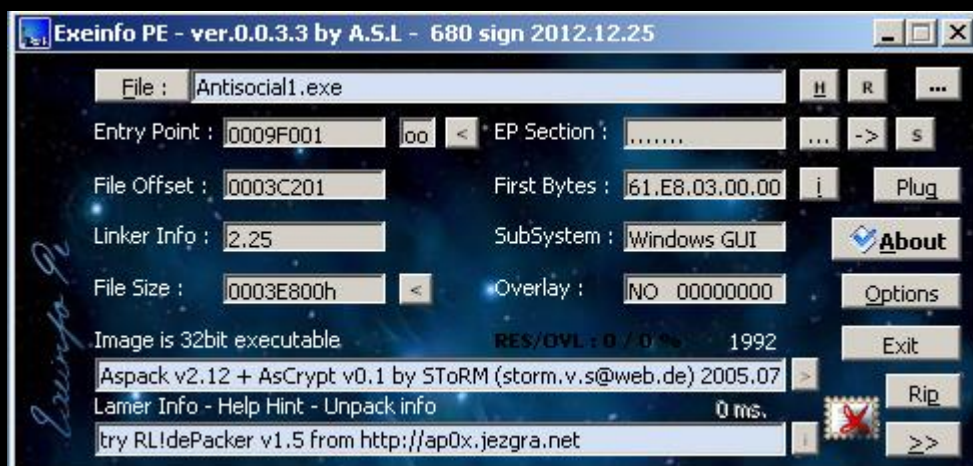
Mưa to gió lớn chả biết làm gì, lại ngồi lọ mọ viết lách để giết thời gian. Lướt FB thì gặp quả status của lão **xnohat** quảng cáo kinh quá: "Sau 3 năm lặn mất tăm trên giang hồ mạng. Cao thủ Reverse & Cracking Kienmanowar đã quay lại với phần 23 của loạt Tut huyền thoại "Ollydbg Tutorials". Mình giờ già rồi, mà có phải cao thủ đếch gì đâu. Các bạn trẻ bây giờ ngon hơn nhiều, mình lót dép ngồi học hỏi cho đỡ bị coi là lạc hậu.

Ở cuối bài 23, tôi có giới thiệu về một crackme có tên là **Antisocial1.exe**, mà theo bác Ricardo thì crackme này hội tụ đủ các tricks mà các bạn đã gặp trong các bài viết có liên quan đến Anti-Debug, cộng thêm một vài trick mới ☺. Phần 24 này sẽ dành thời gian để tìm hiểu về crackme này. Now let's go.....☺

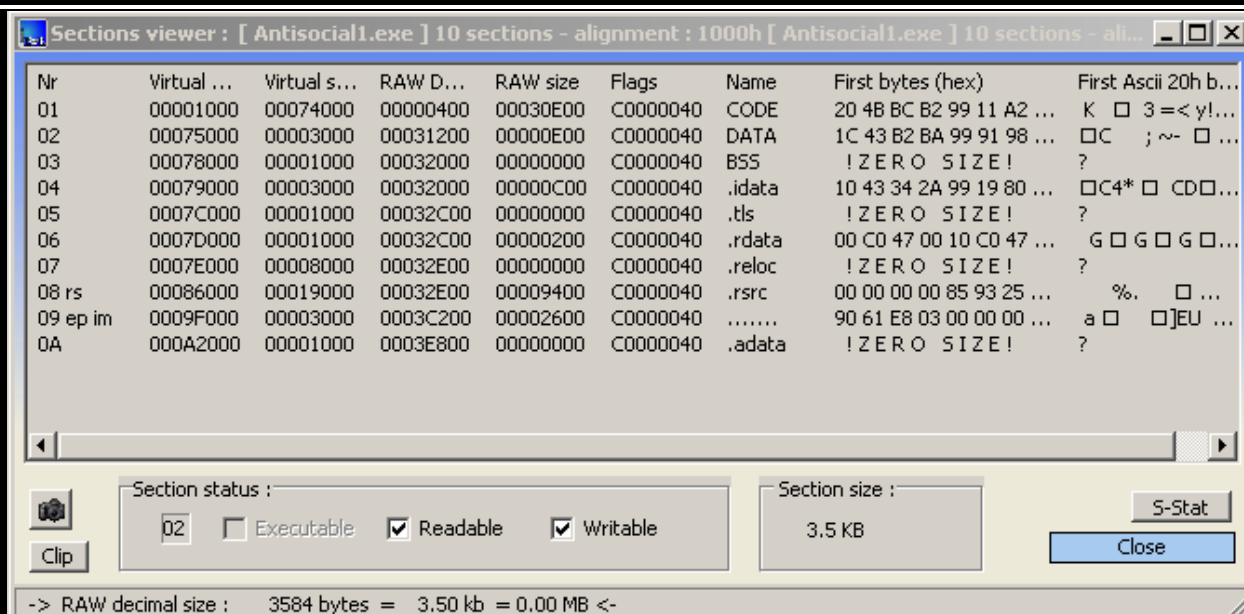
II. Phân tích và xử lý target

1. Thông tin sơ lược

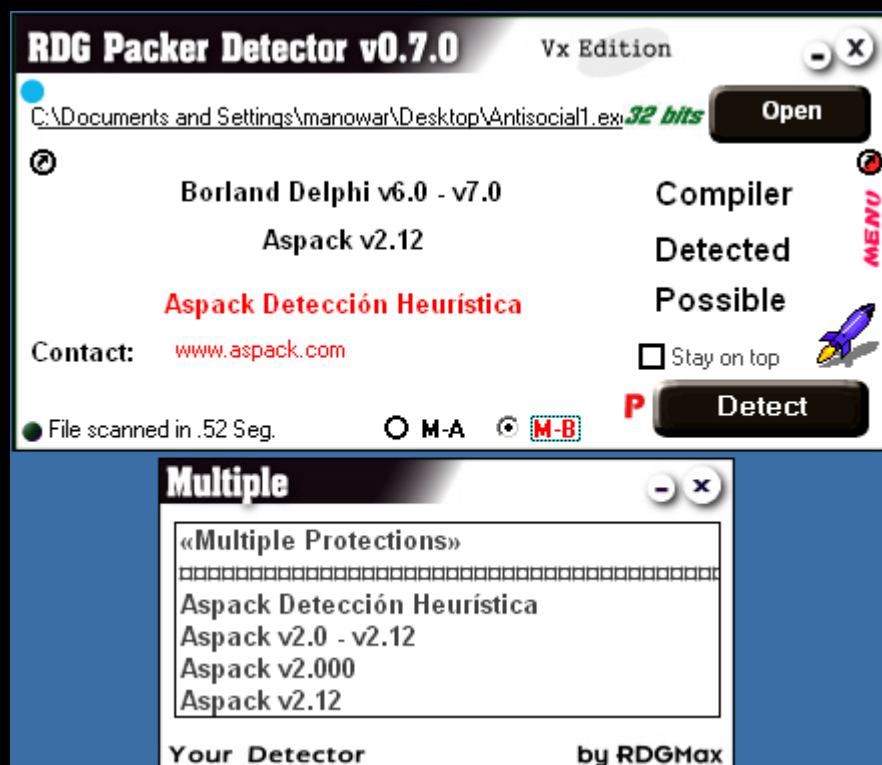
Theo như lão Ricardo thì crackme này bị packed, ta kiểm tra thử xem nó bị packed bởi packer nào:



Ồ Aspack, thử ngó qua thông tin về Section xem thế nào:



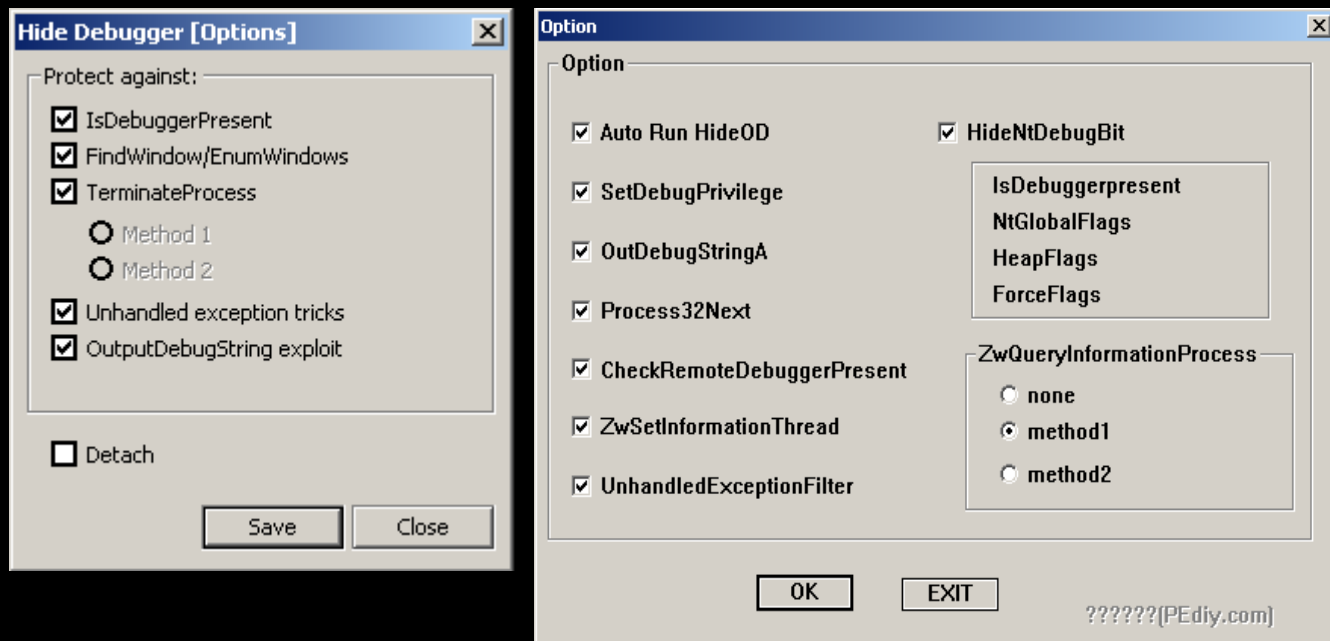
Dòm cái Section thì thấy đâu có giống Section bị packed bằng Aspack, thêm nữa section kiểu này đích thị là được code bởi **Borland Delphi** rồi. Thử kiểm tra tiếp bằng **RDG Packer Detector** xem thế nào:



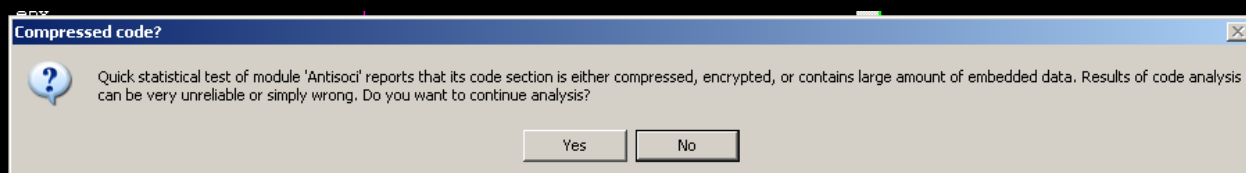
Hình như tác giả của crackme này chơi trick để đánh lừa các trình PE detector thì phải? Thôi kệ, mục tiêu của chúng ta là làm sao thực thi được crackme này trong OllyDBG, còn việc unpack như thế nào không nằm trong phạm vi của bài viết này.

2. Sử dụng OllyDBG đã fix kèm Plugins

Như đã giới thiệu trong phần trước, theo bác Ricardo thì crackme này hội tụ đủ các tricks mà các bạn đã gặp trong các bài viết có liên quan đến Anti-Debug, cộng thêm một vài trick mới, do vậy trước tiên tôi sẽ dùng bản OllyDBG đã được fix bằng cộng cụ **repair_v0.6** (nếu không nhớ các bạn có thể xem lại [OllyDbg_tut21](#)), kèm theo đó là các plugin chuyên dụng là HideOD và HideDebugger.



Súng đạn chuẩn bị đầy đủ cả rồi, tiến hành load crackme vào OllyDBG, ta nhận được thông báo sau:



Thông báo này gợi ý cho ta biết khả năng crackme có thể bị packed, nhấn Yes để tiếp tục, ta dừng lại tại EP của crackme:

0049F001	61	popad
0049F002	E8 03000000	call Antisoci.0049F00A
0049F007	E9 EB045D45	jmp 45A6F4F7
0049F00C	55	push ebp
0049F00D	C3	retn
0049F00E	E8 01000000	call Antisoci.0049F014
0049F013	EB 5D	jmp short Antisoci.0049F072
0049F015	BB EDFFFFFF	mov ebx, -13
0049F01A	03DD	add ebx, ebp
0049F01C	81EB 00F00900	sub ebx, 9F000
0049F022	83BD 22040000	cmp dword ptr [ebp+422], 0
0049F029	899D 22040000	mov dword ptr [ebp+422], ebx
0049F02F	0F85 65030000	jnz Antisoci.0049F39A
0049F035	8D85 2E040000	lea eax, dword ptr [ebp+42E]

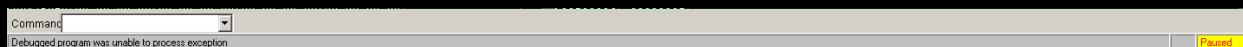
Quan sát đầu tiên của tôi là có gì đó hơi lạ, lệnh **POPAD** thì tôi đã giới thiệu với các bạn ở phần 4, nó sẽ lấy giá trị tại Stack và cất vào các thanh ghi theo thứ tự từ **EDI** về **EAX**. Tuy nhiên, như đã kết luận trong phần 4, cặp lệnh **PUSHAD** và **POPAD** thường đi cùng với nhau. Nếu như ta thấy ở đâu đó trong mã của chương trình xuất hiện lệnh **PUSHAD** thì có nghĩa là đâu đó ở bên dưới chắc chắn sẽ có câu lệnh **POPAD**.

Chính vì lý do trên, với crackme này, ta chưa thấy lệnh **PUSHAD** đâu mà vào đầu chương trình đã là lệnh **POPAD**, đó chính là điều bất thường.

OK, ta thử nhấn F9 để run xem thế nào:

0049F3BA	68 88494700	push	Antisoci.00474988
0049F3BF	C3	retn	
0049F3C0	8B85 26040000	mov	eax,dword ptr [ebp+426]
0049F3C6	8D8D 3B040000	lea	ecx,dword ptr [ebp+43B]
0049F3CC	51	push	ecx
0049F3CD	50	push	eax
0049F3CE	FF95 490F0000	call	near dword ptr [ebp+F49]
0049F3D4	8985 55050000	mov	dword ptr [ebp+555],eax
0049F3DA	8D85 47040000	lea	eax,dword ptr [ebp+447]
0049F3E0	50	push	eax
0049F3E1	FF8F 510F0000	call	near dword ptr [ebp+5F1]

Bị break tại dòng 0049F3BA 68 88494700 push Antisoci.00474988. Nhìn xuống dưới status bar ta thấy như sau:



Thông báo "Debugged program was unable to process exception", tạm dịch là "Chương trình đang bị debug không thể xử lý exception". Vậy exception là cái gì, mượn tạm bài viết của cu **trickyboy** để giải thích sơ lược cho các bạn:

Exception là các lỗi phát sinh trong quá trình Runtime. Để ngăn ngừa chương trình bị crash khi gặp các lỗi này, người lập trình cố gắng ghi lại để xử lý. Nếu xử lý thành công thì chương trình có thể tiếp tục chạy bình thường. Ngược lại thì sẽ gửi thông báo loại lỗi gì và gọi Exit Process để thoát chương trình. Cách làm những thao tác trên chính Handle Exception. (tạm dịch là điều khiển lỗi)

Có nhiều phương pháp để handle exception, phổ biến là SEH (Structured exception handling). SEH có thể dùng để xử lý cả 2 loại lỗi: exception cứng và mềm. Exception cứng thường phát sinh trong quá trình tính toán như lỗi chia không, tràn bộ đếm. Exception mềm như lỗi truy xuất con trỏ null, các tham số ko hợp lệ...

SEH ban đầu chỉ để dùng trong các chương trình code bằng C, nhưng về sau thì cho cả C++, MFC mặc dù theo lời khuyên của các chuyên gia thì: C++ exception handling nên dùng cho C++ và MFC exception handling nên dùng cho MFC. (các bạn tự tìm hiểu thêm nhé)

Để có được thông tin chính xác hơn liên quan tới exception ta mở cửa sổ Log:

0049F001	Program entry point Analysing Antisoci 0 heuristical procedures
0049F3BA	Access violation when writing to [00130000]
0049F3BA	Access violation when writing to [00130000]
	Debugged program was unable to process exception

Như vậy là tại 0049F3BA, lệnh `PUSH` đang cố gắng ghi (đẩy dữ liệu) vào Stack mà tại vùng đó không được quyền ghi. Quan sát cửa sổ Stack ta thấy như sau:

00130000	78746341
00130004	00000020
00130008	00000001
0013000C	00002498
00130010	000000C4
00130014	00000000
00130018	00000020
0013001C	00000000
00130020	00000014
00130024	00000001
00130028	00000006
0013002C	00000034
00130030	00000114

Đỉnh của Stack là 0x00130000, nhấn **Alt+M** để mở cửa sổ **Memory map**:

Address	Size	Owner	Section	Contains	Type	Access	Initial
00010000	00001000				Priv	RW	RW
00020000	00001000				Priv	RW	RW
0012B000	00001000				Priv	RW	Gua: RW
0012C000	00004000			stack of main thread	Priv	RW	Gua: RW
00130000	00003000				Map	R	R
00140000	00006000				Priv	RW	RW
00240000	00006000				Priv	RW	RW
00250000	00003000				Map	RW	RW
00260000	00016000				Map	R	R
00280000	0003D000				Map	R	R
002C0000	00041000				Map	R	R
00310000	00006000				Map	R	R
00320000	00001000				Priv	RWE	RWE
00330000	00001000				Priv	RWE	RWE
00340000	00001000				Priv	RWE	RWE

Tại màn hình **Memory map**, như các bạn thấy vùng Stack trên máy tôi bắt đầu từ 0x0012C000 tới 0x0012FFFF (có thể khác trên máy các bạn), lỗi mà chúng ta nhận được ở trên lại nằm ngoài vùng Stack, hiện tại đang chỉ tới Section tiếp theo bắt đầu bởi địa chỉ 0x00130000, vùng này không phải Stack đồng thời như các bạn thấy thông tin tại cột Access thì Section này chỉ có quyền Read (đọc) chứ không có quyền Write (ghi). Như vậy, ta dính exception trên là đương nhiên rồi.

Có thể có bạn nào đó sẽ nảy ra ý nghĩ là sao không thử gán thêm quyền Write cho Section đó thì xin thưa với các bạn là bỏ ý nghĩ đó đi nhé ☺.

OK, giờ ta quay lại cửa sổ CPU, ta đang dừng tại lệnh `Push` gây ra exception. Cuộn chuột lên một chút, các bạn sẽ thấy crackme này thực hiện thêm một lệnh `POPAD` khác và sau nó là một lệnh `JNZ`, mà theo như mũi tên định hướng của lệnh nhảy thì nó sẽ nhảy tới lệnh `PUSH` mà con trỏ của chúng ta đang đứng:

0049F3A7	0BC9	or	ecx,ecx
0049F3A9	8985 A8030000	mov	dword ptr [ebp+3A8],eax
0049F3AF	61	popad	
0049F3B0	75 08	jnz	short Antisoci.0049F3BA
0049F3B2	B8 01000000	mov	eax,1
0049F3B7	C2 0C00	retn	0C
0049F3BA	68 88494700	push	Antisoci.00474988
0049F3BF	C3	retn	

Thử đặt một BP tại lệnh `POPAD`, sau đó restart OllyDBG và nhấn F9, ta dừng lại tại lệnh `POPAD`:

0049F3AF	61	popad	
0049F3B0	75 08	jnz	short Antisoci.0049F3BA
0049F3B2	B8 01000000	mov	eax,1
0049F3B7	C2 0C00	retn	0C
0049F3BA	68 88494700	push	Antisoci.00474988
0049F3BF	C3	retn	

Quan sát cửa sổ Stack, ta thấy:

0012FFE4	7C8399F3	SE handler
0012FFE8	7C816D58	kernel32.7C816D58
0012FFEC	00000000	
0012FFF0	00000000	
0012FFF4	00000000	
0012FFF8	0049F001	offset Antisoci.<ModuleEntryPoint>
0012FFFC	00000000	

Có thể thấy rằng, trước khi thực hiện lệnh `POPAD`, chúng ta vẫn đang ở trong vùng Stack. Giờ ta thử nhấn F8 để trace qua lệnh `POPAD` xem thế nào:

00130004	00000020
00130008	00000001
0013000C	00002498
00130010	000000C4
00130014	00000000
00130018	00000020
0013001C	00000000
00130020	00000014
00130024	00000001
00130028	00000006
0013002C	00000034
00130030	00000114

Như vậy, sau khi thực hiện lệnh ta đã nhảy ra ngoài vùng Stack. Nếu bạn đã nắm được nguyên lý hoạt động của lệnh `POPAD` thì sẽ hiểu được tại sao ESP lại trở tới `0x00130004`. Do lệnh `POPAD` hoàn toàn không tác động tới cờ, nên cờ ZF lúc này giữ nguyên giá trị 0 trước đó. Do $ZF = 0x0$ cho nên lệnh nhảy `JNZ` sẽ thực hiện và nhảy tới chỗ lệnh `0049F3BA 68 88494700 push Antisoci.00474988`, như vậy chúng ta sẽ dính exception:

0049F3AF	61	popad
0049F3B0	75 08	jnz short Antisoci.0049F3BA
0049F3B2	B8 01000000	mov eax,1
0049F3B7	C2 0C00	retn 0C
0049F3BA	68 88494700	push Antisoci.00474988
0049F3BF	C3	retn

Vậy các bạn có thể thấy ngay cách đầu tiên để giải quyết việc không bị nhảy ra khỏi vùng Stack là NOP luôn cái lệnh POPAD trước lệnh nhảy JNZ là ok. Tuy nhiên, trước khi thực hiện tôi cũng muốn tìm hiểu xem tại sao lệnh PUSH lại gây ra exception. Lăn ngược lên một chút các bạn sẽ thấy lệnh nhảy tại 0049F395 ^\E9 EBFEFFFF jmp Antisoci.0049F285. Tiến hành đặt BP tại địa chỉ 0049F39A:

0049F395	^ E9 EBFEFFFF	jmp Antisoci.0049F285
0049F39A	B8 88490700	mov eax,74988

Đặt BP xong, restart lại OllyDBG, nhấn F9 để run. OllyDBG sẽ dừng lại tại địa chỉ 0x0049F39A:

0049F395	^ E9 EBFEFFFF	jmp Antisoci.0049F285
0049F39A	B8 88490700	mov eax,74988
0049F39F	50	push eax
0049F3A0	0385 22040000	add eax,dword ptr [ebp+422]
0049F3A6	59	pop ecx
0049F3A7	0BC9	or ecx,ecx
0049F3A9	8985 A8030000	mov dword ptr [ebp+3A8],eax
0049F3AF	61	popad
0049F3B0	75 08	jnz short Antisoci.0049F3BA
0049F3B2	B8 01000000	mov eax,1
0049F3B7	C2 0C00	retn 0C
0049F3BA	68 88494700	push Antisoci.00474988
0049F3BF	C3	retn

Quan sát cửa sổ Stack, ta thấy vẫn ổn. Phân tích một chút đoạn code ở trên, lệnh đầu tiên sẽ chuyển (gán) giá trị 74988 vào thanh ghi eax, lệnh tiếp theo sẽ cất giá trị của eax vào trong Stack. Ta nhấn F8 để trace qua hai lệnh này và dừng lại tại lệnh 0049F3A0 0385 22040000 add eax,dword ptr [ebp+422] ; Antisoci.00400000. Quan sát thông tin tại cửa sổ Tip, ta có được như sau:

```
ss:[0049F435]=00400000 (Antisoci.00400000), ASCII "MZP"
eax=00074988
```

Như vậy là giá trị Imagebase được cộng vào thanh ghi eax, thanh ghi eax sẽ mang giá trị là 00474988. Lệnh tiếp theo sẽ lấy giá trị tại đỉnh của Stack gán vào thanh ghi ecx, như các bạn quan sát lúc đầu thì đỉnh của stack lúc này đang là: 0012FFEO 00074988 Pointer to next SEH record, như vậy ecx sẽ mang giá trị 74988. Nhấn F8 để trace tiếp tới địa chỉ 0049F3A9, quan sát cửa sổ Tip các bạn sẽ thấy [EBP+3A8] = 49F3BB. Vậy 49F3BB là gì? Để ý câu lệnh tại địa chỉ 49F3BA:

0049F3A9	8985 A8030000	mov	dword ptr [ebp+3A8], eax	Antisoci.00474988
0049F3AF	61	popad		
0049F3B0	75 08	jnz	short Antisoci.0049F3BA	
0049F3B2	B8 01000000	mov	eax, 1	
0049F3B7	C2 0C00	retn	0C	
0049F3BA	68 00000000	push	0	
0049F3BF	C3	retn		

Các bạn quan sát ngược lại lên hình trên một chút thì thấy hai lệnh `push` tại 49F3BA có giá trị khác nhau, như vậy các bạn có thể kết luận 49F3BB là một địa chỉ chứa giá trị 0x00, sau khi thực hiện lệnh tại 0049F3A9 8985 A8030000 `mov dword ptr [ebp+3A8], eax ; Antisoci.00474988`, giá trị tại 49F3BB sẽ bị ghi đè bởi giá trị của `eax` (00474988). Cái kỹ thuật này theo như giang hồ hay gọi là **"Self-Modifying-Code"** gì gì đó ☺. Trace qua đoạn code trên, quan sát sẽ thấy lệnh `PUSH 0` được thay đổi thành `PUSH 00474988`:

0049F3A9	8985 A8030000	mov	dword ptr [ebp+3A8], eax	
0049F3AF	61	popad		
0049F3B0	75 08	jnz	short Antisoci.0049F3BA	
0049F3B2	B8 01000000	mov	eax, 1	
0049F3B7	C2 0C00	retn	0C	
0049F3BA	68 88494700	push	Antisoci.00474988	
0049F3BF	C3	retn		

Lúc này ta dừng lại tại lệnh `POPAD`, trace qua lệnh này thì Stack của chúng ta sẽ bị biến đổi. Như vậy, tới đây các bạn có thể lờ mờ đoán ra, nếu ta `NOP` lệnh `POPAD` thì lệnh `Retn` sẽ tương ứng với một lệnh nhảy tới địa chỉ 00474988, và địa chỉ này chính là OEP gốc của target.

Tới đây, ta có thể kết luận, để vượt qua được exception thì ta sẽ patch lệnh `POPAD` bằng lệnh `NOP`. Tuy nhiên, các này hơi thô bạo, có một cách làm khác có vẻ lịch sự hơn. Như các bạn đã theo dõi từ đầu đến giờ thì sẽ thấy có hai lệnh `POPAD`, một tại đầu lúc ta load target vào OllyDBG: 0049F001 > 61 `popad` và hai là tại 0049F3AF 61 `popad`. Mà như tôi đã phân tích thì `pushad/popad` là cặp lệnh thường đi cùng nhau, do đó để qua mặt được exception, ta chỉ cần thay lệnh `popad` tại 0049F001 thành `pushad` là xong. OK, bỏ hết các BP đi và chỉ giữ lại BP tại lệnh `POPAD` thứ hai tại 0049F3AF, restart lại OllyDBG và sửa lại lệnh `POPAD` thành lệnh `PUSHAD`, tương tự như sau:

0049F001	60	pushad		
0049F002	E8 03000000	call	Antisoci.0049F00A	
0049F007	- E9 EB045D45	jmp	45A6F4F7	
0049F00C	55	push	ebp	
-----	--			

Sau khi sửa xong, nhấn F9 sẽ dừng lại tại lệnh `POPAD` đã đặt BP. Nhấn F8 để trace qua lệnh `POPAD` và quan sát cửa sổ Stack:

0012FFC4	7C816D4F	RETURN to kernel32.7C816D4F
0012FFC8	7C910738	ntdll.7C910738
0012FFCC	FFFFFFFF	
0012FFD0	7FFDE000	
0012FFD4	8054A6ED	
0012FFD8	0012FFC8	
0012FFDC	86461020	
0012FFE0	FFFFFFFF	End of SEH chain
0012FFE4	7C8399F3	SE handler

Stack lúc này OK rồi, không bị thay đổi nữa. Tiếp tục trace tiếp và trace qua lệnh Retn, ta sẽ tới OEP của chương trình:

00474988	55	db	55	CHAR 'U'
00474989	8B	db	8B	
0047498A	EC	db	EC	
0047498B	83	db	83	
0047498C	C4	db	C4	
0047498D	F0	db	F0	
0047498E	53	db	53	CHAR 'S'
0047498F	B8	db	B8	
00474990	E0	db	E0	

Nhìn code có vẻ không chuẩn, nhấn chuột phải chọn **Analysis > Remove analysis from module**, ta có được như sau:

00474988	55	push	ebp	
00474989	8BEC	mov	ebp,esp	
0047498B	83C4 F0	add	esp,-10	
0047498E	53	push	ebx	
0047498F	B8 E0464700	mov	eax,Antisoci.004746E0	
00474994	E8 731EF9FF	call	Antisoci.0040680C	
00474999	8B1D B86F4700	mov	ebx,dword ptr [476FB8]	Antisoci.00478D7C
0047499F	8B03	mov	eax,dword ptr [ebx]	
004749A1	E8 42F9FFFF	call	Antisoci.004742E8	
004749A6	8B03	mov	eax,dword ptr [ebx]	
004749A8	E8 67F9FFFF	call	Antisoci.00474314	

Ok có vẻ ngon lành rồi, nhấn F9 run thử cái xem thế nào:



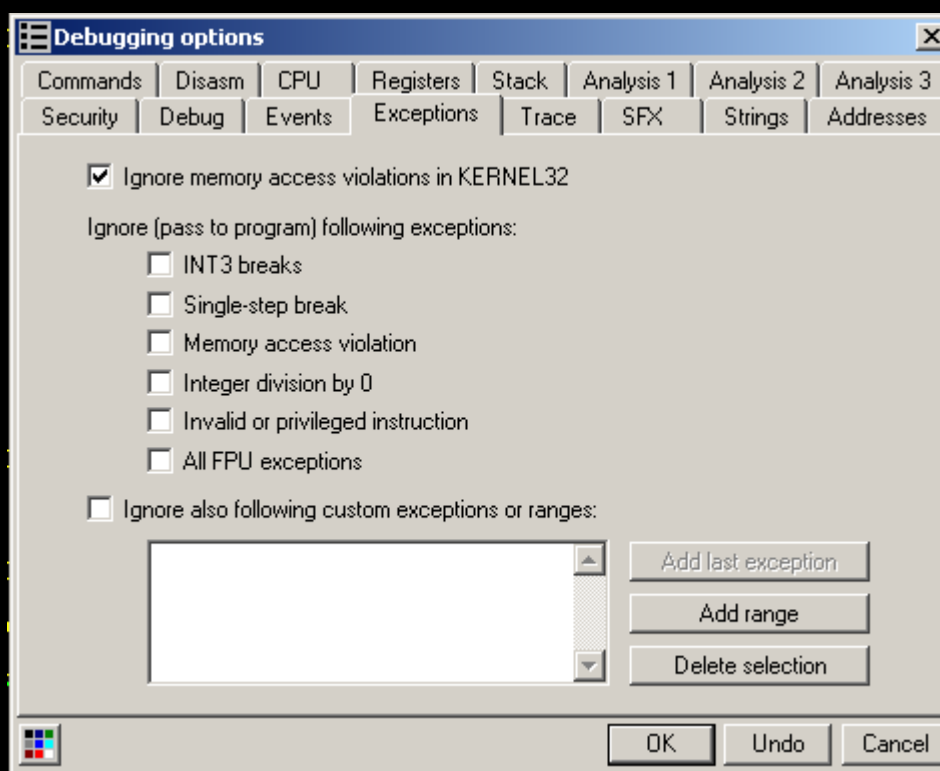
Oạch Terminated, thông tin nhận được như hình trên. Quan sát cửa sổ Log xem có kiểm được thông tin gì không:

```

0049F001 Program entry point
0049F3AF Breakpoint at Antisoci 0049F3AF
004730EA Access violation when reading [FFFFFFFF]
69450000 Module C:\WINDOWS\system32\faultrep.dll
77C00000 Module C:\WINDOWS\system32\VERSION.dll
769C0000 Module C:\WINDOWS\system32\USERENV.dll
76360000 Module C:\WINDOWS\system32\WINSTA.dll
5B860000 Module C:\WINDOWS\system32\NETAPI32.dll
76F50000 Module C:\WINDOWS\system32\WTSAPI32.dll
77920000 Module C:\WINDOWS\system32\SETUPAPI.dll
77F60000 Module C:\WINDOWS\system32\SHLWAPI.dll
77B40000 Module C:\WINDOWS\system32\Apphelp.dll
5B860000 Unload C:\WINDOWS\system32\NETAPI32.dll
69450000 Unload C:\WINDOWS\system32\faultrep.dll
76360000 Unload C:\WINDOWS\system32\WINSTA.dll
769C0000 Unload C:\WINDOWS\system32\USERENV.dll
76F50000 Unload C:\WINDOWS\system32\WTSAPI32.dll
77920000 Unload C:\WINDOWS\system32\SETUPAPI.dll
Process terminated, exit code C0000005 (-1073741819.)

```

Chà lại dính exception khác. Restart lại OllyDBG, thực hiện lại bước trên để tới OEP của crackme : `00474988 55 push ebp`. Nhấn **Alt+O** để mở cửa sổ **Debugging Options**, chọn tab **Exceptions**, bỏ hết các tùy chọn trừ cái đầu tiên:



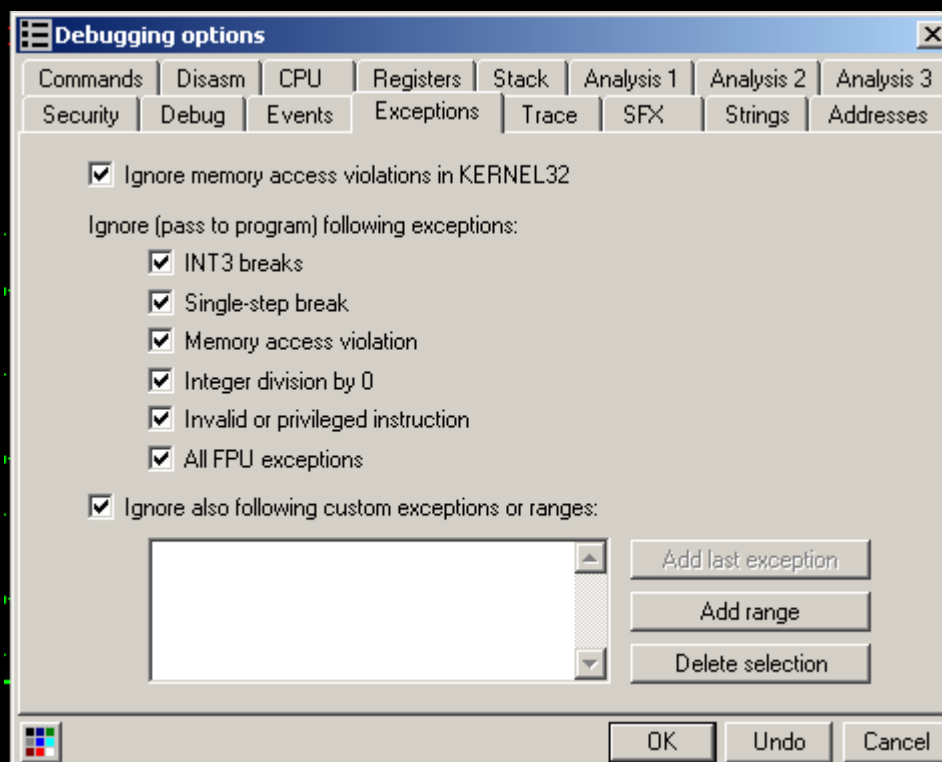
Nhấn OK để trở về màn hình CPU, tiếp tục nhấn F9 để run crackme xem thế nào:

004730EA	CD 68	int	68	
004730EC	66:3D 86F3	cmp	ax,0F386	
004730F0	75 07	jnz	short Antisoci.004730F9	
004730F2	66:B8 FE00	mov	ax,0FE	
004730F6	66:E7 64	out	64,ax	I/O command
004730F9	C3	retn		

OllyDBG đã dừng lại, quan sát thanh Status ta thấy thông tin exception trùng với thông tin thu được ở cửa sổ Log phía trên:

Access violation when reading [FFFFFFFF] - use Shift+F7/F8/F9 to pass exception to program

Như vậy, lệnh `int 68` chính là nguyên nhân gây ra exception, theo như giảng hồ truyền tai nhau thì đây là một trick để anti-SoftIce. Nếu như sử dụng SoftIce thì sau lệnh `int 68` thanh ghi eax sẽ thay đổi thành `0F386h`, đối với các hệ điều hành XP, 2000, NT thì ngắt này sẽ tung ra một "Access violation" như ta thấy ở trên. Đến đây ta đã biết nguyên nhân vì sao, giờ restart lại OllyDBG, tới OEP như đã thực hiện ở trên. Sau đó tại OEP, tiến hành tìm kiếm toàn bộ các lệnh `int 68`, mục đích là không bỏ sót lệnh nào. Trước khi tìm kiếm ta nhấn **Alt+O** để mở **Debugging options** và chỉnh lại như sau:



Sau đó chuột phải chọn **Search for > All commands:**

Assembly code snippet (addresses 00474988 to 004749F0):

```

00474988 55      push    ebp
00474989 8BEC    mov     ebp, esp
0047498B 83C4 F0 add     esp, -10
0047498E 53      push    ebx
0047498F B8 E0464700 mov     eax, 46E04700
00474994 E8 731EF9FF call    Antisoci.00478D7C
00474999 8B1D B86F4700 mov     ebx, 4700B86F
0047499F 8B03    mov     eax, [ebx]
004749A1 E8 42F9FFFF call    Antisoci.00478D7C
004749A6 8B03    mov     eax, [ebx]
004749A8 E8 67F9FFFF call    Antisoci.00478D7C
004749AD 8B03    mov     eax, [ebx]
004749AF E8 90F9FFFF call    Antisoci.00478D7C
004749B4 8B03    mov     eax, [ebx]
004749B6 E8 8DE4FFFF call    Antisoci.00478D7C
004749BB 8B03    mov     eax, [ebx]
004749BD E8 3AE7FFFF call    Antisoci.00478D7C
004749C2 8B03    mov     eax, [ebx]
004749C4 E8 DFE7FFFF call    Antisoci.00478D7C
004749C9 8B03    mov     eax, [ebx]
004749CB E8 18E7FFFF call    Antisoci.00478D7C
004749D0 A1 DC6E4700 mov     eax, dword ptr [476EDC]
004749D5 8B00    mov     ebx, [eax]
004749D7 E8 6CE7FDEF call    Antisoci.00453148
004749DC 8BCB    mov     ecx, ebx
004749DE A1 DC6E4700 mov     eax, dword ptr [476EDC]
004749E3 8B00    mov     ebx, [eax]
004749E5 8B15 CC284700 mov     ebx, dword ptr [4728CC]
004749EB E8 70E7FDEF call    Antisoci.00453160
004749F0 A1 DC6E4700 mov     eax, dword ptr [476EDC]
ebp=0012FFF0

```

Search menu options:

- Backup
- Copy
- Binary
- Assemble
- Label
- Comment
- Breakpoint
- Run trace
- Go to
- Follow in Dump
- Search for
 - Name (label) in current module Ctrl+N
 - Name in all modules
 - Command Ctrl+F
 - Sequence of commands Ctrl+S
 - Constant
 - Binary string Ctrl+B
 - Next Ctrl+L
 - All intermodular calls
 - All commands
 - All sequences
 - All constants
 - All switches
 - All referenced text strings
- Find references to
- View
- Copy to executable
- Analysis
- Appearance

Assembly code snippet (addresses 00474988 to 004749F0):

```

00474988 55      push    ebp
00474989 8BEC    mov     ebp, esp
0047498B 83C4 F0 add     esp, -10
0047498E 53      push    ebx
0047498F B8 E0464700 mov     eax, 46E04700
00474994 E8 731EF9FF call    Antisoci.00478D7C
00474999 8B1D B86F4700 mov     ebx, 4700B86F
0047499F 8B03    mov     eax, [ebx]
004749A1 E8 42F9FFFF call    Antisoci.00478D7C
004749A6 8B03    mov     eax, [ebx]
004749A8 E8 67F9FFFF call    Antisoci.00478D7C
004749AD 8B03    mov     eax, [ebx]
004749AF E8 90F9FFFF call    Antisoci.00478D7C
004749B4 8B03    mov     eax, [ebx]
004749B6 E8 8DE4FFFF call    Antisoci.00478D7C
004749BB 8B03    mov     eax, [ebx]
004749BD E8 3AE7FFFF call    Antisoci.00478D7C
004749C2 8B03    mov     eax, [ebx]
004749C4 E8 DFE7FFFF call    Antisoci.00478D7C
004749C9 8B03    mov     eax, [ebx]
004749CB E8 18E7FFFF call    Antisoci.00478D7C
004749D0 A1 DC6E4700 mov     eax, dword ptr [476EDC]
004749D5 8B00    mov     ebx, [eax]
004749D7 E8 6CE7FDEF call    Antisoci.00453148
004749DC 8BCB    mov     ecx, ebx
004749DE A1 DC6E4700 mov     eax, dword ptr [476EDC]
004749E3 8B00    mov     ebx, [eax]
004749E5 8B15 CC284700 mov     ebx, dword ptr [4728CC]
004749EB E8 70E7FDEF call    Antisoci.00453160
004749F0 A1 DC6E4700 mov     eax, dword ptr [476EDC]
ebp=0012FFF0

```

Find all commands dialog box:

Search field:

Buttons: Find, Cancel

Nhấn Find, ta có các vị trí sau có lời gọi `int 68`:

Address	Disassembly	Comment
00450F8E	int 68	
004730EA	int 68	
00473EB7	int 68	
00474988	push ebp	(Initial CPU selection)

Chọn từng lệnh, nhấn Enter để follow và NOP:

Assembly code snippet (addresses 00450F8E to 00450F93):

```

00450F8E 90      nop
00450F8F 90      nop
00450F90 007F 00 add     byte ptr [edi], bh
00450F93 006A 00 add     byte ptr [edx], ch

```

Assembly code snippet (addresses 004730EA to 004730F9):

```

004730EA 90      nop
004730EB 90      nop
004730EC 66:3D 86F3 cmp     ax, 0F386
004730F0 75 07    jnz     short Antisoci.004730F9
004730F2 66:B8 FE00 mov     ax, 0FE
004730F6 66:E7 64 out     64, ax
004730F9 C3      retm

```

00473EB7	90	nop	
00473EB8	90	nop	
00473EB9	66:3D 86F3	cmp	ax,0F386
00473EBD	75 07	inz	short Antisoci 00473EC6

Sau khi patch toàn bộ các lệnh `int 68` như trên, ta nhấn F9 để thử run crackme xem tình hình thế nào:



Crackme thực thi mượt mà ☺, có vẻ không bị crash. Như vậy là với bản OllyDBG đã được fix bằng công cụ **repair_v0.6**, cộng thêm các Plugins như đã nói ở đầu bài thì chúng ta đã có thể tạm run được crackme một cách bình thường. Vậy nếu như ta sử dụng OllyDBG nguyên bản không fix, không sử dụng plugins thì sẽ thế nào (bỏ hết các Plugin là HideOD và HideDebugger).

3. Sử dụng OllyDBG nguyên bản không Plugins

Làm lần lượt các bước như trên cho tới khi patch hết toàn bộ các lệnh `int 68` thành NOP, sau đó nhấn F9 để run, các bạn sẽ thấy OllyDBG của chúng ta bị "bay" luôn, thế mới ác ☹. Bị Terminate như thế này thì đầu tiên nghi ngờ có dùng hàm API `TerminateProcess`, tôi nghi ngờ là để có căn cứ mà mò tiếp.

Ok, khởi động lại Olly và thực hiện lại các thao tác trên, sau đó tại OEP của crackme thực hiện tìm kiếm như sau **Search for -> All intermodulas calls**:

Address	Hex	Mnemonic	Comment
00474988	55	push	ebp
00474989	8BEC	mov	edi,ebp
0047498B	83C4 F0	add	esp,0F0
0047498E	53	push	ebx
0047498F	B8 E0464700	mov	eax,00464700
00474994	E8 731EF9FF	call	Antisoci.00478D7
00474999	8B1D B86F4700	mov	ebx,00464700
0047499F	8B03	mov	ebx,00464700
004749A1	E8 42F9FFFF	call	Antisoci.00478D7
004749A6	8B03	mov	ebx,00464700
004749A8	E8 67F9FFFF	call	Antisoci.00478D7
004749AD	8B03	mov	ebx,00464700
004749AF	E8 90F9FFFF	call	Antisoci.00478D7
004749B4	8B03	mov	ebx,00464700
004749B6	E8 8DE4FFFF	call	Antisoci.00478D7
004749BB	8B03	mov	ebx,00464700
004749BD	E8 3AE7FFFF	call	Antisoci.00478D7
004749C2	8B03	mov	ebx,00464700
004749C4	E8 DFE7FFFF	call	Antisoci.00478D7
004749C9	8B03	mov	ebx,00464700
004749CB	E8 18E7FFFF	call	Antisoci.00478D7
004749D0	A1 DC6E4700	mov	eax,dword ptr [476E4700]
004749D5	8B00	mov	ebx,00464700
004749D7	E8 6CE7FDFD	call	Antisoci.00453148
004749DC	8BCB	mov	ecx,ebx
004749DE	A1 DC6E4700	mov	eax,dword ptr [476E4700]
004749E3	8B00	mov	ebx,00464700
004749E5	8B15 CC284700	mov	ebx,dword ptr [47284700]
004749EB	E8 70E7FDFD	call	Antisoci.00453160
004749F0	A1 DC6E4700	mov	eax,dword ptr [476E4700]

ebp=0012FFFF

Nhấn vào Destination để sắp xếp lại, sau đó tìm kiếm hàm `TerminateProcess`, ta có được thông tin như sau:

00451C62	call	Antisoci.00407200	user32.SystemParametersInfoA
00472F09	call	Antisoci.00406AF8	kernel32.TerminateProcess
00472F3E	call	Antisoci.00406AF8	kernel32.TerminateProcess
00472F73	call	Antisoci.00406AF8	kernel32.TerminateProcess
00472FA8	call	Antisoci.00406AF8	kernel32.TerminateProcess
00472FDD	call	Antisoci.00406AF8	kernel32.TerminateProcess
00473012	call	Antisoci.00406AF8	kernel32.TerminateProcess
0047302F	call	Antisoci.00406AF8	kernel32.TerminateProcess

Nhấn đúp vào hàm đầu tiên ta sẽ tới đây:

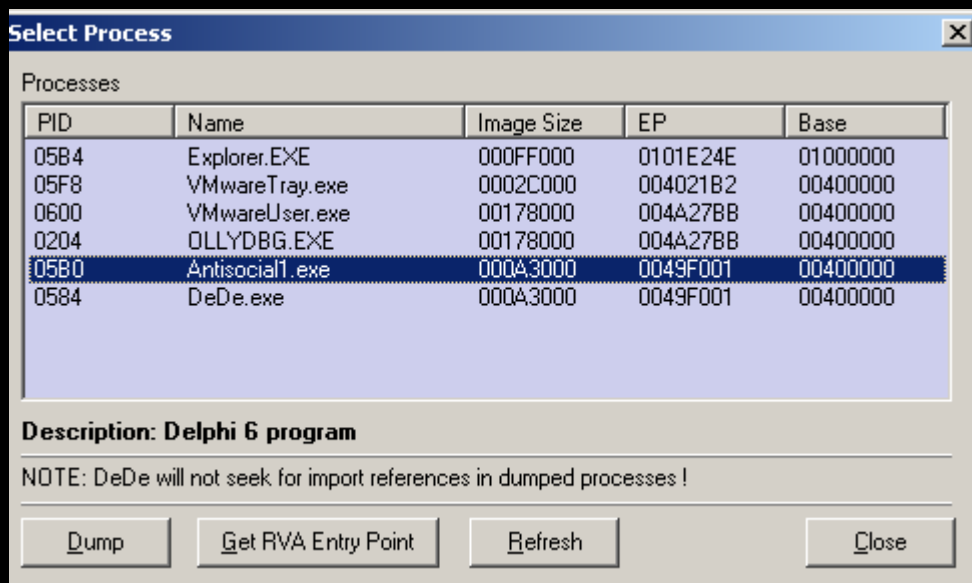
00472ED4	E8 AB5BF9FF	call	Antisoci.00408A84	
00472ED9	8D95 CCFEFFFF	lea	edx,dword ptr [ebp-134]	
00472EDF	B8 6C304700	mov	eax,Antisoci.0047306C	ASCII "FYF/SFEBPMQG"
00472EE4	E8 BBFEFFFF	call	Antisoci.00472DA4	
00472EE9	8B95 CCFEFFFF	mov	edx,dword ptr [ebp-134]	
00472EEF	8B45 FC	mov	eax,dword ptr [ebp-4]	
00472EF2	E8 F918F9FF	call	Antisoci.004047F0	
00472EF7	75 15	jnz	short Antisoci.00472F0E	
00472EF9	6A 00	push	0	
00472EFB	8B46 08	mov	eax,dword ptr [esi+8]	
00472EFE	50	push	eax	
00472EFF	6A FF	push	-1	
00472F01	6A 01	push	1	
00472F03	E8 883BF9FF	call	Antisoci.00406A90	jmp to kernel32.OpenProcess
00472F08	50	push	eax	
00472F09	E8 EA3BF9FF	call	Antisoci.00406AF8	jmp to kernel32.TerminateProcess
00472F0E	8D95 C8FEFFFF	lea	edx,dword ptr [ebp-138]	
00472F14	B8 84304700	mov	eax,Antisoci.00473084	ASCII "FYF/HCEZMMP"

Nhìn sơ sơ thì thấy cái chuỗi `ASCII "FYF/SFEBPMQG"` như kiểu một dạng encrypted string. Chuỗi này khả năng sau đó sẽ được decrypt thành clear text, rồi đem đi so sánh,

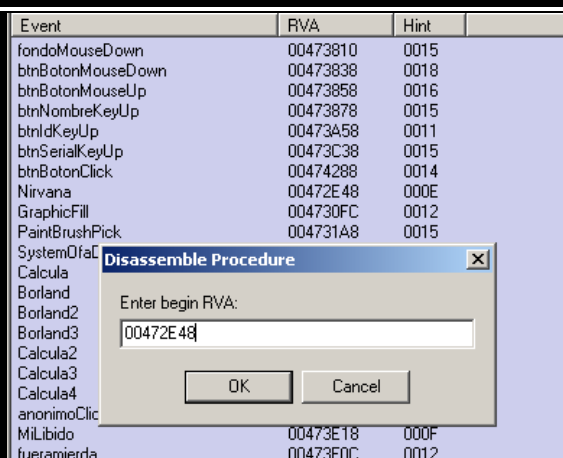
nếu giống thì sẽ gọi hàm `TerminateProcess`. Tạm thời phân tích và phán đoán như vậy đã, giờ cuộn chuột lên trên đầu để tìm chỗ bắt đầu của hàm và đặt BP tại đó:

00472E48	55	push	ebp
00472E49	8BEC	mov	ebp,esp
00472E4B	81C4 B8FEFFFF	add	esp,-148
00472E51	53	push	ebx
00472E52	56	push	esi
00472E53	33D2	xor	edx,edx
00472E55	8995 B8FEFFFF	mov	dword ptr [ebp-148],edx
00472E5B	8995 BCFEFFFF	mov	dword ptr [ebp-144],edx
00472E61	8995 C0FEFFFF	mov	dword ptr [ebp-140],edx
00472E67	8995 C4FEFFFF	mov	dword ptr [ebp-13C],edx
00472E6D	8995 C8FEFFFF	mov	dword ptr [ebp-138],edx
00472E73	8995 CCFEFFFF	mov	dword ptr [ebp-134],edx

Nhấn F9 để run, ta sẽ dừng lại tại `00472E48 55 push ebp`. Như đã phân tích từ đầu, thông tin về section của crackme này cho ta biết nó được code bằng Delphi, mà Delphi thì lại có hàng đặc trị là DeDe. Mở DeDe lên, chọn **Tools > Dump Active Process**:



Nhấn Dump để cho DeDe tiến hành analyze crackme. Sau khi DeDe analyze xong, chọn tab Procedures, sau đó chọn tiếp **OFFS (Disassemble proc by RVA)** ở góc phải màn hình, nhập thông tin RVA như sau (điểm mà ta đang break trong OllyDbg):



Nhấn OK, ta sẽ tới đây:

```

00472E48  CC          int      3
00472E49  8BEC        mov     ebp, esp
00472E4B  81C4B8FEFFFF add    esp, $FFFFFFEB8
00472E51  53          push   ebx
00472E52  56          push   esi
00472E53  33D2        xor     edx, edx
00472E55  8995B8FEFFFF mov     [ebp+FFFFFFEB8], edx
00472E5B  8995BCFEFFFF mov     [ebp+FFFFFFEBC], edx
00472E61  8995C0FEFFFF mov     [ebp+FFFFFFEC0], edx
00472E67  8995C4FEFFFF mov     [ebp+FFFFFFEC4], edx
00472E6D  8995C8FEFFFF mov     [ebp+FFFFFFEC8], edx
00472E73  8995CCFEFFFF mov     [ebp+FFFFFFECC], edx
00472E79  8995D0FEFFFF mov     [ebp+FFFFFFED0], edx
00472E7F  8955FC        mov     [ebp-04], edx
00472E82  8DB5D4FEFFFF lea     esi, [ebp+FFFFFFED4]
00472E88  33C0        xor     eax, eax
00472E8A  55          push   ebp

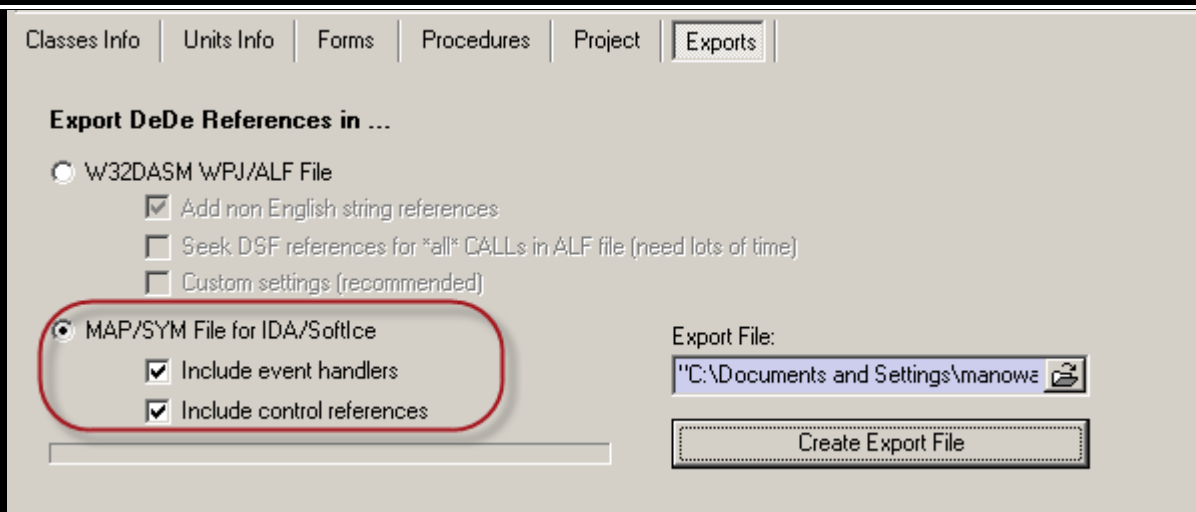
* Possible String Reference to: 'éÀÛÿèÀ^(<À]Ã'
|
00472E8B  6854304700   push   $00473054

***** TRY
|
00472E90  64FF30        push   dword ptr fs:[eax]
00472E93  648920        mov     fs:[eax], esp
00472E96  C70628010000 mov     dword ptr [esi], $00000128
00472E9C  33D2        xor     edx, edx
00472E9E  B80F000000   mov     eax, $0000000F

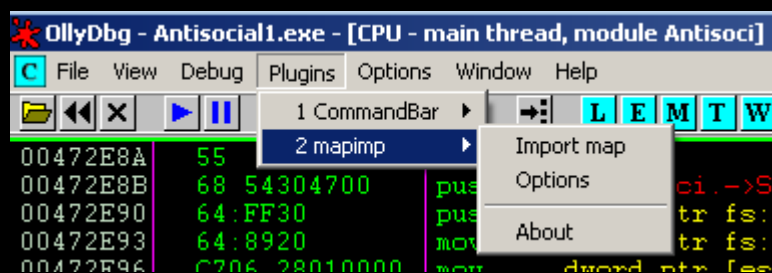
* Reference to: TlHelp32.CreateToolhelp32Snapshot(DWORD;DWORD):Windows.THandle;
|
00472EA3  E88CF9FFFF   call    00472834
00472EA8  8BD8        mov     ebx, eax
00472EAA  8BD6        mov     edx, esi
00472EAC  8BC3        mov     eax, ebx

```

Như các bạn thấy DeDe đã analyze và nhận diện các hàm rất tốt, điều mà OllyDBG chưa làm được. Đó là lý do vì sao tôi dùng DeDe. Giờ tại DeDe, chọn **Navigation > Close** để đóng cửa sổ Proc lại, sau đó chọn tab **Exports**, lựa chọn như sau và tiến hành Export ra file map:



Sau khi có được file map, đóng DeDe lại. Quay trở lại OllyDBG, dùng plugin mamimp để import map file vừa được tạo bởi DeDe.



Sau khi import xong ta có được như sau trong OllyDBG:

```

00472E9C  33D2          xor     edx,edx
00472E9E  B8 0F000000   mov     eax,0F
00472EA3  E8 8CF9FFFF   call    Antisoci.00472834      ->TlHelp32.CreateToolhelp32Snapshot
00472EA8  8BD8          mov     ebx,eax
00472EAA  8BD6          mov     edx,esi
00472EAC  8BC3          mov     eax,ebx
00472EAE  E8 A1F9FFFF   call    Antisoci.00472854      ->TlHelp32.CreateToolhelp32Snapshot
00472EB3  E9 68010000   jmp     Antisoci.00473020
00472EB8  8D85 D0FEFFFF lea     eax,dword ptr [ebp-130]
00472EBE  8D56 24       lea     edx,dword ptr [esi+24]
00472EC1  B9 04010000   mov     ecx,104
00472EC6  E8 9117F9FF   call    Antisoci.0040465C      ->System.@LStrFromArray(String;S
00472ECB  8B85 D0FEFFFF mov     eax,dword ptr [ebp-130]
00472ED1  8D55 FC       lea     edx,dword ptr [ebp-4]
00472ED4  E8 AB5BF9FF   call    Antisoci.00408A84      ->SysUtils.ExtractFileName(AnsiS
00472ED9  8D95 CCFEFFFF lea     edx,dword ptr [ebp-134]
00472EDF  B8 6C304700   mov     eax,Antisoci.0047306C  ASCII "FYF/SFEBPMQG"
00472EE4  E8 BBFEFFFF   call    Antisoci.00472DA4
00472EE9  8B95 CCFEFFFF mov     edx,dword ptr [ebp-134]
00472EEF  8B45 FC       mov     eax,dword ptr [ebp-4]
00472EF2  E8 F918F9FF   call    Antisoci.004047F0      ->System.@LStrCmp;
00472EF7  75 15         jnz     short Antisoci.00472F0E
00472EF9  6A 00         push    0

```

Thông tin như vậy là OK quá rồi, không còn đòi hỏi gì hơn. Như các bạn thấy, tại 00472EA3 là lời gọi tới hàm `CreateToolhelp32Snapshot`. Hàm này thì ở bài 21 các bạn đã biết rồi, nó được sử dụng để chụp nhanh thông tin của các Process, nó nhận hai tham số truyền vào là `dwFlags` và `th32ProcessID`. Sau khi thực hiện, hàm sẽ trả về một handle (gọi là handle của snapshot), mà handle này sẽ được sử dụng bởi các hàm khác. Tiếp theo bên dưới ta sẽ thấy lệnh `call Antisoci.00472DA4` sẽ làm gì đó với chuỗi

ASCII "FYF/SFEBPMQG", sau đó tại 00472EF2 có sử dụng hàm so sánh 2 chuỗi, nếu hai chuỗi giống nhau thì sẽ gọi hàm `TerminateProcess`. Giờ nhấn F8 để trace tới đoạn code có chứa chuỗi trên và trace qua lệnh `call Antisoci.00472DA4`. Quan sát trên cửa sổ Info/Stack ta sẽ thấy:

```
00472E3F: E8 D52FE9FF call Antisoci.00472DA4
Stack ss:[0012FE70]=00901C40, (ASCII "FPLOADER.EXE")
edx=0047306C (Antisoci.0047306C), ASCII "FYF/SFEBPMQG"
```

0012FE48	0012FFB4	Pointer to next SEH record
0012FE4C	00473054	SE handler
0012FE50	0012FFA4	
0012FE54	FFFFFFFF	
0012FE58	00478D7C	Antisoci.00478D7C
0012FE5C	00000000	
0012FE60	00000000	
0012FE64	00000000	
0012FE68	00000000	
0012FE6C	00000000	
0012FE70	00901C40	ASCII "FPLOADER.EXE"
0012FE74	00901C8C	ASCII "[System Process]"

Như vậy, cái chuỗi ASCII "FYF/SFEBPMQG" sẽ được giải mã thành ASCII "FPLOADER.EXE". Trace tiếp qua hai lệnh `mov` sẽ tới lệnh 00472EF2 > E8 F918F9FF `call Antisoci.004047F0; ->System.@LStrCmp;`, nó sẽ so sánh hai chuỗi:

```
Registers (FPU)
EAX 00902380 ASCII "[System Process]"
ECX FFFFFFFE
EDX 00901C40 ASCII "FPLOADER.EXE"
EBX 00000054
ESP 0012FE48
EBP 0012FFA4
ESI 0012FE78
EDI 7C910738 ntdll.7C910738
EIP 00472EF2 <Antisoci.->System.@LStrCmp;>
```

Hai chuỗi này khác nhau nên ta sẽ nhảy qua lời gọi hàm `TerminateProcess`. Trace tiếp tới chuỗi ASCII "FYF/HCEZMMP", chuỗi này sẽ được giải mã thành:

```
00472E3F: E8 D52FE9FF call Antisoci.00472DA4
Stack ss:[0012FE6C]=00901D90, (ASCII "OLLYDBG.EXE")
edx=00473084 (Antisoci.00473084), ASCII "FYF/HCEZMMP"
```

Sau đó lại được đem đi so sánh:

```
Registers (FPU)
EAX 00902380 ASCII "[System Process]"
ECX FFFFFFFE
EDX 00901D90 ASCII "OLLYDBG.EXE"
EBX 00000054
ESP 0012FE48
```

Tới đây, ta hoàn toàn có thể kết luận được rằng, crackme sẽ lấy lần lượt tên các Process trên hệ thống, sau đó giải mã các chuỗi đã hard code sẵn thành tên các Process

ví dụ **FPLOADER.EXE**, **"OLLYDBG.EXE"** v.v.v..Đem so sánh chúng với nhau, trong trường hợp của chúng ta, nếu tên Process là OLLYDBG.EXE thì sẽ bị Terminate luôn. OK, tiếp tục trace để tìm nốt thông tin về các chuỗi encrypt tiếp theo bên dưới, ta có được đầy đủ các chuỗi được giải mã thành tên các Process như sau:

```

0012FE48 0012FFB4 Pointer to next SEH record
0012FE4C 00473054 SE handler
0012FE50 0012FFA4
0012FE54 FFFFFFFF
0012FE58 00478D7C Antisoci.00478D7C
0012FE5C 0090246C ASCII "W32DSM89.EXE"
0012FE60 00902434 ASCII "DEDE.EXE"
0012FE64 009023F8 ASCII "LOADER32.EXE"
0012FE68 009023C0 ASCII "TRW2000.EXE"
0012FE6C 00901D90 ASCII "OLLYDBG.EXE"
0012FE70 00901C40 ASCII "FPLOADER.EXE"

```

Chà, có cả DeDe và W32DSM89 kìa. Trace tiếp một vài lượt nữa để khẳng định các suy luận trên là đúng, tôi có được như sau:

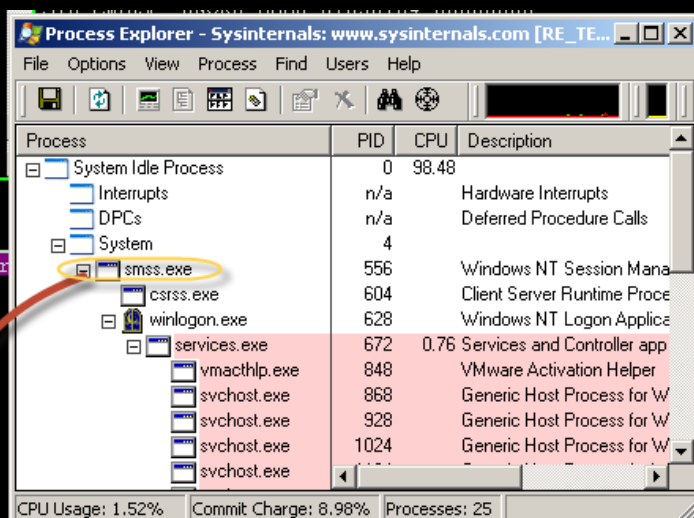
/HCEZMMP"

IStrCmp;

```

0012FE48 0012FFB4 Pointer to next SEH r
0012FE4C 00473054 SE handler
0012FE50 0012FFA4
0012FE54 FFFFFFFF
0012FE58 00478D7C Antisoci.00478D7C
0012FE5C 009025DC ASCII "W32DSM89.EXE"
0012FE60 009025A4 ASCII "DEDE.EXE"
0012FE64 00902568 ASCII "LOADER32.EXE"
0012FE68 00902530 ASCII "TRW2000.EXE"
0012FE6C 009024F8 ASCII "OLLYDBG.EXE"
0012FE70 009024BC ASCII "FPLOADER.EXE"
0012FE74 00901D90 ASCII "smss.exe"

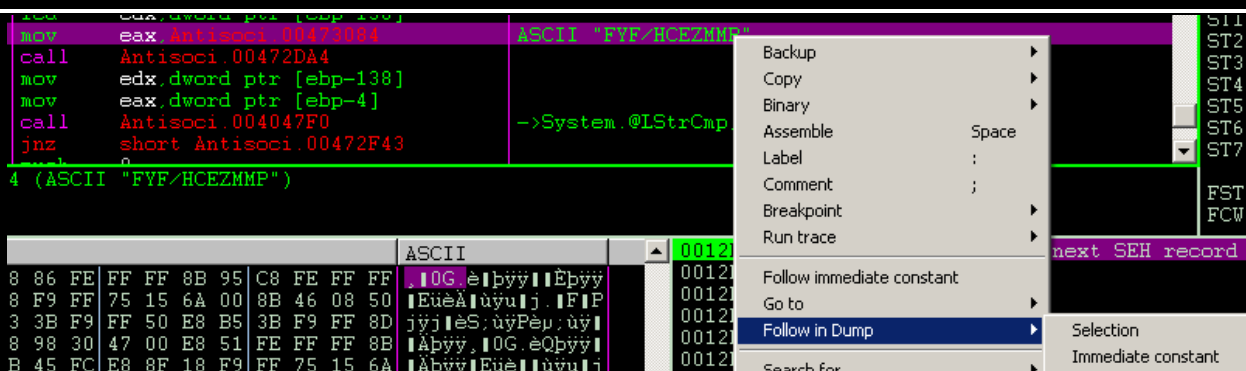
```



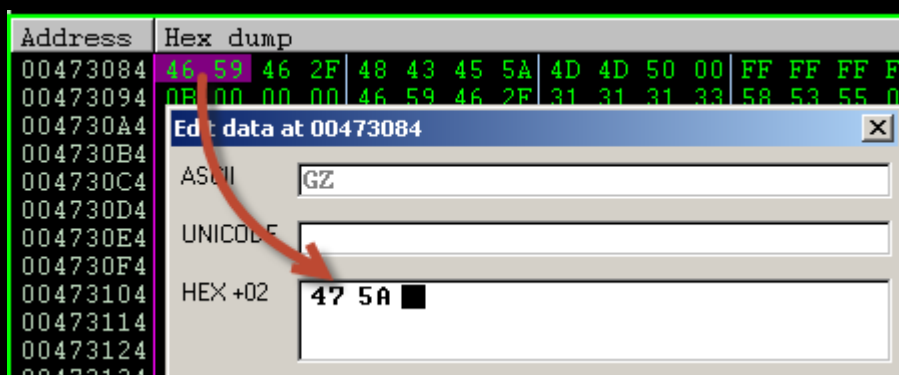
Vậy để tránh bị Terminate thì làm thế nào? Có hai cách:

- Cách thứ nhất: Đổi tên của OLLYDBG thành Blah Blah (bất cứ tên gì ta muốn)
- Cách thứ hai: Patch cái chuỗi sẽ được giải mã thành OLLYDBG.EXE sao cho sau khi giải mã nó sẽ thành tên khác mà không phải là OLLYDBG.EXE.

Ở đây tôi chọn cách 2, ta sẽ patch chuỗi **'FYF/HCEZMMP'**, thay bằng gì tùy bạn. Trước tiên ta tìm tới vùng chứa chuỗi đó đã, chuột phải tại chuỗi và chọn **Follow in Dump > Immediate constant:**



Ví dụ tôi Patch thành như sau:



Sau khi sửa xong ta xem nó sẽ được decrypt thành gì:

```

Stack ss:[0012FE6C]=009027E0, (ASCII "OLLYDBG.EYF")
edx=00473084 (Antisoci.00473084), ASCII "GZF/HCEZMMP"

```

Khà khà, đã chuyển thành 'OLLYDBG.EYF', vậy tạm coi như là xong. Giờ ta disable hết các BP đã đặt, sau đó nhấn F9 để run thử xem thế nào. Ta thấy crackme hiện lên chớp nhoáng và một thông báo exception lại xuất hiện:



Có exception, mà status bar vẫn báo là running, OllyDBG thì lại không bị Terminate. Vậy có nghĩa là đâu đó vẫn còn có trick để anti-Olly. Tiến hành Restart lại OllyDBG và thực hiện tới đoạn patch như trên, nhấn chuột phải chọn **Search for > all intermodular calls**, mục đích để tìm xem có hàm nào khác lạ nữa không. Ta thấy có hàm:

Address	Disassembly	Destination
0045456E	call Antisoci.004070D8	user32.PostMessageA
004532D7	call Antisoci.004070E0	user32.PostQuitMessage
00430E2C	call Antisoci.004070E8	user32.PtInRect

Hàm PostQuitMessage ta đã tìm hiểu trong bài 19 rồi, nhấp đúp vào ta sẽ tới đây:

004532CA	C3	retn	
004532CB	90	nop	
004532CC	E8 FB7FFBFF	call	Antisoci.0040B2CC
004532D1	84C0	test	al, al
004532D3	74 07	je	short Antisoci.004532DC
004532D5	6A 00	push	0
004532D7	E8 043EFBFF	call	Antisoci.004070E0
004532DC	C3	retn	
004532DD	8D40 00	lea	eax, dword ptr [eax]
004532E0	53	push	ebx
004532E1	56	push	esi
004532E2	8BF2	mov	esi, edx
004532E4	8BD8	mov	ebx, eax
004532E6	E8 B53BFBFF	call	Antisoci.00406EA0
004532EB	85C0	test	eax, eax
004532ED	74 11	je	short Antisoci.004532F1
004532EF	6A 00	push	0
004532F1	6A 00	push	0
004532F3	6A 1F	push	1F
004532F5	E8 A63BFBFF	call	Antisoci.00406EA0
004532FA	50	push	eax
004532FB	E8 403EFBFF	call	Antisoci.00407140
00453300	E8 AFF4FAFF	call	Antisoci.004027B4

Dừng ở đây và quan sát chút, ta đang đứng tại địa chỉ có lời gọi tới hàm `PostQuitMessage`, như vậy ở đâu đó trong code của crackme sẽ có lời gọi tới khúc này. Dưới hàm là lệnh `retn`, phía trên một chút cũng có lệnh `retn`, vậy làm sao tìm được đoạn code nào gọi tới đoạn code trên. Để ý một chút là phía trên lời gọi tới hàm `PostQuitMessage` có hàm `call` tại địa chỉ `004532CC`, theo sau là lệnh `test al, al` sẽ kiểm tra xem `al` có bằng không hay không? Nếu bằng `0x0` thì cờ `Z` sẽ được bật và lệnh nhảy `JE` sẽ nhảy qua lời gọi hàm `PostQuitMessage`, còn không sẽ thực hiện hàm. Như vậy, ta sẽ tìm đoạn code gọi tới địa chỉ `004532CC` như sau:

004532CC	E8 FB7FFBFF	call	Antisoci.0040B2CC	
004532D1	84C0	test	al, al	
004532D3	74 07	je	short Antisoci.004532DC	
004532D5	6A 00	push	0	
004532D7	E8 043EFBFF	call	Antisoci.004070E0	
004532DC	C3	retn		
004532DD	8D40 00	lea	eax, dword ptr [eax]	
004532E0	53	push	ebx	
004532E1	56	push	esi	
004532E2	8BF2	mov	esi, edx	
004532E4	8BD8	mov	ebx, eax	
004532E6	E8 B53BFBFF	call	Antisoci.00406EA0	
004532EB	85C0	test	eax, eax	
004532ED	74 11	je	short Antisoci.004532F1	
004532EF	6A 00	push	0	
004532F1	6A 00	push	0	
004532F3	6A 1F	push	1F	
004532F5	E8 A63BFBFF	call	Antisoci.00406EA0	
004532FA	50	push	eax	
004532FB	E8 403EFBFF	call	Antisoci.00407140	
00453300	E8 AFF4FAFF	call	Antisoci.004027B4	

Kết quả có được:

Address	Disassembly	Comment
0044FC33	call Antisoci.004532CC	
004532CC	call Antisoci.0040B2CC	(Initial CPU selection)
0047324E	call Antisoci.004532CC	->Forms.TApplication.Terminate(TApplication);
004732A7	call Antisoci.004532CC	->Forms.TApplication.Terminate(TApplication);
00473300	call Antisoci.004532CC	->Forms.TApplication.Terminate(TApplication);
00473359	call Antisoci.004532CC	->Forms.TApplication.Terminate(TApplication);
004733B2	call Antisoci.004532CC	->Forms.TApplication.Terminate(TApplication);
004742C3	call Antisoci.004532CC	->Forms.TApplication.Terminate(TApplication);
004743A7	call Antisoci.004532CC	->Forms.TApplication.Terminate(TApplication);
004743FF	call Antisoci.004532CC	->Forms.TApplication.Terminate(TApplication);
00474457	call Antisoci.004532CC	->Forms.TApplication.Terminate(TApplication);

Nhấp đúp vào địa chỉ `0047324E`, ta sẽ tới đoạn code sau:

```

00473229 8D95 F4FEFFFF lea     edx,dword ptr [ebp-10C]
0047322F B8 FC334700 mov     eax, Antisoci.004733FC      ASCII "y:!txpeojX!spg!1113XSU"
00473234 E8 6BFBEFFF call    Antisoci.00472DA4
00473239 8B95 F4FEFFFF mov     edx,dword ptr [ebp-10C]
0047323F 58          pop     eax
00473240 E8 AB15F9FF call    Antisoci.004047F0      ->System.@ILStrCmp;
00473245 75 0C          jnz     short Antisoci.00473253
00473247 A1 DC6E4700 mov     eax,dword ptr [476EDC]
0047324C 8B00          mov     eax,dword ptr [eax]
0047324E E8 7900FEFF call    Antisoci.004533CC      ->Forms.TApplication.Terminate(TApplic
00473253 8D85 ECFEFFFF lea     eax,dword ptr [ebp-114]
00473259 8BD6          mov     edx,esi

```

Ta thấy đoạn code này cũng tương tự như đoạn code mà ta đã gặp ở trên. Cuộn chuột lên trên một chút ta sẽ thấy được cơ chế lấy thông tin để so sánh như sau:

```

004731CB 64 8920          mov     dword ptr fs:[eax],esp
004731CE E8 153DF9FF call    Antisoci.00406EE8      jmp to user32.GetDesktopWindow
004731D3 6A 05          push    5
004731D5 50          push    eax
004731D6 E8 ED3DF9FF call    Antisoci.00406FC8      jmp to user32.GetWindow
004731DB 8BD8          mov     ebx,eax
004731DD 85DB          test    ebx,ebx
004731DF 0F84 E4010000 je      Antisoci.004733C9
004731E5 68 FF000000 push    0FF
004731EA 56          push    esi
004731EB 53          push    ebx
004731EC E8 FF3DF9FF call    Antisoci.00406FF8      jmp to user32.GetWindowTextA
004731F1 8D85 FCFEFFFF lea     eax,dword ptr [ebp-104]
004731F7 8BD6          mov     edx,esi
004731F9 B9 00010000 mov     ecx,100
004731FE E8 5914F9FF call    Antisoci.0040465C      ->System.@ILStrFromArray(String:String;
00473203 83BD FCFEFFFF cmov     dword ptr [ebp-104],0

```

Như vậy, ở đây đoạn code này khác đoạn code trên ở chỗ nó không lấy thông tin về các process nữa mà lấy thông tin về window's title bar, tức là lấy tiêu đề của cửa sổ.

The **GetWindowText** function copies the text of the specified window's title bar (if it has one) into a buffer. If the specified window is a control, the text of the control is copied.

```

int GetWindowText(
    HWND hWnd,           // handle of window or control with text
    LPTSTR lpString,     // address of buffer for text
    int nMaxCount        // maximum number of characters to copy
);

```

Giờ ta đặt BP tại các đoạn encoded string xem nó giải mã ra chuỗi gì:

```

00473228 50          push    eax
00473229 8D95 F4FEFFFF lea     edx,dword ptr [ebp-10C]
0047322F B8 FC334700 mov     eax, Antisoci.004733FC      ASCII "y:!txpeojX!spg!1113XSU"
00473234 E8 6BFBEFFF call    Antisoci.00472DA4
00473239 8B95 F4FEFFFF mov     edx,dword ptr [ebp-10C]
0047323F 58          pop     eax
00473240 E8 AB15F9FF call    Antisoci.004047F0      ->System.@ILStrCmp;
00473245 75 0C          jnz     short Antisoci.00473253
00473247 A1 DC6E4700 mov     eax,dword ptr [476EDC]
0047324C 8B00          mov     eax,dword ptr [eax]
0047324E E8 7900FEFF call    Antisoci.004533CC      ->Forms.TApplication.Terminate(TApplic

```

Nhấn F9 để run, ta sẽ break tại chỗ đặt bp ở trên. Nhấn F8 để trace qua, ta thu được các thông tin sau:

```

Stack ss:[0012FE98]=00902440, (ASCII "TRW2000 for Windows 9x")
edx=004733FC (Antisoci.004733FC), ASCII "y:!txpeojX!spg!1113XSU"

```

```

Stack ss:[0012FE8C]=009024D4, (ASCII "Symbol Loader")
edx=0047341C (Antisoci.0047341C), ASCII "sfebpM!mpcnzT"

```

Address | Hex dump | ASCII


```

00473300 E8 C7FFDFF call Antisoci.004532CC
00473305 0000 00000000 lea eax, dword ptr [ebp+12C1]
Stack ss:[0012FE80]=00902560, (ASCII "OllYDbg")
edx=00473434 (Antisoci.00473434), ASCII "hcEzmmP"

```

Ồ tới đoạn string (ASCII "hcEzmmP") thì hàm call đã decrypt thành ASCII "OllYDbg". Như vậy ta cần phải patch string này. Nhưng trước khi restart và thực hiện lại, ta tìm nốt các string phía dưới xem là những gì:

```

00473305 0000 00000000 lea eax, dword ptr [ebp+12C1]
Stack ss:[0012FE74]=009025F8, (ASCII "DeDe")
edx=00473444 (Antisoci.00473444), ASCII "fEfE"

0047330C 57 push ebx
0047330D 58 pop ebx
Stack ss:[0012FE68]=00902690, (ASCII "URSoft")
edx=00473454 (Antisoci.00473454), ASCII "ugpTSW"

```

Thông tin đầy đủ có được trên Stack như sau:

```

0012FE58 004733E7 SE handler
0012FE5C 0012FFA4
0012FE60 FFFFFFFF
0012FE64 00478D7C Antisoci.00478D7C
0012FE68 00902690 ASCII "URSoft"
0012FE6C 0090260C ASCII "OllYDbg - Antisocial1.exe - [CPU - main thread, module Antisoci]"
0012FE70 0090265C ASCII "OllYDb"
0012FE74 009025F8 ASCII "DeDe"
0012FE78 00902574 ASCII "OllYDbg - Antisocial1.exe - [CPU - main thread, module Antisoci]"
0012FE7C 009025C4 ASCII "OllY"
0012FE80 00902560 ASCII "OllYDbg"
0012FE84 009024F0 ASCII "OllYDbg - Antisocial1.exe - [CPU - main thread, module Antisoci]"
0012FE88 00900C24 ASCII "OllYDbg"
0012FE8C 009024D4 ASCII "Symbol Loader"
0012FE90 00902464 ASCII "OllYDbg - Antisocial1.exe - [CPU - main thread, module Antisoci]"
0012FE94 00901C40 ASCII "ule Antisoci]"
0012FE98 00902440 ASCII "TRW2000 for Windows 9x"
0012FE9C 009023D0 ASCII "OllYDbg - Antisocial1.exe - [CPU - main thread, module Antisoci]"
0012FEA0 00902380 ASCII "OllYDbg - Antisocial1.exe - [CPU - main thread, module Antisoci]"
0012FEA4 786C6C6F

```


OK tạm thời ta đã có thêm thông tin là cần phải patch thêm một string nữa là ASCII "hcEzmmP". Giờ restart lại Olly và thực hiện các bước sau:

- Patch popad thành pushad
- Tới OEP, tìm toàn bộ int 68 và NOP.
- Patch chuỗi ASCII "FYF/HCEZMMP"
- Patch chuỗi ASCII "hcEzmmP"

Sau khi thực hiện toàn bộ các bước trên xong, nhấn F9 để kiểm tra xem thế nào:

004731E5	68 FF000000	push	0FF	
004731EA	56	push	esi	
004731EB	53	push	ebx	
004731EC	E8 FF3DF9FF	call	Antisoci.00406E90	jmp to user32.GetAsyncKeyState
004731F1	8D85 FCFEFFFF	lea	ea	
004731F7	8BD6	mov	ed	
004731F9	B9 00010000	mov	ec	
004731FE	E8 5914F9FF	call	An	u.@LStrFromI
00473203	83BD FCFEFFFF	cmp	dw	
0047320A	0F84 A7010000	je	An	
00473210	8D85 F8FEFFFF	lea	ea	
00473216	8BD6	mov	ed	
00473218	B9 00010000	mov	ec	
0047321D	E8 3A14F9FF	call	An	u.@LStrFromI
00473222	8B85 F8FEFFFF	mov	ea	
00473228	50	push	ea	
00473229	8D95 F4FEFFFF	lea	ed	
0047322F	B8 FC334700	mov	ea	
00473234	E8 6BF8FFFF	call	An	z: !txpeojXls
00473239	8B95 F4FEFFFF	mov	ed	
0047323F	58	pop	ea	
00473240	E8 AB15F9FF	call	An	u.@LStrCmp;
00473245	75 0C	jnz	sh	
00473247	A1 DC6E4700	mov	ea	TApplicatio
0047324C	8B00	mov	ea	
0047324E	E8 7900FEFF	call	An	
00473253	8D85 ECFEFFFF	lea	ea	
00473259	8BD6	mov	ed	

esi=0012FEA4, (ASCII "OllyDbg - Anti



Address	Hex dump	ASCII
		0012F

Phù run mượt mà rồi nhé!!! Đang sướng âm ỉ, tự nhiên bụp cái lại nhận được thêm cái exception nữa, nản quá ☹.

Access violation when reading [009029D4] - use Shift+F7/F8/F9 to pass exception to program

Như vậy là vẫn còn thằng nào đó gọi tới `PostQuitMessage` mà ta chưa tìm hết. Lại Restart lại Olly và làm lại các bước trên, quay lại cửa sổ **References** (nơi có các thông tin về các đoạn code sẽ gọi tới hàm `PostQuitMessage`) tìm thêm mấy hàm call ở dưới, ta tới đoạn code sau:

004742AF	6A 11	push	11	
004742B1	E8 E22BF9FF	call	Antisoci.00406E98	jmp to user32.GetAsyncKeyState
004742B6	66:83F8 01	cmp	ax, 1	
004742BA	75 0C	jnz	short <Antisoci.->System.Random	
004742BC	A1 DC6E4700	mov	eax, dword ptr [476EDC]	
004742C1	8B00	mov	eax, dword ptr [eax]	
004742C3	E8 04F0FDFD	call	Antisoci.004532CC	->Forms.TApplication.Terminate(TApplica
004742C8	E8 9BE7F8FF	call	Antisoci.00402A68	->System.Randomize;
004742CD	B8 88130000	mov	eax, 1388	
004742D2	E8 C9ECF8FF	call	Antisoci.00402FA0	

Đặt một BP tại lệnh `push 11`, nhấn F9 để run, ta dừng lại tại BP:

004742AF	6A 11	push	11	
004742B1	E8 E22BF9FF	call	Antisoci.00406E98	jmp to user32.GetAsyncKeyState
004742B6	66:83F8 01	cmp	ax, 1	
004742BA	75 0C	jnz	short <Antisoci.->System.Randomize;	
004742BC	A1 DC6E4700	mov	eax, dword ptr [476EDC]	
004742C1	8B00	mov	eax, dword ptr [eax]	
004742C3	E8 04F0FDEF	call	Antisoci.004532CC	->Forms.TApplication.Terminate(TApplica
004742C8	E8 9BE7F8FF	call	Antisoci.00402A68	->System.Randomize;
004742CD	B8 88130000	mov	eax, 1388	
004742D2	E8 C9ECF8FF	call	Antisoci.00402FA0	
004742D7	8ED0	mov	edx, eax	
004742D9	8B83 60030000	mov	eax, dword ptr [ebx+360]	*Time
004742DF	E8 2882FBFF	call	Antisoci.0042C50C	->Ex!
004742E4	5B	pop	ebx	
004742E5	C3	retn		
004742E6	8BC0	mov	eax, eax	
004742E8	BA D0030000	mov	edx, 3D0	
004742ED	33C0	xor	eax, eax	
004742EF	E8 68F1FFFF	call	Antisoci.0047345C	
004742F4	3D A8D88200	cmp	eax, 82D8A8	



Thử tra thông tin về cái hàm lạ lạ với tên `user32.GetAsyncKeyState` xem nó có tác dụng gì:

The `GetAsyncKeyState` function determines whether a key is up or down at the time the function is called, and whether the key was pressed after a previous call to `GetAsyncKeyState`.

```
SHORT GetAsyncKeyState(
    int vKey // virtual-key code
);
```

Parameters

vKey
Specifies one of 256 possible virtual-key codes.

Windows NT: You can use left- and right-distinguishing constants to specify certain keys. See the **Remarks** section for further information.

Windows 95: Windows 95 does not support the left- and right-distinguishing constants available on Windows NT.

Return Values

If the function succeeds, the return value specifies whether the key was pressed since the last call to `GetAsyncKeyState`, and whether the key is currently up or down. If the most significant bit is set, the key is down, and if the least significant bit is set, the key was pressed after the previous call to `GetAsyncKeyState`. The return value is zero if a window in another thread or process currently has the keyboard focus.

Dịch nôm na như sau, `GetAsyncKeyState` là hàm dùng để xác định xem một phím có được nhấn hay là không tại một thời điểm xác định khi hàm được gọi. Tham số truyền vào là `vKey`, mà `vKey` ở đây có giá trị `0x11`, thử Google một lúc thì biết được `VK_CONTROL = 0x11`. Điều này có nghĩa là nếu ta nhấn phím CTRL thì crackme sẽ detect được và quit luôn. Như vậy, để bypass được trick này ta patch lệnh nhảy `jnz` thành `jmp`:

004742AF	6A 11	push	11	
004742B1	E8 E22BF9FF	call	Antisoci.00406E98	jmp to user32.GetAsyncKeyState
004742B6	66:83F8 01	cmp	ax, 1	
004742BA	EB 0C	jmp	short <Antisoci.->System.Randomize;>	
004742BC	A1 DC6E4700	mov	eax, dword ptr [476EDC]	
004742C1	8B00	mov	eax, dword ptr [eax]	
004742C3	E8 04F0FDEF	call	Antisoci.004532CC	->Forms.TApplication.Terminate(TApplica
004742C8	E8 9BE7F8FF	call	Antisoci.00402A68	->System.Randomize;
004742CD	B8 88130000	mov	eax, 1388	

Tìm nốt các đoạn code tương tự ở dưới và patch tiếp như sau:

0047438C	8EC3	mov	eax, ebx	
0047438E	E8 55EDFFFF	call	<Antisoci.<>	->
00474393	6A 11	push	11	
00474395	E8 FE2AF9FF	call	Antisoci.00406E98	jmp to user32.GetAsyncKeyState
0047439A	66:83F8 01	cmp	ax, 1	
0047439E	EB 0C	jmp	short <Antisoci.->System.Randomize;>	
004743A0	A1 DC6E4700	mov	eax, dword ptr [476EDC]	
004743A5	8B00	mov	eax, dword ptr [eax]	
004743A7	E8 20EFFDEF	call	Antisoci.004532CC	->Forms.TApplication.Terminate(TApplica
004743AC	E8 B7E6F8FF	call	Antisoci.00402A68	->System.Randomize;
004743B1	B8 88130000	mov	eax, 1388	
004743B6	E8 E5EBF8FF	call	Antisoci.00402FA0	

004743DD	8BC3	mov	eax,ebx	
004743DF	E8 C4EDFFFF	call	<Antisoci.<>	->
004743E4	8BC3	mov	eax,ebx	
004743E6	E8 FDECFDFF	call	<Antisoci.<>	->
004743EB	6A 11	push	11	
004743ED	E8 A62AF9FF	call	Antisoci.00406E98	jmp to user32.GetAsyncKeyState
004743F2	66:83F8 01	cmp	ax,1	
004743F6	EB 0C	jmp	short <Antisoci.->System.Random	
004743F8	A1 DC6E4700	mov	eax,dword ptr [476EDC]	
004743FD	8B00	mov	eax,dword ptr [eax]	
004743FF	E8 C8EEFDFF	call	Antisoci.00402A68	->Form1.TApplication.Terminate(TApplic
00474404	E8 5FE6F8FF	call	Antisoci.00402A68	->System.Randomize;
00474435	8BC3	mov	eax,ebx	
00474437	E8 6CEDFFFF	call	<Antisoci.<>	->
0047443C	8BC3	mov	eax,ebx	
0047443E	E8 A5ECFFFF	call	<Antisoci.<>	->
00474443	6A 11	push	11	
00474445	E8 4E2AF9FF	call	Antisoci.00406E98	jmp to user32.GetAsyncKeyState
0047444A	66:83F8 01	cmp	ax,1	
0047444E	EB 0C	jmp	short <Antisoci.->System.Random	
00474450	A1 DC6E4700	mov	eax,dword ptr [476EDC]	
00474455	8B00	mov	eax,dword ptr [eax]	
00474457	E8 70EEFDFF	call	Antisoci.00402A68	->Form1.TApplication.Terminate(TApplic
0047445C	E8 07E6F8FF	call	Antisoci.00402A68	->System.Randomize;
00474461	B8 88130000	mov	eax,1388	

Sau khi patch xong, disable BP đi và nhấn F9 để run crackme. Crackme thực thi OK, nhưng đánh giá lại thì cái trick trên chẳng liên quan gì tới việc bị dính exception ở trên, vì thực tế là đoạn trick này chỉ để trap khi ta nhấn phím Ctrl mà thôi. Lúc crackme đang run mà nhấn Ctrl bậy bạ là bay luôn.



III. Kết luận

Toàn bộ bài 24 đến đây là kết thúc, nhìn lại thấy mình viết vừa dài vừa lủng củng, có chỗ nào thiếu sót thì các bạn bỏ quá cho. Tổng kết lại bài 24 ta sẽ thấy các kiểu Anti-OllyDBG mà crackme này áp dụng:

- Sử dụng kĩ thuật **"Self-Modifying-Code"**, kết hợp với phá vỡ cấu trúc lệnh `pushad/popad` làm cho vùng Stack của Crackme bị thay đổi, gây ra exception.
- Sử dụng một trick để anti-SoftIce đem áp dụng cho OllyDBG là `int 68` để tạo ra exception.
- Sử dụng hàm `CreateToolhelp32Snapshot` và các API liên quan để lấy thông tin về các Process đang chạy trên hệ thống, sau đó đem so sánh với các chuỗi encrypt được decrypt về dạng clear text, nếu trình Debug/ Disassembler trùng tên với danh sách các black list thì sẽ gọi hàm `TerminateProcess`.
- Sử dụng hàm `GetWindowText` để lấy thông tin về tiêu đề hay tên của trình Debug, sau đó cũng đem so sánh với một danh sách black list, nếu giống gọi hàm `PostQuitMessage`.
- Sử dụng hàm `GetAsyncKeyState` để phát hiện xem có nhấn phím Ctrl hay không, nếu có cũng sẽ gọi hàm `PostQuitMessage`.

Cảm ơn các bạn đã dành thời gian để theo dõi. Hẹn gặp lại ở phần 25!

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, QHQCkrker, 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, dzungltn, takada, hurt_heart, haule_nth, hytkl,

moth, XIANUA, nhc1987, 0xdie, Unregistered!, akira, mrangle v.v.. các bạn đã đóng góp rất nhiều cho REA. Hi vọng các bạn sẽ tiếp tục phát huy ☺

I want to thank **Teddy 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](mailto:kienbigmummy[at]gmail.com)