

2009

[Cracking with OllyDbg]

Based on OllyDbg tuts of Ricardo Narvaja (CrackLatinos Team)



www.reasonline.net

kienmanowar



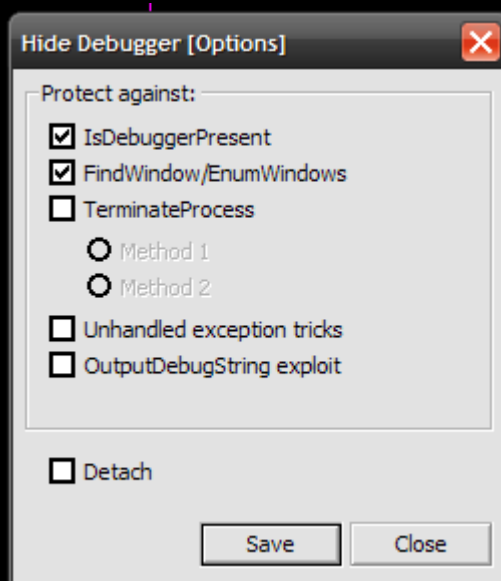
27/01/2010

Mục Lục

I. Giới thiệu chung	2
II. Phân tích và xử lý target	3
1. Phân tích Sphynx.exe	3
III. Kết luận.....	14

I. Giới thiệu chung

Tiếp tục với chủ đề Anti-Debug, ở phần 22 này chúng ta sẽ tìm hiểu thêm hai "thủ thuật" mới, thường được áp dụng cùng nhau hoặc riêng lẻ. Crackme để chúng ta nghiên cứu trong phần này là [Sphynx.exe](#), của tác giả có nick name là [d@b](#). Mặc định tôi xem như các bạn đã hiểu hết những gì tôi viết ở các phần trước, trong phần này chúng ta sẽ sử dụng bản Olly đã được chỉnh sửa bởi chương trình repair0.6 mà tôi giới thiệu ở bài 21. Trên máy của tôi bản OllyDbg gốc được repair0.6 sửa lại và đặt tên là [Ltp10.exe](#), thêm vào đó plugin HideDebugger được cấu hình như sau :



Các bạn để ý ở phần cấu hình của plugin HideDebugger, ta thấy có option cho phép bypass *Unhandled exception tricks*, đó chính là thủ thuật mà ta nghiên cứu trong bài này. Ngoài ra, thủ thuật này kết hợp với một thủ thuật khác mà ta cũng nghiên cứu luôn, đó sử dụng API `ZwQueryInformationProcess` để phát hiện ra OllyDBG. Now let's go.....☺

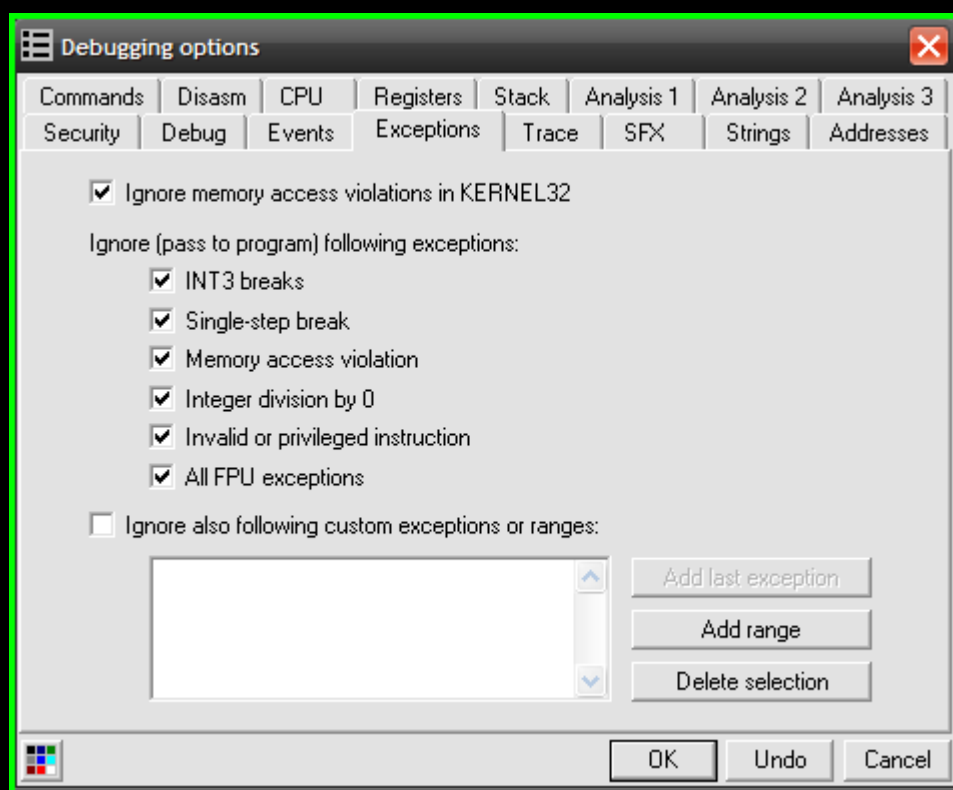
II. Phân tích và xử lý target

1. Phân tích Sphynx.exe

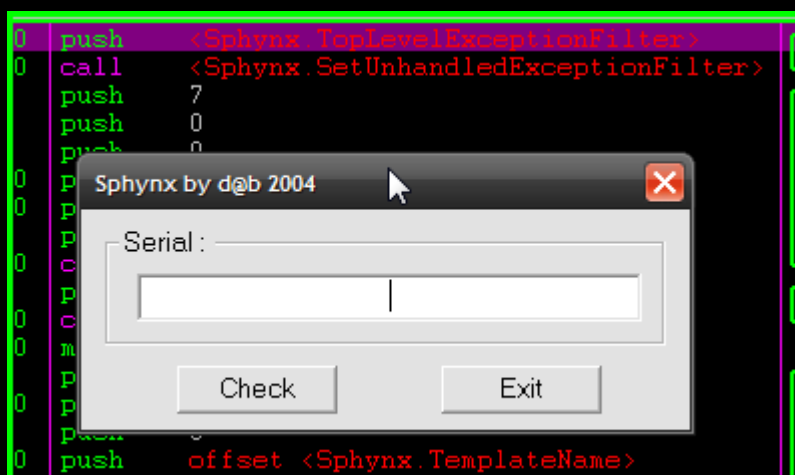
Mở Olly lên và load target vào, chúng ta dừng lại tại EP của chương trình :

00401000	68 08114000	push	<Sphynx.TopLevelExceptionHandler>	[start
00401005	E8 AA010000	call	<Sphynx.SetUnhandledExceptionHandler>	SetUnhandledExceptionHandler
0040100A	6A 07	push	7	IsShown = 7
0040100C	6A 00	push	0	DefDir = NULL
0040100E	6A 00	push	0	Parameters = NULL
00401010	68 39304000	push	offset <Sphynx.File>	FileName = ".\Readme.hta"
00401015	68 46304000	push	offset <Sphynx.Operation>	Operation = "Open"
0040101A	6A 00	push	0	hWnd = NULL
0040101C	E8 99010000	call	<Sphynx.ShellExecuteA>	ShellExecuteA
00401021	6A 00	push	0	pModule = NULL
00401023	E8 80010000	call	<Sphynx.GetModuleHandleA>	GetModuleHandleA
00401028	A3 50304000	mov	dword ptr [<hInstance>],eax	
0040102D	6A 00	push	0	
0040102F	68 4C104000	push	<Sphynx.DialogFunc>	
00401034	6A 00	push	0	
00401036	68 00304000	push	offset <Sphynx.TemplateName>	
0040103B	FF35 50304000	push	dword ptr [<hInstance>]	
00401041	E8 44010000	call	<Sphynx.ShowDialogBoxParamA>	
00401046	50	push	eax	
00401047	E8 56010000	call	<Sphynx.ExitProcess>	
0040104C	55	push	ebp	
0040104F	EB	jmp	short 0040104C	

Ta cấu hình lại Olly như sau :



Cấu hình xong, nhấn F9 để run thử crackme xem thế nào :



Ồ, như các bạn thấy crackme này chạy bình thường chứ có thấy nó chơi Anti-Debug gì đâu nhỉ. Khả khả, khoan mừng vội... tôi thử nhập đại 1 kí tự trong textbox Serial và nhấn nút Check. Olly sẽ dừng lại tại đây :

004010CC	8915 0411400	mov	dword ptr [< dword_401104 >] edx	
004010D2	81FA 7293010	cmp	edx,19372	
004010D8	74 09	je	short <Sphynx.loc_4010E3>	
004010DA	EB 22	jmp	short <Sphynx.loc_4010FE>	
004010DC	E8 BB000000	call	<Sphynx.DebugBreak>	[loc_4010DC
004010E1	EB 13	jmp	short <Sphynx.loc_4010F6>	

Hơi ngạc nhiên ở chỗ là ta có đặt BP nào đâu mà Olly lại break nhỉ? Để ý thông tin ở dưới, tôi nhận được như sau :



Như vậy là có "cái gì đó" đã cản trở quá trình debug của Olly! Giờ nếu ta nhấn F7/F8 hay F9 thì ngay lập tức Olly sẽ bị Terminate luôn. Ok, ta restart lại Olly và tìm kiếm xem danh sách các hàm APIs mà crackme này sử dụng :

Address	Section	Type	Name	Comment
00401000	.text	Export	<ModuleEntryPoint>	start
00402000	.rdata	Import	kernel32.GetModuleHandleA	
00402004	.rdata	Import	kernel32.SetUnhandledExceptionHandler	
00402008	.rdata	Import	kernel32.IsDebuggerPresent	
0040200C	.rdata	Import	kernel32.DebugBreak	
00402010	.rdata	Import	kernel32.ExitProcess	
00402018	.rdata	Import	shell32.ShellExecuteA	
00402020	.rdata	Import	user32.MessageBoxA	
00402024	.rdata	Import	user32.GetDlgItemTextA	
00402028	.rdata	Import	user32.DialogBoxParamA	
0040202C	.rdata	Import	user32.wsprintfA	

Để ý một chút ta thấy có hàm `SetUnhandledExceptionHandler`, mà trong phần cấu hình của plugin HideDebugger ta thấy có tùy chọn dùng để bypass *Unhandled exception tricks*. Như vậy, ta có thể khẳng định chắc chắn rằng hàm API `SetUnhandledExceptionHandler` có liên quan tới việc Anti-Debug.

Tuy nhiên, mục đích của ta là nghiên cứu cơ chế hoạt động của hàm API này xem nó làm gì, do đó ta không chọn tùy chọn ở plugin HideDebugger. Tìm kiếm thông tin về hàm `SetUnhandledExceptionFilter`, tôi có được như sau :

SetUnhandledExceptionFilter

Quick Info

The `SetUnhandledExceptionFilter` function lets an application supersede the top-level exception handler that Win32 places at the top of each thread and process.

After calling this function, if an exception occurs in a process that is not being debugged, and the exception makes it to the Win32 unhandled exception filter, that filter will call the exception filter function specified by the `lpTopLevelExceptionFilter` parameter.

```
LPTOP_LEVEL_EXCEPTION_FILTER SetUnhandledExceptionFilter(
    LPTOP_LEVEL_EXCEPTION_FILTER lpTopLevelExceptionFilter    // exception filter function
);
```

Đọc những thông tin giải thích về hàm và tham khảo tài liệu ta rút ra được chức năng của hàm `SetUnhandledExceptionFilter` :

Cho phép chương trình ngăn chặn việc xử lý lỗi ngoại lệ mặc định của hệ thống. Sau khi gọi hàm này, nếu có lỗi ngoại lệ phát sinh trong chương trình không bị debug, thì việc xử lý lỗi sẽ được chuyển giao cho bộ quản lý lỗi đã được chỉ định ở trên bởi thông số `lpTopLevelExceptionFilter` của hàm.

Theo tài liệu mà anh Còm đã dịch của tác giả Pumqara, tôi có được cái nhìn rõ ràng hơn như sau :

Ý tưởng của phương pháp này là ta sẽ tạo một bộ quản lý lỗi ngoại lệ bằng hàm `SetUnhandledExceptionFilter`. Sau đó ta sẽ cố tình phát sinh một lỗi bất thường trong chương trình, khi đó nếu chương trình đang bị debug thì lỗi này sẽ không được chuyển giao cho bộ quản lý lỗi đã được chỉ định ở trên. Khi ấy, debugger sẽ không biết phải làm gì và dẫn tới crash chương trình. Chương trình không bị debug thì mọi thứ diễn ra bình thường. Điều này có nghĩa là, khi ấy chương trình sẽ gọi bộ quản lý lỗi, xử lý lỗi phát sinh và chuyển tới đoạn lệnh kế tiếp.

OK, vậy là ta đã có được những thông tin rất bổ ích, quay trở lại màn hình chính của Olly ta thấy crackme này gọi hàm `SetUnhandledExceptionFilter` ngay từ những dòng code đầu tiên :



Hàm này chỉ nhận một tham số truyền vào `lpTopLevelExceptionFilter`, đó là địa chỉ mà crackme định nghĩa để tiếp tục thực thi từ địa chỉ đó khi có một ngoại lệ phát sinh từ chương trình, nếu như chương trình không bị debug bởi một trình Debugger nào. Cụ thể ở crackme này, địa chỉ đó là `00401108 = <Sphynx.TopLevelExceptionFilter>`. Thông tin về `lpTopLevelExceptionFilter` ta có được như sau:

`lpTopLevelExceptionFilter`

Supplies the address of a top-level exception filter function that will be called whenever the `UnhandledExceptionFilter` function gets control, and the process is not being debugged. A value of `NULL` for this parameter specifies default handling within `UnhandledExceptionFilter`.

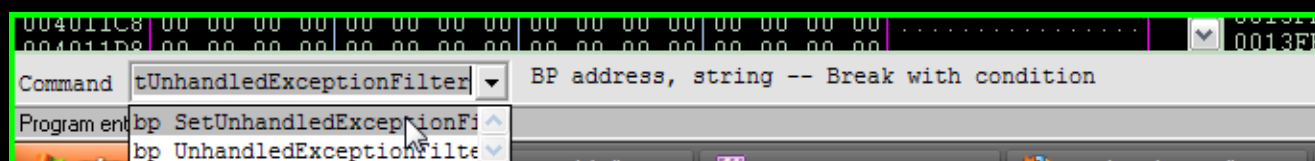
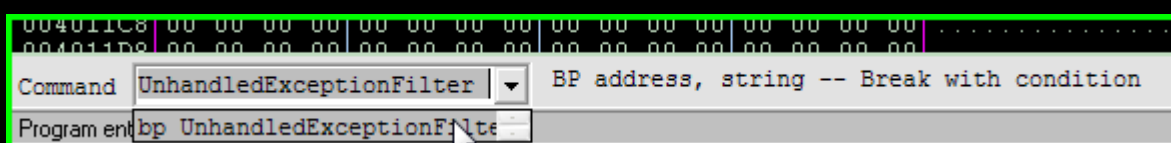
Qua thông tin trên ta thấy xuất hiện thêm một hàm API nữa là : **UnhandledExceptionFilter**. Hàm này có nhiệm vụ là nếu như chương trình của chúng ta đang được debug bởi một trình Debugger nào đó thì nó sẽ truyền các *unhandled exceptions* cho trình Debugger.

UnhandledExceptionFilter:

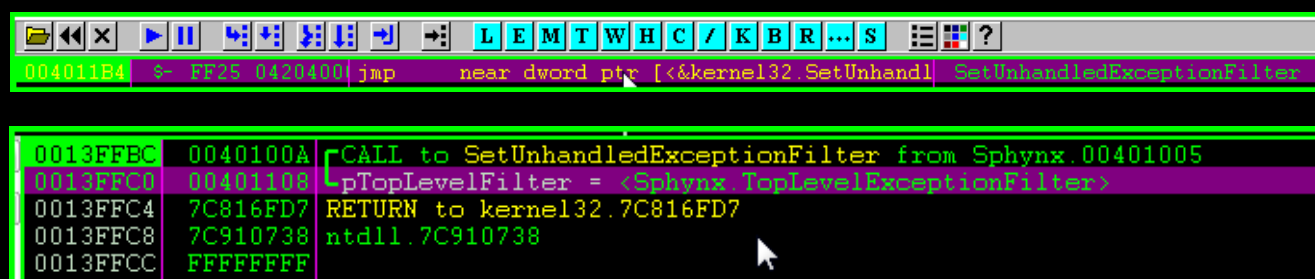
When an exception occurs, with Windows XP SP>=2, Windows 2003, and Windows Vista, the usual way the OS processes the exception is:

- If it is not debugged, it will call the user-defined filter function (set via kernel32!SetUnhandledExceptionFilter).
- If it debugged, the program will be terminated.

Tại OllyDbg, chúng ta tiến hành thiết lập 2 BP quan trọng tại hai hàm **SetUnhandledExceptionFilter** và **UnhandledExceptionFilter** như sau :



Sau khi đặt BP xong, nhấn F9 để run crackme. Olly sẽ break tại đây :



Đặt tiếp một BP tại địa chỉ chứa **lpTopLevelFilter** :



Nhấn F9 để tiếp tục run crackme, khi crackme run hoàn toàn ta nhập đại fake serial vào và nhấn nút Check. Olly sẽ break tại hàm **UnhandledExceptionFilter**, là vì trong chương trình phát sinh ra một exception không kiểm soát được và do chương trình đang được debug cho nên nó truyền exception này cho OllyDbg. Ngoại lệ này là do người code có tình đưa vào nhằm mục đích thông qua nó phát hiện xem chương trình có đang bị debug hay không.

7C862E62	68 74060000	push 674	UnhandledExceptionFilter
7C862E67	68 7039867C	push kernel32.7C863970	
7C862E6C	E8 55F6F9FF	call kernel32.7C8024C6	
7C862E71	44 82860000	call kernel32.7C8024C6	

0013F70C	7C8436DA	CALL to UnhandledExceptionFilter from kernel32.7C8436D5	
0013F710	0013F730	pExceptionInfo = 0013F730	
0013F714	7C839B09	RETURN to kernel32.7C839B09	

Như đã nói, hàm API này dùng để xác minh xem process có bị debug không, nếu đang bị debug thì sau đó sẽ không gọi một lệnh nhảy tới vùng nhớ 00401108=<Sphynx.TopLevelExceptionHandler> được thiết lập bởi hàm `SetUnhandledExceptionFilter`. Để tìm hiểu chi tiết ta nhấn F8 để trace và quan sát xem cơ chế phát hiện Debugger :

7C862ED9	53	push ebx	
7C862EDA	8D85 B4FAFFFF	lea eax,dword ptr [ebp-54C]	
7C862EE0	50	push eax	
7C862EE1	6A 07	push 7	
7C862EE3	E8 0DAFFAFF	call kernel32.GetCurrentProcess	
7C862EE8	50	push eax	
7C862EE9	FF15 AC10007C	call near dword ptr [&ntdll.NtQueryInformationProcess]	
7C862EEF	85C0	test eax,eax	
7C862EF1	0F8C A4000000	j1 kernel32.7C862F9B	
7C862EF7	33DB	xor ebx,ebx	
7C862EF8	8B85 B4FAFFFF	mov ebx,dword ptr [ebp-54C]	

Ta có gì nào, `ZwQueryInformationProcess` ☺. Như tôi đã nói ở đầu, API này được kết hợp với cơ chế `UnhandledException` để phát hiện ra debugger. Ngoài ra, cũng có thể sử dụng trực tiếp hàm `ZwQueryInformationProcess` để phát hiện ra Debugger bằng cách truyền vào tham số `InfoClass = 7`. Quan sát cửa sổ Stack ta thấy các tham số truyền vào như sau :

0013F064	FFFFFFFF	hProcess = FFFFFFFF
0013F068	00000007	InfoClass = 7
0013F06C	0013F1BC	Buffer = 0013F1BC
0013F070	00000004	Bufsize = 4
0013F074	00000000	pReqsie = NULL
0013F078	00000000	

ZwQueryInformationProcess Function

[**ZwQueryInformationProcess** may be altered or unavailable in future versions of Windows. To help with development, we have provided this function. It should use the alternate functions listed in this topic.]

Retrieves information about the specified process.

Syntax

C++

```
NTSTATUS WINAPI ZwQueryInformationProcess(
    __in     HANDLE ProcessHandle,
    __in     PROCESSINFOCLASS ProcessInformationClass,
    __out     PVOID ProcessInformation,
    __in     ULONG ProcessInformationLength,
    __out_opt PULONG ReturnLength
);
```

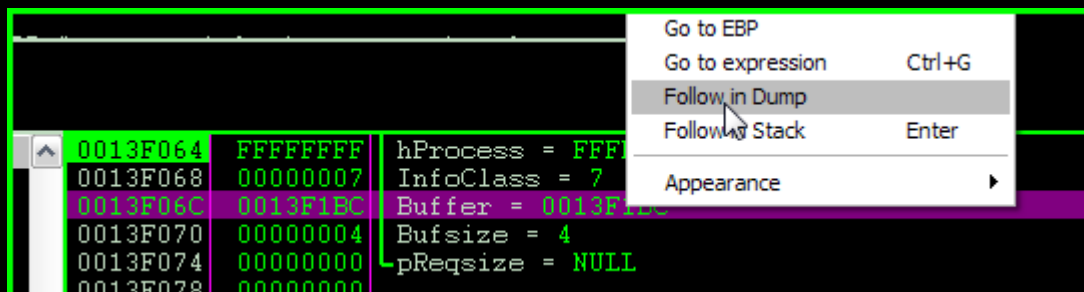
Ở đây ta quan tâm tới một tham số quan trọng là **ProcessInformationClass**, tương ứng với **0013F068 00000007 |InfoClass = 7** trên cửa sổ Stack. Thông tin về nó như sau :

ProcessInformationClass [in]

The type of process information to be retrieved. This parameter can be one of the following values from the **PROCESSINFOCLASS** enumeration.

ProcessDebugPort 7	Retrieves a DWORD_PTR value that is the port number of the debugger for the process. A nonzero value indicates that the process is being run under the control of a ring 3 debugger. It is best to use the CheckRemoteDebuggerPresent or IsDebuggerPresent function.
-----------------------	--

Ta thấy tham số này sẽ tìm một giá trị là port number của debugger đang debug process, nếu có giá trị nonzero thì đồng nghĩa với việc chương trình đang bị debug ☺. Ta **Follow in Dump** tại vùng buffer :



Address	Hex dump	ASCII
0013F1BC	00 00 00 00 FC FD FE FF 00 00 34 01 04 05 00 07	úýþý..4 .
0013F1CC	08 09 0A 0B 14 F1 13 00 10 11 12 13 A8 F2 13 00	...ñ...ò.
0013F1DC	18 EE 00 7C C8 6A 92 7C 28 F1 13 00 BE 6A 92 7C	ì.Èj' (ñ.j'
0013F1EC	BC F2 13 00 18 EE 90 7C C8 6A 92 7C FF FF FF FF	ò.ì.Èj'ýýýý
0013F1FC	BE 6A 92 7C AD 68 92 7C 00 00 34 01 60 00 00 40	j' -h'..4'..@
0013F20C	6D 05 91 7C 04 27 34 01 38 1F 34 01 00 00 00 00	m' ' 4 8 4 ...

Kích thước của vùng buffer này là 4 bytes, giá trị hiện tại đang là **00 00 00 00**. Nếu như sau khi thực hiện hàm API này mà kết quả không phải như trên mà là một giá trị nonzero nào đó, thì chắc chắn một điều là đã phát hiện ra target đang bị debug ☺. Ta nhấn F8 để thực hiện lời gọi hàm `ZwQueryInformationProcess` và quan sát kết quả trả về tại vùng buffer :

Address	Hex dump	ASCII
0013F1BC	FF FF FF FF FC FD FE FF 00 00 34 01 04 05 00 07	ýýýýúýþý..4 .
0013F1CC	08 09 0A 0B 14 F1 13 00 10 11 12 13 A8 F2 13 00	...ñ...ò.
0013F1DC	18 EE 00 7C C8 6A 92 7C 28 F1 13 00 BE 6A 92 7C	ì.Èj' (ñ.j'
0013F1EC	BC F2 13 00 18 EE 90 7C C8 6A 92 7C FF FF FF FF	ò.ì.Èj'ýýýý
0013F1FC	BE 6A 92 7C AD 68 92 7C 00 00 34 01 60 00 00 40	j' -h'..4'..@
0013F20C	6D 05 91 7C 04 27 34 01 38 1F 34 01 00 00 00 00	m' ' 4 8 4 ...

Kết quả trả về như ta thấy ở trên hình là **0xFFFFFFFF**, đồng nghĩa với việc có Debugger. Tiếp tục nhấn F8 để trace xuống đoạn code sau :

L E M T W H C / K B R ... S			
7C862EE9	FF15 AC10807C	call	near dword ptr [&ntdll.NtQueryInform ntdll.ZwQueryInformationProcess
7C862EEF	85C0	test	eax, eax
7C862EF1	0F8C A4000000	j1	kernel32.7C862F9B
7C862EF7	33DB	xor	ebx, ebx
7C862EF9	399D B4FAFFFF	cmp	dword ptr [ebp-54C], ebx
7C862EFF	0F84 96000000	je	kernel32.7C862F9B
7C862F05	64:A1 18000000	mov	eax, dword ptr fs:[18]
7C862F0B	8B40 30	mov	eax, dword ptr [eax+30]
7C862F0E	F640 69 01	test	byte ptr [eax+69], 1
7C862F12	0F84 A6080000	je	kernel32.7C8637BE
7C862F18	8B06	mov	eax, dword ptr [esi]
7C862F1A	3938	cmp	dword ptr [eax], edi
7C862F1C	75 3F	jnz	short kernel32.7C862F5D
7C862F1E	6A 01	push	1
7C862F20	68 1054887C	push	kernel32.7C885410
7C862F25	59 6420B1FF	call	kernel32.7C862F9B

00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000	
ebx=00000000	
Stack ss:[0013F1BC]=FFFFFFFF	
Address	Hex dump

Lệnh `7C862EF9 399D B4FAFFFF cmp dword ptr [ebp-54C], ebx` như các bạn thấy nhằm mục đích so sánh kết quả có được tại vùng Buffer có bằng 0x0 hay không? Nếu bằng thì sẽ thực hiện lệnh nhảy bên dưới. Nhưng ở trong trường hợp của chúng ta, giá trị tại Buffer hiện đang là nonzero cho nên lệnh nhảy sẽ không thực hiện.

Kết quả này đồng nghĩa với lệnh JE sẽ thực hiện :

7C862EFF	0F84 96000000	je	kernel32.7C862F5D	
7C862F05	64 A1 18000000	mov	eax,dword ptr fs:[18]	
7C862F0B	8B40 30	mov	eax,dword ptr [eax+30]	
7C862F0E	F640 69 01	test	byte ptr [eax+69],1	
7C862F12	0F84 A6080000	je	kernel32.7C8637BE	
7C862F18	8B06	mov	eax,dword ptr [esi]	
7C862F1A	3938	cmp	dword ptr [eax],edi	
7C862F1C	75 3F	jnz	short kernel32.7C862F5D	
7C862F1E	6A 01	push	1	
7C862F20	68 1054887C	push	kernel32.7C885410	
7C862F25	E8 6468FAFF	call	kernel32.InterlockedExchange	
7C862F2A	85C0	test	eax,eax	
7C862F2C	75 2F	jnz	short kernel32.7C862F5D	
7C862F2E	8B06	mov	eax,dword ptr [esi]	
7C862F30	68 5439867C	push	kernel32.7C863954	ASCII ".cxr (context record)"
7C862F35	FF76 04	push	dword ptr [esi+4]	
7C862F38	68 3C39867C	push	kernel32.7C86393C	ASCII ".exr (exception record)"
7C862F3D	50	push	eax	
7C862F3E	68 1C39867C	push	kernel32.7C86391C	ASCII "Code performing invalid access"
7C862F43	FF70 0C	push	dword ptr [eax+C]	
7C862F46	68 FC38867C	push	kernel32.7C8638FC	ASCII "Invalid address being accessed"
7C862F4B	FF70 18	push	dword ptr [eax+18]	
7C862F4E	68 C838867C	push	kernel32.7C8638C8	ASCII "access violation exception for current thread"
7C862F53	68 02000000	push	20000000	

Vậy là xong, ta nhấn F9 để run crackme xem thế nào. Olly sẽ break tại địa chỉ 401108 mà ta đã thiết lập BP từ trước :

00401108	BB75 08	mov	esi,dword ptr [ebp+8]	TopLevelExceptionFilter
0040110B	8B46 04	mov	eax,dword ptr [esi+4]	
0040110E	05 B8000000	add	eax,0B8	
00401113	8BF0	mov	esi,eax	
00401115	8B00	mov	eax,dword ptr [eax]	
00401117	83C0 0F	add	eax,0F	

Quan sát đoạn code ở phía dưới ta thấy có lời gọi tới hàm `MessageBoxA` :

00401165	83F9 31	cmp	ecx,31	
00401168	75 F2	jnz	short <Sphynx.loc_40115C>	
0040116A	6A 00	push	0	Style = MB_OK MB_APPLMODAL
0040116C	68 63314000	push	offset <Sphynx.Caption>	Title = ""
00401171	68 76314000	push	offset <Sphynx.Text>	Text = ""
00401176	6A 00	push	0	hOwner = NULL
00401178	E8 19000000	call	<Sphynx.MessageBoxA>	MessageBoxA
0040117D	33C0	xor	eax,eax	loc_40117D

Trước nó là một đoạn code tính toán, so sánh giá trị sau khi tính toán với giá trị C3B42A38, nếu không bằng thì sẽ nhảy qua `MessageBoxA`. Như vậy ta đoán khả năng thông báo này là "Good Message" © :

00401143	81FA 382AB4C	cmp	edx,C3B42A38	
00401149	75 32	jnz	short <Sphynx.loc_40117D>	
0040114B	33C9	xor	ecx,ecx	
0040114D	33D2	xor	edx,edx	
0040114F	BE 07304000	mov	esi,offset <Sphynx.unk_403007>	
00401154	BF 63314000	mov	edi,offset <Sphynx.Caption>	
00401159	8A56 12	mov	dl,byte ptr [esi+12]	
0040115C	8A0431	mov	al,byte ptr [ecx+esi]	loc_40115C
0040115F	32C2	xor	al,dl	
00401161	880439	mov	byte ptr [ecx+edi],al	
00401164	41	inc	ecx	
00401165	83F9 31	cmp	ecx,31	
00401168	75 F2	jnz	short <Sphynx.loc_40115C>	
0040116A	6A 00	push	0	Style = MB_OK MB_APPLMODAL
0040116C	68 63314000	push	offset <Sphynx.Caption>	Title = ""
00401171	68 76314000	push	offset <Sphynx.Text>	Text = ""
00401176	6A 00	push	0	hOwner = NULL
00401178	E8 19000000	call	<Sphynx.MessageBoxA>	MessageBoxA
0040117D	33C0	xor	eax,ecx	loc_40117D
0040117F	48	dec	eax	
00401180	C9	leave	eax	
00401181	C2 0400	retn	4	

Sửa lại cờ ZF hoặc patch lệnh nhảy, nhấn F9 và quan sát kết quả mà ta có được :

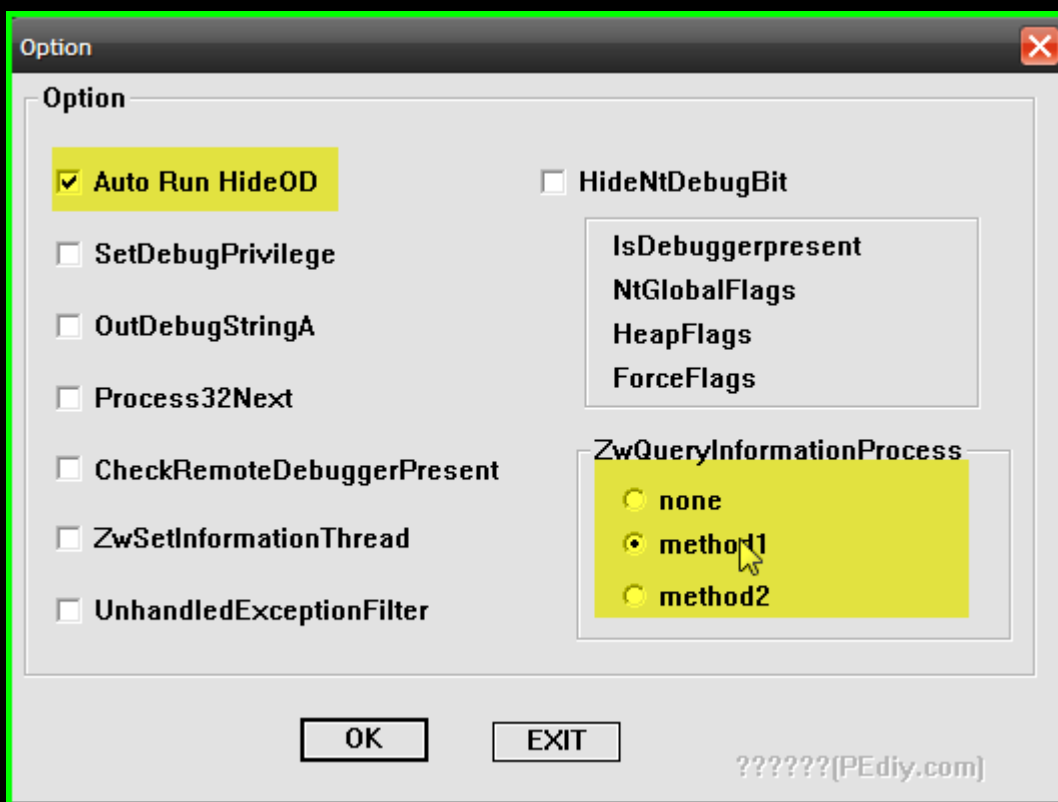
00401149	75 32	jnz	short <Sphynx.loc_40117D>	
0040114B	33C9	xor	ecx,ecx	
0040114D	33D2	xor	edx,edx	
0040114F	BE 07304000	mov	esi,offset <Sphynx.unk_403007>	
00401154	BF 63314000	mov	edi,offset <Sphynx.Caption>	ASCII "Sphynx by d@b 2004"
00401159	8A56 12	mov	dl,byte ptr [esi+12]	
0040115C	8A0431	mov	al,byte ptr [ecx+esi]	loc_40115C
0040115F	32C2	xor	al,dl	
00401161	880439	mov	byte ptr [ecx+edi],al	
00401164	41	inc	ecx	
00401165	83F9 31	cmp	ecx,31	
00401168	75 F2	jnz	short <Sphynx.loc_40115C>	
0040116A	6A 00	push	0	Style = MB_OK MB_APPLMODAL
0040116C	68 63314000	push	offset <Sphynx.Caption>	Title = "Sphynx by d@b 2004"
00401171	68 76314000	push	offset <Sphynx.Text>	Text = "Well Done ! Write a Tutorial !"
00401176	6A 00	push	0	hOwner = NULL
00401178	E8 19000000	call	<Sphynx.MessageBoxA>	MessageBoxA
0040117D	33C0	xor	eax,ecx	loc_40117D

OK vậy là xong! Như đã nói ở phần đầu bài viết, plugin HideDebugger có tùy chọn cho phép ta vượt qua kĩ thuật Anti-Debug này. Do đó thay vì phải thực hiện đi thực hiện lại việc manual bypass thì ta chọn option **Unhandled exception tricks**, sau đó save lại và restart Olly để thiết lập có hiệu lực. Load thử target và nhấn F9 để kiểm tra, thấy crackme run ngon lành.

Tuy nhiên, ngoài ra ta còn một vấn đề cần phải chú ý nữa là làm sao vượt qua được cơ chế kiểm tra của hàm **ZwQueryInformationProcess**, khi mà hàm này được gọi một cách trực tiếp chứ không phải thông qua hàm **UnhandledExceptionFilter** như ở crackme này. Mà để vượt qua được thì như các bạn đã thấy khi chúng ta thực hiện bằng tay là thay đổi giá trị trả về tại vùng Buffer thành zero. Vậy có plugin nào cho phép thực hiện công việc này một cách tự động không? Rất may mắn là thường thì **"Vô quýt dày có móng tay nhọn"**, lão kanxue bên PeiDy đã code một plugin có tên là HideOD (plugin này một thời tung hoành ngang dọc cho tới khi có sự xuất hiện của Phantom, Poison và StrongOD). Các bạn có thể download phiên bản mới nhất của nó tại đây :

<http://www.pediy.com/tools/Debuggers/ollydbg/plugin/hideOD/hideod.rar>

Extract vào trong thư mục chứa plugin của OllyDbg, sau đó chạy OllyDbg và cấu hình plugin này như sau :



Plugin này như các bạn thấy có thêm nhiều tùy chọn hơn HideDebugger, tuy nhiên với bài viết này ta chỉ chọn những option liên quan như trên hình minh họa. Việc tích chọn *Auto Run HideOD* nhằm mục đích cho plugin này tự động kích hoạt mỗi khi ta dùng OllyDbg để debug chương trình, đỡ phải vào **Menu Plugins > HideOD > Hide**.

Tôi xin kết thúc toàn bộ phần 22 tại đây. Toàn bộ bài viết này được tổng kết ngắn gọn như sau :

- 1) When the exception fires, set a breakpoint on the exported API, `UnhandledExceptionFilter`, inside of Kernel32;
- 2) Run until after the call to `NtQueryInformationProcess` inside of this API;
- 3) The last parameter to this call will hold the results of the call (which will be `0xFFFFFFFF` if a debugger is attached);
- 4) Change this value to zero - this tells the process that there is no debugger attached to the process and allows the application's exception handler to fire.

Ref : <http://www.openrce.org/forums/posts/45>

III. Kết luận

OK, toàn bộ bài 22 đến đây là kết thúc. Tổng kết toàn bộ bài viết này tôi đã trình bày cho các bạn biết thêm về các hàm API mới, được sử dụng trong việc Anti-Debug chương trình đó là các hàm sau : `SetUnhandledExceptionFilter`, `UnhandledExceptionFilter`, `ZwQueryInformationProcess`, các hàm này có thể được sử dụng kết hợp hoặc tách biệt tùy vào mục đích của người code. Qua bài viết này ta cũng nắm được phương pháp để manual bypass cũng như áp dụng các plugin có sẵn để giúp OllyDbg vượt qua các cơ chế Anti-Debug này. Hẹn gặp lại các bạn ở bài 23, hi vọng sẽ mang lại nhiều điều thú vị khác!

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]

Kien

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

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

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

iamidiot, WhyNotBar, trickyboy, dzungltn, takada, hurt_heart, haule_nth, hytkl, moth, XIANUA, nhc1987, 0xdie, Unregistered!, akira, 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:
[kienmanowar\[at\]reaonline.net](mailto:kienmanowar[at]reaonline.net)