



مقدمه مولف

آنپک کردن یک هنر است. یکی از هیجان انگیزترین بازی های فکری که در دنیای Reversing باقی مانده است. آنطور که من متوجه شدم اکثر کرکرهای تازه کار، نمی توانند فایل ها را دستی آنپک کنند و برای آنپک کردن به آنپکرهای نرم افزاری پناه می برند. به همین دلیل این مقاله را نوشتم. تا کرکرهای تازه کار با راه و روش های کلی آنپکینگ آشنا شوند. در این مقاله ما ۵۵ فایل پک شده را آنپک می کنیم. سعی کردم در این قسمت از مقاله، از پکرهای ساده تر استفاده کنم. اگر هم پکر سفتی را انتخاب کردم، سعی کردم تنها تعداد کمی از Protection های آن را فعال کنم تا این مقاله برای شما زیاد سفت نباشد. توصیه ما برای خواندن مقاله این است که تمام کارهایی که من انجام دادم را شما همراه من قدم به قدم انجام دهید و علاوه بر روش های من سعی کنید روش های بهتری پیدا کنید ☺ یعنی بعد از خواندن این مقاله شما باید ۵۵ فایل آنپک شده در اختیار داشته باشید. اگر زنده باشیم، این مقاله قسمت دومی هم خواهد داشت که در آن فایل های سفت تری را آنپک می کنم. لیست نیمه کاملی از مباحثی که در قسمت دوم مطرح می شود:

Analysing Virtual Machine in Asprotect – Unpack Armadillo with Nanomites – Unpack SDProtector 1.16 – How to write a simple unpacker in MASM – Analysis Protections in ActiveMark – Unpack Thinstall – Unpack .net Applications – How to write a plugin for ImportREC

در پایان از تمامی دوستانی که ما را در سافت این مقاله سفت و طولانی کمک کردند، تشکر می کنیم:

1. Tactools → به خاطر تست فایل های آنپک شده + تست اسکریپت ها با فایل های مختلف + آپلود مقاله و نرم افزارهای هدف
2. Y_ashar@yahoo.com → تست فایل های آنپک شده + راهنمایی در نوشتن مقالات
3. MnF → خواندن نسخه های اولیه مقاله + نظرات و پیشنهادات مفید
4. mJoOT → نرم افزار InfoViewer از ایشان بود، که ما به عنوان نرم افزار هدف از آن استفاده کردیم + قالب فایل آموزشی
5. Hakhamanesh → تست فایل های آنپک شده

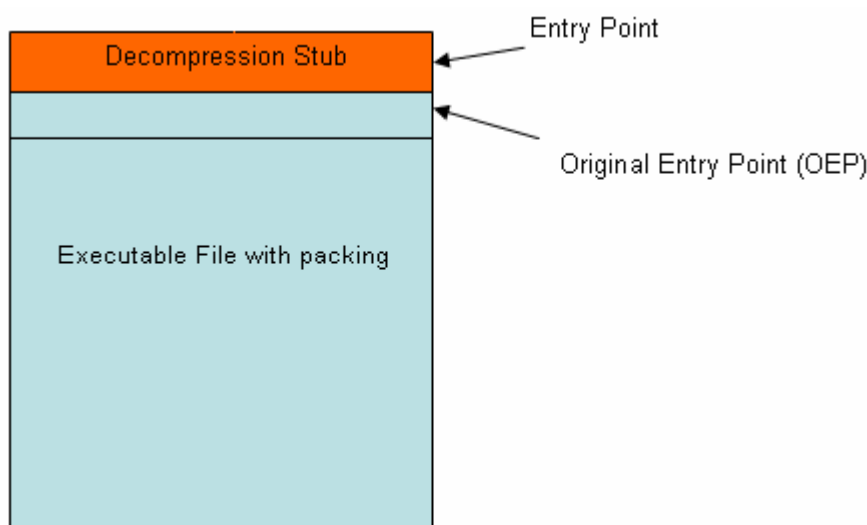


مقدمه ۱: پکر چیست ؟

پک کردن یک فایل اجرایی^۱، عملیاتی است که طی آن یک فایل اجرایی فشرده شده و برنامه ای از طرف پکر که به آن Decompression Stub می گوییم، به ابتدای آن تزریق می شود. Decompression Stub که به ابتدای فایل ما تزریق می شود، وظیفه اجرای فایل ما را دارد.

وقتی ما فایل پک شده را اجرا می کنیم، ابتدا Decompression Stub اجرا می شود. فایل اصلی ما را در داخل خودش پیدا می کند، و بعد شروع به باز کردن آن فایل و به اصطلاح Decompress کردن فایل می کند. بعد از اینکه فایل ما به طور کامل Decompress شد. آنوقت اختیار به دست فایل اصلی می افتد و برنامه ما اجرا می شود. این عملیات به طور کامل از دید یک کاربر معمولی مخفی می ماند و کاربر فکر می کند که این فایل همان فایل قبلی ما می باشد.

Entry Point که می دانید یعنی چی؟... یعنی جایی که برنامه ما از آنجا شروع به کار می کند. در مورد Packer ها ما دو تا Entry Point داریم. یکی Entry Point مربوط به Decompression Stub و دیگری Entry Point فایل اصلی ما که به آن Original Entry Point می گوییم:



فرقی نمی کند که Packer ما چي باشد، هر چي باشد، به هر حال، در یک جا و در یک زمانی، اختیار از دست Decompression Stub خارج شده و به دست فایل اصلی ما می افتد. اولین کاری که برای آنپک کردن باید بکنیم، این است که ببینیم در کجا و در چه خطی اختیار از دست برنامه ما خارج شده و فایل اصلی ما اجرا می شود؟... کجا اولین دستور فایل اصلی ما اجرا می شود؟... یا به زبان ساده، OEP فایل ما کجاست؟

مقدمه ۲: IAT چیست؟

همانطور که می دانید سیستم عامل ویندوز ورژن های مختلفی دارد و همه آنها آدرس های متفاوتی برای توابع API دارند. موقعی که یک برنامه شروع به کار می کند، لیست کاملی از توابع API مورد نیاز خود را در درون خود دارد. این توابع که به آنها Imports گفته می شود در درون فایل های DLL ویندوز قرار دارند. اما برنامه نمی داند که این توابع در کجای فایل های DLL قرار دارند. لذا اینجاست که نیاز به IAT (Import Address table) احساس می شود. IAT در واقع همانند یک جدول رجوع است که هر وقت برنامه نیازی به استفاده از یک تابع API داشت به آن مراجعه می کند. بنابراین لازم است که Loader ویندوز قبل از این که برنامه شروع به کار کند، لیستی از تمامی توابع API که برنامه لازم دارد را تهیه کند و آدرس توابع را در IAT بنویسد، تا هر وقت که برنامه به یک تابع احتیاج پیدا کرد، بداند که باید به کجا برود.

وقتی پکر فایلی را پک می کند، معمولا IAT برنامه را از بین می برد (تا کار آنپک کردن را سخت کند). بنابراین برای آنپک کردن باید IAT را دوباره بسازیم. به همین دلیل حتما باید با ساختار IAT آشنایی داشته باشیم.

حالا برای اینکه با IAT آشنایی بهتری پیدا کنیم، به بررسی IAT یک فایل واقعی می پردازیم. برنامه زیر را در OllyDBG اجرا کنید:

Targets/___Examples___/IAT/AspackDie.exe

Hex dump	Disassembly	Comment
E8 070A0000	CALL <JMP.<KERNEL32.GetModuleHandleA>	[pModule = NULL GetModuleHandleA
A3 74114000	MOV DWORD PTR DS:[4011741],EAX	
E8 97090000	CALL 004023B4	
A3 00124000	MOV DWORD PTR DS:[401200],EAX	
C705 10114000 96174000	MOV DWORD PTR DS:[401110],00401796	ASCII "EXE files"
C705 30114000 94174000	MOV DWORD PTR DS:[401130],00401794	
C705 38114000 04182800	MOV DWORD PTR DS:[401138],281804	
C705 20114000 00104000	MOV DWORD PTR DS:[401120],00401000	
C705 24114000 04010000	MOV DWORD PTR DS:[401124],104	
C705 34114000 B6174000	MOV DWORD PTR DS:[401134],004017B6	ASCII "[AspackDie 1.21 by yoda - Select t
68 04114000	PUSH 00401104	[pOpenFileName = AspackDi.00401104
E8 600A0000	CALL <JMP.<kernel32.GetOpenFileNameA>	[GetOpenFileNameA
0BC0	OR EAX,EAX	
74 0A	JE SHORT 00401A76	
68 00104000	PUSH 00401000	
E8 07000000	CALL 00401A7D	
6A 00	PUSH 0	[ExitCode = 0
E8 91090000	CALL <JMP.<kernel32.ExitProcess>	[ExitProcess
55	PUSH EBP	
8BEC	MOV EBP,ESP	
6A 00	PUSH 0	[hTemplateFile = NULL
68 80000000	PUSH 80	[Attributes = NORMAL
6A 03	PUSH 3	[Mode = OPEN_EXISTING
6A 00	PUSH 0	[pSecurity = NULL
6A 01	PUSH 1	[ShareMode = FILE_SHARE_READ
68 00000000	PUSH 00000000	[Access = GENERIC_READ
FF75 0A	PUSH [ARG_1]	[FileName

در خط دوم برنامه تابع GetModuleHandleA صدا زده شده، بذار ببینیم که چه طور این کار انجام شده؟ بر روی این تابع دوبار کلیک کنید:

Hex dump	Disassembly	Comment
E8 070A0000	CALL <JMP.<kernel32.GetModuleHandleA>	[pModule = NULL GetModuleHandleA
A3 74114000	MOV DWORD PTR DS:[4011741],EAX	
001, EAX		
101, 00401796		
301, 00401794		
381, 281804		
201, 00401000		
241, 104		
341, 004017B6		
tOpenFileNameA>		
itProcess>		
ExitProcess>		
hTemplateFile = NULL		
Attributes = NORMAL		
Mode = OPEN_EXISTING		
pSecurity = NULL		
ShareMode = FILE_SHARE_READ		
Access = GENERIC_READ		
FileName		
CreateFileA		
pFileSizeHigh = NULL		
hFile		
GetFileSize		

همانطور که می بینید `Call <jmp.Kernel32.GetModuleHandleA>=Call 0040241A` بریم، کافیه که به خط 0040241A برویم. خوب بهتر است برنامه را با F8 اجرا یعنی اینکه هر وقت قرار بود به تابع GetModuleHandleA برویم، کافیه که به خط 0040241A برویم. خوب بهتر است برنامه را با F8 اجرا کنیم و وارد تابع GetModuleHandleA شویم (با F7):

004023E0	CC	INT3	
004023F0	FF25 AC134000	JMP DWORD PTR DS:[<KERNEL32.CloseHandle>]	kernel32.CloseHandle
004023F6	FF25 B0134000	JMP DWORD PTR DS:[<KERNEL32.ContinueDebugEvent>]	kernel32.ContinueDebugEvent
004023FC	FF25 84134000	JMP DWORD PTR DS:[<KERNEL32.CreateFileA>]	kernel32.CreateFileA
00402402	FF25 B8134000	JMP DWORD PTR DS:[<KERNEL32.CreateProcessA>]	kernel32.CreateProcessA
00402408	FF25 BC134000	JMP DWORD PTR DS:[<KERNEL32.CreateThread>]	kernel32.CreateThread
0040240E	FF25 C0134000	JMP DWORD PTR DS:[<KERNEL32.ExitProcess>]	kernel32.ExitProcess
00402414	FF25 C4134000	JMP DWORD PTR DS:[<KERNEL32.GetFileSize>]	kernel32.GetFileSize
0040241A	FF25 C8134000	JMP DWORD PTR DS:[<KERNEL32.GetModuleHandleA>]	kernel32.GetModuleHandleA
00402420	FF25 CC134000	JMP DWORD PTR DS:[<KERNEL32.GetProcAddress>]	kernel32.GetProcAddress
00402426	FF25 D0134000	JMP DWORD PTR DS:[<KERNEL32.GetVersion>]	kernel32.GetVersion
0040242C	FF25 D4134000	JMP DWORD PTR DS:[<KERNEL32.GlobalAlloc>]	kernel32.GlobalAlloc
00402432	FF25 D8134000	JMP DWORD PTR DS:[<KERNEL32.GlobalFree>]	kernel32.GlobalFree
00402438	FF25 DC134000	JMP DWORD PTR DS:[<KERNEL32.ReadFile>]	kernel32.ReadFile
0040243E	FF25 E0134000	JMP DWORD PTR DS:[<KERNEL32.ReadProcessMemory>]	kernel32.ReadProcessMemory
00402444	FF25 E4134000	JMP DWORD PTR DS:[<KERNEL32.ResumeThread>]	kernel32.ResumeThread
0040244A	FF25 E8134000	JMP DWORD PTR DS:[<KERNEL32.SuspendThread>]	kernel32.SuspendThread
00402450	FF25 EC134000	JMP DWORD PTR DS:[<KERNEL32.TerminateProcess>]	kernel32.TerminateProcess
00402456	FF25 F0134000	JMP DWORD PTR DS:[<KERNEL32.VirtualAlloc>]	kernel32.VirtualAlloc
0040245C	FF25 F4134000	JMP DWORD PTR DS:[<KERNEL32.VirtualAllocEx>]	kernel32.VirtualAllocEx
00402462	FF25 F8134000	JMP DWORD PTR DS:[<KERNEL32.VirtualFree>]	kernel32.VirtualFree
00402468	FF25 FC134000	JMP DWORD PTR DS:[<KERNEL32.VirtualFreeEx>]	kernel32.VirtualFreeEx
0040246E	FF25 00144000	JMP DWORD PTR DS:[<KERNEL32.WaitForDebugEvent>]	kernel32.WaitForDebugEvent
00402474	FF25 04144000	JMP DWORD PTR DS:[<KERNEL32.WaitForSingleObject>]	kernel32.WaitForSingleObject
0040247A	FF25 08144000	JMP DWORD PTR DS:[<KERNEL32.WriteFile>]	kernel32.WriteFile
00402480	FF25 0C144000	JMP DWORD PTR DS:[<KERNEL32.WriteProcessMemory>]	kernel32.WriteProcessMemory
00402486	FF25 10144000	JMP DWORD PTR DS:[<USER32.lstrcpwA>]	kernel32.lstrcpwA
0040248C	FF25 14144000	JMP DWORD PTR DS:[<USER32.lstrcpyA>]	kernel32.lstrcpyA
00402492	FF25 18144000	JMP DWORD PTR DS:[<USER32.lstrlenA>]	kernel32.lstrlenA
00402498	FF25 1C144000	JMP DWORD PTR DS:[<USER32.CreateWindowExA>]	USER32.CreateWindowExA
0040249E	FF25 20144000	JMP DWORD PTR DS:[<USER32.DefWindowProcA>]	USER32.DefWindowProcA
004024A4	FF25 24144000	JMP DWORD PTR DS:[<USER32.DestroyWindow>]	USER32.DestroyWindow
004024AA	FF25 28144000	JMP DWORD PTR DS:[<USER32.DispatchMessageA>]	USER32.DispatchMessageA
004024B0	FF25 2C144000	JMP DWORD PTR DS:[<USER32.GetMessageA>]	USER32.GetMessageA
004024B6	FF25 30144000	JMP DWORD PTR DS:[<USER32.MessageBoxA>]	USER32.MessageBoxA
004024BC	FF25 34144000	JMP DWORD PTR DS:[<USER32.RegisterClassA>]	USER32.RegisterClassA
004024C2	FF25 38144000	JMP DWORD PTR DS:[<USER32.TranslateMessage>]	USER32.TranslateMessage
004024C8	FF25 3C144000	JMP DWORD PTR DS:[<comdlg32.GetOpenFileNameA>]	comdlg32.GetOpenFileNameA
004024CE	FF25 40144000	JMP DWORD PTR DS:[<comdlg32.GetSaveFileNameA>]	comdlg32.GetSaveFileNameA
004024D4	FF25 44144000	JMP DWORD PTR DS:[<IMAGEHLP.ImageNtHeader>]	IMAGEHLP.ImageNtHeader
004024DA	FF25 48144000	JMP DWORD PTR DS:[<IMAGEHLP.ImageRvaToSection>]	IMAGEHLP.ImageRvaToSection
004024E0	FF25 4C144000	JMP DWORD PTR DS:[<IMAGEHLP.ImageRvaToVa>]	IMAGEHLP.ImageRvaToVa
004024E6	FF25 50144000	JMP DWORD PTR DS:[<ForceLib.ForceLibTrapEntry>]	ForceLib.ForceLibTrapEntry
004024EC	FF25 54144000	JMP DWORD PTR DS:[<ForceLib.ForceLibLibraryDBG>]	ForceLib.ForceLibLibraryDBG
004024F2	FF25 58144000	JMP DWORD PTR DS:[<ForceLib.ForceLibCleanup>]	ForceLib.ForceLibCleanup
004024F8	00	DB 00	
004024F9	00	DB 00	

ما الان در Jump Table هستیم. تمامی توابع اول به اینجا منتقل می شوند و بعد با استفاده از این Jump ها به تابع مورد نظر می روند. البته وجود Jump Table الزامی نیست، یک برنامه می تواند مستقیم از IAT استفاده کند. حالا کافیهست که فقط یکبار F7 بزنی تا وارد تابع GetModuleHandleA در فایل Kernel32.dll بشوی. خوب، برای اینکه IAT را پیدا کنیم، کافی است که بر روی این Jump کلیک راست کرده، گزینه Follow In dump و بعد Memory Address را بزنی. به پنجره Dump نگاه کنی:

Address	Hex dump	ASCII
004013C8	A1 B6 80 7C A0 AD 80 7C	!@!@!@!@
004013D0	DA 11 81 7C 2D FD 80 7C	!@!@!@!@
004013D8	2F FC 80 7C 0E 18 80 7C	!@!@!@!@
004013E0	CC 21 80 7C F7 28 83 7C	!@!@!@!@
004013E8	32 97 83 7C 16 1E 80 7C	!@!@!@!@
004013F0	51 9A 80 7C 72 9A 80 7C	!@!@!@!@
004013F8	E4 9A 80 7C 02 9B 80 7C	!@!@!@!@
00401400	80 A4 85 7C 20 25 80 7C	!@!@!@!@
00401408	87 0D 81 7C 0F 22 80 7C	!@!@!@!@
00401410	74 0D 83 7C 01 BE 80 7C	!@!@!@!@
00401418	B6 BD 80 7C 00 00 00 00	!@!@!@!@
00401420	33 FF 41 7E EE D4 41 7E	!@!@!@!@
00401428	EA DA 41 7E B8 96 41 7E	!@!@!@!@
00401430	02 E0 42 7E 8A 05 45 7E	!@!@!@!@
00401438	36 0A 42 7E F6 8B 41 7E	!@!@!@!@
00401440	00 00 00 00 1E 31 3B 76	!@!@!@!@
00401448	D8 7C 3C 76 00 00 00 00	!@!@!@!@
00401450	AD 73 C9 76 11 75 C9 76	!@!@!@!@
00401458	54 75 C9 76 00 00 00 00	!@!@!@!@
00401460	00 22 00 0F 80 22 00 0F	!@!@!@!@
00401468	C0 23 00 0F 00 00 00 00	!@!@!@!@
00401470	19 00 43 6C 6F 73 65 48	!@!@!@!@
00401478	61 6E 64 6C 65 00 23 00	!@!@!@!@

Command:

start IAT

همانطور که می بینید الان محتویات پنجره Dump ما همان IAT ماست و آدرس توابع API ما در این آدرس قرار دارد. دقت کنید که در پنجره Dump اطلاعات را باید جفت جفت از آنور بخوانید، برای مثال در تصویر بالا در خط 004013C8 نوشته شده:

A1 B6 80 7C

ما باید خط بالا را اینجوری بخوانیم:

7C80B6A1

حالا برای اینکه ببینیم این تابع به کجا می رود، کافیهست که به آدرس 7C80B6A1 برویم:

7C80B69F	90	NOP	
7C80B6A0	90	NOP	
7C80B6A1	8BFF	MOV EDI,EDI	ntdll.7C910738
7C80B6A3	55	PUSH EBP	
7C80B6A4	8BEC	MOV EBP,ESP	
7C80B6A6	837D 08 00	CMP DWORD PTR SS:[EBP+8],0	
7C80B6AA	74 19	JE SHORT 7C80B6A4	
7C80B6AC	FF75 08	PUSH DWORD PTR SS:[EBP+8]	
7C80B6AF	E8 C0290000	CALL 7C80E074	
7C80B6B4	85C0	TEST EAX,EAX	
7C80B6B6	74 08	JE SHORT 7C80B6C0	
7C80B6B8	FF70 04	PUSH DWORD PTR DS:[EAX+4]	
7C80B6BB	E8 7D200000	CALL GetModuleHandle@	
7C80B6C0	5D	POP EBP	
7C80B6C1	C2 0400	RET 4	
7C80B6C4	64+A1 18000000	MOV EAX,DWORD PTR FS:[18]	
7C80B6CA	8B40 30	MOV EAX,DWORD PTR DS:[EAX+30]	
7C80B6CD	8B40 08	MOV EAX,DWORD PTR DS:[EAX+8]	
7C80B6D0	EB EE	JMP SHORT 7C80B6C8	
7C80B6D2	90	NOP	
7C80B6D3	90	NOP	
7C80B6D4	90	NOP	
7C80B6D5	90	NOP	
7C80B6D6	90	NOP	
7C80B6D7	8BFF	MOV EDI,EDI	
7C80B6D9	55	PUSH EBP	
7C80B6DA	8BEC	MOV EBP,ESP	
7C80B6DC	81EC 40060000	SUB ESP,640	
7C80B6E2	837D 08 01	CMP DWORD PTR SS:[EBP+8],1	
7C80B6E6	A1 CC46887C	MOV EAX,DWORD PTR DS:[7C8846CC]	
7C80B6EB	53	PUSH EBX	
7C80B6EC	56	PUSH ESI	
7C80B6ED	8B75 0C	MOV ESI,DWORD PTR SS:[EBP+C]	
7C80B6F0	8945 FC	MOV DWORD PTR SS:[EBP-4],EAX	
7C80B6F3	0F84 F9030000	JE 7C818AF2	
7C80B6F9	837D 08 02	CMP DWORD PTR SS:[EBP+8],2	
7C80B6FD	75 0E	JNZ SHORT 7C80B700	
7C80B6FF	E8 1E000000	CALL 7C805722	
7C80B704	84C0	TEST AL,AL	
7C80B706	74 05	JE SHORT 7C80B700	
7C80B708	E8 C8F80000	CALL 7C81AFD5	
7C80B70D	B0 01	MOV AL,1	
7C80B70F	8B40 FC	MOV ECX,DWORD PTR SS:[EBP-4]	
7C80B712	5E	POP ESI	
7C80B713	5B	POP EBX	

EDI=7C910738 (ntdll.7C910738)

می بینید این همان تابع GetModuleHandleA ماست که در کامپیوتر من در آدرس 7C80B6A1 قرار دارد. مطمئناً تابع GetModuleHandleA در سیستم شما در آدرسی متفاوت از کامپیوتر من قرار دارد. (به همین دلیل است که استفاده از IAT لازم است).

خوب، حالا می خواهیم IAT را با دقت بیشتری بررسی کنیم. برای این کار بهتر است در پنجره Dump کلیک راست کرده گزینه Long و سپس گزینه Address with ASCII dump را به این صورت ببینیم:

7C80B713	5B	POP EBX	
EDI=7C910738 (ntdll.7C910738)			
Address	Value	ASCII	Comment
004013E0	7C8021CC	kernel32	kernel32.ReadProcessMemory
004013E4	7C8328F7	kernel32	kernel32.ResumeThread
004013E8	7C839732	kernel32	kernel32.SuspendThread
004013EC	7C801E16	kernel32	kernel32.TerminateProcess
004013F0	7C809A51	kernel32	kernel32.VirtualAlloc
004013F4	7C809A72	kernel32	kernel32.VirtualAllocEx
004013F8	7C809AE4	kernel32	kernel32.VirtualFree
004013FC	7C809B02	kernel32	kernel32.VirtualFreeEx
00401400	7C85A480	kernel32	kernel32.WaitForDebugEvent
00401404	7C802520	kernel32	kernel32.WaitForSingleObject
00401408	7C810D87	kernel32	kernel32.WriteFile
0040140C	7C80220F	kernel32	kernel32.WriteProcessMemory

به تصویر زیر نگاه کنید، متوجه می شوید که توابع مربوط به هر DLL با یک 00000000 از هم جدا می شوند:

004013E8	7C839732	kernel32	kernel32.SuspendThread
004013EC	7C801E16	kernel32	kernel32.TerminateProcess
004013F0	7C809A51	kernel32	kernel32.VirtualAlloc
004013F4	7C809A72	kernel32	kernel32.VirtualAllocEx
004013F8	7C809AE4	kernel32	kernel32.VirtualFree
004013FC	7C809B02	kernel32	kernel32.VirtualFreeEx
00401400	7C85A480	kernel32	kernel32.WaitForDebugEvent
00401404	7C802520	kernel32	kernel32.WaitForSingleObject
00401408	7C810D87	kernel32	kernel32.WriteFile
0040140C	7C80220F	kernel32	kernel32.WriteProcessMemory
00401410	7C830D74	kernel32	kernel32.lstrcmpA
00401414	7C80BE01	kernel32	kernel32.lstrcpyA
00401418	7C80BDB6	kernel32	kernel32.lstrlenA
0040141C	00000000	USER32	USER32.CreateWindowExA
00401420	7E41FF33	USER32	USER32.DefWindowProcA
00401424	7E41D4EE	USER32	USER32.DestroyWindow
00401428	7E41DAEA	USER32	USER32.DispatchMessageA
0040142C	7E4196B8	USER32	USER32.GetMessageA
00401430	7E42E002	USER32	USER32.MessageBoxA
00401434	7E45058A	USER32	USER32.RegisterClassA
00401438	7E420A36	USER32	USER32.TranslateMessage
0040143C	7E418BF6	USER32	USER32.TranslateMessage
00401440	00000000		

Command:

همانطور که می بینید توابعی که مربوط به Kernel32.dll هستند از توابعی که مربوط به User32.dll هستند در آدرس 0040141C جدا شده اند. خوب، می خواهیم ببینیم که ابتدای IAT ما کجاست؟ در پنجره Dump آنقدر به بالا بروید تا به جایی برسید که دیگری وجود نداشته باشد:

Address	Value	ASCI	Comment
00401398	00000000		
0040139C	00001750	P+..	
004013A0	0000175C	\+..	
004013A4	0000176E	n+..	
004013A8	00000000		
004013AC	7C809B47	G+C:	kernel32.CloseHandle
004013B0	7C85A565	eN+:	kernel32.ContinueDebugEvent
004013B4	7C801A24	s+C:	kernel32.CreateFileA
004013B8	7C802367	g+C:	kernel32.CreateProcessA
004013BC	7C810637	7+u:	kernel32.CreateThread
004013C0	7C81CDDA	=u:	kernel32.ExitProcess
004013C4	7C810A77	w.u:	kernel32.GetFileSize
004013C8	7C80B6A1	l C:	kernel32.GetModuleHandleA
004013CC	7C80ADA0	a+C:	kernel32.GetProcAddress
004013D0	7C8111DA	r+u:	kernel32.GetVersion
004013D4	7C80FD2D	->C:	kernel32.GlobalAlloc
004013D8	7C80FC2F	/^C:	kernel32.GlobalFree
004013DC	7C80180E	stC:	kernel32.ReadFile
004013E0	7C8021CC	ltC:	kernel32.ReadProcessMemory
004013E4	7C8328F7	%l+:	kernel32.ResumeThread
004013E8	7C839732	2u+:	kernel32.SuspendThread
004013EC	7C801E16	-A+C:	kernel32.TerminateProcess
004013F0	7C809A51	QuC:	kernel32.VirtualAlloc

Command:

start IAT Generic Unpac... - [CP

پس خط 004013A8 که توابع ما بعد از آن شروع می شود، شروع IAT ماست. حالا بهتر است انتهای IAT را هم پیدا کنیم:

Address	Value	ASCI	Comment
00401440	00000000		
00401444	763B311E	!;v	comdlg32.GetOpenFileNameA
00401448	763C7CD8	!<v	comdlg32.GetSaveFileNameA
0040144C	00000000		
00401450	76C973AD	!sfv	IMAGEHLP.ImageNtHeader
00401454	76C97511	!ufv	IMAGEHLP.ImageRvaToSection
00401458	76C97554	!ufv	IMAGEHLP.ImageRvaToVa
0040145C	00000000		
00401460	0F002200	..*.	ForceLib.TrapEntry
00401464	0F002200	C".*	ForceLib.ForceLibraryDBG
00401468	0F0023C0	!#.*	ForceLib.PerformCleanup
0040146C	00000000		
00401470	6C430019	+Cl	
00401474	4865736F	oseH	
00401478	6C646E61	andl	
0040147C	00230065	e.#.	
00401480	746E6F43	Cont	
00401484	65756E69	inue	
00401488	75626544	Debu	
0040148C	65764567	gEve	
00401490	0000746E	nt..	
00401494	72430032	2.Cr	
00401498	65746165	eat	

Command:

پس در خط 00401470 که هیچ تابعی بعد از آن وجود ندارد، انتهای IAT ما می باشد. حالا بهتر است اندازه IAT خودمان را به دست بیاوریم. برای این کار از ماشین حساب ویندوز استفاده می کنیم، چون این اطلاعات برای مبنای هکس هست، لذا باید ابتدا ماشین حساب را بر مبنای Hex قرار دهیم، لذا در منوی View گزینه Hex را بزنید. و بعد انتهای IAT را منهای ابتدای IAT بکنید:

$$00401470 - 004013A8 = C8$$

پس :

$$\text{Start of IAT} = 4013A8$$

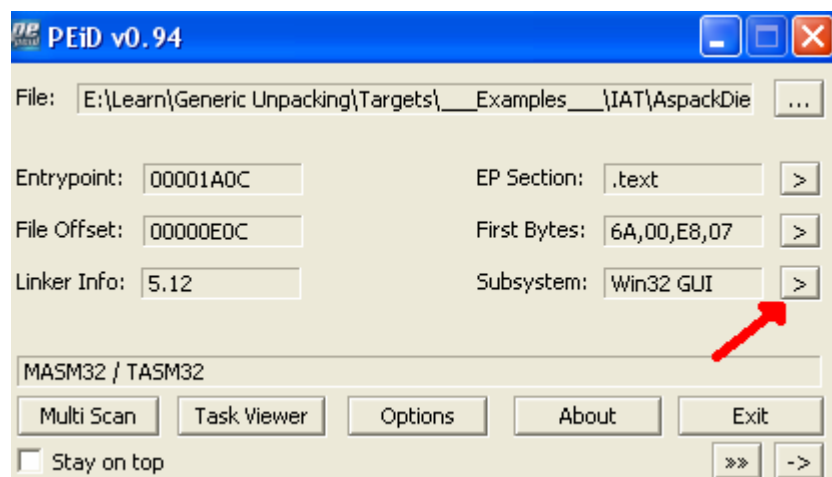
$$\text{End of IAT} = 401470$$

$$\text{Size of IAT} = C8$$

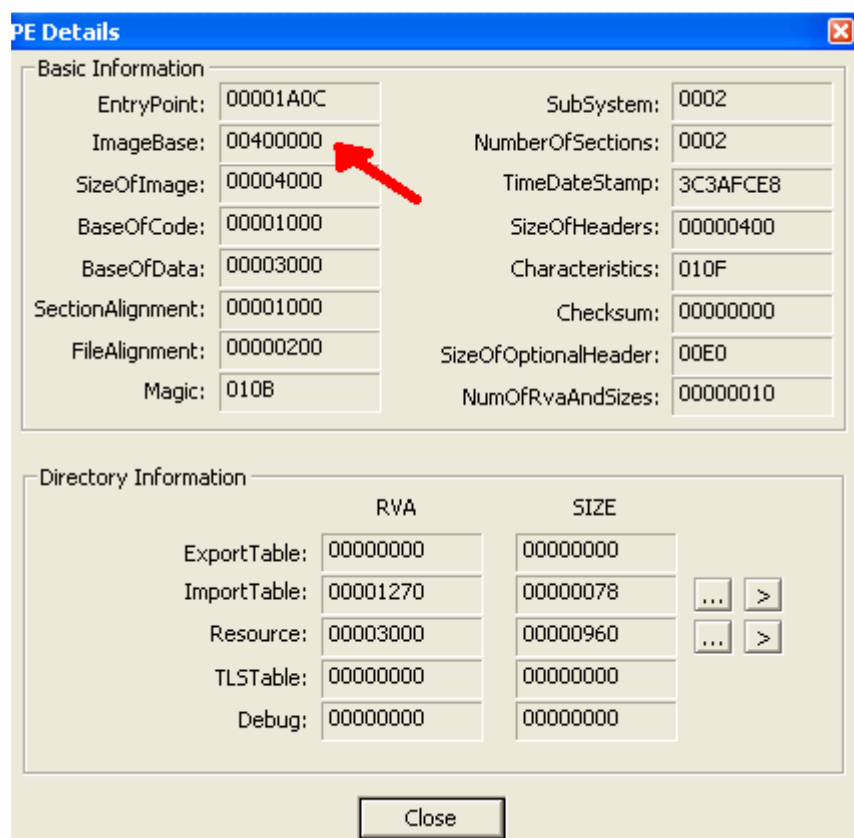
در بالا شروع IAT بر حسب VA محاسبه شده، ما باید آنها را بر مبنای RVA محاسبه کنیم. برای اینکار یک فرمول کلی داریم:

$$RVA = VA - \text{ImageBase}$$

ImageBase در واقع آدرس قرارگیری فایل در Memory هست که معمولاً برابر 00400000 می باشد. اما برای اینکه مطمئن شویم، با PEID بررسی می کنیم:



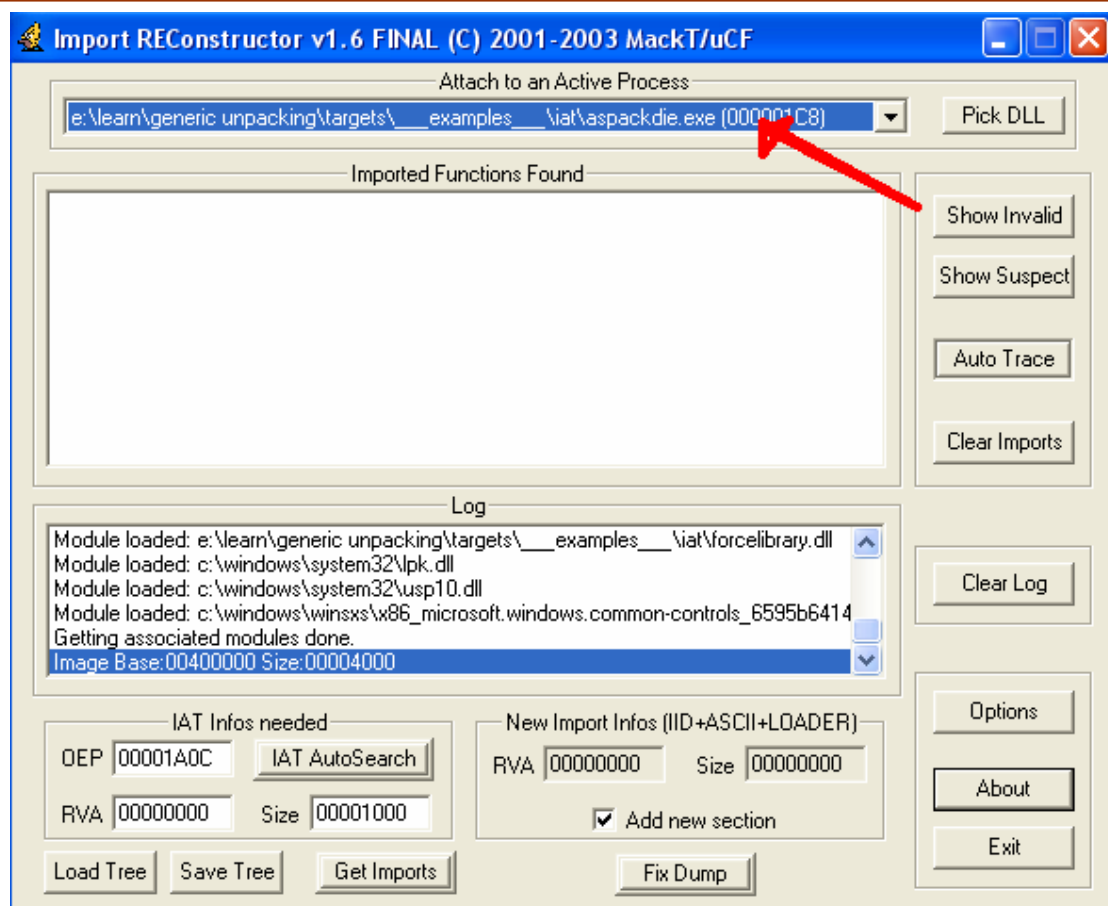
بر روی دکمه نشان داده شده کلیک کنید:



همانطور که می بینید ImageBase ما برابر 00400000 می باشد، لذا با توجه به فرمول بالا:

Start of IAT = 13A8
End of IAT = 1470
Size of IAT = C8

خوب، در مورد آنپک کردن فایل ما نیاز به برنامه ای داریم که IAT ما را دوباره بسازد برای این کار از ImportREC استفاده می کنیم، این برنامه را باز کنید تا اندکی با آن آشنایی پیدا کنیم:
اول از همه پروسه مورد نظرم را انتخاب می کنیم:



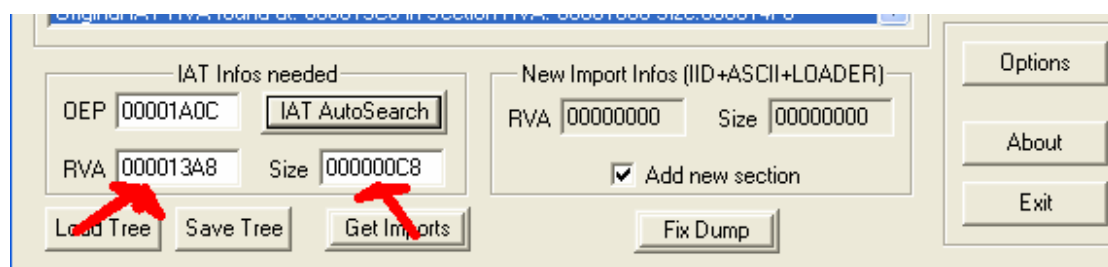
ما برای ساخت دوباره IAT باید اطلاعاتی را به برنامه بدهیم:

OEP

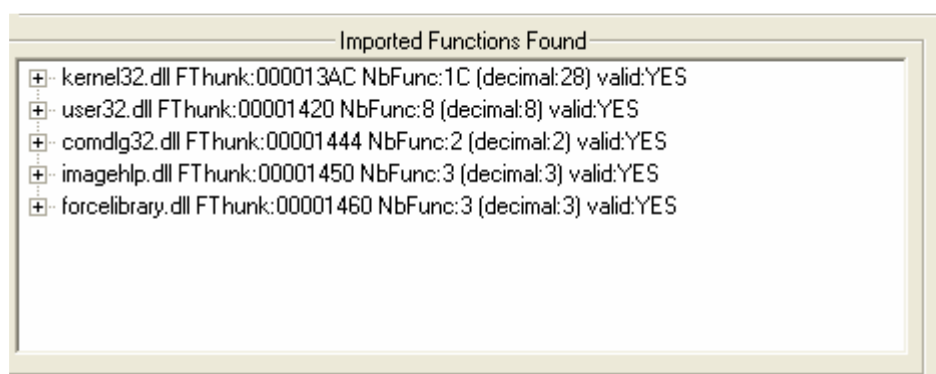
Start of IAT

Size of IAT

اطلاعات باید برحسب RVA باشد. البته خود ImportREC قابلیت دارد که اگر آدرس Entry Point (در مورد فایل های پک شده (Original Entry Point) را به برنامه بدهیم، برنامه خودش IAT را پیدا می کند © حالا گزینه IAT Auto Search را بزنید، می بینید که برنامه Start of IAT و Size of IAT را خودش می فهمد.



ولی خوب در مورد بعضی پکرها، این برنامه نمی تواند IAT را پیدا کند یا اینکه مشخصات IAT را اشتباه پیدا می کند، لذا در آن مواقع ما باید خودمان کل اطلاعات را پیدا کنیم، بعد از وارد کردن مشخصات گزینه Get Imports را می زنیم تا تمامی توابعی که برنامه استفاده کرده و در IAT وجود دارد را ببینیم:



همانطور که می بینید توابع ما تماما معتبر هستند و وجود دارند. همچنین این برنامه در کل از توابع ۵ فایل DLL استفاده می کند:

Kernel32.dll
User32.dll
Comdlg32.dll
Imagehlp.dll
Forcelibrary.dll

در ImportREC توابع مربوط به هر DLL در یک قسمت قرار می گیرند. حالا اگر می خواستیم IAT را دوباره بسازیم کافی بود گزینه Fix Dump را بزنیم و فایلمان را انتخاب کنیم تا برنامه IAT آن فایل را Fix کند. توصیه می کنم برای یادگیری بهتر IAT خودتان چند فایل را مورد بررسی قرار دهید.

۳-۱: ویژوال بیسیک

فایل زیر را در OllyDBG باز کنید:

Targets\Examples\Language Structure\Visual Basic\VB.exe

برنامه های ویژوال بیسیک از توابع API به طور مستقیم استفاده نمی کنند، بلکه فایل MSVBVM60.dll به عنوان واسطه این کار را انجام می دهد. لذا IAT قابل در ویژوال بیسیک فقط شامل آدرس توابع فایل MSVBVM60.dll می باشد:

Address	Hex dump	Disassembly	Comment
0040107E	00 00	DB 00	
0040107F	00 00	DB 00	
00401080	-- FF25 20104000	JMP DWORD PTR DS:[XMSUBVMB60_...vbaChkstk]	MSVBUM60_...vbaChkstk
00401081	-- FF25 38104000	JMP DWORD PTR DS:[XMSUBVMB60_...vbaExceptionHandler]	MSVBUM60_...vbaExceptionHandler
00401082	-- FF25 44104000	JMP DWORD PTR DS:[XMSUBVMB60_...vbaFPEException]	MSVBUM60_...vbaFPEException
00401083	-- FF25 14104000	JMP DWORD PTR DS:[XMSUBVMB60_...adj_fdivi_m16i]	MSVBUM60_...adj_fdivi_m16i
00401084	-- FF25 10104000	JMP DWORD PTR DS:[XMSUBVMB60_...adj_fdivi_m32i]	MSVBUM60_...adj_fdivi_m32i
00401085	-- FF25 4C104000	JMP DWORD PTR DS:[XMSUBVMB60_...adj_fdivi_m32i]	MSVBUM60_...adj_fdivi_m32i
00401086	-- FF25 08104000	JMP DWORD PTR DS:[XMSUBVMB60_...adj_fdivi_m64i]	MSVBUM60_...adj_fdivi_m64i
00401087	-- FF25 58104000	JMP DWORD PTR DS:[XMSUBVMB60_...adj_fdivr]	MSVBUM60_...adj_fdivr
00401088	-- FF25 18104000	JMP DWORD PTR DS:[XMSUBVMB60_...adj_fdivr_m16i]	MSVBUM60_...adj_fdivr_m16i
00401089	-- FF25 54104000	JMP DWORD PTR DS:[XMSUBVMB60_...adj_fdivr_m32i]	MSVBUM60_...adj_fdivr_m32i
0040108A	-- FF25 50104000	JMP DWORD PTR DS:[XMSUBVMB60_...adj_fdivr_m32i]	MSVBUM60_...adj_fdivr_m32i
0040108B	-- FF25 40104000	JMP DWORD PTR DS:[XMSUBVMB60_...adj_fdivr_m64i]	MSVBUM60_...adj_fdivr_m64i
0040108C	-- FF25 28104000	JMP DWORD PTR DS:[XMSUBVMB60_...adj_fpatan]	MSVBUM60_...adj_fpatan
0040108D	-- FF25 3C104000	JMP DWORD PTR DS:[XMSUBVMB60_...adj_fpreni]	MSVBUM60_...adj_fpreni
0040108E	-- FF25 0C104000	JMP DWORD PTR DS:[XMSUBVMB60_...adj_fpreni]	MSVBUM60_...adj_fpreni
0040108F	-- FF25 04104000	JMP DWORD PTR DS:[XMSUBVMB60_...adj_fpreni]	MSVBUM60_...adj_fpreni
00401090	-- FF25 60104000	JMP DWORD PTR DS:[XMSUBVMB60_...C1atan]	MSVBUM60_...C1atan
00401091	-- FF25 00104000	JMP DWORD PTR DS:[XMSUBVMB60_...C1cosi]	MSVBUM60_...C1cosi
00401092	-- FF25 0C104000	JMP DWORD PTR DS:[XMSUBVMB60_...C1cosi]	MSVBUM60_...C1cosi
00401093	-- FF25 48104000	JMP DWORD PTR DS:[XMSUBVMB60_...C1leqi]	MSVBUM60_...C1leqi
00401094	-- FF25 1C104000	JMP DWORD PTR DS:[XMSUBVMB60_...C1leqi]	MSVBUM60_...C1leqi
00401095	-- FF25 30104000	JMP DWORD PTR DS:[XMSUBVMB60_...C1sani]	MSVBUM60_...C1sani
00401096	-- FF25 68104000	JMP DWORD PTR DS:[XMSUBVMB60_...C1sani]	MSVBUM60_...C1sani
00401097	-- FF25 64104000	JMP DWORD PTR DS:[XMSUBVMB60_...C1tani]	MSVBUM60_...C1tani
00401098	-- FF25 34104000	JMP DWORD PTR DS:[XMSUBVMB60_...EVENT_SINK_QueryInter]	MSVBUM60_...EVENT_SINK_QueryInter
00401099	-- FF25 24104000	JMP DWORD PTR DS:[XMSUBVMB60_...EVENT_SINK_AddRef]	MSVBUM60_...EVENT_SINK_AddRef
0040109A	-- FF25 2C104000	JMP DWORD PTR DS:[XMSUBVMB60_...EVENT_SINK_Release]	MSVBUM60_...EVENT_SINK_Release
0040109B	-- FF25 5C104000	JMP DWORD PTR DS:[XMSUBVMB60_...m102]	MSVBUM60_...TurnRTMain
00401128 <ModuleEntryPoint>	58 64124000	CALL [XMSUBVMB60_...m100]	
0040112D	E8 00FFFF	CALL [XMSUBVMB60_...m100]	
00401132	00 0000	ADD BYTE PTR DS:[ESI],AL	
00401134	00 0000	ADD BYTE PTR DS:[ESI],AL	
00401136	00 0000	ADD BYTE PTR DS:[ESI],AL	
00401138	00 0000	XOR BYTE PTR DS:[ESI],AL	
0040113A	00 0000	ADD BYTE PTR DS:[ESI],AL	
0040113C	40	INC EAX	
0040113D	00 0000	ADD BYTE PTR DS:[ESI],AL	
0040113F	00 0000	ADD BYTE PTR DS:[ESI],AL	
00401141	00 0000	ADD BYTE PTR DS:[ESI],AL	
00401143	0076 2C	ADD BYTE PTR DS:[ESI+2C],DH	

همانطور که می بینید در ویژوال بیسیک Entry Point فایل ما درست زیر Jump Table قرار دارد.(گاهی اوقات دو بایت "00" بین Entry Point و Jump table فاصله وجود دارد.)

بعد از Entry Point دستور Call ThunRTMain وجود دارد. با اجرا شدن دستور Call ThunRTMain برنامه اجرا می شود. در تمامی برنامه های ویژوال بیسیک Entry point ما یک دستور مثل این هست:

Push xxxxxxxx

در مثال ما Entry Point برابر است با:

Push 401264

اگر در پنجره Dump به آدرس 401264 بروید، خواهید دید که در این آدرس رشته "VB5!" وجود دارد.

اصولا آنیک کردن برنامه های ویژوال بیسیک آسانتر است، چرا که درست زیر سر Jump Table، Entry Point، (در مورد فایل پک شده OEP) قرار دارد. پس یافتن OEP در اینجور برنامه ها خیلی راحت تر است. از طرفی چون IAT ما فقط از MSVBVM60.dll تشکیل شده، لذا ساخت دوباره IAT هم آسانتر است. و همچنین اگر پکر ما تعدادی از بایت های ما را دزدیده باشد، خیلی راحت تر می شود آن بایت ها را حدس زد ☺

۳-۲ : دلفی

فایل زیر را در OllyDBG باز کنید:

Targets__Examples__\Language Structure\Borland Delphi\Delphi.exe

0044CA94	90	NOP
0044CA95	C8	DB C8
0044CA96	44	DB 44
0044CA97	00	DB 00
0044CA98	55	PUSH EBP
0044CA99	8BEC	MOV EBP,ESP
0044CA9B	33C4 F0	ADD ESP,-10
0044CA9E	B8 B8C84400	MOV EAX,0044C8B8
0044CAA3	E8 2091FBFF	CALL 00405BC8
0044CAAB	A1 B8DF4400	MOV EAX,DWORD PTR DS:[44DFB8]
0044CAAD	3B00	MOV EAX,DWORD PTR DS:[EAX]
0044CAAF	E8 9CE6FFFF	CALL 0044B150
0044CAB4	3B00 94E04400	MOV ECX,DWORD PTR DS:[44E094]
0044CABA	A1 B8DF4400	MOV EAX,DWORD PTR DS:[44DFB8]
0044CABF	3B00	MOV EAX,DWORD PTR DS:[EAX]
0044CAC1	3B15 F0C64400	MOV EDX,DWORD PTR DS:[44C6F0]
0044CAC7	E8 9CE6FFFF	CALL 0044B168
0044CACC	A1 B8DF4400	MOV EAX,DWORD PTR DS:[44DFB8]
0044CAD1	3B00	MOV EAX,DWORD PTR DS:[EAX]
0044CAD3	E8 10E7FFFF	CALL 0044B1E8
0044CAD8	E8 4372FBFF	CALL 00403D20
0044CADD	3D40 00	LEA EAX,DWORD PTR DS:[EAX]
0044CAE0	0000	ADD BYTE PTR DS:[EAX],AL
0044CAE2	0000	ADD BYTE PTR DS:[EAX],AL
0044CAE4	0000	ADD BYTE PTR DS:[EAX],AL
0044CAE6	0000	ADD BYTE PTR DS:[EAX],AL
0044CAE8	0000	ADD BYTE PTR DS:[EAX],AL
0044CAEA	0000	ADD BYTE PTR DS:[EAX],AL
0044CAEC	0000	ADD BYTE PTR DS:[EAX],AL
0044CAEE	0000	ADD BYTE PTR DS:[EAX],AL
0044CAF0	0000	ADD BYTE PTR DS:[EAX],AL
0044CAF2	0000	ADD BYTE PTR DS:[EAX],AL
0044CAF4	0000	ADD BYTE PTR DS:[EAX],AL
0044CAF6	0000	ADD BYTE PTR DS:[EAX],AL
0044CAF8	0000	ADD BYTE PTR DS:[EAX],AL

معمولا دو دستور اول در برنامه های دلفی این است:

Push EBP

MOV EBP, ESP

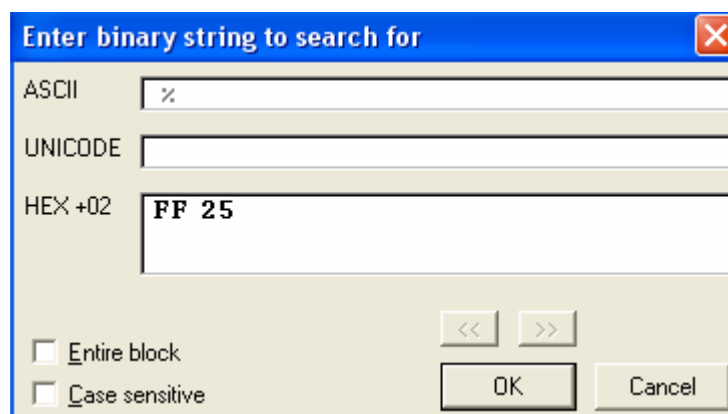
و چند خط بعد از آن تابع GetModuleHandleA اجرا می شود. در تصویر بالا در داخل تابعی که در خط 44CAA3 قرار دارد، تابع GetModuleHandleA قرار دارد.

00405BB7	74 DB	TEST EAX,EAX
00405BB8	C3	JE SHORT 00405B96
00405BBC	B8 A4D04400	MOV EAX,0044D0A4
00405BC1	E8 06F8FFFF	CALL 004053CC
00405BC6	C3	RET
00405BC7	90	NOP
00405BC8	53	PUSH EAX
00405BC9	8BDB	MOV EBX,EAX
00405BCB	33C0	XOR EAX,EAX
00405BCD	A3 9CD04400	MOV DWORD PTR DS:[44D09C],EAX
00405BD2	6A 00	PUSH 0
00405BD4	E8 2BFFFFFF	CALL <JMP.&kernel32.GetModuleHandleA>
00405BD9	A3 64F64400	MOV DWORD PTR DS:[44F664],EAX
00405BDE	A1 64F64400	MOV EAX,DWORD PTR DS:[44F664]
00405BE3	A3 A8D04400	MOV DWORD PTR DS:[44D0A8],EAX
00405BE8	33C0	XOR EAX,EAX
00405BEA	A3 ACD04400	MOV DWORD PTR DS:[44D0AC],EAX
00405BEF	33C0	XOR EAX,EAX
00405BF1	A3 B0D04400	MOV DWORD PTR DS:[44D0B0],EAX
00405BF6	E8 C1FFFFFF	CALL 00405B8C
00405BF8	BA A4D04400	MOV EDI,0044D0A4
00405C00	3BC3	MOV EAX,EBX
00405C02	E8 51DFFFFFF	CALL 00403B58
00405C07	5B	POP EBX
00405C08	C3	RET
00405C09	8D40 00	LEA EAX,DWORD PTR DS:[EAX]
00405C0C	55	PUSH EBP
00405C0D	8BEC	MOV EBP,ESP
00405C0F	33C0	XOR EAX,EAX
00405C11	55	PUSH EBP
00405C12	68 315C4000	PUSH 00405C31
00405C17	64:FF30	PUSH DWORD PTR FS:[EAX]
00405C1A	64:9920	MOV DWORD PTR FS:[EAX],ESP
00405C1D	FFB5 68F64400	INC DWORD PTR DS:[44F668]

دلفی به داشتن Call های زیاد معروف هست. همچنین معمولا در آخر بعضی از توابع در دلفی ۲ Return وجود دارد.

0044B298	. 8B45 FC	MOV EAX,DWORD PTR SS:[EBP-4]
0044B29D	. E8 4B000000	CALL 0044B2E8
0044B2A2	> 8B45 FC	CALL 004038C8
0044B2A5	. 80B8 9C000000 00	MOV EAX,DWORD PTR SS:[EBP-4]
0044B2AC	> 74 BF	CMP BYTE PTR DS:[EAX+9C],0
0044B2AE	> 33C0	JE SHORT 0044B260
0044B2B0	. 5A	XOR EAX,EAX
0044B2B1	. 59	POP EDX
0044B2B2	. 59	POP ECX
0044B2B3	. 64:8910	POP EBX
0044B2B6	. 68 CDB24400	MOV DWORD PTR FS:[EAX],EDX
0044B2BB	> 8B45 FC	PUSH 0044B2CD
0044B2BE	. C680 A5000000 00	MOV EAX,DWORD PTR SS:[EBP-4]
0044B2C5	. C3	MOV BYTE PTR DS:[EAX+A5],0
0044B2C6	> E9 4985FBFF	RET
0044B2CB	> EB EE	JMP 00403814
0044B2CD	> 5F	JMP SHORT 0044B2B8
0044B2CE	. 5E	POP EDI
0044B2CF	. 5B	POP ESI
0044B2D0	. 59	POP EBX
0044B2D1	. 5D	POP ECX
0044B2D2	. C3	POP EBP
0044B2D3	. 90	RET
0044B2D4	. E8 C711FCFF	NOP
0044B2D9	. 84C0	CALL 0040C4A0
0044B2DB	> 74 07	TEST AL,AL
0044B2DD	. 59	JE SHORT 0044B2E4

بهتر است نگاهی به Jump Table هم بیندازیم. برای پیدا کردن Jump Table هم به ابتدا (خط 00401000) بروید. و کلیدهای CTRL + B را بزنید و در پنجره باز شده، مقدار FF25 را تایپ کنید:



004011DC	. C9304000	MOV EAX,DWORD PTR DS:[&kernel32.GetStdHandle]	kernel32.GetStdHandle
004011E0	. DB 11	MOV EAX,EAX	
004011E1	. 54 49 6E 74 65 72 66	ASCII "TInterfacedObjec"	
004011F1	. 8BC0	ASCII "t"	
004011F2	. FF25 9C014500	JMP DWORD PTR DS:[&kernel32.RaiseException]	kernel32.RaiseException
004011FA	. 8BC0	JMP DWORD PTR DS:[&kernel32.RtlUnwind]	ntdll.RtlUnwind
004011FC	. FF25 94014500	JMP DWORD PTR DS:[&kernel32.UnhandledExceptionFilter]	kernel32.UnhandledExceptionFilter
00401204	. 8BC0	JMP DWORD PTR DS:[&kernel32.WriteFile]	kernel32.WriteFile
0040120C	. FF25 90014500	JMP DWORD PTR DS:[&user32.CharNextA]	user32.CharNextA
00401212	. 8BC0	JMP DWORD PTR DS:[&kernel32.ExitProcess]	kernel32.ExitProcess
00401214	. FF25 80014500	JMP DWORD PTR DS:[&user32.MessageBoxA]	user32.MessageBoxA
0040121A	. 8BC0	JMP DWORD PTR DS:[&kernel32.FindClose]	kernel32.FindClose
0040121C	. FF25 80014500	JMP DWORD PTR DS:[&kernel32.FindFirstFileA]	kernel32.FindFirstFileA
00401222	. 8BC0	JMP DWORD PTR DS:[&kernel32.FreeLibrary]	kernel32.FreeLibrary
00401224	. FF25 7C014500	JMP DWORD PTR DS:[&kernel32.GetCommandLineA]	kernel32.GetCommandLineA
0040122C	. 8BC0	JMP DWORD PTR DS:[&kernel32.GetLocaleInfoA]	kernel32.GetLocaleInfoA
00401232	. FF25 74014500	JMP DWORD PTR DS:[&kernel32.GetModuleFileNameA]	kernel32.GetModuleFileNameA
00401234	. 8BC0	JMP DWORD PTR DS:[&kernel32.GetModuleHandleA]	kernel32.GetModuleHandleA
0040123A	. FF25 6C014500	JMP DWORD PTR DS:[&kernel32.GetProcAddress]	kernel32.GetProcAddress
0040123C	. 8BC0	JMP DWORD PTR DS:[&kernel32.GetStartupInfoA]	kernel32.GetStartupInfoA
00401242	. FF25 60014500	JMP DWORD PTR DS:[&kernel32.GetThreadLocale]	kernel32.GetThreadLocale
00401244	. 8BC0	JMP DWORD PTR DS:[&kernel32.LoadLibraryExA]	kernel32.LoadLibraryExA
0040124A	. FF25 58014500	JMP DWORD PTR DS:[&user32.LoadStringA]	user32.LoadStringA
0040124C	. 8BC0	JMP DWORD PTR DS:[&kernel32.IsTopLevel]	kernel32.IsTopLevel
00401252	. FF25 54014500	JMP DWORD PTR DS:[&kernel32.IsMultiByteWideChar]	kernel32.IsMultiByteWideChar
00401254	. 8BC0	JMP DWORD PTR DS:[&kernel32.IsMultiByteWideChar]	kernel32.IsMultiByteWideChar

همانطور که می بینید در بین Jump ها یک دستور Mov Eax, Eax هم قرار دارد.

۳-۳ : MASM32 و TASM32

فایل زیر را در OllyDBG باز کنید:

Targets__Examples__\Language Structure\MASM & TASM\MASM.exe

همانطور که می بینید برنامه های MASM یا TASM بسیار راحت خوانده می شوند و دیگر خبری از Call های تو در تو و خسته کننده نیست 😊... این خصوصیت برنامه های MASM هست که راحت تر از سایر زبان های برنامه نویسی خوانده می شوند. **معمولا** Entry Point در برنامه های MASM برابر ابتدای سکشن text است.

۳-۴ : ویژوال سی پلاس پلاس

فایل زیر را در OllyDBG باز کنید:

Targets__Examples__\Language Structure\Visual C++\VC++.exe

معمولا برنامه های کمپایل شده در سی پلاس پلاس با این سه دستور شروع می شوند:

Push EBP
MOV EBP, ESP
Push -1

و بعد از این دستورها یک SEH (SE Handler) نصب می شود.

توضیح اضافه: Structured Exception Handler که به طور خلاصه به آن SEH گفته می شود، تابعی است که اگر خطایی در برنامه اتفاق بیفتد، برنامه به آن مراجعه می کند، به امید اینکه این تابع بتواند خطا را برطرف نماید.

در سي پلاس پلاس بعد از SEH معمولاً توابع زیر اجرا مي شوند :

GetVersion يا GetVersionExA (براي به دست آوردن ورژن ويندوز)
 GetCommandLineA (اشاره گري براي رشته Command-Line براي پروسه فعلي برمي گرداند).
 GetStartupInfo (Startup Info) را بازگرداني مي کند.
 GetModuleHandleA (براي ماژول مشخص شده برمي گرداند).

۳-۵: .net

براي اجرا شدن برنامه هاي نوشته شده در VB.net يا VC.net نياز به .net Framework Microsoft وجود دارد. فايل زیر را در OllyDBG باز کنید:

Targets__Examples__\Language Structure\Visual Basic.net\ a1.exe

همانطور که مي بينيد برنامه بدون توقف در Entry Point اجرا شده، به Entry Point برنامه برويد (خط 110027CE) :

00	00	DB 00	
00	00	DB 00	
00	00	DB 00	
00	00	DB 00	
EntryPoint>	FF25 00200011	JMP DWORD PTR DS:[<&mscorlib_CoreExeMain>]	mscorlib_CoreExeMain
00	00	DB 00	
00	00	DB 00	
00	00	DB 00	
00	00	DB 00	
00	00	DB 00	
00	00	DB 00	
00	00	DB 00	
00	00	DB 00	
00	00	DB 00	
00	00	DB 00	

همانطور که مي بينيد در Entry Point برنامه ما تابع CorExeMain صدا زده شده، معمولاً اين تنها Import ماست و IAT ما فقط يك تابع دارد و معمولاً در برنامه هاي .net، IAT ما در شروع سکشن (Section) که در آن Imports قرار دارد، وجود دارد. کافيست Memory Map را باز کنيد (کلید ALT + M)

00100000	00004000			
003710000	00004000			
003720000	00007000			
10000000	00001000	CAPTLib	.text	PE header
10001000	00014000	CAPTLib	.rdata	code
10015000	00003000	CAPTLib	.data	imports, exports
10018000	00007000	CAPTLib	.SHAR DAT	data
1001F000	00001000	CAPTLib	.rsrc	resources
10020000	00001000	CAPTLib	.reloc	relocations
10021000	00002000	al		PE header
11000000	00001000	al	.text	code, imports
11002000	00001000	al	.sdata	data
11006000	00001000	al	.rsrc	resources
11008000	00001000	al	.reloc	relocations
4EC50000	00001000	gdiplus	.text	PE header
4EC51000	00179000	gdiplus	.data	code, imports, exports
4EDCA000	0000D000	gdiplus	Shared	data
4EDD7000	00001000	gdiplus	.rsrc	resources
4EDD8000	00012000	gdiplus	.reloc	relocations
4EDEA000	00009000	gdiplus		PE header
5AD70000	00001000	uxtheme	.text	code, imports, exports
5AD71000	00030000	uxtheme	.data	data
5ADA1000	00001000	uxtheme	.rsrc	resources
5ADA2000	00004000	uxtheme		relocations
5ADA6000	00002000	uxtheme	.reloc	

ابتدای سکشنی که در آن Imports قرار دارد در اینجا برابر 11002000 هست. لذا شروع IAT ما هم همانجاست.

معمولاً برنامه هاي نوشته شده در .net، اگر پک بشوند، بسيار راحت آنپک مي شوند. کافيست فايل خودمان را در Memory Map پيدا کنيم و از آن قسمت از Memory با Lord PE يا PETools دامپ بگيريم.
 البته اين را هم بگويم که پکرها و پروتکتورهايي مخصوص برنامه هاي .net نوشته شده اند. اگر با چنين پکرهايي روبرو باشيم، آنپک کردن به اين راحتی نيست. ولي اگر پک مخصوص برنامه هاي .net نباشيد، کافيست که فايل را Dump بگيريم. همچنين یک نرم افزار براي آنپک کردن برنامه .net به اسم Generic Unpacker .net هست که به طور خودکار فايل ما را در Memory پيدا مي کند و آن را دامپ مي کند. ☺

مقدمه ۴ : آنتی دیباگ

اکثر پکرهای امروزی از روش هایی برای شناسایی دیباگر استفاده می کنند، تا کار آنپک کردن را سخت کنند. یادگیری این روش های آنتی دیباگ و چگونگی مقابله با آن برای آنپک کردن فایل ها لازم است. اصولا روش های آنتی دیباگ زیادی وجود دارد. بعضی از آنها برای شناسایی دیباگر بکار می روند، همانند:

IsDebuggerPresent (این تابع می تواند دیباگر را شناسایی کند. اگر دیباگر وجود داشته باشد، مقدار بازگشتی آن برابر یک است)

FindWindowA (Caption پنجره فعلی را برمی گرداند.)

RTDSC (زمان انجام یک دستور را برمی گرداند.)

شناسایی Breakpoint ها (همانند بررسی Opcode "CC" برای شناسایی Software Breakpoint)

و بعضی هم برای سخت کردن دیباگ برنامه استفاده می شوند:

ایجاد خطای Access Violation on write برای کند کردن دیباگ برنامه

بعد از شناخته شدن دیباگر هر اتفاقی ممکن است بیفتد... هر اتفاقی که نویسنده پکر طراحی کرده است، می افتد:

۱. ممکن است برنامه بسته شود

۲. ممکن است دیباگر بسته شود

۳. ممکن است دیباگر دچار خطا شود

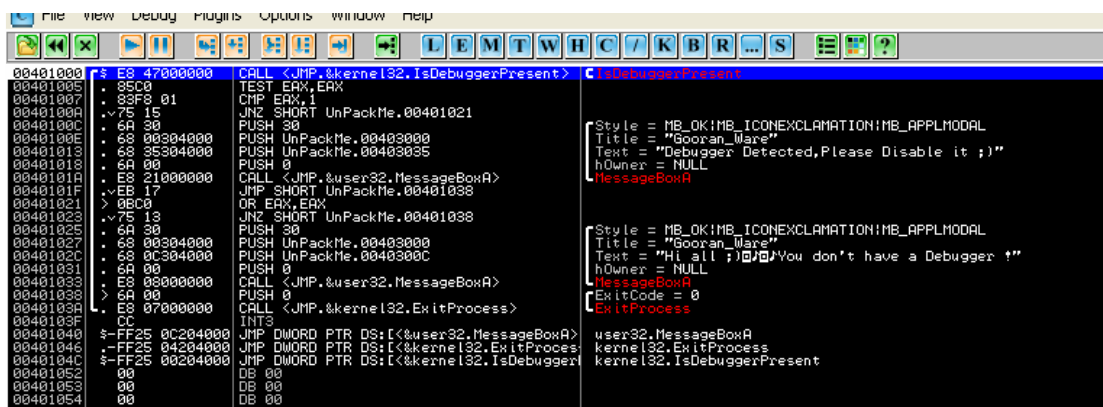
۴. ممکن است پیغامی مبنی بر شناسایی دیباگر داده شود

حتی من یک پکر را دیده بودم که اگر دیباگر را شناسایی می کرد، فایل Boot.ini ویندوز را دستکاری می کرد، طوری که ویندوز دیگر اجرا نمی شد.

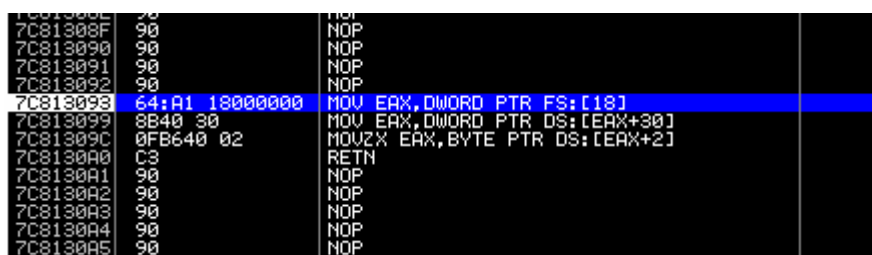
یک پکر دیگر را هم دیده بودم که در صورت شناسایی دیباگر یک فایل Bat از درون خودش Extract می کرد و آن را اجرا می کرد، آن فایل Bat تمامی درایوها به غیر از درایو ویندوز را فرمت می کرد ☹، لذا توصیه می کنم موقع آنپک کردن پکرهای ناشناخته حتما از اطلاعاتی که توی کامپیوترتون دارید، Backup تهیه کنید.

برای آشنایی بیشتر با راه های جلوگیری از شناسایی شدن فایل زیر را در یک OlllyDBG بدون Plug-in باز کنید:

Targets__Examples__\Anti-Debug\Anti-Debug.exe



همانطور که می بینید در اولین خط از تابع IsDebuggerPresent برای شناسایی دیباگر استفاده می شود. حالا چه طور می توانیم کاری کنیم که این تابع را از بین ببریم؟ دوبار کلید F7 را بزنید تا وارد تابع IsDebuggerPresent در فایل Kernel32.dll بشوید.



این تابع اگر دیباگر را شناسایی کند، مقدار یک را برمی گرداند. و در غیر این صورت مقدار صفر را بازمی گرداند. ما می توانیم با دستکاری کدهای این قسمت از فایل Kernel32.dll کاری کنیم که این تابع همواره مقدار صفر را باز گردانی کند، مثلاً کدهای بالا را به این صورت تغییر دهیم:

7C813090	90	NOP
7C813091	90	NOP
7C813092	90	NOP
7C813093	B8 00000000	MOV EAX,0
7C813098	90	NOP
7C813099	90	NOP
7C81309A	90	NOP
7C81309B	90	NOP
7C81309C	90	NOP
7C81309D	90	NOP
7C81309E	90	NOP
7C81309F	90	NOP
7C8130A0	C3	RETN
7C8130A1	90	NOP
7C8130A2	90	NOP
7C8130A3	90	NOP
7C8130A4	90	NOP
7C8130A5	90	NOP

همانطور که می بینید برنامه نتوانسته دیباگر را شناسایی کند:



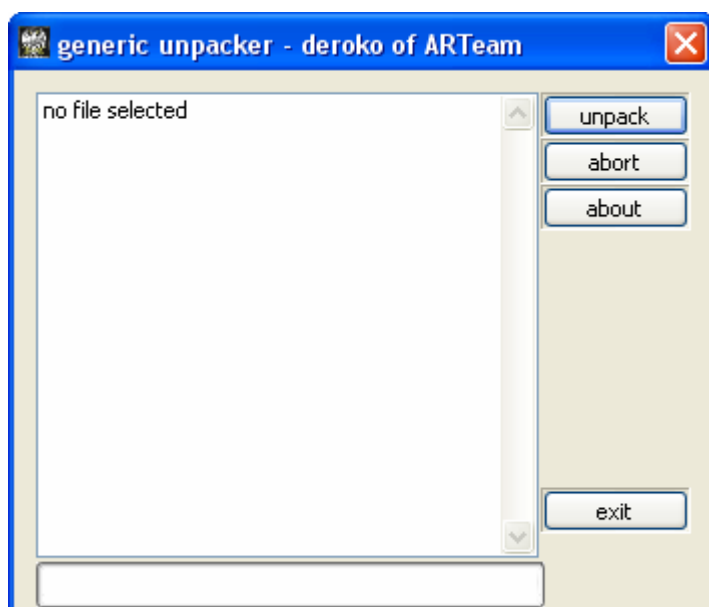
این تغییراتی که ما در فایل Kernel32.dll دادیم، ذخیره نمی شود، چرا که ویندوز اجازه دستکاری فایل های اصلی خود را نمی دهد. پلاگین هایی که برای شناسایی نشدن دیباگر نوشته شده اند هم **تقریباً** یک همچین کاری می کنند. یعنی هر وقت که OllyDBG باز شد، به طور خودکار این تغییرات را در فایل های DLL ویندوز انجام می دهند. بنابراین توصیه ما این است که آخرین پلاگین هایی که OllyDBG را از این توابع پنهان می کنند را داشته باشید.

مقدمه ۵ : Unpackers & OEP finders

برنامه هایی نوشته شده اند که می توانند پکرهای ساده را به طور خودکار آنپک کنند. به این برنامه ها Generic Unpacker می گوئیم. هر کدام از روش های خاصی استفاده می کنند. بعضی از آنها با گذاشتن Memory Breakpoint روی سکشن اصلی OEP را پیدا می کنند و فایل را دامپ می کنند و بعد IAT را دوباره می سازند. در این قسمت با این جور برنامه ها بیشتر آشنا می شویم:

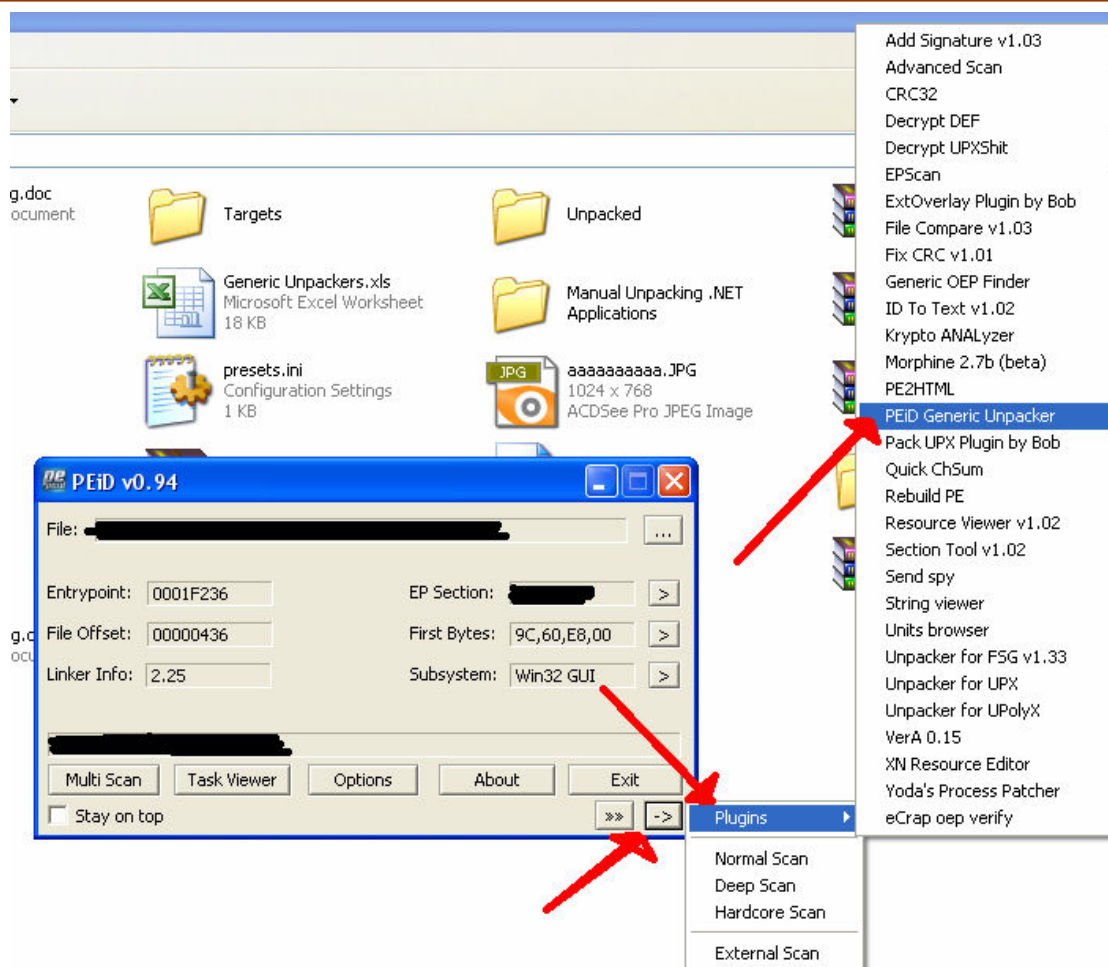
۱-۵ : Dereko Generic Unpacker

این برنامه توسط یکی از اعضای ARTeam به نام Dereko نوشته شده است و بسیار عالی عمل می کند. برای آنپک کردن فایل با استفاده از این برنامه گزینه Unpack را می زنیم، فایل را به برنامه می دهیم و بعد برنامه به طور خودکار فایل را آنپک می کند ☺

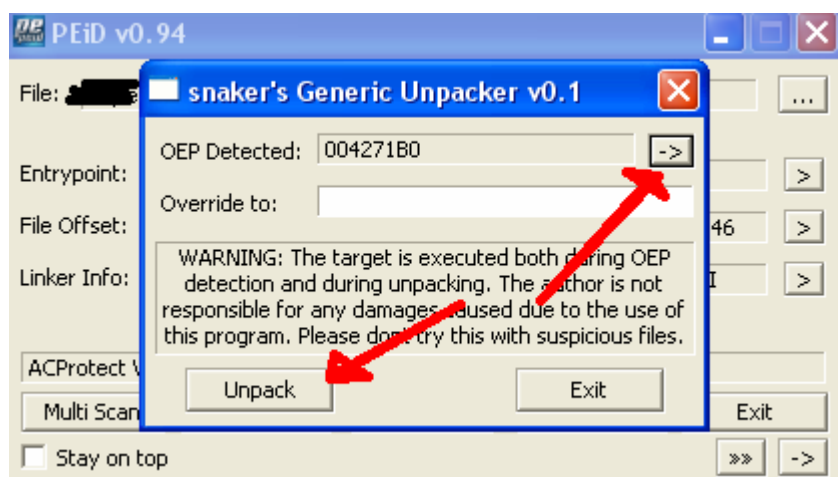


۲-۵ : PEID Generic Unpacker

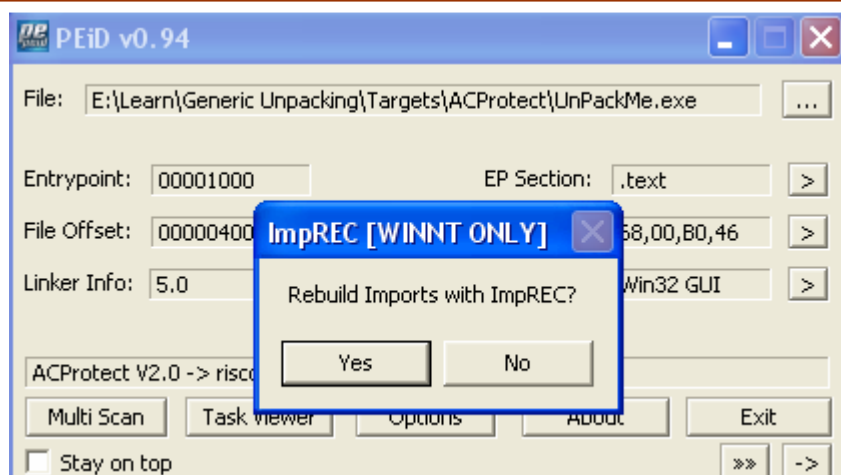
پلاگینی برای برنامه PEID نوشته شده که می تواند پکرهای ساده را آنپک کند، برای این کار بعد از باز کردن برنامه در PEID همانند تصویر عمل کنید:



حالا هم همانند تصویر زیر دکمه کنار OEP Detected را بزنید و بعد گزینه Unpack را بزنید:



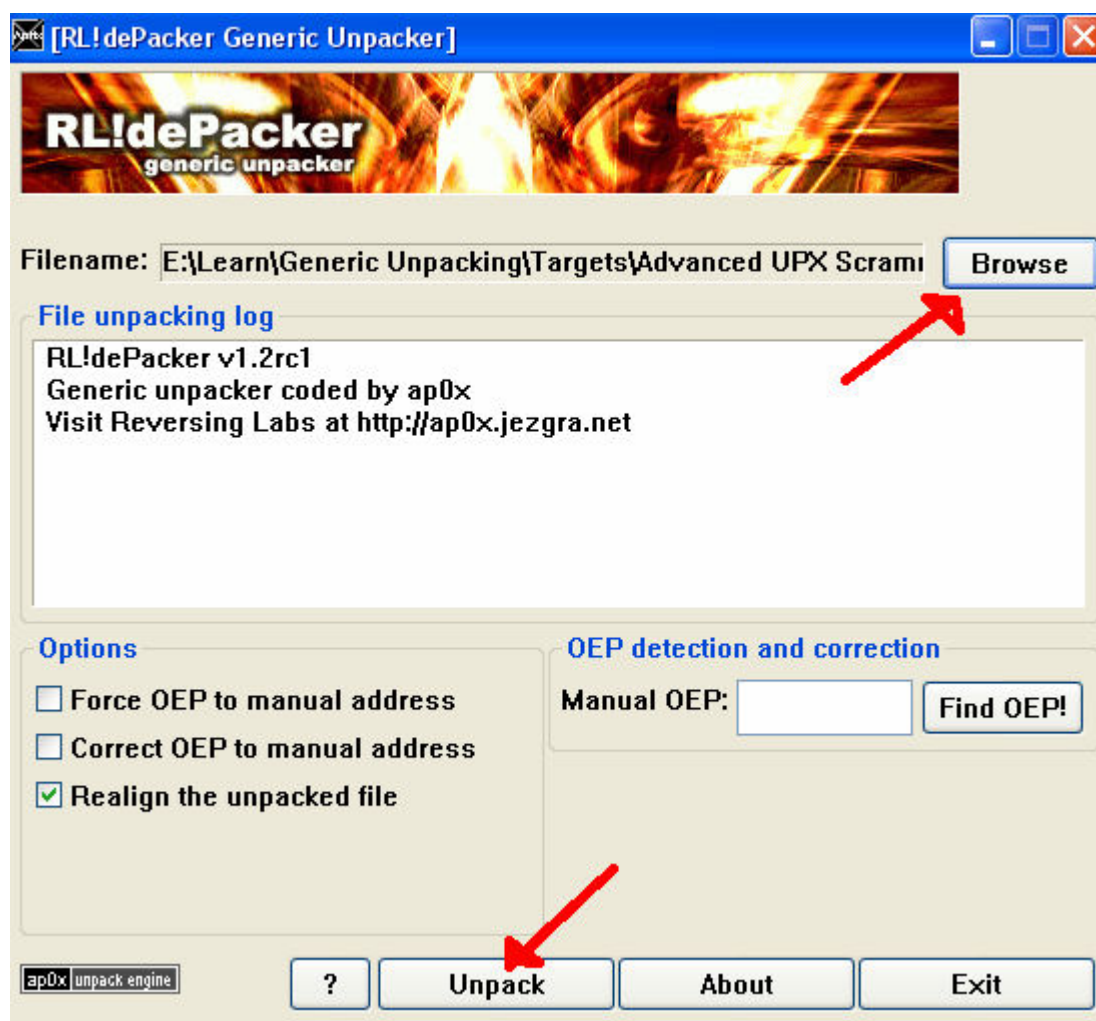
بعد از آن هم برنامه از شما می پرسد آیا می خواهید با استفاده از DLL برنامه ImportREC، IAT برنامه را دوباره بسازم؟ شما گزینه Yes را بزنید تا فایل آنپک شود ©
اگر بعد از زدن دکمه Yes پیغامی داده شده، یعنی برنامه آنپک نشد ©



۳-۵ : RL!dePacker

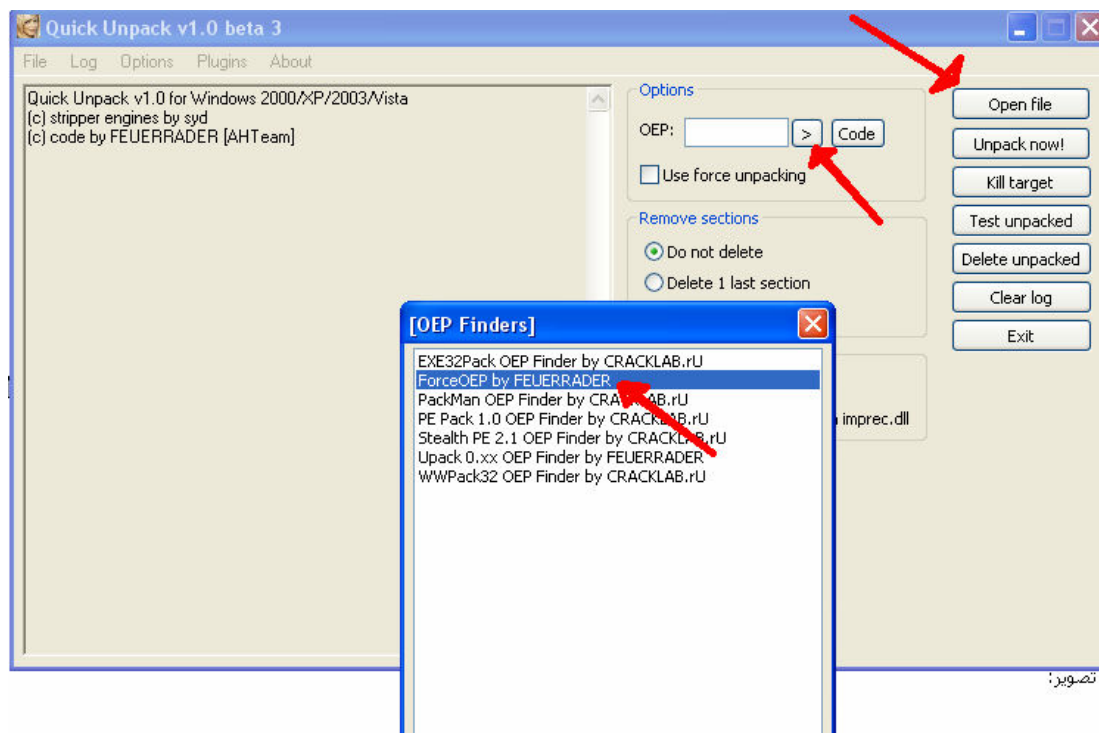
این هم یک Generic Unpacker دیگر هست که یه خرده قاطیه ☺...گاهی وقت ها یک فایل بهش می دهی، برات در عرض سه سوت آپک می کنه...حالا اگر یک بار دیگر همون فایل رو بهش بدی، نمی تونه آپک کنه ☹

برای آپک کردن یک فایل، همانند تصویر ابتدا آن را به برنامه بدهید، و بعد گزینه Unpack را بزنید :



Quick Unpack : ۴-۵

این هم یک Generic Unpacker دیگر هست، که برای آنپک کردن فایل از طریق این برنامه ابتدا فایل را در برنامه باز کنید و بعد باید OEP را پیدا کنید، همانند تصویر:



و بعد گزینه Unpack now را بزنید و بعد گزینه Save (اگر توابع Invalid داشتیم اول گزینه Delete Invalid و بعد Save را بزنید)، تا به این ترتیب فایل آنپک شود.

VMUnpacker : ۵-۵

این هم Generic Unpacker بعدی، که البته نمی توان بهش گفت Generic Unpacker، چون اول نوع پکر را شناسایی می کند و بعد هم با توجه به نوع پکر، اقدام به آنپک کردن آن می کند. برای آنپک کردن فایل با استفاده از این برنامه ابتدا برنامه را باز می کنید، فایل را به برنامه می دهید و بعد گزینه Unpack را می زنید. (در صورتی که برنامه نتواند فایل را آنپک کند، گزینه Unpack غیرفعال است)

۶-۵ : نتیجه تست

تمام این ۵۲ پکری که قرار هست در ادامه مقاله آنپک کنیم را با این استفاده از این ۵ نرم افزار تست کردم، این هم نتیجه تست:

نام پکر	Dereko	RL!DePacker	PEID	Quick Unpack	VMUnpacker
Acprotect			✓		
Advanced UPX Scrambler	✓	✓			
AHPacker	✓	✓			✓
ARM Protector		✓		✓	
Armadillo					
ASDPack	✓				
AsPack	✓		✓	✓	✓
Asprotect					
BamBam	✓	✓	✓	✓	✓
Crunch	✓			✓	
CrypKey SDK	✓	✓	✓	✓	
Enigma					
EP! ExePack					
Exe32Pack					✓

ExeCryptor	✓				
ExeShield			✓		
eXpressor	✓				
EZIP	✓	✓	✓		✓
Fearz Packer	✓				
FSG	✓			✓	✓
Fusion	✓			✓	
Gooran_Ware UnpackMe#1	✓				
JdPack	✓	✓			✓
KByS Packer	✓		✓	✓	✓
Mario Pack					
MEW	✓	✓	✓		✓
mJo0T MIV InfoViewer	✓	✓			
MoleBox					
Morphine	✓				
NeoLite	✓				
NoName Packer	✓	✓	✓	✓	
NoName Packer 2	✓				
nPack	✓	✓			
NSPack			✓	✓	✓
Orien	✓				
PCGuard	✓		✓		
PEBundle	✓		✓		
PECompact	✓				
PESpin					
PeTite		✓	✓		✓
PeX					
PKLite	✓		✓	✓	✓
PolyEnE	✓		✓	✓	✓
SLVc0deProtector			✓		
SPLayer	✓	✓		✓	
TeLock					✓
Themida					
UPolyX			✓		
UPX	✓	✓	✓		✓
WinUPack	✓	✓		✓	✓
XXPack	✓		✓	✓	
Yoda's Protector					

همانطور که می بینید Derekko Generic Unpacker توانسته ۳۳ مورد از ۵۲ فایل را آنپک کند، و از بین این چهار برنامه بهتر عمل کرده است، و بعد از آن برنامه های زیر قرار دارند:

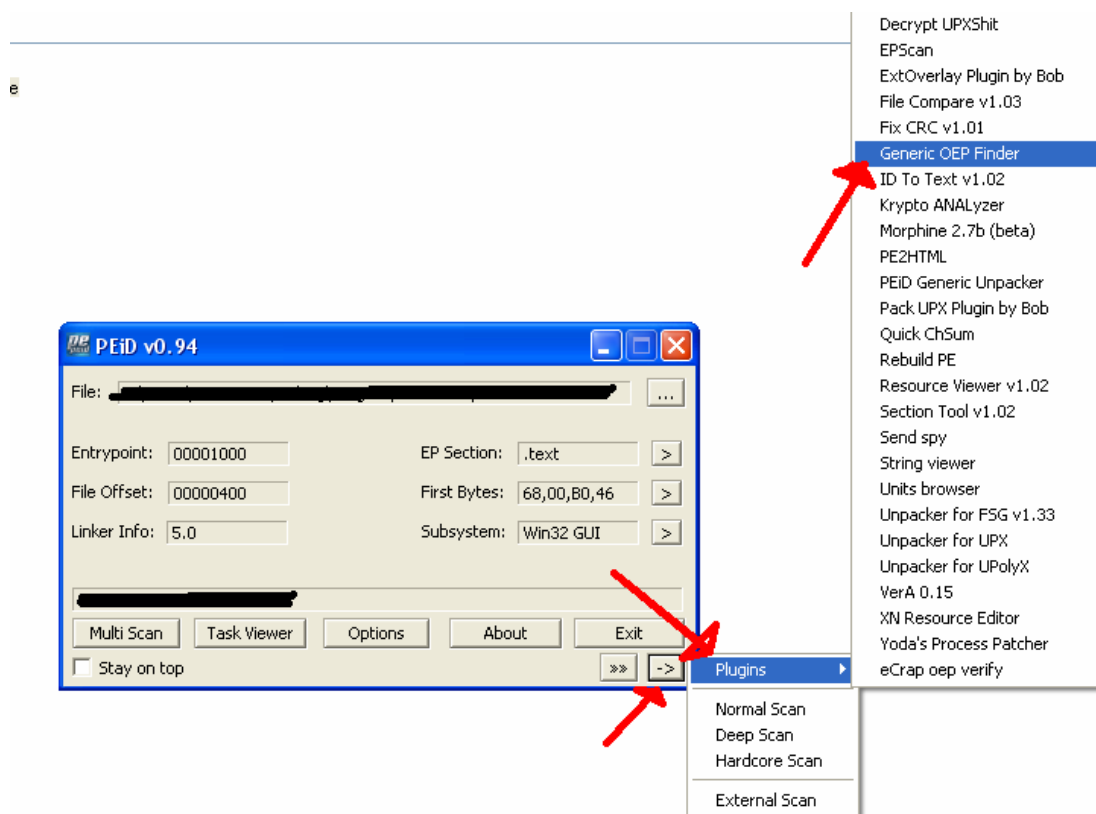
۲. PEID Generic Unpacker (۱۹ فایل آنپک کرد)

۳. VMUnpacker (۱۶ فایل آنپک کرد).

۴. Quick Unpack و RL!DePacker (هر کدام ۱۵ فایل را آنپک کردند)

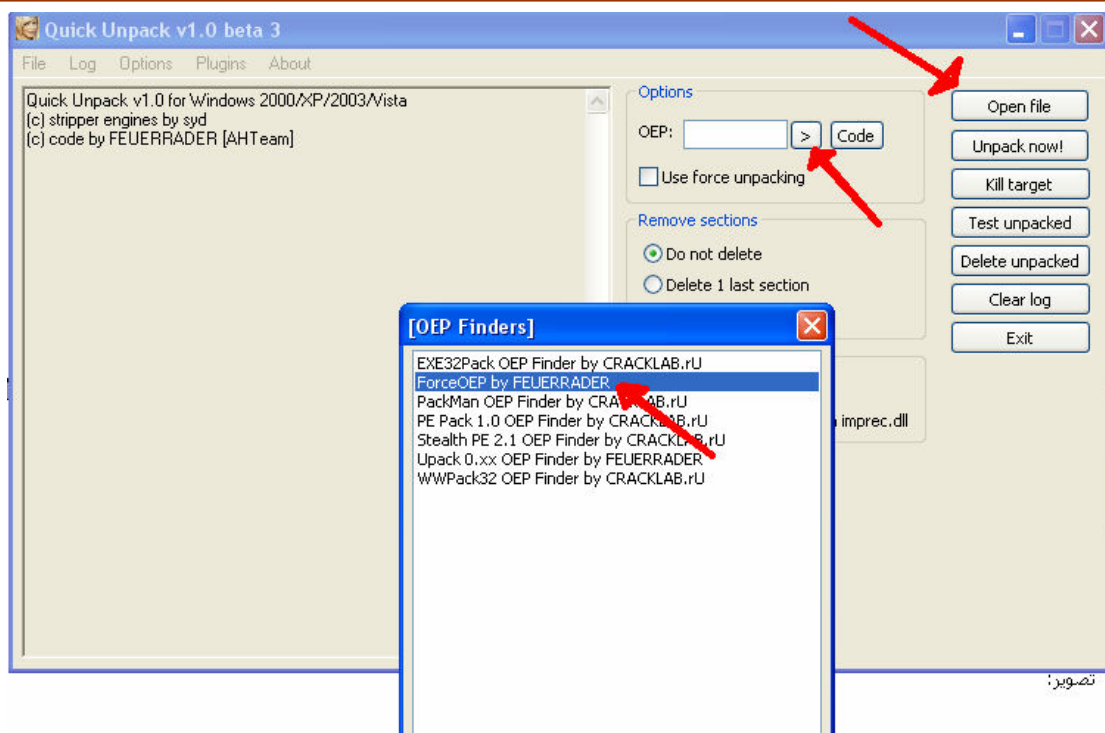
۷-۵ PEID OEP Finder :

یافتن OEP یکی از مهمترین مراحل در آنپک کردن یک فایل است. برای این کار هم برنامه هایی وجود دارند که می توانند OEP برنامه را به طور خودکار پیدا کنند. یکی از آنها پلاگین برنامه PEID است. برای پیدا کردن OEP از طریق این پلاگین همانند تصویر عمل می کنیم:



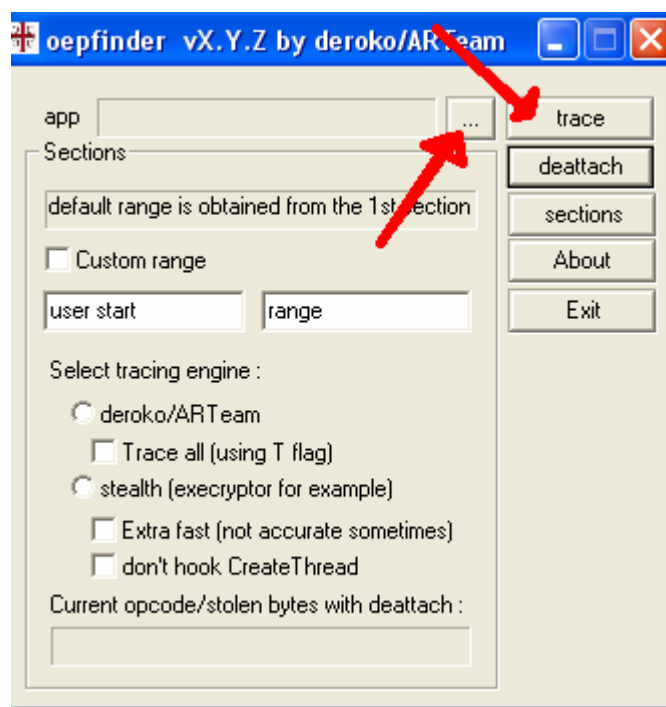
Quick Unpack : ۸-۵

برای یافتن OEP در Quick Unpack همانطور که قبلا گفتم همانند این تصویر عمل می کنیم:

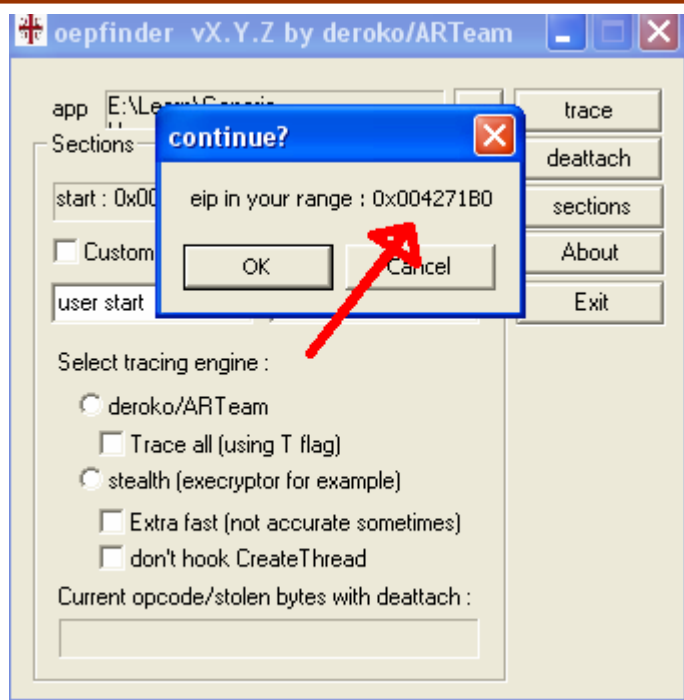


۹-۵ : Dereko OEP finder

این برنامه قابلیت های دیگری به غیر از یافتن OEP هم دارد. اما در اینجا فقط هدف ما یافتن OEP است. بعد از اجرای برنامه همانند تصویر فایل را به برنامه می دهید و گزینه Trace را می زنید:



بعد از مدت کوتاهی، یک همچنین پیغامی داده می شود:



همانطور که می بینید برنامه OEP ما را که 4271B0 بوده را به درستی پیدا کرده است. کلید OK را بزنید تا برنامه باز شود.

۵-۱۰: نتیجه تست :

در قسمت های قبلی سه OEP Finder به شما معرفی کردیم، این هم نتیجه تست گرفته شده از این OEP finder ها:

نام پکر	PEID OEP Finder	Quick Unpack	Dereko OEP Finder
Acprotect	✓	✓	
Advanced UPX Scrambler			✓
AHPacker			✓
ARM Protector	✓	✓	✓
Armadillo			
ASDPack	✓	✓	✓
AsPack	✓	✓	
Asprotect	✓	✓	
BamBam	✓	✓	✓
Crunch	✓	✓	
CrypKey SDK	✓	✓	✓
Enigma	✓	✓	✓
EP! ExePack			
Exe32Pack			
ExeCryptor			✓
ExeShield	✓	✓	✓
eXpressor			
EZIP	✓	✓	
Fearz Packer			
FSG		✓	✓
Fusion	✓	✓	
Gooran_Ware UnpackMe#1	✓		✓

JdPack			✓
KByS Packer	✓	✓	
Mario Pack			
MEW	✓	✓	✓
mJo0T MIV InfoViewer			✓
MoleBox	✓	✓	
Morphine			
NeoLite			
NoName Packer	✓	✓	✓
NoName Packer 2			
nPack			✓
NSPack	✓	✓	
Orien	✓	✓	✓
PCGuard	✓	✓	✓
PEBundle	✓	✓	✓
PECompact	✓	✓	
PESpin			✓
PeTite	✓	✓	
PeX			✓
PKLite	✓	✓	✓
PolyEnE	✓	✓	
SLVc0deProtector			
SPLayer	✓	✓	
TeLock	✓	✓	✓
Themida			
UPolyX	✓	✓	
UPX	✓		✓
WinUPack	✓	✓	
XXPack	✓	✓	✓
Yoda's Protector			

همانطور که می بینید PEID و Quick Unpack بسیار شبیه به هم عمل کردند و فقط PEID در یک مورد موفق تر عمل کرد. پس:

۱. PEID (۳۱ مورد درست)

۲. Quick Unpack (۳۰ مورد درست)

۳. Dereko (۲۵ مورد درست)

۵-۱۱: آپکرهای نرم افزاری

معمولا بعد از اینکه یک پکر یا پروتکتور منتشر می شود، یک آپکر مخصوص آن پکر نوشته می شود. که کار ما را برای کرک کردن برنامه بسیار راحت می کنند. برای به دست آوردن آپکر مورد نظر خود می توانید از طریق گوگل به نتیجه برسید، یا اینکه توی این سایت ها جستجو کنید:

<http://www.tuts4you.com>

<http://www.exetools.com/>

<http://www.woodmann.com/>

<http://www.unpack.cn/>

<http://forums.accessroot.com/>

<http://www.openrce.com/>

<http://www.aoreteam.com/>

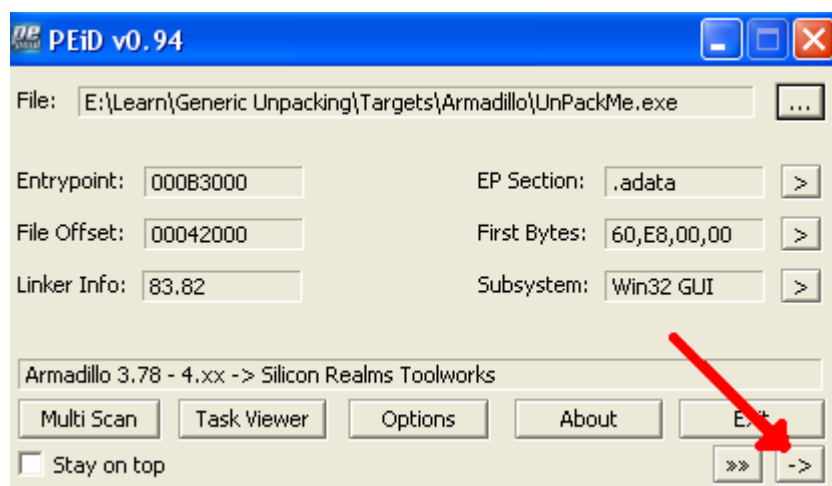
☺ <http://forum.p30world.com/forumdisplay.php?f=133>

مقدمه ۶ : شناسایی پکر

شناسایی پکر اولین قدم برای آنپک کردن یک فایل است. برای شناسایی یک پکر هم تا دلتون بخواد نرم افزار هست ☺
ولی خوب از بین اینها ما فقط پنج مورد را مورد بررسی قرار می دهیم :

۶-۱ : PEiD

این برنامه را همه می شناسند. معروفترین نرم افزار برای شناسایی پکر ☺

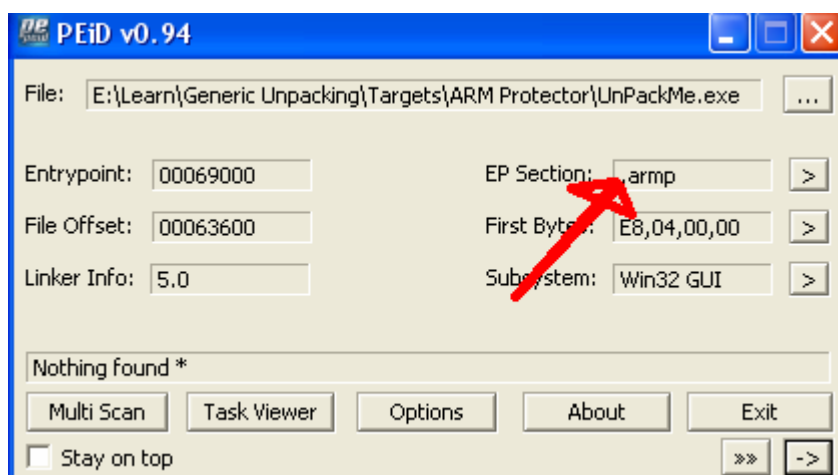


اگر برنامه موفق به شناسایی پکر نشد، می توانید با زدن دکمه نشان داده شده در بالا گزینه های زیر را امتحان کنید:

Deep Scan
Hardcore Scan
External Scan

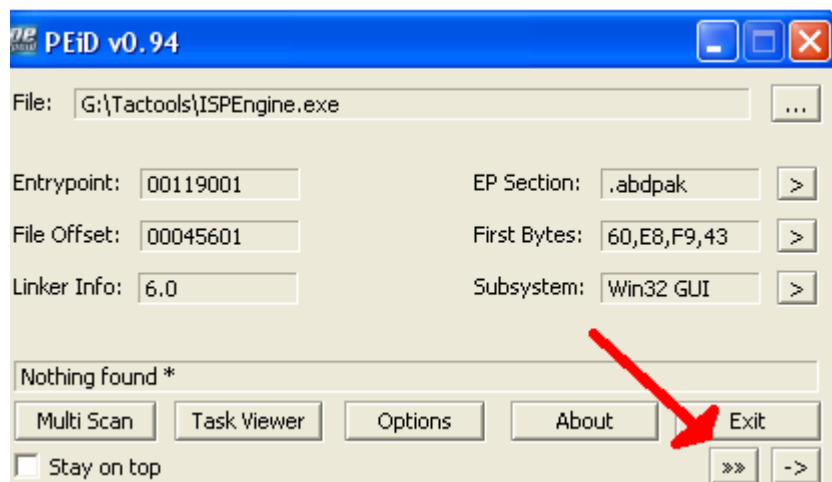
همچنین این برنامه اطلاعات بسیار مفیدی درباره برنامه به ما می دهد ☺

مثلا Entry point فایل ما (Entry point) سکشنی که در آن Entry Point قرار دارد. (EP Section) و همچنین اگر بر روی دکمه کنار آن EP Section کلیک کنید، می توانید تمامی سکشن های برنامه را ببینید. و همچنین اولین دستورات برنامه را هم به ما نشان داده می شود. (بر روی دکمه کنار First Bytes کلیک کنید) اگر برنامه موفق به شناسایی پکر نشد، گاهی اوقات نام Entry point section می تواند راهنمای خوبی برای شناسایی پکر باشد. مثلا تصویر زیر را نگاه کنید، برنامه نتوانسته پکر را نشان دهد. اما نام سکشن armp هست. لذا **ممکن** است پکر ما ARM Protector باشد. ☺

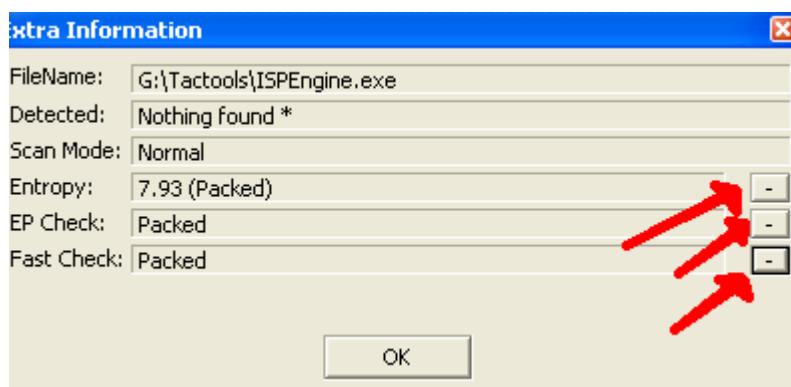


یک نکته را اشاره کنم، اینکه PEiD می گوید: Nothing found دلیل نمی شود که فایل ما حتما با یک پکر ناشناخته پک شده باشد... ممکن است فایل ما در یک زبان برنامه نویسی ناشناخته نوشته شده باشد. پس در این موارد ابتدا باید بفهمیم که آیا اصلا فایل پک شده است یا خیر؟

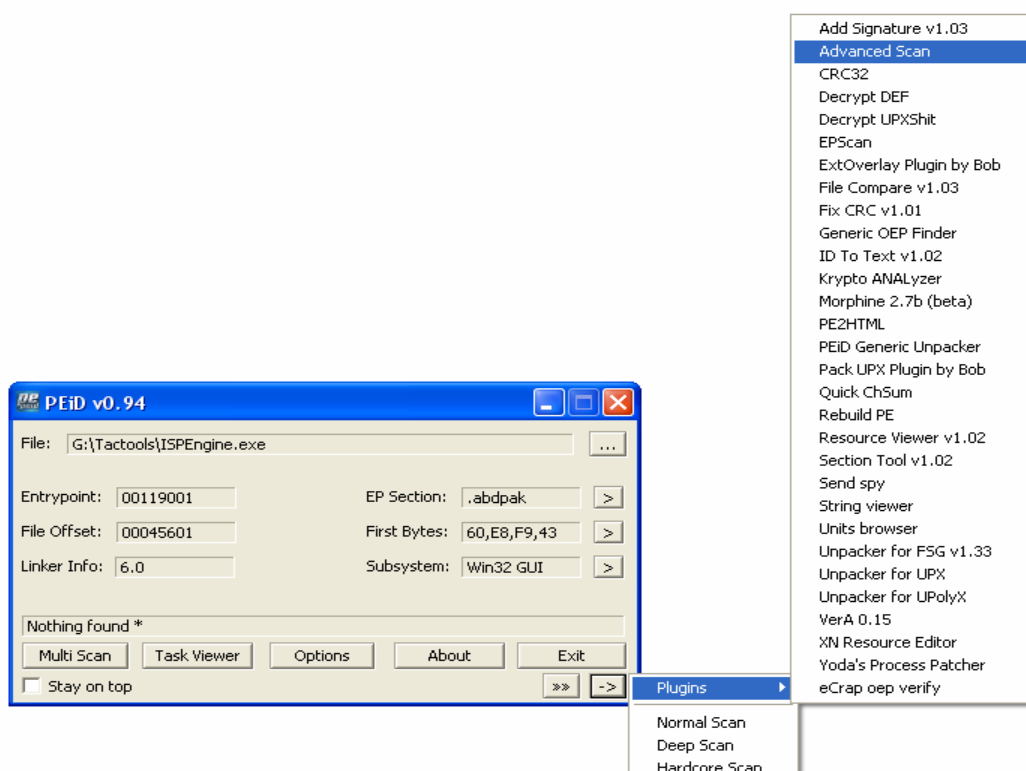
روشی هست که می توان فهمید آیا یک فایل پک شده است یا خیر؟...طبق تست هایی که من انجام دادم، این روش در % 78.6 موارد جواب می دهد. پس زیاد نمی شه بهش اعتماد کرد. برای این کار دکمه نشان داده شده در تصویر را بزنید:



حالا سه دکمه نشان داده شده را می زنیم:



همانطور که می بینید هر سه دکمه به می گویند که فایل پک شده است...لذا فایل به احتمال بسیار زیاد پک شده است. توصیه من این است که از پلاگین Advanced Scan هم استفاده کنید، تا لیست تمامی پکرهای ممکن را ببینید، همانند تصویر:

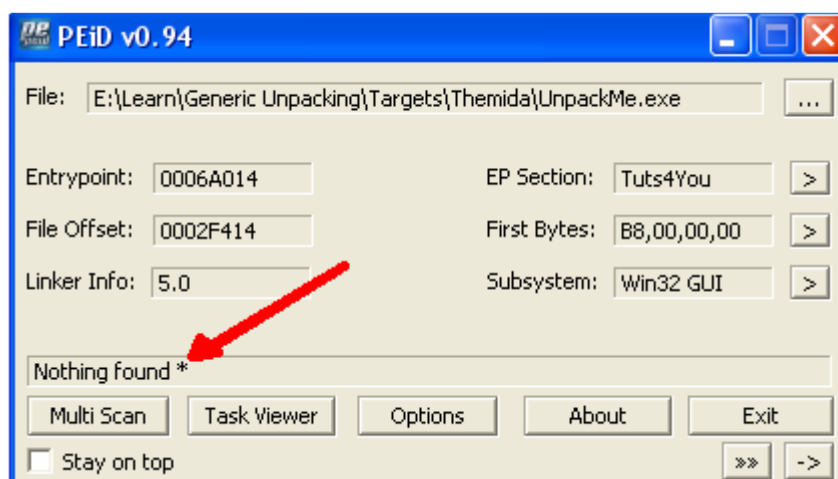


۲-۶ : اضافه کردن Signature

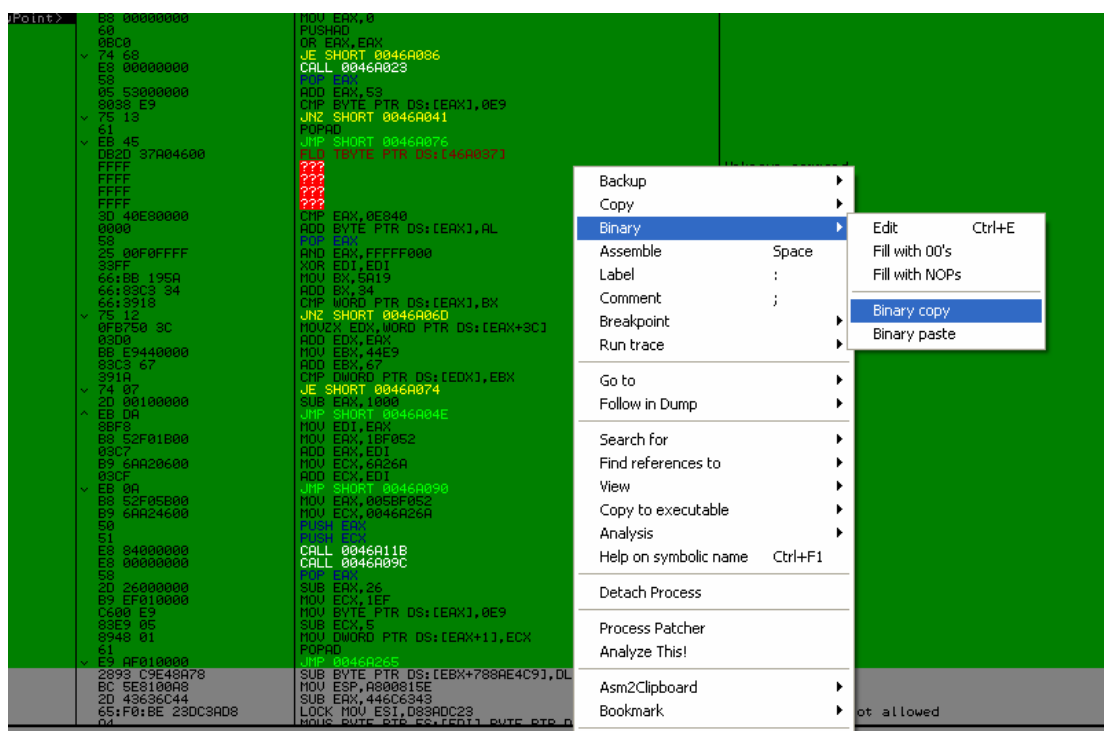
سوالی که اینجا پیش می آید این است که PEID چه طور نوع پکر را تشخیص می دهد؟...ساده است...PEID...دستورات ابتدای فایل یا کل فایل را بررسی می کند. برای مثال اکثر فایل های UPX در ابتدای فایلشان یک همچین دستوری دارند:

```
ModuleEntryPoint> 60          PUSHAD
                   BE 15504400    MOV ESI,00445015
                   8DBE EBBFBFF  LEA EDI,DWORD PTR DS:[ESI+FFFBFBFF]
                   57          PUSH EDI
                   83CD FF      OR EBP,FFFFFFFF
                   EB 10       JMP SHORT 00460CF2
                   90          NOP
                   90          NOP
                   90          NOP
                   90          NOP
                   90          NOP
                   90          NOP
                   8A06       MOV AL,BYTE PTR DS:[ESI]
                   46          INC ESI
                   8B07       MOV BYTE PTR DS:[EDI],AL
```

PEID می آید بررسی می کند اگر ابتدای فایل دستوراتی شبیه این وجود دارد، فایل را UPX تشخیص می دهد. PEID قابلیت اضافه کردن اطلاعات به DataBase خودش را دارد. ما می توانیم به PEID بگوییم که اگر یک همچین دستوراتی در ابتدای فایلی پیدا شد، آن را به نام این پکر شناسایی کند. برای مثال PEID نمی تواند Themida را تشخیص بدهد: ☹



من خود پکر(Themida) را دانلود می کنم و حدود ۱۰ یا ۲۰ فایل را با آن پک می کنم. و بعد دستورات ابتدایی(یا کل دستورات) این فایل ها را با هم مقایسه می کنیم، من یک فایل Themida را در OllyDBG باز کردم. همانند تصویر چند دستور اول را انتخاب می کنیم و آنها را Binary Copy می کنیم.



```

B8 00 00 00 00 60 0B C0 74 68 E8 00 00 00 00 58 05 53 00 00 00 80 38 E9 75 13 61 EB 45 DB 2D 37
A0 46 00 FF FF FF FF FF FF FF 3D 40 E8 00 00 00 00 58 25 00 F0 FF FF 33 FF 66 BB 19 5A 66 83
C3 34 66 39 18 75 12 0F B7 50 3C 03 D0 BB E9 44 00 00 83 C3 67 39 1A 74 07 2D 00 10 00 00 EB DA
8B F8 B8 52 F0 1B 00 03 C7 B9 6A A2 06 00 03 CF EB 0A B8 52 F0 5B 00 B9 6A A2 46 00 50 51 E8 84
00 00 00 E8 00 00 00 00 58 2D 26 00 00 00 B9 EF 01 00 00 C6 00 E9 83 E9 05 89 48 01 61 E9 AF 01
00 00

```

من این بایت های ابتدایی را اینجا Paste کردم، برای ده یا بیست فایل دیگر هم همین کار را می کنم و تمام اطلاعات به دست آمده را با هم مقایسه می کنم. در آخر به نتیجه زیر می رسم:

```

B8 ?? ?? ?? ?? 60 0B C0 74 68 E8 00 00 00 00 58 05 53 00 00 00 80 38 E9 75 13 61 EB 45 DB 2D 37 ?? ?? ?? FF FF FF
FF FF FF FF FF FF FF 3D 40 E8 00 00 00 00 58 25 00 F0 FF FF 33 FF 66 BB 19 5A 66 83 C3 34 66 39 18 75 12 0F B7 50 3C 03
D0 BB E9 44 00 00 83 C3 67 39 1A 74 07 2D 00 10 00 00 EB DA 8B F8 B8 ?? ?? ?? ?? 03 C7 B9 ?? ?? ?? ?? 03 CF EB 0A
B8 ?? ?? ?? ?? B9 ?? ?? ?? ?? 50 51 E8 84 00 00 00 E8 00 00 00 00 58 2D 26 00 00 00 B9 EF 01 00 00 C6 00 E9 83 E9
05 89 48 01 61 E9

```

علامت ؟؟ یعنی این بایت در فایل های مختلف پک شده با Themida متفاوت است. حالا می توانیم این Signature را به PEID اضافه کنیم، برای اینکار فایل userdb.txt که در پوشه PEID قرار دارد را باز می کنیم و اطلاعات زیر را به آن اضافه می کنیم:

]Themida 1.8.x.x > Oreans Technologies[

```

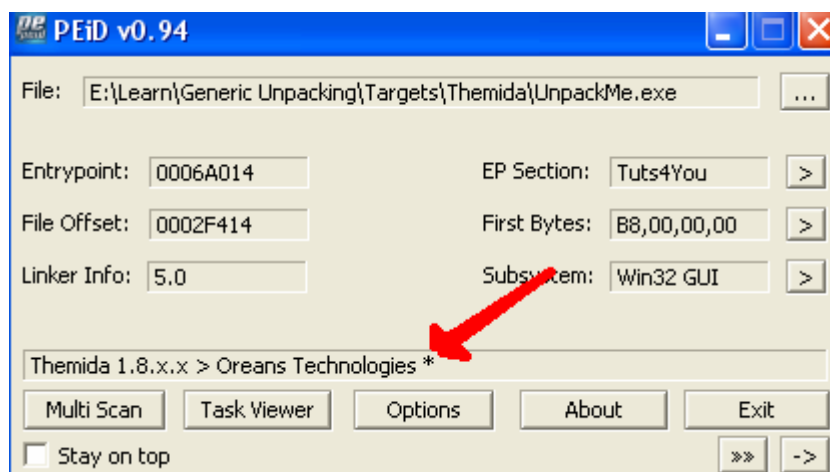
signature=B8 ?? ?? ?? ?? 60 0B C0 74 68 E8 00 00 00 00 58 05 53 00 00 00 80 38 E9 75 13 61 EB 45 DB 2D 37 ?? ??
?? FF FF FF FF FF FF FF 3D 40 E8 00 00 00 00 58 25 00 F0 FF FF 33 FF 66 BB 19 5A 66 83 C3 34 66 39 18 75 12 0F
B7 50 3C 03 D0 BB E9 44 00 00 83 C3 67 39 1A 74 07 2D 00 10 00 00 EB DA 8B F8 B8 ?? ?? ?? ?? 03 C7 B9 ?? ?? ?? ??
03 CF EB 0A B8 ?? ?? ?? ?? B9 ?? ?? ?? ?? 50 51 E8 84 00 00 00 E8 00 00 00 00 58 2D 26 00 00 00 B9 EF 01 00 00 C6
00 E9 83 E9 05 89 48 01 61 E9

```

ep_only=true

همانطور که در بالا می بینید در داخل کروش نام پکر را نوشتم. بعد از Signature، فایل را نوشتم و در کنار ep_only= باید مشخص کنم که آیا برنامه فقط باید در ابتدای فایل دنبال این بایت ها باشد، یا در کل فایل؟ که من نوشتم true، یعنی فقط ابتدای فایل را بررسی کن!

حالا فایلمان را دوباره با PEID بررسی می کنیم:



اینبار برنامه می تواند Themida را تشخیص دهد. علامت * به این معناست که این فایل با استفاده از Signature های خارجی (فایل Userdb.txt) شناسایی شده است.

فایل userdb.txt من حدود ۱۲۰۰ Signature در خود دارد. همچنین اگر پکر جدیدی پیدا کردم سعی می کنم Signature آن را اضافه کنم. برای دانلود آن از لینک زیر استفاده کنید:

<http://subtitle.persianguig.com/userdb.rar>

۳-۴ : RDG Packer Detector

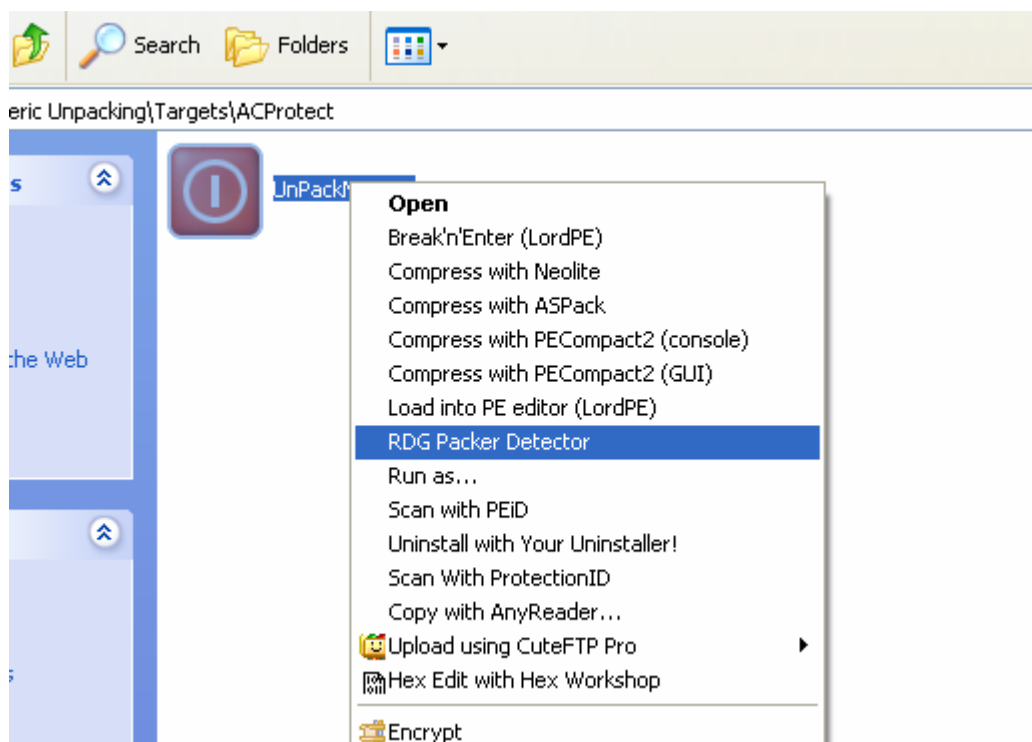
این هم برنامه دیگری برای شناسایی پکر است، بهتر است این تنظیمات را در برنامه RDG Packer Detector انجام دهید:



و بعد :



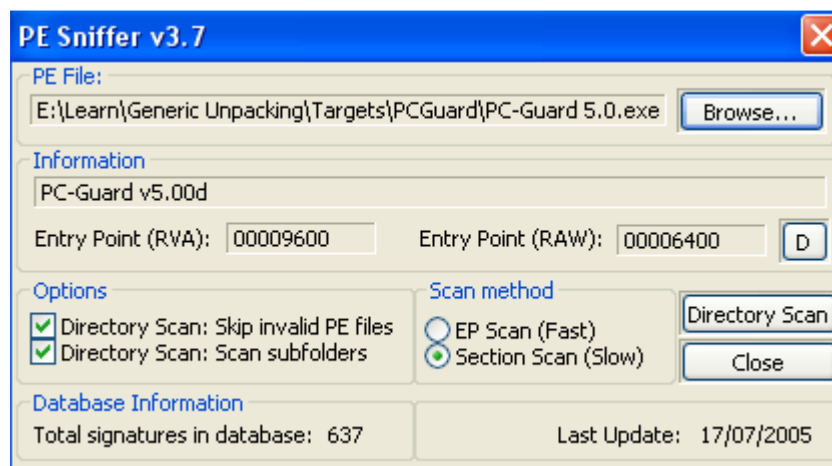
حالا براي بررسی یک فایل از طریق از این برنامه کافست بر روی فایل کلیک راست کرده و گزینه RDG Packer Detector را بزنید:



در قسمت Compilador نام کمپایلر (همان زبان برنامه نویسی) نوشته شده... در بالا برنامه نتوانسته زبان برنامه نویسی را تشخیص بدهد. در قسمت Detectado نام پکر نوشته می شود. در قسمت Posible، پروتکشن هایی که ممکن است فایل داشته باشد، نوشته می شود. مثلاً اگر پکری از تابع IsDebuggerPresent استفاده کرده باشد. آنجا نوشته می شود. در قسمت Contacto نام سایت نویسنده پکر یا پروتکتور نوشته می شود. همچنین اگر برنامه موفق به شناسایی پکر نشد، یعنی نوشته شد "Nada" گزینه M-A را تیک بزنید و سپس گزینه Detectar را بزنید، شاید در این صورت برنامه موفق به تشخیص پکر شد. در ضمن اگر باز هم موفق به شناسایی پکر نشدید. دکمه 3 را بزنید تا دیتابیس خارجی هم بررسی شود، شاید اینجوری نوع پکر شناسایی شد

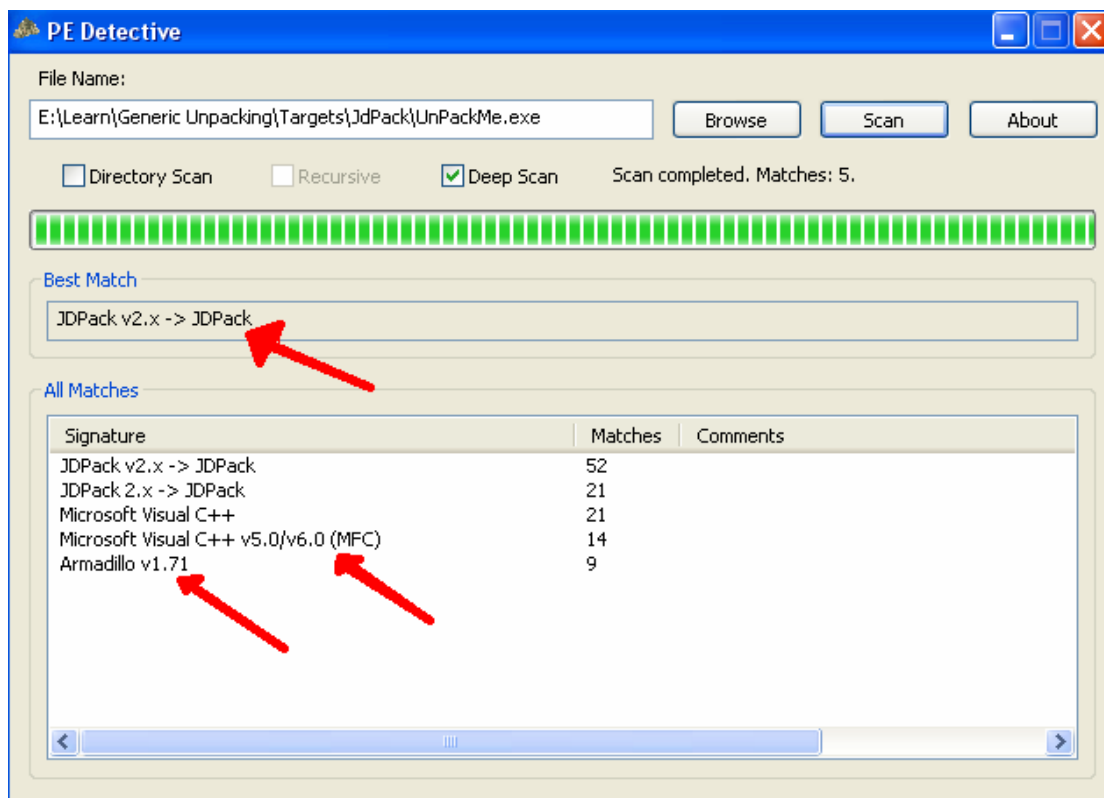
PeTools : ۴-۶

PeTools قابلیت های بسیار زیادی دارد که یکی از قابلیت های آن شناسایی نوع پکر می باشد. برای شناسایی نوع پکر از طریق این برنامه در منوی Tools گزینه PE Sniffer را بزنید، و بعد گزینه Browse را بزنید و فایل خود را انتخاب کنید:



PE Detective : ۵-۶

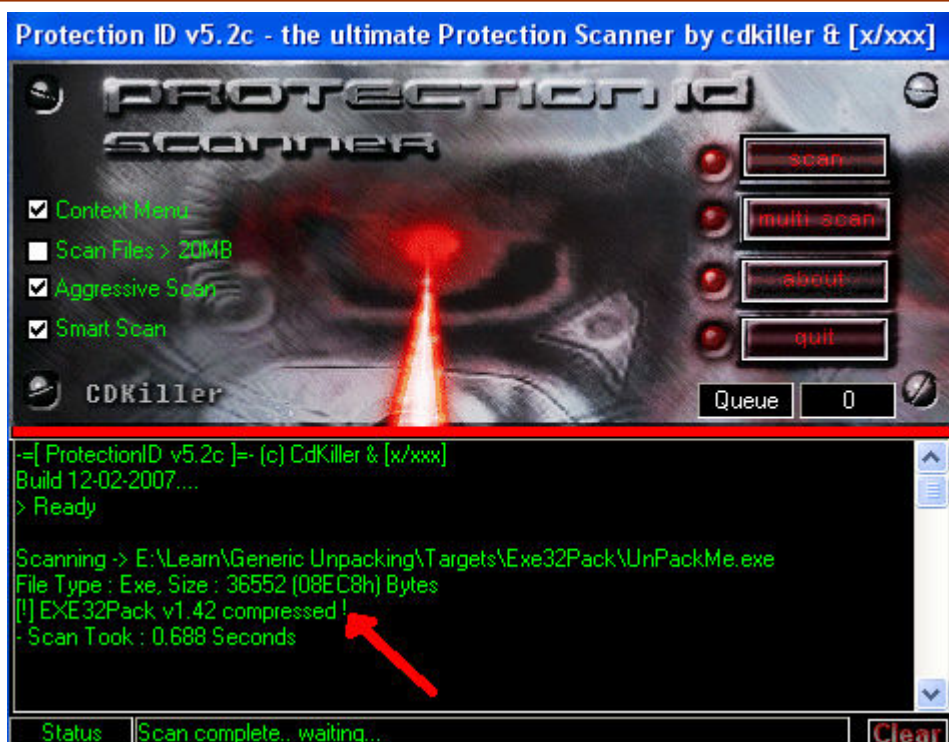
این برنامه به خاطر راحتی کاری که دارد، یکی از برنامه های مورد علاقه من هست. برای بررسی یک فایل با استفاده از این برنامه کفایت گزینه Browse را بزنید فایل را به برنامه بدهید و گزینه Scan را بزنید:



همانطور که می بینید این برنامه می گوید فایل داده شده به احتمال زیاد JdPack می باشد ولی ممکن است Visual C++ یا Armadillo 1.71 هم باشد. با این برنامه می توانید تمام فایل های یک پوشه را هم به طور یکسره بررسی کنید. برای این کار تیک گزینه Directory Scan را بزنید. اگر می خواهید زیر پوشه ها بررسی شوند گزینه Recursive را هم تیک بزنید.

Protection ID : ۶-۶

این هم یک برنامه دیگر، برای شناسایی پکر از طریق این پکر برنامه را اجرا کنید، گزینه Scan را بزنید و فایل را به برنامه بدهید:



همانطور که می بینید فایل ما با Exe32Pack پک شده است. اگر می خواهید با کلیک راست بر روی فایل و انتخاب گزینه "Scan with protection ID" فایل خود را با این برنامه بررسی کنید. کافست تیک گزینه Context menu را بزنید.

۷-۶ : نتیجه تست

هر چند تست گرفته شده نشان دهنده قدرت این ۵ برنامه نیست (چون اگر برنامه ای دیتابیسش آپدیت باشد، می تواند بیشتر پکرها را تشخیص دهد و در غیر این صورت خیر). ولی به هر حال این هم نتیجه تست گرفته شده از این ۵ برنامه :

نام پکر	PEID	RDG	PeTools	PeDetective	Protection ID
Acprotect		✓		✓	✓
Advanced UPX Scrambler		✓			
AHPacker	✓			✓	
ARM Protector		✓	✓		
Armadillo	✓	✓	✓	✓	✓
ASDPack					
AsPack	✓	✓	✓	✓	✓
Asprotect	✓	✓	✓	✓	✓
BamBam					
Crunch	✓	✓	✓	✓	
CrypKey SDK		✓			
Enigma	✓	✓		✓	
EP! ExePack					
Exe32Pack	✓	✓	✓		✓
ExeCryptor		✓		✓	✓
ExeShield	✓				
eXpressor	✓	✓		✓	
EZIP	✓	✓	✓	✓	✓
Fearz Packer			✓		
FSG	✓	✓	✓	✓	✓
Fusion			✓		
Gooran_Ware UnpackMe#1	✓	✓	✓	✓	✓
JdPack		✓		✓	

KByS Packer				✓	
Mario Pack					
MEW	✓	✓	✓	✓	✓
mJo0T MIV InfoViewer		✓			✓
MoleBox		✓	✓		✓
Morphine		✓		✓	
NeoLite	✓	✓	✓	✓	✓
nPack					
NSPack		✓		✓	✓
Orien		✓	✓	✓	
PCGuard	✓	✓	✓	✓	✓
PEBundle	✓	✓	✓		✓
PECompact	✓	✓	✓	✓	✓
PESpin	✓	✓		✓	✓
PeTite	✓	✓	✓	✓	✓
PeX	✓	✓	✓	✓	✓
PKLite	✓	✓	✓	✓	
PolyEnE		✓		✓	
SLVc0deProtector	✓	✓		✓	
SPLayer	✓	✓	✓	✓	
TeLock	✓	✓		✓	
Themida		✓		✓	✓
UPolyX		✓			
UPX	✓	✓	✓	✓	✓
WinUPack	✓	✓		✓	✓
XXPack					
Yoda's Protector	✓	✓	✓	✓	

به این ترتیب RDG با ۳۹ مورد صحیح از ۵۰ مورد بهترین DataBase را داشت. و رتبه های بعدی به ترتیب:

۲. PeDetective (مورد صحیح ۳۳)

۳. PEID (مورد صحیح ۲۷)

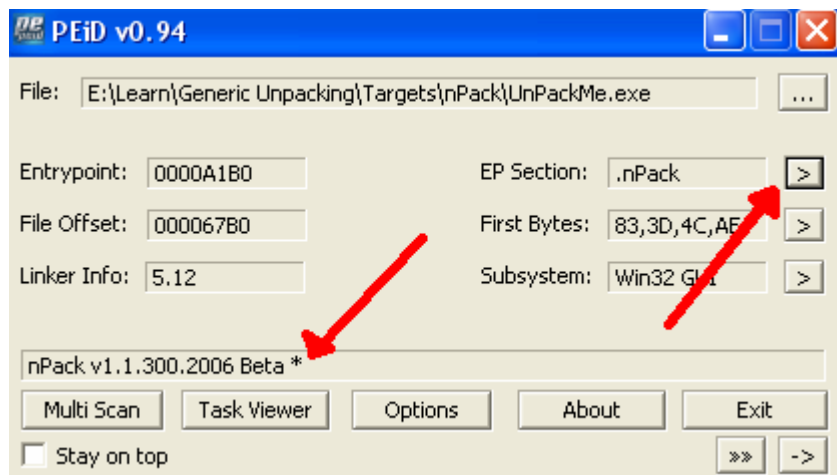
۴. Protection ID و PeTools (هر کدام ۲۳ مورد صحیح)

از اینجا دیگر شروع به آپیک کردن برنامه ها می کنیم:

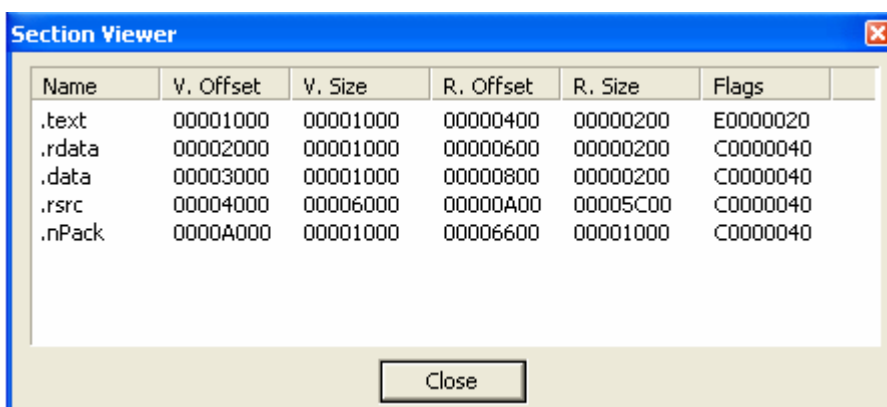
NPack

توضیحی کوتاه در مورد پکر: این پکر توسط NEOx نوشته شده، همان کسی که PeTools را نوشت در کل امکانات خاصی نداره و فقط یک پکر ساده است. مثل UPX

فایل زیر را در PEID باز کنید :



دکمه نشان داده شده در تصویر بالا (کنار EP Section) را بزنید :



همانطور که می بینید ما ۵ سکشن داریم:

.text → 1000 تا 2000
 .Rdata → 2000 تا 3000
 .data → 3000 تا 4000
 .rsrc → 4000 تا A000
 .nPack → A000 تا B000

این اطلاعات بر مبنای RVA است، بهتر است آنها را به اضافه ImageBase کنیم تا اطلاعات را بر مبنای VA به دست آوریم:

.text → 401000 تا 402000
 .Rdata → 402000 تا 403000
 .data → 403000 تا 404000
 .rsrc → 404000 تا 40A000
 .nPack → 40A000 تا 40B000

از بین این ۵ سکشن، دو سکشن Text و nPack برای ما اهمیت بیشتری دارند. چرا که در سکشن Text کدهای اصلی ما وجود دارد و در سکشن nPack کدهای پکر. اطلاعات لازم را به دست آوردیم. فایل خود را در OllyDBG باز کنید:

0040A180	74 65 64 20 69 6E 20 7	ASCII "ted in the dynam"
0040A18C	69 63 20 6C 69 6E 6B 2	ASCII "ic link library"
0040A19C	25 73 2E 00	ASCII "%s.",0
0040A1A0	<ModuleEntryPoint>	833D 4C4E4000 00
0040A1B0	75 05	CMPL DWORD PTR DS:[40AE4C],0
0040A1B4	E9 01000000	JNZ SHORT 0040A1BE
0040A1B8	C3	JMP 0040A1BF
0040A1BC	E8 46000000	RET
0040A1C0	E8 73000000	CALL 0040A20A
0040A1C4	B8 B0A14000	CALL 0040A23C
0040A1C8	2B05 08AE4000	MOV EAX,<ModuleEntryPoint>
0040A1D0	A3 48AE4000	SUB EAX,DWORD PTR DS:[40AE08]
0040A1D4	E8 9C000000	MOV DWORD PTR DS:[40AE48],EAX
0040A1D8	E8 2D020000	CALL 0040A27A
0040A1DC	E8 DD060000	CALL 0040A410
0040A1E0	E8 2C060000	CALL 0040A8C5
0040A1E4	A1 48AE4000	CALL 0040A819
0040A1E8	C705 4C4E4000 01000000	MOV EAX,DWORD PTR DS:[40AE48]
0040A1EC	0105 08AE4000	MOV DWORD PTR DS:[40AE4C],1
0040A1F0	FF35 08AE4000	ADD DWORD PTR DS:[40AE00],EAX
0040A1F4	C3	PUSH DWORD PTR DS:[40AE00]
0040A1F8	C3	RET
0040A200	C3	RET

الان ما در خط 0040A1B0 هستیم، یعنی در سکشن nPack (با توجه به اطلاعات بالا)... ما باید ببینیم که در چه زمانی وارد سکشن اصلی برنامه یعنی Text Section می شویم؟... برنامه را آنقدر با کلید F8 جلو ببرید تا به خط 40A208 برسید... حالا نگاهی به پنجره Stack بندازید:

0012FFC0	00401000	UnPackMe.00401000
0012FFC4	7C816FD7	RETURN to kern
0012FFC8	7C910738	ntdll.7C910738
0012FFCC	FFFFFFFF	
0012FFD0	7FFDE000	
0012FFD4	8054AB38	
0012FFD8	0012FFC8	
0012FFDC	81B4A690	

این یعنی اینکه این Return ما را به خط 401000 می برد. با توجه با اطلاعات بالا خط 401000 در کجا قرار دارد؟... درست است، در Text Section یعنی جایی که کدهای اصلی ما قرار دارد. پس با این Ret وارد Text Section می شویم. یعنی به OEP می رسم.

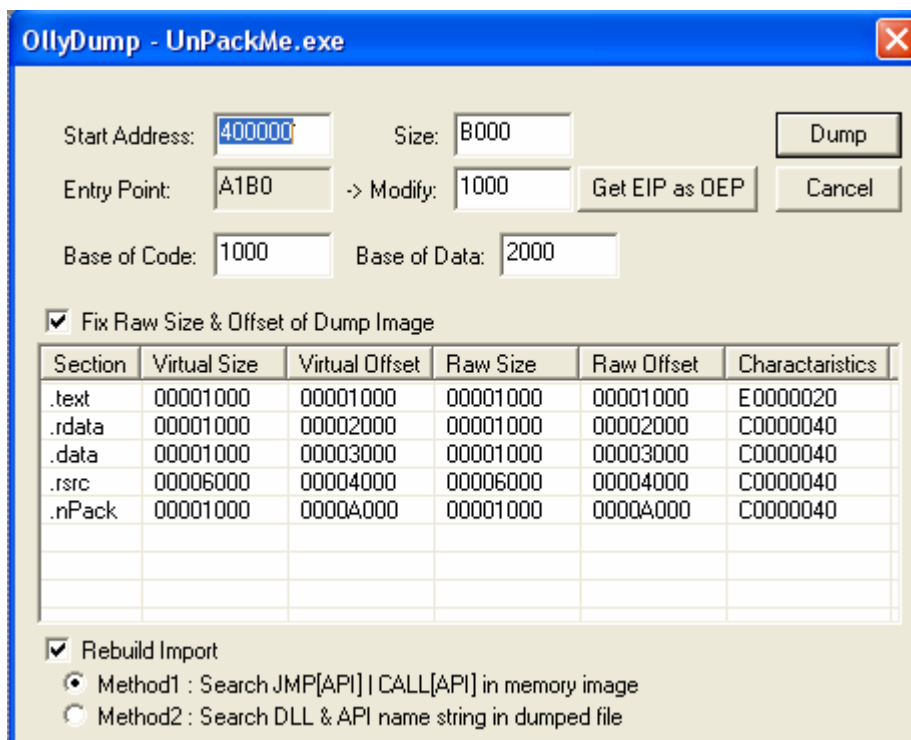
حال کافیه یک بار دیگر کلید F8 را بزنی تا وارد Text Section بشوی:

Address	Hex dump	Disassembly	Comment
00401000	6A	DB 6A	CHAR 'j'
00401001	30	DB 30	CHAR '0'
00401002	68	DB 68	CHAR 'h'
00401003	00	DB 00	
00401004	30	DB 30	CHAR '0'
00401005	40	DB 40	CHAR '@'
00401006	00	DB 00	
00401007	68	DB 68	CHAR 'h'
00401008	0C	DB 0C	
00401009	30	DB 30	CHAR '0'
0040100A	40	DB 40	CHAR '@'
0040100B	00	DB 00	
0040100C	6A	DB 6A	CHAR 'j'
0040100D	00	DB 00	
0040100E	E8	DB E8	
0040100F	07	DB 07	
00401010	00	DB 00	
00401011	00	DB 00	
00401012	00	DB 00	
00401013	6A	DB 6A	
00401014	00	DB 00	CHAR 'j'
00401015	E8	DB E8	
00401016	06	DB 06	
00401017	00	DB 00	
00401018	00	DB 00	
00401019	00	DB 00	
0040101A	FF	DB FF	
0040101B	25	DB 25	CHAR '%'
0040101C	08	DB 08	
0040101D	20	DB 20	
0040101E	40	DB 40	CHAR '@'
0040101F	00	DB 00	
00401020	FF	DB FF	
00401021	25	DB 25	CHAR '%'
00401022	00	DB 00	
00401023	20	DB 20	
00401024	40	DB 40	CHAR '@'
00401025	00	DB 00	
00401026	00	DB 00	

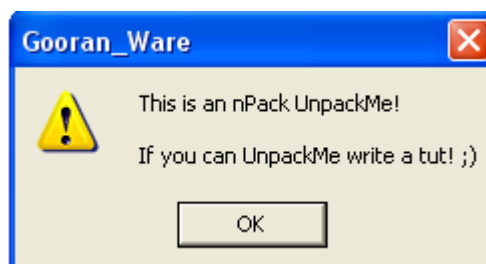
حالا چرا کدهای اصلی برنامه ما اینجوری است؟ به خاطر اینکه OllyDBG به طور پیشفرض فقط کدهای سکشنی که در آن Entry Point قرار دارد را مورد بررسی قرار می دهد. و سکشن های دیگر را آنالیز نمی کند. کلید های CTRL + A را بزنید تا کدهای این قسمت بررسی شود:

Hex dump	Disassembly	Comment
. 6A 30	PUSH 30	Style = MB_OK MB_ICONEXCLAMATION MB_AB
. 68 00304000	PUSH 00403000	Title = "Gooran_Ware"
. 68 00304000	PUSH 00403000	Text = "This is an nPack UnpackMe!",LF
. 6A 00	PUSH 0	hOwner = NULL
. E8 07000000	CALL 0040101A	MessageBoxA
. 6A 00	PUSH 0	ExitCode = 0
. E8 06000000	CALL 00401020	ExitProcess
\$. FF25 08204000	JMP 00400020 PTR DS:[402008]	USER32.MessageBoxA
FF25 08204000	JMP 00400020 PTR DS:[402008]	kernel32.ExitProcess
. 00	DB 00	
. 00	DB 00	
. 00	DB 00	
. 00	DB 00	
. C6	DB C6	
. 00	DB 00	
. 00	DB 00	
. 00	DB 00	
. 00	DB 00	

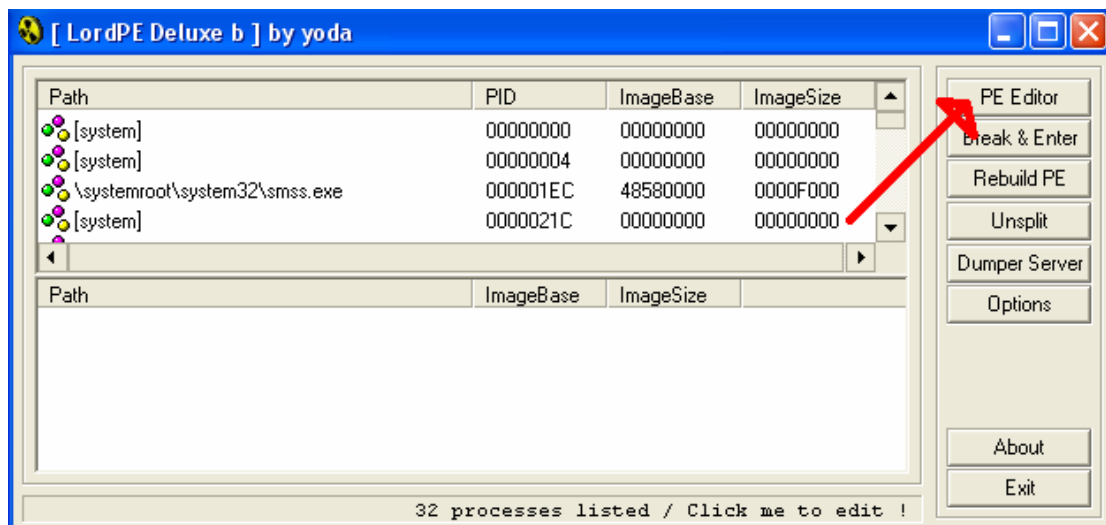
بله، برنامه ما در MASM نوشته شده و فقط با تابع MessageBoxA یک پیغام نشان می دهد و بعد با تابع ExitProcess بسته می شود. حالا کافیه با پلاگین OllyDump از فایلمان Dump بگیریم، برای این کار کلیک راست کرده و گزینه Dump Debugged Process را بزنید:



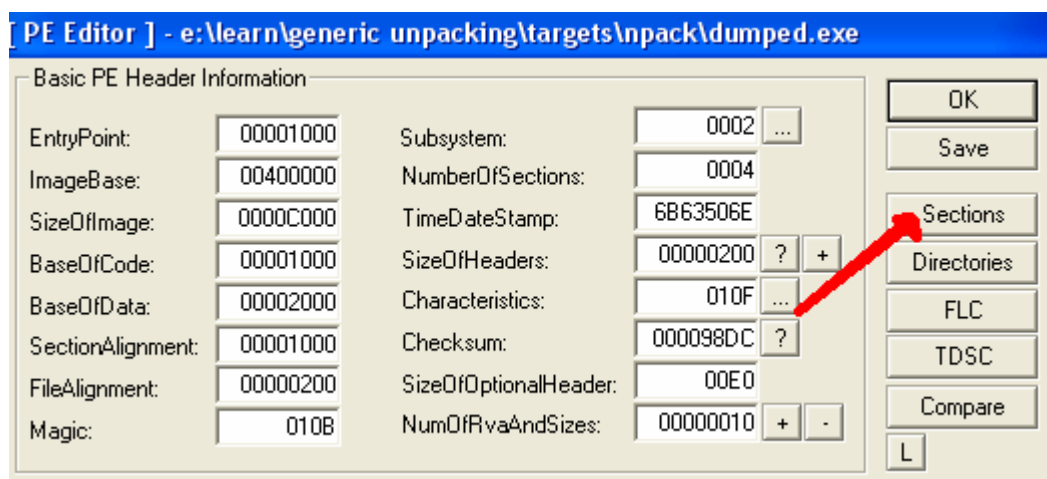
در تصویر بالا گزینه Rebuild Import را باید تیک بزنیم، چرا که خود OllyDump می تواند IAT ما را دوباره بسازد. و نیازی به استفاده از برنامه ای مثل ImportREC نیست. ولی خوب، اگر قرار بود IAT را با ImportREC بسازیم، باید تیک این گزینه را برمی داشتیم. حالا گزینه Dump را بزنید و فایل دامپ شده را ذخیره کنید. و بعد فایل دامپ شده را اجرا کنید:



می بینید که فایل دامپ شده بی هیچ مشکلی اجرا می شود. در مورد اینکه با چه دستوری وارد OEP می شویم، باید بگویم که هر دستوری ممکن است ما را به OEP ببرد. ممکن است یک Call باشد. یا اینکه مثل این مورد یک Ret ما را به OEP ببرد. خوب، حالا باید سکشن nPack را از فایل آنپک شده خودمان پاک کنیم، چرا که فایل آنپک شده و دیگر نیازی به آن نیست. برای پاک کردن یک سکشن هم می توانید از نرم افزاری همانند LordPE استفاده کنیم:



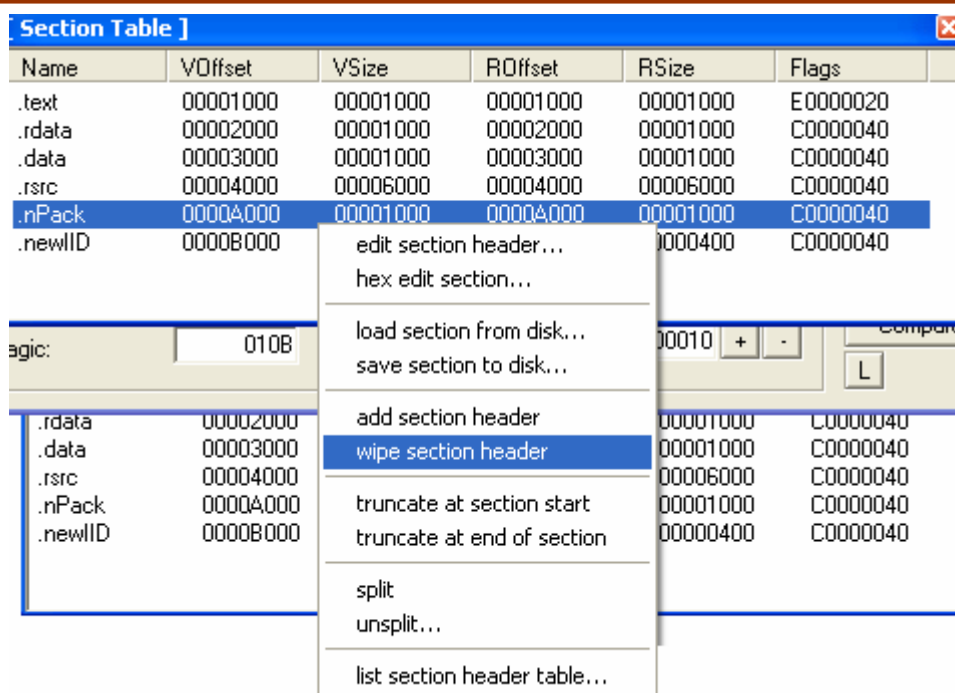
گزینه PE Editor را بزنید و فایل خود را انتخاب کنید:



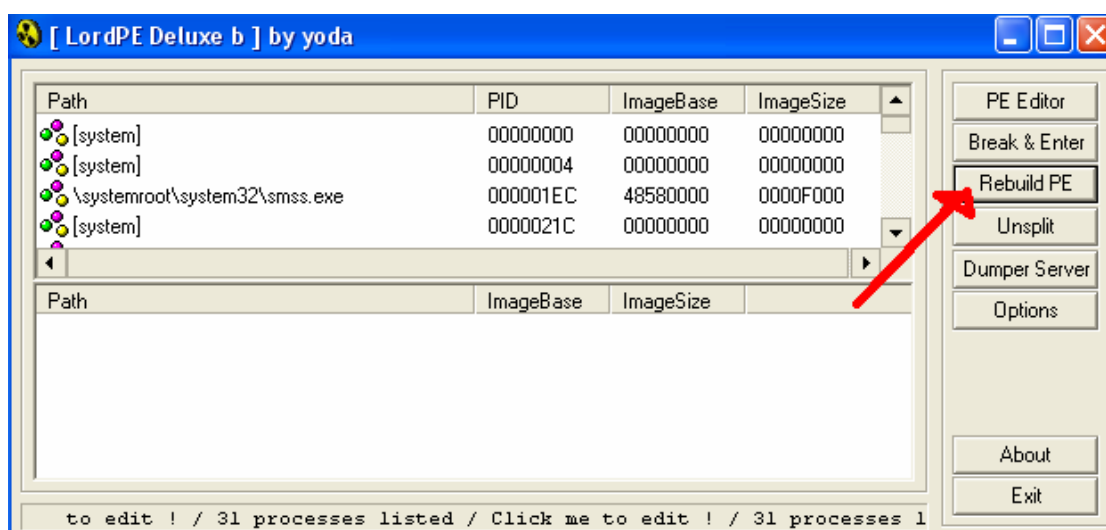
گزینه Sections را بزنید:

[Section Table]					
Name	VOffset	VSize	ROffset	RSize	Flags
.text	00001000	00001000	00001000	00001000	E0000020
.rdata	00002000	00001000	00002000	00001000	C0000040
.data	00003000	00001000	00003000	00001000	C0000040
.rsrc	00004000	00006000	00004000	00006000	C0000040
.nPack	00004000	00001000	00004000	00001000	C0000040
.newIID	00008000	00001000	00008000	00004000	C0000040

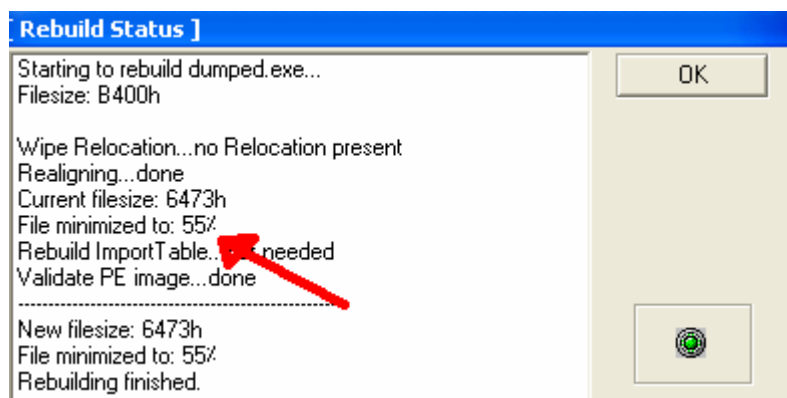
همانطور که می بینید یک سکشن جدید به نام newIID به فایل آپیک شده اضافه شده که حاوی Import های ماست. خوب، همانند تصویر بر روی سکشن nPack کلیک راست کرده و گزینه Wipe Section Header را بزنید:



و بعد این پنجره را ببندید و گزینه Save را بزنید. و بعد OK کنید. حالا گزینه Rebuild PE را بزنید.



و فایل آپک شده را به برنامه بدهید تا LordPE فایل را دوباره بسازد.



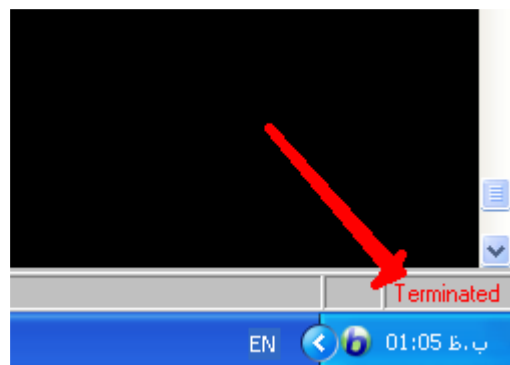
همانطور که می بینید فایل دوباره ساخته شده ما تنها ۵۵٪ حجم فایل اولیه را دارد. یعنی ۴۵٪ درصد از حجم فایل کم شد ☺

BamBam

در مثال قبل برای یافتن OEP از روش معمولی استفاده کردیم. یعنی با کلید F8 برنامه را آنقدر به جلو بردیم تا به سکشن اصلی برنامه رسیدیم. در اینجا هم می خواهیم از همین روش استفاده کنیم ولی باید بدانیم که این روش همیشه قابل استفاده نیست. ممکن است کدهای پکر ما بسیار زیاد باشد و اگر بخواهیم از این روش استفاده کنیم ممکن است وقت زیادی از ما بگیرد. لذا باید راه های راحت تری پیدا کرد ☺
این دو سکشن برای ما اهمیت بیشتری دارند:

.text → 401000 تا 44A000
.BedRock → 46B000 تا 46E000

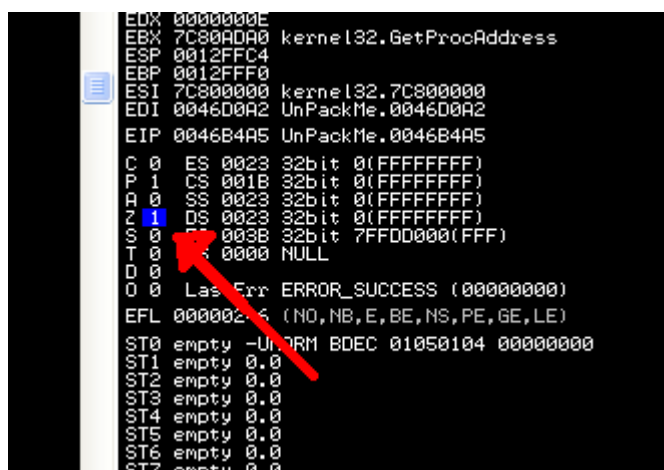
فایل را در داخل یک OllyDBG بدون پلاگین باز کند و برنامه را با کلید F9 اجرا کنید:



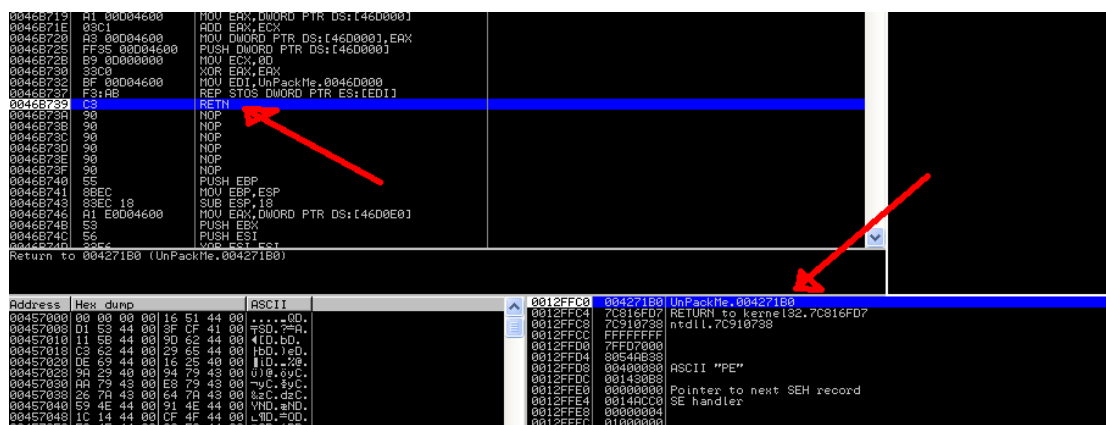
همانطور که می بینید برنامه Terminate شده... اما برنامه که خارج از OllyDBG بدون هیچ مشکلی اجرا می شد؟ پس مشکل چیه؟ بهتر است برنامه را Restart کرده و این بار با کلید F8 برنامه را ادامه دهیم تا به این خط برسیم:

0046B48E	8BD1	MOV EDX, ECX	
0046B490	C1E9 02	SHR ECX, 2	
0046B493	F3:AB	REP STOS DWORD PTR ES:[EDI]	
0046B495	8BCA	MOV ECX, EDX	
0046B497	83E1 03	AND ECX, 3	
0046B49A	F3:AA	REP STOS BYTE PTR ES:[EDI]	
0046B49C	FF15 08D04600	CALL DWORD PTR DS:[46D008]	kernel32.IsDebuggerPresent
0046B4A2	83F8 01	CMP EAX, 1	
0046B4A5	75 08	JNZ SHORT UnPackMe.0046B4AF	
0046B4A7	6A 00	PUSH 0	
0046B4A9	FF15 08D04600	CALL DWORD PTR DS:[46D008]	kernel32.ExitProcess
0046B4AF	6A 18	PUSH 18	
0046B4B1	6A 40	PUSH 40	
0046B4B3	FF15 04D04600	CALL DWORD PTR DS:[46D004]	kernel32.GlobalAlloc
0046B4B9	8BD8	MOV EBX, EAX	
0046B4BB	53	PUSH EBX	
0046B4BC	68 00D04600	PUSH UnPackMe.0046D000	
0046B4C1	E8 3AFBFFFF	CALL UnPackMe.0046B000	
0046B4C6	83C4 08	ADD ESP, 8	
0046B4C9	B9 06000000	MOV ECX, 6	

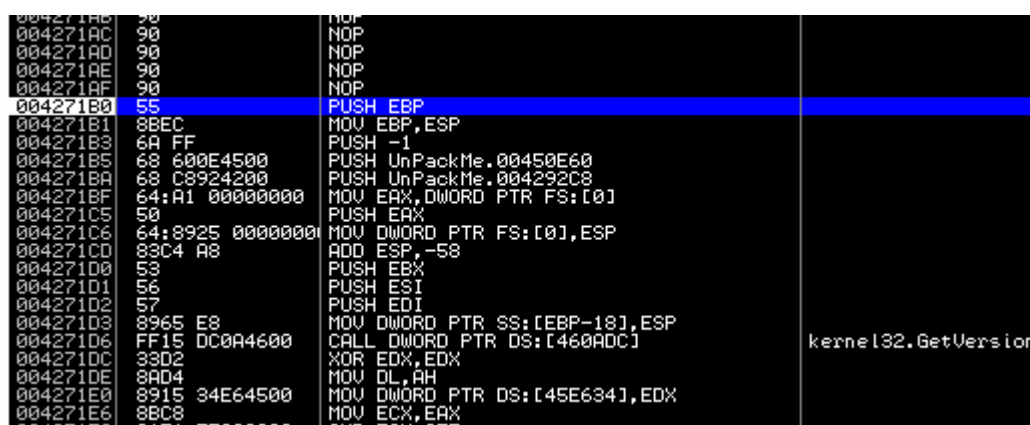
همانطور که می بینید برنامه در این خط از تابع IsDebuggerPresent استفاده می کند. و اگر مقدار بازگشتی این تابع برابر یک باشد. برنامه Terminate می شود. لذا حتما باید پرشی که در خط 0046BA45 قرار دارد. حتما باید انجام شود. در غیر این صورت به تابع ExitProcess می رسیم و از برنامه خارج می شویم ☹
پس دو بار دیگر کلید F8 را بزنید تا به این پرش که تصمیم گیرنده وجود یا عدم وجود دیباگر هست برسیم.



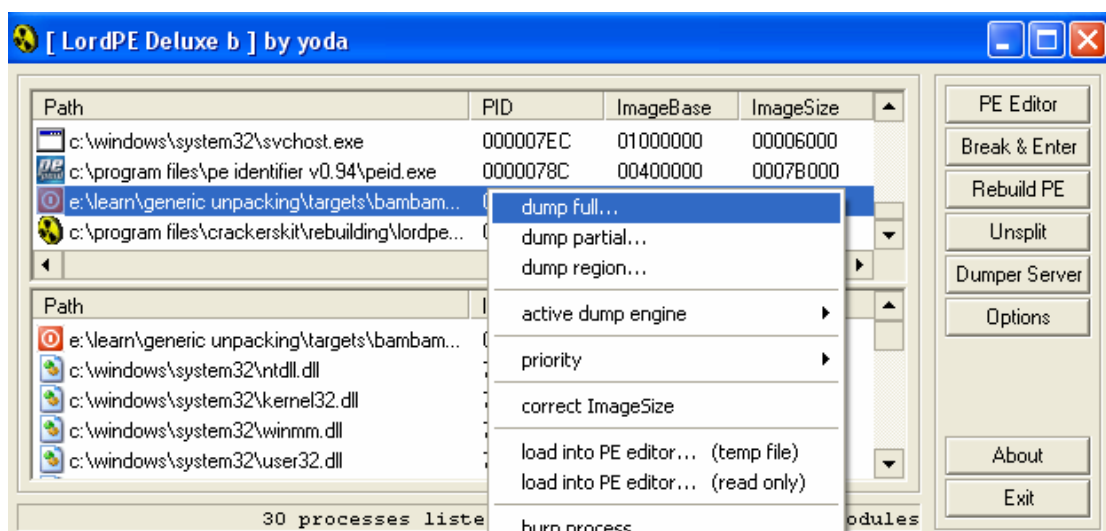
الان مقدار Z-flag در کامپیوتر من برابر ۱ است. یکبار بر روی آن کلیک می کنیم تا مقدار آن صفر شود. به این ترتیب این پرش، انجام می شود ☺
حالا اگر برنامه را با کلید F9 اجرا کنید. می بینید که برنامه بدون هیچ مشکلی اجرا می شود. حالا باید OEP را پیدا کنیم. برای یافتن OEP هم از همان روش قبلی استفاده می کنیم. یعنی آنقدر در برنامه به جلو می روم تا به یک تغییر سکشن برسیم. آنقدر در برنامه با کلید F8 به جلو بروید تا بالاخره به این خط برسید:



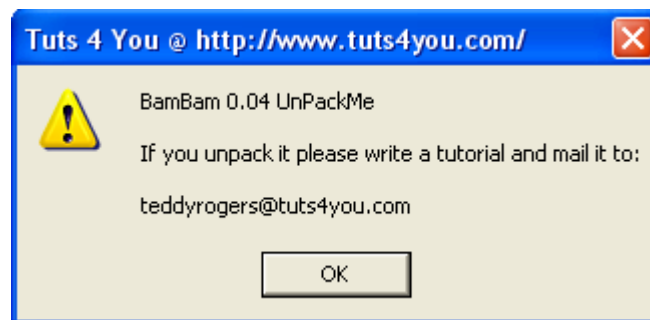
همانطور که می بینید این دستور ما را به خط 4271B0 می برد. یعنی از سکشن Bedrock به سکشن text می رویم. یعنی با این دستور به OEP می رویم ☺



حالا این بار با نرم افزار LordPE از فایل دامپ می گیریم.
LordPE را اجرا کنید، در لیست پروسه ها، پروسه مورد نظر خود را انتخاب کنید و کلیک راست کرده و گزینه Dump Full را بزنید، همانند تصویر:



حالا هم مثل قبل OEP برنامه را بر حسب RVA (271B0) به ImportREC می دهیم و فایل Dump را فیکس می کنیم. همانطور که می بینید فایل آپیک شده ما بدون هیچ مشکلی اجرا می شود ☺



CrypKey SDK

این بار هم برای یافتن OEP از همان روش استفاده می کنیم. یافتن OEP در این فایل بسیار راحت است. به طوری که با سه یا چهار بار زدن کلید F8 به OEP می رسیم ☺ از بین سکشن های برنامه، این دو سکشن برای ما اهمیت بیشتری دارد:

.text → 401000 → 44A17F

Have → 46B000 تا 46D000

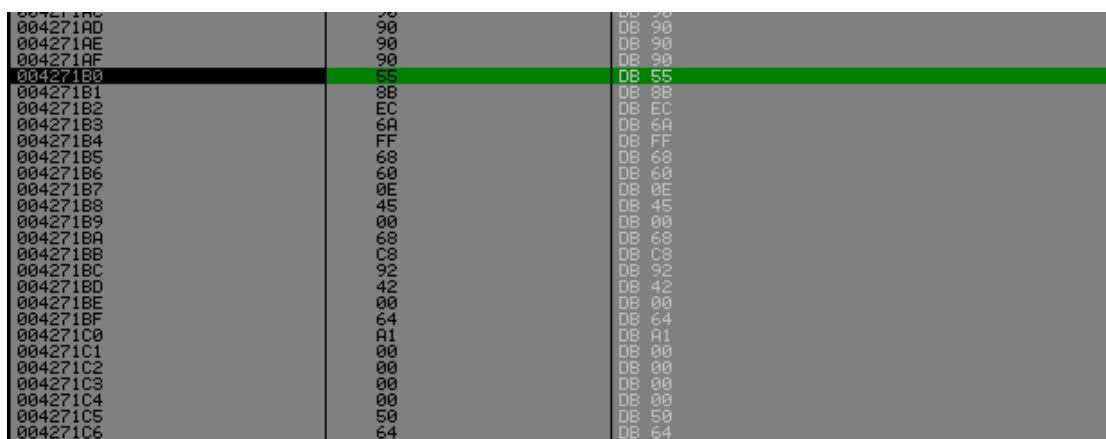
برنامه را در OllyDBG باز کنید:



برنامه را با F8 به جلو ببرید تا به خط 0046B6F9 برسید:



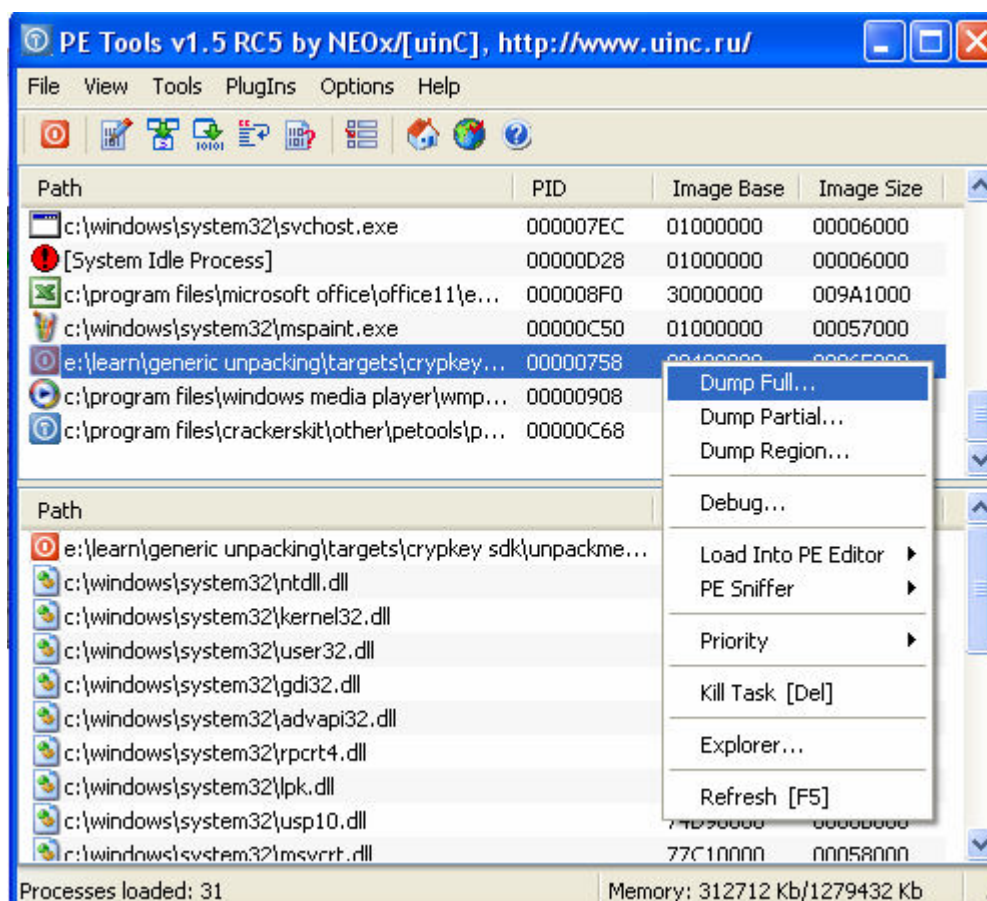
همانطور که می بینید این Jump ما را به خط 4271B0 می برد. یعنی از سکشن Have به سکشن text می رویم. پس این پرش به OEP می رود. ☺



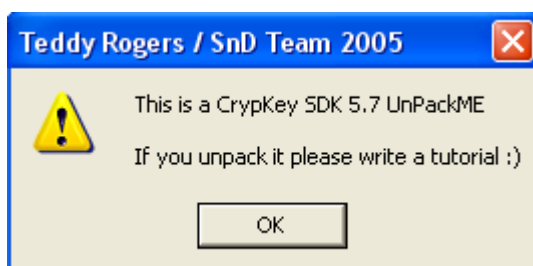
کلیدهای CTRL + A را بزنید، تا کدهای این سکشن آنالیز شود.

004271B0	55	PUSH ESP
004271B1	8BEC	MOV EBP,ESP
004271B3	6A FF	PUSH -1
004271B5	68 60E4500	PUSH 00450E60
004271B7	68 C8924200	PUSH 004392C8
004271B9	64:A1 00000000	MOV EAX,DWORD PTR FS:[0]
004271BB	50	PUSH EAX
004271BD	64:8925 00000000	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 SS:[EBP-18],ESP
004271C9	FF15 DC0A4600	CALL DWORD PTR DS:[460ADC]
004271CB	33D2	XOR EDX,EDX
004271CD	8AD4	MOV DL,AH
004271CE	8915 34E64500	MOV DWORD PTR DS:[45E634],EDX
004271D0	8BC8	MOV ECX,EAX
004271D2	81E1 FF000000	AND ECX,0FF
004271D4	890D 30E64500	MOV DWORD PTR DS:[45E630],ECX
004271D6	C1E1 08	SHL ECX,8
004271D8	83CA	ADD ECX,EDX
004271DA	890D 2CE64500	MOV DWORD PTR DS:[45E62C],ECX
004271DC	C1E8 10	SHR EAX,10
004271DE	8B 28F64500	MOV DWORD PTR DS:[45E628],EAX

حالا این بار فایل را با PeTools دامپ می کنیم. برنامه PeTools را اجرا کنید. در لیست پروسه ها، پروسه مورد نظر خود را انتخاب کنید و کلیک راست کرده و گزینه Dump Full را بزنید. همانند تصویر:



حالا ImportREC را باز کنید. OEP برنامه را بر حسب RVA به برنامه بدهید. (271B0) و بعد فایل دامپ شده را فیکس کنید. می بینید که فایل آنپک شده بدون هیچ مشکلی اجرا می شود ☺

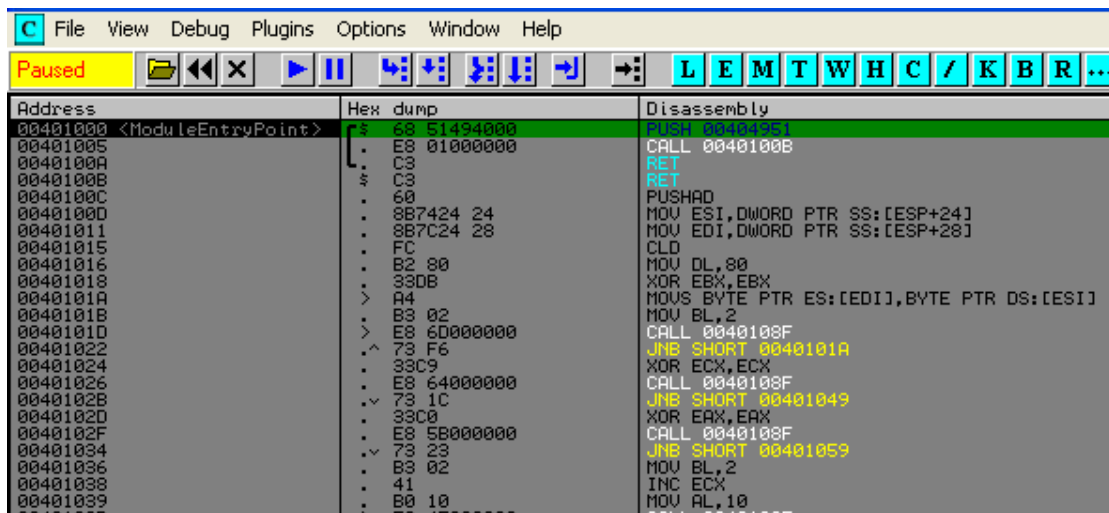


KByS Packer

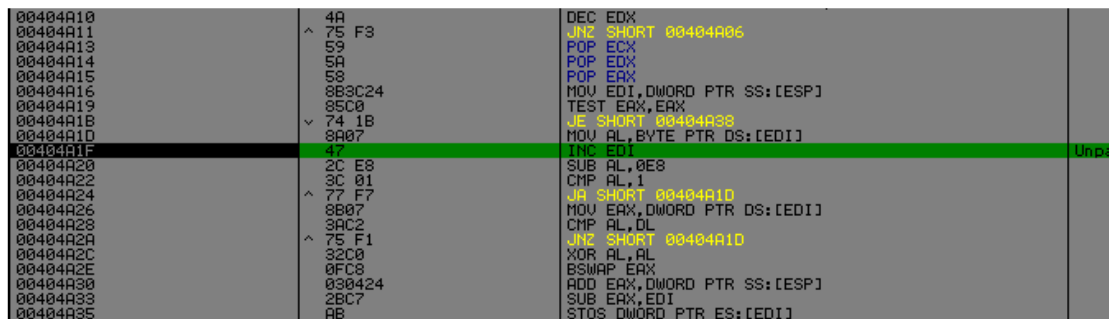
توضیحی کوتاه در مورد پکر: این پکر یکی از پکرهای مورد علاقه منه... که توسط یک نفر با نام مستعار (یا شاید هم واقعی) Shoooo نوشته شده. برای پک کردن فایل با این برنامه تنها کافیت فایل مورد نظر خود را به داخل برنامه بکشید. (Drag کنید). این دو سکشن برای ما اهمیت بیشتری دارند:

.text → 401000 تا 400E2C
.Shoooo → 404000 تا 405000

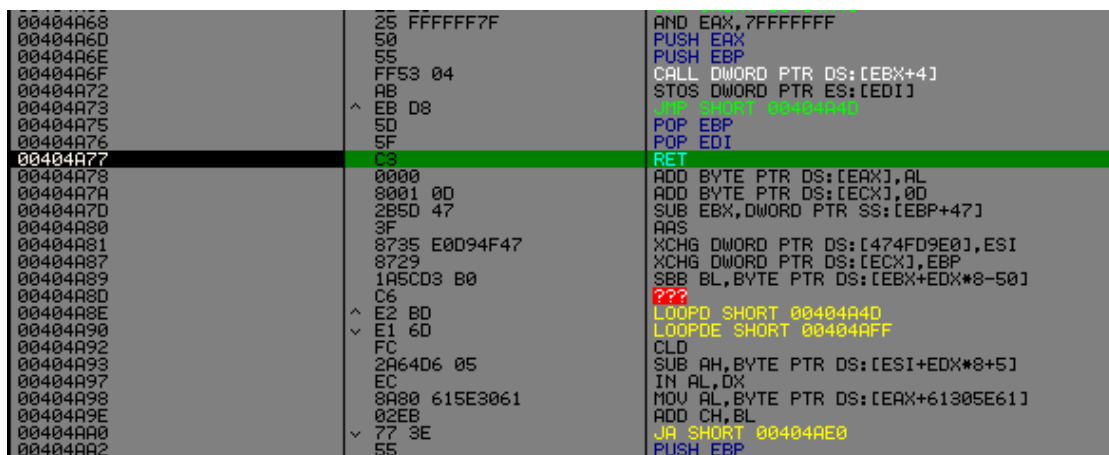
فایل مورد نظر را در OllyDBG باز کنید:



حالا هم همانند مثال های قبل با کلید F8 آنقدر برنامه را ادامه می دهیم تا به OEP برسیم. با F8 به جلو بروید تا به اینجا برسید:



در اینجا ما در این حلقه گیر می کنیم. برای اینکه راحت تر این حلقه را رد کنیم. بر روی دستور RET که چند خط پایینتر در آدرس 00404A77 قرار دارد. یک Break Point می گذاریم و برنامه را با کلید F9 اجرا می کنیم:



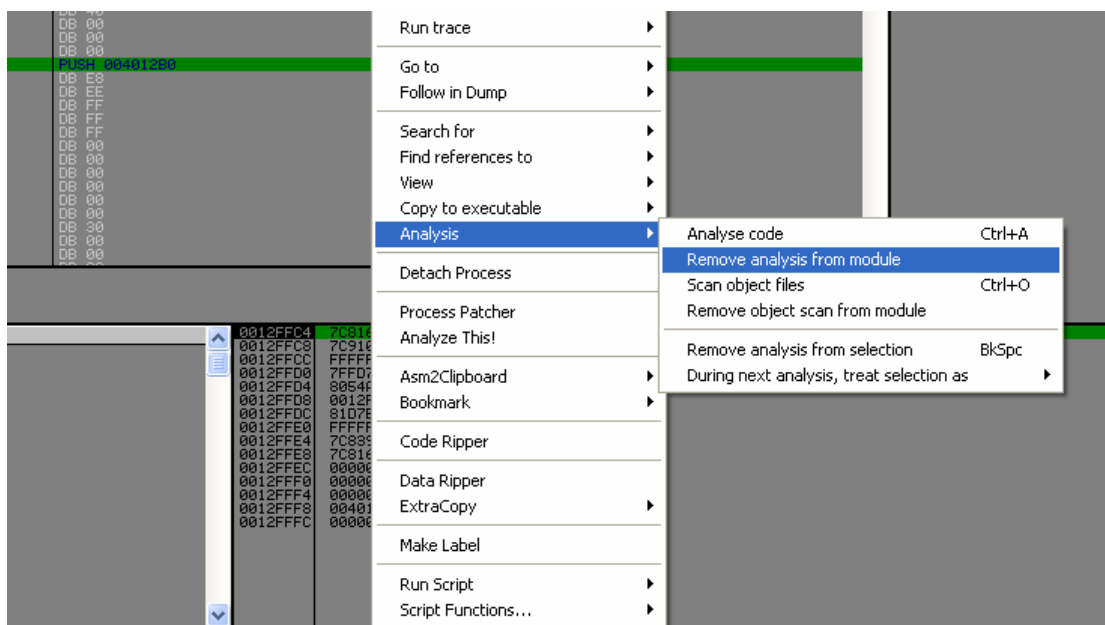
حالا هم با F8 برنامه را ادامه می دهیم تا به اینجا برسیم:

0040494C	0000	ADD BYTE PTR DS:[EAX],AL	
0040494E	0000	ADD BYTE PTR DS:[EAX],AL	
00404950	61	POPAD	
00404951	B8 74110000	MOV EAX,1174	
00404956	BA 00004000	MOV EDI,00400000	
0040495B	83C2	ADD EAX,EDX	
0040495D	- FFEB	JMP EBX	UnpackMe.00401174
0040495F	B1 15	MOV CL,15	
00404961	0000	ADD BYTE PTR DS:[EAX],AL	
00404963	68	PUSHAD	
00404964	E9 00000000	JMP 00404969	
00404969	5E	POP ESI	
0040496A	83EE 0A	SUB ESI,0A	
0040496D	8B06	MOV EAX,DWORD PTR DS:[ESI]	
0040496F	83C2	ADD EAX,EDX	
00404971	8B06	MOV ECX,DWORD PTR DS:[EAX]	
00404973	894E F3	MOV DWORD PTR DS:[ESI-DI],ECX	
00404976	83EE 0F	SUB ESI,0F	
00404979	56	PUSH ESI	
0040497A	52	PUSH EDX	
0040497B	8BFA	MOV ESI,EBX	

این دستور JMP EAX ما را از سکشن Shoooo به سکشن text یعنی خط 401174 می برد. پس ممکن است با این دستور به OEP برویم. یکبار کلید F7 را بزنید تا همه چیز روشن شود.

00401167	3C	DB 3C	CHAR '<'
00401169	10	DB 10	CHAR '>'
0040116A	40	DB 40	CHAR '@'
0040116B	00	DB 00	
0040116C	FF	DB FF	
0040116D	25	DB 25	CHAR '%'
0040116E	70	DB 70	CHAR 'p'
0040116F	10	DB 10	
00401170	40	DB 40	CHAR '@'
00401171	00	DB 00	
00401172	00	DB 00	
00401173	00	DB 00	
00401174	> 80 01240000	JMP 00401200	
00401179	E8	DB E8	
0040117A	EE	DB EE	
0040117B	FF	DB FF	
0040117C	FF	DB FF	
0040117D	FF	DB FF	
0040117E	00	DB 00	
0040117F	00	DB 00	
00401180	00	DB 00	
00401181	00	DB 00	
00401182	00	DB 00	
00401183	00	DB 00	
00401184	30	DB 30	
00401185	00	DB 00	CHAR '0'
00401186	00	DB 00	
00401187	00	DB 00	
004012B0=004012B0 Jump from 00401176			

بله... خط 401174 همان OEP ماست. اینبار بهتر است آنالیز OillyDBG را برداریم. برای اینکار کلیک راست کرده گزینه Analysis و سپس گزینه Remove analysis from module را بزنید.



004010A1	25 80104000	AND EAX,401030	MSUBUH60._vbaExceptionHandler
004010A2	FF25 40104000	JMP DWORD PTR DS:[40104000]	MSUBUH60._vbaFPException
004010A3	FF25 54104000	JMP DWORD PTR DS:[40105400]	MSUBUH60._adj_fdiv_m16i
004010A4	FF25 24104000	JMP DWORD PTR DS:[40102400]	MSUBUH60._adj_fdiv_m32i
004010A5	FF25 1C104000	JMP DWORD PTR DS:[40101C00]	MSUBUH60._adj_fdiv_m32i
004010A6	FF25 5C104000	JMP DWORD PTR DS:[40105C00]	MSUBUH60._adj_fdiv_m64
004010A7	FF25 10104000	JMP DWORD PTR DS:[40101000]	MSUBUH60._adj_fdiv_r
004010A8	FF25 6C104000	JMP DWORD PTR DS:[40106C00]	MSUBUH60._adj_fdivr_m16i
004010A9	FF25 28104000	JMP DWORD PTR DS:[40102800]	MSUBUH60._adj_fdivr_m32i
004010AA	FF25 68104000	JMP DWORD PTR DS:[40106800]	MSUBUH60._adj_fdivr_m64
004010AB	FF25 60104000	JMP DWORD PTR DS:[40106000]	MSUBUH60._adj_fpatan
004010AC	FF25 50104000	JMP DWORD PTR DS:[40105000]	MSUBUH60._adj_fprem
004010AD	FF25 38104000	JMP DWORD PTR DS:[40103800]	MSUBUH60._adj_fprem1
004010AE	FF25 4C104000	JMP DWORD PTR DS:[40104C00]	MSUBUH60._adj_fptan
004010AF	FF25 14104000	JMP DWORD PTR DS:[40101400]	MSUBUH60._Catan
004010B0	FF25 04104000	JMP DWORD PTR DS:[40100400]	MSUBUH60._Ccos
004010B1	FF25 78104000	JMP DWORD PTR DS:[40107800]	MSUBUH60._Cexp
004010B2	FF25 00104000	JMP DWORD PTR DS:[40100000]	MSUBUH60._Cilog
004010B3	FF25 88104000	JMP DWORD PTR DS:[40108800]	MSUBUH60._Cisin
004010B4	FF25 58104000	JMP DWORD PTR DS:[40105800]	MSUBUH60._Cisqrt
004010B5	FF25 2C104000	JMP DWORD PTR DS:[40102C00]	MSUBUH60._Citan
004010B6	FF25 40104000	JMP DWORD PTR DS:[40104000]	MSUBUH60._alnw1
004010B7	FF25 84104000	JMP DWORD PTR DS:[40108400]	MSUBUH60._vbaEnd
004010B8	FF25 80104000	JMP DWORD PTR DS:[40108000]	MSUBUH60._vbaFreeVarList
004010B9	FF25 0C104000	JMP DWORD PTR DS:[40100C00]	MSUBUH60._vbaFreeStrList
004010BA	FF25 08104000	JMP DWORD PTR DS:[40100800]	MSUBUH60._vbaStrCmp
004010BB	FF25 64104000	JMP DWORD PTR DS:[40106400]	MSUBUH60._vbaStrCat
004010BC	FF25 7C104000	JMP DWORD PTR DS:[40107C00]	MSUBUH60._vbaStrDup
004010BD	FF25 18104000	JMP DWORD PTR DS:[40101800]	MSUBUH60._vbaVarDup
004010BE	FF25 20104000	JMP DWORD PTR DS:[40102000]	MSUBUH60._vbaQueryInterface
004010BF	FF25 44104000	JMP DWORD PTR DS:[40104400]	MSUBUH60._vbaStrCmp
004010C0	FF25 34104000	JMP DWORD PTR DS:[40103400]	MSUBUH60._vbaStrCat
004010C1	FF25 3C104000	JMP DWORD PTR DS:[40103C00]	MSUBUH60._vbaStrDup
004010C2	FF25 70104000	JMP DWORD PTR DS:[40107000]	MSUBUH60._vbaVarDup
004010C3	ADD BYTE PTR DS:[EAX],AL		MSUBUH60._vbaQueryInterface
004010C4	CALL 00401160		MSUBUH60._vbaStrCmp
004010C5	JMP to MSUBUH60.ThunRTMain		MSUBUH60._vbaStrCat
004010C6	ADD BYTE PTR DS:[EAX],AL		MSUBUH60._vbaStrDup
004010C7	ADD BYTE PTR DS:[EAX],AL		MSUBUH60._vbaVarDup
004010C8	ADD BYTE PTR DS:[EAX],AL		MSUBUH60._vbaQueryInterface
004010C9	XOR BYTE PTR DS:[EAX],AL		MSUBUH60._vbaStrCmp
004010CA	ADD BYTE PTR DS:[EAX],AL		MSUBUH60._vbaStrCat
004010CB	INC EAX		MSUBUH60._vbaStrDup
004010CC	ADD BYTE PTR DS:[EAX],AL		MSUBUH60._vbaVarDup

همانطور که می بینید در این برنامه Jump Table درست بالای OEP قرار دارد. این یعنی برنامه ما در ویرال بیسیک نوشته شده.

راه حل دیگر:

راه حل دیگر برای یافتن OEP این است که ما از خصوصیات زبان برنامه نویسی Visual Basic استفاده کنیم. یعنی اول برنامه را کامل با کلید F9 اجرا کنید و بعد کلیدهای CTRL + G را بزنیم و به ابتدای سکشن text یعنی 401000 برویم. و در آنجا به دنبال Jump Table باشیم.

Address	Hex dump	Disassembly	Comment
00401000	06	PUSH ES	
00401001	800F	MOV BYTE PTR DS:[EDI],CL	
00401003	66:79 FE	JNS SHORT 00001004	
00401006	0E	PUSH CS	
00401007	66:A2 721066A8	MOV BYTE PTR DS:[A8661072],AL	
0040100D	BD 0C663AF7	MOV EBP,F73A660C	
00401012	0E	PUSH CS	
00401013	66:C1FD 0E	SAR BP,0E	
00401017	66:3A5F 0E	CMP BL,BYTE PTR DS:[EDI+E]	
0040101B	66:EE	OUT DX,AL	
0040101D	F6	???	Unknown
0040101E	0E	PUSH CS	
0040101F	66:F3:	PREFIX REP:	
00401021	C50D 6686F70E	LDS ECX,F70E6666	
00401027	66:86F8	XCHG AL,BH	
0040102A	0E	PUSH CS	
0040102B	66:6E	OUTS DX,BYTE PTR ES:[EDI]	
0040102D	890F	MOV DWORD PTR DS:[EDI],ECX	
0040102F	66:6A 57	PUSH 57	
00401032	0F66949A 0C6676FE	PCMPGTD MM2,QWORD PTR DS:[EDX+EBX*4+FE76660C]	
0040103A	0E	PUSH CS	
0040103B	66:A7	CMPS WORD PTR DS:[ESI],WORD PTR DS:[EDI]	
0040103D	9A 0C66138A 0F66	CALL FAR 660F:8A13660C	
00401044	A5	MOVS DWORD PTR ES:[EDI],DWORD PTR DS:[ESI]	
00401045	29	CDQ	

در تصویر بالا من در خط 401000 هستم. حالا کافیه کلیدهای CTRL + B را بزنم و به دنبال FF25 Opcode بگردم... چرا که Jump های Jump Table با FF25 Opcode شروع می شود:

Enter binary string to search for

ASCII %

UNICODE

HEX +02 FF 25

☒ Entire block

☐ Case sensitive

<< >>

OK Cancel

کلید OK را بزنید.

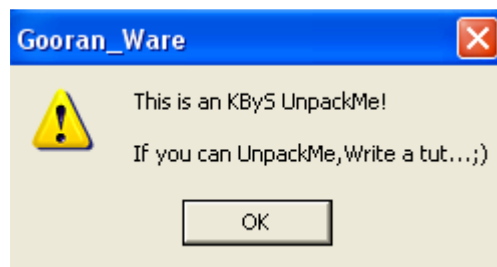
0040109C	0C 1B	OR AL,1B	
0040109E	40	INC EAX	
0040109F	00FF	ADD BH,BH	
004010A1	25 30104000	RND EAX,401030	
004010A6	FF25 40104000	JMP DWORD PTR DS:[40103040]	MSUBUH60.__vbaExceptHandler
004010AC	FF25 54104000	JMP DWORD PTR DS:[40103040]	MSUBUH60.__vbaFPException
004010B2	FF25 24104000	JMP DWORD PTR DS:[40103040]	MSUBUH60._adj_fdiv_m16i
004010B8	FF25 1C104000	JMP DWORD PTR DS:[40103040]	MSUBUH60._adj_fdiv_m32i
004010BE	FF25 5C104000	JMP DWORD PTR DS:[40103040]	MSUBUH60._adj_fdiv_m32l
004010C4	FF25 10104000	JMP DWORD PTR DS:[40103040]	MSUBUH60._adj_fdiv_m64
004010CA	FF25 6C104000	JMP DWORD PTR DS:[40103040]	MSUBUH60._adj_fdiv_r
004010D0	FF25 28104000	JMP DWORD PTR DS:[40103040]	MSUBUH60._adj_fdivr_m16i
004010D6	FF25 68104000	JMP DWORD PTR DS:[40103040]	MSUBUH60._adj_fdivr_m32i
004010DC	FF25 60104000	JMP DWORD PTR DS:[40103040]	MSUBUH60._adj_fdivr_m32l
004010E2	FF25 50104000	JMP DWORD PTR DS:[40103040]	MSUBUH60._adj_fdivr_m64
004010E8	FF25 38104000	JMP DWORD PTR DS:[40103040]	MSUBUH60._adj_fprem
004010EE	FF25 4C104000	JMP DWORD PTR DS:[40103040]	MSUBUH60._adj_fprem1
004010F4	FF25 14104000	JMP DWORD PTR DS:[40103040]	MSUBUH60._adj_fptan
004010FA	FF25 04104000	JMP DWORD PTR DS:[40103040]	MSUBUH60._adj_fptan
00401100	FF25 78104000	JMP DWORD PTR DS:[40103040]	MSUBUH60._Catan
00401106	FF25 08104000	JMP DWORD PTR DS:[40103040]	MSUBUH60._Ccos
0040110C	FF25 88104000	JMP DWORD PTR DS:[40103040]	MSUBUH60._Cexp
00401112	FF25 58104000	JMP DWORD PTR DS:[40103040]	MSUBUH60._Clog
00401118	FF25 2C104000	JMP DWORD PTR DS:[40103040]	MSUBUH60._Csin
0040111E	FF25 48104000	JMP DWORD PTR DS:[40103040]	MSUBUH60._Csqrt
00401124	FF25 84104000	JMP DWORD PTR DS:[40103040]	MSUBUH60._Ctan
0040112A	FF25 80104000	JMP DWORD PTR DS:[40103040]	MSUBUH60._alnw1
00401130	FF25 9C104000	JMP DWORD PTR DS:[40103040]	MSUBUH60.__vbaEnd
00401136	FF25 08104000	JMP DWORD PTR DS:[40103040]	MSUBUH60.__vbaFreeVarList
0040113C	FF25 64104000	JMP DWORD PTR DS:[40103040]	MSUBUH60.__vbaFreeStrList
00401142	FF25 7C104000	JMP DWORD PTR DS:[40103040]	MSUBUH60.__vbaStrHove
00401148	FF25 18104000	JMP DWORD PTR DS:[40103040]	MSUBUH60.__vbaStrCat
0040114E	FF25 74104000	JMP DWORD PTR DS:[40103040]	MSUBUH60.__vbaVarDup
00401154	FF25 20104000	JMP DWORD PTR DS:[40103040]	MSUBUH60.rtcMsgBox
0040115A	FF25 44104000	JMP DWORD PTR DS:[40103040]	MSUBUH60.EVENT_SINK_QueryInterface
00401160	FF25 34104000	JMP DWORD PTR DS:[40103040]	MSUBUH60.EVENT_SINK_AddRef
00401166	FF25 3C104000	JMP DWORD PTR DS:[40103040]	MSUBUH60.EVENT_SINK_Release
0040116C	FF25 70104000	JMP DWORD PTR DS:[40103040]	MSUBUH60.ThunRTMain
00401172	0000	ADD BYTE PTR DS:[EAX],AL	
00401174	68 80124000	PUSH 004012B0	
00401179	E8 EFFFFFFF	CALL 0040116C	
0040117E	0000	ADD BYTE PTR DS:[EAX],AL	JMP to MSUBUH60.ThunRTMain
00401180	0000	ADD BYTE PTR DS:[EAX],AL	
00401182	0000	ADD BYTE PTR DS:[EAX],AL	

همانطور که می بینید به Jump Table درست زیر Jump Table یک دستور Push وجود دارد که همان OEP ماست.

حالا بعد از پیدا کردن OEP بهتر است از فایل‌مان Dump بگیریم. برای این کار اینبار از نرم افزار WARK استفاده می کنیم. برای این کار ابتدا برنامه WARK را اجرا کنید در لیست پروسه ها فایل مورد نظر خودمان را انتخاب کرده و کلیک راست می کنیم و گزینه Dump Full را می زنیم، همانند تصویر:

PEID	00000D44	00400000	0007B000	00401000	C:\Program Files\PE Identifier v0.94\PEID.exe
PEID	000001E8	00400000	0007B000	00401000	C:\Program Files\PE Identifier v0.94\PEID.exe
UnpackMe	00000258	00400000	00005000	00401000	Refresh
wark	00000E40	00400000	00062000	004204A4	Dump Full
					Dump Partial
					Dump Section
					Set Priority
					Kill Process
					Open Folder
					File Properties
					Open Process File With WPE (ReadOnly)
					Open Process File With WPE (Temporary)
Module	Address	Size	Entry Point	Location	
ntdll	7C900000	000B0000	7C913156	C:\WINDOWS\system32\ntdll.dll	
kernel32	7C800000	000F5000	7C80B5AE	C:\WINDOWS\system32\kernel32.dll	
MSVBVM60	66000000	00152000	66001AEC	C:\WINDOWS\system32\MSVBVM60.dll	
USER32	7E410000	00090000	7E42E966	C:\WINDOWS\system32\USER32.dll	
GDI32	77F10000	00047000	77F16597	C:\WINDOWS\system32\GDI32.dll	
ADVAPI32	77DD0000	0009B000	77DD70D4	C:\WINDOWS\system32\ADVAPI32.dll	
RPCRT4	77E70000	00091000	77E76284	C:\WINDOWS\system32\RPCRT4.dll	
ole32	774E0000	0013D000	774FD0A1	C:\WINDOWS\system32\ole32.dll	
msvcrt	77C10000	00058000	77C1F2A1	C:\WINDOWS\system32\msvcrt.dll	
OLEAUT32	77120000	0008B000	77121558	C:\WINDOWS\system32\OLEAUT32.dll	
LPK	629C0000	00009000	629C2EAD	C:\WINDOWS\system32\LPK.DLL	

حالا هم نرم افزار ImportREC را باز می کنیم. OEP را بر حسب RVA به برنامه می دهیم (1174) و بعد فایل دامپ شده خودمان را فیکس می کنیم.



فایل آپک شده ما بدون هیچ مشکلی اجرا می شود ☺

PKLite

این پکر هم بسیار راحت OEP آن پیدا می شود.

برنامه را در OllyDBG باز کنید و یک نگاهی به Memory Map (کلید های ALT + M) می اندازیم:

003A0000	00001000	003A0000	(itself)		
003B0000	00004000	003B0000	(itself)		
003C0000	00008000	003C0000	(itself)		
003D0000	00010000	003D0000	(itself)		
003E0000	00010000	003E0000	(itself)		
003F0000	00003000	003F0000	(itself)		
00400000	00001000	UnPackMe	00400000		PE header
00401000	0002E000	UnPackMe	00400000	CODE	
0042F000	00001000	UnPackMe	00400000	DATA	
00430000	00001000	UnPackMe	00400000	BSS	
00431000	00002000	UnPackMe	00400000	.idata	imports
00433000	00001000	UnPackMe	00400000	.tls	
00434000	00001000	UnPackMe	00400000	.rdata	
00435000	00004000	UnPackMe	00400000	.delete	
00439000	00004000	UnPackMe	00400000	.rsrc	resources
0043D000	00022000	UnPackMe	00400000	.pklstb	code
0045F000	00001000	UnPackMe	00400000	.relo2	relocations
00460000	00006000		00460000	(itself)	
00520000	00002000		00460000		
00530000	00011000		00530000	(itself)	
00640000	000A3000		00640000	(itself)	
00940000	00010000		00940000	(itself)	
00D40000	00004000		00D40000	(itself)	
00D50000	00001000		00D50000	(itself)	
00DF0000	00002000		00DF0000	(itself)	
50090000	00001000	comctl32	50090000	(itself)	PE header

همانطور که در Memory Map می بینید دو سکشن CODE و pklstb در این آدرس قرار دارند:

.CODE → 401000 تا 42F000
.pklstb → 43D000 تا 45F000

همانطور که می بینید می توانیم به راحتی از Memory map برای پیدا کردن محل سکشن های برنامه استفاده کنیم و دیگر نیازی به استفاده از PEID برای این کار نیست. حالا به Entry Point برنامه نگاهی می اندازیم

Address	Hex dump	Disassembly
0043D000 <ModuleEntryPoint>	68 80D04300	PUSH 0043D000
0043D005	68 53A54500	PUSH 0045A553
0043D00A	68 00000000	PUSH 0
0043D00F	E8 3FD50100	CALL 0045A553
0043D014	- E9 AB0FFFFF	JMP 0042DFC4
0043D019	40	INC EAX
0043D01A	2823	SUB BYTE PTR DS:[EBX],AH
0043D01C	2900	SUB DWORD PTR DS:[EAX],EAX
0043D01E	0000	ADD BYTE PTR DS:[EAX],AL
0043D020	0000	ADD BYTE PTR DS:[EAX],AL
0043D022	0000	ADD BYTE PTR DS:[EAX],AL
0043D024	0000	ADD BYTE PTR DS:[EAX],AL
0043D026	0000	ADD BYTE PTR DS:[EAX],AL
0043D028	0000	ADD BYTE PTR DS:[EAX],AL
0043D02A	0000	ADD BYTE PTR DS:[EAX],AL
0043D02C	0000	ADD BYTE PTR DS:[EAX],AL

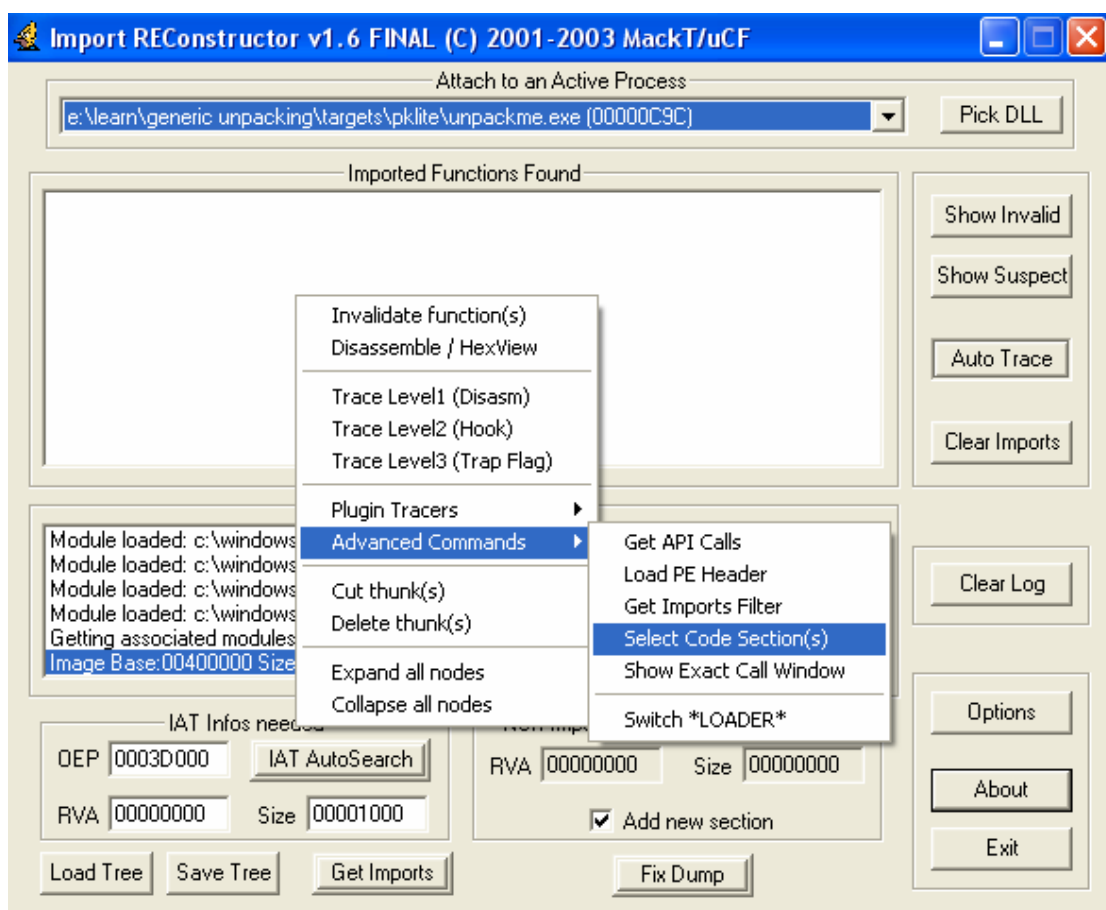
چند بار کلید F8 را بزنید تا به خط 0043D014 برسید:

Hex dump	Disassembly
68 80D04300	PUSH 0043D000
68 53A54500	PUSH 0045A553
68 00000000	PUSH 0
E8 3FD50100	CALL 0045A553
- E9 AB0FFFFF	JMP 0042DFC4
40	INC EAX
2823	SUB BYTE PTR DS:[EBX],AH
2900	SUB DWORD PTR DS:[EAX],EAX
0000	ADD BYTE PTR DS:[EAX],AL
0000	ADD BYTE PTR DS:[EAX],AL
0000	ADD BYTE PTR DS:[EAX],AL
0000	ADD BYTE PTR DS:[EAX],AL
0000	ADD BYTE PTR DS:[EAX],AL
0000	ADD BYTE PTR DS:[EAX],AL
0000	ADD BYTE PTR DS:[EAX],AL
0000	ADD BYTE PTR DS:[EAX],AL

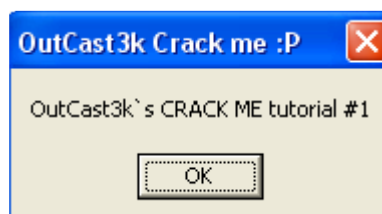
با این پرش از سکشن pklstb به سکشن CODE می رویم. یعنی این پرش احتمالا به OEP می رود. ☺

0000	0000	ADD BYTE PTR DS:[EAX],AL
00CC	0000	ADD AH,CL
DE42 00	0000	FIADD WORD PTR DS:[EDX]
55	0000	PUSH ESP
8BEC	0000	MOV EBP,ESP
83C4 F4	0000	ADD ESP,-0C
B8 F4DE4200	0000	MOV EAX,0042DEF4
E8 846FFDFF	0000	CALL 00404F58
A1 DCF94200	0000	MOV EAX,DWORD PTR DS:[42F90C]
8B00	0000	MOV EAX,DWORD PTR DS:[EAX]
E8 A0A7FFFF	0000	CALL 00428780
A1 DCF94200	0000	MOV EAX,DWORD PTR DS:[42F90C]
8B00	0000	MOV EAX,DWORD PTR DS:[EAX]
BA 24E04200	0000	MOV EDX,0042E024
E8 B7A4FFFF	0000	CALL 004284A8
8B0D 50FA4200	0000	MOV ECX,DWORD PTR DS:[42FA50]
A1 DCF94200	0000	MOV EAX,DWORD PTR DS:[42F90C]
8B00	0000	MOV EAX,DWORD PTR DS:[EAX]
8B15 FCD84200	0000	MOV EDX,DWORD PTR DS:[42D8FC]
E8 8FA7FFFF	0000	CALL 00428798
A1 DCF94200	0000	MOV EAX,DWORD PTR DS:[42F90C]
8B00	0000	MOV EAX,DWORD PTR DS:[EAX]
E8 0FA8FFFF	0000	CALL 00428824
E8 FA53FDFF	0000	CALL 00403414
0000	0000	ADD BYTE PTR DS:[EAX],AL
FFFF	0000	???
FFFF	0000	???
15 0000004F	0000	ADC EAX,4F000000
75 74	0000	JNZ SHORT 0042E098
43	0000	INC EBX
61	0000	REP

همانطور که می بینید به OEP برنامه رسیدیم. اگر با دقت بیشتری نگاه کنیم، می بینیم که برنامه در دلفی نوشته شده. (تابع GetModuleHandleA در درون اولین Call بعد از OEP قرار دارد... از جهتی Call های فراوان خود نشان دلفی است.) حالا اینبار از خود ImportREC برای گرفتن Dump از فایل استفاده می کنیم. برای این کار برنامه ImportREC را باز کنید. و پروسه مورد نظر را انتخاب کنید. حال همانند تصویر کلیک راست کرده و گزینه Advanced commands و سپس گزینه Select code section را بزنید:



حالا گزینه Full Dump را بزنید و از فایل خود دامپ بگیرید. و بعد OEP را بر حسب RVA (2DFC4) به برنامه بدهید و فایل دامپ شده خود را فیکس کنید. می بینید که فایل دامپ شده شما بدون هیچ مشکلی اجرا می شود ©



Exe32Pack

در مثال های قبل برای یافتن OEP آنقدر در برنامه به جلو رفتیم تا به OEP رسیدیم. اما در این مثال می خواهیم از روش راحت تری استفاده کنیم. روشی که در بیشتر **پکرها** جواب می دهد. این دو سکشن از فایل برای ما اهمیت بیشتری دارند:

.text → 40A000 تا 40C000

.f → 417000 تا 418851

خوب، برنامه را در OllyDBG باز کنید:

Address	Disassembly	Comment
00417000	00	DB 00
00417001	00	DB 00
00417002	00	DB 00
00417003	00	DB 00
00417004	08	DB 08
00417005	00	DB 00
00417006	00	DB 00
00417007	00	DB 00
00417008	00	DB 00
00417009	00	DB 00
0041700A	00	DB 00
0041700B	00	DB 00
0041700C <ModuleEntryPoint>	3BC0	CMP EAX,EAX
0041700E	74	DB 74
0041700F	02	DB 02
00417010	81	DB 81
00417011	83	DB 83
00417012	55	PUSH EBP
00417013	3BC0	CMP EAX,EAX
00417015	74 02	JE SHORT 00417019
00417017	8183 538BC974 01BC563E	ADD DWORD PTR DS:[EBX+74C93B53],3B56BC01
00417021	027402 81	SAL BYTE PTR DS:[EDI-18],EDX
00417025	8557 E8	TEST DWORD PTR DS:[EDI-18],EDX
00417028	0000	ADD BYTE PTR DS:[EAX],AL
00417029	0000	ADD BYTE PTR DS:[EAX],AL

همانطور که می بینید کدهای آنالیز شده چندان جالب نیست ☹️ اصولا بهتر است کدهای پکر را آنالیز نکنیم و وقتی به فایل اصلی خودمان رسیدیم، آن موقع آنالیز کنیم. برای از بین بردن آنالیز این کدها همانند تصویر عمل کنید:

The screenshot shows the OllyDBG interface. On the left, the disassembly window displays assembly code starting with 'CMP EAX,EAX' and 'JE SHORT 00417019'. In the center, the 'Analysis' menu is open, showing options like 'Analyse code', 'Remove analysis from module', 'Scan object files', and 'Remove object scan from module'. The 'Remove analysis from module' option is highlighted. On the right, the 'Registers' window shows the state of various registers, including EAX, ECX, EDI, and ESP.

حالا کدهای ما بهتر می شود:

```

0000 ADD BYTE PTR DS:[EAX],AL
0000 ADD BYTE PTR DS:[EAX],AL
0000 ADD BYTE PTR DS:[EAX],AL
0000 CMP EAX,EAX
74 02 JE SHORT 00417012
8183 553BC074 02818353 ADD DWORD PTR DS:[EBX+74C03B55],53838102
3BC9 CMP ECX,ECX
74 01 JE SHORT 0041701F
BC 563BD274 MOV ESP,74D23B56
0281 8557E800 ADD AL,BYTE PTR DS:[ECX+E85785]
0000 ADD BYTE PTR DS:[EAX],AL
003B ADD BYTE PTR DS:[EBX],BH
0B ???
74 01 JE SHORT 00417031
BE 5D8BD581 MOV ESI,81D58B5D
ED IN EAX,DX
EC IN AL,DX
3A40 00 CMP AL,BYTE PTR DS:[EAX]
3BE4 CMP ESP,ESP
74 02 JE SHORT 00417040
8187 2B95FD3B 400081EA ADD DWORD PTR DS:[EDI+3BFD952B],EA810040
2C 00 SUB AL,0
0000 ADD BYTE PTR DS:[EAX],AL
80BD 383C4000 00 CMP BYTE PTR SS:[EBP+403C38],0
74 18 JE SHORT 00417060

```

خوب، دو بار کلید F8 را بزنید تا به اینجا برسید:

Address	Hex dump	Disassembly
0012	55	PUSH ESP
0013	3BC0	CMP EAX,EAX
0015	74 02	JE SHORT 00417019
0017	8183 533BC974 01BC563B	ADD DWORD PTR DS:[EBX+74C93B53],3B56BC01
0021	027402 81	SAL BYTE PTR DS:[EDX+EAX-7F1],CL
0025	8557 E8	TEST DWORD PTR DS:[EDI-18],EDX
0028	0000	ADD BYTE PTR DS:[EAX],AL
002A	0000	ADD BYTE PTR DS:[EAX],AL
002C	3BD8	CMP EBX,EBX
002E	74 01	JE SHORT 00417031
0030	BE 5D8BD581	MOV ESI,81D58B5D
0035	ED	IN EAX,DX
0036	EC	IN AL,DX
0037	3A40 00	CMP AL,BYTE PTR DS:[EAX]
003A	3BE4	CMP ESP,ESP
003C	74 02	JE SHORT 00417040
003E	8187 2B95FD3B 400081EA	ADD DWORD PTR DS:[EDI+3BFD952B],EA810040
0048	2C 00	SUB AL,0
004A	0000	ADD BYTE PTR DS:[EAX],AL
004C	80BD 383C4000 00	CMP BYTE PTR SS:[EBP+403C38],0
0053	74 18	JE SHORT 00417060
0055	8B85 1D3C4000	MOV EAX,DWORD PTR SS:[EBP+403C1D]
005B	0385 273C4000	ADD EAX,DWORD PTR SS:[EBP+403C27]
0061	3BC9	CMP ECX,ECX
0063	74 01	JE SHORT 00417066
0065	BA 051B0500	MOV EDX,51B05
006A	00FF	ADD BH,BH

خوب قبل از هر کاری اول بگویم که رجیستر ESP چیست؟ رجیستر ESP نگهدارنده خطی است که در Stack قرار است مورد استفاده قرار گیرد. (همانند رجیستر EIP که نشاندهنده خطی است که قرار است در پنجره Dissembler اجرا شود.) وقتی مقداری وارد Stack می شود یا از آن خارج می شود، مقدار رجیستر ESP تغییر می کند.

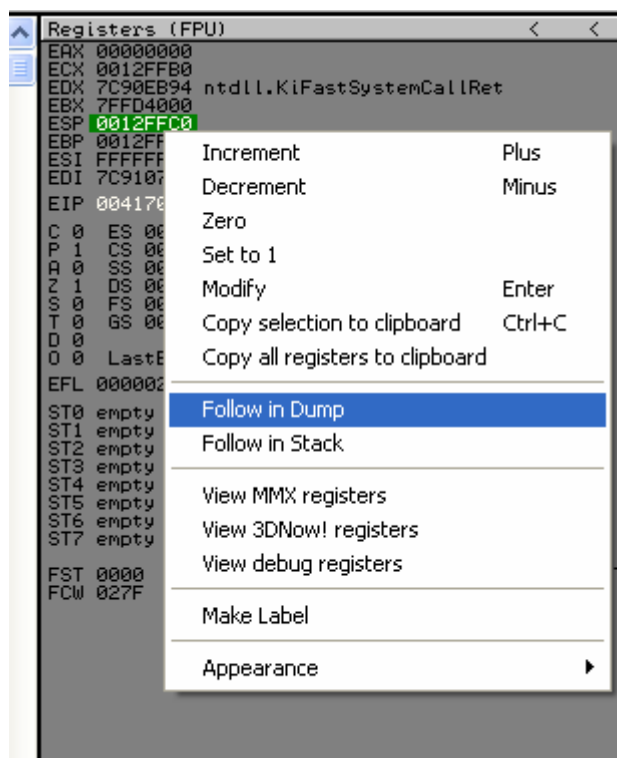
وقتی یک مقدار در ابتدای برنامه وارد Stack می شود (Push می شود)، در انتهای برنامه از Stack خارج می شود. (Pop می شود). خوب... الان در تصویر بالا مقدار EBP وارد حافظه Stack می شود (Push می شود)... من می خواهم ببینم که در کجا این مقدار از Stack خارج می شود؟... چرا؟ اصلاً چه نیازی هست؟... خوب، چون این اولین مقداری بوده که وارد Stack شده... احتمالاً آخرین مقداری خواهد بود که از حافظه خارج می شود... یعنی در آخر کدهای پکر ما این مقدار از Stack خارج می شود... به این ترتیب من می توانم بفهمم که انتهای کدهای پکر ما کجاست و به نزدیک های فایل اصلی (OEP) برسیم ☺ حالا یکبار کلید F8 را بزنید تا این خط را اجرا کنید:

```

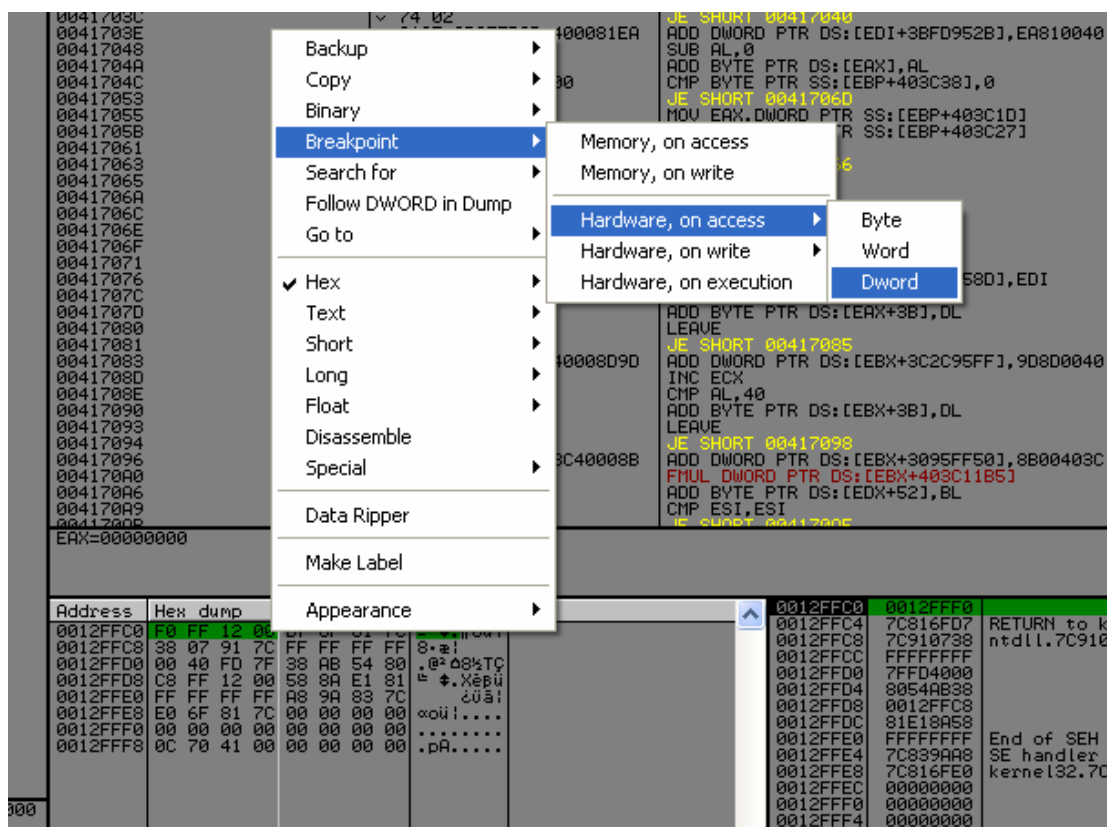
Registers (FPU)
EAX 00000000
ECX 0012FFB0
EDX 7C90EB94 ntdll.K
EBX 7FFD4000
ESP 0012FFC0
EBP 0012FFF0
ESI FFFFFFFF
EDI 7C910738 ntdll.7C
EIP 00417013 UnPackMe
C 0 ES 0023 32bit 00
P 1 CS 001B 32bit 00
A 0 SS 0023 32bit 00
Z 1 DS 0023 32bit 00
S 0 FS 003B 32bit 7F
T 0 GS 0000 NULL
D 0
O 0 LastErr ERROR_NO

```

همانطور که می بینید مقدار رجیستر ESP تغییر کرده است. اگر من بر روی این رجیستر یک Hardware Breakpoint بگذارم، می توانم بفهمم که در کجا این مقدار از Stack خارج می شود. برای Hardware Breakpoint گذاشتن روی این آدرس، همانند تصویر عمل کنید:



حالا در پنجره Dump بر روی آدرس مورد نظر(در اینجا 0012FFC0) Breakpoint hardware on access بگذارید،همانند تصویر:



خوب...حالا برنامه را با کلید F9 اجرا کنید:

Address	Hex dump	Disassembly
00418825	3BE4	CMP ESP,ESP
00418827	74 01	JE SHORT 0041882A
00418829	BF FFE0B801	MOV EDI,1B8E0FF
0041882E	0000	ADD BYTE PTR DS:[EAX],AL
00418830	003B	ADD BYTE PTR DS:[EBX],BH
00418832	C9	LEAVE
00418833	74 02	JE SHORT 00418837
00418835	81845F 3BD27401 BD5E38DE	ADD DWORD PTR DS:[EDI+EBX*2+174D23B1],DB
00418840	74 02	JE SHORT 00418844
00418842	8186 5B3BD874 0281865D	ADD DWORD PTR DS:[ESI+74DB3B5B],5D868102
0041884C	3BE4	CMP ESP,ESP
0041884E	74 01	JE SHORT 00418851
00418850	BF C20C0000	MOV EDI,0CC2
00418855	0000	ADD BYTE PTR DS:[EAX],AL
00418857	0000	ADD BYTE PTR DS:[EAX],AL
00418859	0000	ADD BYTE PTR DS:[EAX],AL
0041885B	0000	ADD BYTE PTR DS:[EAX],AL
0041885D	0000	ADD BYTE PTR DS:[EAX],AL
0041885F	0000	ADD BYTE PTR DS:[EAX],AL
00418861	0000	ADD BYTE PTR DS:[EAX],AL
00418863	0000	ADD BYTE PTR DS:[EAX],AL
00418865	0000	ADD BYTE PTR DS:[EAX],AL
00418867	0000	ADD BYTE PTR DS:[EAX],AL
00418869	0000	ADD BYTE PTR DS:[EAX],AL
0041886B	0000	ADD BYTE PTR DS:[EAX],AL

می بینید؟ اینجا جایی است که آن مقدار از حافظه خارج شده است. بنابراین ما به آخر کدهای پکر رسیدیم. پس اگر چند بار کلید F8 را بزنیم و به طول کامل از کدهای پکر خارج می شویم و به OEP می رسیم.
دو بار کلید F8 را بزنید:

Hex dump	Disassembly
- FFE0	JMP EAX
B8 01000000	MOV EAX,1
3BC9	CMP ECX,ECX
74 02	JE SHORT 00418837
81845F 3BD27401 BD5E38DE	ADD DWORD PTR DS:[EDI+EBX*2+174D23B1],DB3B5EBD
74 02	JE SHORT 00418844
8186 5B3BD874 0281865D	ADD DWORD PTR DS:[ESI+74DB3B5B],5D868102
3BE4	CMP ESP,ESP
74 01	JE SHORT 00418851
BF C20C0000	MOV EDI,0CC2
0000	ADD BYTE PTR DS:[EAX],AL
0000	ADD BYTE PTR DS:[EAX],AL
0000	ADD BYTE PTR DS:[EAX],AL
0000	ADD BYTE PTR DS:[EAX],AL
0000	ADD BYTE PTR DS:[EAX],AL
0000	ADD BYTE PTR DS:[EAX],AL

با این پرش از سکشن f. وارد سکشن text. می شویم. پس این پرش ما را به OEP می برد. یکبار کلید F7 را بزنید تا به OEP برسید، بعد هم کلیدهای CTRL + A را بزنید تا کدها آنالیز شود.

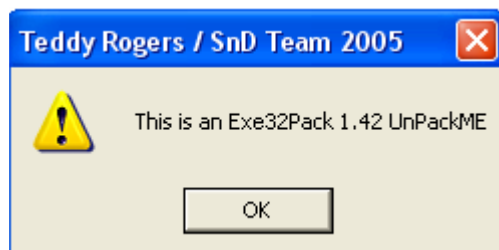
Hex dump	Disassembly	Comment
8B	WAIT	
DBE3	FINIT	
9B	WAIT	
DBE2	FCLEX	
D92D 00C04000	FLDCW WORD PTR DS:[40C000]	
55	PUSH EBP	
89E5	MOV EBP,ESP	
E8 81000000	CALL 0040A095	[pModule GetModuleHandleA]
68 00000000	PUSH 0	
FF15 E4114000	CALL DWORD PTR DS:[4011E4]	
A3 07504100	MOV DWORD PTR DS:[415007],EAX	
60	PUSHAD	
8925 0B504100	MOV DWORD PTR DS:[41500B],ESP	
E9 30000000	JMP 0040A060	
8B25 0B504100	MOV ESP,DWORD PTR DS:[41500B]	
61	POPAD	
E8 99050000	CALL 0040A5D5	
E8 ED000000	CALL 0040A12E	
89EC	MOV ESP,EBP	
5D	POP EBP	
FF35 04514100	PUSH DWORD PTR DS:[4151D4]	
FF15 DC114000	CALL DWORD PTR DS:[4011DC]	[ExitProcess]
9B	WAIT	
DBE2	FCLEX	
D92D 00C04000	FLDCW WORD PTR DS:[40C000]	
C3	RET	
00	DB 00	
00	DB 00	
00	DB 00	

بله... ما به OEP رسیدیم... برنامه هم در MASM یا TASM نوشته شده.

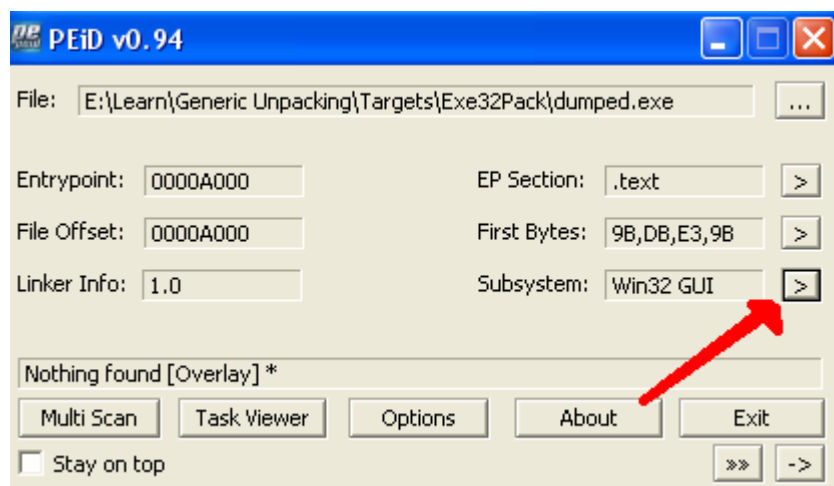
حالا فایل را با OllyDump دامپ می گیریم:

کلیک راست ← Dump Debugged process

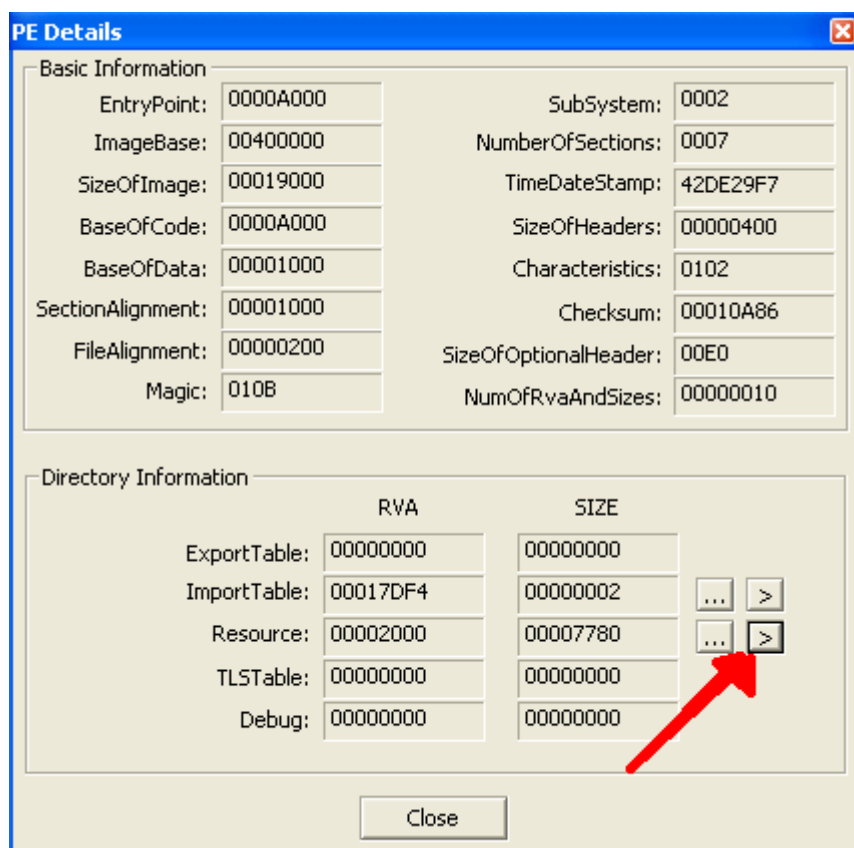
تیک گزینه Rebuild Import را بردارید و فایل را Dump کنید. حالا فایل دامپ شده را اجرا کنید:



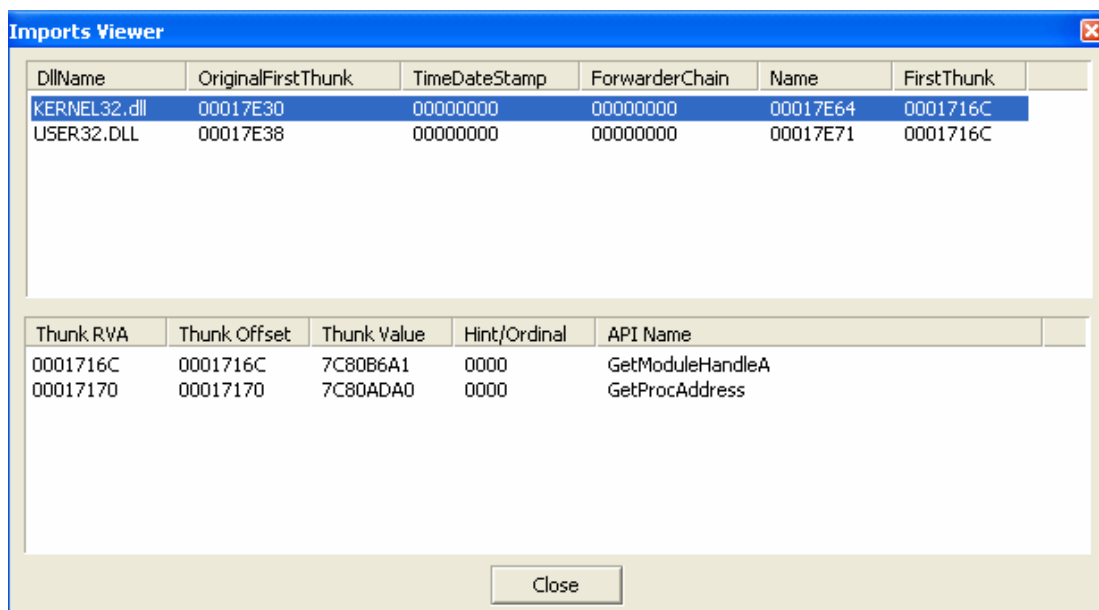
خوب، فایل آنپک شد... اما این فایل آنپک شده فقط در سیستم شما اجرا می شود، کافیه آن را در سیستم دیگر تست کنید، می بینید که اجرا نمی شود. چرا... چون این فایل فقط دامپ شده و اصلا IAT ندارد. پس قاعدتا نباید اجرا شود. دلیل اینکه این فایل در سیستم شما اجرا می شود، این است که آدرس توابع API هم به همراه فایل، دامپ شده است. برای اینکه مطمئن شویم این فایل IAT درست و حسابی ندارد، فایل آنپک شده را در PEID باز کنید:



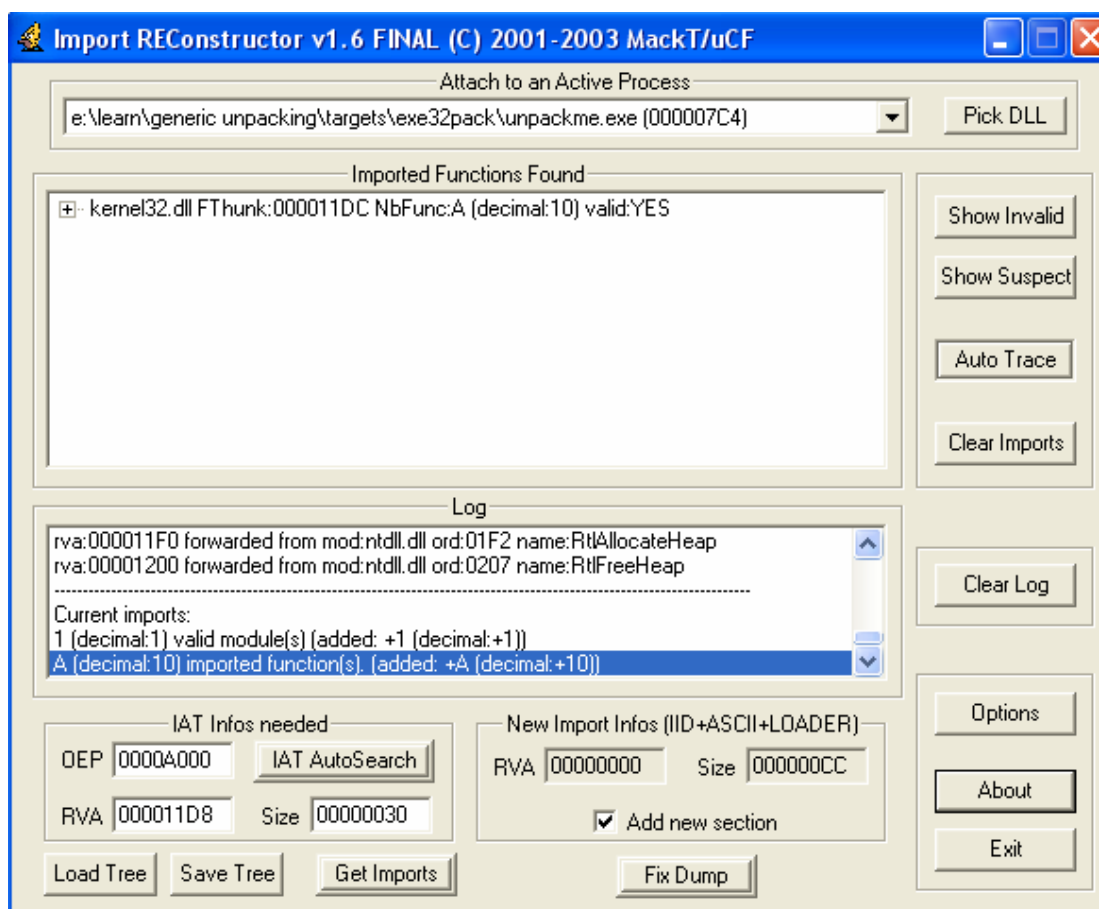
حالا این دکمه را بزنید:



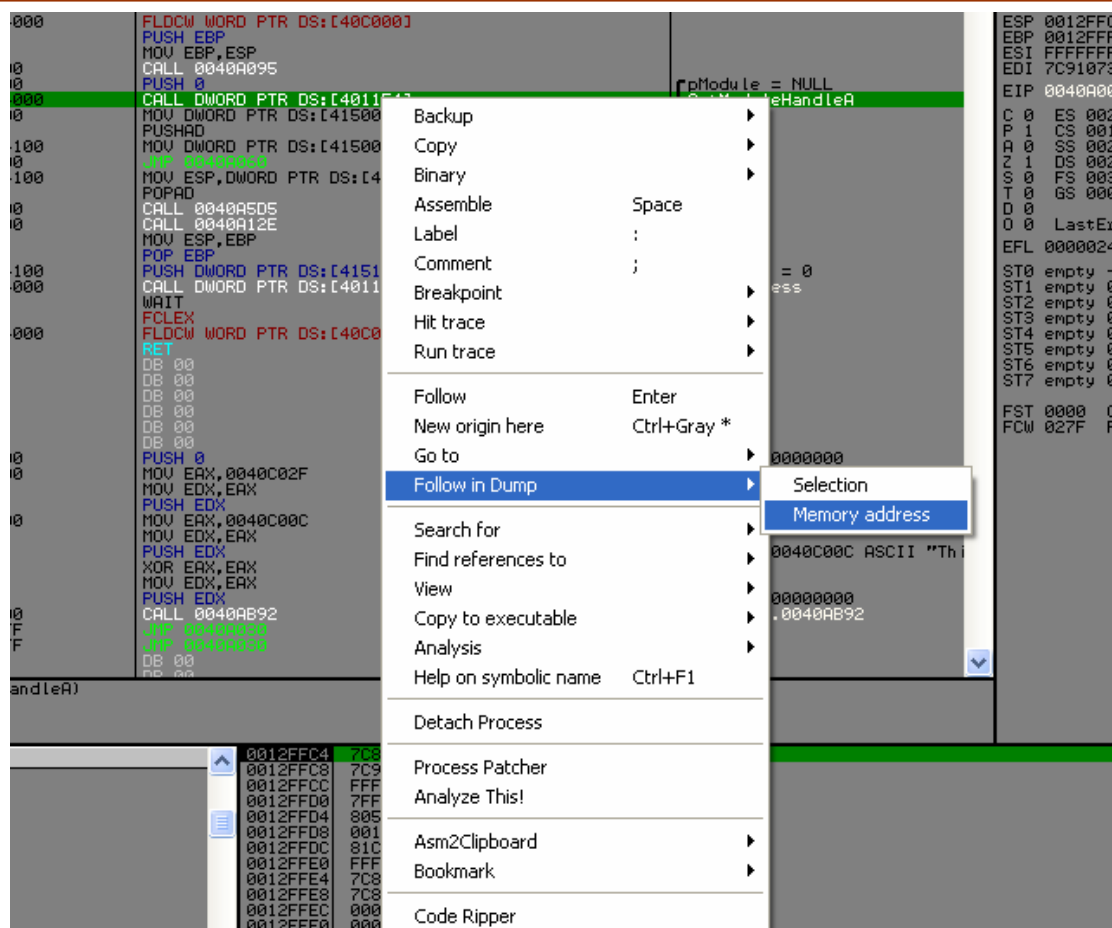
این IAT (در واقع Import Table) فایل ماست:



همانطور که می بینید فقط دو تابع GetProcAddress و GetModuleHandleA در فایل موجود است. و اصلا خبری از دیگر توابع API که برنامه استفاده کرده است (مثل MessageBoxA) نیست. پس لازم است IAT فایل را با ImportREC تعمیر کنیم. برنامه ImportREC را باز کنید، پروسه مورد نظر را انتخاب کنید. OEP را بر حسب RVA به برنامه بدهید. (A000). بعد گزینه IAT Auto-Search را بزنید. و بعد Get Imports:



لعنتی ImportREC © نتوانسته است IAT فایل مرا درست پیدا کند... از کجا فهمیدم؟... برنامه من از تابع MessageBoxA برای نمایش پیغام استفاده کرده بود. اما چنین تابعی در اینجا وجود ندارد. فقط ۱۰ تابع پیدا شده است که مربوط به فایل Kernel32.dll می باشد. پس بهتر است خودمان محل شروع IAT برنامه و اندازه آن را به دست آوریم. ImportREC که می گوید IAT ما از خط 11D8 شروع می شود. حالا دوباره به OllyDBG برگردید. و بر روی یکی از توابع کلیک راست کرده و گزینه Follow in Dump و سپس Memory Address را بزنید. من در تصویر زیر بر روی تابع GetModuleHandleA این کار را کردم.



Address	Hex dump	ASCII
004011E4	A1 B6 80 7C 2D FD 80 7C	! Ç!→Ç!
004011EC	2F FC 80 7C 04 05 91 7C	!Ç! Ç! Ç!
004011F4	1E 61 83 7C B6 2B 81 7C	!Ç! Ç! Ç!
004011FC	F8 0E 81 7C 3D 04 91 7C	!Ç! Ç! Ç!
00401204	00 00 00 00 00 00 00 00	
0040120C	00 00 00 00 00 00 00 00	
00401214	00 00 00 00 00 00 00 00	
0040121C	00 00 00 00 00 00 00 00	
00401224	00 00 00 00 00 00 00 00	
0040122C	00 00 00 00 00 00 00 00	
00401234	00 00 00 00 00 00 00 00	
0040123C	00 00 00 00 00 00 00 00	
00401244	00 00 00 00 00 00 00 00	
0040124C	00 00 00 00 00 00 00 00	
00401254	00 00 00 00 00 00 00 00	
0040125C	00 00 00 00 00 00 00 00	
00401264	00 00 00 00 00 00 00 00	
0040126C	00 00 00 00 00 00 00 00	
00401274	00 00 00 00 00 00 00 00	
0040127C	00 00 00 00 00 00 00 00	
00401284	00 00 00 00 00 00 00 00	
0040128C	00 00 00 00 00 00 00 00	
00401294	00 00 00 00 00 00 00 00	

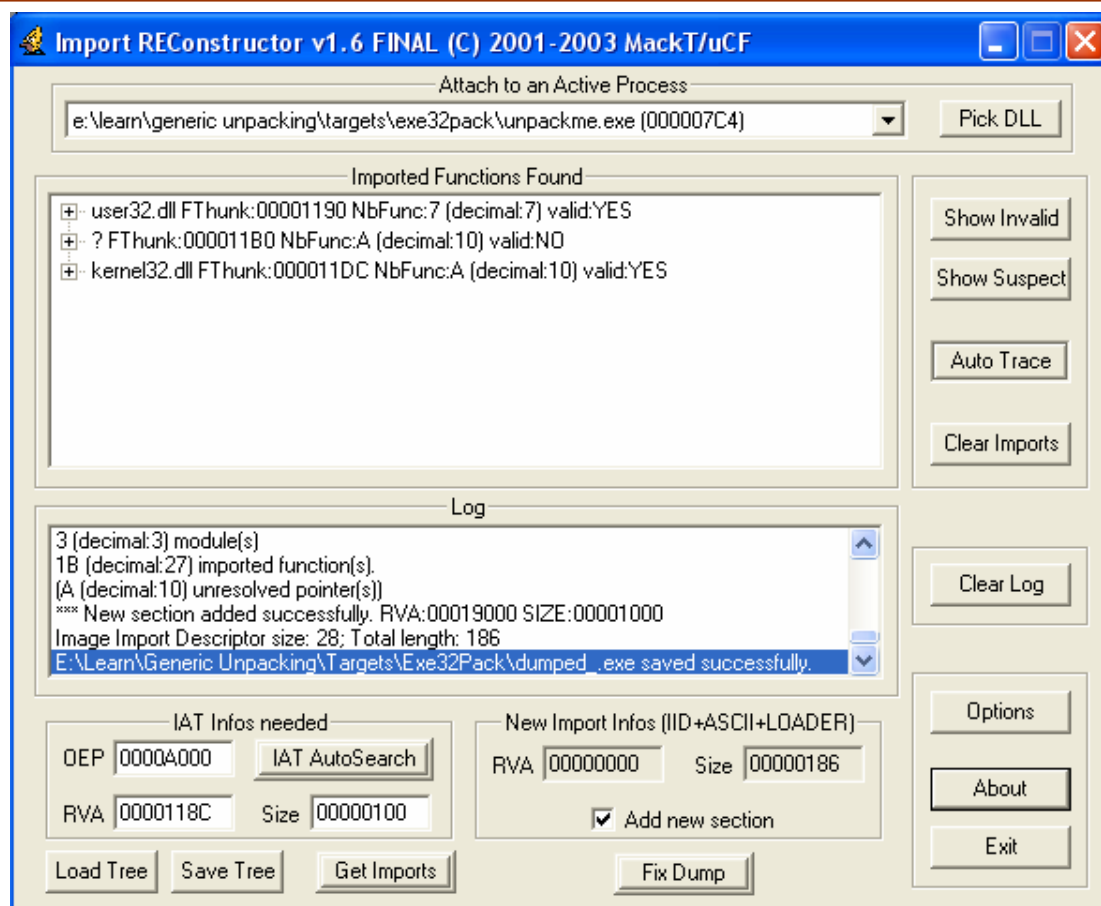
در پنجره Dump کلیک راست کرده گزینه Long و سپس گزینه Address with ASCII dump را بنویسید:

Address	Value	ASCI	Comment
0040110C	00001130	04..	
0040110D	0000113E	>4..	
0040110E	0000114C	L4..	
0040110F	00000000		
00401110	7C81CDDA	Fi	kernel32.ExitProcess
00401111	7C809728	(u	kernel32.GetCurrentThreadId
00401112	7C80B6A1	i	kernel32.GetModuleHandleA
00401113	7C80FD2D	-2	kernel32.GlobalAlloc
00401114	7C80FC2F	/n	kernel32.GlobalFree
00401115	7C9105D4	!a	ntdll.RtlAllocateHeap
00401116	7C83611E	Aa	kernel32.HeapCompact
00401117	7C812BB6	+u	kernel32.HeapCreate
00401118	7C810EF8	o u	kernel32.HeapDestroy
00401119	7C91043D	=a	ntdll.RtlFreeHeap
0040111A	00000000		
0040111B	00000000		
0040111C	00000000		
0040111D	00000000		
0040111E	00000000		
0040111F	00000000		
00401120	00000000		
00401121	00000000		
00401122	00000000		
00401123	00000000		
00401124	00000000		
00401125	00000000		
00401126	00000000		
00401127	00000000		
00401128	00000000		
00401129	00000000		
0040112A	00000000		
0040112B	00000000		
0040112C	00000000		
0040112D	00000000		
0040112E	00000000		
0040112F	00000000		
00401130	00000000		
00401131	00000000		
00401132	00000000		
00401133	00000000		
00401134	00000000		
00401135	00000000		
00401136	00000000		
00401137	00000000		
00401138	00000000		
00401139	00000000		
0040113A	00000000		
0040113B	00000000		
0040113C	00000000		
0040113D	00000000		
0040113E	00000000		
0040113F	00000000		
00401140	00000000		
00401141	00000000		
00401142	00000000		
00401143	00000000		
00401144	00000000		
00401145	00000000		
00401146	00000000		
00401147	00000000		
00401148	00000000		
00401149	00000000		
0040114A	00000000		
0040114B	00000000		
0040114C	00000000		
0040114D	00000000		
0040114E	00000000		
0040114F	00000000		
00401150	00000000		
00401151	00000000		
00401152	00000000		
00401153	00000000		
00401154	00000000		
00401155	00000000		
00401156	00000000		
00401157	00000000		
00401158	00000000		
00401159	00000000		
0040115A	00000000		
0040115B	00000000		
0040115C	00000000		
0040115D	00000000		
0040115E	00000000		
0040115F	00000000		
00401160	00000000		
00401161	00000000		
00401162	00000000		
00401163	00000000		
00401164	00000000		
00401165	00000000		
00401166	00000000		
00401167	00000000		
00401168	00000000		
00401169	00000000		
0040116A	00000000		
0040116B	00000000		
0040116C	00000000		
0040116D	00000000		
0040116E	00000000		
0040116F	00000000		
00401170	00000000		
00401171	00000000		
00401172	00000000		
00401173	00000000		
00401174	00000000		
00401175	00000000		
00401176	00000000		
00401177	00000000		
00401178	00000000		
00401179	00000000		
0040117A	00000000		
0040117B	00000000		
0040117C	00000000		
0040117D	00000000		
0040117E	00000000		
0040117F	00000000		
00401180	00000000		
00401181	00000000		
00401182	00000000		
00401183	00000000		
00401184	00000000		
00401185	00000000		
00401186	00000000		
00401187	00000000		
00401188	00000000		
00401189	00000000		
0040118A	00000000		
0040118B	00000000		
0040118C	00000000		
0040118D	00000000		
0040118E	00000000		
0040118F	00000000		
00401190	00000000		
00401191	00000000		
00401192	00000000		
00401193	00000000		
00401194	00000000		
00401195	00000000		
00401196	00000000		
00401197	00000000		
00401198	00000000		
00401199	00000000		
0040119A	00000000		
0040119B	00000000		
0040119C	00000000		
0040119D	00000000		
0040119E	00000000		
0040119F	00000000		
004011A0	00000000		
004011A1	00000000		
004011A2	00000000		
004011A3	00000000		
004011A4	00000000		
004011A5	00000000		
004011A6	00000000		
004011A7	00000000		
004011A8	00000000		
004011A9	00000000		
004011AA	00000000		
004011AB	00000000		
004011AC	00000000		
004011AD	00000000		
004011AE	00000000		
004011AF	00000000		
004011B0	00000000		
004011B1	00000000		
004011B2	00000000		
004011B3	00000000		
004011B4	00000000		
004011B5	00000000		
004011B6	00000000		
004011B7	00000000		
004011B8	00000000		
004011B9	00000000		
004011BA	00000000		
004011BB	00000000		
004011BC	00000000		
004011BD	00000000		
004011BE	00000000		
004011BF	00000000		
004011C0	00000000		
004011C1	00000000		
004011C2	00000000		
004011C3	00000000		
004011C4	00000000		
004011C5	00000000		
004011C6	00000000		
004011C7	00000000		
004011C8	00000000		
004011C9	00000000		
004011CA	00000000		
004011CB	00000000		
004011CC	00000000		
004011CD	00000000		
004011CE	00000000		
004011CF	00000000		
004011D0	00000000		
004011D1	00000000		
004011D2	00000000		
004011D3	00000000		
004011D4	00000000		
004011D5	00000000		
004011D6	00000000		
004011D7	00000000		
004011D8	00000000		
004011D9	00000000		
004011DA	00000000		
004011DB	00000000		
004011DC	00000000		
004011DD	00000000		
004011DE	00000000		
004011DF	00000000		
004011E0	00000000		
004011E1	00000000		
004011E2	00000000		
004011E3	00000000		
004011E4	00000000		
004011E5	00000000		
004011E6	00000000		
004011E7	00000000		
004011E8	00000000		
004011E9	00000000		
004011EA	00000000		
004011EB	00000000		
004011EC	00000000		
004011ED	00000000		
004011EE	00000000		
004011EF	00000000		
004011F0	00000000		
004011F1	00000000		
004011F2	00000000		
004011F3	00000000		
004011F4	00000000		
004011F5	00000000		
004011F6	00000000		
004011F7	00000000		
004011F8	00000000		
004011F9	00000000		
004011FA	00000000		
004011FB	00000000		
004011FC	00000000		
004011FD	00000000		
004011FE	00000000		
004011FF	00000000		

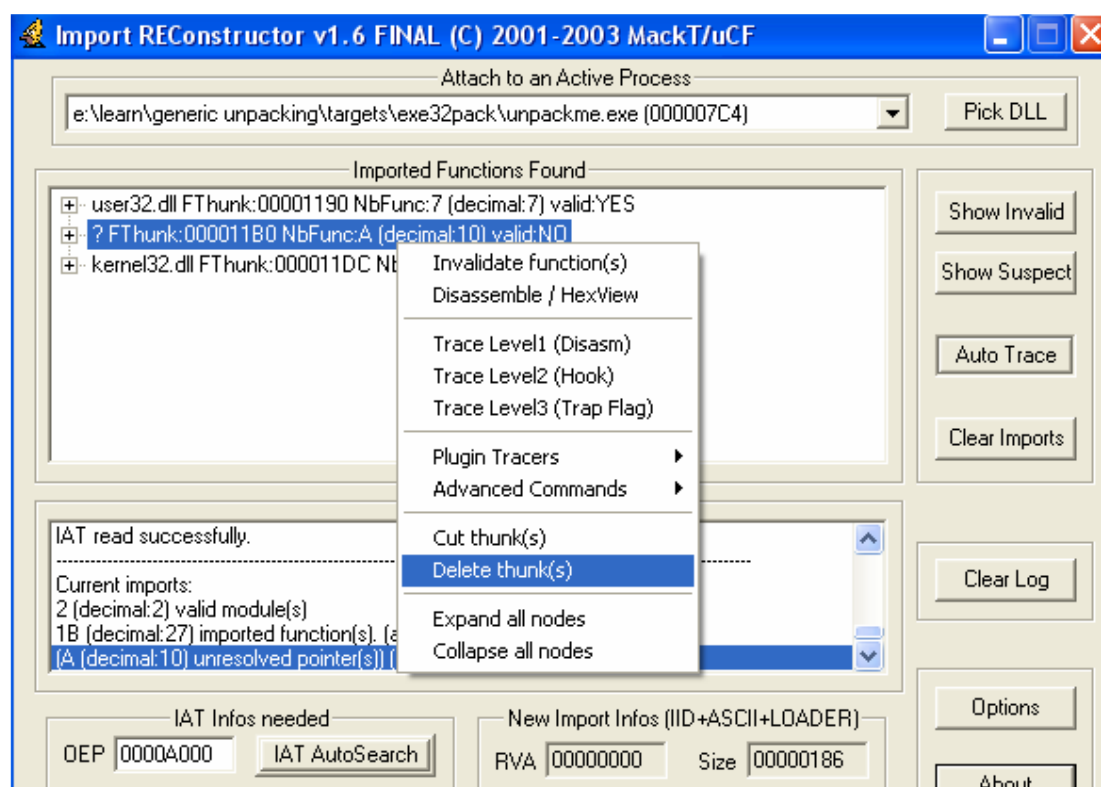
همانطور که می بینید به نظر ImportREC درست حدس زده است و IAT ما از خط 4011D8 شروع می شود. ولی اگر در پنجره Dump اندکی به بالا بروید:

00401184	00001094	6>..	
00401188	000010AC	%>..	
0040118C	00000000		
00401190	7E41F85B	[>A"	USER32.CallNextHookEx
00401194	7E41EED5	FeA"	USER32.GetDesktopWindow
00401198	7E41B6D4	H A"	USER32.GetWindowRect
0040119C	7E45058A	e+E"	USER32.MessageBoxA
004011A0	7E4311D1	T4C"	USER32.SetWindowsHookExA
004011A4	7E420762	b.B"	USER32.SystemParametersInfoA
004011A8	7E41F21E	A>A"	USER32.UnhookWindowsHookEx
004011AC	00000000		
004011B0	000010C2	T>..	
004011B4	000010D0	L>..	
004011B8	000010E6	p>..	
004011BC	000010FA	r>..	
004011C0	00001108	Q>..	
004011C4	00001116	>4..	
004011C8	00001122	>4..	
004011CC	00001130	04..	
004011D0	0000113E	>4..	
004011D4	0000114C	L4..	
004011D8	00000000		
004011DC	7C81CDDA	Fi	kernel32.ExitProcess
004011E0	7C809728	(u	kernel32.GetCurrentThreadId

چند تا تابع API اینجا قرار دارد. ☺ پس IAT ما از خط 40118C یا بر حسب RVA از خط 118C شروع می شود. خوب، دوباره به ImportREC برگردید. این بار در قسمت RVA بنویسید : 118C و در قسمت Size بنویسید : 100 ... چون این برنامه بسیار کوچک هست... اندازه IAT آن هر چه باشد، بزرگتر از صد نیست. به همین دلیل من در اینجا نوشتم 100 حالا گزینه Get Imports را بزنید:



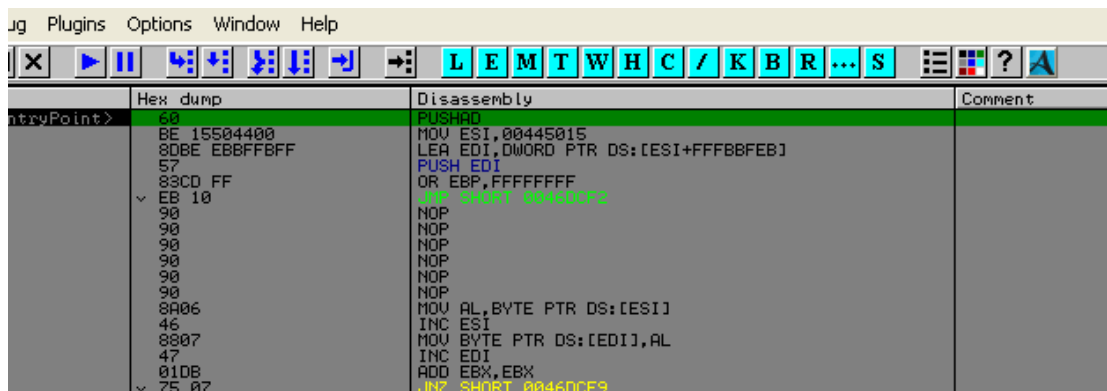
همانطور که انتظارش را داشتیم در بین توابع user32 و توابع kernel32 چند تابع Invalid وجود دارد. پس بهتر است این Thunk را کلاً از بین ببریم... هر چند وجود آن مشکلی ایجاد نمی‌کند و ما می‌توانیم در همین حالت که توابع Invalid هست، دامپ خودمان را فیکس کنیم... ولی بهتر است Invalid ها را از بین ببریم، همانند تصویر بر روی Thunk کلیک راست کرده و گزینه Delete Thunk را بزنید تا این Thunk کلاً پاک شود و تمام توابع Valid شوند:



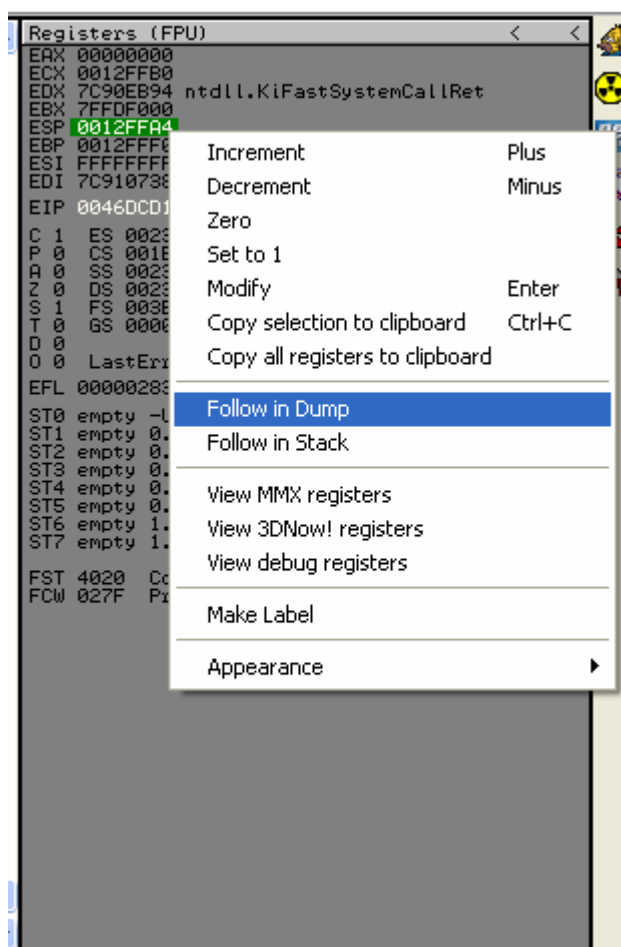
حالا دامپ خود را فیکس کنید، می‌بینید که فایل دامپ شده شما در هر سیستمی اجرا می‌شود ☺

UPX

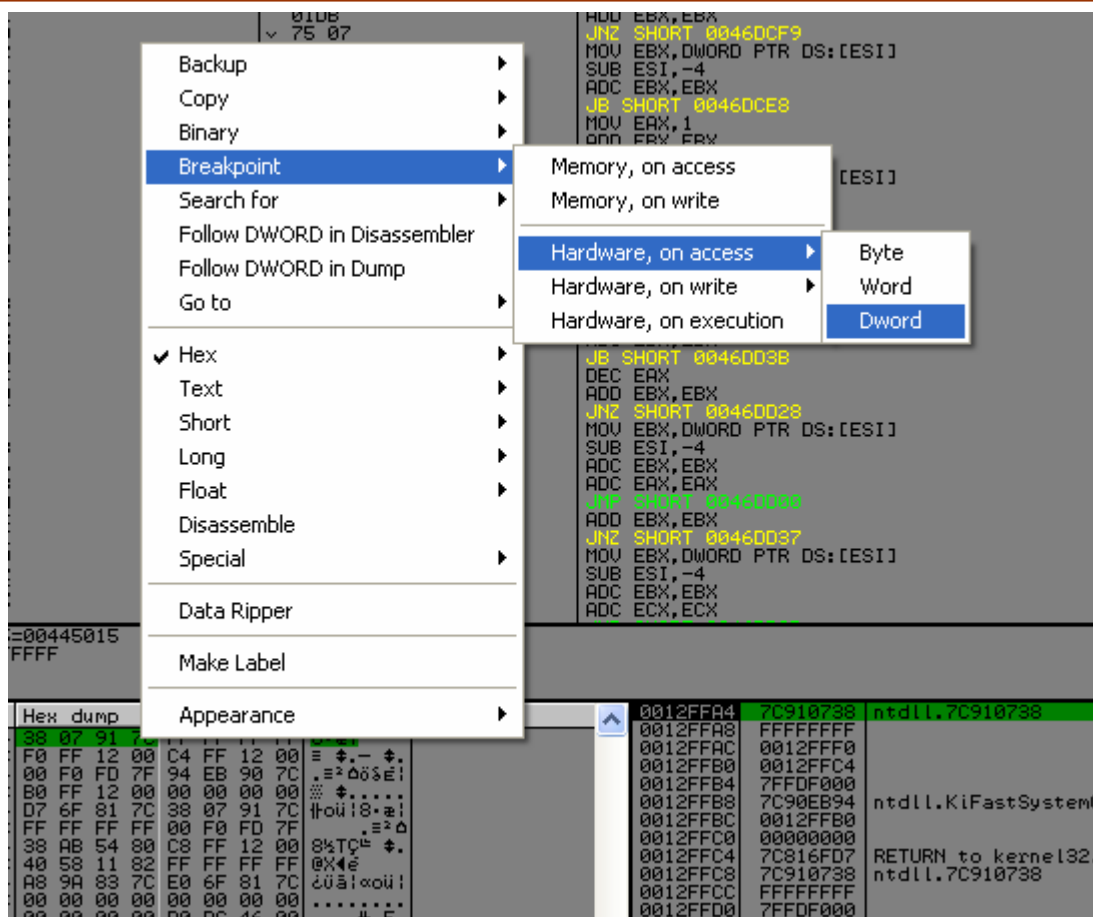
فایل را در OllyDBG باز کنید:



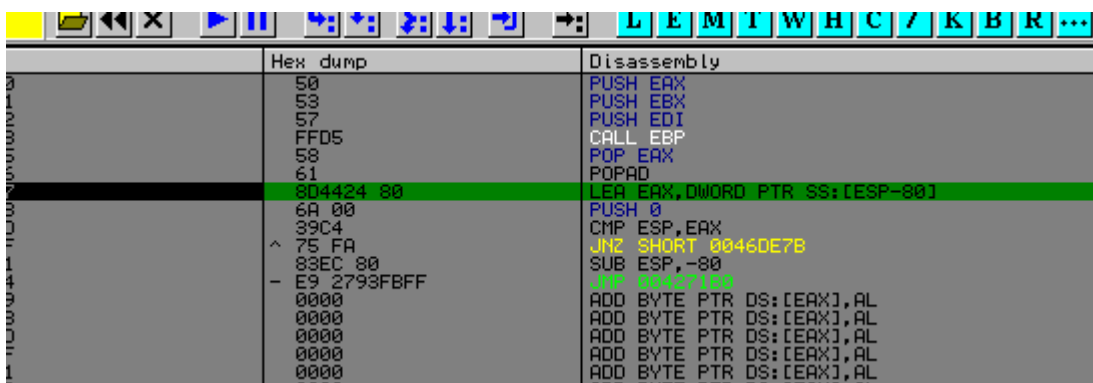
همانطور که می بینید در اولین خط از پکر ما دستور PushAD اجرا می شود. این دستور تمامی رجیسترهای ما را به غیر از EIP وارد حافظه Stack می کند. پس چون مقادیری وارد حافظه Stack می شود... مقدار ESP تغییر می کند و ما می توانیم با Hardware Breakpoint گذاشتن روی رجیستر ESP جایی که این مقادیر از حافظه خارج می شود، یعنی انتهای کدهای پکر خود را پیدا کنیم، یک بار کلید F8 را بزنید تا این دستور اجرا شود، حالا همانند تصویر عمل کنید:



و بعد:



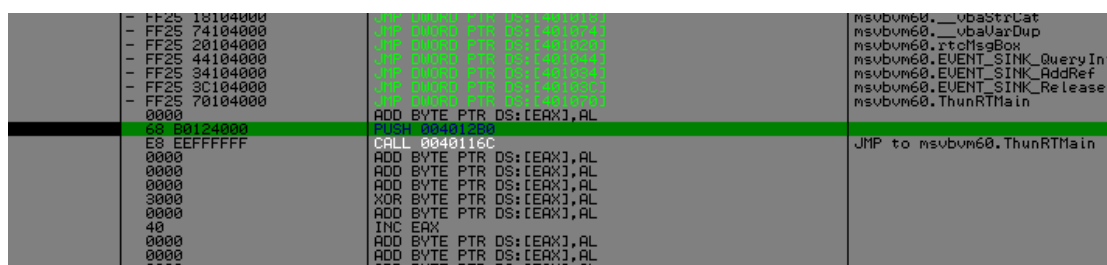
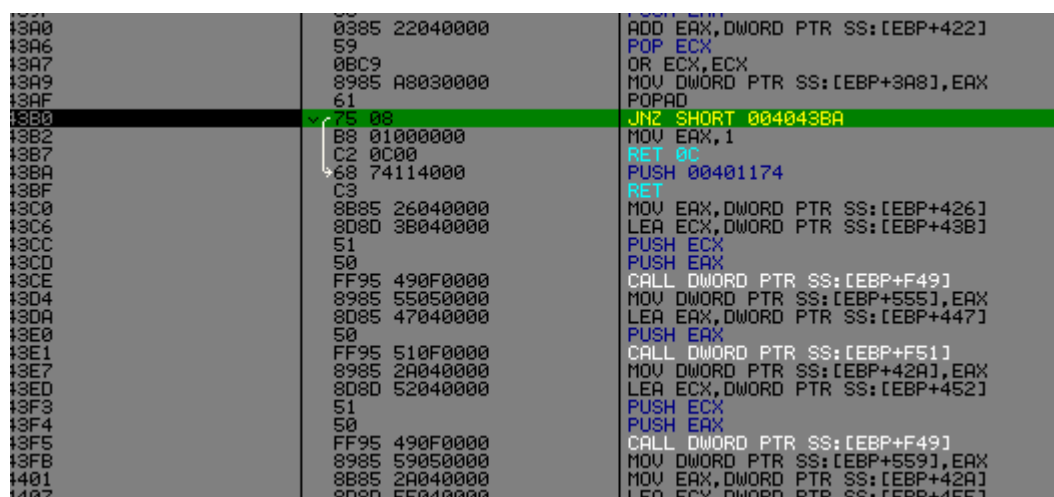
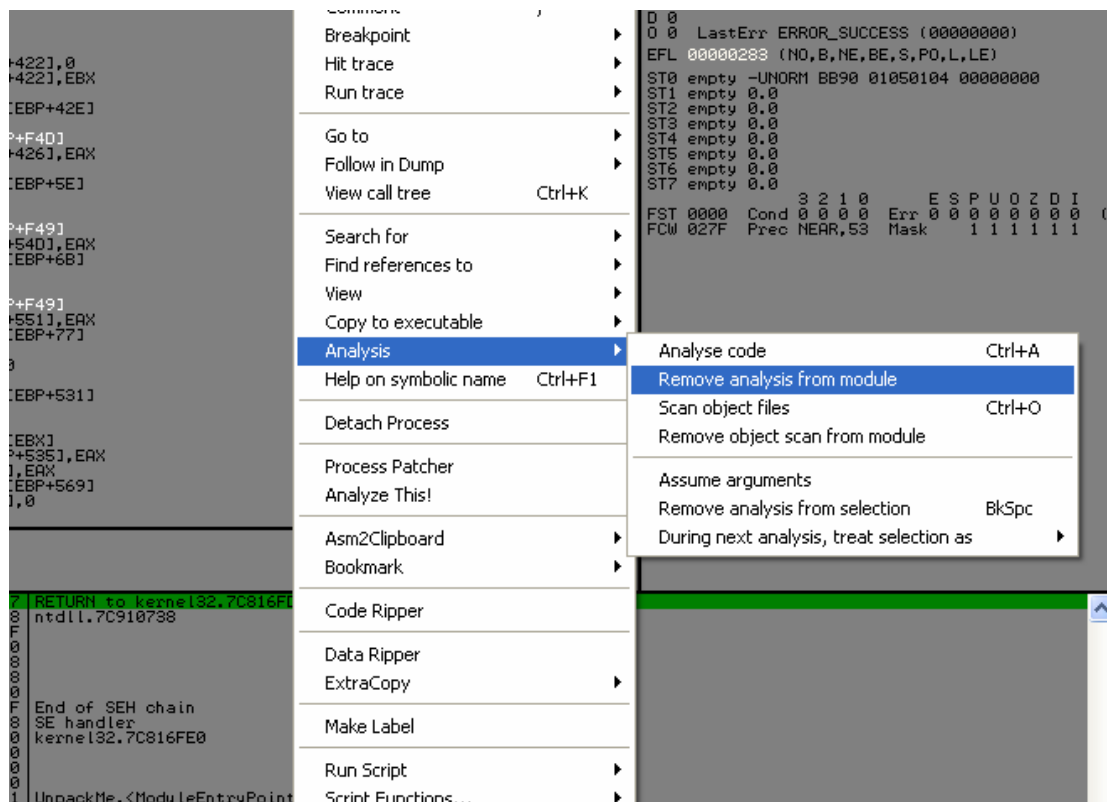
حالا بعد از Hardware Breakpoint گذاشتن روی مقدار رجیستر ESP، برنامه را با F9 اجرا می کنیم تا به آخر کدهای پکر برسیم:



خوب به آخر کدهای پکر رسیدیم. در اینجا یک حلقه وجود دارد که این حلقه چند مقدار صفر را وارد حافظه Stack می کند و بعد با دستور JMP 004271B0 که آخرین دستور پکر ماست به OEP می ریم. پس بر روی دستور JMP 004271B0 یک Breakpoint می گذاریم و با F9 برنامه را اجرا می کنیم و بعد کلید F7 را می زنیم تا به OEP برسیم:

B	90	NOP
C	90	NOP
D	90	NOP
E	90	NOP
F	90	NOP
0	55	PUSH EBP
1	8BEC	MOV EBP,ESP
3	6A FF	PUSH -1
5	68 600E4500	PUSH 00450E60
6	68 C8924200	PUSH 004292C8
7	64:A1 00000000	MOV EAX,DWORD PTR FS:[0]
8	50	PUSH EAX
9	64:8925 00000000	MOV DWORD PTR FS:[0],ESP
A	83C4 A8	ADD ESP,-58
B	53	PUSH EBX
C	56	PUSH ESI
D	57	PUSH EDI
E	8965 E8	MOV DWORD PTR SS:[EBP-18],ESP
6	FF15 DC0A4600	CALL DWORD PTR DS:[460ADC]
7	33D2	XOR EDX,EDX
8	8AD4	MOV DL,AH
9	8915 34E64500	MOV DWORD PTR DS:[45E634],EDX
A	8BC8	MOV ECX,EAX
B	81E1 FF000000	AND ECX,0FF
C	890D 30E64500	MOV DWORD PTR DS:[45E630],ECX
D	C1E1 08	SHL ECX,8
E	03CA	ADD ECX,EDX
7	890D 2CE64500	MOV DWORD PTR DS:[45E62C],ECX
9	8BFC	MOV ECX,EBP

خوب، بعد از اینکه دیگر گمون نمی کنم نیازی به توضیح داشته باشد. فایل را دامپ می کنید و با ImportREC دامپ را فیکس می کنید.



بله، برنامه ما در ویژوال بیسیک نوشته شده...از اینجا دیگر مشکلی نخواهید داشت...کافیست فایل را با برنامه مورد علاقه خود دامپ بگیرید و با ImportREC دامپ را فیکس کنید.

AHPacker

فایل مورد نظر را در OllyDBG باز می کنیم، اولین دستور PushAD است. یک بار کلید F8 را می زنیم تا این دستور اجرا شود. بر روی مقدار رجیستر ESP یک Hardware breakpoint on access می گذاریم و با F9 برنامه را اجرا می کنیم تا به اینجا برسیم:

Address	Hex dump	Disassembly	Comment
0040A29A	BA 00104000	MOV EDX, 00401000	ASC
0040A29F	FFE2	JMP EDX	
0040A2A1	90	NOP	
0040A2A2	C3	RET	
0040A2A3	44	INC ESP	
0040A2A4	0000	ADD BYTE PTR DS:[EAX], AL	
0040A2A6	0000	ADD BYTE PTR DS:[EAX], AL	
0040A2A8	0000	ADD BYTE PTR DS:[EAX], AL	
0040A2AA	0000	ADD BYTE PTR DS:[EAX], AL	
0040A2AC	0000	ADD BYTE PTR DS:[EAX], AL	
0040A2AE	0000	ADD BYTE PTR DS:[EAX], AL	
0040A2B0	0000	ADD BYTE PTR DS:[EAX], AL	
0040A2B2	0000	ADD BYTE PTR DS:[EAX], AL	
0040A2B4	0000	ADD BYTE PTR DS:[EAX], AL	
0040A2B6	0000	ADD BYTE PTR DS:[EAX], AL	
0040A2B8	0000	ADD BYTE PTR DS:[EAX], AL	
0040A2BA	0000	ADD BYTE PTR DS:[EAX], AL	
0040A2BC	0000	ADD BYTE PTR DS:[EAX], AL	
0040A2BE	0000	ADD BYTE PTR DS:[EAX], AL	
0040A2C0	0000	ADD BYTE PTR DS:[EAX], AL	
0040A2C2	0000	ADD BYTE PTR DS:[EAX], AL	
0040A2C4	0000	ADD BYTE PTR DS:[EAX], AL	

این دیگر آخرین کدهای پکر ماست، دستور JMP EDX ما را به OEP می برد ☺
بعد از آن هم مشکلی به وجود نمی آید. فایل را دامپ می کنید و با ImportREC فیکس می کنید.

EXpressor

اولین دستور ما Push EBP است. یعنی مقدار EBP وارد Stack می شود. پس مقدار رجیستر ESP تغییر می کند. یک بار کلید F8 را می زنیم تا این دستور اجرا شود و بعد بر روی مقدار رجیستر ESP یک Hardware Breakpoint می گذاریم و با F9 برنامه را اجرا می کنیم، ابتدا پیغامی مربوط به رجیستر نبودن پکر داده می شود و بعد به اینجا می رسیم:

5E	POP ESI	
5B	POP EBX	
83C4 64	ADD ESP, 64	
5D	POP EBP	
FFE0	JMP EAX	UnPackMe.004271B0
5F	POP EDI	
5E	POP ESI	
5B	POP EBX	
C9	LEAVE	
C3	RET	
B8 02400080	MOV EAX, 80004002	
C2 0C00	RET 0C	
8B4424 04	MOV EAX, DWORD PTR SS:[ESP+4]	
FF40 04	INC DWORD PTR DS:[EAX+4]	
8B40 04	MOV EAX, DWORD PTR DS:[EAX+4]	
C2 0400	RET 4	
8B4C24 04	MOV ECX, DWORD PTR SS:[ESP+4]	
FF49 04	DEC DWORD PTR DS:[ECX+4]	
8B41 04	MOV EAX, DWORD PTR DS:[ECX+4]	
75 09	JNZ SHORT 0046BE18	
51	PUSH ECX	
E8 B9020000	CALL 0046C0CE	
59	POP ECX	
33C0	XOR EAX, EAX	
C2 0400	RET 4	
8D41 0C	LEA EAX, DWORD PTR DS:[ECX+C]	
8D48 04	LEA ECX, DWORD PTR DS:[EAX+4]	
51	PUSH ECX	
50	PUSH EAX	
E8 57000000	CALL 0046BE7F	

این دستور درست به OEP می رود. پس یک بار کلید F8 ما را به OEP می برد. بعد از این هم دیگر مشکلی پیش نمی آید. فایل را دامپ کنید و با ImportREC فیکس کنید. همین!

NSPack

اولین دستور ما PushFD است. اصولاً بهتر است بعد از اجرا شدن دستور PushAD روی مقدار رجیستر ESP یک hardware breakpoint بگذاریم. پس دو بار کلید F8 را می زنیم تا دستور PushAD اجرا شود و بعد روی مقدار رجیستر ESP یک hardware breakpoint on access می گذاریم و با F9 برنامه را اجرا می کنیم تا در اینجا متوقف شویم:

Hex dump	Disassembly	Comments
B8 00000000	MOV EAX,0	
83F8 00	CMP EAX,0	
74 0A	JE SHORT 0046D615	
61	POPAD	
9D	POPFD	
B8 01000000	MOV EAX,1	
C2 0C00	RET 0C	
61	POPAD	
9D	POPFD	
E9 9498FBFF	JMP 004271B8	
8BB5 65FCFFFF	MOV ESI,DWORD PTR SS:[EBP-39B]	
0BF6	OR ESI,ESI	
0F84 97000000	JE 0046D6C1	
8B95 6DFCFFFF	MOV EDX,DWORD PTR SS:[EBP-393]	
03F2	ADD ESI,EDX	
833E 00	CMP DWORD PTR DS:[ESI],0	
75 0E	JNZ SHORT 0046D645	
837E 04 00	CMP DWORD PTR DS:[ESI+4],0	
75 0E	JNZ SHORT 0046D645	

خوب، اینجا دیگر آخر کدهای پکر ماست و دستور JMP 004271B0 ما را به OEP می برد. حالا که OEP را پیدا کردیم، کافیه از فایل دامپ بگیریم و با ImportREC آن را فیکس کنیم.

NeoLite

در مورد این پکر هم خیلی راحت می شود OEP را پیدا کرد. کافیه که چند بار کلید F8 را بزنییم تا به دستور JMP EAX برسیم... ممکن است قبل از رسیدن به این دستور پیامی مبنی بر رجیستر نبودن NeoLite داده شود که با زدن کلید No می توانید این پیام را از بین ببرید، در اینجا ما به دستور JMP Eax در خط 004311D0 رسیدیم. این دستور ما را از سکشن neolit به سکشن text می برد. یعنی ما را به OEP می رساند. اصولاً طبق تجربه ای که به دست آوردیم... معمولاً دستورهای مثل:

JMP Register (JMP Eax, JMP EBX, JMP EBP....)

Call Register (Call EAX, Call EBX, Call ECX...)

Return

دستورهای هستند که ممکن است ما را به OEP ببرند، در اینجا دستور JMP Eax ما را به OEP برد. حالا یک بار کلید F8 را بزنیید تا به OEP برسید. همانطور که می بینید برنامه ما در MASM نوشته شده:

Hex dump	Disassembly	Comment
6A 00	PUSH 0	
E8 30070000	CALL 00401744	JMP to key
A3 C0824000	MOV DWORD PTR DS:[4082C0],EAX	
6A 00	PUSH 0	
68 2B104000	PUSH 0040102B	
6A 00	PUSH 0	
68 E9030000	PUSH 3E9	
FF35 C0824000	PUSH DWORD PTR DS:[4082C0]	
E8 AD060000	CALL 004016D2	JMP to use
50	PUSH EAX	
E8 0D070000	CALL 00401738	JMP to key
55	PUSH EBP	
8BEC	MOV EBP,ESP	
817D 0C 10010000	CMP DWORD PTR SS:[EBP+C],110	
0F85 4D010000	JNZ 00401188	
6A EC	PUSH -14	
FF75 08	PUSH DWORD PTR SS:[EBP+8]	
E8 A5060000	CALL 004016EA	JMP to use
0D 00000000	OR EAX,00000	
50	PUSH EAX	
6A EC	PUSH -14	
FF75 08	PUSH DWORD PTR SS:[EBP+8]	
E8 CB060000	CALL 00401720	JMP to use
6A 00	PUSH 0	

بعد از این دیگر کاری نداره. فایل را دامپ می کنید و با ImportREC فیکس می کنید.

UPolyX

برنامه را در OllyDBG باز کنید. همانطور که می بینید در Entry point برنامه یکسری کدهای عجیب و غریب وجود دارد:

Address	Hex dump	Disassembly
004B469F <ModuleEntryPoint>	89EE	MOV ESI,EBP
004B46A1	F3:	PREFIX REP:
004B46A2	C1F3 91	SAL EBX,91
004B46A5	FEC8	DEC AL
004B46A7	D1D6	RCL ESI,1
004B46A9	C0DC DB	RCR AH,0DB
004B46AC	80CA 69	OR DL,69
004B46AF	F3:	PREFIX REP:
004B46B0	11EE	ADC ESI,EBP
004B46B2	0FA3FD	BT EBP,EDI
004B46B5	0FA4D3 41	SHLD EBX,EDX,41
004B46B9	0FBDC3	BSR EAX,EBX
004B46BC	0FB9FF 0D	BTC EDI,0D
004B46C0	85C3	TEST EBX,EAX
004B46C2	1C 5B	SBB AL,5B
004B46C4	F3:	PREFIX REP:
004B46C5	0FA5C1	SHLD ECX,EAX,CL
004B46C8	BE 352457BE	MOV ESI,BE572435
004B46CD	0FAFC8	IMUL ECX,EAX
004B46D0	87F1	XCHG ECX,ESI
004B46D2	69EF 930A05D4	IMUL EBP,EDI,04A50A93
004B46D8	69FE 756497FE	IMUL EDI,ESI,FE976475
004B46DE	F7C3 F4E70E99	TEST EBX,990EE7F4
004B46E4	65:85C3	TEST EBX,EAX
004B46E7	B4 23	MOV AH,23
004B46E9	69D5 7170736A	IMUL EDX,EBP,6A737071
004B46EF	84F1	TEST CL,DH
004B46F1	0FB7D9	MOVZX EBX,CX
004B46F4	0FA5C1	SHLD ECX,EAX,CL
004B46F7	C1D6 D0	RCL ESI,0DD
004B46FA	0FA5F7	SHLD EDI,ESI,CL
004B46FD	C0DC AB	RCR AH,0AB
004B4700	D2CA	ROR DL,CL
004B4702	25 2457BE89	AND EAX,89BE5724
004B4707	F6D8	NEG AL
004B4709	11EE	ADC ESI,EBP
004B470B	F3:	PREFIX REP:
004B470C	43	INC EBX
004B470D	89F9	MOV ECX,EDI
004B470F	F7C0 2D3C0F96	TEST EAX,960F3C2D
004B4715	85DA	TEST EDX,EBX
004B4717	89EE	MOV ESI,EBP
004B4719	8D2D CB629DEC	LEA EBP,DWORD PTR DS:[EC9D62CB]
004B471F	C7C6 6D7C4FD6	MOV ESI,D64F7C6D
004B4725	87C8	XCHG EAX,ECX
004B4727	0FB9FE	BSF EDI,ESI
004B472A	F6DC	NEG AH
004B472C	F7D3	NOT EBX
004B472E	0FB9E9 04	BTS ECX,4
004B4732	13F5	ADC ESI,EBP

خوب برنامه را با F8 به جلو می بریم. تا به این حلقه برسیم:

004B47CE	0FBDC3	BSR EAX,EBX
004B47D1	C1D6 05	RCL ESI,5
004B47D4	55	PUSH EBP
004B47D5	8BEC	MOV EBP,ESP
004B47D7	B8 6E000000	MOV EAX,6E
004B47DC	50	PUSH EAX
004B47DD	59	POP ECX
004B47DE	B8 00454B00	MOV EAX,004B4500
004B47E3	50	PUSH EAX
004B47E4	8000 B4	ADD BYTE PTR DS:[EAX],0B4
004B47E7	83C0 02	ADD EAX,2
004B47EA	83E9 01	SUB ECX,1
004B47ED	E2 F5	LOOPD SHORT 004B47E4
004B47EF	C3	RET
004B47F0	0000	ADD BYTE PTR DS:[EAX],AL
004B47F2	0000	ADD BYTE PTR DS:[EAX],AL
004B47F4	0000	ADD BYTE PTR DS:[EAX],AL
004B47F6	0000	ADD BYTE PTR DS:[EAX],AL

Loop is taken
ECX=00000039 (decimal 57.)
004B47E4=004B47E4

Address	Hex dump	ASCII	0012FFBC	004B4500	UnPackMe.004B4500
---------	----------	-------	----------	----------	-------------------

برای رد کردن این حلقه روی RET پایینی Breakpoint گذاشته و با F9 برنامه را اجرا کنید. حالا یک بار F8 را بزنید تا به اینجا برسید:

Address	Hex dump	Disassembly
500	60	PUSHAD
501	BE 00704700	MOV ESI,00477000
506	80BE 00A0F8FF	LEA EDI,DWORD PTR DS:[ESI+FFF8A000]
50C	C787 10770900 49061B91	MOV DWORD PTR DS:[EDI+977101,911B0649]
516	57	PUSH EDI
517	83CD FF	OR EBP,FFFFFFFF
51A	EB 0E	JMP SHORT 004B4528
51C	90	NOP
51D	90	NOP
51E	90	NOP
51F	90	NOP
520	8A06	MOV AL,BYTE PTR DS:[ESI]
522	46	INC ESI
523	8807	MOV BYTE PTR DS:[EDI],AL
525	47	INC EDI
526	01DB	ADD EBX,EBX
528	75 07	JNZ SHORT 004B4531
52A	8B1E	MOV EBX,DWORD PTR DS:[ESI]
52C	83EE FC	SUB ESI,-4
52F	11DB	ADC EBX,EBX
531	72 ED	JB SHORT 004B4520
533	B8 01000000	MOV EAX,1

همانطور که می بینید به دستور PushAD رسیدیم.حالا می توانیم از رجیستر ESP برای پیدا کردن آخر کدهای پکر استفاده کنیم.برای این کار یک بار کلید F8 را می زنیم و بر روی مقدار رجیستر ESP یک Hardware breakpoint on access می گذاریم.و بعد کلید F9 را می زنیم تا به آخر کدهای پکر برسیم:

Address	Hex dump	Disassembly
4644	74 0C	JE SHORT 004B4628
464C	89F9	MOV ECX,EDI
464E	57	PUSH EDI
464F	48	DEC EAX
4650	F2:AE	REPNE SCAS BYTE PTR ES:[EDI]
4652	55	PUSH EBP
4653	FF96 28510B00	CALL DWORD PTR DS:[ESI+B5128]
4659	09C0	OR EAX,EAX
465B	74 07	JE SHORT 004B4664
465D	8903	MOV DWORD PTR DS:[EBX],EAX
465F	83C3 04	ADD EBX,4
4662	EB E1	JMP SHORT 004B4645
4664	FF96 2C510B00	CALL DWORD PTR DS:[ESI+B512C]
466A	61	POPAD
466B	74 0C	JE SHORT 004B4628
4670	8846 4B	MOV BYTE PTR DS:[ESI+4B],AL
4673	0098 464B0010	ADD BYTE PTR DS:[EAX+10004B46],BL
4679	8749 00	XCHG DWORD PTR DS:[ECX],ECX
467C	0000	ADD BYTE PTR DS:[EAX],AL
467E	0000	ADD BYTE PTR DS:[EAX],AL
4680	0000	ADD BYTE PTR DS:[EAX],AL
4682	0000	ADD BYTE PTR DS:[EAX],AL
4684	0000	ADD BYTE PTR DS:[EAX],AL
4686	0000	ADD BYTE PTR DS:[EAX],AL

این پرش به OEP می رود.کافیست یک بار کلید F8 را بزیند تا به OEP برسید ©

Address	Hex dump	Disassembly
004958EC	55	PUSH ESP
004958ED	8BEC	MOV EBP,ESP
004958EF	83C4 F0	ADD ESP,-10
004958F2	53	PUSH EBX
004958F3	B8 84564900	MOV EAX,00495684
004958F8	E8 C712F7FF	CALL 00406BC4
004958FD	8B1D 08744900	MOV EBX,DWORD PTR DS:[497408]
00495903	8B03	MOV EAX,DWORD PTR DS:[EBX]
00495905	E8 8E19F0FF	CALL 00467298
0049590A	8B0D 00744900	MOV ECX,DWORD PTR DS:[497400]
00495910	8B03	MOV EAX,DWORD PTR DS:[EBX]
00495912	8B15 08064900	MOV EDX,DWORD PTR DS:[490608]
00495918	E8 9319F0FF	CALL 004672B0
0049591D	8B0D B4724900	MOV ECX,DWORD PTR DS:[4972B4]
00495923	8B03	MOV EAX,DWORD PTR DS:[EBX]
00495925	8B15 74FD4900	MOV EDX,DWORD PTR DS:[48FD74]
0049592B	E8 8019F0FF	CALL 004672B0
00495930	8B0D 6C744900	MOV ECX,DWORD PTR DS:[49746C]
00495936	8B03	MOV EAX,DWORD PTR DS:[EBX]
00495938	8B15 4C024900	MOV EDX,DWORD PTR DS:[49024C]
0049593E	E8 6D19F0FF	CALL 004672B0
00495943	8B0D 0C734900	MOV ECX,DWORD PTR DS:[49730C]
00495949	8B03	MOV EAX,DWORD PTR DS:[EBX]
0049594B	8B15 04044900	MOV EDX,DWORD PTR DS:[490404]
00495951	E8 5A19F0FF	CALL 004672B0
00495956	8B03	MOV EAX,DWORD PTR DS:[EBX]
00495958	E8 D319F0FF	CALL 00467330
0049595D	5B	POP EBX
0049595E	E8 31EBF6FF	CALL 00404494
00495963	90	NOP
00495964	0000	ADD BYTE PTR DS:[EAX],AL
00495966	0000	ADD BYTE PTR DS:[EAX],AL
00495968	0000	ADD BYTE PTR DS:[EAX],AL
0049596A	0000	ADD BYTE PTR DS:[EAX],AL
0049596C	0000	ADD BYTE PTR DS:[EAX],AL
0049596E	0000	ADD BYTE PTR DS:[EAX],AL
00495970	0000	ADD BYTE PTR DS:[EAX],AL
00495972	0000	ADD BYTE PTR DS:[EAX],AL
00495974	0000	ADD BYTE PTR DS:[EAX],AL
00495976	0000	ADD BYTE PTR DS:[EAX],AL
00495978	0000	ADD BYTE PTR DS:[EAX],AL

خوب،حالا به OEP رسیدیم،از کجا فهمیدیم؟...به این کدها با دقت نگاه کنید.به نظر شما شبیه کدهایی که در Entry Point برنامه های دلفی قرار دارد نیست؟...Call های فراوان...همچنین در درون اولین Call تابع GetModuleHandleA قرار دارد.) پس اینجا OEP ماست و برنامه ما در دلفی نوشته شده است.بعد از این دیگر مشکلی نیست.فقط دامپ بگیرید و با ImportREC دامپ خود را فیکس کنید.

ASDPack

برنامه را در OllyDBG باز کنید:

Address	Hex dump	Disassembly
004694AA <ModuleEntryPoint>	8B4424 04	MOV EAX,DWORD PTR SS:[ESP+4]
004694AE	56	PUSH ESI
004694AF	57	PUSH EDI
004694B0	53	PUSH EBX
004694B1	E8 CD010000	CALL 00469683
004694B6	C3	RET
004694B7	0000	ADD BYTE PTR DS:[EAX],AL
004694B9	0000	ADD BYTE PTR DS:[EAX],AL
004694BB	0000	ADD BYTE PTR DS:[EAX],AL
004694BD	0000	ADD BYTE PTR DS:[EAX],AL
004694BF	0000	ADD BYTE PTR DS:[EAX],AL
004694C1	0000	ADD BYTE PTR DS:[EAX],AL
004694C3	0010	ADD BYTE PTR DS:[EAX],DL
004694C5	0000	ADD BYTE PTR DS:[ESI+EAX],AL
004694C7	007C06 00	ADD BYTE PTR DS:[ESI+EAX],BH
004694C8	0000	ADD BYTE PTR DS:[EAX],AL

همانطور که می بینید دومین دستور ما Push ESI هست که مقدار ESI را وارد Stack می کند. پس مقدار رجیستر ESP تغییر می کند. لذا کافیت دو بار کلید F8 را بزنیم. و بعد روی مقدار رجیستر ESP یک Hardware breakpoint on access بگذاریم و برنامه را با F9 اجرا کنیم.

Address	Hex dump	Disassembly	Comment
51	8BCB	PUSH ECX	
0341 14		MOV ECX,EBX	
59		ADD EAX,DWORD PTR DS:[ECX+14]	
5B		POP ECX	
5F		POP EBX	
5E		POP EDI	
5E		POP ESI	
50		PUSH EAX	UnPackMe.004271B0
C3		RET	
0000		ADD BYTE PTR DS:[EAX],AL	
0000		ADD BYTE PTR DS:[EAX],AL	
0000		ADD BYTE PTR DS:[EAX],AL	
0000		ADD BYTE PTR DS:[EAX],AL	
0000		ADD BYTE PTR DS:[EAX],AL	
0000		ADD BYTE PTR DS:[EAX],AL	
0000		ADD BYTE PTR DS:[EAX],AL	
0000		ADD BYTE PTR DS:[EAX],AL	
0000		ADD BYTE PTR DS:[EAX],AL	
0000		ADD BYTE PTR DS:[EAX],AL	
0000		ADD BYTE PTR DS:[EAX],AL	
0000		ADD BYTE PTR DS:[EAX],AL	
0000		ADD BYTE PTR DS:[EAX],AL	

دستور Push که می دانید چه کار می کند...مقداری را وارد حافظه Stack می کند. و Ret هم آخر یک تابع را نشان می دهد و ما را به جایی منتقل می کند که در حافظه Stack قرار دارد. با توجه به این اطلاعات:

اگر قبل از دستور RET یک Push بیاید. دستور RET ما را به آدرسی می برد که Push شده است.

به زبانی ساده :

Push + Ret = Jump

یعنی با این دو دستور که در تصویر بالا می بینید ما به خطی که در EAX نوشته شده است، می رویم. یعنی خط : 004271B0 کافیت دو بار کلید F8 را بزنید. تا به OEP برسیم:

Address	Hex dump	Disassembly	Comment
80CC 10		OR AH,10	
C3		RET	
90		NOP	
90		NOP	
90		NOP	
90		NOP	
90		NOP	
55		PUSH EBP	
8BEC		MOV EBP,ESP	
6A FF		PUSH -1	
68 600E4500		PUSH 00450E60	
68 C8924200		PUSH 004292C8	
64:A1 00000000		MOV EAX,DWORD PTR FS:[0]	
50		PUSH EAX	
64:8925 00000000		MOV DWORD PTR FS:[0],ESP	
83C4 A8		ADD ESP,-58	
53		PUSH EBX	
56		PUSH ESI	
57		PUSH EDI	
8965 E8		MOV DWORD PTR SS:[EBP-18],ESP	
FF15 DC0A4600		CALL DWORD PTR DS:[460ADC]	
33D2		XOR EDX,EDX	
8AD4		MOV DL,AH	
8915 34E64500		MOV DWORD PTR DS:[45E634],EDX	
8BC8		MOV ECX,EAX	
81E1 FF000000		AND ECX,0FF	
890D 30E64500		MOV DWORD PTR DS:[45E630],ECX	
C1E1 08		SHL ECX,8	
03CA		ADD ECX,EDX	
890D 2CE64500		MOV DWORD PTR DS:[45E62C],ECX	
C1E8 10		SHR EAX,10	
A3 28E64500		MOV DWORD PTR DS:[45E628],EAX	
E8 94210000		CALL 004293A0	
85C0		TEST EAX,EAX	
75 0A		JNZ SHORT 0042721A	
6A 1C		PUSH 1C	
E8 49010000		CALL 00427360	
83C4 04		ADD ESP,4	

خوب...دیگر از این به بعد مشکلی نیست. کافی است فایل را دامپ کنید(توصیه من این است با LordPE این کار را بکنید.) و با ImportREC آن را فیکس کنید.

EZIP

برنامه را در OllyDBG باز کنید:



یک بار کلید F8 را بزنید تا به دستور Push EBP برسید. یک بار دیگر کلید F8 را بزنید تا رجیستر ESP تغییر کند و بعد روی مقدار رجیستر ESP یک Hardware breakpoint on access بگذارید و بعد کلید F9 را بزنید تا به این خط برسید:

5E	POP ESI	
5B	POP EBX	
8BE5	MOV ESP,EBP	
5D	POP EBP	
FFE0	JMP EAX	UnPackMe.004012C0
5F	POP EDI	
5E	POP ESI	
5B	POP EBX	
C9	LEAVE	
C3	RET	
CC	INT3	
CC	INT3	
CC	INT3	
CC	INT3	
CC	INT3	
CC	INT3	
CC	INT3	
CC	INT3	

این پرش ما را درست به OEP می برد ☺
بعد از آن هم مشکلی وجود نخواهد داشت. فقط فایل را دامپی می کنید و با ImportREC فیکس می کنید.

WinUPack

برنامه را در OllyDBG باز کنید:

	Hex dump	Disassembly
eEntryPoint>	BE B0114000	MOV ESI,004011B0
	AD	LODS DWORD PTR DS:[ESI]
	50	PUSH EAX
	FF76 34	PUSH DWORD PTR DS:[ESI+34]
	EB 7C	JMP SHORT 00401000
	48	DEC EAX
	010F	ADD DWORD PTR DS:[EDI],ECX
	010B	ADD DWORD PTR DS:[EBX],ECX
	014C6F 61	ADD DWORD PTR DS:[EDI+EBP*2+61],ECX
	64:4C	DEC ESP
	6962 72 61727941	IMUL ESP,DWORD PTR DS:[EDX+72],41797261
	0000	ADD BYTE PTR DS:[EAX],AL
	1810	SBB BYTE PTR DS:[EAX],DL
	0000	ADD BYTE PTR DS:[EAX],AL
	1000	ADC BYTE PTR DS:[EAX],AL
	0000	ADD BYTE PTR DS:[EAX],AL

سه بار کلید F8 را بزنید تا دستور Push EAX اجرا شود و مقدار رجیستر ESP تغییر کند. حالا همانند مثال های قبل یک Hardware Breakpoint روی مقدار رجیستر ESP بگذارید و کلید F9 را بزنید:

0042719C	80CC 03	OR AH,3
0042719F	F7C2 00000400	TEST EDX,40000
004271A5	74 03	JE SHORT 004271AA
004271A7	80CC 10	OR AH,10
004271AA	C3	RET
004271AB	90	NOP
004271AC	90	NOP
004271AD	90	NOP
004271AE	90	NOP
004271AF	90	NOP
004271B0	55	PUSH ESP
004271B1	8BEC	MOV EBP,ESP
004271B3	6A FF	PUSH -1
004271B5	68 600E4500	PUSH 00450E60
004271BA	68 C8924200	PUSH 004292C8
004271BF	64:A1 00000000	MOV EAX,DWORD PTR FS:[0]
004271C5	50	PUSH EAX
004271C6	64:8925 00000000	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 SS:[EBP-18],ESP
004271D6	FF15 DC0A4600	CALL DWORD PTR DS:[460ADC]
004271DC	33D2	XOR EDX,EDX
004271DE	8AD4	MOV DL,AH
004271E0	8915 34E64500	MOV DWORD PTR DS:[45E634],EDX
004271E6	8BC8	MOV ECX,EAX
004271E8	81E1 FF000000	AND ECX,0FF
004271EE	890D 30E64500	MOV DWORD PTR DS:[45E630],ECX
004271F4	C1E1 08	SHL ECX,8
004271F7	03CA	ADD ECX,EDX
004271F9	890D 2CE64500	MOV DWORD PTR DS:[45E62C],ECX
004271FF	C1E8 10	SHR EAX,10
00427202	A3 28E64500	MOV DWORD PTR DS:[45E628],EAX
00427207	E8 94210000	CALL 004293A0
0042720C	85C0	TEST EAX,EAX
0042720E	75 0A	JNZ SHORT 0042721A
00427210	6A 1C	PUSH 1C
00427212	E8 49010000	CALL 00427360
00427217	83C4 04	ADD ESP,4
0042721A	E8 D12F0000	CALL 0042A1F0
0042721F	85C0	TEST EAX,EAX
00427221	75 0A	JNZ SHORT 0042722D
00427223	6A 10	PUSH 10
00427225	E8 26010000	CALL 00427360

همانطور که می بینید این بار مستقیم به OEP رسیدیم. ©
بعد از این هم مشکلی نیست. فایل را دامپ می کنید و بعد فیکس می کنید.

Mario Packer

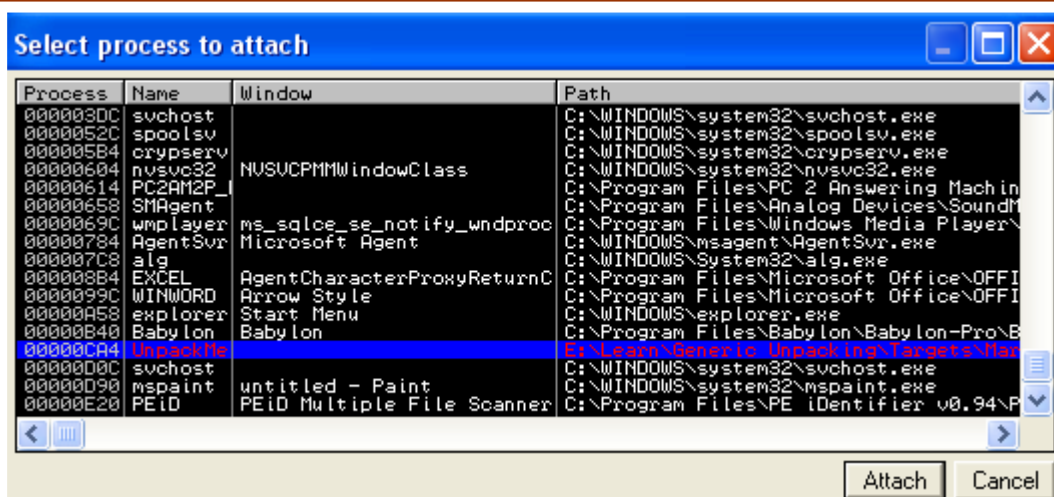
برنامه را در OllyDBG باز کنید، کلید F9 را بزنید تا برنامه به طور کامل اجرا شود:



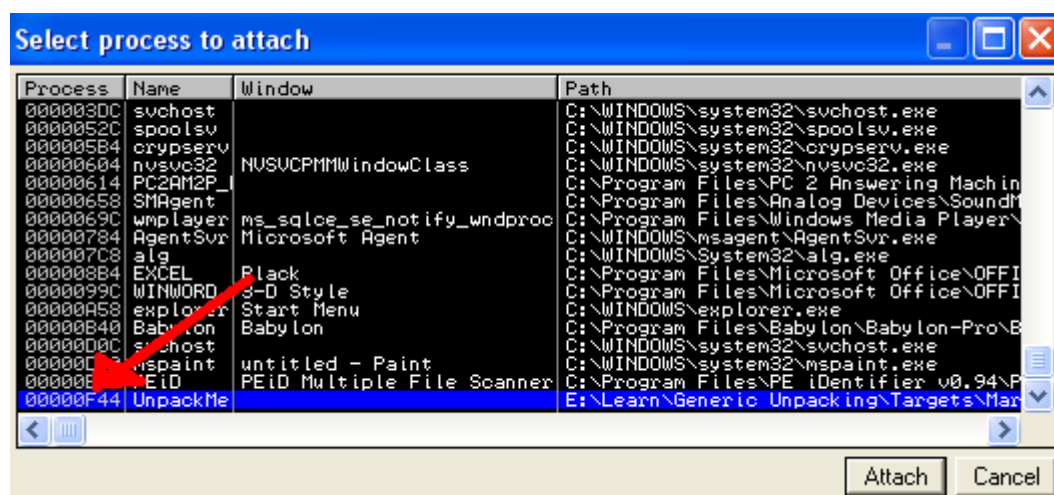
حالا در همین حالت که برنامه اجرا شده است، OllyDBG را کامل ببندید:



می بینید؟...خیلی عجیبه!...با اینکه OllyDBG به طور کامل بسته شده ولی برنامه هنوز اجرا می باشد؟...چه طور ممکنه؟...همیشه اگر OllyDBG را ببندید، برنامه ای که در OllyDBG باز کرده بودید هم بسته می شد...اما این بار چنین اتفاقی نیفتاد؟...چرا؟...خوب بهتر است برنامه را دوباره در OllyDBG باز کنیم تا ببینیم چرا این طور شده؟
بعد از باز کردن برنامه در منوی File گزینه Attach را بزنید:



همانطور که در تصویر بالا می بینید، PID (Process ID) پروسه ما برابر CA4 می باشد. (در کامپیوتر من اینگونه است، در کامپیوتر شما PID فرق خواهد داشت) خوب، این صفحه را ببندید و دوباره برنامه را با F9 اجرا کنید. حالا همینطور که برنامه اجرا می باشد، دوباره در منوی File گزینه Attach را بزنید:



می بینید پروسه فایل ما حالا هست : F44 !
این یعنی چی؟...این یعنی برنامه ما بعد از این که اجرا می شود، فایل اصلی ما را با یک پروسه جدید به اجرا در می آورد...یعنی اینکه Mario Packer یک پکر نیست...بلکه یک Self-Extractor هست...اینجاست که باید فرق بین یک پکر با یک Self-Extractor را تشخیص بدین:

یک پکر کدهای خودش را به فایل اصلی ما تزریق می کند. در حالی که یک Self-Extractor فایل اصلی ما را به کدهای خودش تزریق می کند

حالا چه طور فایل اصلی را از داخل Mario Packer بیوریم؟...خیلی ساده است...کافیست برنامه را اجرا کنیم و بعد از فایلمان Dump بگیریم. (بهتره با PeTools این کار را بکنید)
هیچ کار دیگه ای لازم نیست...چون Mario packer در واقع اصلا پکر نیست ©

FSG

گاهی اوقات برای یافتن OEP بهتر است روش های خاصی را بلد بود. برای مثال در مورد پکر FSG چند خط زیر Entry point یک دستور مثل `JMP DWORD PTR DS:[EBX+C]` وجود دارد که درست می رود OEP... یادگیری این روش ها در مورد پکرهای معروف خیلی به ما کمک می کند ☺
خوب، برنامه را در OllyDBG باز کنید:

Address	Hex dump	Disassembly	Comment
00400154	<ModuleEntryPoint>		
0040015A	8725 003E4100	XCHG DWORD PTR DS:[413E00],ESP	
0040015B	61	POPAD	
0040015C	94	XCHG EAX,ESP	
0040015D	55	PUSH EBP	
0040015E	A4	MOVS BYTE PTR ES:[EDI],BYTE PTR DS:[ESI]	
00400160	B6 80	MOV DH,80	
00400162	FF13	CALL DWORD PTR DS:[EBX]	
00400164	73 F9	JNB SHORT 0040015D	
00400166	33C9	XOR ECX,ECX	
00400168	FF13	CALL DWORD PTR DS:[EBX]	
0040016A	73 16	JNB SHORT 00400160	
0040016C	33C9	XOR EAX,EAX	
0040016E	FF13	CALL DWORD PTR DS:[EBX]	
00400170	73 1F	JNB SHORT 0040016F	
00400172	B6 80	MOV DH,80	
00400173	41	INC ECX	
00400175	B0 10	MOV AL,10	
00400177	FF13	CALL DWORD PTR DS:[EBX]	
00400179	12C0	ADC AL,AL	
0040017B	73 FA	JNB SHORT 00400175	
0040017D	75 3A	JNZ SHORT 004001B7	
0040017E	AA	STOS BYTE PTR ES:[EDI]	
00400180	EB E0	JMP SHORT 00400183	
00400182	FF53 08	CALL DWORD PTR DS:[EBX+8]	
00400184	02F6	ADD DH,DH	
00400186	83D9 01	SBB ECX,1	
00400188	75 0E	JNZ SHORT 00400199	
0040018A	FF53 04	CALL DWORD PTR DS:[EBX+4]	
0040018C	EB 24	JMP SHORT 00400183	
0040018E	AC	LODS BYTE PTR DS:[ESI]	
00400190	D1E8	SHR EAX,1	
00400192	74 2D	JE SHORT 004001C1	
00400194	13C9	ADC ECX,ECX	
00400196	EB 18	JMP SHORT 00400180	
00400198	91	XCHG EAX,ECX	
0040019A	48	DEC EAX	
0040019C	C1E0 08	SHL EAX,8	
0040019E	AC	LODS BYTE PTR DS:[ESI]	
004001A0	FF53 04	CALL DWORD PTR DS:[EBX+4]	
004001A2	3B43 F8	CMP EAX,DWORD PTR DS:[EBX-8]	
004001A4	73 0A	JNB SHORT 004001B0	
004001A6	80FC 05	CMP AH,5	
004001A8	73 06	JNB SHORT 004001B1	
004001AA	83F8 7F	CMP EAX,7F	
004001AC	77 02	JA SHORT 004001B2	
004001AE	41	INC ECX	
004001B0	41	INC ECX	
004001B2	95	XCHG EAX,EBP	
004001B4	8BC5	MOV EAX,EBP	
004001B6	B6 00	MOV DH,0	
004001B8	56	PUSH ESI	
004001BA	8BF7	MOV ESI,EDI	
004001BC	2BF0	SUB ESI,EAX	
004001BE	F3A4	REP MOVS BYTE PTR ES:[EDI],BYTE PTR DS:[ESI]	
004001BF	5E	POP ESI	
004001C0	EB 9F	JMP SHORT 00400180	

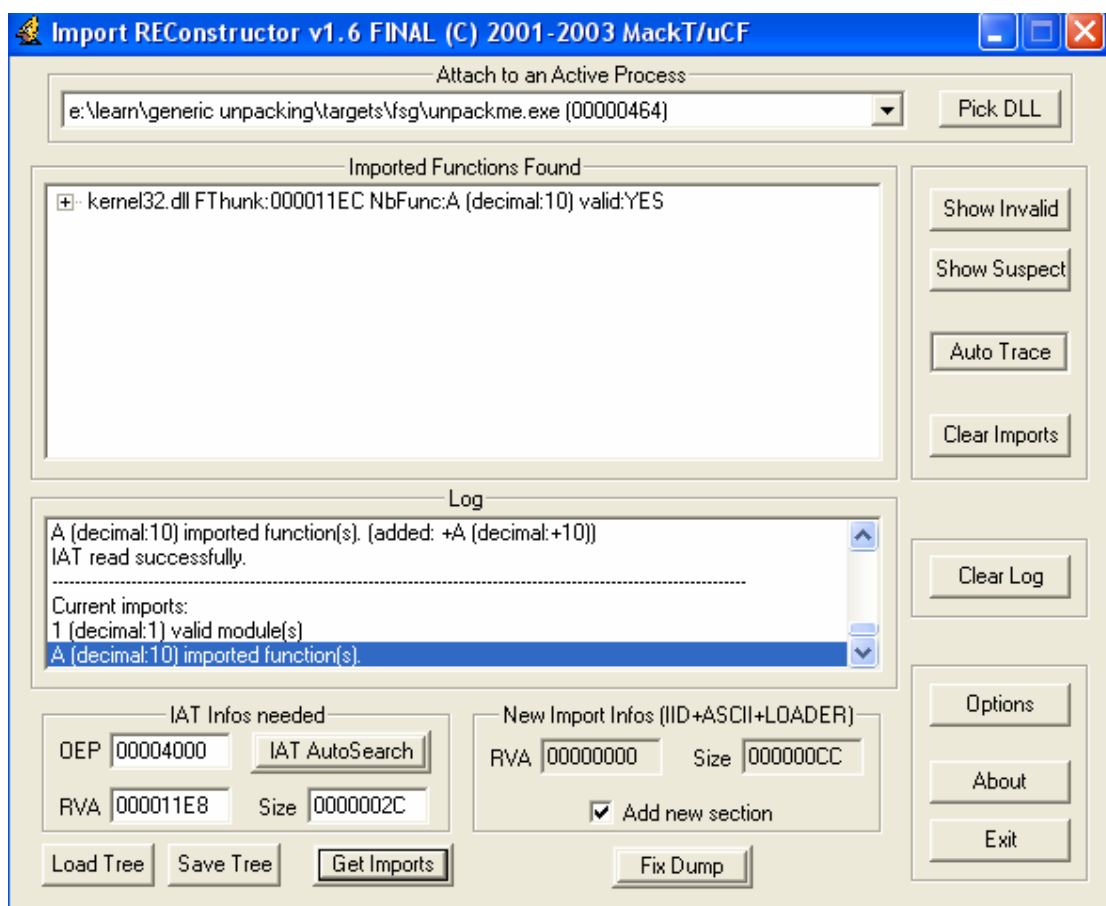
خوب، حالا با موس به سمت پایین بروید تا یک دستور شبیه `JMP DWORD PTR DS:[EBX+C]` پیدا کنید:

5E	POP ESI	
EB 9F	JMP SHORT 00400180	
5E	POP ESI	
AD	LODS DWORD PTR DS:[ESI]	
97	XCHG EAX,EDI	
AD	LODS DWORD PTR DS:[ESI]	
50	PUSH EAX	
FF53 10	CALL DWORD PTR DS:[EBX+10]	
95	XCHG EAX,EBP	
8B07	MOV EAX,DWORD PTR DS:[EDI]	
40	INC EAX	
78 F3	JS SHORT 004001C2	
75 03	JNZ SHORT 004001D4	
FF53 0C	JMP DWORD PTR DS:[EBX+C]	
50	PUSH EAX	
55	PUSH EBP	
FF53 14	CALL DWORD PTR DS:[EBX+14]	
AB	STOS DWORD PTR ES:[EDI]	
EB EE	JMP SHORT 004001CA	
33C9	XOR ECX,ECX	
41	INC ECX	
FF13	CALL DWORD PTR DS:[EBX]	
13C9	ADC ECX,ECX	
FF13	CALL DWORD PTR DS:[EBX]	
72 F8	JB SHORT 004001D0	
C3	RET	
02D2	ADD DL,DL	
75 05	JNZ SHORT 004001F1	
8A16	MOV DL,BYTE PTR DS:[ESI]	
46	INC ESI	
12D2	ADC DL,DL	
C3	RET	
4B	DEC EBX	
45	INC EBP	
52	PUSH EAX	

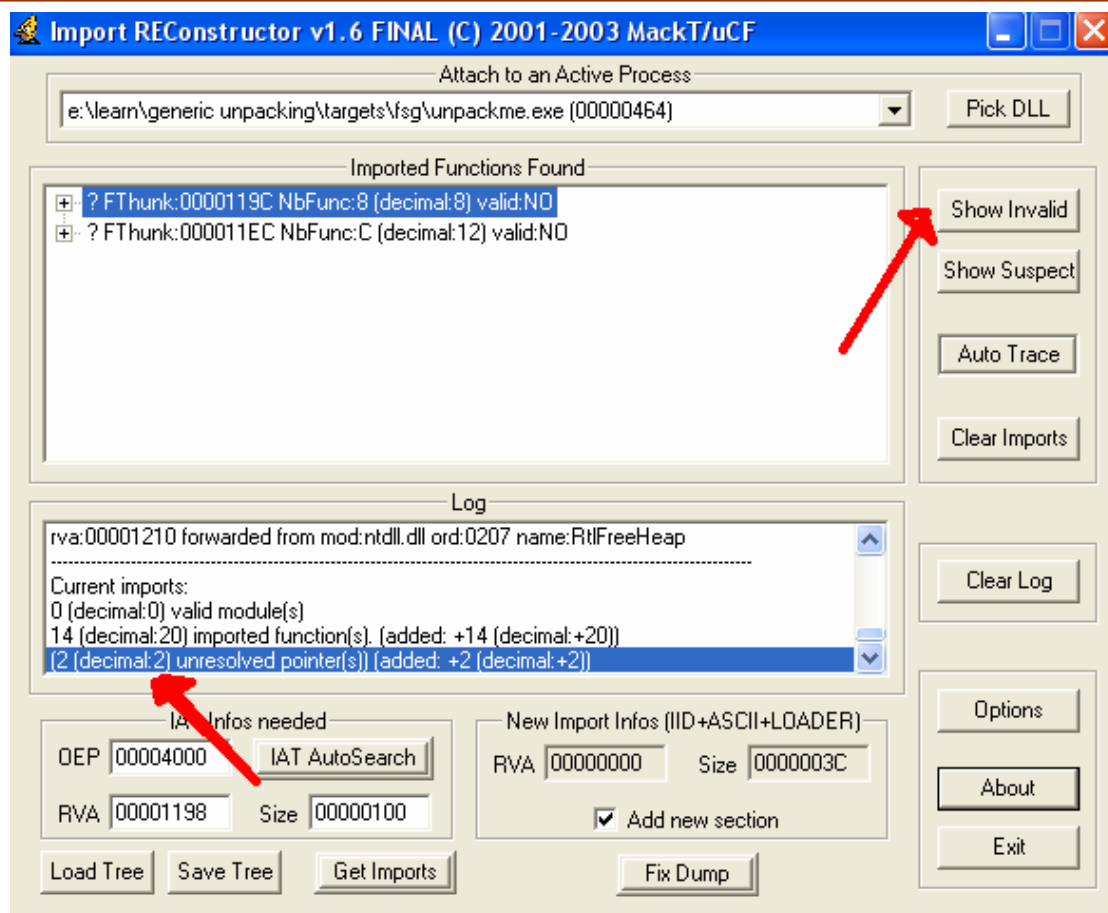
این دستور درست به OEP می رود. کافایت بر روی آن Breakpoint گذاشته و کلید F9 را بزنید تا به این خط برسید و بعد یک F8 به OEP نظر می رود:

00	DB 00		
00	DB 00		
9B	WAIT		
DBE3	FINIT		
9B	WAIT		
DBE2	FCLEX		
D92D 00604000	FLDCW WORD PTR DS:[406000]		
55	PUSH EBP		
89E5	MOV EBP,ESP		
E8 91030000	CALL 004043A5		
68 00000000	PUSH 0		
FF15 F4114000	CALL DWORD PTR DS:[4011F4]		[pModule = NULL GetModuleHandleA
A3 07F04000	MOV DWORD PTR DS:[40F007],EAX		
60	PUSHAD		
8925 0BF04000	MOV DWORD PTR DS:[40F00B],ESP		
E9 30000000	JMP 00404060		
8B25 0BF04000	MOV ESP,DWORD PTR DS:[40F00B]		
61	POPAD		
E8 A9080000	CALL 004048E5		
E8 FD030000	CALL 0040443E		
89EC	MOV ESP,EBP		
5D	POP EBP		
FF35 D4F14000	PUSH DWORD PTR DS:[40F1D4]		
FF15 EC114000	CALL DWORD PTR DS:[4011EC]		[ExitCode = 0 ExitProcess
9B	WAIT		
DBE2	FCLEX		
D92D 00604000	FLDCW WORD PTR DS:[406000]		
C3	RET		
00	DB 00		
00	DB 00		
00	DB 00		

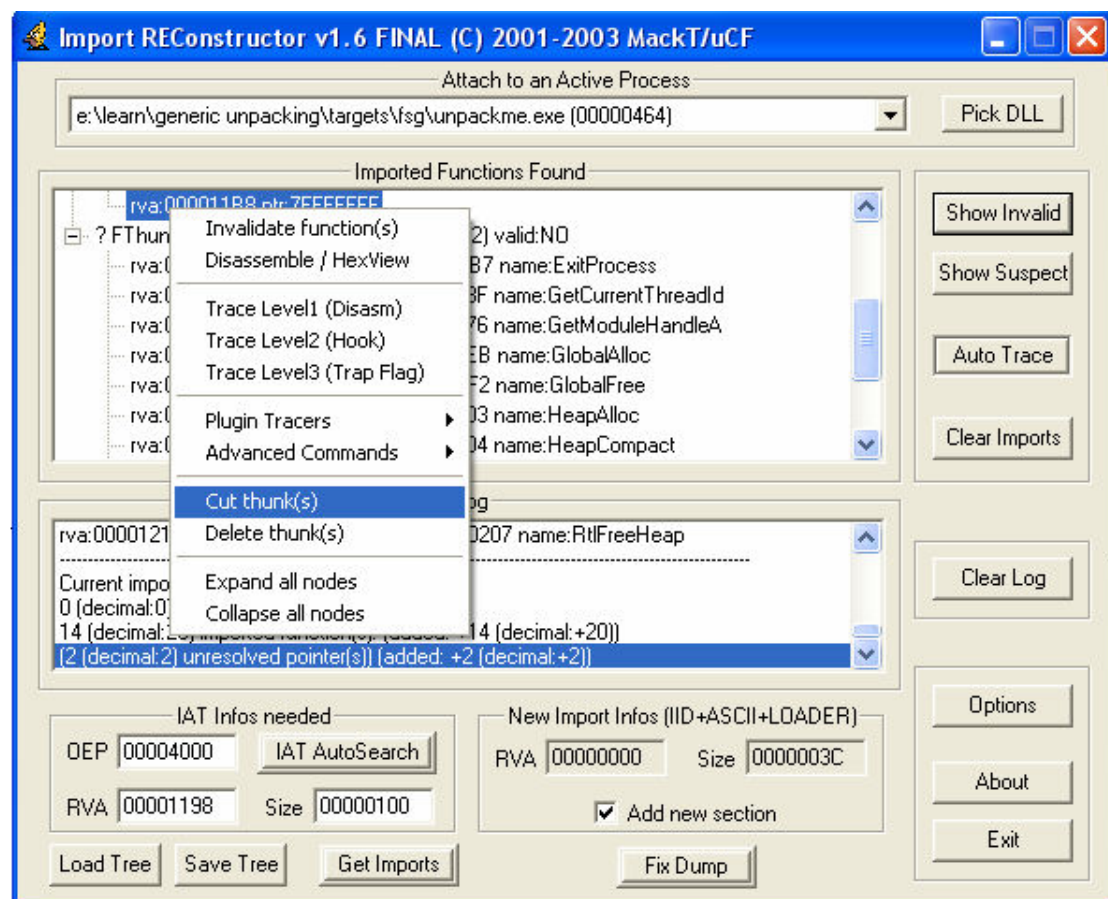
حالا از فایل دامپ بگیرید و بعد ImportREC را باز کنید، OEP را به برنامه بدهید(4000)...گزینه IAT Auto-search را بزنید. و بعد گزینه Get Imports را بزنید:



باز هم OlllyDBG نتوانست IAT را درست تشخیص بدهد، پس باید خودمان محل شروع RVA را پیدا کنیم. در یکی از مثال های قبل این کار را کردم...پس نیازی نیست که دوباره اینجا توضیح بدهم. من محل شروع IAT را بر حسب RVA برابر ۱۱۹۸ بدست آوردم...پس این مقدار را وارد ImportREC می کنم...همچنین در قسمت Size هم می نویسم ۱۰۰، چون برنامه ما تعداد توابعش بسیار کم است و اندازه IAT آن بیشتر از صد نیست. البته بهتر است که خودمان اندازه IAT را به دست بیاوریم...ولی چون من آدم تنبلی هستم و حوصله حساب و کتاب رو ندارم، همان صد را می نویسم و بعد گزینه Get Imports را می زنم:



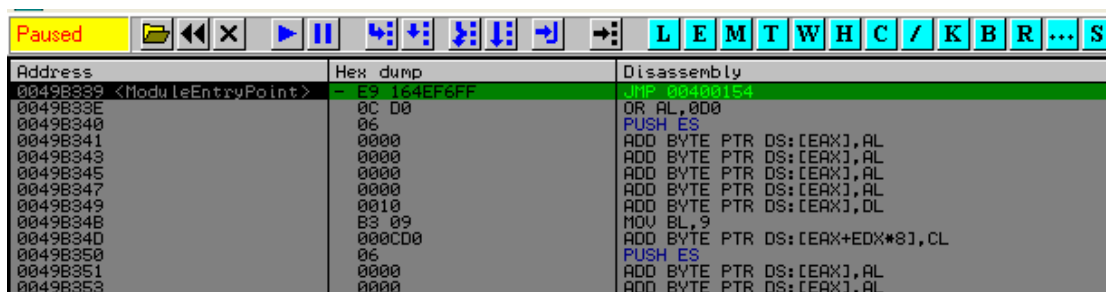
همانطور که می بینید دو تابع Invalid داریم، برای حذف این دو گزینه هم گزینه Show Invalid را بزنید و بعد همانند تصویر کلیک راست کرده و گزینه Cut Thunk را بزنید تا تمامی توابع Valid شود:



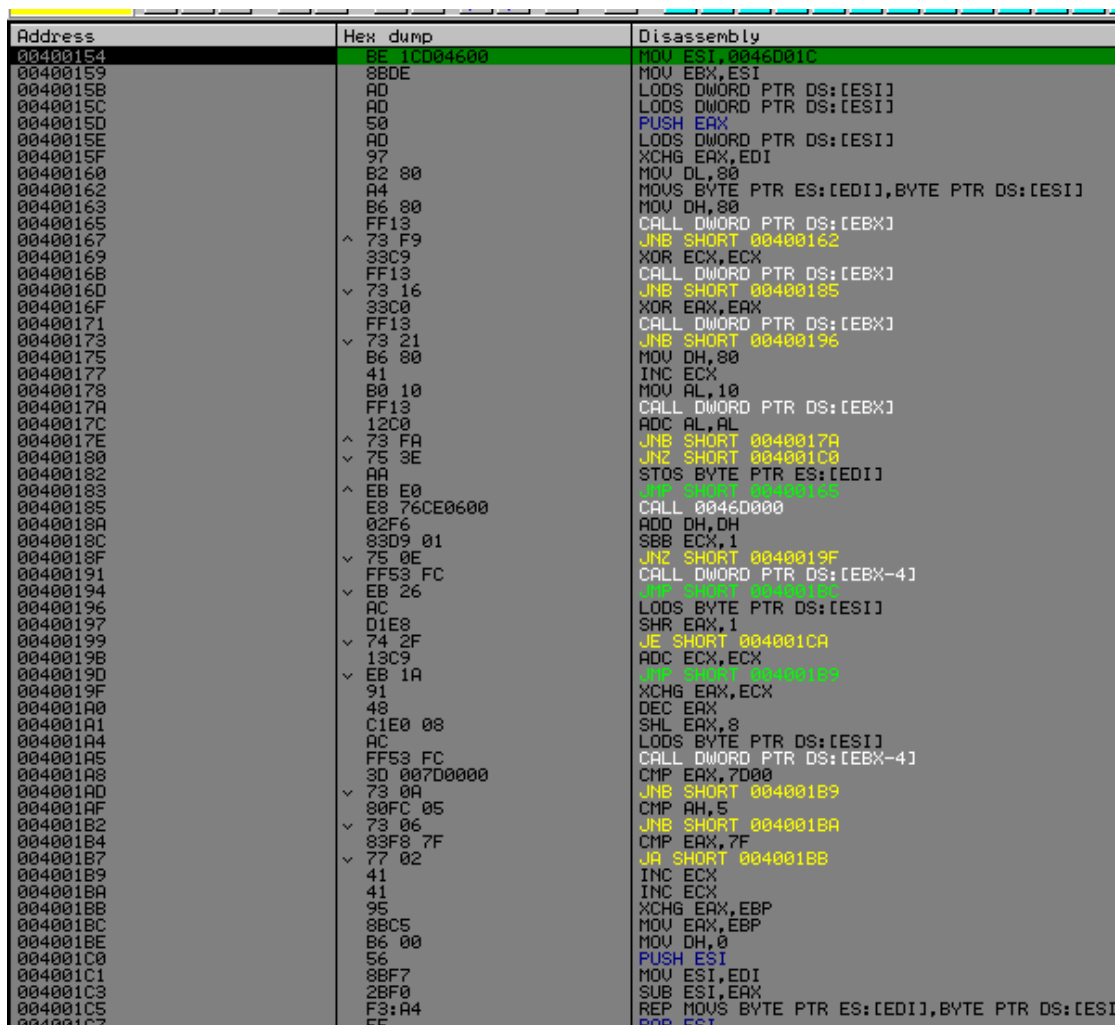
حالا هم فایل دامپ خودتان را فیکس کنید، می بینید که فایل آنپک شده اجرا می شود.

MEW

اصولا در یافتن OEP بهتر است همیشه در آخر کدها، یک Breakpoint (ترجیحا Hardware Breakpoint) بگذاریم تا بتوانیم OEP را پیدا کنیم. چرا که در معمولا در آخر کدهای یک پکر، اختیار از دست پکر خارج می شود و به دست فایل اصلی ما می افتد... در مورد MEW هم ما همین کار را می کنیم، برنامه را در OllyDBG باز کنید:



یکبار F8 بزنی:



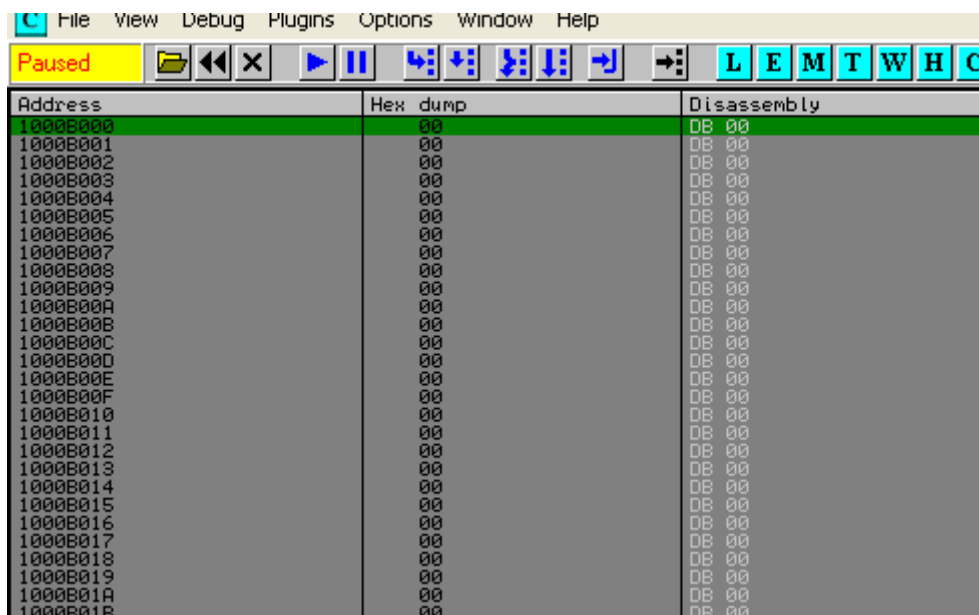
این کدهای اصلی پکر ماست. بهتر است انتهای کد را پیدا کنیم. برای این کار با Mouse به پایین بروید (Scroll down) تا به اینجا برسید:

004001C8	EB 9B	JMP SHORT 004001E5
004001CA	AD	LODS DWORD PTR DS:[ESI]
004001CB	85C0	TEST EAX,EAX
004001CD	75 90	JNZ SHORT 004001F5
004001CF	E8 E1B30900	CALL 0049B5B5
004001D4	AD	LODS DWORD PTR DS:[ESI]
004001D5	96	XCHG EAX,ESI
004001D6	AD	LODS DWORD PTR DS:[ESI]
004001D7	97	XCHG EAX,EDI
004001D8	56	PUSH ESI
004001D9	AC	LODS BYTE PTR DS:[ESI]
004001DA	3C 00	CMP AL,0
004001DC	75 FB	JNZ SHORT 004001D9
004001DE	FF53 F0	CALL DWORD PTR DS:[EBX-10]
004001E1	95	XCHG EAX,EBP
004001E2	56	PUSH ESI
004001E3	AD	LODS DWORD PTR DS:[ESI]
004001E4	0FC8	BSWAP EAX
004001E6	40	INC EAX
004001E7	59	POP ECX
004001E8	74 EC	JE SHORT 004001D6
004001EA	79 07	JNS SHORT 004001F3
004001EC	AC	LODS BYTE PTR DS:[ESI]
004001ED	3C 00	CMP AL,0
004001EF	75 FB	JNZ SHORT 004001EC
004001F1	91	XCHG EAX,ECX
004001F2	40	INC EAX
004001F3	50	PUSH EAX
004001F4	55	PUSH EBP
004001F5	FF53 F4	CALL DWORD PTR DS:[EBX-C]
004001F8	AB	STOS DWORD PTR ES:[EDI]
004001F9	85C0	TEST EAX,EAX
004001FB	75 E5	JNZ SHORT 004001E2
004001FD	C3	RET
004001FE	0000	ADD BYTE PTR DS:[EAX],AL
00400200	0000	ADD BYTE PTR DS:[EAX],AL
00400202	0000	ADD BYTE PTR DS:[EAX],AL
00400204	0000	ADD BYTE PTR DS:[EAX],AL
00400206	0000	ADD BYTE PTR DS:[EAX],AL
00400208	0000	ADD BYTE PTR DS:[EAX],AL
0040020A	0000	ADD BYTE PTR DS:[EAX],AL
0040020C	0000	ADD BYTE PTR DS:[EAX],AL
0040020E	0000	ADD BYTE PTR DS:[EAX],AL
00400210	0000	ADD BYTE PTR DS:[EAX],AL
00400212	0000	ADD BYTE PTR DS:[EAX],AL
00400214	0000	ADD BYTE PTR DS:[EAX],AL
00400216	0000	ADD BYTE PTR DS:[EAX],AL
00400218	0000	ADD BYTE PTR DS:[EAX],AL

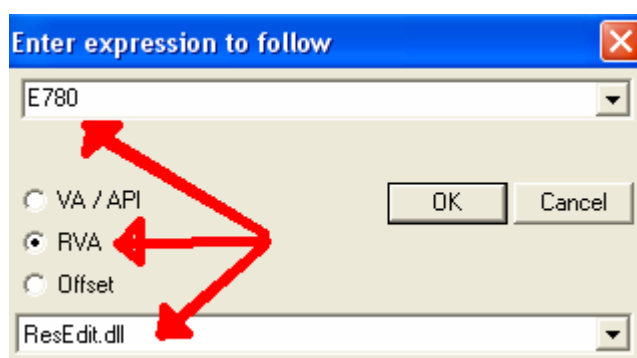
همانطور که می بینید بعد از این دستور Return دیگر دستور خاصی وجود ندارد، پس این RET ممکن است ما را به OEP ببرد. پس بر روی آن یک Breakpoint بگذارید و برنامه را با F9 اجرا کنید، برنامه در جایی که Breakpoint گذاشتیم متوقف شد. حالا یک بار کلید F8 را بزنید تا به OEP برسیم. بعد از یافتن OEP دیگر مشکل خاصی وجود ندارد. کافیه از فایل‌مان Dump بگیریم و آن را با ImportREC فیکس کنیم.

UPX DLL

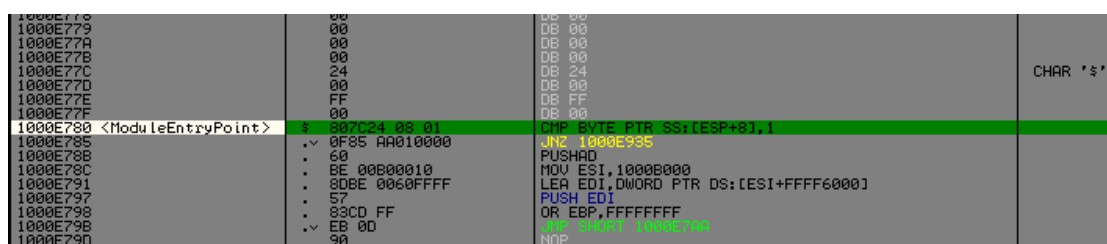
اصولا آپیک کردن یک فایل DLL زیاد فرقی با آپیک کردن یک Exe ندارد. اول از همه بهتر است ببینیم که ImageBase فایل ما چند است؟ برای اینکار هم از PEiD یا نرم افزارهای مشابه استفاده می کنیم. (قبلا توضیح دادم). در این مثال (فایل ResEdit.dll) ImageBase ما برابر 10000000 است. OllyDBG برای اجرای یک فایل DLL از فایل LoadDll.exe استفاده می کند. به عقیده من اگر ImageBase فایل شما غیر از 00400000 باشد. می توانید برای آپیک کردن فایل DLL از همین فایل استفاده می کنیم. (loadDll.exe) ولی اگر ImageBase فایل DLL ما برابر 00400000 بود، بهتر است که ابتدا مشخصات فایل DLL را شبیه یک فایل Exe دریاوریم و بعد اقدام به آپیک کردن آن بکنیم. (در مثال بعد در این باره توضیح خواهم داد). خوب، فایل مورد نظر را در OllyDBG باز کنید:



ابتدا به Entry Point فایل خود بروید. از آنجا که می دانیم Entry Point فایل ما بر حسب RVA برابر E780 است. (این اطلاعات را از PEiD گرفتم)
کلید CTRL + G را بزنید و همانند تصویر تیک گزینه RVA را بزنید.، فایل مورد نظر خود را انتخاب کنید (Resedit.dll) و بعد Entry point فایل را انتخاب کنید:



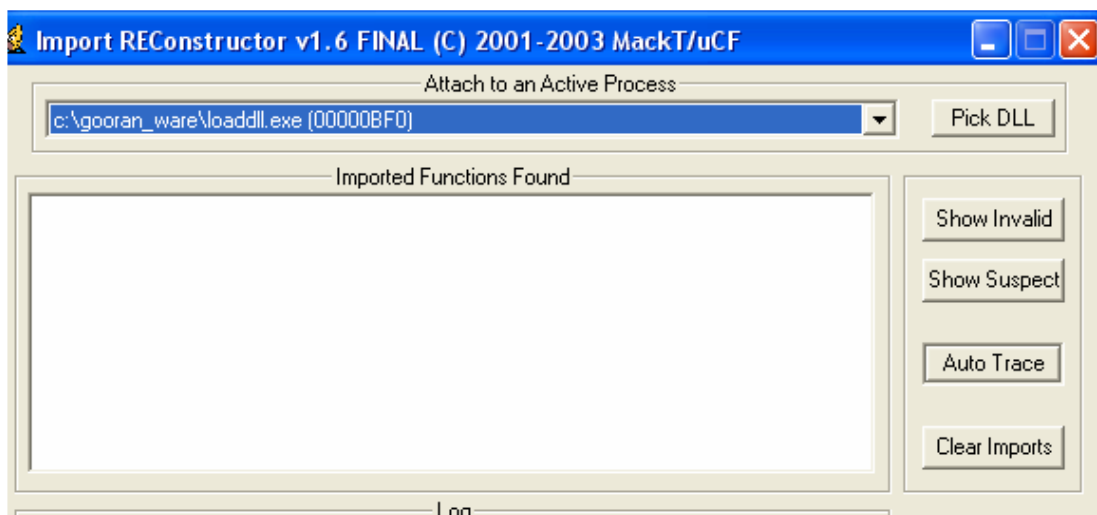
حالا بر روی Entry Point فایل Breakpoint بگذارید و کلید F9 را بزنید:



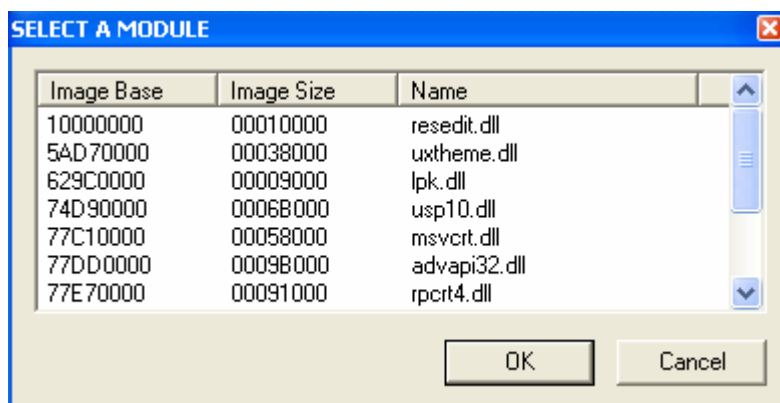
سه بار کلید F8 را بزنید تا به OEP برسید:

10001E18	E8 AF170000	CALL 1000355A
10001E1B	59	POP ECX
10001E1C	6A 01	PUSH 1
10001E1E	58	POP EAX
10001E1F	C2 0C00	RET 0C
10001E22	55	PUSH EBP
10001E23	8BEC	MOV EBP,ESP
10001E25	53	PUSH EBX
10001E26	8B5D 08	MOV EBX,DWORD PTR SS:[EBP+8]
10001E29	56	PUSH ESI
10001E2A	8B75 0C	MOV ESI,DWORD PTR SS:[EBP+C]
10001E2D	57	PUSH EDI
10001E2E	8B7D 10	MOV EDI,DWORD PTR SS:[EBP+10]
10001E31	85F6	TEST ESI,ESI
10001E33	75 09	JNZ SHORT 10001E3E
10001E35	833D 5C9A0010 00	CMP DWORD PTR DS:[10009A5C],0
10001E3C	EB 26	JMP SHORT 10001E64
10001E3E	83FE 01	CMP ESI,1
10001E41	74 05	JE SHORT 10001E48
10001E43	83FE 02	CMP ESI,2
10001E46	75 22	JNZ SHORT 10001E6A
10001E48	A1 40A10010	MOV EAX,DWORD PTR DS:[1000A140]
10001E4D	85C0	TEST EAX,EAX
10001E4F	74 09	JE SHORT 10001E5A
10001E51	57	PUSH EDI
10001E52	56	PUSH ESI
10001E53	53	PUSH EBX
10001E54	FFD0	CALL EAX
10001E56	85C0	TEST EAX,EAX
10001E58	74 0C	JE SHORT 10001E66
10001E5A	57	PUSH EDI
10001E5B	56	PUSH ESI

از کجا فهمیدم اینجا OEP است؟...خوب UPX فایل DLL را Encrypt نمی کند. لذا در مورد فایل های DLL پک شده با UPX کدهای پکر مستقیم به فایل اصلی می رود، یعنی با سه پرش به OEP می رسیم. (حداقل در اینجا که اینطور بوده)
حالا از فایل با OllyDump دامپ بگیرید و ImportREC را باز کنید، فایل Loadll.exe را در ImportREC را باز کنید:



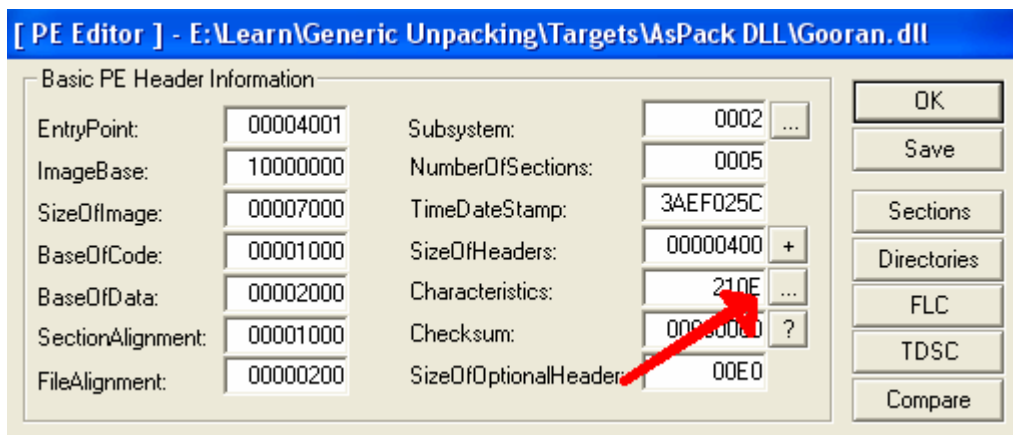
حالا گزینه Pick DLL را بزنید:



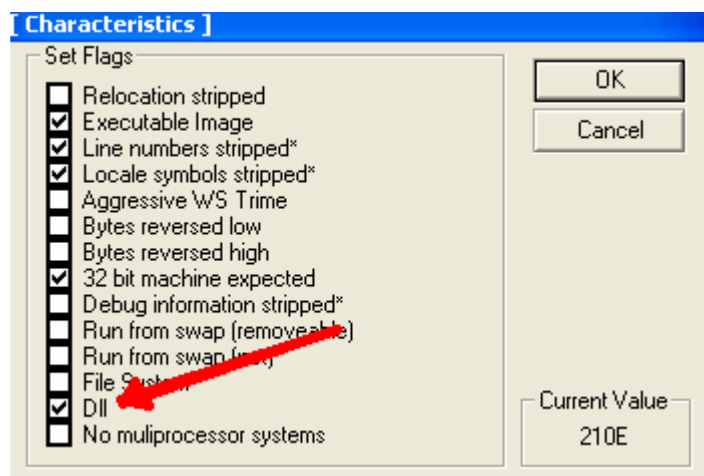
حالا هم فایل resedit.dll را انتخاب کرده و کلید OK را بزنید. از حالا دیگر مشکلی نیست. OEP را بر حسب RVA می دهید (1E22) و بعد فایل دامپ شده خودتان را فیکس کنید.

AsPack DLL

اول از همه در آنپک کردن یک فایل DLL خصوصا در مواردی که ImageBase آنها برابر 00400000 است، بهتر است مشخصات فایل را همانند یک فایل Exe کنیم. به این ترتیب OllyDBG خیال می کند که فایل Exe است و به طور مستقیم، بدون استفاده از LoadDll.exe فایل را اجرا می کند. فایل مورد نظر (Gooran.dll) را در LordPE باز کنید:



دکمه نشان داده شده در تصویر بالا را بزنید تا مشخصات فایل نشان داده شود:



همانطور که می بینید گزینه DLL تیک دارد، ما با برداشتن تیک این گزینه کاری می کنیم که OllyDBG خیال کند این فایل DLL نیست ☺

تیک این گزینه را بردارید و گزینه OK را بزنید و بعد گزینه Save و بعد OK را بزنید. حال فایل همانند یک فایل Exe در OllyDBG اجرا می شود. (بهتر است نام فایل را عوض کنید. بگذارید: Gooran.exe) فایل را در OllyDBG باز کنید:



اولین دستور PushAD است. پس به راحتی می شود از طریق رجیستر ESP، OEP را به دست آورد. OEP ما اینجا است:

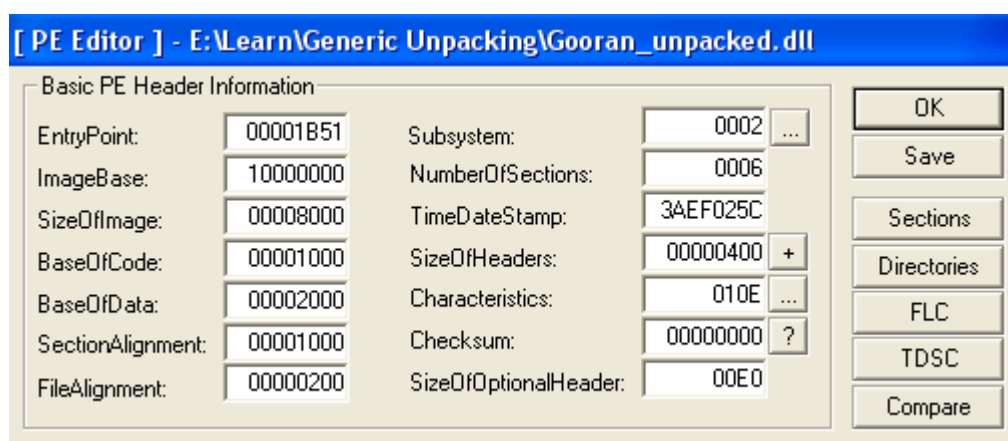
10001B4A	> 5E	POP ESI
10001B4B	> 6A 01	PUSH 1
10001B4D	> 58	POP EAX
10001B4E	> C2 0C00	RET 0C
10001B51	> 55	PUSH EBP
10001B52	> 8BEC	MOV EBP,ESP
10001B54	> 53	PUSH EBX
10001B55	> 8B5D 08	MOV EBX,[ARG.1]
10001B58	> 56	PUSH ESI
10001B59	> 8B75 0C	MOV ESI,[ARG.2]
10001B5C	> 57	PUSH EDI
10001B5D	> 8B7D 10	MOV EDI,[ARG.3]
10001B60	> 85F6	TEST ESI,ESI
10001B62	> 75 09	JNZ SHORT 10001B6D
10001B64	> 833D F41B0010 00	CMP DWORD PTR DS:[10001BF4],0
10001B6B	> EB 26	JMP SHORT 10001B95
10001B6D	> 83FE 01	CMP ESI,1
10001B70	> 74 05	JE SHORT 10001B77
10001B72	> 83FE 02	CMP ESI,2
10001B75	> 75 22	JNZ SHORT 10001B99
10001B77	> A1 C41C0010	MOV EAX,DWORD PTR DS:[10001CC4]
10001B7C	> 85C0	TEST EAX,EAX
10001B7E	> 74 09	JE SHORT 10001B89
10001B80	> 57	PUSH EDI
10001B81	> 56	PUSH ESI
10001B82	> 53	PUSH EBX

حالا از فایل دامپ می گیریم و بعد با ImportREC دامپ خودمان را فیکس می کنیم.حالا هم اسم فایل آپیک شده را عوض کنید،بذارید:

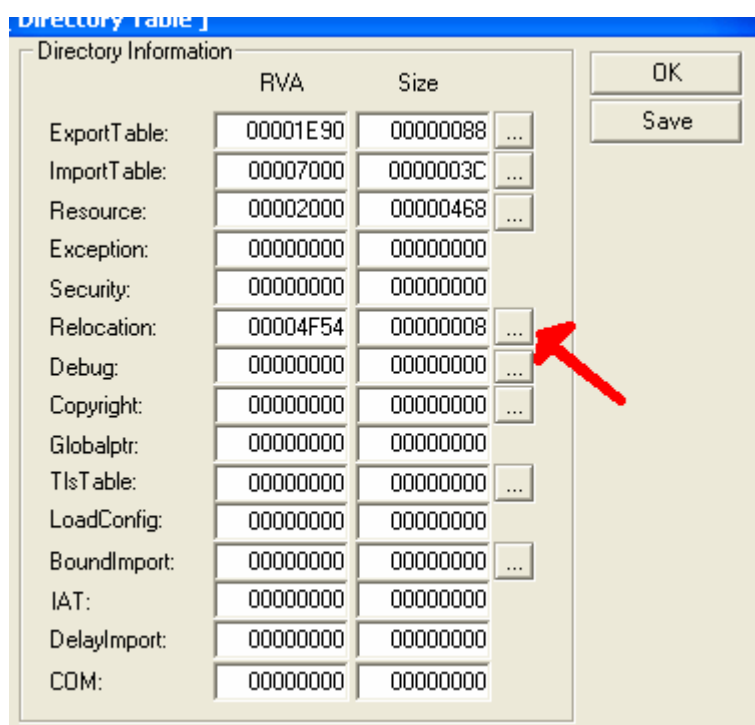
Gooran_Unpacked.dll

در ضمن،تیک آن گزینه DLL که قبلا از مشخصات فایل برداشته بودید را دوباره در فایل آپیک شده تیک بزنید تا مشکلی برای فایل پیش نیاید.

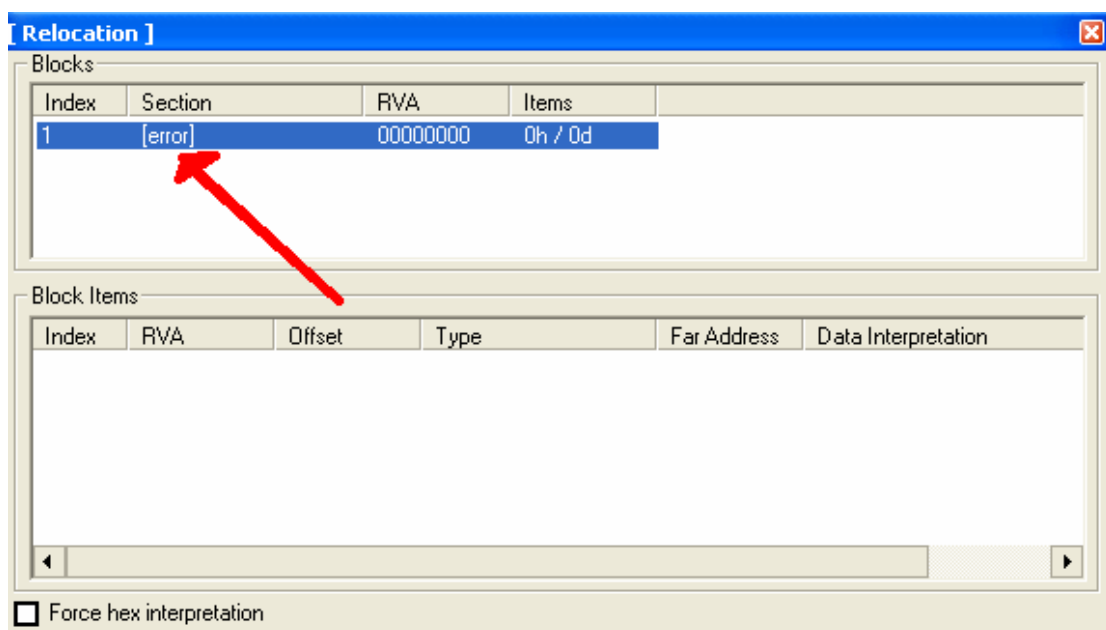
فایل های DLL یک سکشن به نام reloc. دارند که در آن Relocation های فایل DLL را نگه می دارند.فایل آپیک شده را در LordPE اجرا کنید:



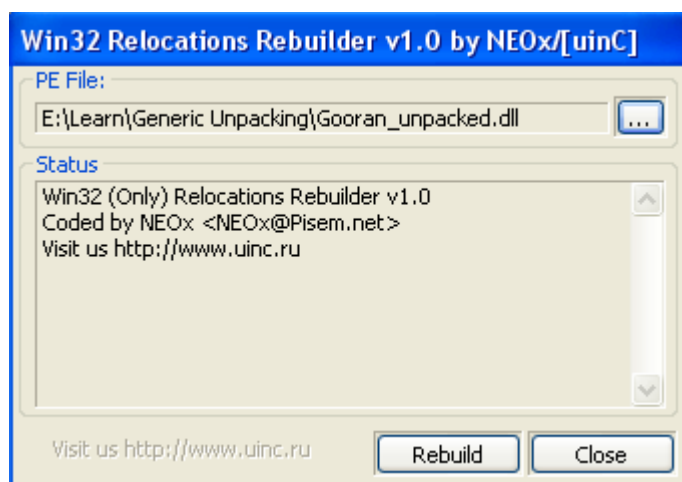
گزینه Directories را بزنید.



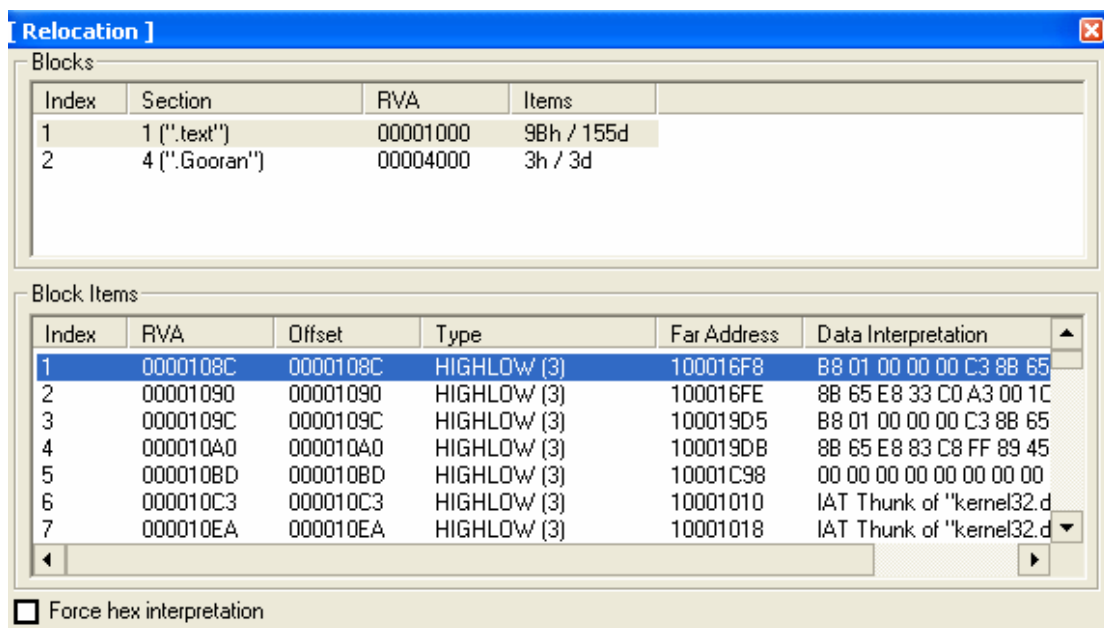
دکمه نشان داده شده در تصویر بالا را بزنید تا Relocation های فایل را ببینید:



همانطور که می بینید Relocation های فایل از بین رفته است.⊗
 برای ساخت دوباره Reloc می توانیم از برنامه PeTools استفاده کنیم. البته نرم افزار تخصصی برای انجام این کار RoleX است.
 برنامه PeTools را باز کنید. در منوی PlugIns گزینه Reloc Rebuilder را بزنید:



فایل مورد نظر را به برنامه بدهید و گزینه Rebuild را بزنید.
 حالا اینبار نگاهی به Relocation های فایل با برنامه LordPE بندازید:

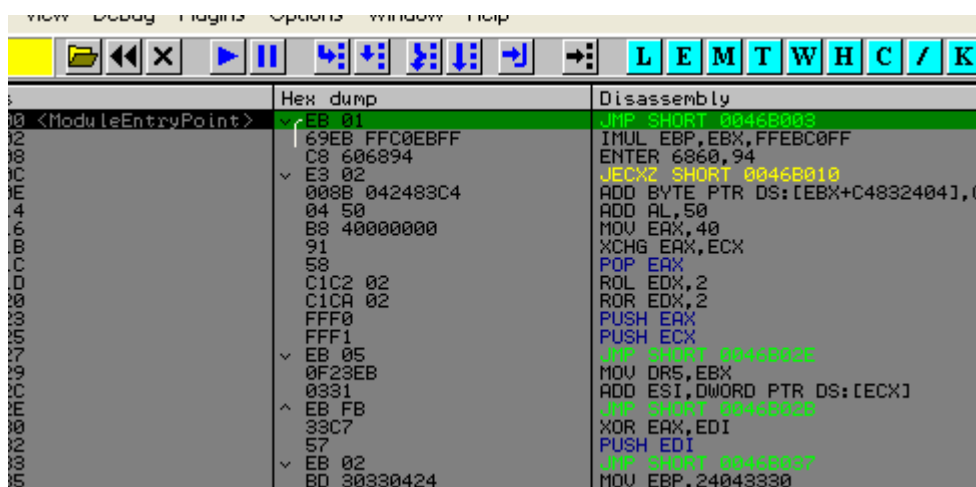


همانطور که می بینید Relocation های فایل درست شده و دیگر مشکلی به وجود نمی آید. اما... اگر در مواردی دیدید که PeTools نمی تواند این کار را انجام دهد. از نرم افزار تخصصی این کار یعنی RoleX استفاده کنید.

آموزش استفاده از RoleX: ابتدا باید ImageBase فایل DLL خود را تغییر بدهید. برای این کار می توانید از فایل Sample.exe (در داخل برنامه موجود است) استفاده کنید. به این ترتیب که فایل اصلی و یک کپی از فایل مورد نظر را در کنار فایل Sample.exe قرار دهید. تا Sample.exe هر دو فایل DLL را اجرا کند و بعد هم از دو فایلی که Load شده، دامپ می گیرد تا به این ترتیب، دو فایل DLL یکسان، با ImageBase متفاوت داشته باشید. (البته برای این کار از نرم افزار DLL Rebase هم می توانید استفاده کنید.) بعد از این هم برنامه Rolex را باز کنید. در قسمت Original فایل اصلی را به برنامه می دهید و در قسمت Compare to فایلی که ImageBase آن تغییر داده شده ست را به برنامه می دهید. بعد گزینه Compare را می زنید تا Relocation های فایل پیدا شود و در آخر هم گزینه Fix PE Module را بزنید و فایلی که Relocation آن مشکل دارد را به برنامه بدهید تا مشکل حل شود.

PolyEnE

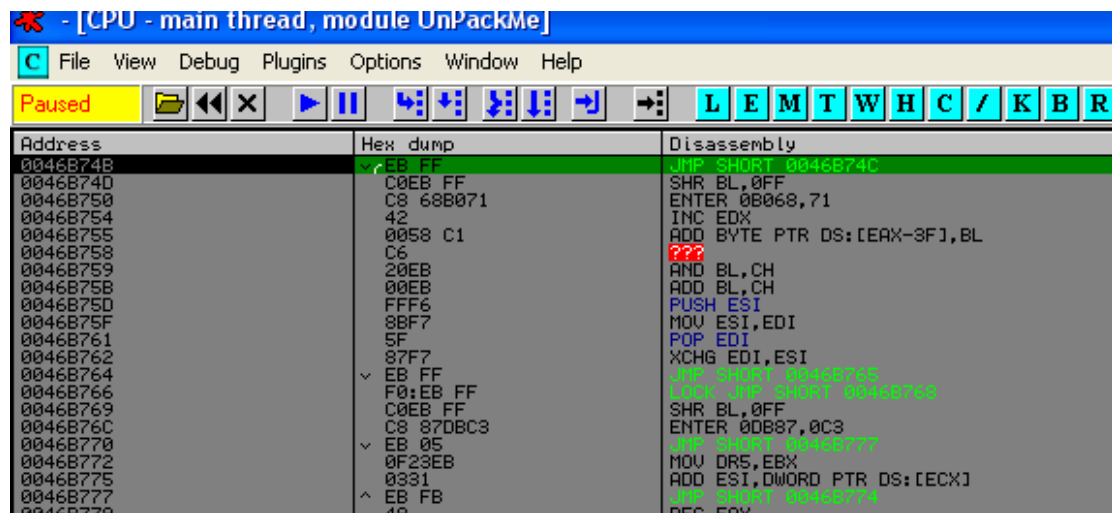
فایل مورد نظر را در OllyDBG باز کنید:



به دلیل قرار دادن پرش های بیخود در داخل کدهای پکر، کدهای پکر مقداری ناخوانا شده است (Junk code) ☹ اما مشکلی نیست. با پلاگین Analysis this اقدام به آنالیز کردن کدهای پکر بکنید. (کلیک راست کرده و گزینه Analysis This را بزنید).



حالا کدها مقداری بهتر شد. با F8 به جلو بروید تا به دستور PushAD برسید و با اجرای دستور PushAD مقدار رجیستر ESP تغییر می کند. حالا هم کلید F9 را بزنید تا برنامه اجرا شود و به انتهای کدهای پکر برسیم.



حالا چند بار کلید F8 را بزنید تا به اینجا برسید:

Address	Hex dump	Disassembly
0046B76B	FFC8	DEC EAX
0046B76D	87DB	XCHG EBX,EBX
0046B76F	C3	RET
0046B770	EB 05	JMP SHORT 0046B777
0046B772	0F23EB	MOV DR5,EBX
0046B775	0331	ADD ESI,DWORD PTR DS:[ECX]
0046B777	EB FB	JMP SHORT 0046B774
0046B779	48	DEC EAX
0046B77A	40	INC EAX
0046B77B	83EC 04	SUB ESP,4
0046B77E	C70424 E6B24600	MOV DWORD PTR SS:[ESP],0046B2E6
0046B785	58	POP EAX
0046B786	33C7	XOR EAX,EDI
0046B788	57	PUSH EDI
0046B789	EB 02	JMP SHORT 0046B78D
0046B78B	BD 30330424	MOV EBP,24043330
0046B790	5F	POP EDI
0046B791	87D2	XCHG EDX,EDX
0046B793	C3	RET

این دستور ما را از سکشن PolyEnE. به سکشن text. می برد. یعنی با این دستور به OEP می رویم، یک بار کلید F8 را بزنید:

Address	Hex dump	Disassembly	Comment
0042719C	80CC 03	OR AH,3	
0042719F	F7C2 00000400	TEST EDX,40000	
004271A5	74 03	JE SHORT 004271AA	
004271A7	80CC 10	OR AH,10	
004271A9	C3	RET	
004271AB	90	NOP	
004271AC	90	NOP	
004271AD	90	NOP	
004271AE	90	NOP	
004271AF	90	NOP	
004271B0	55	PUSH ESP	
004271B1	8BEC	MOV EBP,ESP	
004271B3	6A FF	PUSH -1	
004271B5	68 600E4500	PUSH 00450E60	
004271B8	68 C8924200	PUSH 004292C8	
004271BF	64:A1 00000000	MOV EAX,DWORD PTR FS:[0]	
004271C5	58	PUSH EAX	
004271C6	64:8925 00000000	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 SS:[EBP-18],ESP	
004271D6	FF15 DC0A4600	CALL DWORD PTR DS:[460ADC]	kernel
004271DC	33D2	XOR EDX,EDX	
004271DE	8AD4	MOV DL,AH	
004271E0	8915 34E64500	MOV DWORD PTR DS:[45E634],EDX	
004271E6	8BC8	MOV ECX,EAX	
004271E8	81E1 FF000000	AND ECX,0FF	
004271EE	89D0 30E64500	MOV DWORD PTR DS:[45E630],ECX	
004271F4	C1E1 08	SHL ECX,8	
004271F7	03CA	ADD ECX,EDX	
004271F9	89D0 2CE64500	MOV DWORD PTR DS:[45E62C],ECX	
004271FF	C1E8 10	SHR EAX,10	

از این به بعد دیگر مشکلی نیست. دامپ می گیرید و فیکس می کنید.

Fusion

فایل را در OllyDBG باز کنید:

0046B000	EB 15	JMP SHORT UnPackMe.0046B017
0046B002	0300	ADD EAX,DWORD PTR DS:[EAX]
0046B004	0000	ADD BYTE PTR DS:[EAX],AL
0046B006	06	PUSH ES
0046B007	0000	ADD BYTE PTR DS:[EAX],AL
0046B009	0000	ADD BYTE PTR DS:[EAX],AL
0046B00B	0000	ADD BYTE PTR DS:[EAX],AL
0046B00D	0000	ADD BYTE PTR DS:[EAX],AL
0046B00F	0000	ADD BYTE PTR DS:[EAX],AL
0046B011	0068 00	ADD BYTE PTR DS:[EAX],CH
0046B014	0000	ADD BYTE PTR DS:[EAX],AL
0046B016	0055 E8	ADD BYTE PTR SS:[EBP-18],DL
0046B018	0000	ADD BYTE PTR DS:[EAX],AL

دو بار کلید F8 را بزنید تا رجیستر ESP مقدارش تغییر کند. حالا همانند مثال های قبلی روی مقدار رجیستر ESP یک Hardware breakpoint on access می گذارید و بعد با F9 برنامه را اجرا می کنید:

0046E7F9	83C0 30	ADD EAX,30
0046E7FC	3B00	MOV EAX,DWORD PTR DS:[EAX]
0046E7FE	0385 E8070000	ADD EAX,DWORD PTR SS:[EBP+7E8]
0046E804	80D0 F5000000	LEA EBX,DWORD PTR SS:[EBP+BF5]
0046E80A	5D	POP EBP
0046E808	50	PUSH EAX
0046E80C	60	PUSHAD
0046E80D	55	PUSH EBP
0046E80E	53	PUSH EBX
0046E80F	64:FF35 00000000	PUSH DWORD PTR FS:[0]
0046E816	64:8925 00000000	MOV DWORD PTR FS:[0],ESP
0046E81D	E8 64000000	CALL UnPackMe.0046E886
0046E822	DF01	FILD WORD PTR DS:[ECX]
0046E824	C2 DF21	RETN 210F
0046E827	DF10	FIST WORD PTR DS:[EAX]
0046E829	0048 BE	ADD BYTE PTR DS:[EAX-42],CL
0046E82C	47	INC EDI
0046E82D	0000	ADD BYTE PTR DS:[EAX],AL
0046E82F	00E8	ADD AL,CH
0046E831	51	PUSH ECX
0046E832	0000	ADD BYTE PTR DS:[EAX],AL
0046E834	00DF	ADD BH,BL
0046E836	00C3	ADD BL,AL
0046E838	0321	ADD ESP,DWORD PTR DS:[ECX]
0046E83A	DF10	FIST WORD PTR DS:[EAX]
0046E83C	0048 04	ADD BYTE PTR DS:[EAX-3C],CL

یکبار دیگر کلید F9 را بزنید. اینبار درست در OEP فرود می آید.

004271AE	90	NOP
004271AF	90	NOP
004271B0	55	PUSH EBP
004271B1	8BEC	MOV EBP,ESP
004271B3	6A FF	PUSH -1
004271B5	68 600E4500	PUSH UnPackMe.00450E60
004271B8	68 C8924200	PUSH UnPackMe.004292C8
004271BF	64:A1 00000000	MOV EAX,DWORD PTR FS:[0]
004271C5	50	PUSH EAX
004271C6	64:8925 00000000	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 SS:[EBP-18],ESP
004271D6	FF15 DC0A4600	CALL DWORD PTR DS:[460ADC]
004271DC	33D2	XOR EDX,EDX
004271DE	8AD4	MOV DL,AH
004271E0	8915 34E64500	MOV DWORD PTR DS:[45E634],EDX
004271E6	8BC8	MOV ECX,EAX
004271E8	81E1 FF000000	AND ECX,0FF
004271EE	890D 30E64500	MOV DWORD PTR DS:[45E630],ECX
004271F4	C1E1 08	SHL ECX,8
004271F7	03CA	ADD ECX,EDX
004271F9	890D 2CE64500	MOV DWORD PTR DS:[45E62C],ECX
004271FF	C1E8 10	SHR EAX,10
00427202	A3 28E64500	MOV DWORD PTR DS:[45E628],EAX
00427207	E8 94210000	CALL UnPackMe.004293A0
0042720C	85C8	TEST EAX,EAX

راه حل دیگر:

اصولا اگر بدانیم که برنامه ما در چه زبانی نوشته شده است، خیلی راحت تر می توانیم OEP را پیدا کنیم...چه طوری؟؟...برای مثال من اگر بفهمم که این برنامه در Visual C++ نوشته شده است، و چون می دانم که معمولا بعد از Entry point (در مورد فایل های یک شده OEP) تابع GetVersion یا GetVersionExA استفاده می شود. پس می توانم از همین تابع برای یافتن OEP استفاده کنم. برنامه را ری استارت کرده، کلید های CTRL + G را بزنید و بنویسید: GetVersion
حالا کلید F2 را بزنید تا بر روی این تابع Breakpoint بگذارید. و بعد برنامه را با F9 اجرا کنید. در همانجایی که Breakpoint گذاشته بودیم، متوقف شدیم:

7C8111D8	90	NOP	
7C8111D9	90	NOP	
7C8111DA	90	NOP	
7C8111DB	64 01 18000000	MOV EAX,DWORD PTR FS:[18]	
7C8111DC	8B48 30	MOV ECX,DWORD PTR DS:[EAX+30]	
7C8111DD	8B81 B0000000	MOV EAX,DWORD PTR DS:[ECX+B0]	
7C8111DE	0FB791 AC000000	MOVZX EDX,WORD PTR DS:[ECX+AC]	
7C8111DF	83F0 FE	XOR EAX,FFFFFFFF	
7C8111E0	C1E0 0E	SHL EAX,0E	
7C8111E1	0BC2	OR EAX,EDX	
7C8111E2	C1E0 08	SHL EAX,8	
7C8111E3	0B81 A8000000	OR EAX,DWORD PTR DS:[ECX+A8]	
7C8111E4	C1E0 08	SHL EAX,8	
7C8111E5	0B81 A4000000	OR EAX,DWORD PTR DS:[ECX+A4]	
7C8111E6	C3	RETN	
7C8111E7	90	NOP	
7C8111E8	90	NOP	
7C8111E9	90	NOP	
7C8111EA	90	NOP	
7C8111EB	90	NOP	
7C8111EC	8BFF	MOV EDI,EDI	
7C8111ED	55	PUSH EBP	
7C8111EE	8BEC	MOV EBP,ESP	
7C8111EF	83EC 24	SUB ESP,24	
7C8111F0	57	PUSH EDI	
7C8111F1	6A 07	PUSH 7	
7C8111F2	33C0	XOR EAX,EAX	

حالا کلید ALT + F9 را بزنید(کلید های ALT + F9 کدها را تا خطی که به کدهای فایل اصلی خودمان برسیم،اجرا می کند).

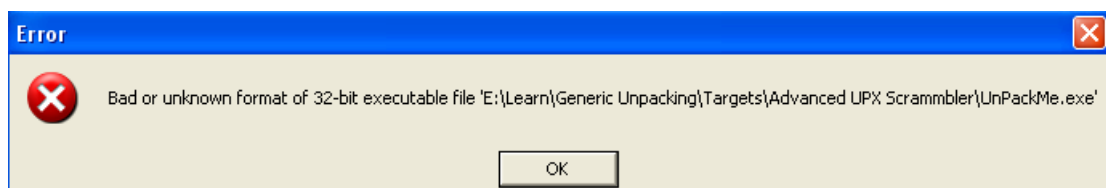
004271AD	90	NOP	
004271AE	90	NOP	
004271AF	90	NOP	
004271B0	55	PUSH EBP	
004271B1	8BEC	MOV EBP,ESP	
004271B2	6A FF	PUSH -1	
004271B3	68 600E4500	PUSH UnPackMe.00450E60	
004271B4	68 C8924200	PUSH UnPackMe.004292C8	
004271B5	64:A1 00000000	MOV EAX,DWORD PTR FS:[0]	SE handler installation
004271B6	50	PUSH EAX	
004271B7	64:8925 000000	MOV DWORD PTR FS:[0],ESP	
004271B8	83C4 A8	ADD ESP,-58	
004271B9	53	PUSH EBX	
004271BA	56	PUSH ESI	
004271BB	57	PUSH EDI	
004271BC	8965 E8	MOV DWORD PTR SS:[EBP-18],ESP	
004271BD	FF15 DC0A4600	CALL DWORD PTR DS:[460ADC]	kernel32.GetVersion
004271BE	33D2	XOR EDX,EDX	
004271BF	8D04	MOV DL,AH	
004271C0	8915 34E64500	MOV DWORD PTR DS:[45E634],EDX	
004271C1	8BC8	MOV ECX,EAX	
004271C2	81E1 FF000000	AND ECX,0FF	
004271C3	8900 30E64500	MOV DWORD PTR DS:[45E630],ECX	
004271C4	C1E1 08	SHL ECX,8	
004271C5	03CA	ADD ECX,EDX	
004271C6	8900 2CE64500	MOV DWORD PTR DS:[45E62C],ECX	
004271C7	C1E8 10	SHR EAX,10	
004271C8	A3 28E64500	MOV DWORD PTR DS:[45E628],EAX	
004271C9	E8 94210000	CALL UnPackMe.004293A0	
004271CA	85C0	TEST EAX,EAX	

همانطور که می بینید چند خط بالاتر از خطی که الان هستیم،یعنی خط 004271B0 OEP ما قرار دارد ☺
همانطور که می بینید بلد بودن Language Structure کار ما را در یافتن OEP بسیار راحت می کند.و به نظر من یکی از راحت ترین راه ها برای یافتن OEP در مورد پکرهایی است که با ساختار آن آشنایی زیادی ندارند.

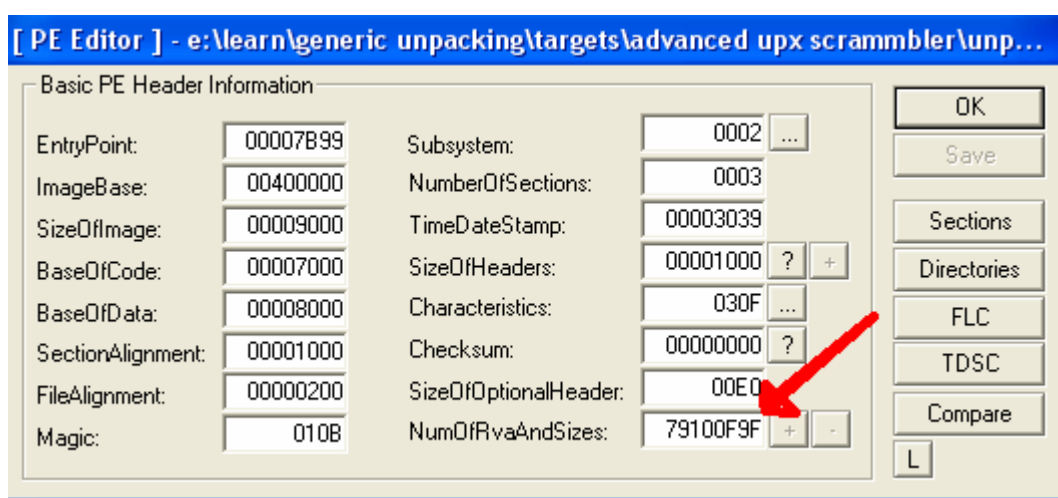
بعد از یافتن OEP مشکل خاصی نیست.دامپ می کنید و دامپ خود را فیکس می کنید.(حواستان باشد اگر فایل دامپ شده در سیستم شما اجرا می شود...در سیستم دیگری اجرا نمی شود،**پس حتما باید IAT را فیکس کنید.**)

Advanced UPX Scrambler

فایل را در OllyDBG (بدون پلاگین OllyAdvanced) باز کنید:



می بینید؟ OllyDBG... نتوانسته است مشخصات فایل را درست تشخیص دهد؟... چرا؟... چون پکر ما مشخصات فایل را مقداری دستکاری کرده است (PE header trick)... شاید این دستکاری مشخصات فایل مشکلی در اجرا شدن فایل در ویندوز ایجاد نکند... ولی باعث ایجاد مشکلاتی در زمان اجرای فایل در OllyDBG می شود. ☹ به همین دلیل ما باید مشخصات فایل را درست کنیم. فایل را در LordPE باز کنید تا ببینیم مشکل کجاست؟



همانطور که می بینید مقدار NumofRvaandSizes برابر 79100F9F است!... این مقدار در فایل های Exe برابر با ۱۰ است. (یا عددی در نزدیکی ۱۰)... لذا این مقدار غیرطبیعی است و به خاطر همین دستکاری، OllyDBG نمی تواند فایل را درست باز کند. این مقدار را ۱۰ کنید تا فایل به درستی در Olly اجرا شود. البته به غیر از این مقدار خیلی از مشخصات دیگر ممکن بود دستکاری شده باشد، مانند:

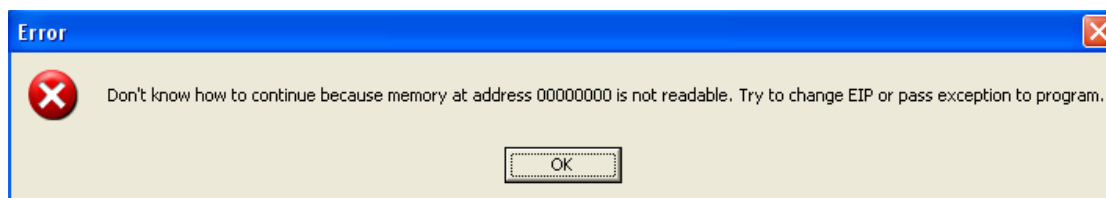
SizeofImage
BaseofCode
BaseofData
TimeDateStamp
SizeofHeaders
Characteristics

پس همیشه حواستان به این مقدارها باشد.

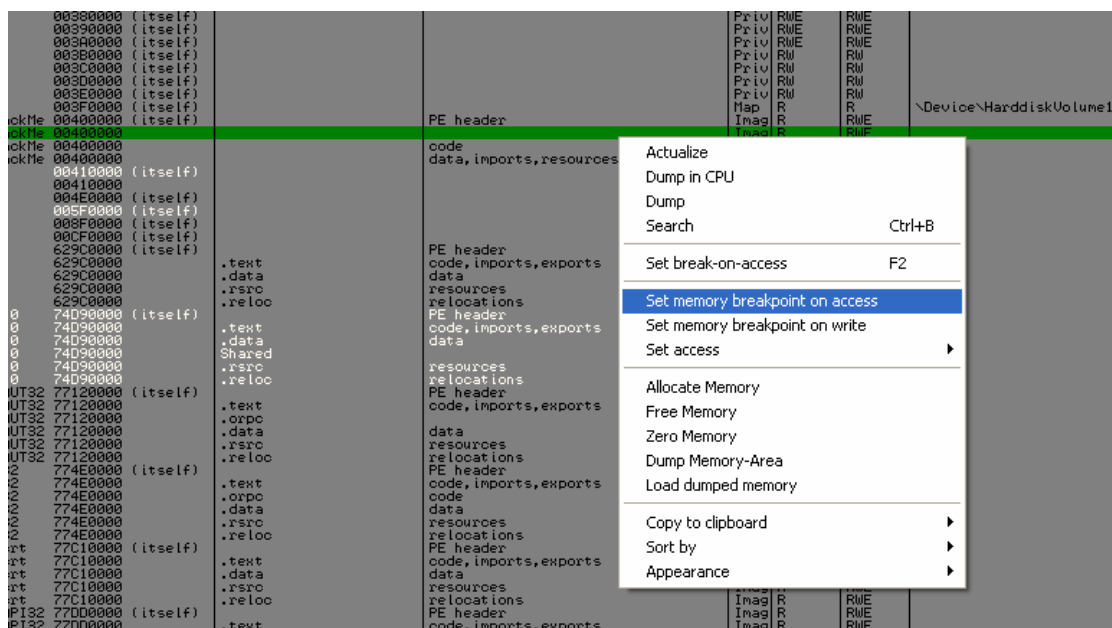
البته توجه داشته باشید که پلاگین OllyAdvanced این مشکل را در اکثر موارد حل می کند (نه همه موارد). یعنی فایل هایی که این مشکل را دارند، با این پلاگین به راحتی در OllyDBG باز می شوند.

یکی دیگر از راه های پراستفاده در یافتن OEP استفاده از Memory Breakpoint on Access می باشد... ما با گذاشتن Memory Breakpoint on access روی سکشنی که در آن کدهای اصلی ما قرار دارد، در صورتی که وارد سکشن اصلی شدیم یا اگر عملیات Decrypt روی سکشن اصلی انجام شود، متوقف می شویم... پس از این طریق می توانیم OEP را پیدا کنیم. اما با Memory breakpoint گذاشتن روی سکشن اصلی ممکن است در خیلی جاها متوقف شویم... لذا ما باید در جایی Memory breakpoint بگذاریم که دیگر عملیات Decrypt تمام شده باشد... معمولا اگر خطایی در کدهای پکر ما اتفاق بیفتد... می توانیم بعد از آخرین خطایی که در کدهای پکر اتفاق می افتد، Memory Breakpoint بگذاریم و به این ترتیب OEP را پیدا کنیم. این روش در مورد خیلی از پکرها و پروتکتورها جواب می دهد.

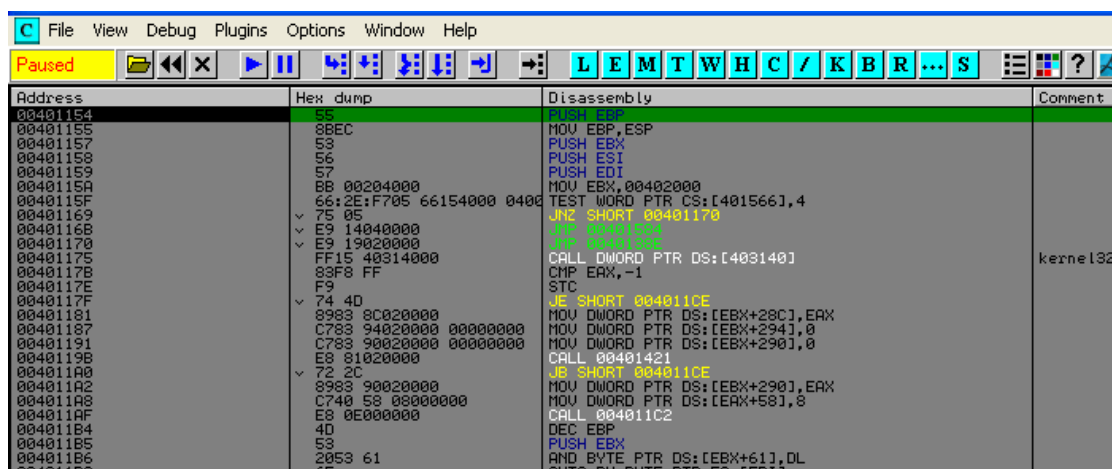
فایل را در OllyDBG باز کنید و کلید F9 را بزنید:



همانطور که می بینید برنامه ما دچار خطا شده، اینجا جایی است که کدهای سکشن اصلی ما Decrypt شده است. کلید OK را بزنید تا این پیغام برود و بعد کلیدهای ALT+M را بزنید و همانند تصویر بر روی سکشن اصلی پکر(در اینجا اولین سکشن زیر PE Header) یک Memory breakpoint on access بگذارید.



حالا کلید Shift + F9 را بزنید.(چون در برنامه خطا افتاده برای ادامه برنامه باید کلید Shift را هم نگه دارید).



همانطور که می بینید وارد سکشن اصلی برنامه شدیم و به OEP رسیدیم...حالا از فایل دامپ بگیرید و آن را فیکس کنید.

Orien

فایل مورد نظر را در OllyDBG باز کنید. دوبار کلید F8 را بزنید تا دستور PushAD اجرا شود. حالا بر روی مقدار رجیستر ESP یک Hardware Breakpoint on access بگذارید و بعد کلید F9 را بزنید:

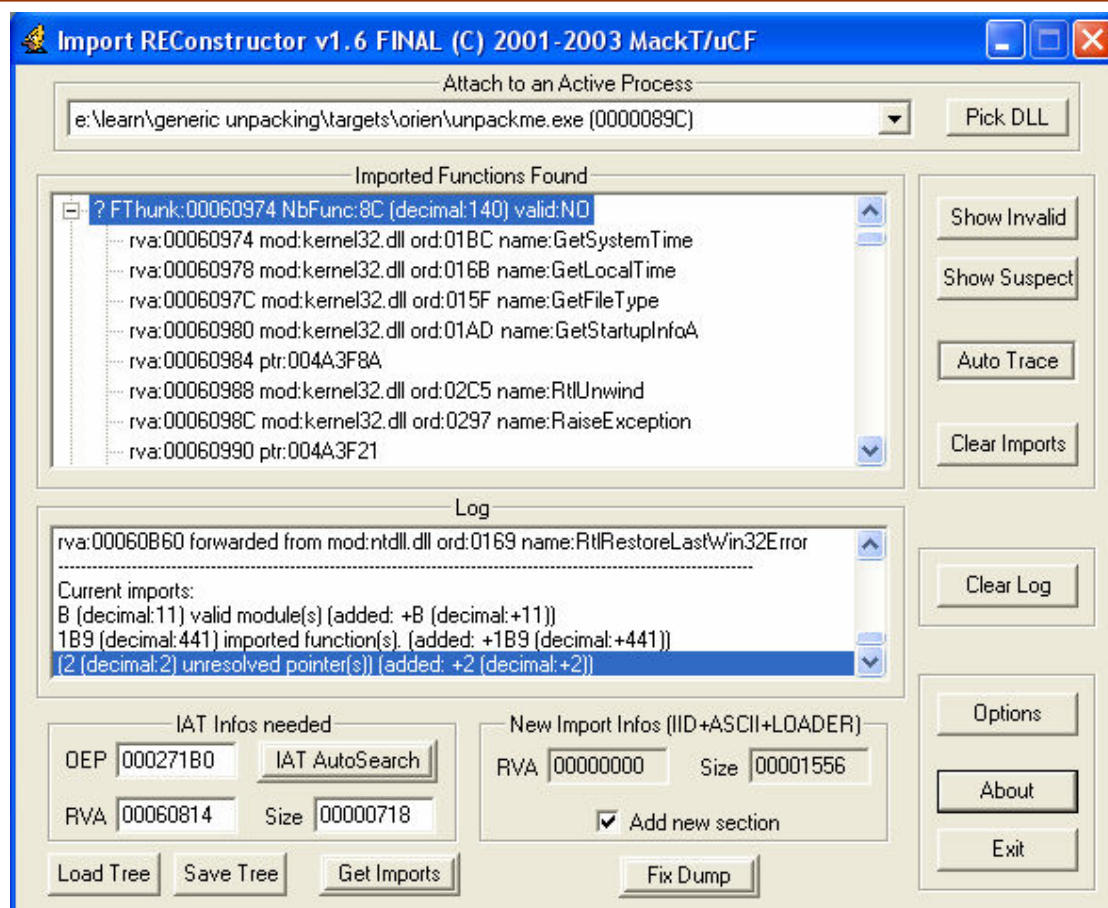
Address	Hex dump	Disassembly
004A3AD3	B6 E9	MOV DH,0E9
004A3AD5	6A 0A	PUSH 0A
004A3AD7	0000	ADD BYTE PTR DS:[EAX],AL
004A3AD9	CD 61	INT 61
004A3A0B	B8 00710200	MOV EAX,271B0
004A3AE0	83F8 00	CMP EAX,0
004A3AE3	74 13	JE SHORT 004A3AF8
004A3AE5	05 00004000	ADD EAX,00400000
004A3AEA	EB 08	JMP SHORT 004A3AF4
004A3AEC	49	DEC ECX
004A3AED	E5 24	IN EAX,24
004A3AEF	15 20FFE0CD	ADC EAX,CDE0FF20
004A3AF4	EB FB	JMP SHORT 004A3AF1
004A3AF6	EB 10	JMP SHORT 004A3B08
004A3AF8	33C0	XOR EAX,EAX
004A3AFA	EB 06	JMP SHORT 004A3B02
004A3AFC	43	INC EBX
004A3AFD	66:B8 83C0	MOV AX,0C083
004A3B01	0083 E8FFC200	ADD BYTE PTR DS:[EBX+C2FFE8],AL
004A3B07	00E9	ADD CL,CH
004A3B09	87E7	XCHG EDI,ESP
004A3B0B	FFFF	???
004A3B0D	E8 1C000000	CALL 004A3B2E
004A3B12	E8 01000000	CALL 004A3B18
004A3B17	E9 83ECFC6A	JMP 6B47273F
004A3B1C	FFE8	JMP FAR EAX
004A3B1E	05 000000EB	ADD EAX,EB000000
004A3B23	0A88 FCB6FFA3	OR CL,BYTE PTR DS:[EAX+A3FFB6FC]
004A3B29	BF 3A0000CD	MOV EDI,CD00003A
004A3B2E	52	PUSH EDX
004A3B2F	83F8 00	CMP EAX,0
004A3B32	74 2E	JE SHORT 004A3B62
004A3B34	8038 00	CMP BYTE PTR DS:[EAX],0
004A3B37	74 29	JE SHORT 004A3B62
004A3B39	EB 01	JMP SHORT 004A3B3C
004A3B3B	A8 8D	TEST AL,8D
004A3B3D	93	XCHG EAX,EBX
004A3B3E	55	PUSH EBP
004A3B3F	3800	CMP BYTE PTR DS:[EAX],AL
004A3B41	00E8	ADD AL,CH
004A3B42	0100	ADD BYTE PTR DS:[EAX],EAX

این دستور دیگر آخر کدهای پکر ماست. کافیست چند بار کلید F8 را بزنید تا به این خط برسید:

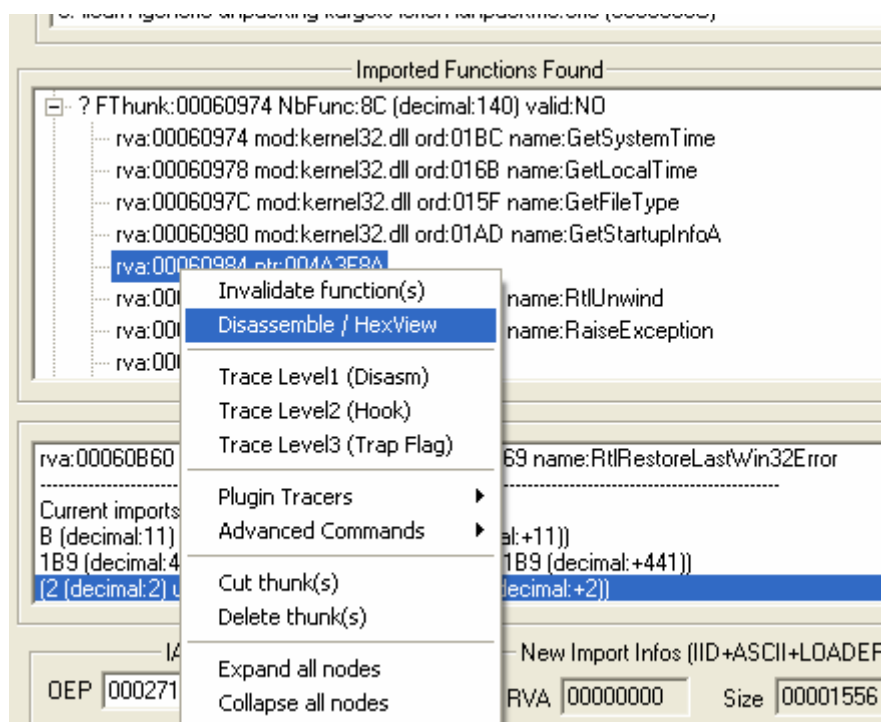
Address	Hex dump	Disassembly
004A3AF1	FFEB	JMP EAX
004A3AF3	CD EB	INT 0EB
004A3AF5	FB	STI
004A3AF6	EB 10	JMP SHORT 004A3B08
004A3AF8	33C0	XOR EAX,EAX
004A3AFA	EB 06	JMP SHORT 004A3B02
004A3AFC	43	INC EBX
004A3AFD	66:B8 83C0	MOV AX,0C083
004A3B01	0083 E8FFC200	ADD BYTE PTR DS:[EBX+C2FFE8],AL
004A3B07	00E9	ADD CL,CH
004A3B09	87E7	XCHG EDI,ESP
004A3B0B	FFFF	???
004A3B0D	E8 1C000000	CALL 004A3B2E
004A3B12	E8 01000000	CALL 004A3B18
004A3B17	E9 83ECFC6A	JMP 6B47273F
004A3B1C	FFE8	JMP FAR EAX
004A3B1E	05 000000EB	ADD EAX,EB000000
004A3B23	0A88 FCB6FFA3	OR CL,BYTE PTR DS:[EAX+A3FFB6FC]
004A3B29	BF 3A0000CD	MOV EDI,CD00003A
004A3B2E	52	PUSH EDX
004A3B2F	83F8 00	CMP EAX,0
004A3B32	74 2E	JE SHORT 004A3B62
004A3B34	8038 00	CMP BYTE PTR DS:[EAX],0
004A3B37	74 29	JE SHORT 004A3B62
004A3B39	EB 01	JMP SHORT 004A3B3C
004A3B3B	A8 8D	TEST AL,8D
004A3B3D	93	XCHG EAX,EBX
004A3B3E	55	PUSH EBP

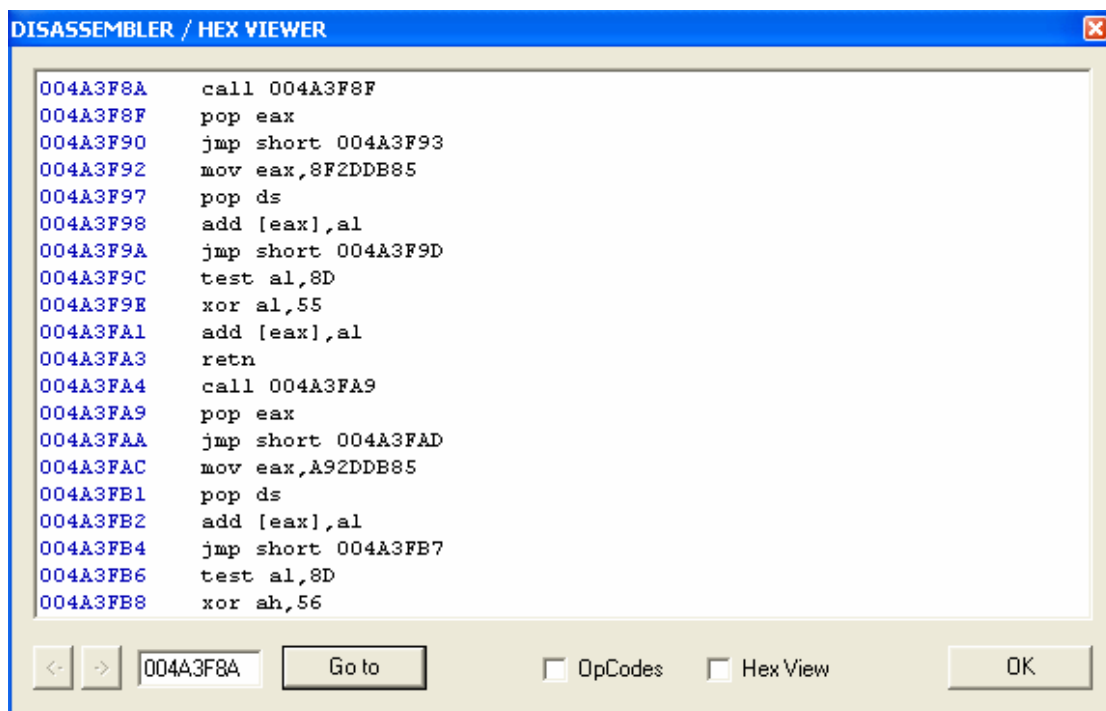
این دستور درست به OEP می رود.

حالا از فایل دامپ بگیرید و بعد ImportREC را باز کنید. فایل مورد نظر را به برنامه می دهید. و OEP بر حسب RVA را هم بنویسید:



همانطور که می بینید دو تابع Invalid داریم ☺ ممکن است فکر کنید که این توابع بدرندخور هستند و می تواند بر روی آنها کلیک راست کرد و گزینه Cut Thunk را زد و آنها را از بین برد. ولی نه!...این طور نیست...از کجا فهمیدم؟...بر روی یک از توابع Invalid کلیک راست کنید و گزینه Disassemble/Hex View را بزنید:

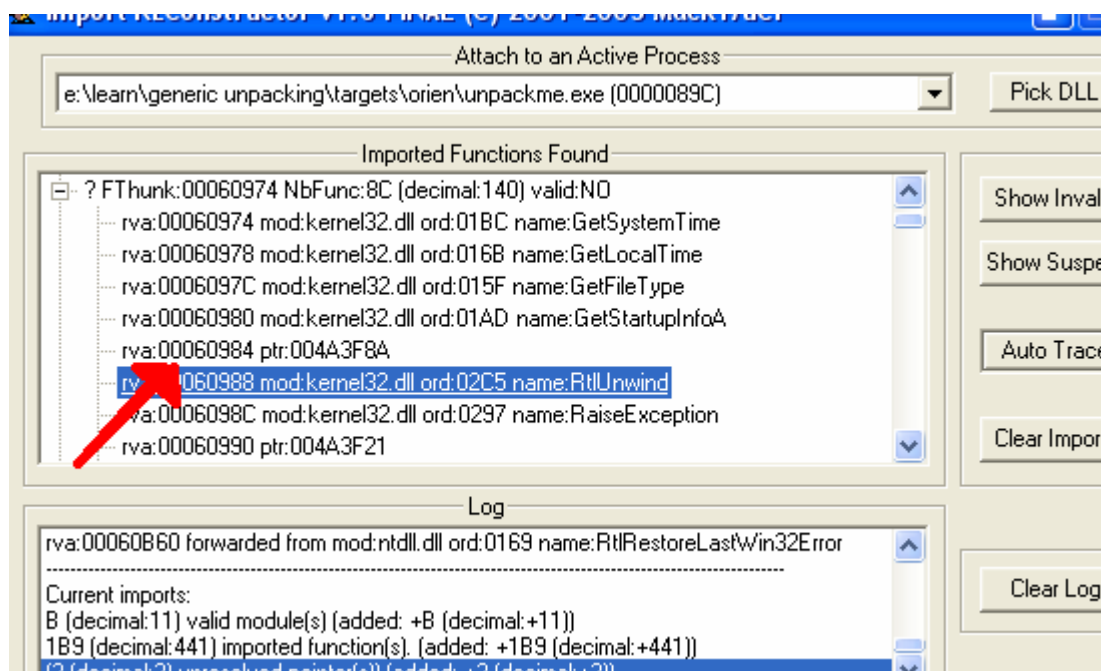




همانطور که در تصویر بالا می بینید دستورهای بالا دستوراتی هستند که مشخص و معنی دار هستند. (تابع دیگر هم همینطور است). لذا این دو تابع نمی توانند دو تابع بدردنخور باشند.

بینید... این پکر از قابلیت Import Redirection استفاده می کند. یعنی به جای اینکه باید آدرس واقعی این دو تابع را بنویسد، آدرس خطی از برنامه را می نویسد که به آدرس واقعی تابع می رود.

خوب، حالا چه کار کنیم؟... راه حل های زیادی برای از بین بردن Import Redirection وجود دارد. که یکی از معمولترین آنها این است که ما در درون کدهای پکر جستجو کنیم و بینم در کجا این اتفاق می افتد؟... یعنی در کجا پکر آدرس واقعی تابع را از بین می برد و جای آن یک آدرس دیگر می نویسد؟ (به این روش یافتن Magic Jump می گویند.)

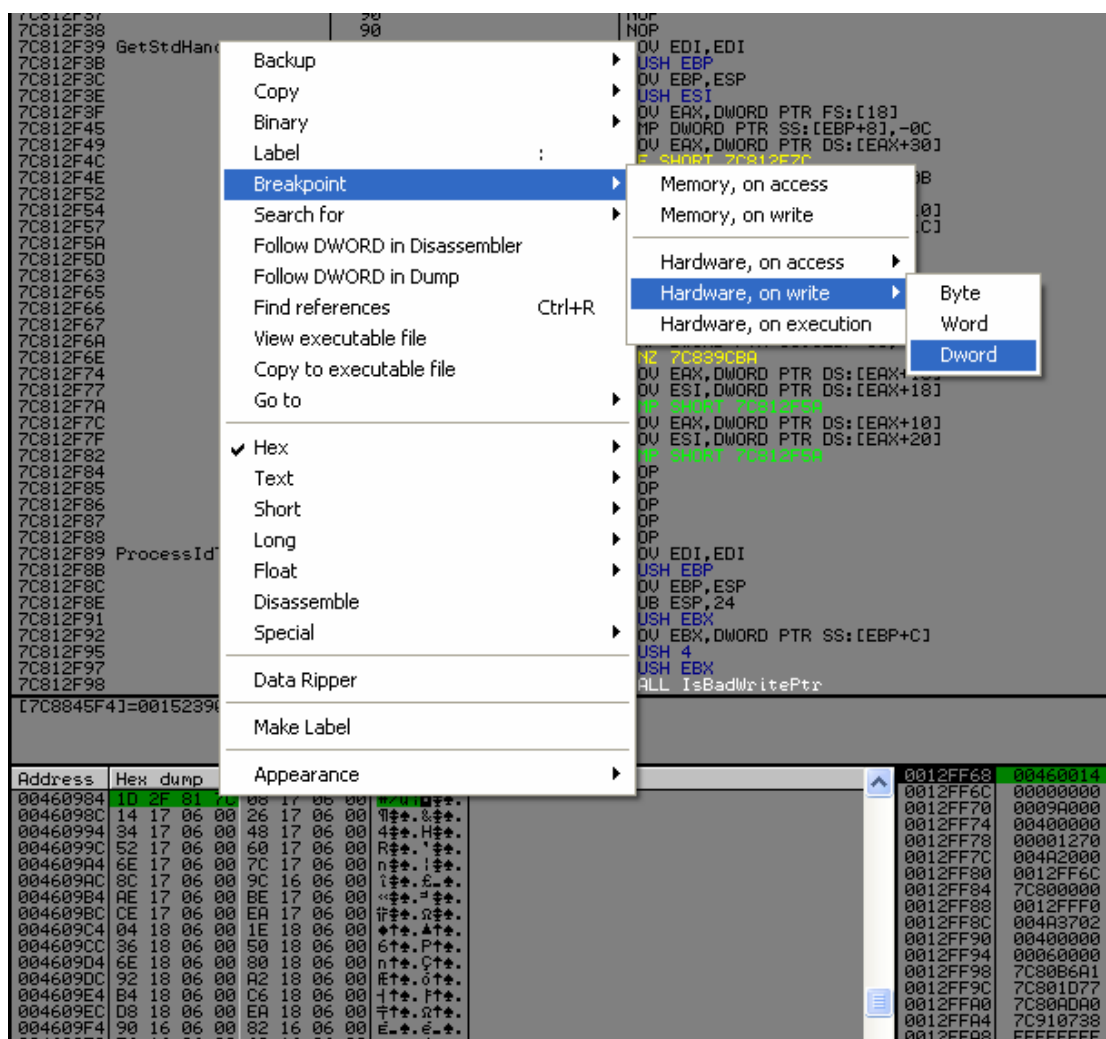


همانطور که در تصویر بالا می بینید یکی از توابع Invalid در خط 00060984 قرار دارد. برای اینکه Magic Jump را پیدا کنیم، باید بر روی این خط یک Hardware Breakpoint بگذاریم.

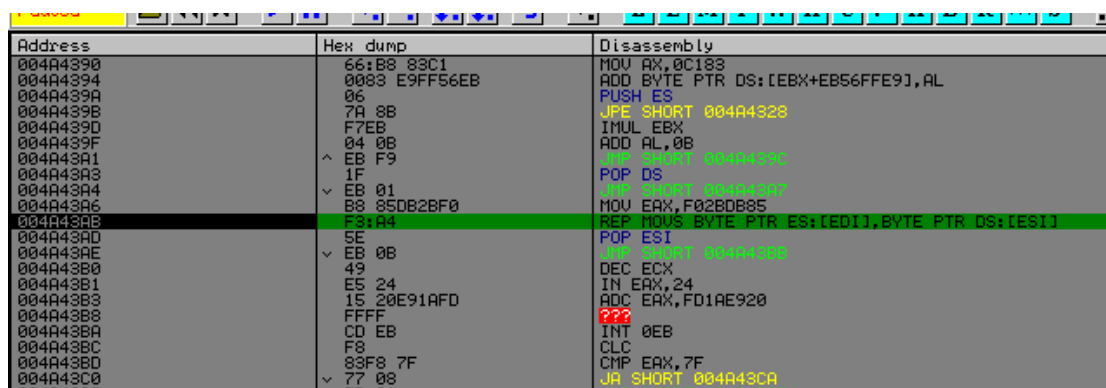
پس در OllyDBG در پنجره دامپ کلیک راست کرده، گزینه Go to و سپس گزینه Expression را بزنید و محل تابع Invalid را به برنامه بدهید، گزینه RVA را هم تیک بزنید:



حالا همانند تصویر بر روی خط مورد نظر (460984) یک Hardware Breakpoint on write ممکن است سوال کنید چرا on write گذاشتم؟... خوب برای اینکه من می خوام ببینم این آدرس در کجای برنامه نوشته شده است.



حالا برنامه را Re-Start کرده و با F9 اجرا کنید. من در اینجا متوقف شدم:



خوب، بهتر است ببینیم که چه بلایی بر روی تابع مورد نظرم می آید؟ برای این کار در منوی Debug گزینه Hardware Breakpoints را بزنید و گزینه Follow (جلوی خط 460984) را بزنید.

ECX=000005A7 (decimal 1447.)			
DS:[ESI]=[00460271]=16			
ES:[EDI]=[00460985]=00			
Address	Hex dump	ASCII	
00460984	F6 00 00 00 00 00 00 00	+.....	
0046098C	00 00 00 00 00 00 00 00		
00460994	00 00 00 00 00 00 00 00		
0046099C	00 00 00 00 00 00 00 00		
004609A4	00 00 00 00 00 00 00 00		
004609AC	00 00 00 00 00 00 00 00		
004609B4	00 00 00 00 00 00 00 00		

همانطور که در تصویر بالا مشاهده می کنید، هنوز آدرس در خط 460984 نوشته نشده است. پس یکبار دیگر کلید F9 را بزنید.

ECX=000005A5 (decimal 1445.)			
DS:[ESI]=[00460273]=00			
ES:[EDI]=[00460987]=00			
Address	Hex dump	ASCII	
00460984	F6 16 06 00 00 00 00 00	+..+.....	
0046098C	00 00 00 00 00 00 00 00		
00460994	00 00 00 00 00 00 00 00		
0046099C	00 00 00 00 00 00 00 00		
004609A4	00 00 00 00 00 00 00 00		

اینبار هم هنوز آدرسی در خط 460984 نوشته نشده است. یکبار دیگر کلید F9 را بزنید:

004A4B6D=004A4B6D			
Address	Hex dump	ASCII	
00460984	1D 2F 81 7C 08 17 06 00	#/U!H..	
0046098C	14 17 06 00 26 17 06 00	..%..&..	
00460994	34 17 06 00 48 17 06 00	4..H..	
0046099C	52 17 06 00 60 17 06 00	R..*..	

آها...! همانطور که می بینید در خط 460984 نوشته شده: 7C812F1D (باید بایت ها را از آنور بخوانید... بلد هستید که؟... قبلا توضیح دادم... در ضمن این آدرس در سیستم شما فرق دارد.)

این آدرس تابع GetCommandLineA دز سیستم من است، پس این آدرس در واقع GetCommandLineA بوده © یکبار دیگر کلید F9 را بزنید:

004A3859=004A3859			
Address	Hex dump	ASCII	
00460984	8A 3F 4A 00 40 7A 93 7C	é?J. @zô!	
0046098C	09 2A 81 7C DA CD 81 7C	.*u!r=u!	
00460994	16 1E 80 7C 15 99 80 7C	..ΔC!S0C!	
0046099C	F8 0E 81 7C B6 2B 81 7C	°fû!H+u!	
004609A4	F4 9A 8A 7C 51 9A 8A 7C	zûC!ôûC!	

می بینید؟... آدرس GetCommandLineA پاک شده و جای آن خط 4A3F8A نوشته شده؟... حالا باید چه کار کنم؟... در پنجره Dissembler با موس اندکی به بالا بروید. تا به اینجا برسید:

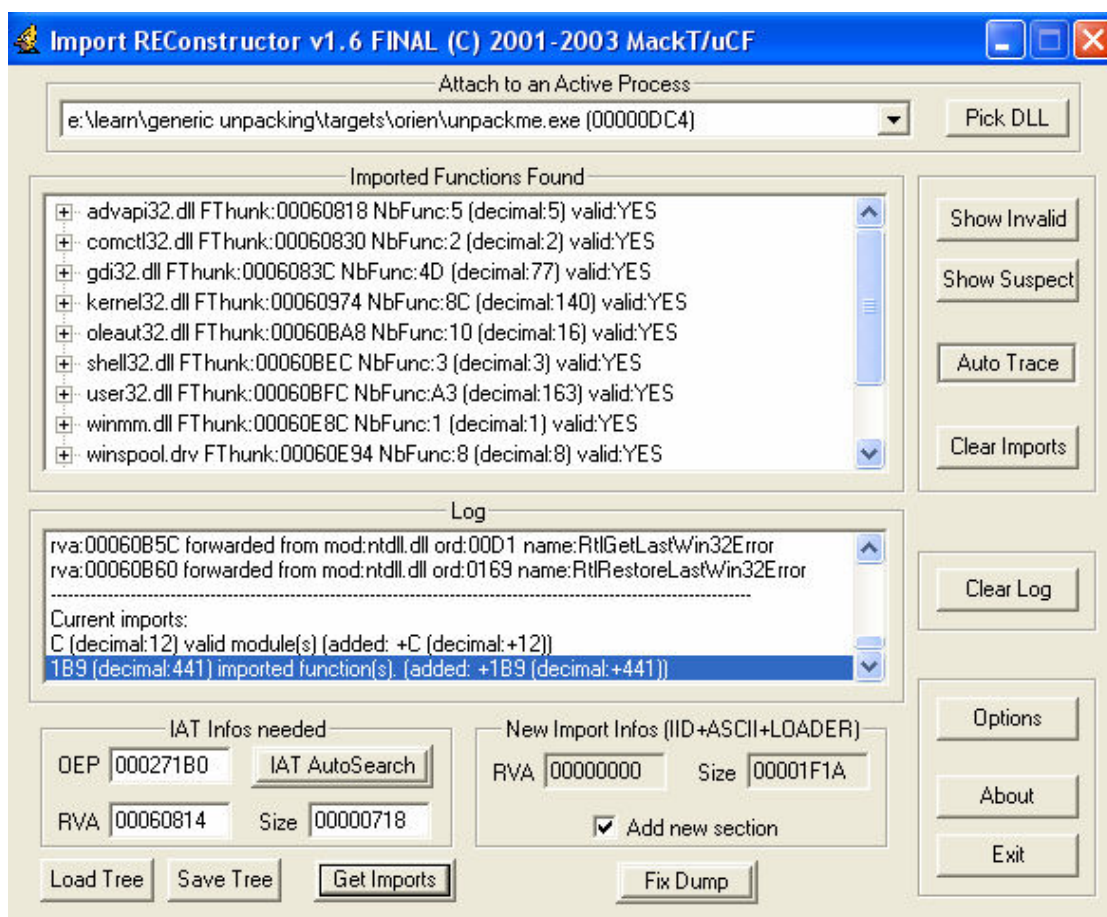
004A375F	74 06	JFE SHORT 004A375C	
004A3761	72 10	JB SHORT 004A3773	
004A3763	EB 04	JMP SHORT 004A3769	
004A3765	08EB	OR EBP,EBX	
004A3767	F8	CLC	
004A3768	1F	POP DS	
004A3769	83FE 00	CMP ESI,0	
004A376C	0F84 10010000	JE 004A3882	
004A3772	EB 01	JMP SHORT 004A3775	
004A3774	7F 3B	JG SHORT 004A37B1	
004A3776	C2 03B3	RET 0B303	
004A3779	E6 1A	OUT 1A,AL	
004A377B	0000	ADD BYTE PTR DS:[EAX],AL	
004A377D	EB 06	JMP SHORT 004A3783	
004A377F	7A 8B	JFE SHORT 004A370C	
004A3781	FE	???	
004A3782	EB 04	JMP SHORT 004A3783	
004A3784	08EB	OR EBP,EBX	
004A3786	F9	STC	
004A3787	1F	POP DS	
004A3788	EB 03	JMP SHORT 004A378D	
004A378A	CD20 9F8B0683	UxDJump 83068B9F	
004A3790	C601 0E	MOV BYTE PTR DS:[ECX],0EB	
004A3793	06	PUSH ES	
004A3794	43	INC EBX	
004A3795	66:B8 83C6	MOV AX,0C683	
004A3799	0083 FFF83FE	ADD BYTE PTR DS:[EBX+FF83FFFF],AL	

این یک پرش بسیار بلند است. به نظر من این پرش تصمیم گیرنده است... یعنی اگر پرش انجام می شود. این کدها اجرا نمی شود و در نتیجه تابع من Invalid نمی شود. پس:

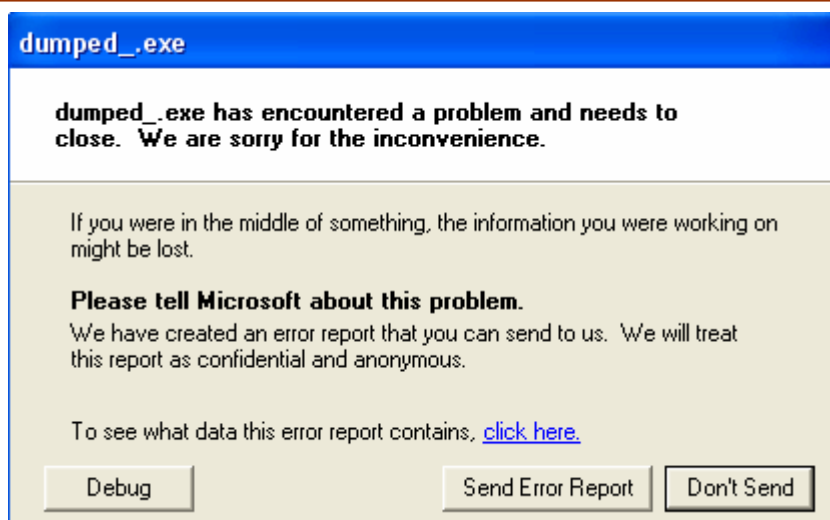
تمامی Breakpoint ها را پاک می کنم و بعد بر روی این پرش یک Hardware Breakpoint on execution می گذارم. (کلیک راست کرده، گزینه Breakpoint و بعد گزینه Hardware on execution می گذارم.) (و بعد برنامه را Restart می کنم. حالا برنامه را با F9 اجرا می کنم تا به پرش مورد نظر برسم:

004A3765	EB 04	JMP SHORT 004A3769
004A3767	0BEB	OR EBP,EBX
004A3768	F8	CLC
004A3769	1F	POP DS
004A376C	83FE 00	CMP ESI,0
004A3772	0F84 10010000	JE 004A3882
004A3774	EB 01	JMP SHORT 004A3775
004A3776	7F 3B	JG SHORT 004A37B1
004A3777	C2 03B3	RET 0B303
004A3779	E6 1A	OUT 1A,AL
004A377B	0000	ADD BYTE PTR DS:[EAX],AL
004A377D	EB 06	JNE SHORT 004A3785
004A377F	7A 8B	JPE SHORT 004A370C
004A3781	FE	???
004A3782	EB 04	JMP SHORT 004A3788
004A3784	0BEB	OR EBP,EBX
004A3786	F9	STC
004A3787	1F	POP DS
004A3788	EB 03	JMP SHORT 004A378D
004A378A	CD20 9F8B0683	UxDJump 83068B9F
004A3790	C601 EB	MOV BYTE PTR DS:[ECX],0EB
004A3793	06	PUSH ES
004A3794	43	INC EBX
004A3795	66:B8 83C6	MOV AX,0C683
004A3799	0083 EEF83EE	ADD BYTE PTR DS:[EBX+EE83FFEE],AL
004A379F	FFEB	JMP FOR EBX
004A37A1	06	PUSH ES
004A37A2	43	INC EBX

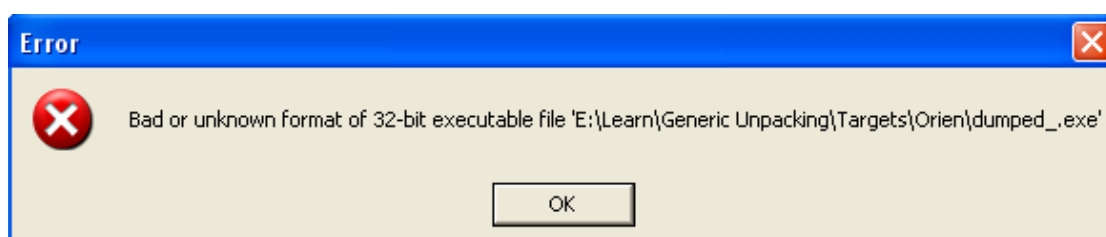
همانطور که می بینید این پرش انجام نمی شود و کدهایی که آن دو تابع ما را Redirect می کنند اجرا می شود. پس باید کاری کنیم که این پرش همواره اجرا شود. (آن را JMP می کنیم). و... بعد برنامه را اجرا می کنیم.
حالا دوباره با ImportREC فایل را باز می کنیم. OEP را می دهیم و گزینه IAT Auto-search را می زنیم و بعد Get Imports



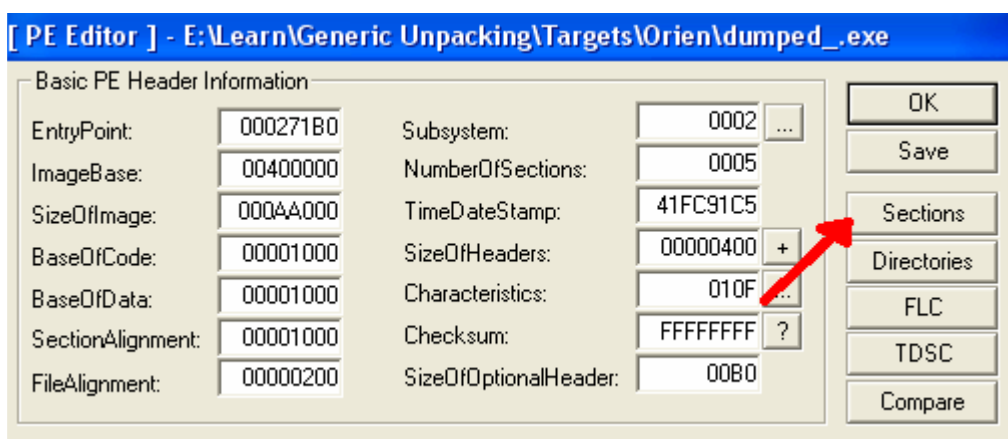
همانطور که می بینید این بار تمامی توابع ما Valid هست...چرا؟! چون آن کدهایی که دو تا از توابع ما را Invalid می کرد، اجرا نشده است.
حالا فایل دامپ شده خود را فیکس کنید. و آن را اجرا کنید:



بهتر است فایل را در OllyDBG باز کنیم تا ببینیم مشکل چیست و در کجا برنامه دچار مشکل می شود؟:



می بینید؟ OllyDBG نمی تواند فایل را درست اجرا کند. حتی با پلاگین OllyAdvanced... چرا؟... چون یکی از مشخصات فایل باید مشکل داشته باشد. فایل را در LordPe باز می کنیم:



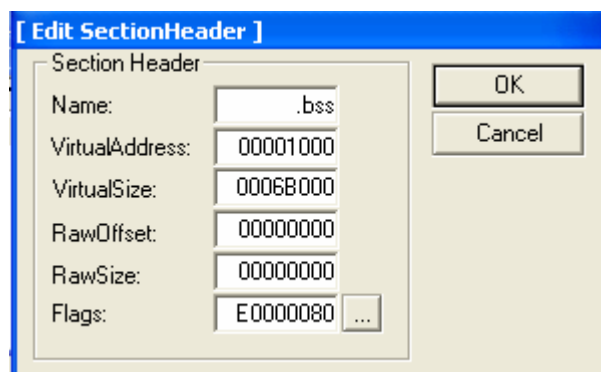
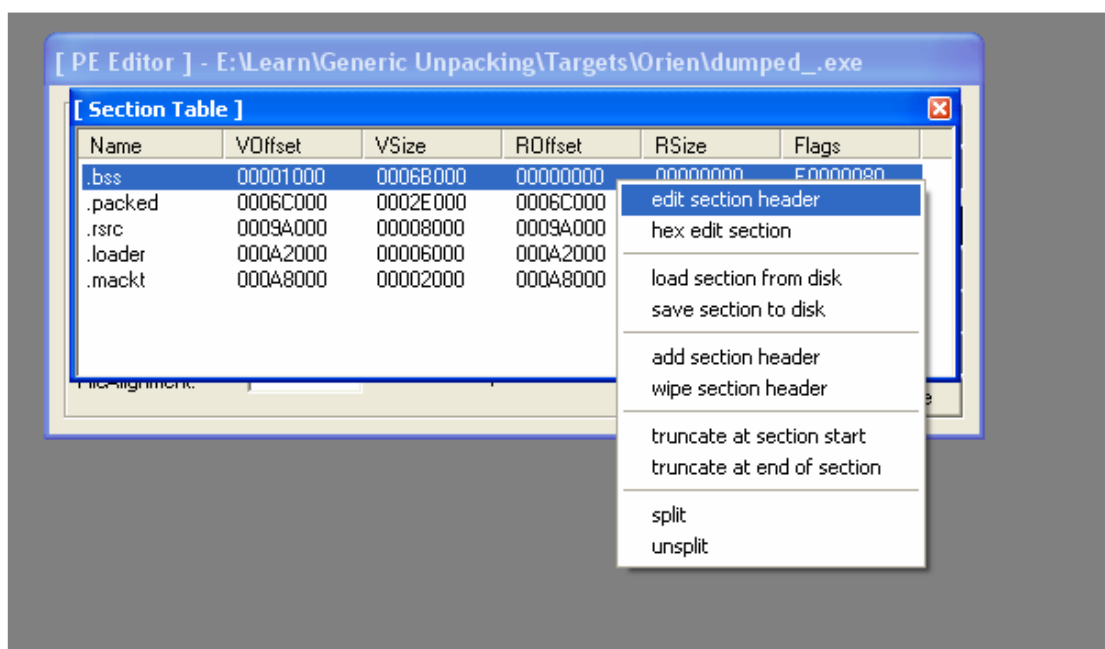
گزینه Sections را بزنید تا تمامی سکشن های فایل آپیک شده نمایش داده شود:

[Section Table]					
Name	VOffset	VSize	ROffset	RSize	Flags
.bss	00001000	00068000	00000000	00000000	E0000080
.packed	0006C000	0002E000	0006C000	0002E000	C0000040
.rsrc	0009A000	00008000	0009A000	00008000	C0000040
.loader	00042000	00006000	00042000	00006000	E0000020
.mact	00048000	00002000	00048000	00002000	E0000060

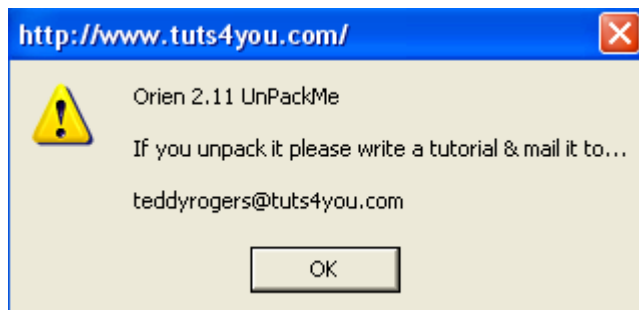
Voffset (مخفف Virtual offset) نشان دهنده آدرسی از حافظه است که وقتی این فایل اجرا می شود این سکشن در این آدرس قرار می گیرد.

ROffset (مخفف RAW Offset) هم نشان دهنده آدرسی است که این سکشن در داخل فایل موجود است
معمولا VOffset با ROffset برابر است. و VSize با RSize

همانطور که در تصویر بالا می بینید در سکشن .bss مقدار ROffset و RSize هر دو برابر صفر می باشد!...چه طور ممکن است...مگر می شود یک سکشن وجود داشته باشد و سایز آن صفر باشد?...پس دلیل اجرا نشدن هم همین است. بر روی این سکشن کلیک راست کرده و گزینه Edit Section header را بزنید:



مقدار Rawoffset را برابر با VirtualAddress کنید(۱۰۰۰) و مقدار Rawsize را هم برابر با Virtual size کنید(6B0000)...حالا کلید OK را بزنید و پنجره Sections را ببندید و گزینه Save را بزنید تا تغییرات ذخیره شود. حالا دوباره فایل را اجرا کنید:



این بار فایل بی هیچ مشکلی اجرا می شود ☺

Scripting in OllyDBG

زبان اسکریپت نویسی در OllyDBG بسیار شبیه به زبان برنامه نویسی اسمبلی است. به همین دلیل نوشتن اسکریپت در این برنامه بسیار راحت است. برای اسکریپت نویسی در OllyDBG لازم است که برخی دستورها را بلد باشید:

- 1. STO. <-- این دستور همان کاری را می کند که کلید F8 انجام می دهد. (Step over)
- 2. STI. <-- این دستور همان کاری را می کند که کلید F7 انجام می دهد. (Step into)
- 3. Run. <-- این دستور همان کاری را می کند که کلید F9 می کنید. (Run Program)
- 4. var. <-- از این دستور برای تعریف یک متغیر که در داخل اسکریپت مورد استفاده قرار می گیرد، استفاده می شود.
- 5. Bphws. <-- با این دستور می توانید Hardware Breakpoint بگذارید.
- 6. CMT. <-- با این دستور می توانید جلوی آدرس خاصی Comment بگذارید.
- 7. an. <-- کدهای یک سکشن را آنالیز می کند. (CTRL + A)
- 8. Msg. <-- از این دستور برای نمایش یک پیغام نمایش داده می شود.
- 9. Ret. <-- با این دستور از اسکریپت خارج می شوید.
- 10. MSGYN. <-- برای نمایش پیغامی که از کاربر سوال می کند، استفاده می شود.
- 11. dpe. <-- با این دستور می توانید از فایل خود دامپ بگیرید.
- 12. log. <-- با این دستور می توانید اطلاعاتی را به پنجره Log در پلاگین ODBGScript اضافه کنید.
- 13. gpa. <-- با این دستور می توانید آدرس یک تابع خاص (MessageBoxA) را بگیرید.
- 14. Rtu. <-- همان کاری را می کند که کلید های ALT + F9 می کنند (Return till user code)
- 15. Findop. <-- این دستور می تواند دستور مشخص شده ای را در کد پیدا کند.
- 16. bp. <-- Breakpoint می گذارد.
- 17. Mov. <-- همان کاری که دستور Mov می کند. البته این دستور در اسکریپت نویسی کاربردهای زیادی دارد. مثلاً برای تغییر یک دستور
- 18. ASM. <-- با این دستور می تواند خط خاصی را تغییر دهید.

البته درباره این دستورات تنها یک توضیح کوچک داده شده، برای به دست آوردن لیست کامل دستورات و توضیحات بیشتر به فایلی که در Attachments به نام Script instructions موجود است مراجعه کنید (صفحات ۸ تا ۲۶). در ضمن برخی از این دستورات ممکن است در ورژن ODBGScript شما پشتیبانی نشود.

ما در اینجا می خواهیم برای مثال قبل یعنی Orien یک اسکریپت بنویسیم که هم بتواند OEP را پیدا کند و هم اینکه از فایل دامپ بگیرد. این هم اسکریپت من:

```
*/
Orien OEP Finder & Dumper
Author=AmirGooran
E-Mail=Amir7687@yahoo.com
/*

Var Path // کنم // یک دستور یک متغیر تعریف می کنم

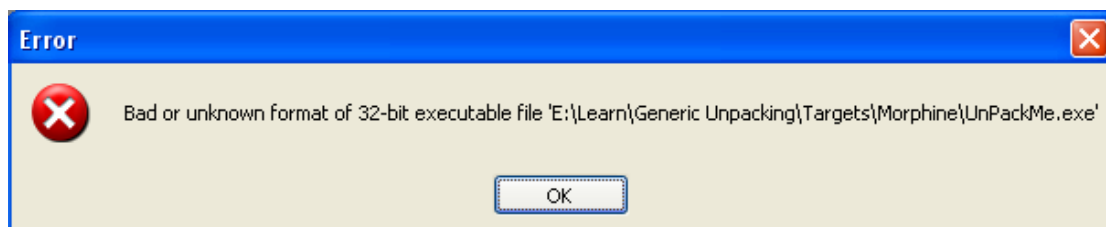
Sto // کلید F8 را می زنم
Sto // کلید F8 را می زنم
Bphws ESP,"r" // می گذارم ESP روی مقدار رجیستر ESP می گذارم
Run // کلید F9 می زنم. همانند کلید F9

یک حلقه ساختم تا ببینم دستور jmp EAX کجاست؟
Loop: //
Sto // کلید F8 را می زنم
Findop eip, #FFE0# // می گردم jmp EAX
Cmp $RESULT, eip // آیا دستور را پیدا کردم؟
Je End // اگر دستور را پیدا کردم به خط End می روم
Jmp loop // اگر این دستور پیدا نشد، حلقه همچنان ادامه پیدا می کند

End: // بعد از پیدا شدن دستور jmp eax اسکریپت به اینجا می آید
Sto // کلید F8 را می زنم
Cmt eip,"This is OEP ;)" // This is oep می نویسم:
Mov Path,"C:\Dumped.exe" // می گیرم C:\dumped.exe را برابر با آدرس
Dpe Path, eip // از فایل دامپ می گیرم
Msg Path // دهم // پیغام به کاربر نمایش می دهم
Ret // اسکریپت در این خط تمام می شود
```

Morphine

فایل مورد نظر را در OllyDBG باز کنید. این پکر یک سکشن جدید در Memory Map می سازد و در این سکشن فایل آپیک شده را قرار می دهد، برای یافتن OEP می توانیم از رجیستر ESP استفاده کنیم.
پنج بار کلید F8 را بزنید تا مقدار رجیستر ESP تغییر کند و بعد بر روی مقدار رجیستر ESP یک Hardware BP on access بگذارید، و کلید F9 را بزنید:



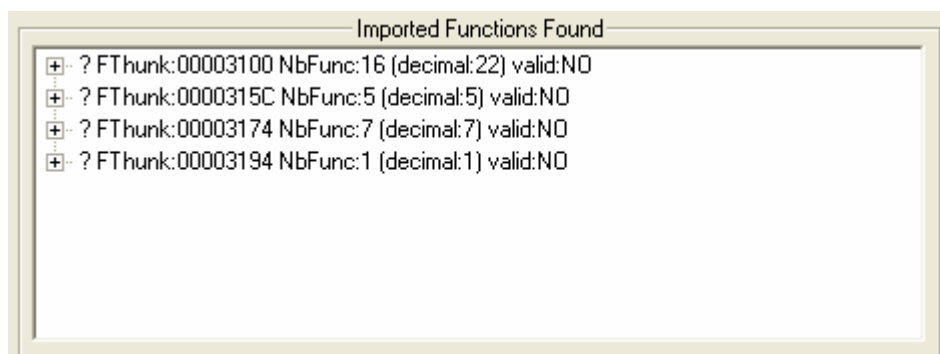
ممکن است سوال کنید چرا اینجا این پیغام داده شده؟... به خاطر اینکه مرفین الان فایل را Decrypt کرده و در یک سکشن جدید آن را ساخته است ☺ کلید OK را بزنید تا این پیغام برود.

Address	Hex dump	Disassembly
111D	5E	POP ESI
111E	5D	POP EBP
111F	83C4 04	ADD ESP, 4
1122	5B	POP EBX
1123	5A	POP EDI
1124	83C4 08	ADD ESP, 8
1127	894C24 04	MOV DWORD PTR SS:[ESP+4], ECX
112B	50	PUSH EAX
112C	C3	RET
112D	55	PUSH EBP
112F	89FE	MOV EBP, ESP

چند بار کلید F8 را بزنید. تا دستور RET اجرا شود و به OEP برسید:

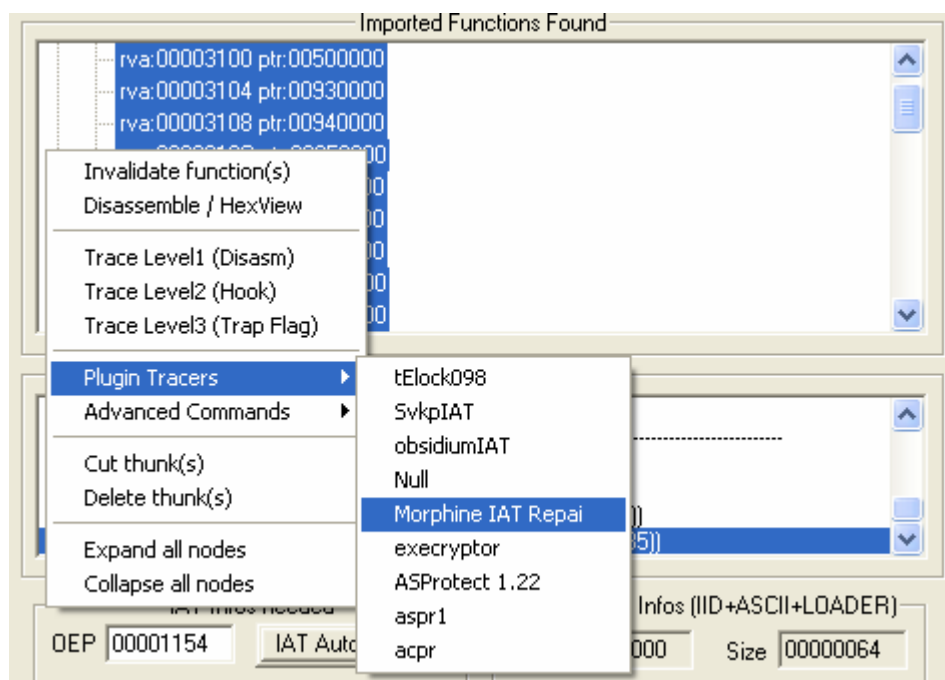
Address	Hex dump	Disassembly
004F1154	55	PUSH EBP
004F1155	8BEC	MOV EBP, ESP
004F1157	53	PUSH EBX
004F1158	56	PUSH ESI
004F1159	57	PUSH EDI
004F115A	BB 00204F00	MOV EBX, 004F2000
004F115F	66:2E:F705 6611	TEST WORD PTR CS:[4F1566], 4
004F1169	75 05	JNZ SHORT 004F1170
004F116B	E9 14040000	JMP 004F1534
004F1170	E9 19020000	JMP 004F138E
004F1175	FF15 40314F00	CALL DWORD PTR DS:[4F3140]
004F117B	83F8 FF	CMPEB EAX, -1
004F117E	F9	STC
004F117F	74 4D	JE SHORT 004F11CE
004F1181	8983 8C020000	MOV DWORD PTR DS:[EBX+28C], EAX
004F1187	C783 94020000	MOV DWORD PTR DS:[EBX+294], 0
004F1191	C783 90020000	MOV DWORD PTR DS:[EBX+290], 0
004F119B	E8 81020000	CALL 004F1421
004F11A0	72 2C	JB SHORT 004F11CE
004F11A2	8983 90020000	MOV DWORD PTR DS:[EBX+290], EAX
004F11A8	C740 58 08000000	MOV DWORD PTR DS:[EAX+58], 8
004F11AF	E8 0E000000	CALL 004F11C2
004F11B4	4D	DEC EBP

حالا از فایل دامپ بگیرید. (برای دامپ گرفتن از پلاگین OllyDump استفاده نکنید).
و ImportREC را باز کنید (OEP بر حسب RVA برابر ۱۱۵۴ است.):

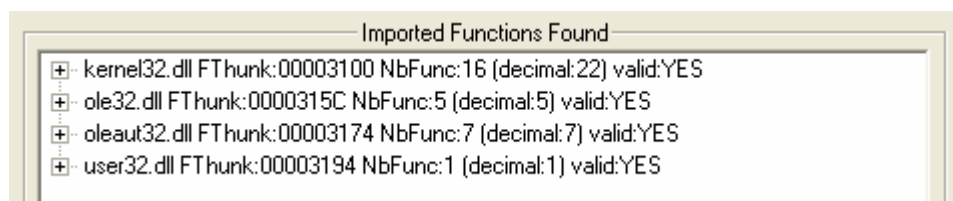


تمامی توابع Redirect شده اند. برای درست کردن آنها، می توانیم از پلاگین استفاده کنیم. یکی از قابلیت های خوب ImportREC همین است که می توان به آن پلاگین اضافه کرد. ☺ گزینه Show Invalid را بزنید و کلیک راست کرده و گزینه Plugin tracers و

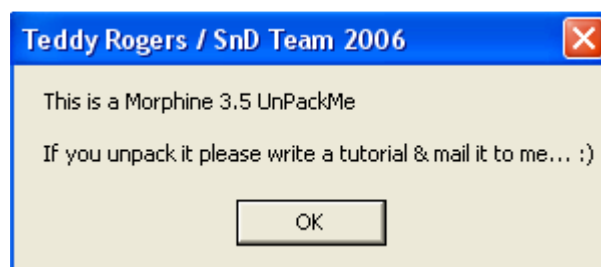
سپس گزینه Morphine IAT Repai را بزنید. (البته برای درست کردن توابع راه های دیگری هم وجود دارد. مثلا این کار را دستی انجام بدهیم ☹)



اگر این پلاگین را نداشته باشید، چنین گزینه ای وجود نخواهد داشت:



حالا فایل دامپ را فیکس کنید:



SPlayer

برنامه را در OllyDBG باز کنید:

Address	Hex dump	Disassembly
00425FD0 <ModuleEntryPoint>	8D40 00	LEA EAX, DWORD PTR DS:[EAX]
00425FD0	B9 CC5F4200	MOV ECX, 00425FCC
00425FE2	6A 0E	PUSH 0E
00425FE4	58	POP EAX
00425FE5	C0C01 04	ROR BYTE PTR DS:[ECX+EAX], 4
00425FE9	48	DEC EAX
00425FEA	75 F9	JNZ SHORT 00425FE5
00425FEC	66:13F0	ADC SI, AX
00425FEF	91	XCHG EAX, ECX
00425FF0	3BD9	CMPL EBX, ECX
00425FF2	81FA B0D0FB6C	CMPL EDX, 6CFB0D0B
00425FF8	EB D2	JMP SHORT 00425FCC
00425FFA	0000	ADD BYTE PTR DS:[EAX], AL
00425FFC	0000	ADD BYTE PTR DS:[EAX], AL
00425FFE	0000	ADD BYTE PTR DS:[EAX], AL
00426000	0000	ADD BYTE PTR DS:[EAX], AL
00426002	0000	ADD BYTE PTR DS:[EAX], AL
00426004	0000	ADD BYTE PTR DS:[EAX], AL
00426006	0000	ADD BYTE PTR DS:[EAX], AL
00426008	0000	ADD BYTE PTR DS:[EAX], AL
0042600A	0000	ADD BYTE PTR DS:[EAX], AL
0042600C	0000	ADD BYTE PTR DS:[EAX], AL
0042600E	0000	ADD BYTE PTR DS:[EAX], AL
00426010	0000	ADD BYTE PTR DS:[EAX], AL
00426012	0000	ADD BYTE PTR DS:[EAX], AL

سه بار کلید F8 را بزنید تا مقدار رجیستر ESP تغییر کند و بعد هم روی مقدار رجیستر ESP یک Hardware breakpoint on access می گذاریم، و بعد کلید F9 را می زنم من در اینجا متوقف شدم:

00425FD1	29C8	SUB EAX,ECX
00425FD3	300C08	XOR BYTE PTR DS:[EAX+ECX],CL
00425FD6	E2 FB	LOOPD SHORT 00425FD3
00425FD8	50	PUSH EAX
00425FD9	C3	RET
00425FDA <ModuleEntryPoint>	D840 00	FADD DWORD PTR DS:[EAX]
00425FDD	B9 CC5F4200	MOV ECX,00425FCC
00425FE2	6A 0E	PUSH 0E
00425FE4	58	POP EAX
00425FE5	C0C001 04	ROR BYTE PTR DS:[ECX+EAX],4
00425FE9	48	DEC EAX
00425FEA	75 F9	JNZ SHORT 00425FE5
00425FEC	66:13F0	ADC SI,AX
00425FEF	91	XCHG EAX,ECX
00425FF0	3BD9	CMP EBX,ECX
00425FF2	81FA B0D0FB6C	CMP EDX,6CFB00B0
00425FF8	EB D2	JMP SHORT 00425FCC
00425FFA	0000	ADD BYTE PTR DS:[EAX],AL
00425FFC	0000	ADD BYTE PTR DS:[EAX],AL

چیز جالبی توی این خط نیست، یکبار دیگر کلید F9 را می زنم:

00401000	B8 E8974700	MOV EAX,004797E8
00401005	50	PUSH EAX
00401006	64:FF35 00000000	PUSH DWORD PTR FS:[0]
0040100D	64:8925 00000000	MOV DWORD PTR FS:[0],ESP
00401014	33C0	XOR EAX,EAX
00401016	8908	MOV DWORD PTR DS:[EAX],ECX
00401018	50	PUSH EAX
00401019	45	INC EBX
0040101A	43	INC EBX
0040101B	3200	XOR AL,BYTE PTR DS:[EAX]
0040101D	8EE0	MOV FS,AX
0040101F	5E	POP ESI
00401020	9D	POPF
00401021	B9 4BF7FB64	MOV ECX,64FBF74B
00401026	A2 F25B3F2	MOV BYTE PTR DS:[F2B35F2],AL
0040102B	BC 0012FF73	MOV ESP,73FF1280
00401030	93	XCHG EAX,EBX
00401031	ED	IN EAX,DX
00401032	ED	IN EAX,DX

اینجا هم چیزی نیست، یکبار دیگر F9 را می زنم:

7C937804	57	PUSH EDI
7C937805	3B5D F8	CMP EBX,DWORD PTR SS:[EBP-8]
7C937806	^ 0F82 1032FFFF	JB 7C92AA2B
7C93780E	8D43 08	LEA EAX,DWORD PTR DS:[EBX+8]
7C937811	3B45 F4	CMP EAX,DWORD PTR SS:[EBP-C]
7C937814	^ 0F87 1132FFFF	JA 7C92AA2B
7C93781A	F6C3 03	TEST BL,3
7C93781D	^ 0F85 0832FFFF	JNZ 7C92AA2B
7C937823	8B43 04	MOV EAX,DWORD PTR DS:[EBX+4]
7C937825	3B45 F8	CMP EAX,DWORD PTR SS:[EBP-8]
7C937829	72 09	JB SHORT 7C937834
7C93782B	3B45 F4	CMP EAX,DWORD PTR SS:[EBP-C]
7C93782E	^ 0F82 F731FFFF	JB 7C92AA2B
7C937834	50	PUSH EAX
7C937835	E8 67000000	CALL 7C9378A1
7C93783A	84C0	TEST AL,AL
7C93783C	^ 0F84 E931FFFF	JE 7C92AA2B
7C937842	F605 5AC3977C 80	TEST BYTE PTR DS:[7C97C35A],80
7C937849	^ 0F85 20720100	JNZ 7C94EA6F
7C93784F	FF73 04	PUSH DWORD PTR DS:[EBX+4]
7C937852	8D45 EC	LEA EAX,DWORD PTR SS:[EBP-14]
7C937855	50	PUSH EAX
7C937856	FF75 0C	PUSH DWORD PTR SS:[EBP+C]
7C937859	53	PUSH EBX
7C93785B	56	PUSH ESI
7C93785D	E8 F3BEFCFF	CALL 7C903753

اینجا هم خبری نیست ☹ یکبار دیگر F9 را می زنم.

7C937854	50	PUSH EAX
7C937855	E8 67000000	CALL 7C9378A1
7C93785A	84C0	TEST AL,AL
7C93783C	^ 0F84 E931FFFF	JE 7C92AA2B
7C937842	F605 5AC3977C 80	TEST BYTE PTR DS:[7C97C35A],80
7C937849	^ 0F85 20720100	JNZ 7C94EA6F
7C93784F	FF73 04	PUSH DWORD PTR DS:[EBX+4]
7C937852	8D45 EC	LEA EAX,DWORD PTR SS:[EBP-14]
7C937855	50	PUSH EAX
7C937856	FF75 0C	PUSH DWORD PTR SS:[EBP+C]
7C937859	53	PUSH EBX
7C93785B	56	PUSH ESI
7C93785D	E8 F3BEFCFF	CALL 7C903753
7C937860	F605 5AC3977C 80	TEST BYTE PTR DS:[7C97C35A],80
7C937867	8BF8	MOV EDI,EAX
7C937869	^ 0F85 16720100	JNZ 7C94EA85
7C93786F	395D 08	CMP DWORD PTR SS:[EBP+8],EBX
7C937872	^ 0F84 1B720100	JE 7C94EA93
7C937878	8BC7	MOV EAX,EDI
7C93787A	33C9	XOR ECX,ECX
7C93787C	2BC1	SUB EAX,ECX
7C93787E	^ 0F85 8631FFFF	JNZ 7C92AA0A
7C937884	F646 04 01	TEST BYTE PTR DS:[ESI+4],1
7C937888	^ 0F85 4F720100	JNZ 7C94EADD
7C93788E	C645 FF 01	MOV BYTE PTR SS:[EBP-11],1

یکبار دیگر F9:

004795AC	64:8F05 00000000	POP DWORD PTR FS:[0]
004795B3	83C4 04	ADD ESP,4
004795B6	55	PUSH EBP
004795B7	53	PUSH EBX
004795B8	51	PUSH ECX
004795B9	57	PUSH EDI
004795BA	56	PUSH ESI
004795BB	52	PUSH EDX
004795BC	8D98 F8108D00	LEA EBX,DWORD PTR DS:[EAX+8D10F8]
004795C2	8B53 18	MOV EDX,DWORD PTR DS:[EBX+18]
004795C5	52	PUSH EDX
004795C6	8BE8	MOV EBP,EAX
004795C8	6A 40	PUSH 40
004795CA	68 00100000	PUSH 1000
004795CF	FF73 04	PUSH DWORD PTR DS:[EBX+4]
004795D2	6A 00	PUSH 0
004795D4	8B4B 10	MOV ECX,DWORD PTR DS:[EBX+10]
004795D7	03CA	ADD ECX,EDX
004795D9	8B01	MOV EAX,DWORD PTR DS:[ECX]
004795DB	FFD0	CALL EAX
004795DD	5A	POP EDX
004795DE	8BF8	MOV EDI,EAX
004795E0	50	PUSH EAX

اینجا هم چیزی نیست.یکبار دیگر:

0047964B	57	PUSH EDI	
0047964C	5B	POP ECX	
0047964D	5B	POP EBX	
0047964E	5D	POP EBP	
0047964F	- FFE0	JMP EAX	SPLayer,00455F1E
00479651	1E	PUSH DS	
00479652	5F	POP EDI	
00479653	45	INC EBP	
00479654	0000	ADD BYTE PTR DS:[EAX],AL	
00479656	0C 0A	OR AL,0A	
00479658	^ E2 FB	LOOPD SHORT 00479655	
0047965A	52	PUSH EDX	
0047965B	C3	RET	
0047965C	8D0B	LEA ECX,DWORD PTR DS:[EBX]	
0047965E	29CA	SUB EDX,ECX	
00479660	D2040A	ROL BYTE PTR DS:[EDX+ECX],CL	
00479663	^ E2 FB	LOOPD SHORT 00479660	
00479665	52	PUSH EDX	
00479666	C3	RET	
00479667	BC 0B29CA00	MOV ESP,0CA290B	
0047966C	0C 0A	OR AL,0A	
0047966E	^ E2 FB	LOOPD SHORT 0047966B	
00479670	52	PUSH EDX	
00479671	C3	RET	
00479672	BC 0B29CA00	MOV ESP,0CA290B	

آها...اینجا همان جایی بود که می خواستم،یک پرش که درست به OEP می رود.بعد از یافتن OEP دیگر مشکلی نیست.دامپ می گیریم و فیکس می کنیم.

PeX

فایل مورد نظر را در OllyDBG باز کنید. برای یافتن OEP می توانیم از اسکریپت استفاده کنیم. در منوی PlugIns گزینه ODBGScript و سپس گزینه Run Script بزنید. اسکریپت مورد نظر (در Attachments موجود است) را انتخاب کنید:

Address	Disassembly	Comment
00401184	55	PUSH EBP
00401185	8BEC	MOV EBP, ESP
00401187	53	PUSH EBX
00401188	56	PUSH ESI
00401189	57	PUSH EDI
0040118A	BB 00204000	MOV EBX, UnPackMe.00402000
0040118F	66:2E:F705 2615	TEST WORD PTR CS:[401596], 4
00401199	75 05	JNZ SHORT UnPackMe.004011A0
0040119B	E9 14040000	JMP UnPackMe.004015B4
004011A0	E9 19020000	JMP UnPackMe.004013BE
004011A5	FF15 40314000	CALL DWORD PTR DS:[403140]
004011AB	83F8 FF	CMF EAX, -1
004011AE	F9	STC
004011AF	74 4D	JE SHORT UnPackMe.004011FE
004011B1	8983 8C020000	MOV DWORD PTR DS:[EBX+28C], EAX
004011B7	C783 94020000 01	MOV DWORD PTR DS:[EBX+294], 0
004011C1	C783 90020000 01	MOV DWORD PTR DS:[EBX+290], 0
004011C8	E8 81020000	CALL UnPackMe.00401451
004011D0	72 2C	JB SHORT UnPackMe.004011FE
004011D2	8983 90020000	MOV DWORD PTR DS:[EBX+290], EAX
004011D8	C740 58 08000000	MOV DWORD PTR DS:[EAX+58], 8
004011DF	E8 0E000000	CALL UnPackMe.004011F2
004011E4	4D	DEC EBP
004011E5	53	PUSH EBX
004011E6	2053 61	AND BYTE PTR DS:[EBX+61], DL
004011E9	6E	OUTS DX, BYTE PTR ES:[EDI]

همانطور که دیدید اسکریپت توانسته OEP را با موفقیت به دست بیاورد. ☺

راه حل دیگر:

راه حل دیگر این است که به صورت دستی OEP را با یک روش مخصوص PeX پیدا کنیم. کاری که این اسکریپت می کند را ما به صورت دستی انجام می دهیم:

Address	Disassembly	Comment
00406000	E9 F5000000	JMP UnPackMe.004060FA
00406005	0D 0AC4C4C4	OR EAX, C4C4C40A
0040600A	C4C4	LES EAX, ESP
0040600C	C4C4	LES EAX, ESP
0040600E	C4C4	LES EAX, ESP
00406010	C4C4	LES EAX, ESP
00406012	C4C4	LES EAX, ESP
00406014	C4C4	LES EAX, ESP
00406016	C4C4	LES EAX, ESP
00406018	C4C4	LES EAX, ESP
0040601A	C4C4	LES EAX, ESP
0040601C	C4C4	LES EAX, ESP
0040601E	C4C4	LES EAX, ESP

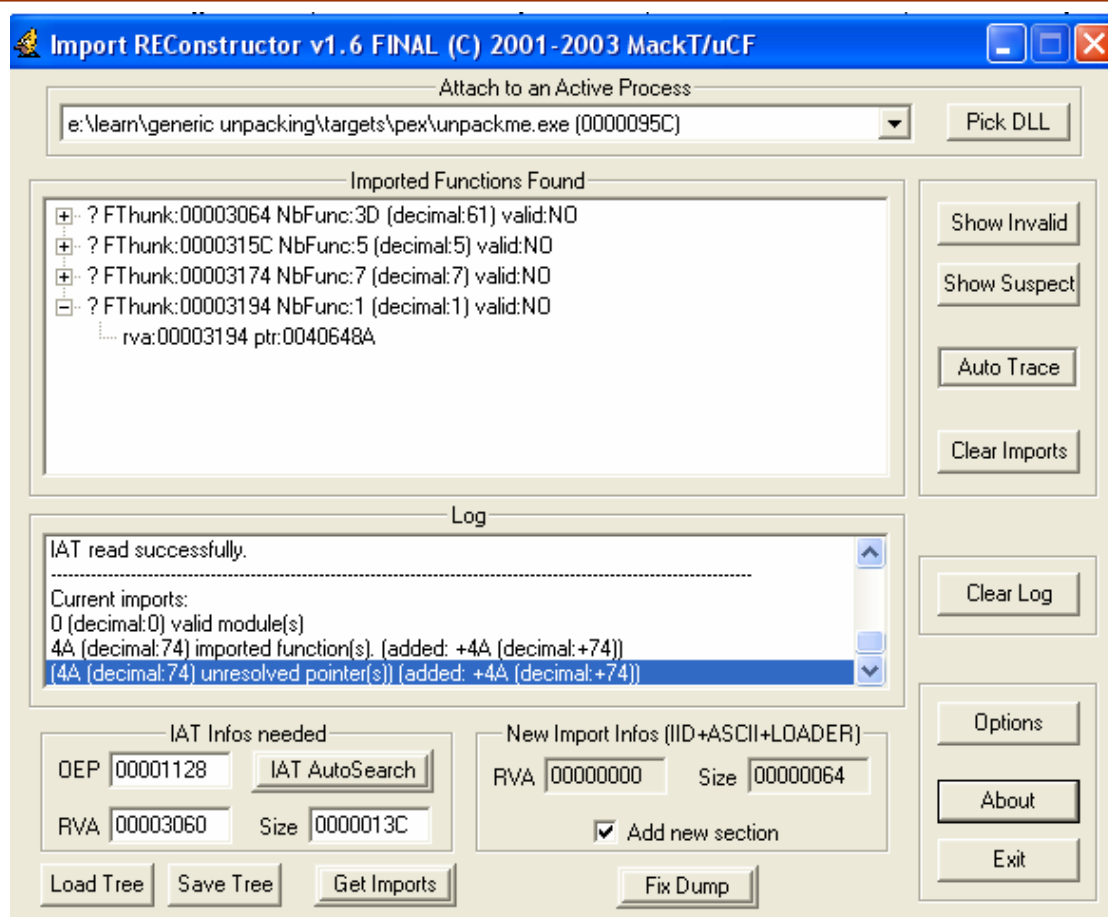
Entry Point ما خط 406000 هزار است. این مقدار را به اضافه یک می کنیم که می شود: 406001
 حالا کلید CTRL + G را می زنیم و به این آدرس می رویم: 406001
 حالا بر روی این آدرس یک Hardware breakpoint on execution می گذاریم. (کلیک راست کرده گزینه Breakpoint و سپس گزینه Hardware on execution را می زنیم). و بعد برنامه را با F9 اجرا می کنیم. برنامه در اینجا متوقف می شود:

Address	Disassembly	Comment
00406001	FF15 81634000 CALL DWORD PTR DS:[<&KERNEL32.VirtualFree	
00406007	E8 01000000 CALL UnPackMe.00406000	
0040600C	-E9 83C4042B JMP 2B452494	
00406011	SHL BYTE PTR DS:[EDI+ECX*4],83	Shift constant out of range 1..31
00406016	LES ECX,FWORD PTR DS:[EAX+EBP*8]	Modification of segment register
00406019	ADD DWORD PTR DS:[EAX],EAX	
0040601B	ADD BYTE PTR DS:[EAX],AL	
0040601D	???	Unknown command
0040601E	POP EAX	
0040601F	POPAD	
00406020	E8 15000000 CALL UnPackMe.0040603A	
00406025	E8 E80F0000 CALL UnPackMe.00407012	
0040602A	ADD BYTE PTR DS:[EDX+9E8],BL	
00406030	ADD CL,CH	
00406032	PUSH UnPackMe.00401183	
00406037	JMP SHORT UnPackMe.0040603A	
00406039	???	Unknown command
0040603A	POP EAX	
0040603B	INC EAX	
0040603C	PUSH EAX	
0040603D	RETN	
0040603E	LES EAX,ESP	Illegal use of register
00406040	C4C4	Illegal use of register
00406042	C4C4	Illegal use of register
00406044	C4C4	Illegal use of register
00406046	C4C4	Illegal use of register
00406048	C4C4	Illegal use of register
0040604A	C4C4	Illegal use of register
0040604C	C4C4	Illegal use of register
0040604E	C4C4	Illegal use of register
00406050	C4C4	Illegal use of register
00406052	C4C4	Illegal use of register
00406054	C4C4	Illegal use of register
00406056	00 0A205065 OR EAX,6550200A	
0040605B	58	
0040605C	2028	
0040605E	6329	
00406060	2062 79	
00406063	2062 61	
00406066	72 74	
00406068	5E	
00406069	43	

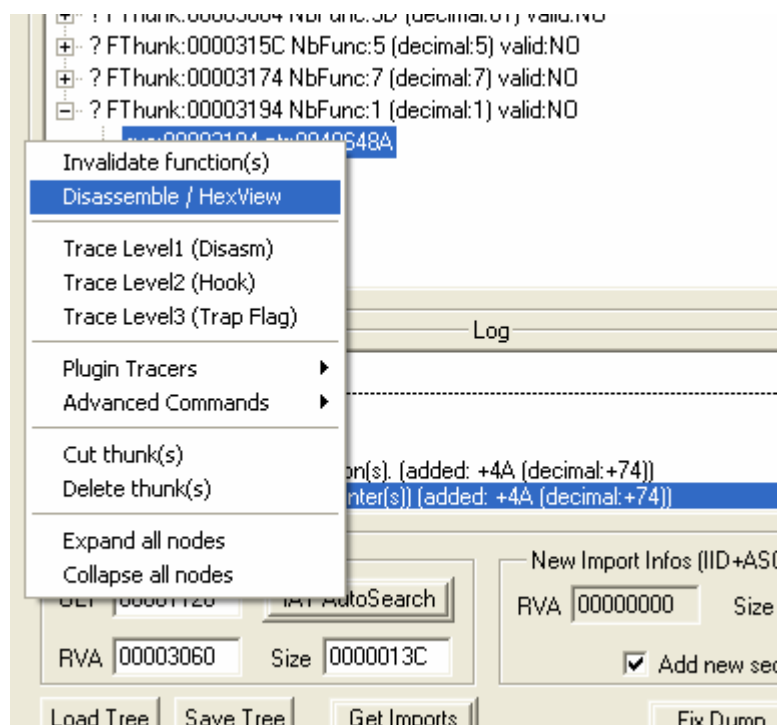
چند خط پایین تر یک دستور JMP (با فلش نشان داده شده) وجود دارد که بر روی آن Breakpoint می گذاریم و برنامه را با F9 اجرا می کنیم. برنامه ما در جایی که Breakpoint گذاشتیم متوقف می شود. حالا اگر چهار بار کلید F8 را بزنیم به OEP می رسیم:

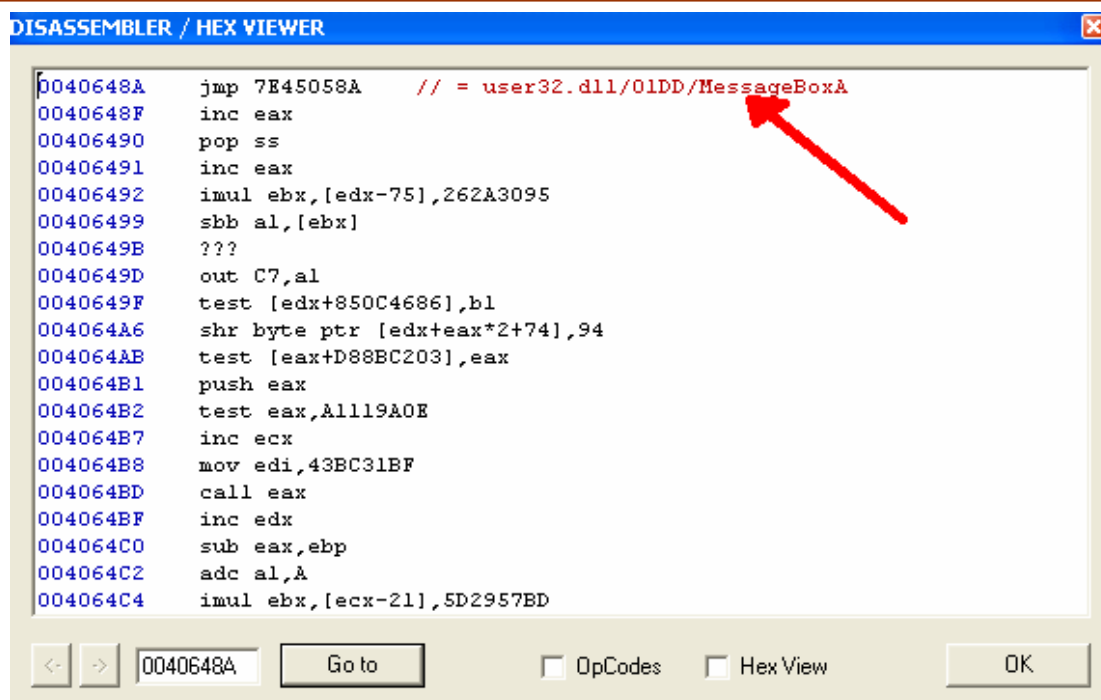
Address	Disassembly	Comment
00401184	55	OEP
00401185	8B	DB 8B
00401186	EC	DB EC
00401187	53	DB 53
00401188	56	DB 56
00401189	57	DB 57
0040118A	BB	DB BB
0040118B	00 00204000	DD UnPackMe.00402000
0040118F	66	DB 66
00401190	2E	DB 2E
00401191	F7	DB F7
00401192	05	DB 05
00401193	00 36154000	DD UnPackMe.00401596
00401197	04	DB 04
00401198	00	DB 00
00401199	75	DB 75
0040119A	05	DB 05
0040119B	E9	DB E9
0040119C	14	DB 14
0040119D	04	DB 04
0040119E	00	DB 00
0040119F	00	DB 00
004011A0	E9	DB E9
004011A1	19	DB 19
004011A2	02	DB 02
004011A3	00	DB 00
004011A4	00	DB 00
004011A5	FF	DB FF

حالا از فایل دامپ بگیری. و ImportREC را اجرا کنی. OEP را بر حسب RVA را به برنامه بدهی (من در تصاویر پایین OEP را اشتباه وارد کردم، باید وارد می کردم 1184... البته بعدا مشکل رو حل کردم، ولی وقتی این تصاویر رو گرفتم، OEP را اشتباه وارد کردم. شما اشتباه منو تکرار نکنید.) و ...:

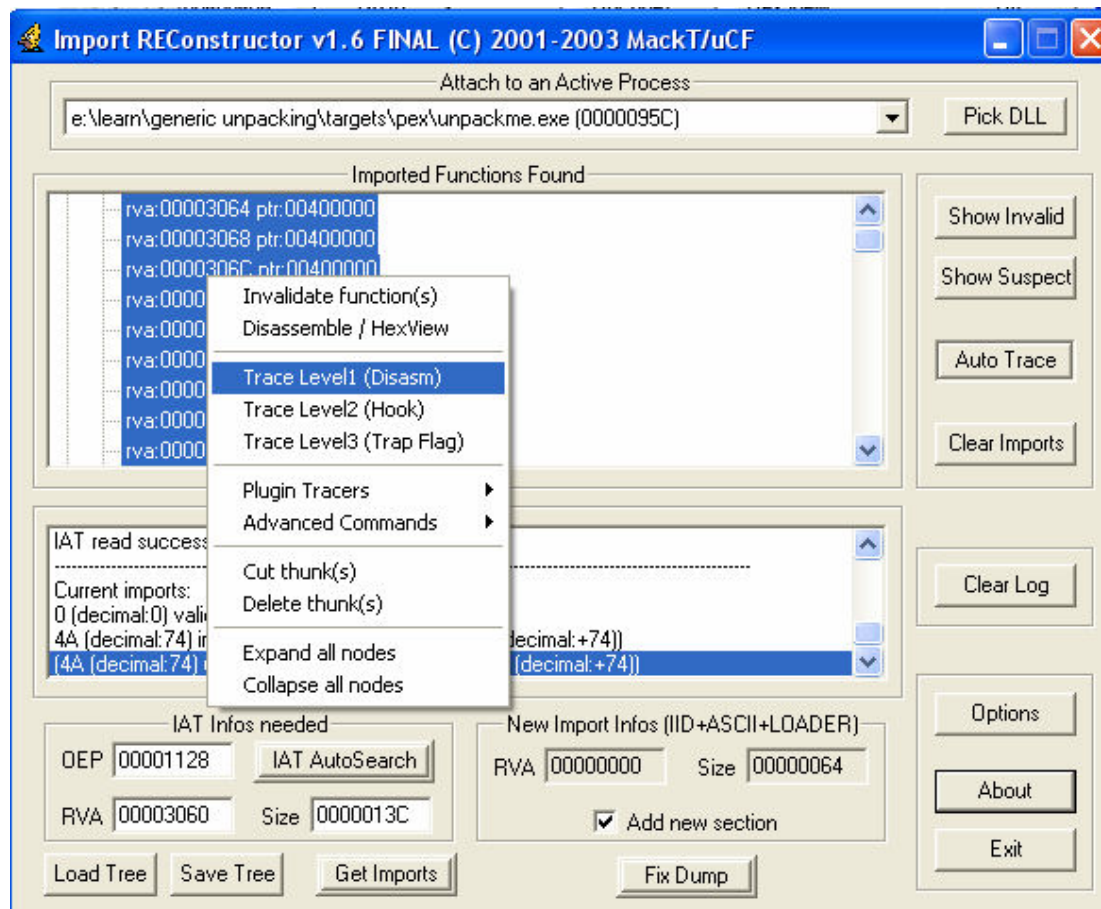


همانطور که می بینید تمامی توابع ما Invalid هستند. برخی از آنها واقعا Invalid هستند. اما برخی دیگر Redirect شده اند. همانند تصویر زیر بر روی آخرین تابع کلیک راست کرده و گزینه Disassemble/Hex View را بزنید:

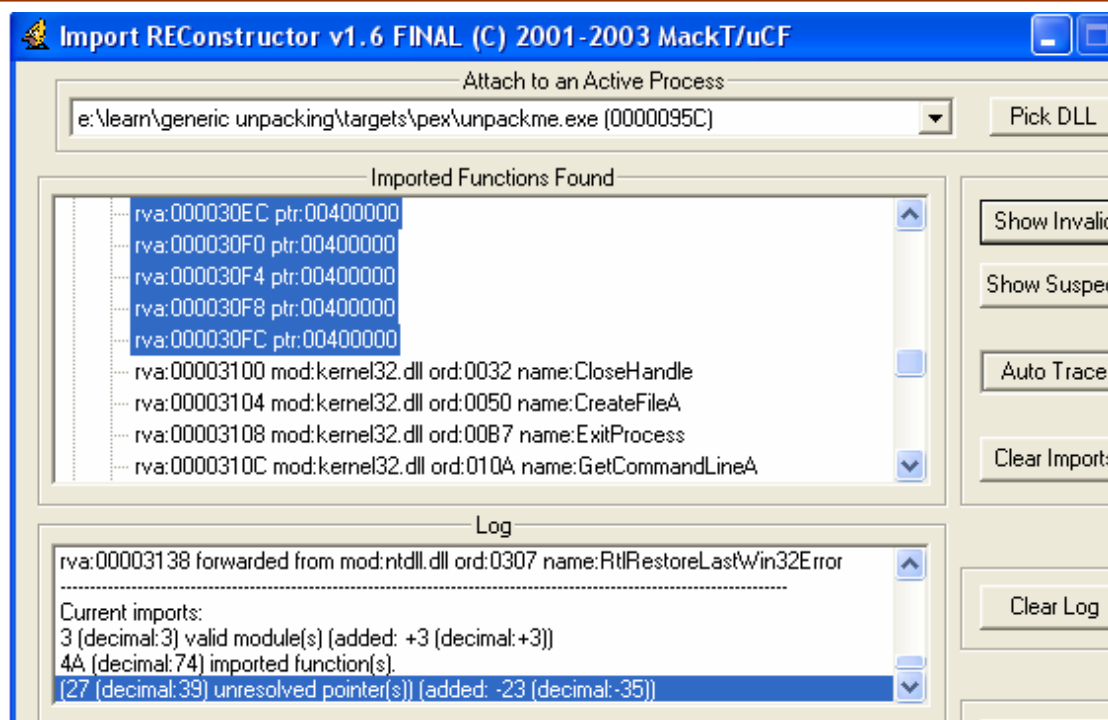




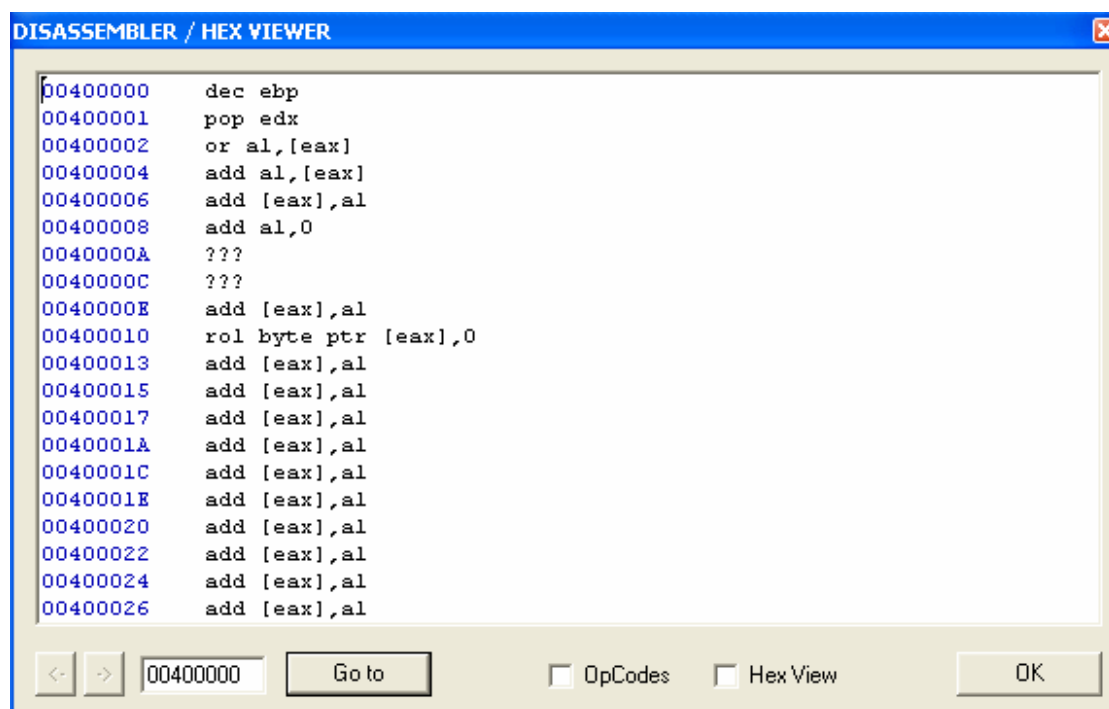
همانطور که می بینید این تابع ما به MessageBoxA می رود. پس این تابع Redirect شده است. خوب حالا چه طور توابع Redirect شده را فیکس کنیم؟
ما می توانیم از قابلیت ImportREC که می تواند توابعی که به سادگی Redirect شده اند را درست کند، استفاده کنیم. برای این کار گزینه Show Invalid را بزنید تا تمامی توابع مشخص شود، حالا همانند تصویر کلیک راست کرده و گزینه (disasm) Trace level 1 را Trace level 1 (disasm) را بزنید:



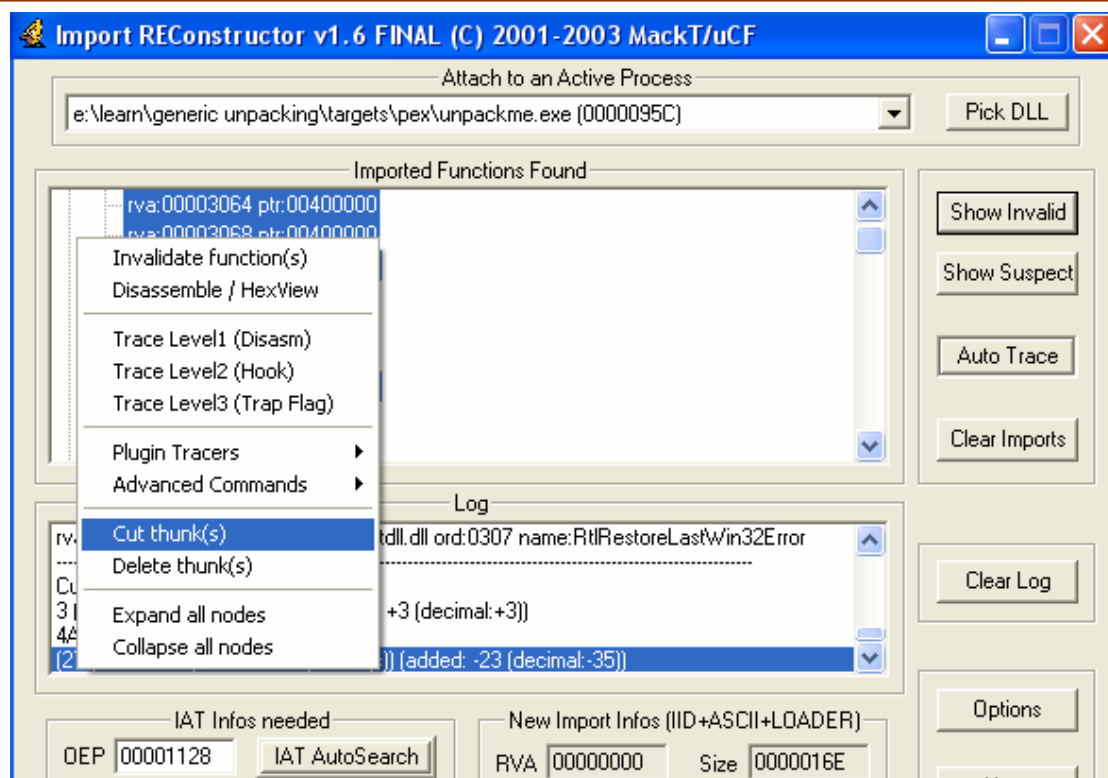
حالا دوباره گزینه Show Invalid را بزنید:



می بینید قسمتی از توابع ما Valid شده...توابع باقیمانده توابع بیخود هستند و باید پاک شوند...از کجا فهمیدم؟...خوب، بر روی یکی از توابع Invalid کلیک راست کرده و گزینه Disassemble / Hex View را بزنید:



می بینید؟...کدهای این قسمت هیچ معنای خاصی ندارد...پس این تابع نمی تواند یک تابع واقعی باشد. پس می توانیم آنها را از بین ببریم، برای این کار کلیک راست کرده و گزینه Cut Thunk را می زنیم:



بعد از این هم فایل دامپ شده خودتان را فیکس می کنید. فایل دامپ شده بی هیچ مشکلی اجرا می شود.

PeCompact

فایل مورد نظر را در OllyDBG باز کنید، برای یافتن OEP می توانیم از رجیستر ESP استفاده کنیم. دو بار کلید F8 را بزنید تا مقدار رجیستر ESP تغییر کند. بعد بر روی آن یک Hardware BP on access بگذارید و کلید F9 را بزنید:

Address	Hex dump	Disassembly
7C937826	3B45 F8	CMP EAX, DWORD PTR SS:[EBP-8]
7C937829	72 09	JB SHORT 7C937834
7C93782B	3B45 F4	CMP EAX, DWORD PTR SS:[EBP-C]
7C93782E	0F82 F731FFFF	JB 7C92AA2B
7C937834	50	PUSH EAX
7C937835	E8 67000000	CALL 7C9378A1
7C93783A	84C0	TEST AL, AL
7C93783C	0F84 E931FFFF	JE 7C92AA2B
7C937842	F605 5AC3977C	TEST BYTE PTR DS:[7C97C35A], 80
7C937849	0F85 20720100	JNZ 7C94EA6F
7C93784F	FF73 04	PUSH DWORD PTR DS:[EBX+4]
7C937852	8D45 EC	LEA EAX, DWORD PTR SS:[EBP-14]
7C937855	50	PUSH EAX
7C937856	FF75 0C	PUSH DWORD PTR SS:[EBP+C]
7C937859	53	PUSH EBX
7C93785A	56	PUSH ESI
7C93785B	E8 F3BEFCFF	CALL 7C903753
7C937860	F605 5AC3977C	TEST BYTE PTR DS:[7C97C35A], 80

الان در فایل ntdll.dll هستیم، یکبار دیگر کلید F9 را بزنید:

7C937842	F605 5AC3977C	TEST BYTE PTR DS:[7C97C35A], 80
7C937849	0F85 20720100	JNZ 7C94EA6F
7C93784F	FF73 04	PUSH DWORD PTR DS:[EBX+4]
7C937852	8D45 EC	LEA EAX, DWORD PTR SS:[EBP-14]
7C937855	50	PUSH EAX
7C937856	FF75 0C	PUSH DWORD PTR SS:[EBP+C]
7C937859	53	PUSH EBX
7C93785A	56	PUSH ESI
7C93785B	E8 F3BEFCFF	CALL 7C903753
7C937860	F605 5AC3977C	TEST BYTE PTR DS:[7C97C35A], 80
7C937867	8BF8	MOV EDI, EAX
7C937869	0F85 16720100	JNZ 7C94EA85
7C93786F	395D 08	CMP DWORD PTR SS:[EBP+8], EBX
7C937872	0F84 1B720100	JE 7C94EA93
7C937878	8BC7	MOV EAX, EDI
7C93787A	33C9	XOR ECX, ECX
7C93787C	2BC1	SUB EAX, ECX
7C93787E	0F85 8631FFFF	JNZ 7C92AA0A
7C937884	F646 04 01	TEST BYTE PTR DS:[ESI+4], 1

هنوزم همانجا هستیم، یک F9 دیگر:

Address	Hex dump	Disassembly
00473B9B	53	PUSH EBX
00473B9C	51	PUSH ECX
00473B9D	57	PUSH EDI
00473B9E	56	PUSH ESI
00473B9F	52	PUSH EDX
00473BA0	8D98 57123C00	LEA EBX, DWORD PTR DS:[EAX+3C1257]
00473BA6	8B53 18	MOV EDX, DWORD PTR DS:[EBX+18]
00473BA9	52	PUSH EDX
00473BAA	8BE8	MOV EBP, EAX
00473BAC	6A 40	PUSH 40
00473BAE	68 00100000	PUSH 1000
00473BB3	FF73 04	PUSH DWORD PTR DS:[EBX+4]
00473BB6	6A 00	PUSH 0
00473BB8	8B4B 10	MOV ECX, DWORD PTR DS:[EBX+10]
00473BBB	03CA	ADD ECX, EDX
00473BBD	8B01	MOV EAX, DWORD PTR DS:[ECX]
00473BBF	FFD0	CALL EAX
00473BC1	50	POP EAX

بار هم F9 :

Address	Hex dump	Disassembly
00473C2A	FFEB	JMP EAX
00473C2C	B0 71	MOV AL, 71
00473C2E	42	INC EDX
00473C2F	0000	ADD BYTE PTR DS:[EAX], AL
00473C31	0000	ADD BYTE PTR DS:[EAX], AL
00473C33	0000	ADD BYTE PTR DS:[EAX], AL
00473C35	0000	ADD BYTE PTR DS:[EAX], AL
00473C37	0000	ADD BYTE PTR DS:[EAX], AL
00473C39	0000	ADD BYTE PTR DS:[EAX], AL
00473C3B	0000	ADD BYTE PTR DS:[EAX], AL
00473C3D	0000	ADD BYTE PTR DS:[EAX], AL
00473C3F	0000	ADD BYTE PTR DS:[EAX], AL
00473C41	0000	ADD BYTE PTR DS:[EAX], AL
00473C43	0000	ADD BYTE PTR DS:[EAX], AL
00473C45	0000	ADD BYTE PTR DS:[EAX], AL
00473C47	0000	ADD BYTE PTR DS:[EAX], AL
00473C49	0000	ADD BYTE PTR DS:[EAX], AL

این پرش درست به OEP می رود. پس یکبار F8 بزنید تا به OEP برسید.

راه حل دیگر:

یک روش مخصوص PeCompact برای یافتن OEP وجود دارد. آن هم اینکه در اولین خط پکر یک دستور مثل `MOV EAX,xxxxxxx` قرار دارد. اگر ما به آدرس xxxxxxxx برویم. چند خط پایینتر دستور `JMP EAX` که به OEP می رود را پیدا می کنیم.

Address	Hex dump	Disassembly
00401000 <M	B8 683B4700	MOV EAX,00473B68
00401005	50	PUSH EAX
00401006	64:FF35 0000	PUSH DWORD PTR FS:[0]
0040100D	64:8925 0000	MOV DWORD PTR FS:[0],ESP
00401014	33C0	XOR EAX,EAX
00401016	8908	MOV DWORD PTR DS:[EAX],ECX
00401018	50	PUSH EAX
00401019	45	INC EBP
0040101A	43	INC EBX
0040101B	6F	OUTS DX,DWORD PTR ES:[EDI]
0040101C	6D	INS DWORD PTR ES:[EDI],DX
0040101D	70 61	JN SHORT 00401080
0040101F	637432 00	ARPL WORD PTR DS:[EDX+ESI],SI
00401023	1B66 23	SBB ESP,DWORD PTR DS:[ESI+23]
00401026	CA 157F	RETF 7F15
00401029	D96A 90	FLOCW WORD PTR DS:[EDX-70]

دستور اول `MOV EAX,00473B68` است. پس به خط 00473B68 بروید:

Address	Hex dump	Disassembly
00473B68	B8 ED280B00	MOV EAX,0B28ED
00473B6D	8D88 9E123C00	LEA ECX,DWORD PTR DS:[EAX+3C12]
00473B73	8941 01	MOV DWORD PTR DS:[ECX+1],EAX
00473B76	8B5424 04	MOV EDX,DWORD PTR SS:[ESP+4]
00473B7A	8B52 0C	MOV EDX,DWORD PTR DS:[EDX+C]
00473B7D	C602 E9	MOV BYTE PTR DS:[EDX],0E9
00473B80	83C2 05	ADD EDX,5
00473B83	2BCA	SUB ECX,EDX
00473B85	894A FC	MOV DWORD PTR DS:[EDX-4],ECX
00473B88	33C0	XOR EAX,EAX
00473B8A	C3	RET
00473B8B	B8 78563412	MOV EAX,12345678
00473B90	64:8F05 000000	POP DWORD PTR FS:[0]
00473B97	83C4 04	ADD ESP,4
00473B9A	55	PUSH EBP
00473B9B	53	PUSH EBX
00473B9C	51	PUSH ECX
00473B9D	57	PUSH EDI
00473B9E	56	PUSH ESI
00473B9F	52	PUSH EDX
00473BA0	8D98 57123C00	LEA EBX,DWORD PTR DS:[EAX+3C12]
00473BA6	8B53 18	MOV EDX,DWORD PTR DS:[EBX+18]
00473BA9	52	PUSH EDX
00473BAA	8BE8	MOV EBP,EAX
00473BAC	6A 40	PUSH 40
00473BAE	68 00100000	PUSH 1000
00473BB3	FF73 04	PUSH DWORD PTR DS:[EBX+4]
00473BB6	6A 00	PUSH 0
00473BB8	8B4B 10	MOV ECX,DWORD PTR DS:[EBX+10]
00473BBB	03CA	ADD ECX,EDX
00473BD0	8B41	MOV EAX,DWORD PTR DS:[ECX+1]

حالا با چرخانک موس مقداری به پایین بروید(Scroll کنید) تا به دستور `JMP EAX` برسید:

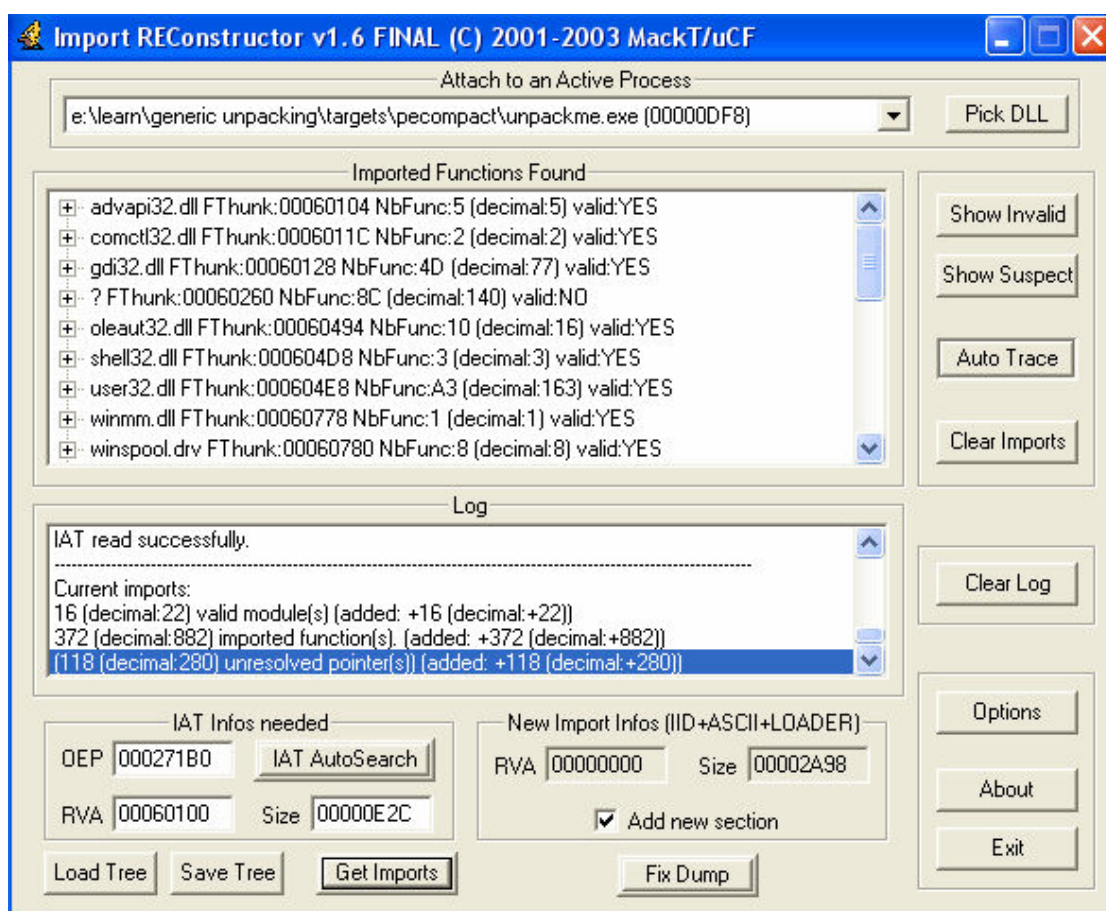
Address	Hex dump	Disassembly
00473C22	8BC6	MOV EAX,ESI
00473C24	5A	POP EDX
00473C25	5E	POP ESI
00473C26	5F	POP EDI
00473C27	59	POP ECX
00473C28	5B	POP EBX
00473C29	5D	POP EBP
00473C2A	FFEB	JMP EAX
00473C2C	0000	ADD BYTE PTR DS:[EAX],AL
00473C2E	0000	ADD BYTE PTR DS:[EAX],AL
00473C30	0000	ADD BYTE PTR DS:[EAX],AL
00473C32	0000	ADD BYTE PTR DS:[EAX],AL
00473C34	0000	ADD BYTE PTR DS:[EAX],AL
00473C36	0000	ADD BYTE PTR DS:[EAX],AL
00473C38	0000	ADD BYTE PTR DS:[EAX],AL

این دستور ما را به OEP می برد. پس بر روی آن BP بگذارید و برنامه را با F9 اجرا کنید:

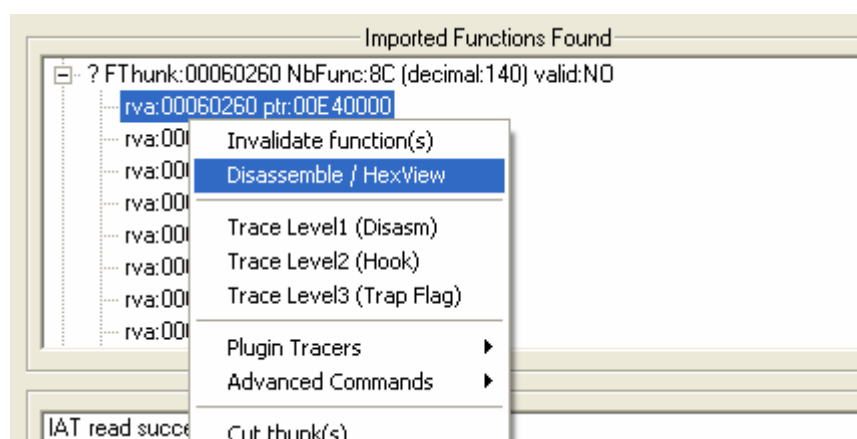
FF11	CALL DWORD PTR DS:[ECX]	
8BC6	MOV EAX,ESI	
5A	POP EDI	
5E	POP ESI	
5F	POP EDI	
59	POP ECX	
5B	POP EBX	
5D	POP EBP	
FFE0	JMP EAX	UnPackMe.004271B0
B8 71	MOV AL,71	
42	INC EDI	
0000	ADD BYTE PTR DS:[EAX],AL	
0000	ADD BYTE PTR DS:[EAX],AL	
0000	ADD BYTE PTR DS:[EAX],AL	
0000	ADD BYTE PTR DS:[EAX],AL	
0000	ADD BYTE PTR DS:[EAX],AL	

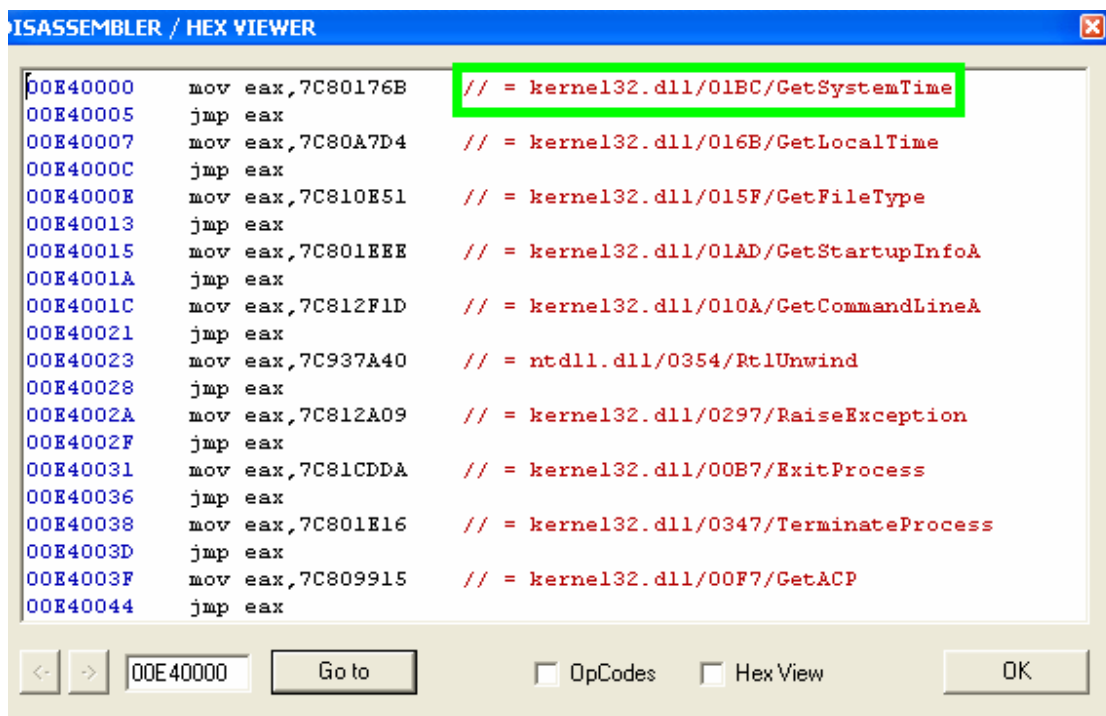
حالا با یک F8 به OEP می رسمیم ☺

از فایل دامپ بگیرد و ImportREC را باز کنید، OEP را بر حسب RVA بدهید، گزینه IAT Auto-search را بزنید و سپس گزینه Get Imports را بزنید.

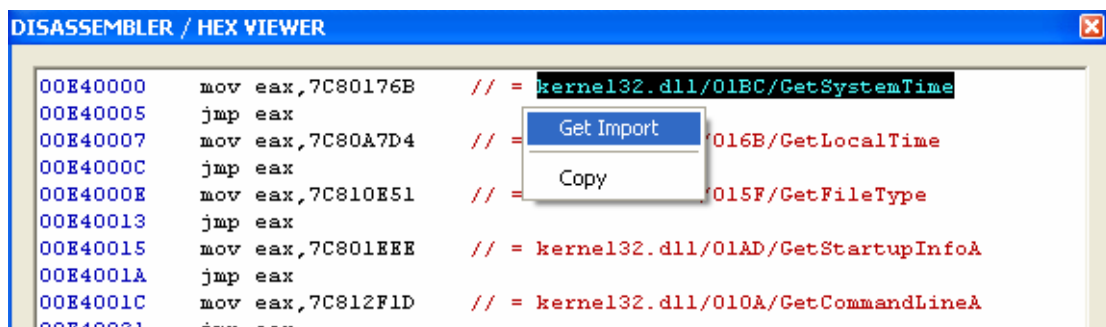


همانطور که می بینید تعدادی از توابع ما Invalid است. من یک تابع Invalid را درست می کنم، بقیه اش با شما... یکی از توابع Invalid را انتخاب کرده، کلیک راست کرده و گزینه Disassemble/hex view را بزنید:





می بینید؟ این آدرس از IAT در واقع به تابع `GetSystemTime` می رود. پس همانند تصویر کلیک راست کرده و گزینه `GetImport` را بزنید:



به همین ترتیب ما می توانیم بقیه توابع `Redirect` شده را درست کنیم. اگر بخواهیم تک تک همه اینها را درست کنیم وقت زیادی از ما گرفته می شود. (چون باید ۲۸۰ تابع را به همین ترتیب درست کنیم.)
بعد از اینکه همه اینها را درست کرده اید، دو آدرس می ماند که به این راحتی ها درست نمی شود. چون `PeCompact` آنها را پیشرفته تر از این `Redirect` کرده است. آن دو تابع `GetProcAddress` هستند © از کجا فهمیدم؟... خوب در داخل برنامه `PeCompact` یک پلاگین وجود دارد که با این پلاگین توابع را `Redirect` می کنند. در آن تابع قابلیت وجود دارد که می تواند تابع `GetProcAddress` را با شیوه پیشرفته تری `Redirect` کند. پس بر روی آن دو آدرس دابل کلیک می کنید و آدرس تابع `GetProcAddress` را به برنامه می دهید و بعد هم فایل دامپ خود را فیکس می کنید.

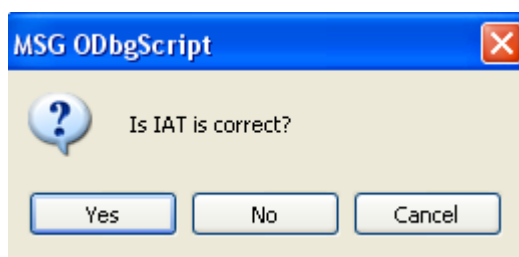
راه حل دیگر:

همانطور که می بینید، `Fix` کردن توابع `Invalid` در `PeCompact` کار سختی نیست. اما وقت ما را می گیرد. به همین دلیل یک اسکریپت نوشتم که این کار را به طور خودکار انجام می دهد. فایل را در `OlllyDBG` باز کنید. (تمامی BPها را بردارید.) و در منوی `Plugins` گزینه `ODBGScript` و سپس گزینه `Run Script` را بزنید و بعد اسکریپت (در داخل `Attachment` موجود است.) را به برنامه بدهید:

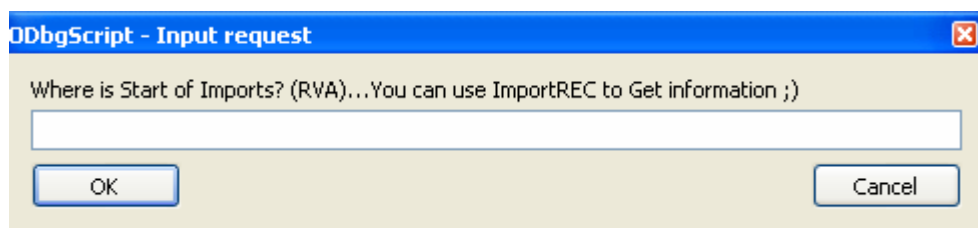
dump	Disassembly	Comment
55	PUSH EBP	This is Oep ;)
8BEC	MOV EBP, ESP	
6A FF	PUSH -1	
68 600E4500	PUSH 00450E60	
68 C8924200	PUSH 004292C8	
64:A1 00000000	MOV EAX, DWORD PTR FS:[0]	SE handler installation
50	PUSH EAX	
64:8925 00000000	MOV DWORD PTR FS:[0], ESP	
83C4 A8	ADD ESP, -58	

خوب، پس اسکریپت `OEP` را درست پیدا کرده. (اگر درست پیدا نشد، کلیدهای `ALT + O` را بزنید و در قسمت `Exceptions` تیک تمامی گزینه ها را بزنید. همچنین تمامی BPها را پاک کنید.)

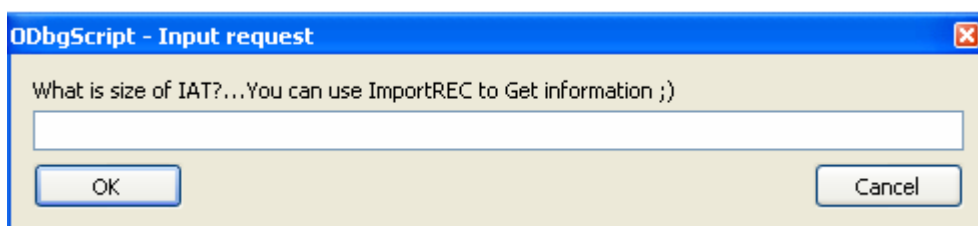
اسکرپت از شما سوال می کند:



کلید No را بزنید. (چون IAT مشکل دارد.):



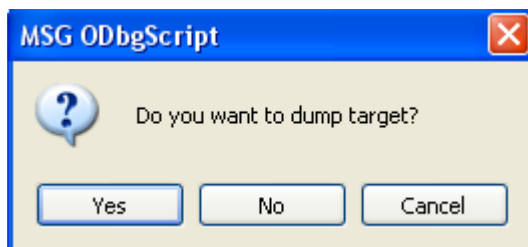
اسکرپت از ما می پرسد محل شروع IAT کجاست?...همانطور که در ImportREC دیدیم، محل شروع بر حسب RVA برابر 60100 بوده. پس این مقدار را وارد کنید و کلید OK را بزنید:



حالا سایز IAT چقدر بوده?... ImportREC می گوید: E2C، پس ما هم همین مقدار را وارد می کنیم:



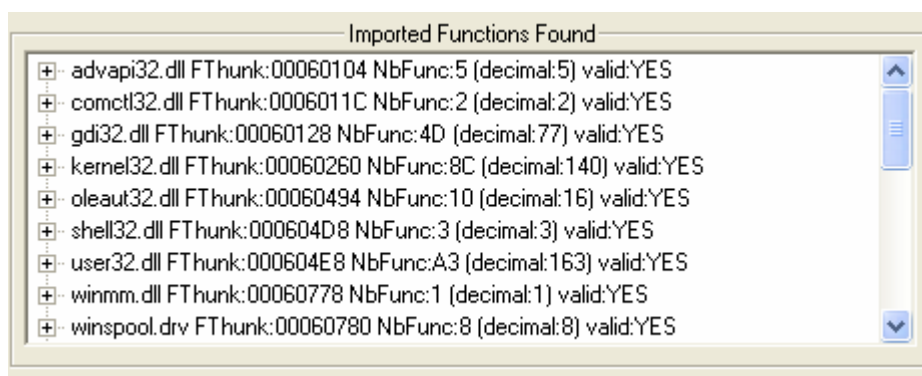
کلید OK را بزنید:



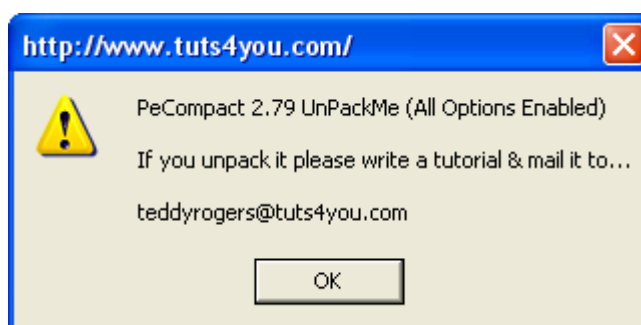
اگر می خواهید از فایل دامپ بگیرد، کلید Yes و در غیر این صورت کلید No را بزنید:



حالا دوباره توابع را در ImportREC وارد کنید:

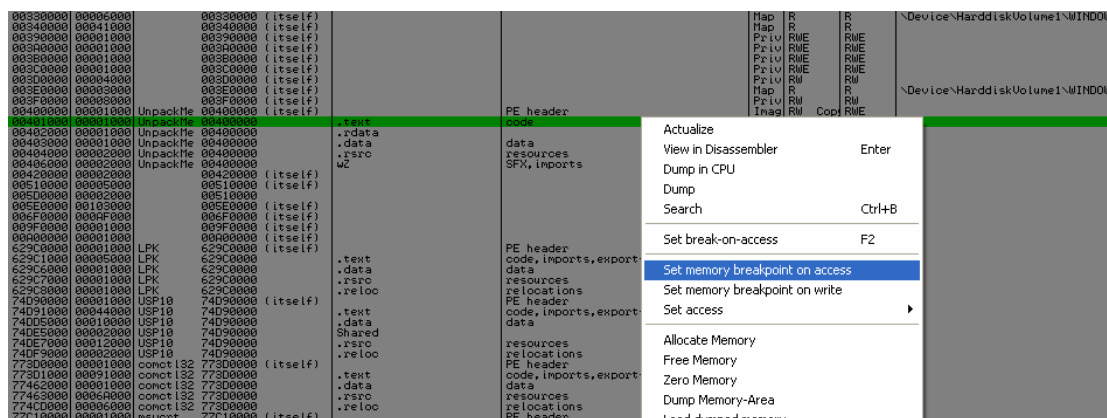
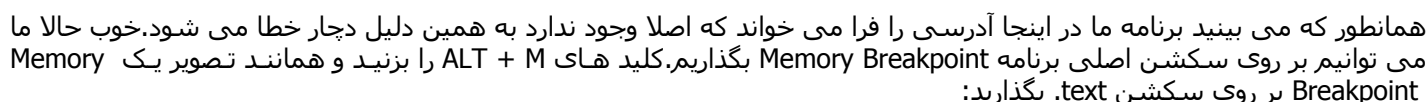
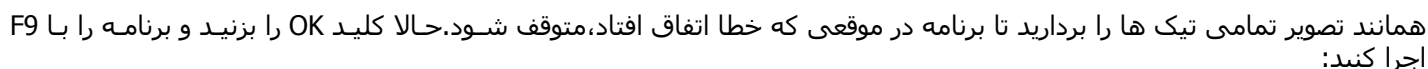


این بار تمامی توابع درست شده اند. حالا فایل دامپ خود را فیکس کنید:



برای یافتن OEP در ExeShield هم می توانم به راحتی از Memory Breakpoint استفاده کنم.

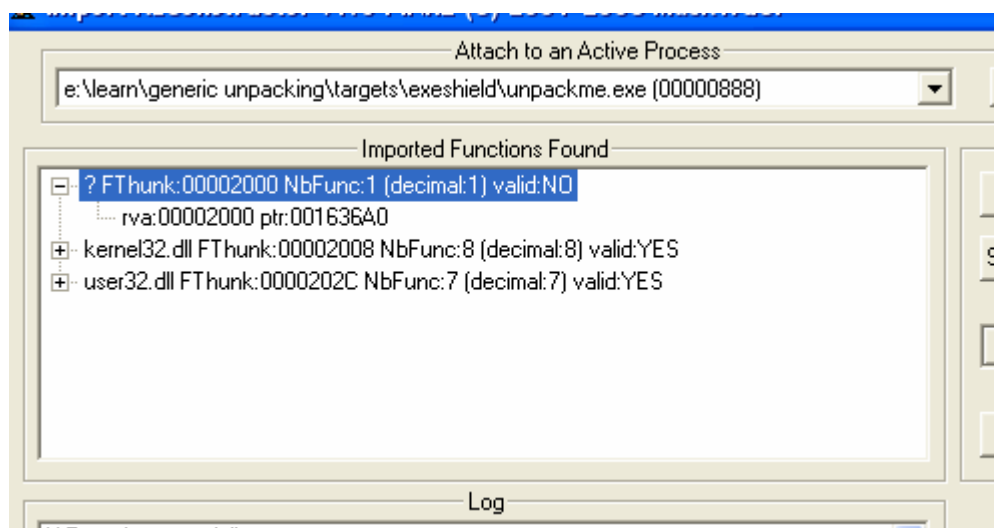
اول باید ببینم که در کجا باید Memory Breakpoint بگذارم؟...خوب،اگه خطایی در برنامه اتفاق بیفتد،می توانم بعد از اینکه آخرین خطا در برنامه اتفاق افتاد،انجا Memory Breakpoint بگذارم.پس اول باید به OllyDBG بگویم اگر خطایی در برنامه اتفاق افتاد،خطا را رد نکن و در همانجا متوقف شو.در منوی Options گزینه Debugging Options را بزنی و وارد قسمت Exceptions شوی:



حالا هم کلیدهای Shift + F9 را بزنید. (چون در برنامه خطا اتفاق افتاده، برای رد کردن خطا باید کلید Shift را هم نگه دارید).

Address	Hex dump	Disassembly
00401000	6A 00	PUSH 0
00401002	E8 0B030000	CALL 00401312
00401007	A3 14364000	MOV DWORD PTR DS:[403614],EAX
0040100C	E8 4F030000	CALL 00401360
00401011	6A 00	PUSH 0
00401013	68 2E104000	PUSH 0040102E
00401018	6A 00	PUSH 0
0040101A	6A 65	PUSH 65
0040101C	FF35 14364000	PUSH DWORD PTR DS:[403614]
00401022	E8 0F030000	CALL 00401336
00401027	6A 00	PUSH 0
00401029	E8 DE020000	CALL 0040130C
0040102E	55	PUSH EBP
0040102F	8BEC	MOV EBP,ESP
00401031	8B45 0C	MOV EAX,DWORD PTR SS:[EBP+C]
00401034	3D 10010000	CMP EAX,110
00401039	75 30	JNZ SHORT 0040106B
0040103B	6A 01	PUSH 1
0040103D	FF35 14364000	PUSH DWORD PTR DS:[403614]
00401043	E8 FA020000	CALL 00401342
00401048	68 40304000	PUSH 00403040

همانطور که می بینید ما درست در OEP فرود آمدیم ©
حالا از فایل دامپ بگیریید و ImportREC را اجرا کرده، OEP را به برنامه بدهید و کارهای لازم را انجام دهید:



همانطور که می بینید یکی از توابع ما Redirect شده، ولی مشکلی نیست. چون کافیسیت بر روی آن کلیک راست کرده و گزینه Trace Level1 (disasm) را بزنیم تا تابع اصلی ما مشخص شود ©
حالا هم فایل دامپ شده خودمان را فیکس می کنیم. فایل آپک شده ما بی هیچ مشکلی اجرا می شود.

Crunch

یافتن OEP در Crunch هم بسیار راحت است. به راحتی می توانیم از رجیستر ESP استفاده کنیم. برنامه را در OllyDBG باز کنید، دو بار کلید F8 را بزنید، تا مقدار رجیستر ESP تغییر کند، بعد هم بر روی مقدار رجیستر ESP یک Hardware breakpoint on access می گذارید و کلید F9 را می زنید، من در اینجا متوقف شدم:

#6E7F7	03C5	ADD EAX,EBP	
#6E7F9	83C0 30	ADD EAX,30	
#6E7FC	8B00	MOV EAX,DWORD PTR DS:[EAX]	
#6E7FE	0385 E8070000	ADD EAX,DWORD PTR SS:[EBP+7E8]	
#6E804	8D9D F5000000	LEA EBX,DWORD PTR SS:[EBP+BF5]	
#6E80A	5D	POP EBP	
#6E80B	5A	PUSH EAX	bitarts_..004271B0
#6E80C	60	PUSHAD	
#6E80D	55	PUSH EBP	
#6E80E	53	PUSH EBX	
#6E80F	64:FF35 00000000	PUSH DWORD PTR FS:[0]	
#6E816	64:8925 00000000	MOV DWORD PTR FS:[0],ESP	
#6E81D	E8 64000000	CALL 0046E886	
#6E822	DF01	FILL WORD PTR DS:[ECX]	
#6E824	C2 DF21	RET 21DF	
#6E827	DF10	FIST WORD PTR DS:[EAX]	
#6E829	0048 BE	ADD BYTE PTR DS:[EAX-42],CL	
#6E82C	47	INC EDI	
#6E82D	0000	ADD BYTE PTR DS:[EAX],AL	
#6E82F	00E8	ADD AL,CH	
#6E831	51	PUSH ECX	
#6E832	0000	ADD BYTE PTR DS:[EAX],AL	
#6E834	00DF	ADD BH,BL	
#6E836	00C3	ADD BL,AL	
#6E838	0321	ADD ESP,DWORD PTR DS:[ECX]	

یکبار دیگر F9 را می زنم:

004271AC	90	NOP	
004271AD	90	NOP	
004271AE	90	NOP	
004271AF	90	NOP	
004271B0	55	PUSH EBP	
004271B1	8BEC	MOV EBP,ESP	
004271B3	6A FF	PUSH -1	
004271B5	68 600E4500	PUSH 00450E60	
004271B8	68 C8924200	PUSH 004292C8	
004271BF	64:A1 00000000	MOV EAX,DWORD PTR FS:[0]	
004271C5	50	PUSH EAX	
004271C6	64:8925 00000000	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 SS:[EBP-18],ESP	
004271D6	FF15 DC0A4600	CALL DWORD PTR DS:[460ADC]	
004271DC	33D2	XOR EDX,EDX	
004271DE	8AD4	MOV DL,AH	
004271E0	8915 34E64500	MOV DWORD PTR DS:[45E634],EDX	
004271E6	8BC8	MOV ECX,EAX	
004271E8	81E1 FF000000	AND ECX,0FF	
004271EE	890D 30E64500	MOV DWORD PTR DS:[45E630],ECX	
004271F4	C1E1 08	SHL ECX,8	
004271F7	03CA	ADD ECX,EDX	
004271F9	890D 2CE64500	MOV DWORD PTR DS:[45E62C],ECX	
004271FF	C1E8 10	SHR EAX,10	
00427202	A3 28E64500	MOV DWORD PTR DS:[45E628],EAX	

اینبار به OEP رسیدیم ☺
بعد از این دیگر مشکلی نیست، دامپ می گیرید و فیکس می کنید.

Acprotect

OEP در ورژن دوم Acprotect به راحتی پیدا می شود. کافیه بعد از دستور PushAD روی مقدار رجیستر ESP یک Breakpoint بگذاریم و بعد از چند بار زدن کلید F9 به پرشی می رسیم که ما را به OEP می برد. ☺

برنامه را در OllyDBG اجرا کنید، ۵ بار کلید F8 را بزنید تا دستور PushAD اجرا شود و بعد بر روی مقدار رجیستر ESP یک Breakpoint on access را بگذارید و بعد کلید F9 را بزنید:

0048050E	0000	ADD BYTE PTR DS:[EAX],AL
00480510	66:D3EE	SHR SI,CL
00480513	41	INC ECX
00480514	61	POPAD
00480515	54	PUSH ESP
00480516	8F05 F9B94600	POP DWORD PTR DS:[46B9F9]
0048051C	60	PUSHAD
0048051D	EB 0A	JMP SHORT 00480529
0048051F	E9 EB19EBE8	JMP 00381E8F
00480524	0B00	OR EAX,DWORD PTR DS:[EAX]
00480526	0000	ADD BYTE PTR DS:[EAX],AL
00480528	78 7A	JS SHORT 004805A4
0048052A	F8	CLC
0048052B	7B F6	JPD SHORT 00480523
0048052D	E8 7AF07BEE	CALL EEC3F5AC
00480532	9A 83C40474 F675	CALL FAR 75F6:7404C483
00480539	F4	HLT
0048053A	7C 7B	JL SHORT 004805B7
0048053C	0366 D3	ADD ESP,DWORD PTR DS:[ESI-2D]
0048053F	CF	IRETD
00480540	68 25074800	PUSH 00480725
00480545	66:C1FE 26	SAR SI,26
00480549	5A	POP EDX
0048054A	7C 0E	JL SHORT 0048055A
0048054C	7D 0C	JGE SHORT 0048055A
0048054E	7C 7C	JL SHORT 004805CC
0048054F	7C 7C	JL SHORT 004805CC

اینجا اتفاق خاصی نمی افتد، یکبار دیگر کلید F9 را بزنید:

00480513	41	INC ECX
00480514	61	POPAD
00480515	54	PUSH ESP
00480516	8F05 F9B94600	POP DWORD PTR DS:[46B9F9]
0048051C	60	PUSHAD
0048051D	EB 0A	JMP SHORT 00480529
0048051F	E9 EB19EBE8	JMP 00381E8F
00480524	0B00	OR EAX,DWORD PTR DS:[EAX]
00480526	0000	ADD BYTE PTR DS:[EAX],AL
00480528	78 7A	JS SHORT 004805A4
0048052A	F8	CLC
0048052B	7B F6	JPD SHORT 00480523
0048052D	E8 7AF07BEE	CALL EEC3F5AC
00480532	9A 83C40474 F675	CALL FAR 75F6:7404C483
00480539	F4	HLT
0048053A	7C 7B	JL SHORT 004805B7
0048053C	0366 D3	ADD ESP,DWORD PTR DS:[ESI-2D]
0048053F	CF	IRETD
00480540	68 25074800	PUSH 00480725
00480545	66:C1FE 26	SAR SI,26
00480549	5A	POP EDX
0048054A	7C 0E	JL SHORT 0048055A
0048054C	7D 0C	JGE SHORT 0048055A
0048054E	7C 7C	JL SHORT 004805CC
00480550	17	POP SS
00480551	7D 15	JGE SHORT 00480568

یکبار دیگر هم F9 بزنید:

Address	Hex dump	Disassembly
00481167	71 83	JNO SHORT 004810EC
00481169	C40474	LES EAX,FWORD PTR SS:[ESP+ESI*2]
0048116C	F675 F4	DIV BYTE PTR SS:[EBP-C]
0048116F	7F 43	JG SHORT 004811B4
00481171	8115 25BA4600 B1C16DCD	ADC DWORD PTR DS:[46BA25],CD6DC1B1
0048117B	61	POPAD
0048117C	50	PUSH EAX
0048117D	8F05 21BB4600	POP DWORD PTR DS:[46BB21]
00481183	60	PUSHAD
00481184	EB 0A	JMP SHORT 00481198
00481186	9A EB197AE8 0B00	CALL FAR 000B:E87A19EB
0048118D	0000	ADD BYTE PTR DS:[EAX],AL
0048118F	71 78	JNO SHORT 00481209
00481191	F8	CLC
00481192	79 F6	JNS SHORT 0048118A
00481194	E9 76F077EE	JMP 00380529F
00481199	7D 83	JGE SHORT 0048111E
0048119B	C40478	LES EAX,FWORD PTR DS:[EAX+EDI*2]
0048119E	F679 F4	IDIV BYTE PTR DS:[ECX-C]
004811A1	E8 6681ED16	CALL 1735930C
004811A6	60	PUSHAD
004811A7	66:D3CD	ROR BP,CL
004811AA	68 A9134800	PUSH 004813A9
004811AF	85D5	TEST EBP,EDX
004811B1	5E	POP ESI
004811B2	78 0E	JS SHORT 004811C2
004811B4	79 0C	JNS SHORT 004811C2
004811B6	E8 7E147F12	CALL 12072689

یک F9 دیگر:

0048117D	8F05 21B4600	POP DWORD PTR DS:[46BB21]
00481183	60	PUSHAD
00481184	EB 0A	JMP SHORT 00481190
00481186	9A EB197AE8 0B00	CALL FAR 000B:E87A19EB
0048118D	0000	ADD BYTE PTR DS:[EAX],AL
0048118F	71 78	JNO SHORT 00481209
00481191	F8	CLC
00481192	79 F6	JNS SHORT 0048118A
00481194	E9 76F077EE	JMP 0000020F
00481199	7D 83	JGE SHORT 0048111E
0048119B	C40478	LES EAX,FWORD PTR DS:[EAX+EDI*2]
0048119E	F679 F4	IDIV BYTE PTR DS:[ECX-C]
004811A1	E8 6681ED16	CALL 1735930C
004811A6	60	PUSHAD
004811A7	66:D3CD	ROR BP,CL
004811AA	68 A9134800	PUSH 004813A9
004811AF	85D5	TEST EBP,EDX
004811B1	5E	POP ESI
004811B2	78 0E	JS SHORT 004811C2
004811B4	79 0C	JNS SHORT 004811C2
004811B6	E8 7E147F12	CALL 12C72639
004811BB	7F E8	JG SHORT 004811A5
004811BD	07	POP ES
004811BE	0000	ADD BYTE PTR DS:[EAX],AL

بار هم F9:

004815B1	78 8B	JGE SHORT 00481619
004815B3	BF 81E47106	MOV EDI,671E481
004815B8	81D9 AD648C33	SBB ECX,338C64AD
004815BE	61	POPAD
004815BF	54	PUSH ESP
004815C0	8F05 71B94600	POP DWORD PTR DS:[46B971]
004815C6	60	PUSHAD
004815C7	7A 0E	JPE SHORT 004815D7
004815C9	7B 0C	JPO SHORT 004815D7
004815CB	E8 83C40472	CALL 724CDA53
004815D0	14 73	ADC AL,73
004815D2	12E8	ADC CH,AL
004815D4	EB 0A	JMP SHORT 004815E0
004815D6	78 EB	JS SHORT 004815C3
004815D8	FB	STI
004815D9	E9 E8EDFFFF	JMP 004808C6
004815DE	FFE8	JMP FAR EBX
004815E0	72 F8	JB SHORT 004815DA
004815E2	73 F6	JNB SHORT 004815DA
004815E4	E8 E9090000	CALL 00481FD2
004815E9	00D3	ADD BL,DL
004815EB	D20F	ROR BYTE PTR DS:[EDI],CL
004815ED	8601	XCHG BYTE PTR DS:[ECX],AL
004815EF	0000	ADD BYTE PTR DS:[EAX],AL
004815F1	004E 68	ADD BYTE PTR DS:[ESI+68],CL
004815F4	E2 17	LOOPD SHORT 00481600
004815F6	48	DEC EAX
004815F7	00D3	ADD BL,DL
004815F8	ED	IN EAX,DX
004815F9	5F	POP EDI
004815FB	E8 07000000	CALL 00481607
00481600	7A FB	JPE SHORT 004815ED

: F9

004815B8	81D9 AD648C33	SBB ECX,338C64AD
004815BE	61	POPAD
004815BF	54	PUSH ESP
004815C0	8F05 71B94600	POP DWORD PTR DS:[46B971]
004815C6	60	PUSHAD
004815C7	7A 0E	JPE SHORT 004815D7
004815C9	7B 0C	JPO SHORT 004815D7
004815CB	E8 83C40472	CALL 724CDA53
004815D0	14 73	ADC AL,73
004815D2	12E8	ADC CH,AL
004815D4	EB 0A	JMP SHORT 004815E0
004815D6	78 EB	JS SHORT 004815C3
004815D8	FB	STI
004815D9	E9 E8EDFFFF	JMP 004808C6
004815DE	FFE8	JMP FAR EBX
004815E0	72 F8	JB SHORT 004815DA
004815E2	73 F6	JNB SHORT 004815DA
004815E4	E8 E9090000	CALL 00481FD2
004815E9	00D3	ADD BL,DL
004815EB	D20F	ROR BYTE PTR DS:[EDI],CL
004815ED	8601	XCHG BYTE PTR DS:[ECX],AL
004815EF	0000	ADD BYTE PTR DS:[EAX],AL
004815F1	004E 68	ADD BYTE PTR DS:[ESI+68],CL
004815F4	E2 17	LOOPD SHORT 00481600
004815F6	48	DEC EAX
004815F7	00D3	ADD BL,DL
004815F9	ED	IN EAX,DX
004815FA	5F	POP EDI
004815FB	E8 07000000	CALL 00481607
00481600	7A FB	JPE SHORT 004815ED

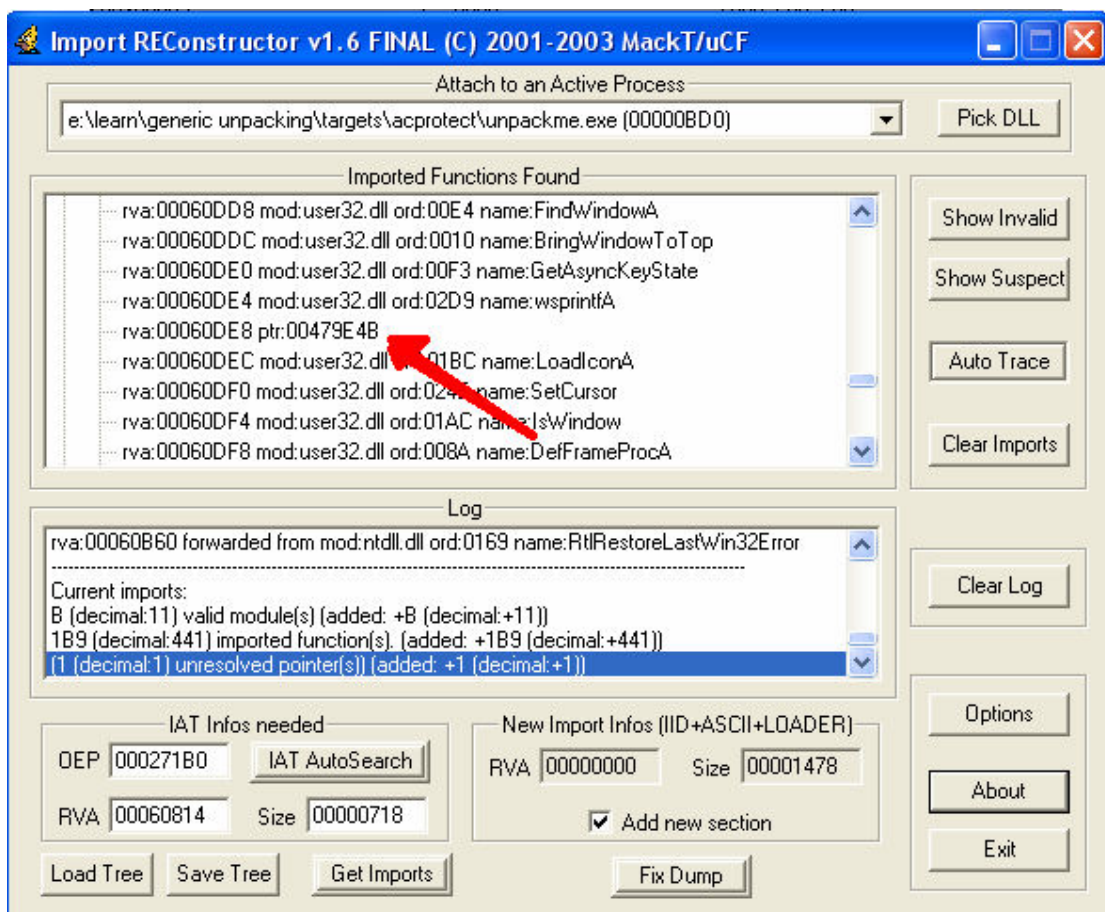
یک F9 دیگر:

00482274	0000	POP EBP
00482276	CA C300	ADD BYTE PTR DS:[EAX],AL
00482278	0002	RETF 0C3
0048227B	E8 D881FFFF	ADD BYTE PTR DS:[EDX],AL
00482282	AB	CALL 0047A45A
00482283	^ E2 F8	STOS DWORD PTR ES:[EDI]
00482285	61	LOOPD SHORT 0048227D
00482286	EB 01	POPAD
00482288	E8 FF25CB22	JMP SHORT 00482289
0048228D	48	CALL 2313488C
0048228E	0060 E8	DEC EAX
00482291	0000	ADD BYTE PTR DS:[EAX-18],AH
00482293	0000	ADD BYTE PTR DS:[EAX],AL
00482295	5E	ADD BYTE PTR DS:[EAX],AL
00482296	83EE 06	POP ESI
00482299	B9 66000000	SUB ESI,6
0048229E	29CE	MOV ECX,66
004822A0	BA 49C760E6	SUB ESI,ECX
004822A5	C1E9 02	MOV EDX,E660C749
004822A8	83E9 02	SHR ECX,2
004822AB	83F9 00	SUB ECX,2
004822AE	7C 1A	CMP ECX,0
004822B0	8B048E	JL SHORT 004822CA
004822B3	8B5C8E 04	MOV EAX,DWORD PTR DS:[ESI+ECX*4]
004822B7	33C3	MOV EBX,DWORD PTR DS:[ESI+ECX*4+1]
004822B9	C1C0 0C	XOR EAX,EBX
		ROL EAX,0C

آها، دیگر تقریبا به OEP رسیدیم. یکبار F8 را بزنید:

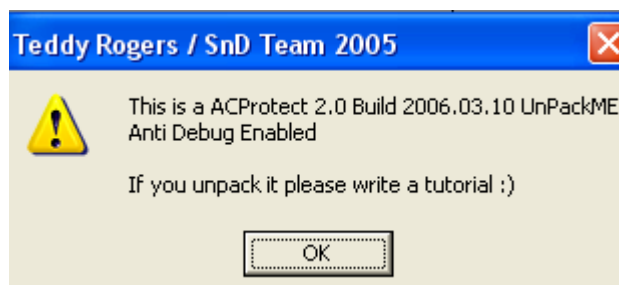
Address	Hex dump	Disassembly
00482289	- FF25 CB224800	JMP DWORD PTR DS:[4822CB]
0048228F	60	PUSHAD
00482290	E8 00000000	CALL 00482295
00482295	5E	POP ESI
00482296	83EE 06	SUB ESI,6
00482299	B9 66000000	MOV ECX,66
0048229E	29CE	SUB ESI,ECX
004822A0	BA 49C760E6	MOV EDX,E660C749
004822A5	C1E9 02	SHR ECX,2
004822A8	83E9 02	SUB ECX,2
004822AB	83F9 00	CMP ECX,0
004822AE	7C 1A	JL SHORT 004822CA
004822B0	8B048E	MOV EAX,DWORD PTR DS:[ESI+ECX*4]
004822B3	8B5C8E 04	MOV EBX,DWORD PTR DS:[ESI+ECX*4+1]
004822B7	33C3	XOR EAX,EBX
004822B9	C1C0 0C	ROL EAX,0C
004822BC	03C2	ADD EAX,EDX
004822BE	81C2 B2B73DA	ADD EDX,DA738BB2
004822C4	89048E	MOV DWORD PTR DS:[ESI+ECX*4],EAX
004822C7	49	DEC ECX
004822C8	^ EB E1	JMP SHORT 00482289

این پرش مستقیم به OEP می رود. یک بار F8 را بزنید تا به OEP برسید و بعد از فایل دامپ بگیرید. حالا ImportREC را باز کنید. OEP را به برنامه بدهید و سایر مراحل لازم را انجام دهید:

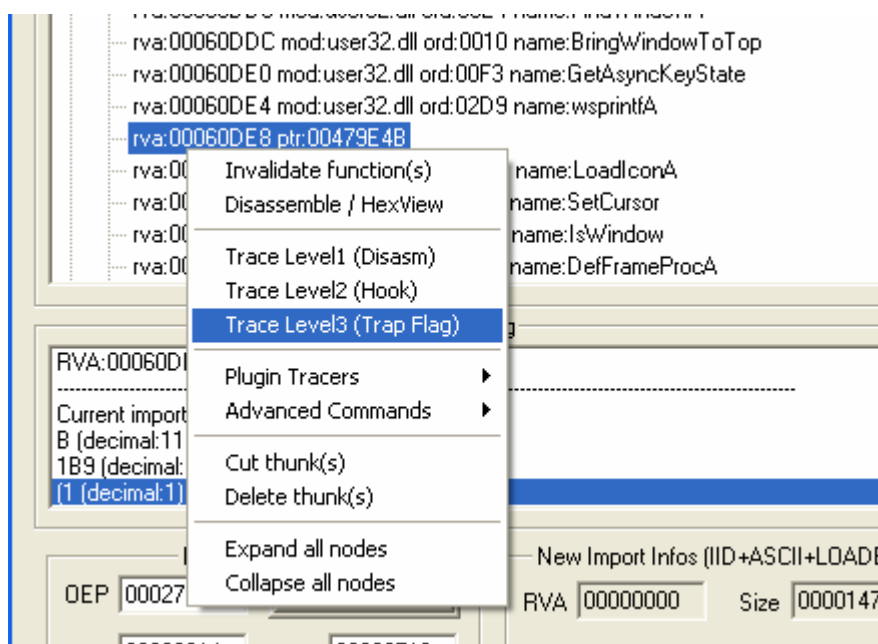


همانطور که می بینید یکی از توابع ما Redirect شده است. برای یافتن تابع اصلی راه حل های زیادی وجود دارد. (حتی من اولین بار توانستم حدس بزنم که تابع اصلی چی ممکن است باشد) من در اینجا می خواهم از قابلیت Trap Flag در ImportREC استفاده کنیم. ImportREC با تزریق یک Thread در پروسه ما متوجه می شود که این آدرس در واقع به کجا می رود ☺ این روش در خیلی از پکرها جواب می دهد. البته پکرهایی هم وجود دارند که برای مقابله با این قابلیت ImportREC در صورتی که ImportREC به آنها Thread اضافه کنند بلافاصله خودشان را می بندند یا کارهای دیگر می کنند.

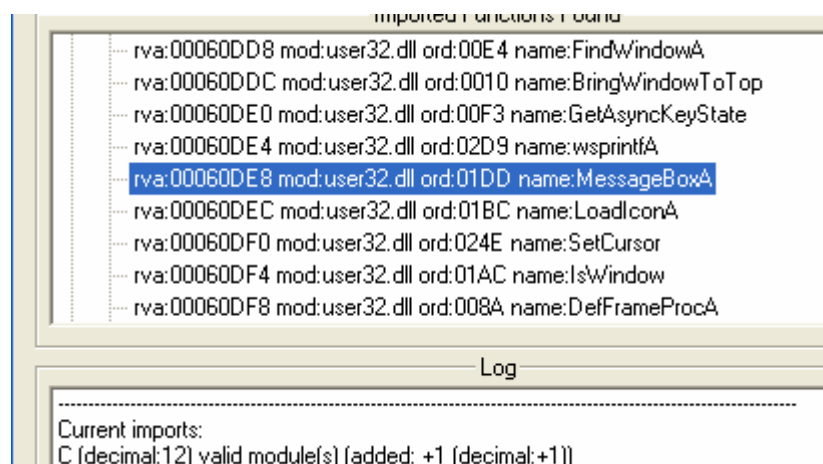
ابتدا در OllyDBG برنامه را به طور کامل اجرا کنید. (برنامه اگر در OllyDBG متوقف شده باشد، ImportREC نمی تواند به آن Thread اضافه کند و در نتیجه هنگ می کند.):



حالا در ImportREC بر روی تابع مورد نظر کلیک راست کرده و گزینه (Trap Flag) Trace Level3 را بزنید. همانند تصویر:



همانطور که در تصویر زیر می بینید ImportREC توانسته تابع اصلی را تشخیص دهد:



تابع Redirect شده ما در واقع MessageBoxA بوده. (البته این موضوع را خودتان هم می توانید حدس بزنید. چرا که این برنامه از تابع MessageBoxA برای نمایش پیغام استفاده می کند. در حالی که در توابع یافت شده چنین تابعی موجود نبوده. لذا این تابع Redirect شده مطمئناً باید MessageBoxA باشد.)

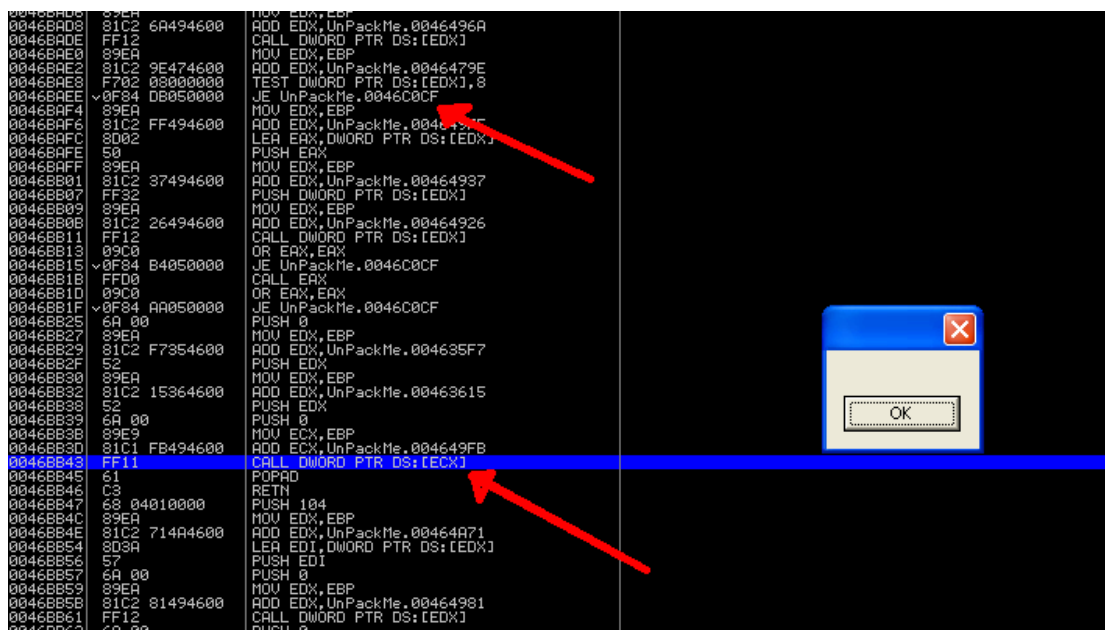
حالا فایل دامپ شده خود را فیکس کنید، می بینید که فایل آنپک شده بی هیچ مشکلی اجرا می شود.

NoName Packer

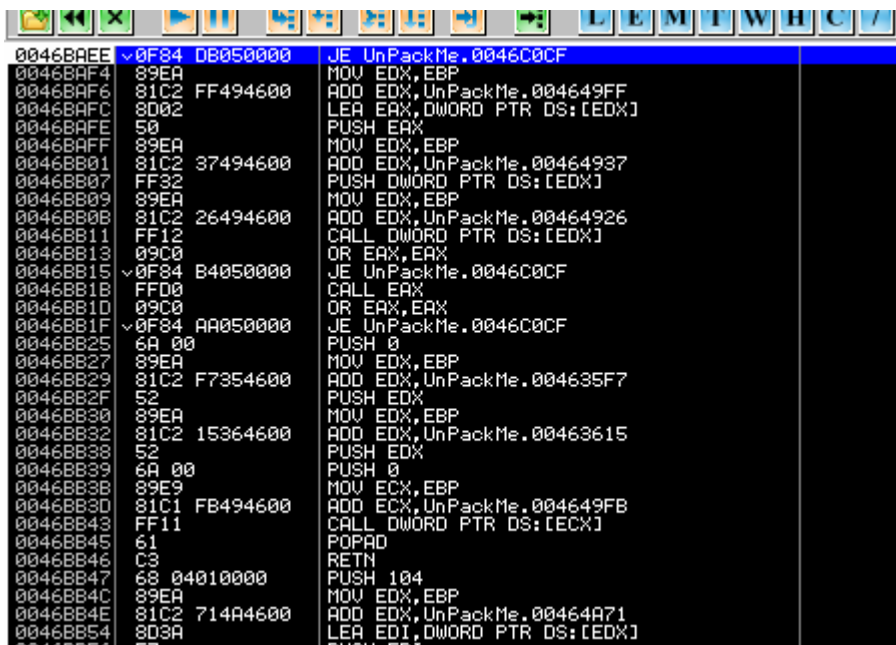
فایل را در OllyDBG بدون پلاگین باز کنید، کلید F9 را بزنید تا برنامه اجرا شود:



می بینید؟...این پیام موقعی که فایل در خارج از OllyDBG قرار داشت، نشان داده نمی شد. پس این یک Anti-Debug است. برنامه را ری استارت کنید و کلید های CTRL + F8 را بزنید تا برنامه به طور خودکار به جلو برود. بعد از مدتی:



پس این پیام در خط 0046BB43 اجرا می شود. چند خط بالاتر از این خط را Hardware Breakpoint بگذارید تا کدهای این قسمت را با دقت بیشتری بررسی کنیم. ممکن است سوال کنید چرا software breakpoint (کلید F2) نگذاشتیم؟...چون پکرها معمولاً کدهای خودشان رو تغییر می دهند (Self-modifying) به همین دلیل اگر ما یک Software breakpoint بگذاریم. چون OllyDBG دستور INT3 را اضافه می کند. پکر در عملیات Self-modifying دچار مشکل شده و برنامه می دهد. پس در مورد پکرها از Software breakpoint استفاده نکنید. اگر هم استفاده می کنید، موقعی که برنامه را ری استارت کردید، آنها را پاک کنید. خوب، برنامه را ری استارت کنید و کلید F9 را بزنید. در اینجا متوقف می شویم:



چند بار کلید F8 را بزنید تا به این خط برسید:

0046BB0B	81C2 26494600	ADD EDX, UnPackMe.00464926	
0046BB11	FF12	CALL DWORD PTR DS:[EDX]	
0046BB13	0000	OR EAX, EAX	
0046BB15	✓0F84 B4050000	JE UnPackMe.0046C0CF	
0046BB1B	FFD0	CALL EDX	kernel32.IsDebuggerPresent
0046BB1D	09C0	OR EAX, EAX	
0046BB1F	✓0F84 AA050000	JE UnPackMe.0046C0CF	
0046BB25	6A 00	PUSH 0	
0046BB27	89EA	MOV EDX, EBP	
0046BB29	81C2 F7354600	ADD EDX, UnPackMe.004635F7	
0046BB2F	52	PUSH EDX	
0046BB30	89EA	MOV EDX, EBP	
0046BB32	81C2 15364600	ADD EDX, UnPackMe.00463615	
0046BB39	52	PUSH EDX	
0046BB39	6A 00	PUSH 0	
0046BB3B	89E9	MOV ECX, EBP	
0046BB3D	81C1 FB494600	ADD ECX, UnPackMe.004649FB	
0046BB43	FF11	CALL DWORD PTR DS:[ECX]	
0046BB45	61	POPAD	
0046BB46	C3	RETN	
0046BB47	68 04010000	PUSH 104	
0046BB4C	89EA	MOV EDX, EBP	
0046BB4E	81C2 714A4600	ADD EDX, UnPackMe.00464A71	
0046BB54	0300	LEO EDI, DWORD PTR DS:[EDX]	

در این خط تابع IsDebuggerPresent اجرا می شود. پس این تابع همان آنتی دیباگ پکر ما بود. خوب... این بار از پلاگین برای رد کردن این آنتی دیباگ از پلاگین استفاده می کنیم... پلاگین HideDBG.dll را در پوشه Plugin های OllyDBG کپی نمایید و OllyDBG را یکبار ببندید و دوباره باز کنید، حالا در منوی Plugins گزینه HideDBG و سپس گزینه HideAll را بزنید. و بعد کلید F9 را بزنید و برنامه را اجرا کنید. انبار برنامه بی هیچ مشکلی اجرا می شود.

خوب برای یافتن OEP می توانیم از Memory Breakpoint استفاده کنیم. اما کجا باید Memory Breakpoint بگذاریم؟... باید با کلید F8 آنقدر به جلو برویم تا کدها به طور کامل Decrypt شود و بعد Memory Breakpoint بگذاریم.

Address	Disassembly	Comment
0046C34C	3E:C600 00	MOV BYTE PTR DS:[EAX],0
0046C350	40	INC EAX
0046C351	3E:8038 00	CMP BYTE PTR DS:[EAX],0
0046C355	75 F5	JNZ SHORT UnPackMe.0046C34C
0046C357	C3	RETN
0046C358	55	PUSH EBP
0046C359	89E5	MOV EBP,ESP
0046C35B	57	PUSH EDI
0046C35C	36:8B45 10	MOV EAX,DWORD PTR SS:[EBP+10]
0046C360	3E:8B8B 9C000000	MOV EDI,DWORD PTR DS:[EAX+9C]
0046C367	89FA	MOV EDX,EDI

خوب، حالا کلیدهای ALT + M را بزنید. و همانند تصویر بر روی سکشن text یک Memory breakpoint on access بگذارید:

00039000	00008000			Map R	RW	K	(Device HarddiskVolume1\Windows\System32\
00039000	00008000			Priv RW			
0003A000	00001000			Priv RW			
0003B000	00001000			Priv RW			
00040000	00001000	UnPackMe		Image RW	Cop	RWE	
00040100	0000A000	UnPackMe	.text code	Image RW	Cop	RWF	
00044E00	0000C000	UnPackMe	.rdata .data	Image RW	Cod		
00045700	00009000	UnPackMe	.data	Image RW	Cod		
00046000	00003000	UnPackMe	.idata	Image RW	Cod		
00046300	0000F000	UnPackMe	.rsrc resources	Image RW	Cod		
00046600	0002F000	UnPackMe	.Prf SFX, imports	Image RW	Cod		
0004A000	00004000			Map R	R mm		
00056000	00002000			Map R	R		
00057000	00103000			Map R	R		
00068000	00007F000			Map R	E		
00098000	00001000			Priv RW			
629C0000	00001000	LPK	PE header	Image RW			
629C1000	00005000	LPK	code, import	Image RW			
629C6000	00001000	LPK	.text .data	Image RW			
629C7000	00001000	LPK	.data .rsrc	Image RW			
629D0000	00001000	LPK	.relloc resources	Image RW			
6B0D0000	00001000	SYNCDIR11	PE header	Image RW			
6B0D1000	00002000	SYNCDIR11	code, import	Image RW			
6B0D3000	00001000	SYNCDIR11	.text .data	Image RW			

حالا کلید F9 را بزنید تا به OEP برسید:

004271B0	55	PUSH EBP			
004271B1	3BEC	MOV EBP,ESP			
004271B3	6A FF	PUSH -1			
004271B5	68 600E4500	PUSH UnPackMe.00450E60			
004271B9	68 C8924200	PUSH UnPackMe.004292C8			
004271BF	64:A1 00000000	MOV EAX,DWORD PTR FS:[0]			
004271C1		PUSH EAX			
004271C3	54:8925 00000000	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	CALL DWORD PTR SS:[EBP-18],ESP			
004271D6	FF15 DC0A4600	CALL DWORD PTR DS:[460ADC]			kernel32.GetVersion
004271DC	33D2	XOR EDX,EDX			
004271DE	8AD4	MOV DL,AH			
004271E0	8915 34E64500	MOV DWORD PTR DS:[45E634],EDX			
004271E6	8BC8	MOV ECX,EBX			
004271E8	81E1 FF000000	AND ECX,0FF			
004271EE	8990 30E64500	MOV DWORD PTR DS:[45E630],ECX			
004271F4	C1E1 08	SHL ECX,8			
004271F7	03CA	ADD ECX,EDX			
004271F9	8990 2CE64500	MOV DWORD PTR DS:[45E62C],ECX			

از این به بعد دیگر مشکلی نیست؛ دامپ می گیرید و فیکس می کنید.

NoName Packer 2

فایل مورد نظر را در OllyDBG باز کنید، سه بار کلید F8 را بزنید تا دستور PushAD اجرا شود و بعد روی مقدار رجیستر ESP یک Hardware Breakpoint on access بگذارید. و کلید F9 را بزنید. برنامه در اینجا متوقف می شود:

Address	Hex dump	Disassembly
004C354F	90	POPF
004C3550	50	PUSH EAX
004C3551	68 30ED4800	PUSH 0048ED30
004C3556	C2 0400	RET 4
004C3559	8B85 5B974000	MOV ESI, DWORD PTR SS:[EBP+40975B]
004C355F	0BF6	OR ESI, ESI
004C3561	74 18	JE SHORT 004C357B
004C3563	8B95 E6904000	MOV EDI, DWORD PTR SS:[EBP+4090E6]
004C3569	03F2	ADD ESI, EDI
004C356B	E8 0F000000	CALL 004C357F
004C3570	72 0B	JB SHORT 004C357D
004C3572	83C6 14	ADD ESI, 14
004C3575	837E 0C 00	CMP DWORD PTR DS:[ESI+0], 0
004C3579	75 F0	JNZ SHORT 004C356B
004C357B	F8	CLC
004C357C	C3	RET
004C357D	F9	STC
004C357E	C3	RET
004C357F	C785 31974000 00000000	MOV DWORD PTR SS:[EBP+409731], 0
004C3589	8B0E	MOV ECX, DWORD PTR DS:[ESI]
004C358B	8B7E 10	MOV EDI, DWORD PTR DS:[ESI+10]
004C358E	0BC9	OR ECX, ECX
004C3590	75 02	JNZ SHORT 004C3594
004C3592	8BCF	MOV ECX, EDI

چهار بار کلید F8 را بزنید تا به اینجا برسید:

Address	Hex dump	Disassembly
0048ED30	55	PUSH EBP
0048ED31	90	NOP
0048ED32	8BEC	MOV EBP, ESP
0048ED34	90	NOP
0048ED35	B9 07000000	MOV ECX, 7
0048ED3A	90	NOP
0048ED3B	6A 00	PUSH 0
0048ED3D	6A 00	PUSH 0
0048ED3F	90	NOP
0048ED40	90	NOP
0048ED41	49	DEC ECX
0048ED42	E9 8EFCFFFF	JMP 0048E9D5
0048ED47	0000	ADD BYTE PTR DS:[EAX], AL
0048ED49	55	PUSH EBP
0048ED4A	90	NOP
0048ED4B	8BEC	MOV EBP, ESP
0048ED4D	90	NOP
0048ED4E	83C4 D4	ADD ESP, -2C
0048ED51	90	NOP
0048ED52	33C9	XOR ECX, ECX
0048ED54	90	NOP
0048ED55	E9 9ACFFFFF	JMP 0048E9C4
0048ED5A	0000	ADD BYTE PTR DS:[EAX], AL
0048ED5C	0000	ADD BYTE PTR DS:[EAX], AL
0048ED5E	0000	ADD BYTE PTR DS:[EAX], AL

اینجا باید OEP باشد... اما کدهای زیادی در اینجا موجود نیست. پس بهتر است با F8 برنامه را ادامه بدهیم تا بینم بقیه کدها کجاست؟... چند بار کلید F8 را بزنید تا به خط 48ED42 برسید، این پرش شما را به سایر کدهای برنامه می برد:

0048E9C9	0000	ADD BYTE PTR DS:[EAX], AL
0048E9CB	0000	ADD BYTE PTR DS:[EAX], AL
0048E9CD	0000	ADD BYTE PTR DS:[EAX], AL
0048E9CF	006A 00	ADD BYTE PTR DS:[EDX], CH
0048E9D2	6A 00	PUSH 0
0048E9D4	49	DEC ECX
0048E9D5	75 F9	JNZ SHORT 0048E9D0
0048E9D7	B8 F8E64800	MOV EAX, 0048E6F8
0048E9DC	E8 A37AF7FF	CALL 00406484
0048E9E1	33C0	XOR EAX, EAX
0048E9E3	55	PUSH EBP
0048E9E4	68 47EC4800	PUSH 0048EC47
0048E9E9	64:FF30	PUSH DWORD PTR FS:[EAX]
0048E9EC	64:8920	MOV DWORD PTR FS:[EAX], ESP
0048E9EF	8D55 E8	LEA EDI, DWORD PTR SS:[EBP-18]
0048E9F2	A1 C0174900	MOV EAX, DWORD PTR DS:[4917C0]
0048E9F7	8B00	MOV EAX, DWORD PTR DS:[EAX]
0048E9F9	E8 1AD3FDFD	CALL 0046BD18
0048E9FE	8B45 E8	MOV EAX, DWORD PTR SS:[EBP-18]
0048EA01	8D55 EC	LEA EDI, DWORD PTR SS:[EBP-14]
0048EA04	E8 0BE5FDFD	CALL 0046CF14
0048EA09	8D45 EC	LEA EAX, DWORD PTR SS:[EBP-14]
0048EA0C	BA 5CEC4800	MOV EDI, 0048EC5C
0048EA11	E8 7E5EF7FF	CALL 00404894
0048EA16	8B45 EC	MOV EAX, DWORD PTR SS:[EBP-14]
0048EA19	E8 AEE4FDFD	CALL 0046CECC
0048EA1E	84C0	TEST AL, AL
0048EA20	0F84 75010000	JE 0048EB9B
0048EA26	8D55 E0	LEA EDI, DWORD PTR SS:[EBP-20]
0048EA29	A1 C0174900	MOV EAX, DWORD PTR DS:[4917C0]
0048EA2E	8B00	MOV EAX, DWORD PTR DS:[EAX]

می بینید؟..سایر کدهای ما در اینجا قرار دارد. این یعنی این پکر از قابلیت Stolen Bytes استفاده می کند. یعنی پکر می آید و چند دستور ابتدایی فایل ما را از ابتدا برمی دارد و در جایی دیگر اجرا می کند...حالا باید چه کار کنیم؟...ما باید بایت های دزدیده شده را از آنجا برداریم و به جای اصلیش منتقل کنیم، یک بار دیگر نگاهی به این تصویر بندازید:

Address	Hex dump	Disassembly
0048ED30	55	PUSH EBP
0048ED31	90	NOP
0048ED32	8BEC	MOV EBP, ESP
0048ED34	90	NOP
0048ED35	B9 07000000	MOV ECX, 7
0048ED3A	90	NOP
0048ED3B	6A 00	PUSH 0
0048ED3D	6A 00	PUSH 0
0048ED3F	90	NOP
0048ED40	90	NOP
0048ED41	49	DEC ECX
0048ED42	E9 8EFCFFFF	JMP 0048E9D5
0048ED47	0000	ADD BYTE PTR DS:[EAX], AL
0048ED49	55	PUSH EBP
0048ED4A	90	NOP
0048ED4B	8BEC	MOV EBP, ESP
0048ED4D	90	NOP
0048ED4E	83C4 D4	ADD ESP, -2C
0048ED51	90	NOP
0048ED52	33C9	XOR ECX, ECX
0048ED54	90	NOP
0048ED55	E9 9ACFFFFF	JMP 0048E9F4
0048ED5A	0000	ADD BYTE PTR DS:[EAX], AL
0048ED5C	0000	ADD BYTE PTR DS:[EAX], AL
0048ED5E	0000	ADD BYTE PTR DS:[EAX], AL

این دستورات دزدیده شده ما هستند که در بین این دستورات از خطوط 0048ED3B تا خط 0048ED41 برای خالی کردن پنجره Stack (قرار دادن مقدارهای صفر در آن) استفاده می شود. و پرشی که در خط 0048ED42 هم ما را به کدهای اصلی برنامه می برد. دستورات NOP هم که هیچ تاثیری ندارد. پس در واقع بایت های دزدیده شده ما همین سه دستور می باشد:

```

0048ED30      55      PUSH EBP
0048ED32      8BEC     MOV EBP, ESP
0048ED35      B9 07000000  MOV ECX, 7

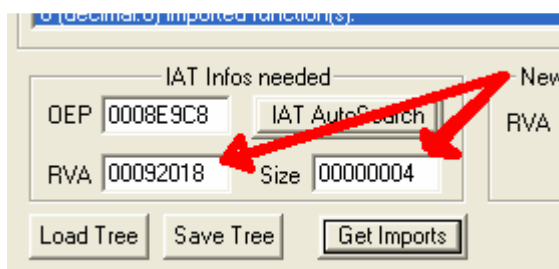
```

حالا کلیدهای CTRL + A را بزنید تا کدهای این سکشن آنالیز شود و بعد این سه دستور را از خط 0048E9C8 به سایر کدهای برنامه اضافه می کنیم تا این مشکل هم حل شود :

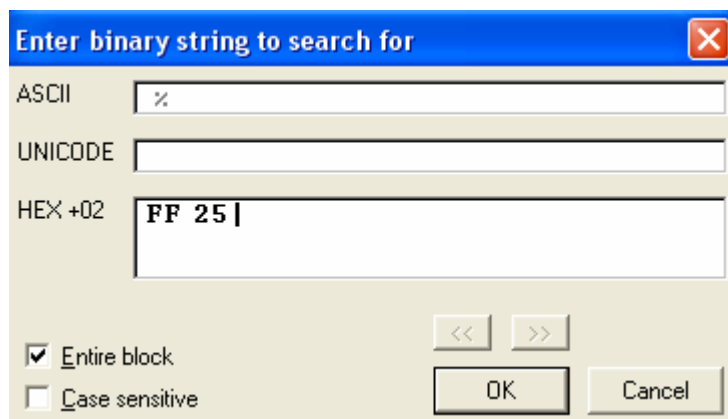
0048E9C3	00	DB 00
0048E9C4	D0E64800	DD NfoUIew.0048E6D0
0048E9C8	55	PUSH EBP
0048E9C9	8BEC	MOV EBP, ESP
0048E9CB	B9 07000000	MOV ECX, 7
0048E9D0	6A 00	PUSH 0
0048E9D2	6A 00	PUSH 0
0048E9D4	49	DEC ECX
0048E9D5	75 F9	JNZ SHORT 0048E9D0
0048E9D7	B8 F8E64800	MOV EAX, 0048E6F8
0048E9DC	E8 A37AF7FF	CALL 00406484
0048E9E1	33C0	XOR EAX, EAX
0048E9E3	55	PUSH EBP
0048E9E4	68 47EC4800	PUSH 0048EC47
0048E9E9	64:FF30	PUSH DWORD PTR FS:[EAX]
0048E9EC	64:8920	MOV DWORD PTR FS:[EAX], ESP
0048E9EF	8D55 E8	LEA EDI, DWORD PTR SS:[EBP-18]
0048E9F2	A1 C0174900	MOV EAX, DWORD PTR DS:[4917C0]
0048E9F7	8B00	MOV EAX, DWORD PTR DS:[EAX]
0048E9F9	E8 1AD3FDFF	CALL 0046BD18
0048E9FF	8B45 F8	MOV EAX, DWORD PTR SS:[EBP-18]

حالا چون اولین دستور ما در خط 48E9C8 قرار دارد. پس OEP ما هم اینجا است. پس بر روی این خط کلیک راست کرده و گزینه New origin here را می زنیم.

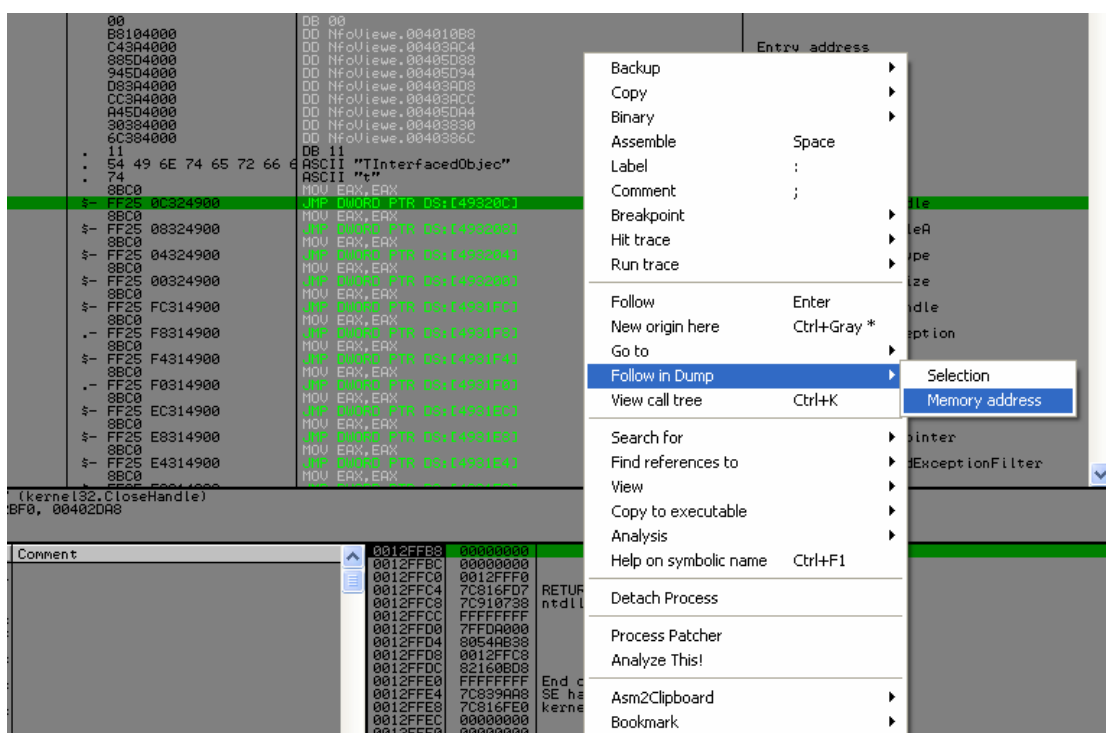
توضیح: گزینه New Origin Here مقدار رجیستر EIP را تغییر می دهد. و چون رجیستر EIP نشان دهنده خطی است که قرار است اجرا شود. ما با تغییر این رجیستر می توانیم روند اجرا شدن کدها را تغییر دهیم. ☺
و بعد هم با پلاگین OllyDump از فایلمان دامپ می گیریم، حالا ImportREC را باز کنید. OEP را به برنامه بدهید (8E9C8) و سایر مراحل را انجام دهید:



همانطور که می بینید ImportREC نتوانسته IAT را درست پیدا کند. پس باید خودمان دست به کار بشویم و IAT را پیدا کنیم. در پنجره OllyDBG کلیدهای CTRL + B را بزنید و تایپ کنید FF25 تا بتوانیم Jump Table را پیدا کنیم:



حالا هم کلیک راست کرده و گزینه Follow in dump و سپس گزینه Memory Address را بزنید:



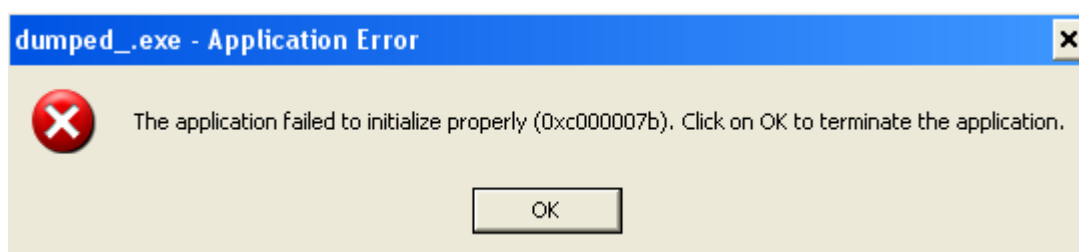
به این ترتیب IAT را پیدا کردیم. بهتر است شروع و پایان IAT را به دست بیاوریم.

Address	Value	ASCII	Comment
004931D4	7C8137D9	7u:	kernel32.FindFirstFileA
004931D8	7C80E0D7	7u:	kernel32.FindClose
004931DC	7C81CDDA	7u:	kernel32.ExitProcess
004931E0	7C810D87	7u:	kernel32.WriteFile
004931E4	7C862E2A	7u:	kernel32.UnhandledExceptionFilter
004931E8	7C810B8E	7u:	kernel32.SetFilePointer
004931EC	7C832044	7u:	kernel32.SetEndOfFile
004931F0	7C937A40	7u:	ntdll.RtlUnwind
004931F4	7C80180E	7u:	kernel32.ReadFile
004931F8	7C812A09	7u:	kernel32.RaiseException
004931FC	7C812F39	7u:	kernel32.GetStdHandle
00493200	7C810A77	7u:	kernel32.GetFileSize
00493204	7C810E51	7u:	kernel32.GetFileType
00493208	7C801A24	7u:	kernel32.CreateFileA
0049320C	7C809B47	7u:	kernel32.CloseHandle
00493210	00000000	7u:	user32.GetKeyboardType
00493214	7E43119B	7u:	user32.LoadStringA
00493218	7E42DFA8	7u:	user32.MessageBoxA
0049321C	7E45058A	7u:	user32.CharNextA
00493220	7E42DF50	7u:	advapi32.RegQueryValueExA
00493224	77DD7883	7u:	advapi32.RegOpenKeyExA
00493228	77DD761B	7u:	advapi32.RegCloseKey
00493230	77DD6BF0	7u:	oleaut32.SysFreeString
00493234	00000000	7u:	
00493238	77124880	7u:	

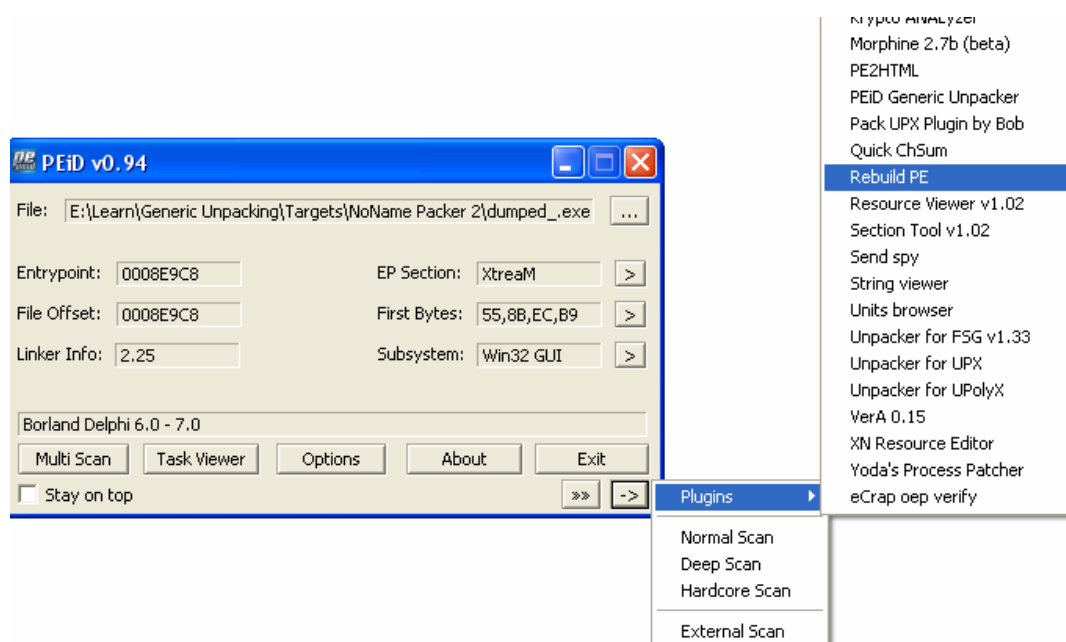
من در مثال های قبل طریقه به دست آوردن IAT را توضیح دادم. پس اینجا هیچ توضیحی نمی دهم و فقط اطلاعات را می دهم:

IAT Start: 493164(RVA: 93164)
 IAT End: 493854(RVA: 93854)
 Size of IAT: 6F0

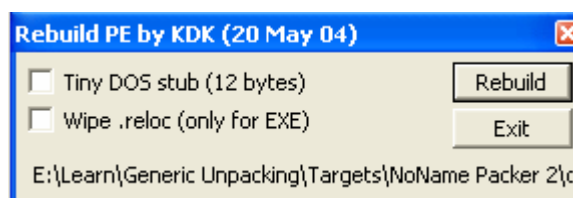
حالا این اطلاعات را در ImportREC وارد کنید و کلید Get Imports را بزنید، و بعد فایل دامپ شده خود را فیکس کنید. حالا فایل آپیک شده خود را اجرا کنید:



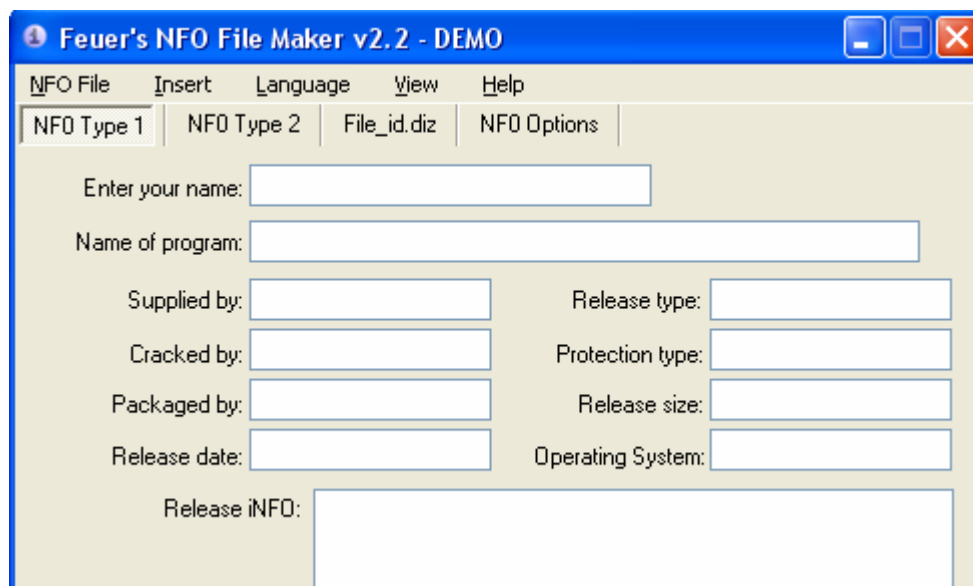
فایل اجرا نمی شود، اما مشکلی نیست. ما با ساختن دوباره فایل آپیک شده می توانیم مشکل را حل کنیم (Rebuild PE). برای ساختن دوباره فایل می توانیم از هر نرم افزاری که دلمان خواست استفاده کنیم. اما در اینجا برای تنوع از PEiD استفاده می کنیم. برای این کار فایل مورد نظر را در PEiD باز کنید، و همانند تصویر زیر عمل کنید:



حالا هم گزینه Rebuild را بزنید:



دوباره فایل را اجرا کنید:



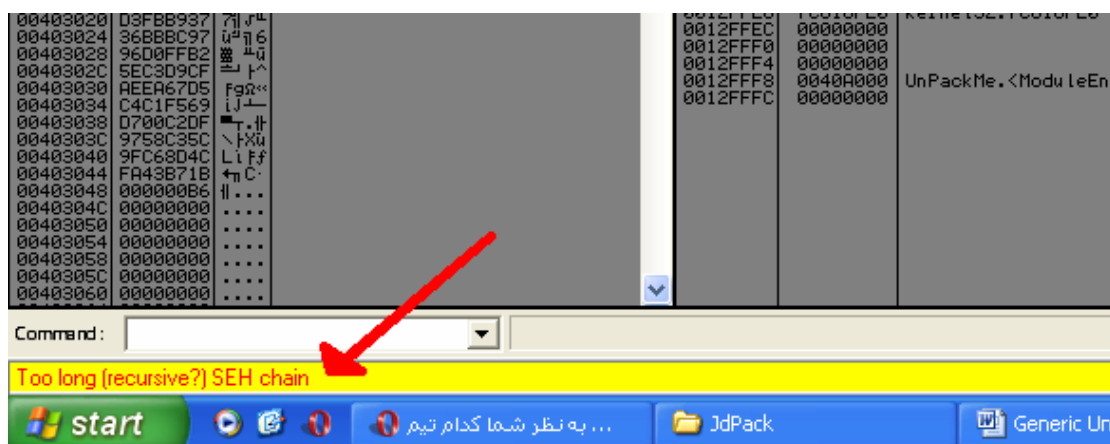
The screenshot shows the 'Feuer's NFO File Maker v2.2 - DEMO' application window. It features a menu bar with 'NFO File', 'Insert', 'Language', 'View', and 'Help'. Below the menu is a tabbed interface with four tabs: 'NFO Type 1' (selected), 'NFO Type 2', 'File_id.diz', and 'NFO Options'. The main area contains several input fields for creating an NFO file:

- 'Enter your name:' followed by a text box.
- 'Name of program:' followed by a text box.
- 'Supplied by:' and 'Release type:' each followed by a text box.
- 'Cracked by:' and 'Protection type:' each followed by a text box.
- 'Packaged by:' and 'Release size:' each followed by a text box.
- 'Release date:' and 'Operating System:' each followed by a text box.
- 'Release iNFO:' followed by a large text area.

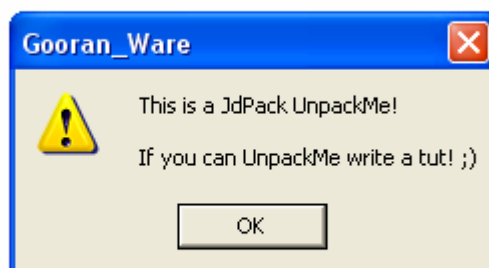
این بار فایل بی هیچ مشکلی اجرا شد.

JDPack

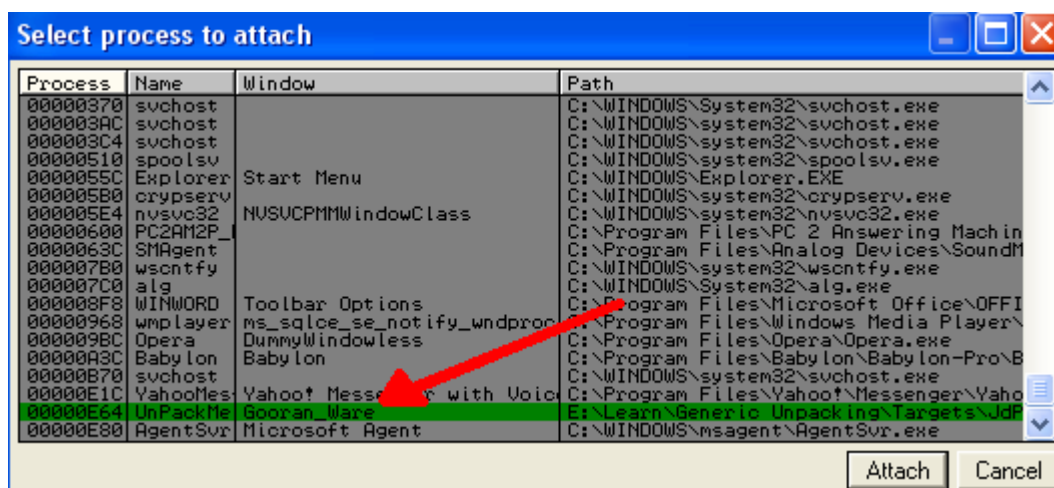
فایل مورد نظر را در OllyDBG باز کنید و بعد کلید F9 را بزنید تا برنامه اجرا شود.



همانطور که می بینید فایل مورد نظر به سختی در OllyDBG دیباگ می شود و خیلی دیر اجرا می شود (شاید این یک آنتی دیباگ باشد). (هر چند من معتقدم که این آنتی دیباگ نیست، چون فایل در داخل ویندوز هم دیر اجرا می شود) خوب... حالا باید چه کار کنیم؟... بی خیال بشیم؟... نه، همیشه یک راه حلی هست. کلید ALT + F2 را بزنید تا برنامه در داخل OllyDBG بسته شود. حالا برنامه را از طریق خود ویندوز اجرا کنید:



بعد از اجرا کردن برنامه توسط ویندوز، حالا به پروسه این فایل با OllyDBG ، Attach می کنیم. در OllyDBG در منوی File گزینه Attach را می زنیم و بعد پروسه مورد نظرمان را انتخاب می کنیم:



حالا گزینه Attach را بزنید تا به پروسه بچسبید.

Address	Hex dump	Disassembly
7C901231	C3	RET
7C901232	8BFF	MOV EDI,EDI
7C901234	90	NOP
7C901235	90	NOP
7C901236	90	NOP
7C901237	90	NOP
7C901238	90	NOP
7C901239	CC	INT3
7C90123A	C3	RET
7C90123B	90	NOP
7C90123C	8BFF	MOV EDI,EDI
7C90123E	90	NOP
7C90123F	90	NOP
7C901240	90	NOP
7C901241	90	NOP
7C901242	90	NOP
7C901243	8B4424 04	MOV EAX,DWORD PTR SS:[ESP+4]
7C901247	CC	INT3
7C901248	C2 0400	RET 4
7C90124B	90	NOP
7C90124C	90	NOP
7C90124D	90	NOP
7C90124E	90	NOP
7C90124F	90	NOP
7C901250	64:A1 18000000	MOV EAX,DWORD PTR FS:[18]
7C901256	C3	RET
7C901257	90	NOP
7C901258	90	NOP
7C901259	90	NOP
7C90125A	90	NOP
7C90125B	90	NOP

حالا در منوی View گزینه Executable modules را بزنید و بر روی پروسه مورد نظرتان دو بار کلیک کنید:

File version	Path
6.0.0.32	E:\Learn\Generic Unpacking\Targets\JdPack\UnPackMe.exe
6.00.2900.2180	C:\Program Files\Babylon\Babylon-Pro\CAPTLib.DLL
1.0.0.2	C:\WINDOWS\system32\uxtheme.dll
5.1.2600.2180	C:\WINDOWS\system32\ole32.dll
4.2.5406.0 (x60)	C:\WINDOWS\system32\OLEACC.dll
1.0420.2600.2180	C:\WINDOWS\system32\USP10.dll
6.02.3104.0	C:\WINDOWS\system32\MSUCP60.dll
5.1.2600.3139	C:\WINDOWS\system32\OLEAUT32.dll
5.1.2600.2726	C:\WINDOWS\system32\ole32.dll
7.0.2600.2180	C:\WINDOWS\system32\msvcp.dll
5.1.2600.2180	C:\WINDOWS\system32\ADVAPI32.dll
5.1.2600.2180	C:\WINDOWS\system32\RPCRT4.dll
5.1.2600.3159	C:\WINDOWS\system32\GDI32.dll
6.00.2900.3157	C:\WINDOWS\system32\SHLWAPI.dll
7.10.3052.4	C:\Program Files\Yahoo!\Messenger\MSUCR71.dll
5.1.2600.3119	C:\WINDOWS\system32\kernel32.dll
5.1.2600.2180	C:\WINDOWS\system32\ntdll.dll
5.1.2600.3099	C:\WINDOWS\system32\user32.dll

Hex dump	Disassembly	Comment
6A 30	PUSH 00403000	ASCII "Gooran_Ware"
68 0C304000	PUSH 00403000	ASCII "This is a JdPack UnpackMe!"
6A 00	PUSH 0	
E8 07000000	CALL 0040101A	JMP to user32.MessageBoxA
6A 00	PUSH 0	
E8 06000000	CALL 00401020	JMP to kernel32.ExitProcess
FF25 00204000	JMP DWORD PTR DS:[00204000]	user32.MessageBoxA
FF25 00204000	JMP DWORD PTR DS:[00204000]	kernel32.ExitProcess

کلیدهای CTRL + A را بزنید تا کدها آنالیز شود.

Address	Hex dump	Disassembly	Comment
00401000	6A 30	PUSH 00403000	Title = "Gooran_Ware"
00401007	68 0C304000	PUSH 00403000	Text = "This is a JdPack UnpackMe!"
0040100C	6A 00	PUSH 0	hOwner = NULL
0040100E	E8 07000000	CALL 0040101A	MessageBoxA
00401013	6A 00	PUSH 0	ExitCode = 0
00401015	E8 06000000	CALL 00401020	ExitProcess
0040101A	FF25 00204000	JMP DWORD PTR DS:[00204000]	user32.MessageBoxA
00401020	FF25 00204000	JMP DWORD PTR DS:[00204000]	kernel32.ExitProcess

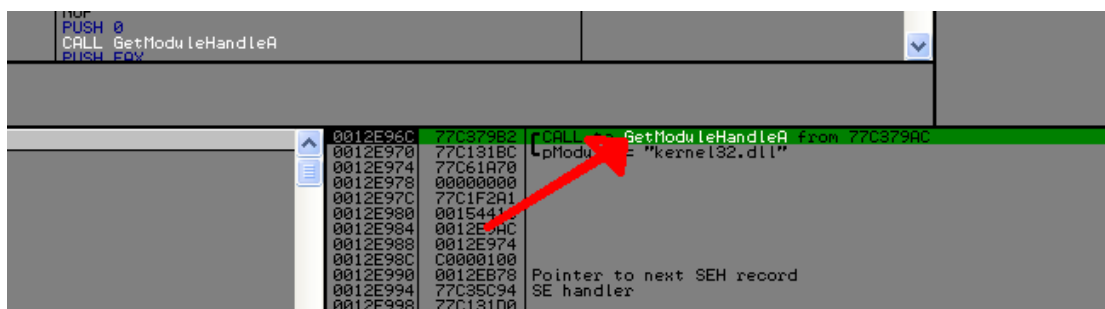
همانطور که می بینید برنامه ما در MASM نوشته شده و همین چند خط کد را دارد. با تابع MessageBoxA یک تابع را نشان می دهد و بعد با تابع ExitProcess برنامه بسته می شود. پس OEP ما اولین خط در سکشن اصلی فایل می باشد. یک خط 00401000 بر روی خط 00401000 کلیک راست کرده و گزینه New origin here را بزنید. حالا از فایل دامپ بگیرید. و بعد هم فیکس کنید.

XXPack

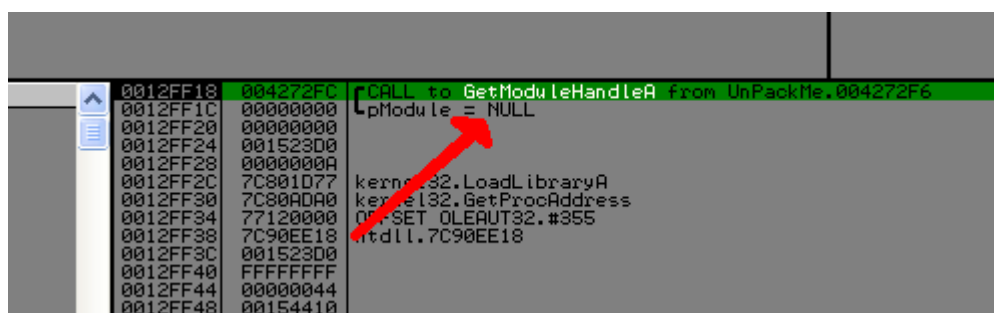
برای یافتن OEP می توانیم از ساختار زبان برنامه نویسی استفاده کنیم. با فرض اینکه برنامه ما در Delphi یا Visual C++ نوشته شده باشد. (در ویژوال بیسیک که نوشته نشده، چون فایل MSVBVM60.dll اجرا نشده.)
ما می توانیم بر روی تابع GetModuleHandleA یک Hardware Breakpoint on execution بگذاریم و بعد OEP را پیدا کنیم. چون معمولاً در نزدیکی های OEP در C و دلفی تابع GetModuleHandleA اجرا می شود.
کلیدهای CTRL + G را بزنید. و تایپ کنید : GetModuleHandleA



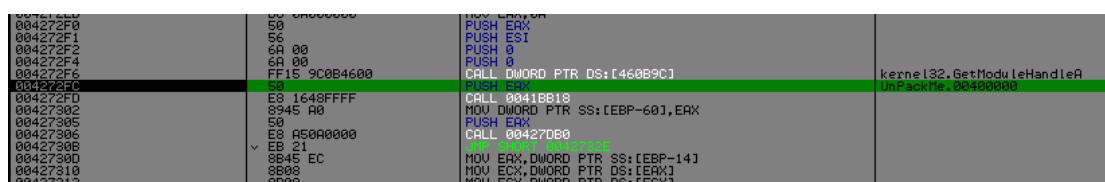
و بعد کلیک راست کرده، گزینه Breakpoint و سپس گزینه Hardware, on execution را بزنید. حالا کلید F9 را بزنید:



پارامتر Pmodule در اینجا Kernel32.dll است. این پارامتر باید Null باشد. پس این چیزی نیست که ما می خواهیم. آنقدر کلید F9 را بزنید تا به اینجا برسید:



در اینجا پارامتر Pmodule برابر Null است. پس اینجا همانجایی هست که می خواهیم. حالا کلید ALT + F9 را بزنید تا وارد کدهای فایل خودمان بشویم.



اینجا نزدیکی های OEP است. مقداری به بالا بروید تا سایر توابعی که در نزدیکی های OEP می آیند را ببینید و بعد به خود OEP رسید:

AD		90	NOP
AE		90	NOP
AF		90	NOP
B0		55	PUSH EBP
B1		8BEC	MOV EBP,ESP
B3		6A FF	PUSH -1
B5		68 600E4500	PUSH 00450E60
B9		68 C8924200	PUSH 004292C8
BF		64:A1 00000000	MOV EAX,DWORD PTR FS:[0]
C5		50	PUSH EAX
C6		64:8925 00000000	MOV DWORD PTR FS:[0],ESP
CD		83C4 A8	ADD ESP,-58
D0		53	PUSH EBX
D1		56	PUSH ESI
D2		57	PUSH EDI
D3		8965 E8	MOV DWORD PTR SS:[EBP-18],ESP
D6		FF15 DC0A4600	CALL DWORD PTR DS:[460ADC]
DC		33D2	XOR EDX,EDX
DE		8AD4	MOV DL,AH

حالا کلیک راست کرده و گزینه new origin here را بزنید و بعد از فایل دامپ بگیرید(ترجیحا با LordPE این کار را بکنید.) و فیکس کنید

MJo0T MIV InfoViewer

برنامه را در OllyDBG باز کنید. ۳ بار کلید F8 را بزنید تا به دستور INT3 برسید.

00417F19	<ModuleEntryPoint>	68 3D7F4100	PUSH 00417F3D
00417F1E		64:FF35 00000000	PUSH DWORD PTR FS:[0]
00417F25		64:8925 00000000	MOV DWORD PTR FS:[0],ESP
00417F2C		CD 03	INT 3
00417F2E		8B0424	MOV EAX,DWORD PTR SS:[ESP]
00417F31		64:A3 00000000	MOV DWORD PTR FS:[0],EAX
00417F37		83C4 08	ADD ESP,8
00417F3A		C3	RET
00417F3B		0000	ADD BYTE PTR DS:[EAX],AL
00417F3D		C7C6 BD8C1F66	MOV ESI,661F8C8D
00417F43		0FBAE9 1C	BTS ECX,1C
00417F47		89EE	MOV ESI,EBP
00417F49		B4 DB	MOV AH,0DB
00417F4B		86D5	XCHG CH,DL
00417F4D		0FC9	BSWAP ECX
00417F4F		47	INC EDI
00417F50		F3:	PREFIX REP:
00417F51		0FA5D3	SHLD EBX,EDX,CL
00417F54		0FABC1	BTS ECX,EAX
00417F57		0FBCFE	BSF EDI,ESI
00417F5A		0FBAE5 0B	BT EBP,0B

برنامه در این خط دچار خطای INT 3 Break می شود. برنامه بعد از رسیدن به خطا به تابع SEH Handler می رود. به پنجره دامپ نگاه کنید:

0012FFBC	0012FFE0	Pointer to next SEH record
0012FFC0	00417F3D	SE handler
0012FFC4	7C816FD7	RETURN to kernel32.7C816FD7
0012FFC8	7C910738	ntdll.7C910738
0012FFCC	FFFFFFFF	
0012FFD0	7FFD4000	
0012FFD4	8054AB38	
0012FFD8	0012FFC8	
0012FFDC	81C53298	
0012FFE0	FFFFFFFF	End of SEH chain
0012FFE4	7C839AA8	SE handler
0012FFE8	7C816FE0	kernel32.7C816FE0
0012FFEC	00000000	
0012FFF0	00000000	
0012FFF4	00000000	
0012FFF8	00417F19	InfoView.<ModuleEntryPoint>
0012FFFC	00000000	

پس برنامه ما با این خطا به خط 417F3D می رود. بر روی این خط Brekpoint بگذارید و برنامه را با F9 اجرا کنید تا در جایی که BP گذاشتیم، متوقف شویم:

00417F37	83C4 08	ADD ESP,8
00417F3A	C3	RET
00417F3B	0000	ADD BYTE PTR DS:[EAX],AL
00417F3D	C7C6 BD8C1F66	MOV ESI,661F8C8D
00417F43	0FBAE9 1C	BTS ECX,1C
00417F47	89EE	MOV ESI,EBP
00417F49	B4 DB	MOV AH,0DB
00417F4B	86D5	XCHG CH,DL
00417F4D	0FC9	BSWAP ECX
00417F4F	47	INC EDI
00417F50	F3:	PREFIX REP:
00417F51	0FA5D3	SHLD EBX,EDX,CL
00417F54	0FABC1	BTS ECX,EAX
00417F57	0FBCFE	BSF EDI,ESI
00417F5A	0FBAE5 0B	BT EBP,0B
00417F5E	86D5	XCHG CH,DL
00417F60	F7D1	NOT ECX
00417F62	0FCF	BSWAP EDI
00417F64	EB 01	JMP SHORT 00417F67
00417F66	A3 0FB3EA87	MOV DWORD PTR DS:[87EAB30F],EAX
00417F68	C8 D1D680	ENTER 0D6D1,80

حال برنامه را با F8 ادامه دهید تا به این خط برسید:

00417FC5	0FCB	BSWAP EBX
00417FC7	BA 807D4100	MOV EDX,00417D80
00417FCC	8BFA	MOV EDI,EDX
00417FCE	B9 6B000000	MOV ECX,6B
00417FD3	8032 74	XOR BYTE PTR DS:[EDX],74
00417FD6	83C2 01	ADD EDX,1
00417FD9	^ E2 F8	LOOPD SHORT 00417FD3
00417FDB	FFE7	JMP EDI
00417FDD	C1E1 64	SHL ECX,64
00417FE0	D1D6	RCL ESI,1
00417FE2	80DC 83	SBB AH,83
00417FE5	C0CA D1	ROR DL,0D1
00417FE8	26:87F1	XCHG ECX,ESI
00417FEB	88D4	MOV AH,DL
00417FED	E2D3	NOT EBX

اینجا در یک حلقه گیر کردیم، بر روی دستور JMP EDI یک BP بگذارید و کلید F9 را بزنید:

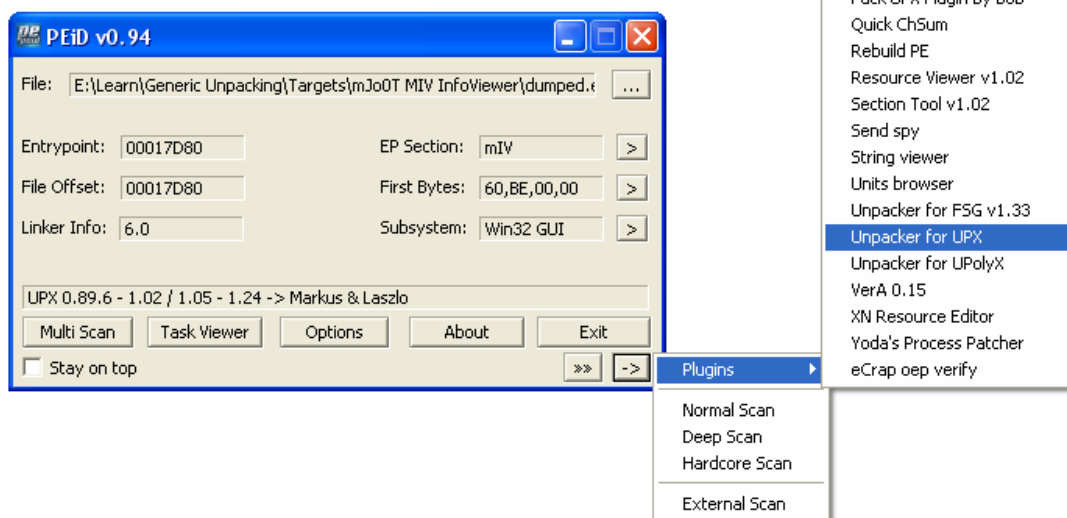
00417FB9	0FACFD 53	SHRD EBP,EDI,53
00417FBD	08C2	OR DL,AL
00417FBF	D1E1	SHL ECX,1
00417FC1	85C3	TEST EBX,EAX
00417FC3	84F1	TEST CL,DH
00417FC5	0FCB	BSWAP EBX
00417FC7	BA 807D4100	MOV EDX,00417D80
00417FCC	8BFA	MOV EDI,EDX
00417FCE	B9 6B000000	MOV ECX,6B
00417FD3	8032 74	XOR BYTE PTR DS:[EDX],74
00417FD6	83C2 01	ADD EDX,1
00417FD9	^ E2 F8	LOOPD SHORT 00417FD3
00417FDB	^ FFE7	JMP EDI
00417FDD	C1E1 64	SHL ECX,64
00417FE0	D1D6	RCL ESI,1
00417FE2	80DC 83	SBB AH,83
00417FE5	C0CA D1	ROR DL,0D1
00417FE8	26:87F1	XCHG ECX,ESI
00417FEB	88D4	MOV AH,DL
00417FED	E2D3	NOT EBX

یکبار F8 بزنید:

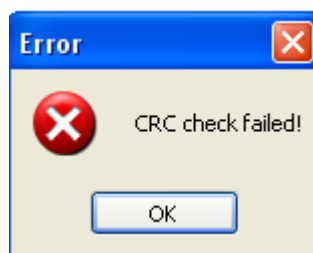
Address	Hex dump	Disassembly
00417D80	60	PUSHAD
00417D81	BE 00004100	MOV ESI,00410000
00417D86	8DBE 0010FFFF	LEA EDI,DWORD PTR DS:[ESI+FFFF1000]
00417D8C	57	PUSH EDI
00417D8D	83CD FF	OR EBP,FFFFFFFF
00417D90	EB 10	JMP SHORT 00417DA2
00417D92	90	NOP
00417D93	90	NOP
00417D94	90	NOP
00417D95	90	NOP
00417D96	90	NOP
00417D97	90	NOP
00417D98	8A06	MOV AL,BYTE PTR DS:[ESI]
00417D9A	46	INC ESI
00417D9B	8807	MOV BYTE PTR DS:[EDI],AL
00417D9D	47	INC EDI
00417D9E	01DB	ADD EBX,EBX
00417DA0	75 07	JNZ SHORT 00417DA9

این دستورات شما را یاد چیزی نمی اندازد؟ بله... این دستوراتی است که در ابتدای UPX قرار دارد. پس این پکر در واقع UPX بوده و Entry point آن دستکاری شده بود... حالا چه کار کنیم؟... همچنین از فایل دامپ می گیریم و با یک UPX Unpacker فایل را آنپک می کنیم ☺

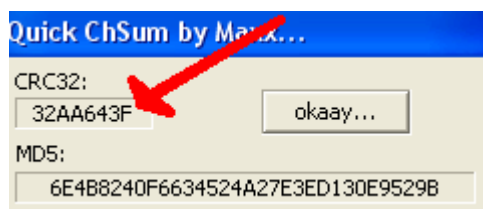
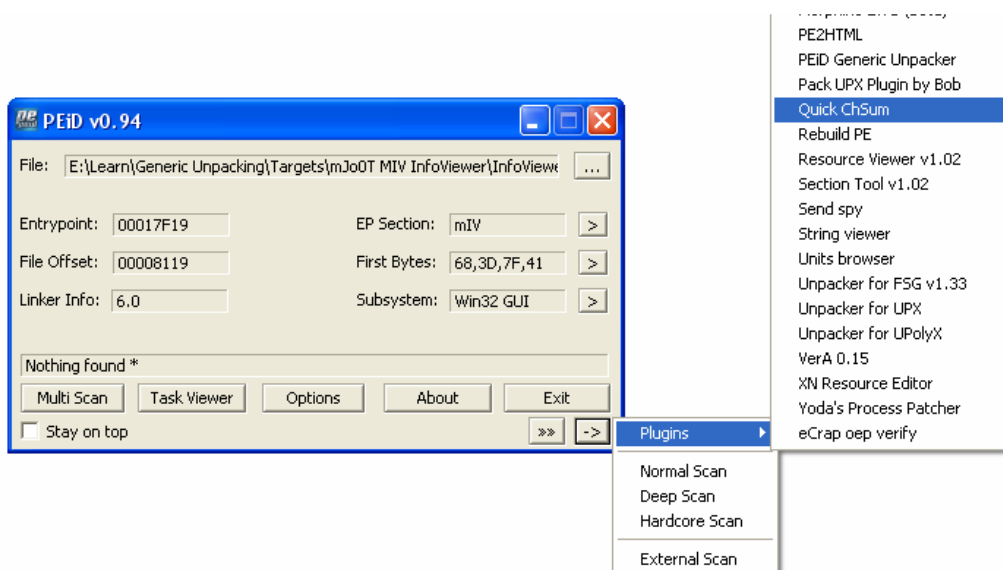
با پلاگین OllyDump از فایل دامپ بگیرید. (تیک گزینه ReBuild Import را بزنید. تا IAT ساخته شود). حالا فایل ما با UPX پک شده، پس با پلاگین PeiD فایل را آنپک می کنیم. فایلی که دامپ گرفتیم را در PeiD باز کنید و همانند تصویر عمل کنید:



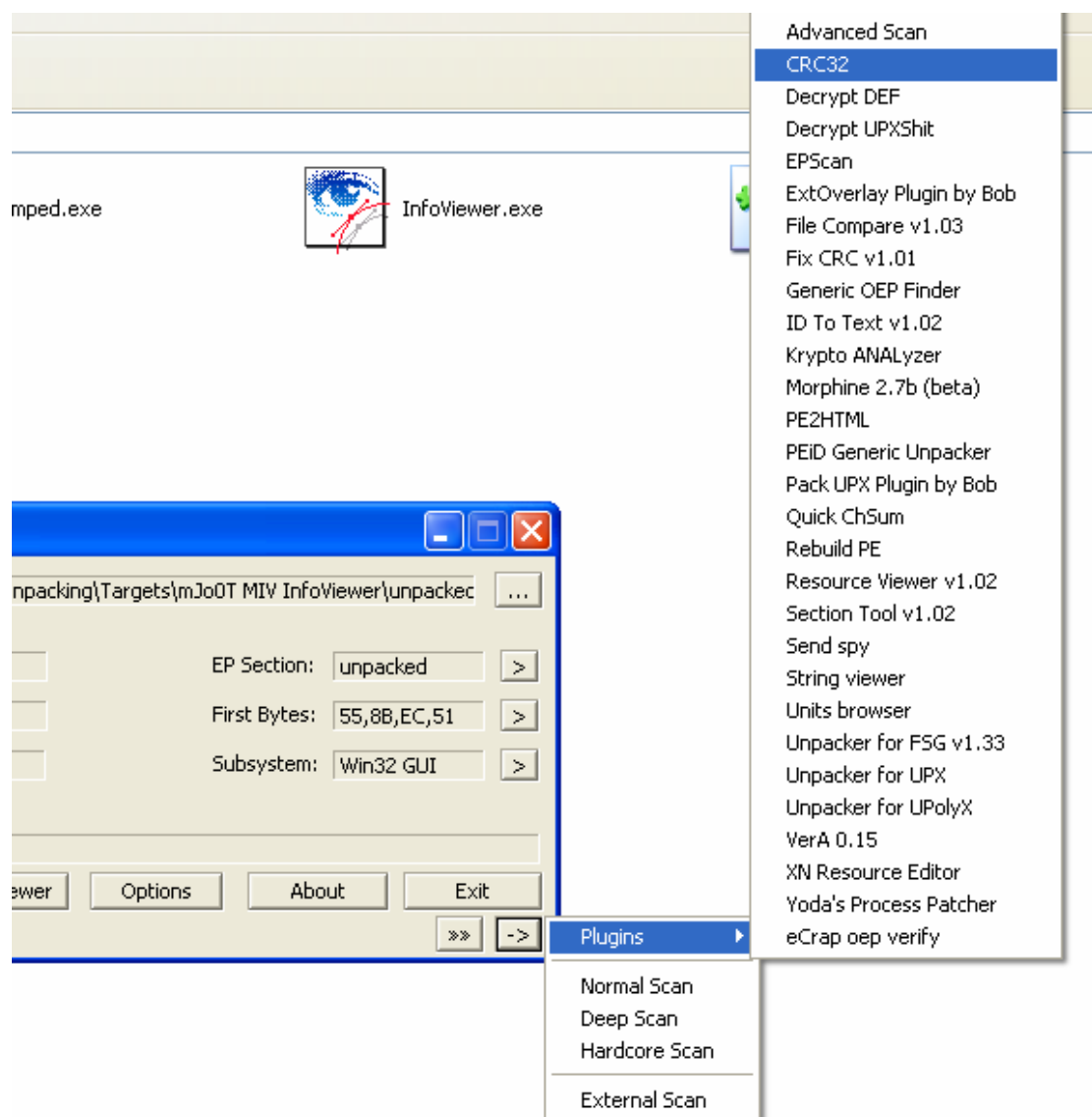
فایل آنپک شده را باز کنید:



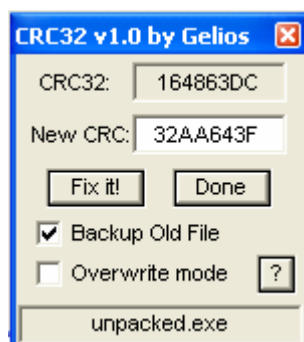
همانطور که می بینید برنامه CRC خودش را چک می کند و اگر بفهمد که فایل دستکاری شده است، جلوی اجرا شدن برنامه را می گیرد و این پیغام را می دهد. حالا چه کار باید بکنیم؟
حالا می توانیم با پلاگین PEiD، CRC فایل را تغییر دهیم. اول باید ببینیم که CRC اصلی فایل چه بوده؟... فایل اصلی (فایل پک شده) را در PEiD باز کنید و همانند تصویر عمل کنید:



پس CRC فایل اصلی ما 32AA643F است. ما CRC فایل آنپک شده خود را تغییر می دهیم، تا به این ترتیب برنامه متوجه دستکاری شدن فایل نشود. حالا فایل آنپک شده را PEiD باز کنید و همانند تصویر عمل کنید:



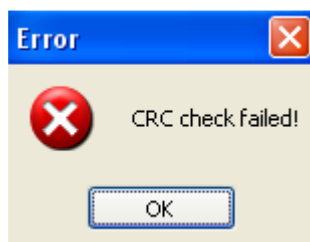
حالا هم CRC را به 32AA643F تغییر می دهید، و گزینه Fix It را بزنید، همانند تصویر:



اینبار فایل آنپک شده را اجرا کنید، می بینید که فایل آنپک شده بی هیچ مشکلی اجرا می شود.

راه حل دیگر:

راه حل دیگر برای تعمیر CRC این است که ما ببینیم در کجا برنامه متوجه تغییر CRC می شود و آنجا را پچ می کنیم. فایل آنپک شده (که CRC آن مشکل دارد.) را در OllyDBG باز کنید. کلید F9 را بزنید تا این پیغام ظاهر شود:



کلید F12 را بزنید تا برنامه متوقف شود و ALT + F9 را بزنید تا وارد حالت Back to user شوید. و بعد کلید OK را بزنید تا این پیغام از بین برود.

100014B5	E8 B8000000	CALL <JMP.&KERNEL32.ReadFile>
100014B8	FF35 04300010	PUSH DWORD PTR DS:[10003004]
100014C0	E8 8F000000	CALL <JMP.&KERNEL32.CloseHandle>
100014C5	FF35 00300010	PUSH DWORD PTR DS:[10003000]
100014CB	FF35 00300010	PUSH DWORD PTR DS:[10003000]
100014D1	E8 2AFBFFFF	CALL 10001000
100014D6	3D 3F64AA32	CMP EAX,32AA643F
100014D8	74 02	JE SHORT 100014DF
100014DD	EB 02	JMP SHORT 100014E1
100014DF	EB 2A	JMP SHORT 1000150B
100014E1	6A 10	PUSH 10
100014E3	68 22300010	PUSH 10003022
100014E8	68 10300010	PUSH 10003010
100014ED	6A 00	PUSH 0
100014EF	E8 5A000000	CALL <JMP.&USER32.MessageBoxA>
100014F4	FF75 08	PUSH DWORD PTR SS:[EBP+8]
100014F7	6A 00	PUSH 0
100014F9	68 FF0F1F00	PUSH 1F0FFF
100014FE	E8 69000000	CALL <JMP.&KERNEL32.OpenProcess>
10001503	6A 01	PUSH 1
10001505	50	PUSH EAX
10001506	E8 6D000000	CALL <JMP.&KERNEL32.TerminateProcess>

همانطور که می بینید از قیافه کدها مشخص است که پرشی که در خط 100014D8 قرار دارد، تصمیم گیرنده است. اگر این پرش انجام شود، CRC مشکلی ندارد و در غیر این صورت با پیغام CRC Check Error روبرو می شویم... حالا چه کار باید بکنیم؟... این پرش را به JMP تغییر دهید و فایل خود را ذخیره کنید. (همان طور که متوجه شدید عملیات بررسی CRC توسط فایل Data.dat انجام می شود)

راه حل دیگر:

چون پکر ما در واقع UPX بوده، ما می توانیم همان کاری که در مورد UPX کردیم، اینجا هم بکنیم. فایل مورد نظر (فایل پک شده) را در OllyDBG باز کنید:

00417F19 <ModuleEntryPoint>	E9 0F32FFFF	JMP 00417F25
00417F1E	68 3D7F4100	PUSH 00417F3D
00417F25	64:FF35 00000000	PUSH DWORD PTR FS:[0]
00417F2C	CD 03	MOV DWORD PTR FS:[0],ESP
00417F2E	8B0424	INT 3
00417F31	64:A3 00000000	MOV EAX,DWORD PTR SS:[ESP]
00417F37	83C4 08	MOV DWORD PTR FS:[0],EAX
00417F3A	C3	ADD ESP,8
00417F3B	0000	RET
00417F3D	C7C6 B08C1F66	ADD BYTE PTR DS:[EAX],AL
00417F43	0FBAE9 1C	MOV ESI,661F8CB0
00417F47	89EE	BTS ECX,1C
00417F49	B4 DB	MOV ESI,EBP
00417F4B	86D5	MOV AH,0DB
00417F4D	0FC9	XCHG CH,DL
00417F4F	47	BSWAP ECX
00417F50	F3:	INC EDI
00417F51	0FA5D3	PREFIX REP:
00417F54	0FABC1	SHLD EBX,EDX,CL
00417F57	0FB0CF	BTS ECX,EAX
00417F5A	0FBAE5 0B	BSF EDI,ESI
00417F5E	86D5	BT EBP,0B
00417F60	F7D1	XCHG CH,DL
00417F62	0FCF	NOT ECX
00417F64	EB 01	BSWAP EDI
00417F66	A3 0FB3EA87	JMP SHORT 00417F67
00417F6B	C8 D1D68A	MOV DWORD PTR DS:[87EAB30F],EAX
00417F6F	E2 0F	ENTER 0D01,8A
00417F71	C1DA D1	LOOPD SHORT 00417F80
00417F74	E1 47	RCR EDX,0D1
		LOOPD SHORT 00417F80

چند خط پایین تر بر روی دستور JMP EDI یک Breakpoint بگذارید و برنامه را با F9 اجرا کنید:

00417FC3	84F1	TEST CL,DH	
00417FC5	0FCB	BSWAP EBX	
00417FC7	BA 807D4100	MOV EDX,00417D80	
00417FCC	8BFA	MOV EDI,EDX	
00417FCE	B9 6B000000	MOV ECX,6B	
00417FD3	8032 74	XOR BYTE PTR DS:[EDX],74	
00417FD6	83C2 01	ADD EDX,1	
00417FD9	E2 F8	LOOPD SHORT 00417FD3	
00417FDB	75 F7	JNE EDI	InfoView.00417D80
00417FDD	C1E1 64	SHL ECX,64	
00417FE0	D1D6	RCL ESI,1	
00417FE2	80DC 83	SBB AH,83	
00417FE5	C0A0 D1	ROR DL,0D1	
00417FE8	26187F1	XCHG ECX,ESI	
00417FEB	88D4	MOV AH,DL	
00417FED	F7D3	NOT EBX	
00417FEF	EB 01	JNE SHORT 00417FF2	
00417FF1	54	PUSH ESP	

یکبار کلید F8 را بزنید:

	Hex dump	Disassembly
	60	PUSHAD
	BE 00004100	MOV ESI,00410000
	8DBE 0010FFFF	LEA EDI,DWORD PTR DS:[ESI+FFFF1000]
	57	PUSH EDI
	83CD FF	OR EBP,FFFFFFFF
▼	EB 10	JMP SHORT 00417DA2
	90	NOP
	90	NOP
	90	NOP
	90	NOP
	90	NOP
	8A06	MOV AL,BYTE PTR DS:[ESI]
	46	INC ESI
	8807	MOV BYTE PTR DS:[EDI],AL
	47	INC EDI
	01DB	ADD EBX,EBX
▼	75 07	JNZ SHORT 00417DA9
	8B1E	MOV EBX,DWORD PTR DS:[ESI]
	83EE FC	SUB ESI,-4
	11DB	ADC EBX,EBX
^	72 ED	JB SHORT 00417D98
	58	MOV EAX,1

خوب، این کدهای ابتدایی UPX است. پس می توانیم همان روشی که برای آنپک کردن UPX استفاده کردیم، اینجا هم استفاده کنیم. یک بار کلید F8 را می زنیم تا مقدار رجیستر ESP تغییر کند و بعد بر روی مقدار رجیستر ESP یک Hardware Breakpoint on access می گذاریم. و بعد کلید F9 را بزنید:

00417F03	FFD5	CALL ESP
00417F05	58	POP EAX
00417F06	61	POPAD
00417F07	8D4424 80	LEA EAX,DWORD PTR SS:[ESP-80]
00417F08	6A 00	PUSH 0
00417F0D	39C4	CMP ESP,EAX
00417F0F	75 FA	JNZ SHORT 00417F08
00417F11	83EC 80	SUB ESP,-80
00417F14	E9 0F92FEFF	JMP 00401128
00417F19 <ModuleEntryPoint>	68 3D7F4100	PUSH 00417F3D
00417F1E	64:FF35 00000000	PUSH DWORD PTR FS:[0]
00417F25	64:8925 00000000	MOV DWORD PTR FS:[0],ESP
00417F2C	CD 03	INT 3
00417F2E	8B0424	MOV EAX,DWORD PTR SS:[ESP]
00417F31	64:A3 00000000	MOV DWORD PTR FS:[0],EAX
00417F37	83C4 08	ADD ESP,8
00417F3A	C3	RET
00417F3B	0000	ADD BYTE PTR DS:[EAX],AL
00417F3D	C7C6 B08C1F66	MOV ESI,661F8C8D
00417F43	0FB9E9 1C	BTS ECX,1C
00417F47	0000	MOV ESI,ESP

پرش 00401128 JMP ما را به OEP می برد. همانطور که می بینید درست زیر این پرش، Entry point، قرار دارد. این یعنی اینکه درست زیر پرشی که به OEP می رود، کدهایی تزریق شده و Entry point به آنجا منتقل شده است. بعد از پیدا کردن OEP از فایل دامپ می گیرید و فیکس می کنید.

راه حل دیگر:

راه حل دیگر برای یافتن OEP این است که ما از ساختار زبان برنامه نویسی استفاده کنیم. فایل مورد نظر را در OllyDBG باز کنید و کلید CTRL + G را بزنید و تایپ کنید: `GetModuleHandleA`. حالا بر روی این تابع کلیک راست کرده و گزینه Breakpoint و سپس گزینه Hardware, on execution را بزنید، تا بر روی این تابع Hardware breakpoint بگذارید.

EAX	79	
ESP	004011A8	
0012FBD4	004011A8	CALL to GetModuleHandleA from InfoView.004011A8
0012FBD8	00000000	lpModule = NULL
0012FBD0	00417D80	InfoView.00417D80
0012FBE0	CBFC2E3B	
0012FBE4	EFA60000	
0012FBE8	004010CF	InfoView.004010CF
0012FBEC	1F81011F	
0012FBF0	7FF5006B	RETURN to 7FF5006B
0012FBF4	0012FCD0	
0012FBF8	0012FFBC	
0012FBFC	0012FCF0	
0012FC00	0012FCA4	
0012FC04	00000000	

پارامتر pModule برابر Null است. پس احتمالاً این تابع در نزدیکی ها OEP اجرا شده است. کلیدهای ALT + F9 را بزنید تا وارد کدهای فایل خودمان بشویم.

00401198	64:8925 00000000	MOV DWORD PTR FS:[0],ESP	
00401199	B8 00000000	MOV EAX,0	
0040119A	C600 01	MOV BYTE PTR DS:[EAX],1	
0040119B	8B0424	MOV EAX,DWORD PTR SS:[ESP]	
0040119C	64:83 00000000	MOV DWORD PTR FS:[0],EAX	
0040119D	83C4 08	ADD ESP,8	
0040119E	6A 00	PUSH 0	
0040119F	FF15 68504000	CALL DWORD PTR DS:[405068]	kernel32.GetModuleHandleA
004011A0	A3 E0634000	MOV DWORD PTR DS:[4063E0],EAX	InfoUw.00400000
004011A1	FF15 6C504000	CALL DWORD PTR DS:[40506C]	kernel32.GetCommandLineA
004011A2	A3 E0634000	MOV DWORD PTR DS:[4063E0],EAX	
004011A3	6A 00	PUSH 0	
004011A4	68 E4114000	PUSH 004011E4	
004011A5	6A 00	PUSH 0	

اینجا در نزدیکی OEP است. چرا که تابع GetCommandLineA هم در اینجا وجود دارد. با موس به بالا بروید تا به ابتدای Routine برسید:

0040111E	FF15 28504000	CALL DWORD PTR DS:[405028]	6D132.CF
00401120	5D	POP EBP	
00401121	C2 1000	RET 10	
00401122	55	PUSH EBP	
00401123	8BEC	MOV EBP,ESP	
00401124	51	PUSH ECX	
00401125	53	PUSH EBX	
00401126	55	PUSH ESI	
00401127	57	PUSH EDI	
00401128	64:A1 18000000	MOV EAX,DWORD PTR FS:[18]	
00401129	3E:8B40 30	MOV EAX,DWORD PTR DS:[EAX+30]	
0040112A	3E:8B40 02	MOV EAX,DWORD PTR DS:[EAX+2]	
0040112B	83E0 01	AND EAX,1	
0040112C	83F8 00	CMPL EAX,0	

همانطور که می بینید خط 00401128، OEP ماست.

Inline Patching

آیا همیشه باید فایل را آنپک کرد؟... آیا همیشه مجبوریم آنپک کنیم؟... خیر، اگر هدف اصلی ما کرک کردن برنامه باشد. ما می توانیم بدون این که فایل را آنپک کنیم اقدام به کرک کردن یا دستکاری آن بکنیم... حالا چه طور این کار را بکنیم؟... همانطور که در ابتدای این مقاله گفتم:

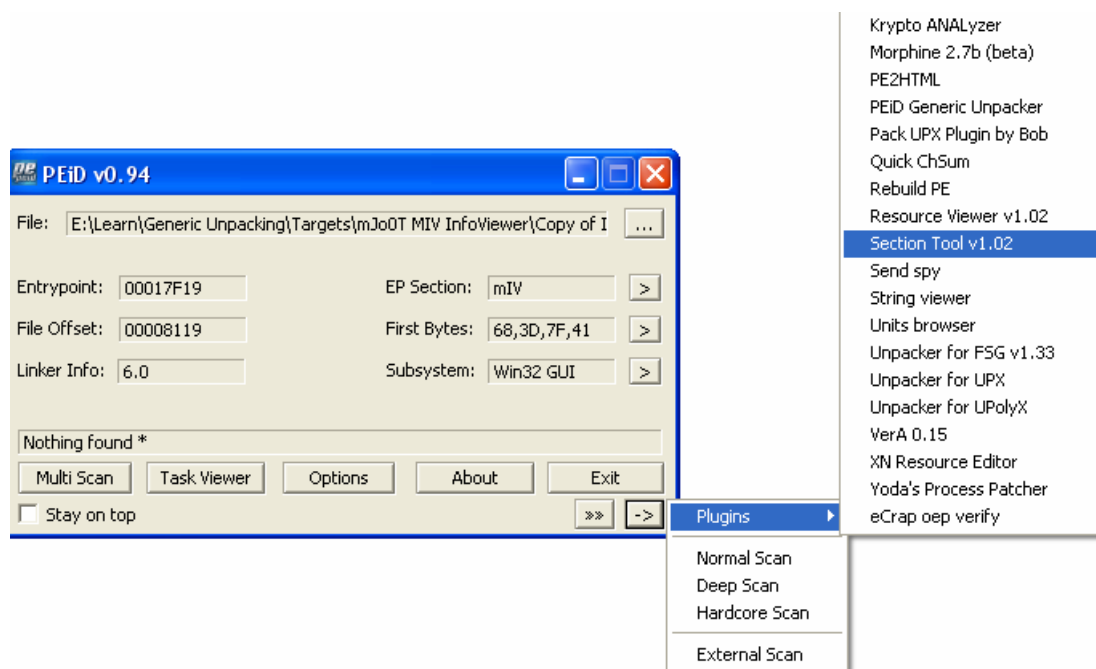
فرقی نمی کند که Packer ما چي باشد، هر چي باشد، به هر حال، در یک جا و در یک زمانی، اختیار از دست Decompression Stub خارج شده و به دست فایل اصلی ما می افتد.

خوب، من می آیم و قبل از اینکه اختیار از دست پکر خارج شود، کدهایی را اضافه می کنم و با این کدها، کدهای فایل اصلی را دستکاری می کنم و به این ترتیب برنامه را کرک می کنم.

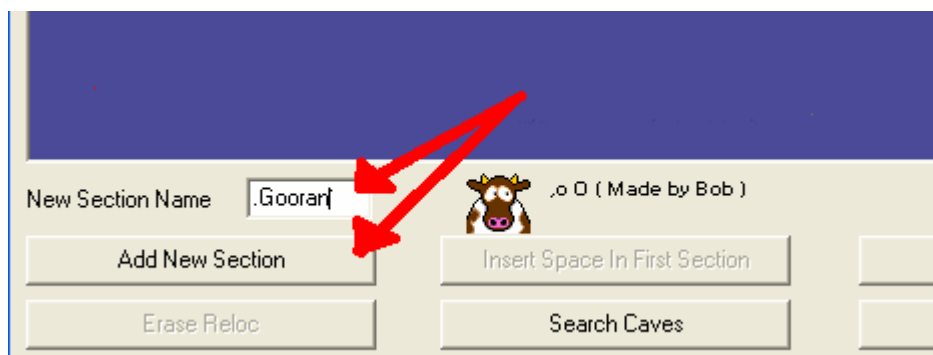
Inline Patch کردن نرم افزار خصوصا در مواردی که حجم فایل بالاست، یکی از بهترین راه حل هاست.

برای مثال ما می خواهیم فایل InfoViewer.exe را که در مثال قبل آنپک کردیم، را Inline patch کنیم. برای این کار باید یک جایی را پیدا کنیم که در آنجا کدهای خودمان را تزریق کنیم... من معمولا برای Inline patch کردن UPX از زیر پرشی که به OEP می رفت استفاده می کردم... اما در این مثال این قسمت با کدهای پکر پر شده ☹ پس نمی توانم این کار را بکنم... بهتر است یک سکشن خالی به برنامه اضافه کنم و کدهایی که لازم دارم را در این سکشن بنویسم.

برنامه را در PeiD باز کنید و همانند تصویر عمل کنید:



در قسمت New Section Name نام سکشن جدید را بنویسید و گزینه Add new section را بزنید و بعد فایل را ذخیره کنید.



حالا فایلی که به آن سکشن اضافه کرده اید را در OllyDBG باز کنید:

Address	Hex dump	Disassembly
0041B000 <ModuleEntryPoint>	E9 14CFFFFF	JMP 00417F19
0041B005	0000	ADD BYTE PTR DS:[EAX],AL
0041B007	0000	ADD BYTE PTR DS:[EAX],AL
0041B009	0000	ADD BYTE PTR DS:[EAX],AL
0041B00B	0000	ADD BYTE PTR DS:[EAX],AL
0041B00D	0000	ADD BYTE PTR DS:[EAX],AL
0041B00F	0000	ADD BYTE PTR DS:[EAX],AL
0041B011	0000	ADD BYTE PTR DS:[EAX],AL
0041B013	0000	ADD BYTE PTR DS:[EAX],AL
0041B015	0000	ADD BYTE PTR DS:[EAX],AL
0041B017	0000	ADD BYTE PTR DS:[EAX],AL
0041B019	0000	ADD BYTE PTR DS:[EAX],AL
0041B01B	0000	ADD BYTE PTR DS:[EAX],AL
0041B01D	0000	ADD BYTE PTR DS:[EAX],AL
0041B01F	0000	ADD BYTE PTR DS:[EAX],AL

کلید F9 را بزنید تا برنامه به طور کامل اجرا شود. این پنجره ظاهر می شود، ما می خواهیم بدون این برنامه را آپیک کنیم، این پنجره را از بین ببریم.



از آنجایی که من نمی خواهم آموزش کرک بدم، بهتون می گم کجاها را باید دستکاری کرد. به خط 4012A7 بروید. در اینجا یک پرش هست که باید Nop شود.

0040129F	52	PUSH EDI
004012A0	E8 EA170000	CALL 00402A8F
004012A5	85C0	TEST EAX,EAX
004012A7	74 29	JE SHORT 004012D2
004012A9	68 20604000	PUSH 00406020
004012AE	8B45 08	MOV EAX,DWORD PTR SS:[EBP+8]
004012B1	50	PUSH EAX
004012B2	FF15 A8504000	CALL DWORD PTR DS:[4050A8]
004012B8	68 00040000	PUSH 4000
004012BD	6A 03	PUSH 3
004012BF	8B4D 08	MOV ECX,DWORD PTR SS:[EBP+8]
004012C2	51	PUSH ECX
004012C3	FF15 A4504000	CALL DWORD PTR DS:[4050A4]
004012C9	50	PUSH EAX
004012CA	FF15 A0504000	CALL DWORD PTR DS:[4050A0]
004012D0	EB 5C	JMP SHORT 004012E2
004012D2	68 01040000	PUSH 4010
004012D7	6A 04	PUSH 4
004012D9	6A 00	PUSH 0
004012DB	8B55 08	MOV EDI,DWORD PTR SS:[EBP+8]
004012DE	52	PUSH EDI

و همچنین یک پرش شرطی در خط 401265 قرار دارد که باید JMP شود. خوب حالا با چه کدی می توانم این دستورات را دستکاری کنم؟ معادل NOP، Opcode برابر 90 است. یعنی برای کردن پرشی که در 4012A7 قرار دارد، باید دو بایت 4012A7 و 4012A8 را برابر 90 کنیم. برای این کار می توانم از کد زیر استفاده کنم:

```
MOV Byte PTR DS:[4012A7], 90
MOV Byte PTR DS:[4012A8], 90
```

البته می توانم این دو دستور را به یک دستور تبدیل کنم و یکدفعه این کار را انجام دهم:

```
MOV WORD PTR DS:[4012A7], 9090
```

حالا برای JMP کردن پرشی که در خط 401265 قرار دارد چه کار کنیم؟... معادل JMP، Opcode برابر EB است. بنابراین باید این بایت را برابر EB کنیم:

```
MOV Byte PTR DS:[401265], 0EB
```

پس برای کرک شدن کامل برنامه فقط همین دو دستور کافیه:

```
MOV WORD PTR DS:[4012A7], 9090
MOV BYTE PTR DS:[401265], 0EB
```

فایل را در OllyDBG ری استارت کنید و این کدها را همانند تصویر به سکشنی که اضافه کردیم، اضافه کنید:

0041B011	0000	ADD BYTE PTR DS:[EAX],AL
0041B013	0000	ADD BYTE PTR DS:[EAX],AL
0041B015	0000	ADD BYTE PTR DS:[EAX],AL
0041B017	0000	ADD BYTE PTR DS:[EAX],AL
0041B019	0000	ADD BYTE PTR DS:[EAX],AL
0041B01B	66 C705 A7124000 9090	MOV WORD PTR DS:[4012A7],9090
0041B024	C605 65124000 EB	MOV BYTE PTR DS:[401265],0EB
0041B02B	0000	ADD BYTE PTR DS:[EAX],AL
0041B02D	0000	ADD BYTE PTR DS:[EAX],AL
0041B02F	0000	ADD BYTE PTR DS:[EAX],AL
0041B031	0000	ADD BYTE PTR DS:[EAX],AL
0041B033	0000	ADD BYTE PTR DS:[EAX],AL
0041B035	0000	ADD BYTE PTR DS:[EAX],AL
0041B037	0000	ADD BYTE PTR DS:[EAX],AL
0041B039	0000	ADD BYTE PTR DS:[EAX],AL
0041B03B	0000	ADD BYTE PTR DS:[EAX],AL
0041B03D	0000	ADD BYTE PTR DS:[EAX],AL
0041B03F	0000	ADD BYTE PTR DS:[EAX],AL
0041B041	0000	ADD BYTE PTR DS:[EAX],AL
0041B043	0000	ADD BYTE PTR DS:[EAX],AL

من در خط 41B01B این کدها را اضافه کنیم. شما می توانید در هر جایی از این سکشن که دلتان خواست، این دو خط کد را اضافه کنید.

حالا چه طور این کدها را اجرا کنیم؟... این که این کدها را اضافه کردیم که برنامه کرک نمی شود. حتما باید این کدها اجرا شود... اما چه طوری؟... ما می توانیم ببینیم برنامه در چه خطی به OEP می رود و بایم آن خط را دستکاری کنیم. یعنی به جای برنامه در آن خط به OEP برود. کاری می کنم که اول کدهای ما را اجرا کند و بعد به OEP برود. یکبار کلید F8 را بزنید:

00417F0B	6A 00	PUSH 0
00417F0D	39C4	CMP ESP,EAX
00417F0F	^ 75 FA	JNZ SHORT 00417F0B
00417F11	83EC 80	SUB ESP,-80
00417F14	- E9 0F92FEFF	JMP 00401128
00417F19	68 3D7F4100	PUSH 00417F30
00417F1E	64:FF35 00000000	PUSH DWORD PTR FS:[0]
00417F25	64:8925 00000000	MOV DWORD PTR FS:[0],ESP
00417F2C	CD 03	INT 3
00417F2E	8B0424	MOV EAX,DWORD PTR SS:[ESP]
00417F31	64:A3 00000000	MOV DWORD PTR FS:[0],EAX
00417F37	83C4 08	ADD ESP,8
00417F3A	C3	RET
00417F3B	0000	ADD BYTE PTR DS:[EAX],AL
00417F3D	C7C6 BD8C1F66	MOV ESI,661F8CBD
00417F43	0FBAE9 1C	BTS ECX,1C
00417F47	89EE	MOV ESI,EBP
00417F49	B4 DB	MOV AH,0DB
00417F4B	86D5	XCHG CH,DL
00417F4D	0FC9	BSWAP ECX

ما الان در Entry point قبلی فایل هستیم. (آن موقعی که سکشن اضافه نکرده بودیم). پرشی که بالای این خط قرار دارد (JMP 00401128) ما را به OEP می برد. (این موضوع را در موقع آپک کردن فایل در قسمت قبل فهمیدیم). خوب من می آیم این پرش را دستکاری می کنم. یعنی به جای اینکه این پرش به OEP برود، اول کدهای ما را اجرا کند و بعد به OEP برود. 😊 خوب، این پرش را دستکاری کنید و بنویسید : JMP 41B01B : به این ترتیب کدهای ما اجرا می شود:

00417F0B	6A 00	PUSH 0
00417F0D	39C4	CMP ESP,EAX
00417F0F	^ 75 FA	JNZ SHORT 00417F0B
00417F11	83EC 80	SUB ESP,-80
00417F14	- E9 02310000	JMP 0041B01B
00417F19	68 3D7F4100	PUSH 00417F30
00417F1E	64:FF35 00000000	PUSH DWORD PTR FS:[0]
00417F25	64:8925 00000000	MOV DWORD PTR FS:[0],ESP
00417F2C	CD 03	INT 3
00417F2E	8B0424	MOV EAX,DWORD PTR SS:[ESP]
00417F31	64:A3 00000000	MOV DWORD PTR FS:[0],EAX
00417F37	83C4 08	ADD ESP,8
00417F3A	C3	RET
00417F3B	0000	ADD BYTE PTR DS:[EAX],AL
00417F3D	C7C6 BD8C1F66	MOV ESI,661F8CBD
00417F43	0FBAE9 1C	BTS ECX,1C
00417F47	89EE	MOV ESI,EBP
00417F49	B4 DB	MOV AH,0DB
00417F4B	86D5	XCHG CH,DL
00417F4D	0FC9	BSWAP ECX
00417F4F	47	INC EDI
00417F50	F3:	PREFIX REP:
00417F51	0FA5D3	SHLD EBX,EDX,CL
00417F54	0FABC1	BTS ECX,EAX
00417F57	0FBCFE	BSF EDI,ESI
00417F5A	0FBAE5 0B	BT EBP,0B
00417F5E	86D5	XCHG CH,DL

حالا بعد از کدهایی که تزریق کردیم، بنویسید JMP 401128، به این ترتیب برنامه بعد از اجرا کردن کدهای ما به OEP می رود:

0041B013	0000	ADD BYTE PTR DS:[EAX],AL
0041B015	0000	ADD BYTE PTR DS:[EAX],AL
0041B017	0000	ADD BYTE PTR DS:[EAX],AL
0041B019	0000	ADD BYTE PTR DS:[EAX],AL
0041B01B	66 C705 A7124000 9090	MOV WORD PTR DS:[4012A7],9090
0041B024	C605 65124000 EB	MOV BYTE PTR DS:[401265],0EB
0041B02B	- E9 F860FEFF	JMP 00401128
0041B030	0000	ADD BYTE PTR DS:[EAX],AL
0041B032	0000	ADD BYTE PTR DS:[EAX],AL
0041B034	0000	ADD BYTE PTR DS:[EAX],AL
0041B036	0000	ADD BYTE PTR DS:[EAX],AL
0041B038	0000	ADD BYTE PTR DS:[EAX],AL
0041B03A	0000	ADD BYTE PTR DS:[EAX],AL
0041B03C	0000	ADD BYTE PTR DS:[EAX],AL
0041B03E	0000	ADD BYTE PTR DS:[EAX],AL
0041B040	0000	ADD BYTE PTR DS:[EAX],AL
0041B042	0000	ADD BYTE PTR DS:[EAX],AL
0041B044	0000	ADD BYTE PTR DS:[EAX],AL
0041B046	0000	ADD BYTE PTR DS:[EAX],AL
0041B048	0000	ADD BYTE PTR DS:[EAX],AL

حالا برنامه را اجرا کنید، می بینید که برنامه به طور کامل کرک شده است 😊

Loader

گاهی اوقات راحت ترین راه برای کرک کردن یک برنامه پک شده ساخت Loader است. اصول کار Loader به این ترتیب است که منتظر می ماند که خطی از فایل اصلی ما به بایت مورد نظرش برسد (یعنی جایی که پکر کدها را کامل Decrypt کند.) و بعد Loader آن فایل را در Memory دستکاری می کند. همانطور که در قسمت قبل گفتم ما برای کرک کردن برنامه فقط باید این سه بایت را دستکاری کنیم:

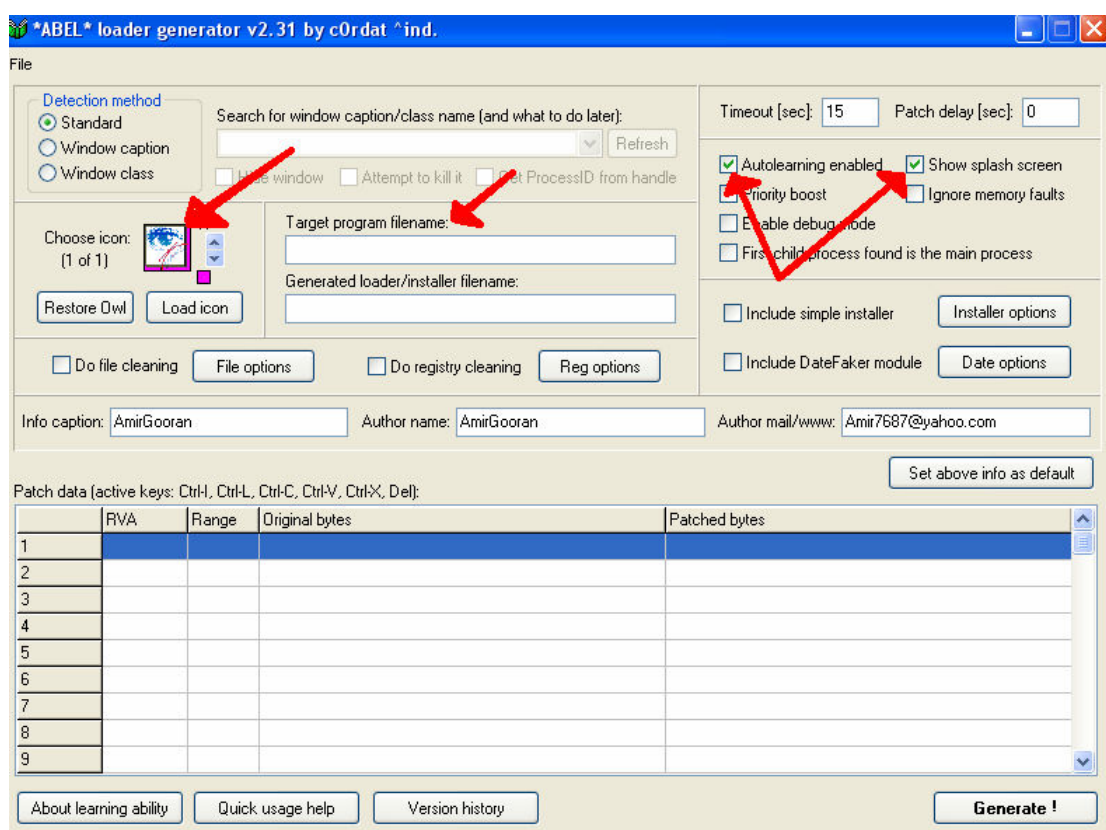
4012A7 → 90

4012A8 → 90

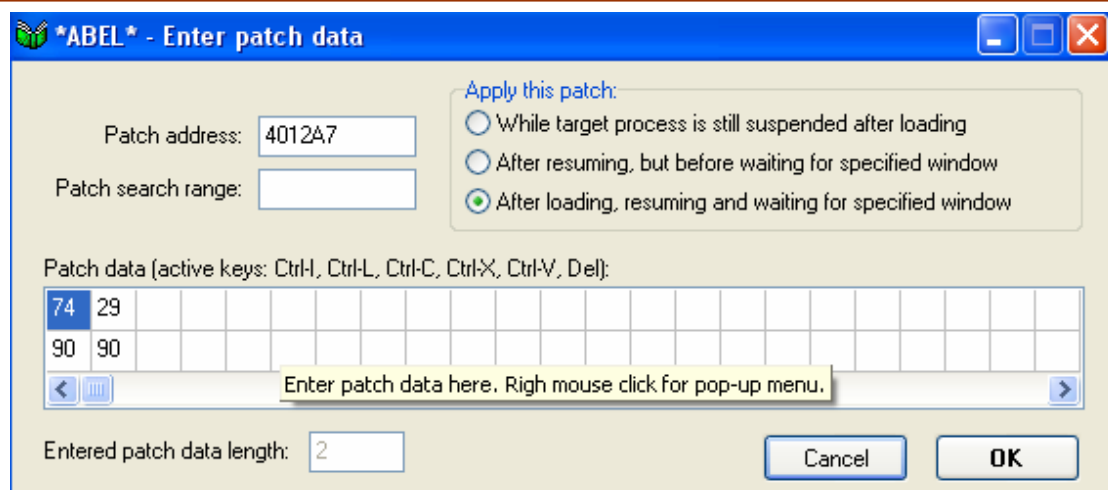
401265 → EB

ABEL Loader

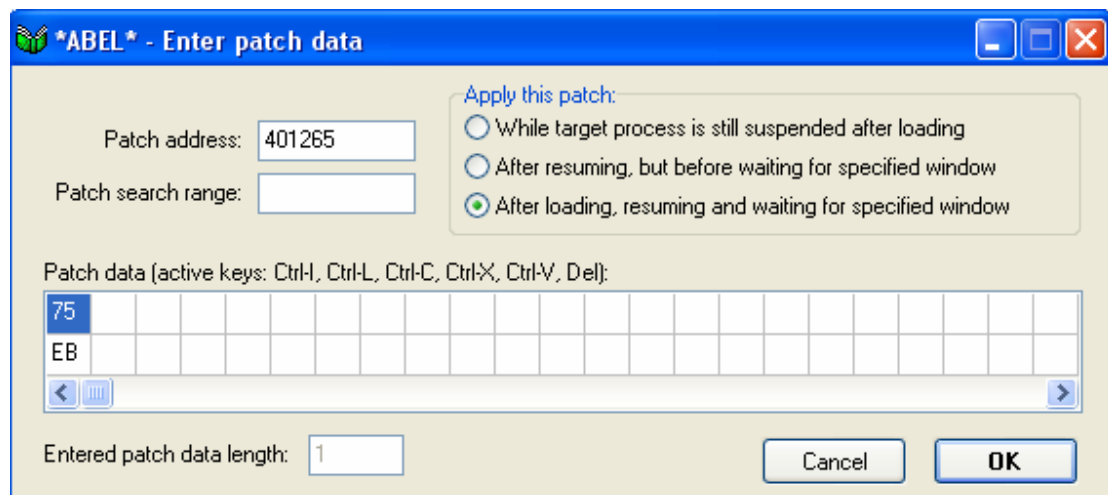
برای ساخت Loader نرم افزارهای زیادی وجود دارد که ما دو تا از معروفترین را آنها استفاده می کنیم. ابتدا برای ساخت Loader از ABEL Loader استفاده می کنیم، برنامه را باز کنید:



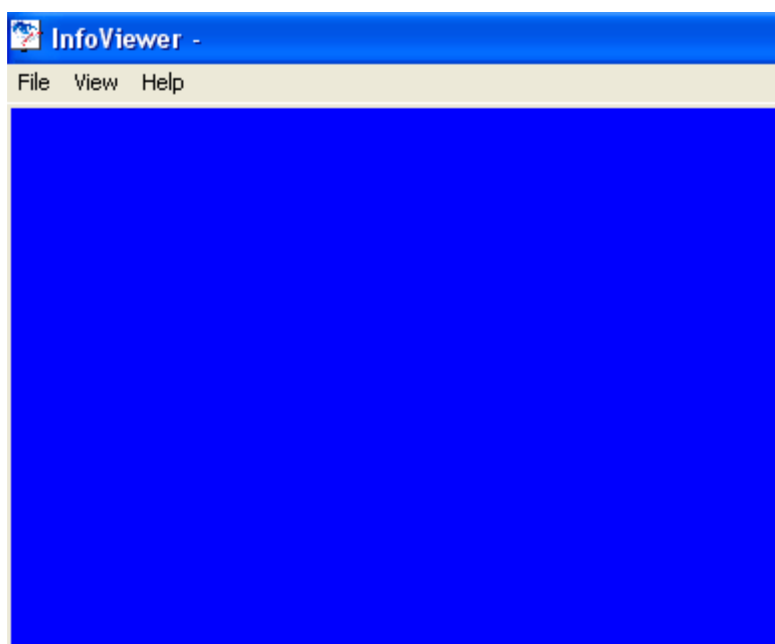
اول از همه تیک دو گزینه Autolearning Enabled و Show splash screen را بردارید. اگر تیک این دو گزینه را بردارید در هنگام اجرای Loader یک Splash screen نمایش داده می شود.
گزینه Load Icon را بزنید و آیکنی برای Loader خود انتخاب کنید.
در قسمت Target program filename هم نام فایل هدف را بنویسید (InfoViewer.exe)
حالا بر روی جدول (در پایین برنامه) کلیک کنید تا صفحه زیر ظاهر شود:



شما می توانید بایت های پشت سر هم را یکجا وارد کنید. برای مثال در تصویر بالا من بایت های 4012A7 و 4012A8 را یکسره وارد کردم. همچنین باید بایت اصلی را هم به برنامه بدهید. برای مثال در تصویر بالا من نوشتم که در خط 4012A7 بایت 74 قرار داشته باشد که Loader باید آن را به 90 تغییر دهد. کلید OK را بزنید و بایت بعدی را هم وارد کنید:

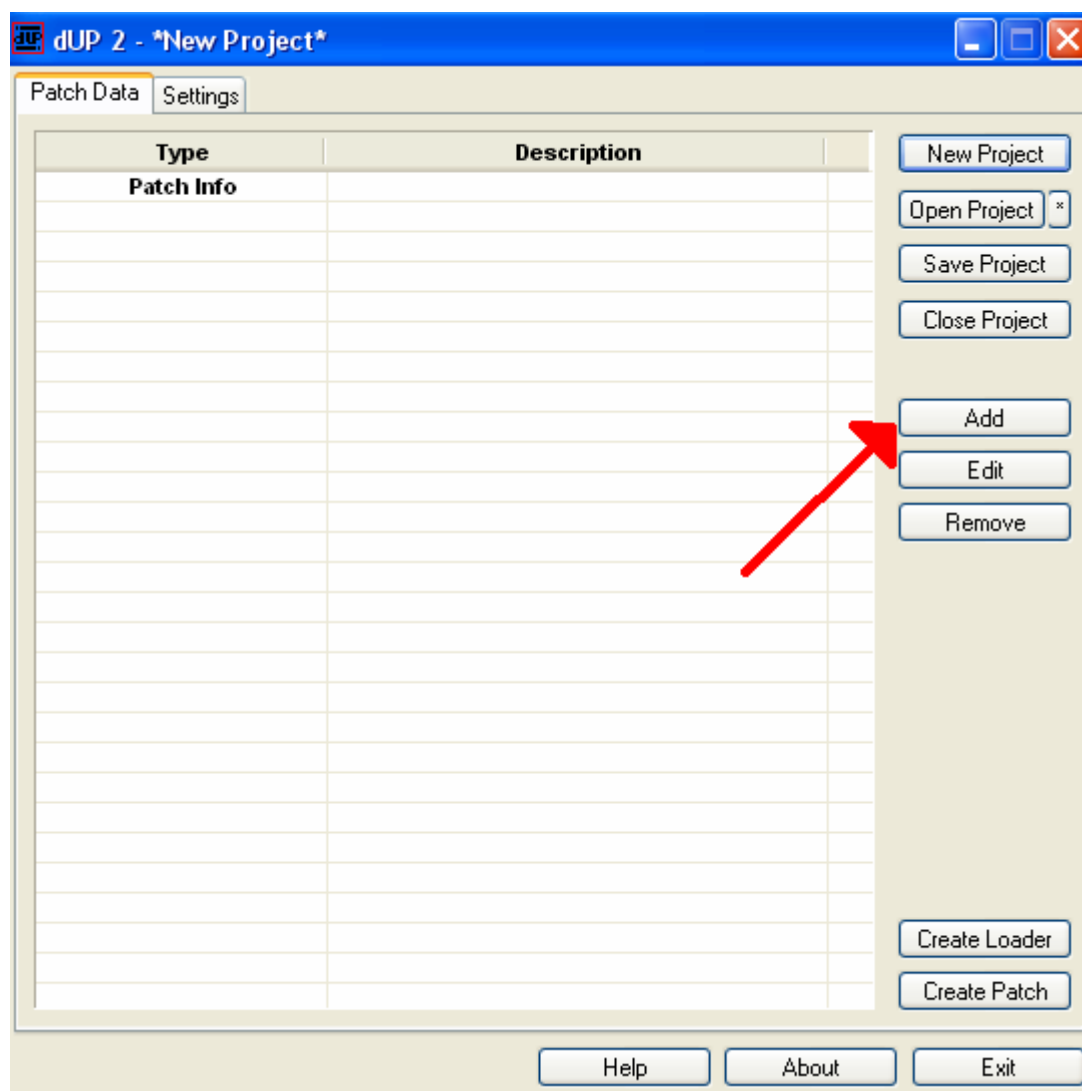


دوباره کلید OK را بزنید و گزینه Generate را بزنید و Loader را در همان پوشه ای که نرم افزار هدف قرار دارد، ذخیره کنید. حالا Loader را اجرا کنید. می بینید که Loader به خوبی توانسته بایت ها را دستکاری کند و برنامه رجیستر شده است.

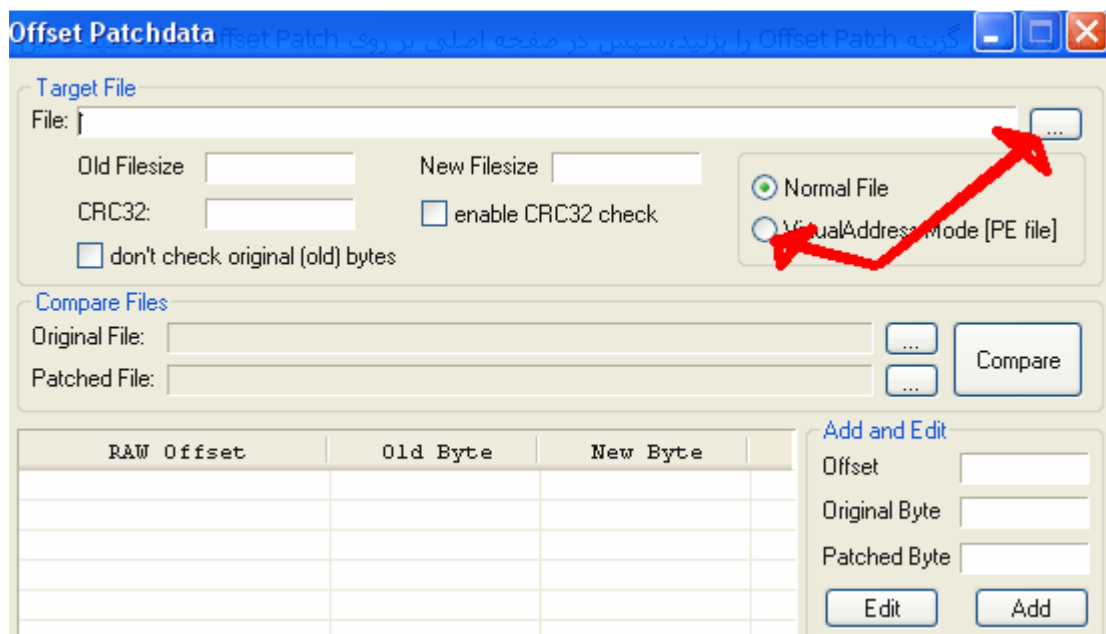


Diablo2oo2

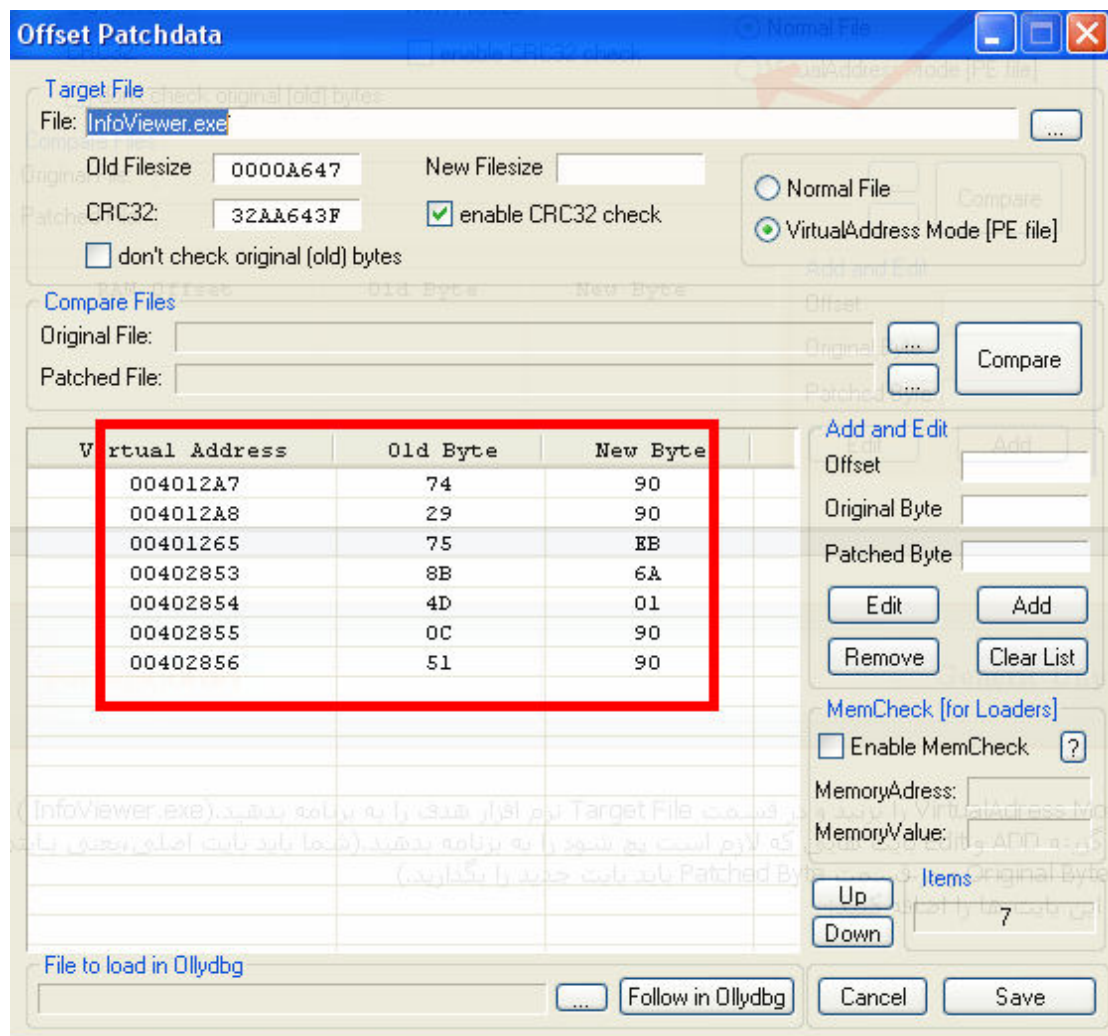
بیا یه بار دیگه، اما این بار با نرم افزار Diablo2oo2 یک Loader بسازیم. برنامه را اجرا کنید و گزینه New Project و سپس گزینه Save را بزنید.



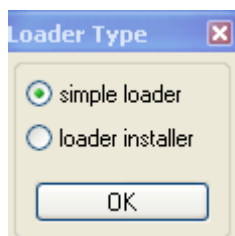
گزینه ADD و سپس گزینه Offset Patch را بزنید، سپس در صفحه اصلی بر روی Patch offset کلیک کنید تا این صفحه ظاهر شود:



تیک گزینه VirtualAddress Mode را بزنید و در قسمت Target File نرم افزار هدف را به برنامه بدهید. (InfoViewer.exe)
 حالا از طریق گزینه ADD و Edit بایت هایی که لازم است پچ شود را به برنامه بدهید. (شما باید بایت اصلی، یعنی بایتی که قبلا بوده را در قسمت Original Byte و در قسمت Patched Byte باید بایت جدید را بگذارید).
 همانند تصویر این بایت ها را اضافه کنید:



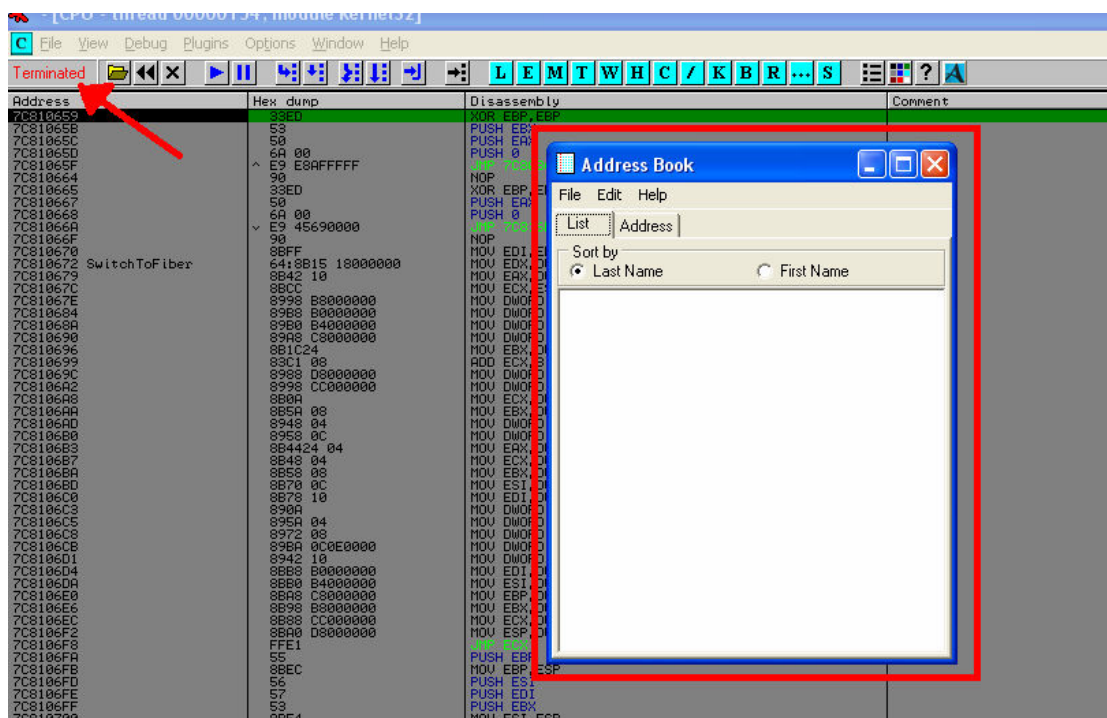
ممکن است سوال کنید چرا هفت بایت را دستکاری کردم؟... به خاطر اینکه یکبار Loader را با تغییر همان سه بایت ساختم. اما دیدم برنامه خطای Can't open the file را می دهد. به همین دلیل مجبور شدم چند بایت دیگر را هم پچ کنم.
 خوب، گزینه Save را بزنید. و سپس گزینه Create Loader را بزنید:



اگر می خواهید فقط یک Loader ساده بسازید، گزینه Simple loader را بزنید. ولی اگر می خواهید یک Loader بسازید که نام فایل اصلی را تغییر دهد و خودش را بگذارد جای فایلی اصلی، گزینه Loader installer را بزنید.
 حالا کلید OK را بزنید و Loader خود را ذخیره کنید. و آن را اجرا کنید. می بینید که برنامه Crack شده است و دیگر خبری از Nag Screen نیست.

! EP Exepack

فایل مورد نظر را در OllyDBG باز کنید و کلید F9 را بزنید:



همانطور که می بینید برنامه به طور کامل اجرا شده...اما OllyDBG می گوید که برنامه Terminate شده است...خوب این یعنی چی؟...یعنی اینکه برنامه بعد از آنیک کردن فایل، یک پروسه جدید می سازد و در آن پروسه جدید، فایل را اجرا می کند...خوب، حالا چه کار کنیم؟...ما باید بینیم که برنامه در کجا پروسه جدید را می سازد؟...بر این کار هم روی تابع CreateProcessA، Breakpoint، می گذاریم.و بعد برنامه را با کلید F9 اجرا می کنیم، تا در اینجا متوقف شویم:

7C802365	90	NOP
7C802366	90	NOP
7C802367 CreateProcessA	8BFF	MOV EDI,EDI
7C802369	55	PUSH EBP
7C80236A	8BEC	MOV EBP,ESP
7C80236C	6A 00	PUSH 0
7C80236E	FF75 2C	PUSH DWORD PTR SS:[EBP+2C]
7C802371	FF75 28	PUSH DWORD PTR SS:[EBP+28]
7C802374	FF75 24	PUSH DWORD PTR SS:[EBP+24]
7C802377	FF75 20	PUSH DWORD PTR SS:[EBP+20]
7C80237A	FF75 1C	PUSH DWORD PTR SS:[EBP+1C]
7C80237D	FF75 18	PUSH DWORD PTR SS:[EBP+18]
7C802380	FF75 14	PUSH DWORD PTR SS:[EBP+14]
7C802383	FF75 10	PUSH DWORD PTR SS:[EBP+10]
7C802386	FF75 0C	PUSH DWORD PTR SS:[EBP+0C]
7C802389	FF75 08	PUSH DWORD PTR SS:[EBP+08]
7C80238C	6A 00	PUSH 0
7C80238E	E8 43BA0100	CALL CreateProcessInternalA
7C802393	5D	POP EBP
7C802394	C2 2800	RET 28
7C802397	90	NOP
7C802398	90	NOP
7C802399	90	NOP
7C80239A	90	NOP

کلید ALT + F9 را بزنید.تا وارد کدهای فایل اصلی خودمان بشویم.

004B3672	50	PUSH EAX
004B3673	6A 00	PUSH 0
004B3675	8D9D 75020000	LEA EBX,DWORD PTR SS:[EBP+275]
004B367B	53	PUSH EBX
004B367C	57	PUSH EDI
004B367D	FFD6	CALL ESI
004B367F	85C0	TEST EAX,EAX
004B3681	0F84 FB010000	JE 004B3882
004B3687	FFD0	CALL EAX
004B3689	8B9D EE020000	MOV EBX,DWORD PTR SS:[EBP+2EE]
004B368F	53	PUSH EBX
004B3690	8D9D 94020000	LEA EBX,DWORD PTR SS:[EBP+294]
004B3696	53	PUSH EBX
004B3697	57	PUSH EDI
004B3698	FFD6	CALL ESI
004B369A	85C0	TEST EAX,EAX
004B369C	0F84 E0010000	JE 004B3882
004B36A2	FFD0	CALL EAX
004B36A4	8B9D F2020000	MOV EBX,DWORD PTR SS:[EBP+2F2]
004B36AA	53	PUSH EBX
004B36AB	8D9D 94020000	LEA EBX,DWORD PTR SS:[EBP+294]
004B36B1	53	PUSH EBX
004B36B2	57	PUSH EDI
004B36B3	FFD6	CALL ESI
004B36B5	85C0	TEST EAX,EAX
004B36B7	0F84 C5010000	JE 004B3882
004B36BD	FFD0	CALL EAX
004B36BF	6A 00	PUSH 0
004B36C1	8D9D A0020000	LEA EBX,DWORD PTR SS:[EBP+2A0]
004B36C3	53	PUSH EBX

همانطور که می بینید پروسه ما اجرا شده، خط بالاتر از دستور EAX Call یک پرش شرطی وجود دارد که تصمیم گیرنده است. اگر این پرش انجام شود، برنامه در پروسه فعلی اجرا می شود و اگر پرش انجام نشود، یک پروسه جدید ساخته می شود... لذا بر روی این پرش Hardware breakpoint on execution بگذارید و برنامه را Restart کنید و کلید F9 را بزنید. همانند تصویر این پرش را به JMP تغییر دهید تا همیشه انجام شود.

004B367C	57	PUSH EDI
004B367D	FFD6	CALL ESI
004B367F	85C0	TEST EAX,EAX
004B3681	0F84 FC010000	JMP 004B3882
004B3686	90	NOOP
004B3687	FFD0	CALL EAX
004B3689	8B9D EE020000	MOV EBX,DWORD PTR SS:[EBP+2EE]
004B368F	53	PUSH EBX
004B3690	8D9D 94020000	LEA EBX,DWORD PTR SS:[EBP+294]
004B3696	53	PUSH EBX

حالا کلید F9 را بزنید می بینید که برنامه این بار در درون پروسه فعلی اجرا شده است. خوب حالا چه طور OEP را پیدا کنیم؟ برنامه را ری استارت کنید. و کلید F9 را بزنید:

004B3675	8D9D 75020000	LEA EBX,DWORD PTR SS:[EBP+275]
004B367B	53	PUSH EBX
004B367C	57	PUSH EDI
004B367D	FFD6	CALL ESI
004B367F	85C0	TEST EAX,EAX
004B3681	0F84 FB010000	JE 004B3882
004B3687	FFD0	CALL EAX
004B3689	8B9D EE020000	MOV EBX,DWORD PTR SS:[EBP+2EE]
004B368F	53	PUSH EBX
004B3690	8D9D 94020000	LEA EBX,DWORD PTR SS:[EBP+294]
004B3696	53	PUSH EBX
004B3697	57	PUSH EDI
004B3698	FFD6	CALL ESI
004B369A	85C0	TEST EAX,EAX
004B369C	0F84 E0010000	JE 004B3882
004B36A2	FFD0	CALL EAX
004B36A4	8B9D F2020000	MOV EBX,DWORD PTR SS:[EBP+2F2]
004B36AA	53	PUSH EBX
004B36AB	8D9D 94020000	LEA EBX,DWORD PTR SS:[EBP+294]
004B36B1	53	PUSH EBX

چون قبلا اینجا Hardware breakpoint گذاشته بودیم. دوباره همینجا متوقف شدیم. همانند قبل این پرش را JMP کنید و کلیدهای ALT + M را بزنید تا وارد Memory map شوید. و بر روی سکشن Code برنامه یک memory breakpoint on access بگذارید. و بعد هم کلید F9 را بزنید تا در اینجا متوقف شوید:

004B212A	BE 00104000	MOV ESI,00401000
004B212F	60	PUSHAD
004B2130	FC	CLD
004B2131	B2 80	MOV DL,80
004B2133	31DB	XOR EBX,EBX
004B2135	A4	MOVS BYTE PTR ES:[EDI],BYTE PTR DS:[ESI]
004B2136	B3 02	MOV BL,2
004B2138	E8 6D000000	CALL 004B21AA
004B213D	73 F6	JNB SHORT 004B2135
004B213F	31C9	XOR ECX,ECX
004B2141	E8 64000000	CALL 004B21AA
004B2146	73 1C	JNB SHORT 004B2164
004B2148	31C0	XOR EAX,EAX
004B214A	E8 5B000000	CALL 004B21AA
004B214F	73 23	JNB SHORT 004B2174
004B2151	B3 02	MOV BL,2
004B2153	41	INC ECX
004B2154	B0 10	MOV AL,10
004B2156	E8 4F000000	CALL 004B21AA
004B215B	10C0	ADC AL,AL
004B215D	73 F7	JNB SHORT 004B2156
004B215F	75 3F	JNZ SHORT 004B21A0

[illegible]

The screenshot shows the Immunity Debugger interface. The main window displays assembly code with registers R0 through R7, all containing 'empty' values. The 'Breakpoint' menu is open, showing options like 'Backup', 'Copy', 'Binary', 'Assemble', 'Label', 'Comment', 'Breakpoint', 'Run trace', 'New origin here', 'Go to', 'Follow in Dump', 'Search for', 'Find references to', 'View', 'Copy to executable', 'Analysis', 'Detach Process', and 'Process Patcher'. The 'Breakpoint' sub-menu is also open, showing options like 'Toggle', 'Conditional', 'Conditional log', 'Run to selection', 'Memory, on access', 'Memory, on write', 'Remove memory breakpoint', and 'Hardware, on execution'. The 'Remove memory breakpoint' option is highlighted in blue.

004B2281	FF10	CALL 00000 PTR DS:[EAX]	ASCII "GlobalFree"
004B2282	68 BF204B00	PUSH 004B206F	
004B2283	59	PUSH EAX	
004B2289	B8 44204B00	MOV EAX,<KERNEL32.GetProcAddress>	
004B228E	FF10	CALL 00000 PTR DS:[EAX]	
004B2290	8B15 CA204B00	MOV EDI,00000 PTR DS:[4B20CA]	
004B2296	52	PUSH EDI	
004B2297	FFD0	CALL EAX	
004B2299	61	POPAD	
004B229A	BA EC584900	MOV EDI,004958EC	
004B229F	FFE2	JMP EBX	
004B22A1	90	NOP	
004B22A2	C3	RET	
004B22A3	44	INC ESP	
004B22A4	0000	ADD BYTE PTR DS:[EAX],AL	
004B22A6	0000	ADD BYTE PTR DS:[EAX],AL	
004B22A8	0000	ADD BYTE PTR DS:[EAX],AL	
004B22AA	0000	ADD BYTE PTR DS:[EAX],AL	
004B22AC	0000	ADD BYTE PTR DS:[EAX],AL	
004B22AE	0000	ADD BYTE PTR DS:[EAX],AL	
004B22B0	0000	ADD BYTE PTR DS:[EAX],AL	
004B22B2	0000	ADD BYTE PTR DS:[EAX],AL	

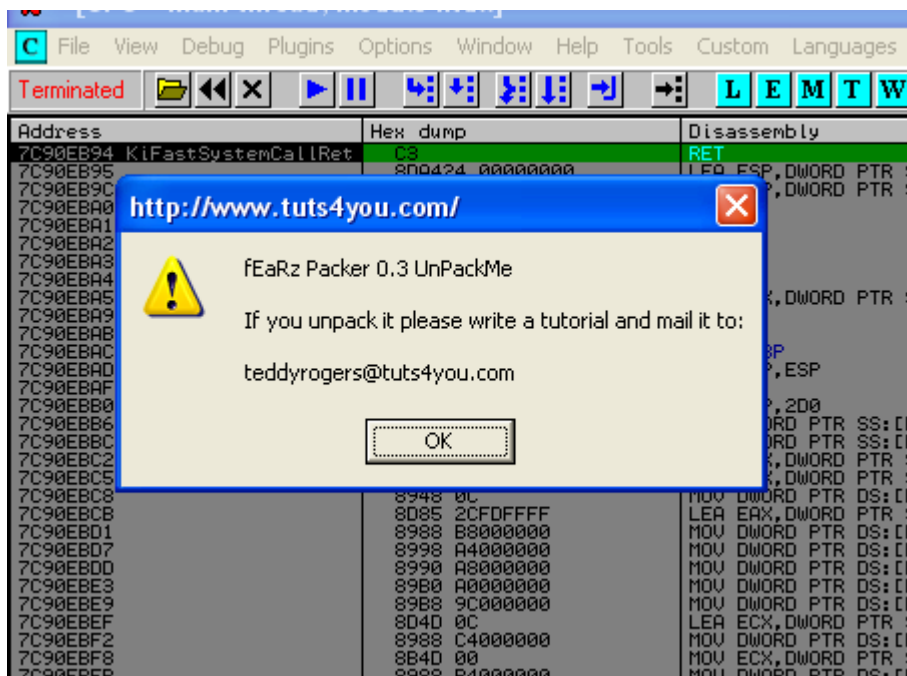
این CALL EAX که به تابع GlobalFree می رود یک بار دیگر F9 بزنید:

004B2290	8B15 CA204B00	MOV EDX,DWORD PTR DS:[4B20CA]
004B2296	52	PUSH EDX
004B2297	FFD0	CALL EAX
004B2299	61	POPAD
004B229A	BA EC584900	MOV EDX,004958EC
004B229F	- FFE2	JMP EDX
004B22A1	90	NOP
004B22A2	C3	RET
004B22A3	44	INC ESP
004B22A4	0000	ADD BYTE PTR DS:[EAX],AL
004B22A6	0000	ADD BYTE PTR DS:[EAX],AL
004B22A8	0000	ADD BYTE PTR DS:[EAX],AL
004B22AA	0000	ADD BYTE PTR DS:[EAX],AL
004B22AC	0000	ADD BYTE PTR DS:[EAX],AL

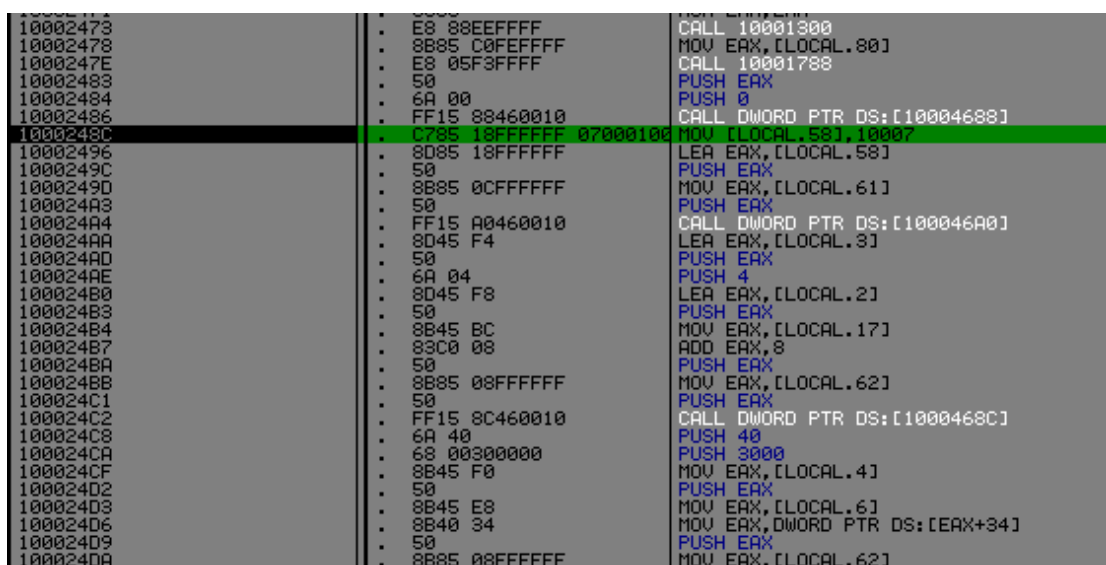
آها ☺ دستور JMP EDX ما را به OEP می برد.
بعد از یافتن OEP مشکلی به وجود نمی آید، دامپ می گیرید و فیکس می کنید.

Fearz Packer

فایل مورد نظر را در OllyDBG باز کنید، و کلید F9 را بزنید تا برنامه اجرا شود:



همانطور که می بینید برنامه در داخل OllyDBG بسته شده، اما پروسه همچنان اجراست. این نشان می دهد که یک پروسه جدید ساخته شده و فایل ما در آن پروسه اجرا شده است. خوب برنامه را Restart کرده، بر روی تابع CreateProcessA بریک پوینت بگذارید. (برای این کار در پلاگین Command Bar بنویسید: BP CreateProcessA) و بعد برنامه را با F9 اجرا کنید. برنامه بر روی تابع CreateProcessA متوقف می شود. حالا کلید ALT + F9 را بزنید تا وارد کدهای فایل خودمان بشویم:



برنامه را با کلید F8 ادامه دهید تا به این خط برسیم:

10002514	50	PUSH EAX
1000251B	8B85 08FFFFFF	MOV EAX,[LOCAL.62]
10002521	50	PUSH EAX
10002522	FF15 90460010	CALL DWORD PTR DS:[10004690]
10002528	8B45 E8	MOV EAX,[LOCAL.6]
1000252B	8B40 34	MOV EAX,DWORD PTR DS:[EAX+34]
1000252E	8B55 E8	MOV EDX,[LOCAL.6]
10002531	0342 28	ADD EAX,DWORD PTR DS:[EDX+28]
10002534	8945 C8	MOV [LOCAL.14],EAX
10002537	8D85 18FFFFFF	LEA EAX,[LOCAL.58]
1000253D	50	PUSH EAX
1000253E	8B85 0CFFFFFF	MOV EAX,[LOCAL.61]
10002544	50	PUSH EAX
10002545	FF15 98460010	CALL DWORD PTR DS:[10004698]
1000254B	8B85 0CFFFFFF	MOV EAX,[LOCAL.61]
10002551	50	PUSH EAX
10002552	FF15 9C460010	CALL DWORD PTR DS:[1000469C]
10002558	33C0	XOR EAX,EAX
1000255A	5A	POP EDX
1000255B	59	POP ECX
1000255C	59	POP ECX
1000255D	64:8910	MOV DWORD PTR FS:[EAX],EDX
10002560	68 77250010	PUSH 10002577

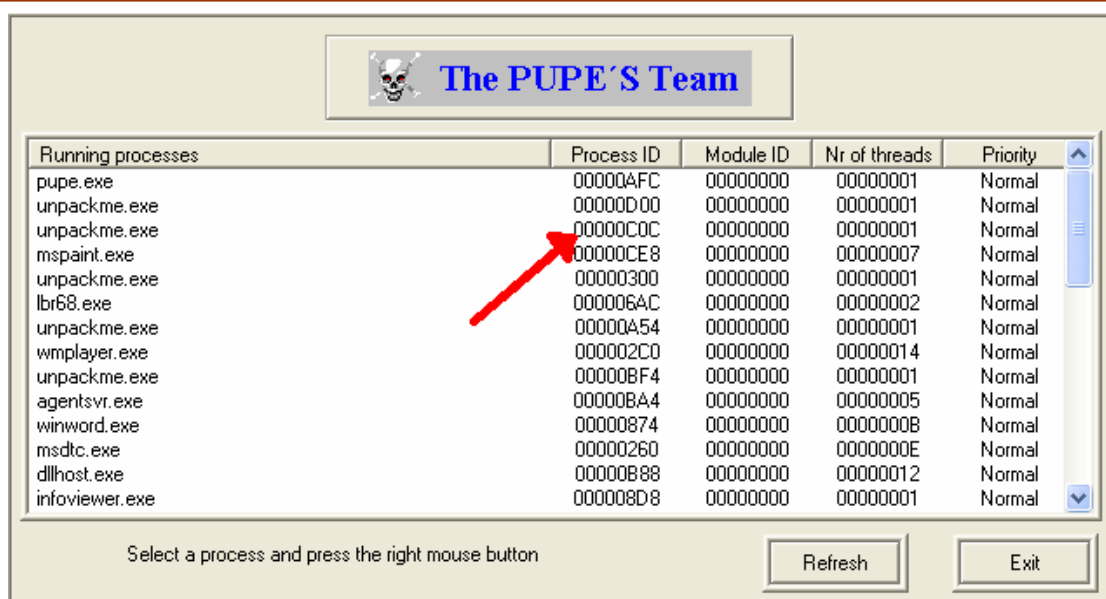
الان مقدار رجیستر EAX همان OEP ماست.(در واقع Entry Point پروسه جدید)
برنامه را ادامه دهید تا به این خط برسید:

8B85 0CFFFFFF	MOV EAX,[LOCAL.61]	
50	PUSH EAX	
FF15 9C460010	CALL DWORD PTR DS:[1000469C]	kernel32.ResumeThread
33C0	XOR EAX,EAX	
5A	POP EDX	
59	POP ECX	
59	POP ECX	
64:8910	MOV DWORD PTR FS:[EAX],EDX	
68 77250010	PUSH 10002577	
8B45 E4	MOV EAX,[LOCAL.7]	
50	PUSH EAX	

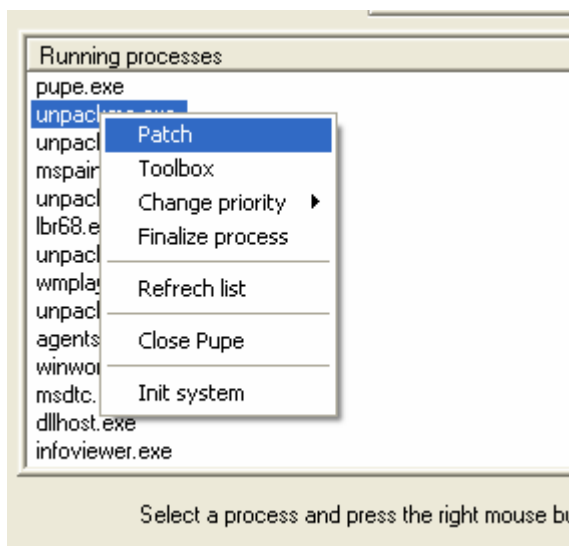
با اجرا شدن تابع Resume Thread برنامه اجرا می شود.خوب،پس اگر یک بار دیگر F8 بزنید،برنامه اجرا می شود...اما ما باید روی OEP پروسه جدیدمان متوقف شویم تا بتوانیم از فایلمان دامپ بگیریم و بعد هم دامپ را فیکس کنیم...اما اگر این دستور (Resume Thread) اجرا شود،برنامه به طور کامل اجرا می شود...چه کار باید بکنیم؟...ما باید به نوعی پروسه ساخته شده جدید را متوقف کنیم.یعنی کاری کنیم برنامه در خط 4271B0 (Entry point پروسه جدید) متوقف شود.
در منوی File گزینه Attach را بزنید :

000005F8	nvsuc32	NUSUCPMMWindowClass	C:\WINDOWS\system32\nvsuc32.dll
00000618	PC2AMP		C:\Program Files\PC 2 AMP\PC2AMP.exe
00000654	SMAgent		C:\Program Files\Analog\SMAgent.exe
0000079C	wscntfy		C:\WINDOWS\system32\wscntfy.exe
000007C8	alg		C:\WINDOWS\System32\alg.exe
00000874	WINWORD	Lines	C:\Program Files\Microsoft Office\WINWORD.EXE
000008B4	dlhhost		C:\WINDOWS\system32\dlhhost.dll
000008B8	AgentSvr	Microsoft Agent	C:\WINDOWS\msagent\AgentSvr.exe
00000C0C	UnPackMe		E:\Learn\Generic Unpacking\UnPackMe.exe
00000CE8	mspaint	untitled - Paint	C:\WINDOWS\system32\mspaint.exe
00000EF4	Babylon	Babylon	C:\Program Files\Babylon\Babylon.exe

همانطور که می بینید پروسه فایل من برابر COC است.(البته در کامپیوتر من)
برای متوقف کردن پروسه جدید از نرم افزار PUPE استفاده می کنیم.این برنامه را باز کنید:



همانطور که می بینید دو پروسه Unpackme داریم یکی که پروسه ای است که در OllyDBG قرار دارد. (که Process ID آن برابر C0C است) و دیگری که PID آن برابر D00 است. من باید پروسه جدید را متوقف کنم. لذا بر روی پروسه ای که با تابع CreateProcessA ایجاد شده (همانی که PID آن در کامپیوتر من D00 است) کلیک راست کرده و گزینه Patch را بزنید:



خوب... حالا ما باید کاری کنیم که پروسه جدید ما از Entry point خودش تکان نخورد و همانجا بماند؟... چه طوری؟... من می توانم از دستور infinite loop استفاده کنم. این دستور یک خط را به تعداد نامحدود اجرا می کند و در واقع برنامه در خطی که این دستور را دارد متوقف می شود. معادل opcode این دستور EBFE است. لذا من باید دو بایت ابتدای فایل را EBFE کنم تا فایل از آنجا تکان نخورد. ☺

در قسمت N? Bytes بنویسید: ۲ (چون برای پچ کردن به دو بایت نیاز داریم).
در قسمت Direction مقدار OEP را وارد کنید. (004271B0)
و بعد گزینه Search را بزنید.

Patching

Process:

N? bytes:

Bytes

Direction:

To change by:

همانطور که می بینید در قسمت Bytes نوشته شده 558B یعنی هم اکنون در بایت 4271B0 مقدار 558B قرار دارد. در TextBox زیر Bytes بنویسید: EBFE و سپس گزینه Patching را بزنید:

Patching

Process:

N? bytes:

Bytes

Direction:

To change by:

Patching

Process:

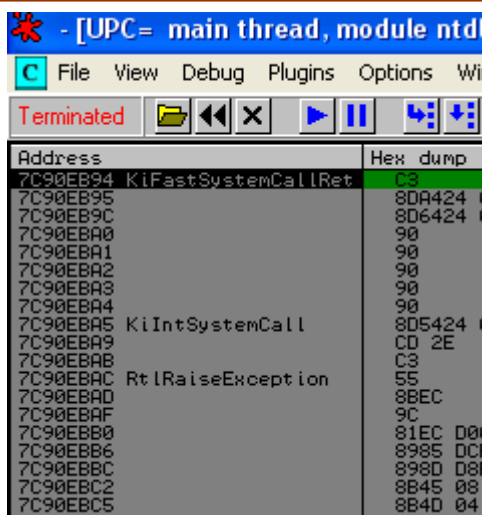
N? bytes:

Bytes

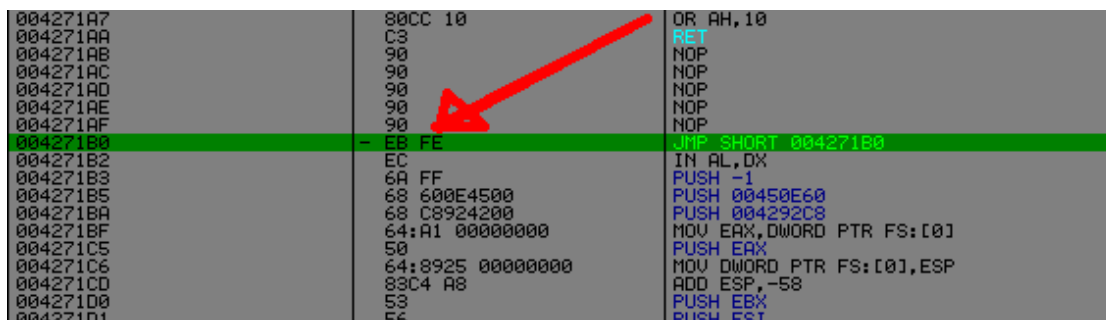
Direction:

To change by:

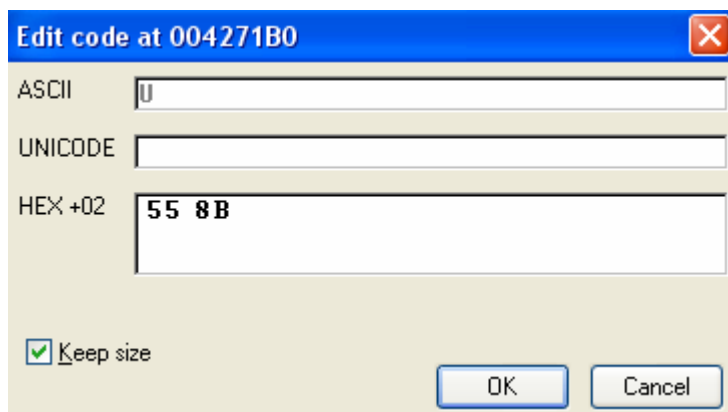
حالا برنامه PUPE را ببندید و به OillyDBG برگردید و کلید F9 را بزنید تا برنامه به طور کامل اجرا شود:



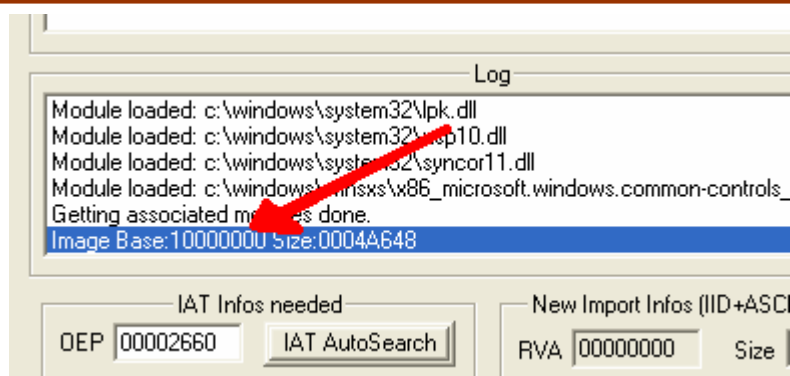
همانطور که می بینید برنامه Terminate شده اما پیغامی داده نشده. حالا به پروسه جدید خودمان Attach می کنیم. در منوی File گزینه Attach را بزنید. و پروسه ی جدید (که در کامپیوتر من PID آن D00 بود) را انتخاب کنید و گزینه Attach را بزنید و بعد به خط 4271B0 بروید، همانطور که می بینید در این خط بایت های EBFE قرار دارد:



خوب، حالا باید این بایت ها را همانند قبلش کنیم. قبلا این بایت ها بوده : 558B
پس کلیدهای CTRL + E را بزنید و همانند تصویر بایت ها را به حالت قبل برگردانید:



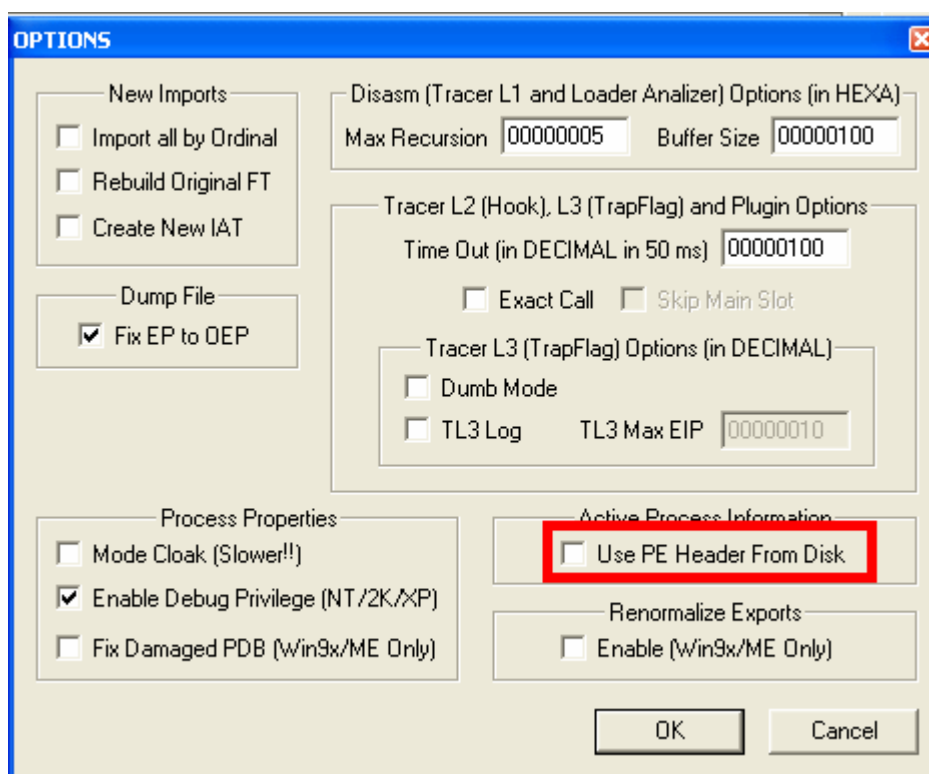
کلید OK را بزنید و بعد هم بر روی OEP برنامه کلیک راست کرده و گزینه here new origin را بزنید. حالا هم از فایل دامپ بگیرید. (ترجیحا با PeTools این کار را بکنید).
حالا ImportREC را باز کنید. و پروسه مورد نظر را در ImportREC باز کنید:



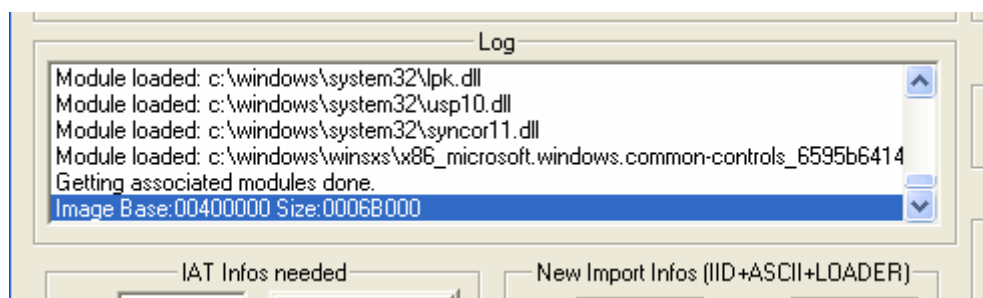
همانطور که می بینید ImportREC می گوید که ImageBase (آدرسی از حافظه که فایل ما در آنجا قرار می گیرد.) برابر 10000000 است. چه طور ممکنه؟... ما می دانیم که OEP ما برابر 4271B0 است. امکان ندارد که ImageBase ما برابر 10000000 باشد. چرا که آدرس تمامی خطوط در برنامه از ImageBase بیشتر است. اما در اینجا اینطور نیست ☹ پس ImportREC، ImageBase را اشتباه به دست آورده. حتی اگر OEP درست را در برنامه وارد کنید و گزینه IAT Auto-search را بزنید، برنامه می گوید: can't find anything...good... خوب، با توجه به اینکه ImageBase را اشتباه به دست آورده است. طبیعی است که نمی تواند چیزی پیدا کند ☹ حالا چرا ImageBase را اشتباه به دست آورده؟... به این خاطر که در تنظیمات ImportREC گزینه ای وجود دارد که تیک دارد:

Use PE Header from Disk

اگر این گزینه تیک داشته باشد، برنامه به جای اینکه ImageBase پروسه را پیدا کند. می رود و ImageBase فایل را در هارد پیدا می کند. پس گزینه Options را بزنید و تیک این گزینه را بردارید:



کلید OK را بزنید و دوباره پروسه را در ImportREC باز کنید:

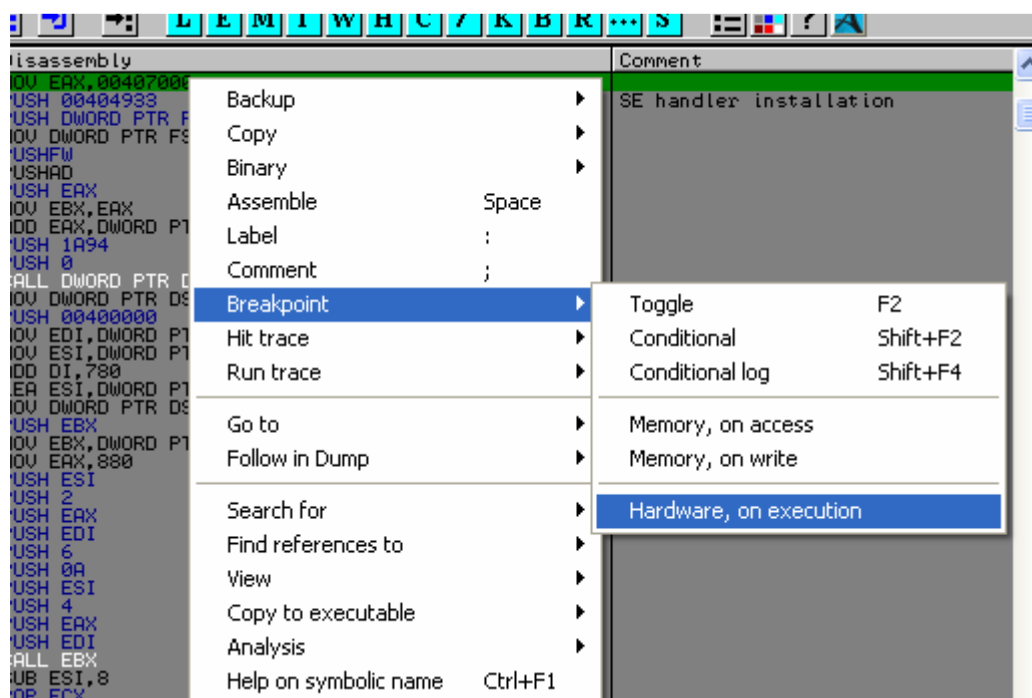


همانطور که می بینید اینبار ImageBase درست نوشته شده ☺ خوب... از این به بعد دیگر مشکلی نیست، کافیه OEP را بدهید و فایل خود را فیکس کنید.

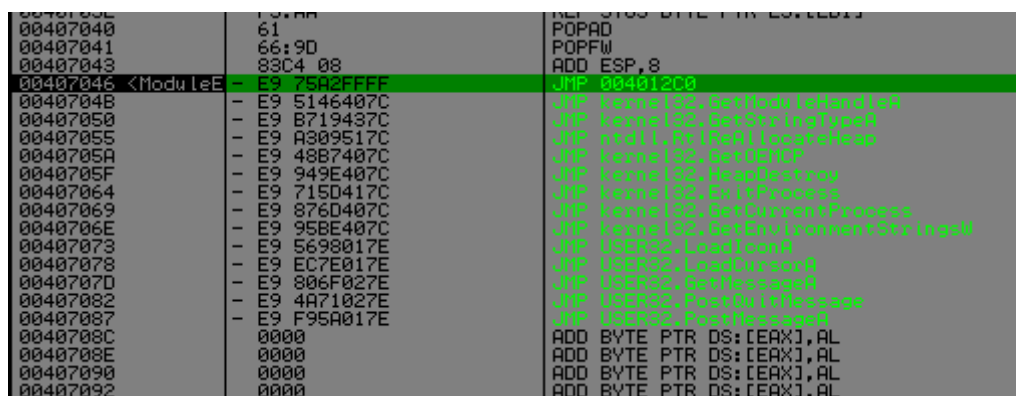
PeTite

فایل مورد نظر را در OllyDBG باز کنید. برای یافتن OEP یک روش خاص برای PeTite وجود دارد. آن هم اینکه چون PeTite از Self-modifying (یعنی کدهای خودش را تغییر می دهد تا کار آنالیز پکر را سخت کند.) استفاده می کند و دستوری که ما را به OEP می برد. درست در Entry point فایل قرار می گیرد. ما می توانیم با hardware breakpoint گذاشتن روی Entry point فایل، OEP را پیدا کنیم.

بر روی Entry point فایل کلیک راست کرده، گزینه Breakpoint و سپس گزینه hardware, on execution را بزنید:

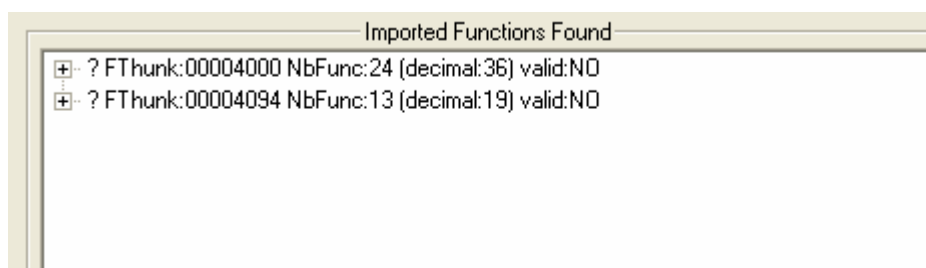


حالا کلید F9 را بزنید تا در Entry point فایل دوباره متوقف شوید (یک خطا قبل از رسیدن به این نقطه اتفاق می افتد که با Shift + F9 آن را رد کنید.):



می بینید؟... اینجا همان Entry point ماست. اما کدهایش تغییر کرده. این پرش هم درست به OEP می رود. یکبار کلید F8 را بزنید تا به OEP برسید.

حالا ImportREC را بزنید. OEP را به برنامه بدهید. و سایر مراحل لازم را انجام دهید. می بینید که چند تابع ما Invalid باشد (Redirect شده است.):



گزینه Show Invalid را بزنید و بر روی توابع کلیک راست کرده و گزینه Trace Level1 (disasm) را بزنید. حالا دیگر مشکلی نیست. دامپ می گیرید و فیکس می کنید.

راه حل دیگر:

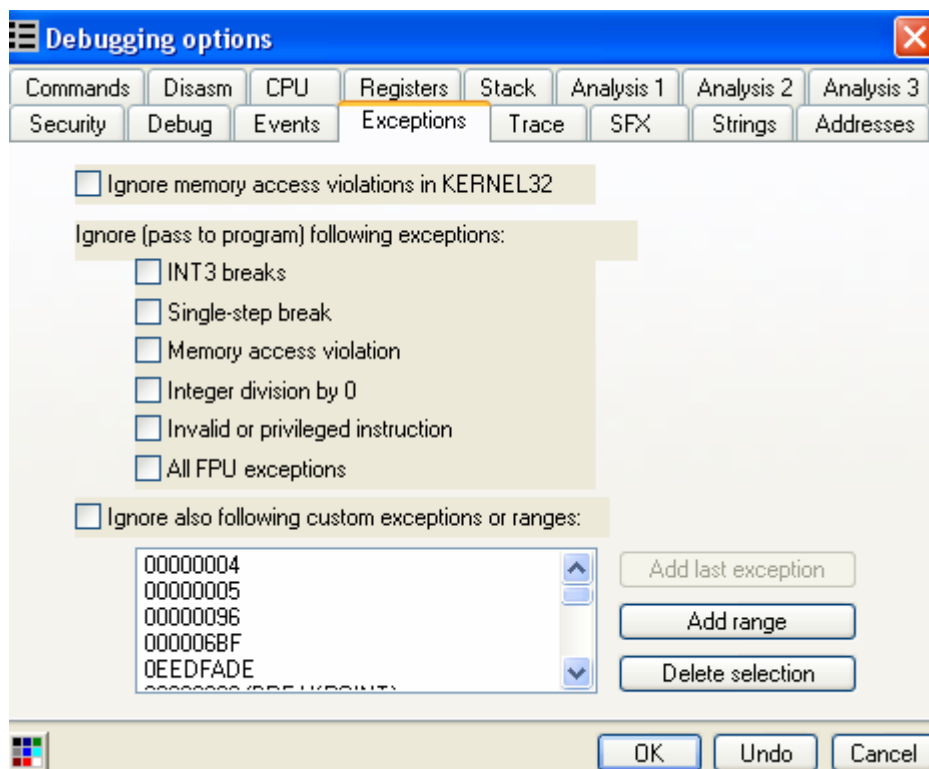
برای یافتن OEP می توانیم از ساختار زبان برنامه نویسی استفاده کنیم. بر روی تابع GetVersion یک Breakpoint بگذارید. (در CommandBar تایپ کنید: BP GetVersion و کلید Enter را بزنید) برنامه را با کلید F9 اجرا کنید تا در تابع GetVersion متوقف شوید. کلید ALT + F9 را بزنید تا وارد کدهای فایل بشوید:

90	NOP	
90	NOP	
90	NOP	
90	NOP	
55	PUSH EBP	
8BEC	MOV EBP,ESP	
6A FF	PUSH -1	
68 F8404000	PUSH 004040F8	
68 F4104000	PUSH 004010F4	
64:A1 00000000	MOV EAX,DWORD PTR FS:[0]	SE handler installation
50	PUSH EAX	
64:8925 00000000	MOV DWORD PTR FS:[0],ESP	
83EC 58	SUB ESP,58	
53	PUSH EBX	
56	PUSH ESI	
57	PUSH EDI	
8965 E8	MOV DWORD PTR SS:[EBP-18],ESP	
FF15 58404000	CALL DWORD PTR DS:[404058]	kernel32.GetVersion
33D2	XOR EDX,EDX	
8AD4	MOV DL,AH	
8915 54564000	MOV DWORD PTR DS:[405654],EDX	
8BC8	MOV ECX,EAX	
31E1 FF000000	AND ECX,0FF	
8900 50564000	MOV DWORD PTR DS:[405650],ECX	
C1E1 08	SHL ECX,8	
03CA	ADD ECX,EDX	
8900 4C564000	MOV DWORD PTR DS:[40564C],ECX	
C1E8 10	SHR EAX,10	
A3 48564000	MOV DWORD PTR DS:[405648],EAX	
33F6	XOR ESI,ESI	
56	PUSH ESI	

همانطور که می بینید چند خط بالاتر از تابع GetVersion یعنی خط 4012C0، همان OEP ماست.

PCGuard

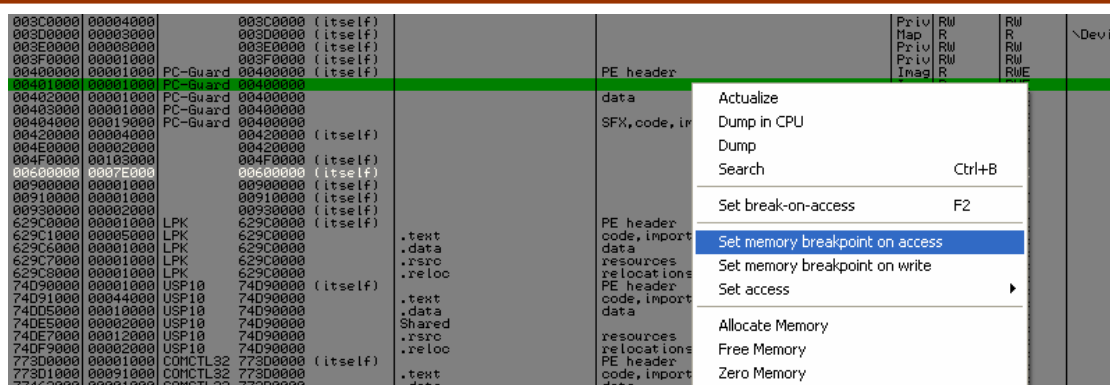
فایل مورد نظر را در OllyDBG باز کنید. (توجه کنید که این فایل محدودیت اجرا دارد. یعنی بعد از چند بار اجرا شدن دیگر در سیستم شما اجرا نمی شود. برای حل این مشکل می توانید از نرم افزارهایی مثل Trial-Reset استفاده کنید.)
برای یافتن OEP می توانیم از Memory Breakpoint استفاده کنیم. ابتدا باید آخرین خطایی که در برنامه اتفاق می افتد را پیدا کنیم. در منوی Options گزینه Debugging options را بزنید و وارد قسمت Exceptions بشوید. حالا تمامی تیک ها را بردارید تا در صورت اتفاق افتادن خطا برنامه متوقف شود.



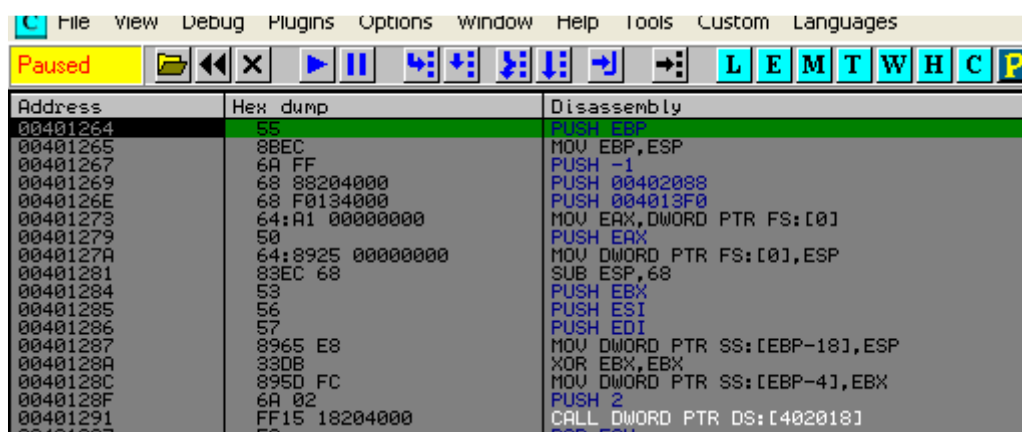
حالا کلید OK را بزنید و با Shift + F9 برنامه را اجرا کنید. چون باید آخرین خطایی که در برنامه اتفاق می افتد را پیدا کنیم. یکبار تعداد خطاهایی که در برنامه اتفاق می افتد را بشمارید. من ۹ بار کلید Shift + F9 را زدم، تا برنامه اجرا شد.
برنامه را ری استارت کنید و این بار ۸ بار کلید Shift + F9 را بزنید تا به آخرین خطا برسید (این کدها را من آنالیز کردم تا به این شکل درآمد):

0041B049	0041B049 27	OR ESP, DWORD PTR DS:[ECX+ECX*4+27]
0041B04D	EB 01	JMP SHORT 0041B058
0041B04F	E3 EB	JECXZ SHORT 0041B03C
0041B051	01D4	ADD ESP, EDX
0041B053	F1	INT1
0041B054	EB 01	JMP SHORT 0041B057
0041B056	89F7	MOV EDI, ESI
0041B058	F7EB	IMUL EBX
0041B05A	01E3	ADD EBX, ESP
0041B05C	EB 01	JMP SHORT 0041B05F
0041B05E	D4 EB	AAM 0EB
0041B060	A6	CMPS BYTE PTR DS:[ESI], BYTE PTR ES:[EDI]
0041B061	EB 01	JMP SHORT 0041B064
0041B063	89EB	MOV EBX, EBP
0041B065	01E3	ADD EBX, ESP
0041B067	EB 01	JMP SHORT 0041B06A
0041B069	D4 85	AAM 85
0041B06B	E4 9C	IN AL, 9C
0041B06D	EB 01	JMP SHORT 0041B070
0041B06F	05 9D	ADD 9D

اینجا جایی که کدهای فایل اصلی ما Decrypt شده. پس یک Memory Breakpoint بر روی اولین سکشن زیر PeHeader بگذارید:



حالا هم کلید F9 را بزنید:



همانطور که می بینید وارد سکشن اصلی برنامه شدیم و به OEP رسیدیم ☺
خوب... بعد از یافتن OEP دیگر کار سختی در پیش ندارید. دامپ می گیرد و فیکس می کنید.

Gooran_Ware UnpackMe#1

فایل مورد نظر را در OllyDBG باز کنید:

0042D140	00	DB 00
0042D14E	00	DB 00
0042D14F	00	DB 00
0042D150	<ModuleE \$ 60	PUSHAD
0042D151	. BE 00204200	MOV ESI,00422000
0042D156	. 8DBE 00F0F0FF	LEA EDI,DWORD PTR DS:[ESI+FFFFF000]
0042D15C	. 57	PUSH EDI
0042D15D	. 83CD FF	OR EBP,FFFFFFFF
0042D160	. EB 10	JMP SHORT 0042D172
0042D162	90	NOP
0042D163	90	NOP
0042D164	90	NOP
0042D165	90	NOP
0042D166	90	NOP
0042D167	90	NOP
0042D168	> 8A06	MOV AL,BYTE PTR DS:[ESI]

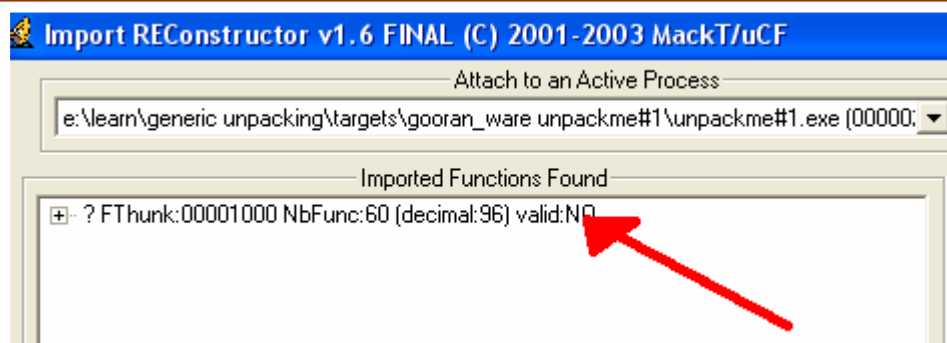
چون فایل با UPX پک شده است، به راحتی می توان از طریق رجیستر ESP، به OEP رسید. کلید F8 را بزنید تا مقدار رجیستر ESP تغییر کند. بعد بر روی مقدار رجیستر ESP یک Hardware breakpoint on access بگذارید و کلید F9 را بزنید تا به این آدرس برسید:

00401400	PUSH EBX
00401401	PUSH EDI
00401402	CALL EBP
00401403	POP EAX
00401404	POPAD
00401405	LEA EAX,DWORD PTR SS:[ESP-80]
00401406	PUSH 0
00401407	CMP ESP,EAX
00401408	JNZ SHORT 00402DEC
00401409	SUB ESP,-80
0040140A	MOV EAX,DWORD PTR DS:[40113C]
0040140B	MOV DWORD PTR DS:[42D400],EAX
0040140C	MOV EAX,DWORD PTR DS:[4010D4]
0040140D	MOV DWORD PTR DS:[42D404],EAX
0040140E	MOV EAX,DWORD PTR DS:[401020]
0040140F	MOV DWORD PTR DS:[42D408],EAX
00401410	MOV DWORD PTR DS:[40113C],0042D340
00401411	MOV DWORD PTR DS:[4010D4],0042D348
00401412	MOV DWORD PTR DS:[401020],0042D34F
00401413	CALL 00401414
00401414	DB 00
00401415	DB 00
00401416	DB 00
00401417	DB 00
00401418	DB 00
00401419	DB 00
0040141A	DB 00
0040141B	DB 00
0040141C	DB 00
0040141D	DB 00
0040141E	DB 00
0040141F	DB 00
00401420	MOV EAX,DWORD PTR DS:[42D400]
00401421	PUSH EAX
00401422	RET

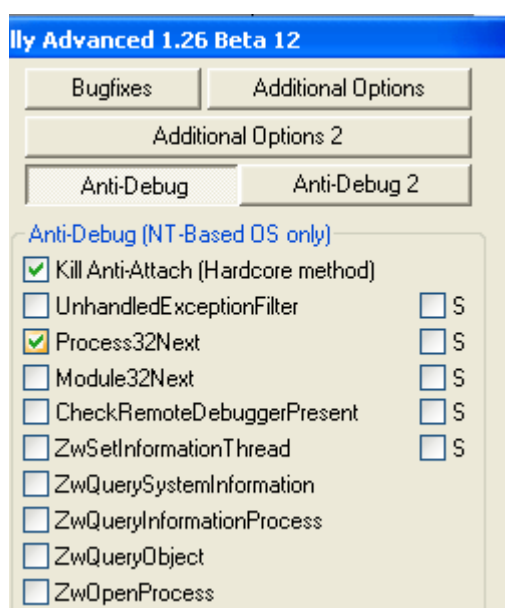
اگر برنامه را با کلید F8 به دقت اجرا کنید، متوجه خواهید شد که این کدها قصد دارند قسمتی از توابع ما را Redirect کنند. ولی مشکلی نیست. شما با F8 ادامه دهید تا به دستور JMP 4014F0 برسید. پرشی که ما را به OEP می برد:

004014C6	- FF25 F0104000	JMP DWORD PTR DS:[401040F0]
004014C8	- FF25 5C114000	JMP DWORD PTR DS:[40113C]
004014D2	- FF25 64104000	JMP DWORD PTR DS:[401064]
004014D8	- FF25 CC104000	JMP DWORD PTR DS:[4010CC]
004014DE	- FF25 88104000	JMP DWORD PTR DS:[401088]
004014E4	- FF25 B8104000	JMP DWORD PTR DS:[4010B8]
004014EA	- FF25 3C114000	JMP DWORD PTR DS:[40113C]
004014F0	68 50644100	PUSH 00416450
004014F5	E8 F0FFFFFF	CALL 004014EA
004014FA	0000	ADD BYTE PTR DS:[EAX],AL
004014FC	0000	ADD BYTE PTR DS:[EAX],AL
004014FE	0000	ADD BYTE PTR DS:[EAX],AL
00401500	3000	XOR BYTE PTR DS:[EAX],AL
00401502	0000	ADD BYTE PTR DS:[EAX],AL
00401504	40	INC EAX
00401505	0000	ADD BYTE PTR DS:[EAX],AL
00401507	0000	ADD BYTE PTR DS:[EAX],AL
00401509	0000	ADD BYTE PTR DS:[EAX],AL

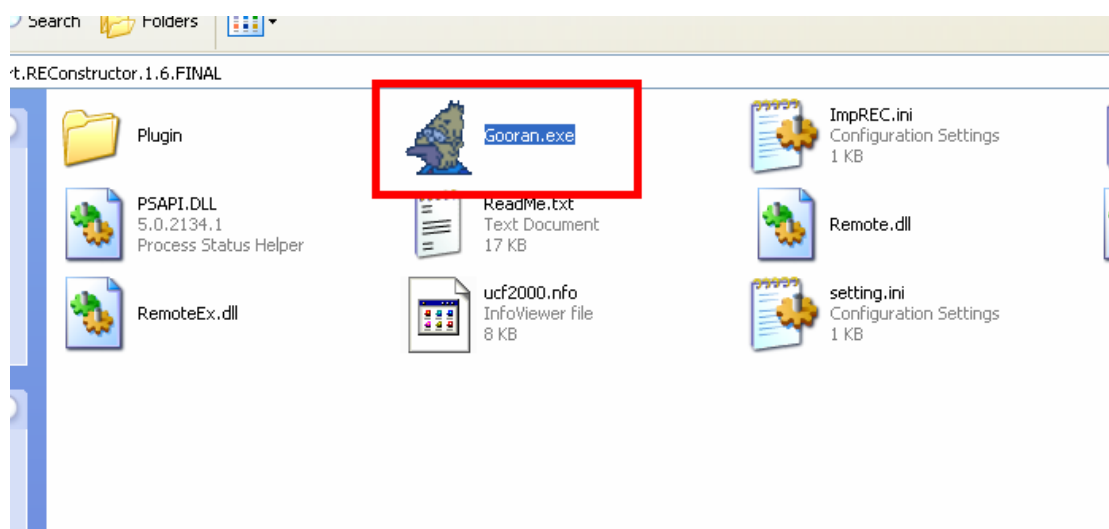
خوب، حالا از فایل دامپ بگیریید و ImportREC را باز کنید:



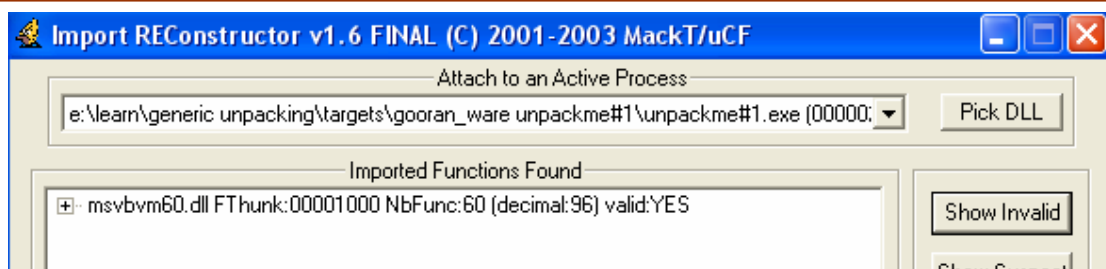
همانطور که می بینید چند تا از توابع ما Invalid می باشد. برای یافتن این توابع می توانیم از قابلیت Trap Flag استفاده کنیم. ابتدا برنامه را در داخل OllyDBG به طور کامل باز کنید. با اجرا شدن برنامه ممکن است OllyDBG شما بسته شود. برای حل این مشکل می توانید از پلاگین OllyAdvanced استفاده کنید. (این پلاگین را در پوشه پلاگین های OllyDBG کپی کنید). در منوی Plugins گزینه OllyAdvanced و سپس گزینه Options را بزنید در قسمت Anti-Debug تیک گزینه Process32Next را بزنید:



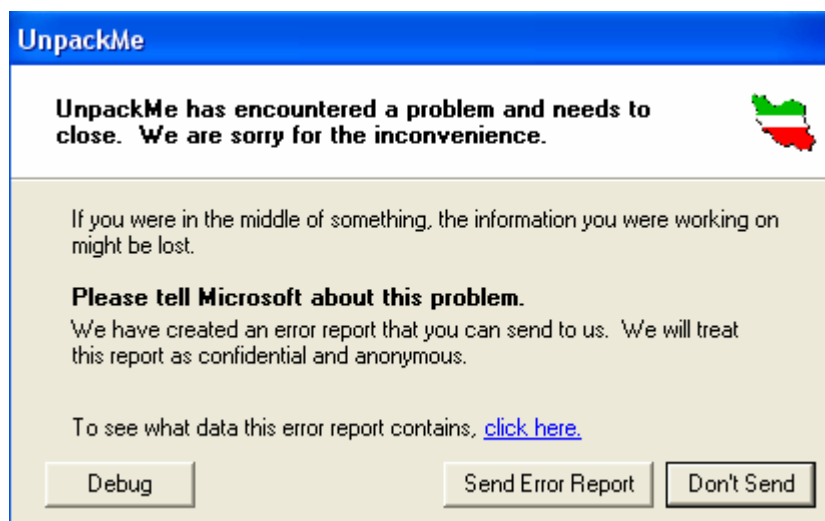
البته از پلاگین های دیگر هم می توان برای از بین بردن این تابع استفاده کرد. حالا دیگر برنامه بدون هیچ مشکلی در OllyDBG اجرا می شود. اگر UnpackMe برنامه ImportREC را هم می بندد. خیلی راحت می توانید نام فایل ImportREC را عوض کنید. مثلاً بگذارید Gooran.exe و بعد ImportREC را اجرا کنید:



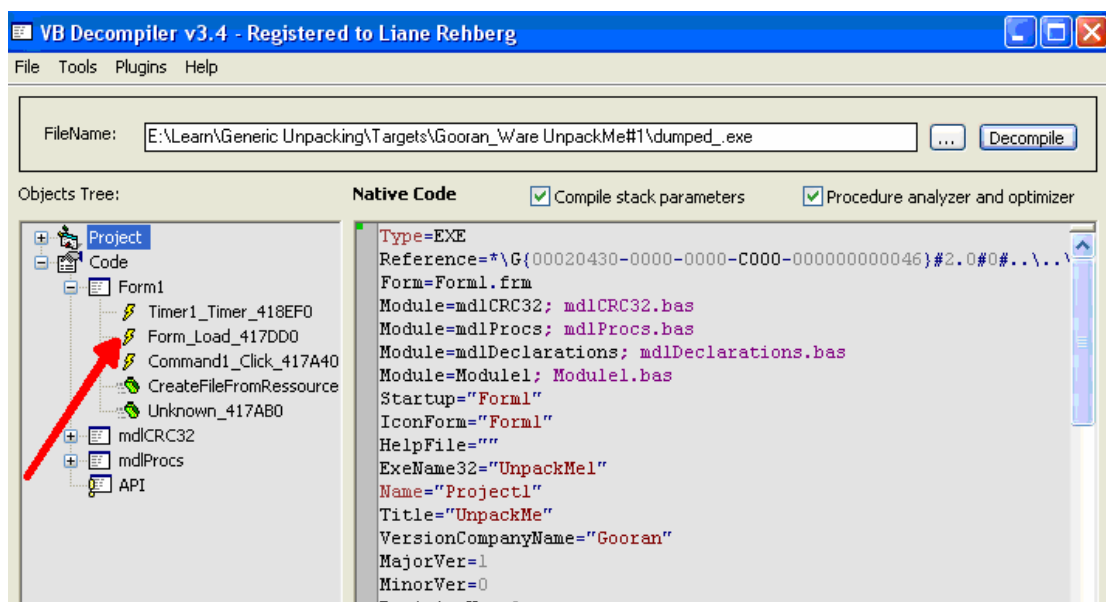
حالا دیگر برنامه قادر به شناسایی ImportREC نیست و از ImportREC هم می توانید به راحتی استفاده کنید. حالا دوباره ImportREC را باز کنید. OEP را بدهید و سایر مراحل را انجام دهید. حالا گزینه Show invalid را بزنید و بر روی توابع Redirect شده کلیک راست کرده و گزینه (Trap flag) Trace Level3 را بزنید تا تمامی توابع Valid شود.



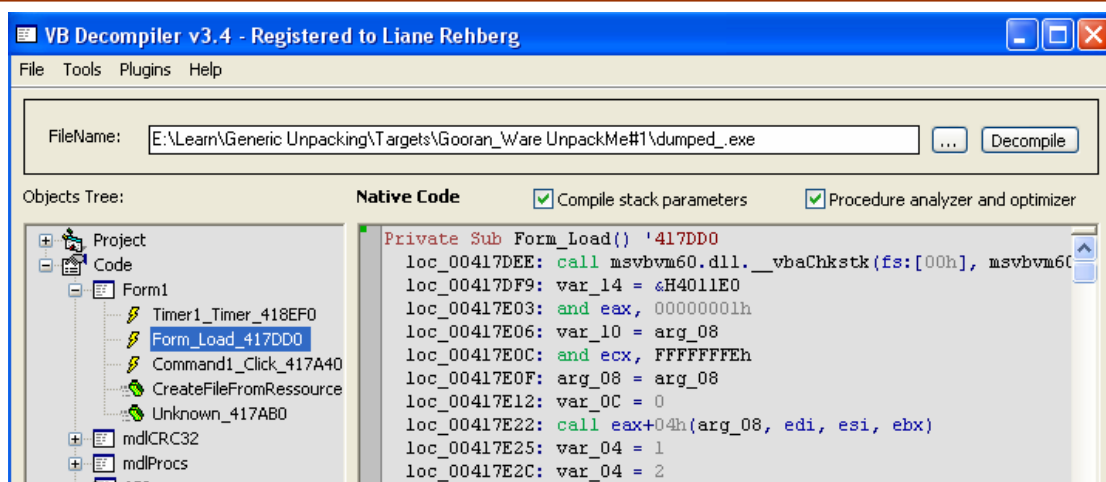
حالا فایل دامپ خود را فیکس کنید و فایل آنپک شده را اجرا کنید:



حالا بهتر است فایل آنپک شده را در VB Decompiler Pro باز کنیم، تا Event های برنامه را بگیریم (برای باز کردن فایل آنپک شده در این برنامه، در منوی File گزینه Open VB Program را بزنید):

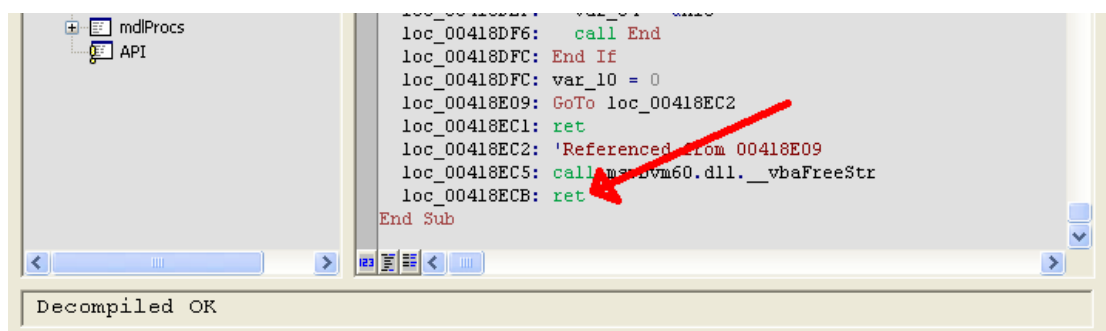


خوب، حالا در تصویر بالا بر روی گزینه Form_Load_417DD0 (در قسمت Object Tree) دو بار کلیک کنید:



Form_Load دستوراتی است که در هنگام اجرا شدن یک فرم در برنامه های VB اجرا می شود. چون فایل آپک شده ما در هنگام اجرا شدن خطا می داد. لذا ممکن است دلیل اجرا نشدن برنامه در همین تابع باشد.

ابتدا و انتهای Form_load را یادداشت کنید و برنامه را ببندید. ابتدای Form_load همانطور که در تصویر بالا می بینید خط 417DD0 می باشد و انتهای آن:



خط 418ECB می باشد. حالا فایل آپک شده را با OllyDBG باز کنید و بر روی خطوط ابتدایی و انتهایی Form_load Breakpoint بگذارید. و بعد برنامه را با F9 اجرا کنید:

00417DCD	90	NOP
00417DCE	90	NOP
00417DCF	90	NOP
00417DD0	55	PUSH EBP
00417DD1	8BEC	MOV EBP,ESP
00417DD3	83EC 18	SUB ESP,18
00417DD6	68 B6124000	PUSH <JMP.&msvbvm60.__vbaExceptionHandler>
00417DD8	64:A1 00000000	MOV EAX,DWORD PTR FS:[0]
00417DE1	50	PUSH EAX
00417DE2	64:8925 00000000	MOV DWORD PTR FS:[0],ESP
00417DE9	B8 14020000	MOV EAX,214
00417DEE	E8 BD94FEFF	CALL <JMP.&msvbvm60.__vbaChkstk>
00417DF3	53	PUSH EBX
00417DF4	56	PUSH ESI
00417DF5	57	PUSH EDI
00417DF6	8965 E8	MOV DWORD PTR SS:[EBP-18],ESP
00417DF9	C745 EC E0114000	MOV DWORD PTR SS:[EBP-14],004011E0
00417E00	8B45 08	MOV EAX,DWORD PTR SS:[EBP+8]
00417E03	83E0 01	AND EAX,1
00417E06	8945 F0	MOV DWORD PTR SS:[EBP-10],EAX
00417E09	8B4D 08	MOV ECX,DWORD PTR SS:[EBP+8]
00417E0C	83E1 FE	AND ECX,FFFFFFFE
00417E0F	894D 08	MOV DWORD PTR SS:[EBP+8],ECX

اینجا ابتدای تابع Form_load ماست. برنامه را با F8 ادامه دهید تا به این خط برسید:

00417E74	C745 8CFEFFFF 00000000	MOV DWORD PTR SS:[EBP-1C],0
00417E7E	C745 FC 03000000	MOV DWORD PTR SS:[EBP-4],3
00417E85	E8 5AF1FFFF	CALL 00416FE4
00417E8A	8965 88FEFFFF	MOV DWORD PTR SS:[EBP-178],EAX
00417E90	FF15 44104000	CALL DWORD PTR DS:[<&msvbvm60.__vbaSetSystemError>]
00417E96	83BD 88FEFFFF 00	CMP DWORD PTR SS:[EBP-178],0
00417E9D	74 0D	JE SHORT 00417EAC
00417E9F	C745 FC 04000000	MOV DWORD PTR SS:[EBP-4],4
00417EA6	FF15 20104000	CALL DWORD PTR DS:[<&msvbvm60.__vbaEnd>]
00417EAC	C745 FC 06000000	MOV DWORD PTR SS:[EBP-4],6
00417EB3	6A FF	PUSH -1
00417EB5	FF15 58104000	CALL DWORD PTR DS:[<&msvbvm60.__vbaOnError>]
00417EBB	C745 FC 07000000	MOV DWORD PTR SS:[EBP-4],7
00417EC2	68 FF000000	PUSH 0FF
00417EC7	FF15 7C104000	CALL DWORD PTR DS:[<&msvbvm60.vbaSpaceBstr>]

الان یک نگاهی به رجیسترها بیندازید:

```

Registers (FPU)
EAX 00000000
ECX 7C813093 kernel32.IsDebuggerPresent
EDX 7C97C0D8 ntdll.7C97C0D8
EBX 00000001
ESP 0012F8D4
EBP 0012FB14
ESI 0012FBF0
EDI 0012FB20
EIP 00417E8A dumped_.00417E8A

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 7FFDF000(FFF)
T 0 GS 0000 NULL
D 0
O 0 LastErr ERROR_SUCCESS (00000000)
EFL 00000246 (NO,NB,E,BE,NS,PE,GE,LE)
ST0 empty 0.0000000426493561690e-4933
ST1 empty -6.0966102422391142400e-2139
ST2 empty -2.8675029113078046720e-890
ST3 empty -1.0798418727110411520e-2138
ST4 empty 4.8693466143822182400e-4932
ST5 empty -NaN FFFF 804E7568 804E2490
ST6 empty 1.00000000000000000000
ST7 empty 1.00000000000000000000

3 2 1 0 E S P U O Z D I
FST 4020 Cond 1 0 0 0 Err 0 0 1 0 0 0 0 0 (EQ)
FCW 137F Prec NEAR,64 Mask 1 1 1 1 1 1

```

مقدار رجیستر ECX برابر IsDebuggerPresent است. این یعنی برنامه در این خطوط می خواهد این تابع را اجرا کند. پرشی که در خط 417E9D قرار دارد، تصمیم گیرنده است. یعنی اگر این پرش انجام شود. نشان می دهد که دیباگر وجود ندارد و اگر این پرش انجام نشود یعنی دیباگر وجود دارد. آنوقت تابع __vbaEnd اجرا می شود و برنامه بسته می شود. (البته من از پلاگین Hide Debugger استفاده می کنم و این تابع مشکلی برای من ایجاد نمی کند). ...بهرتر است این پرش را به JMP تغییر دهید تا در صورتی که حتی IsDebuggerPresent مقدار یک را برگرداند، باز هم برنامه بسته نشود:

00417E7E	C745 FC 00000000	MOV DWORD PTR SS:[EBP-4],3
00417E85	E8 5AF1FFFF	CALL 00416FE4
00417E8A	8985 88FEFFFF	MOV DWORD PTR SS:[EBP-178],EAX
00417E90	FF15 44104000	CALL DWORD PTR DS:[<&msvbum60.__vbaSetSystemEx
00417E96	83BD 88FEFFFF 00	CMF DWORD PTR SS:[EBP-178],0
00417E9D	EB 00	JMP SHORT 00417EAC
00417E9F	C745 FC 04000000	MOV DWORD PTR SS:[EBP-4],4
00417EA6	FF15 20104000	CALL DWORD PTR DS:[<&msvbum60.__vbaEnd>]
00417EAC	C745 FC 06000000	MOV DWORD PTR SS:[EBP-4],6
00417EB3	6A FF	PUSH -1
00417EB5	FF15 58104000	CALL DWORD PTR DS:[<&msvbum60.__vbaOnError>]
00417EB8	C745 FC 07000000	MOV DWORD PTR SS:[EBP-4],7
00417EC2	68 FF000000	PUSH 0FF
00417EC7	FF15 7C104000	CALL DWORD PTR DS:[<&msvbum60.rtcSpaceBstr>]
00417ECD	8BD0	MOV EDX,EAX
00417ECF	8D4D 00	LEA ECX,DWORD PTR SS:[EBP-30]
00417ED2	FF15 60114000	CALL DWORD PTR DS:[<&msvbum60.__vbaStrMove>]
00417ED8	8BD0	MOV EDX,EAX
00417EDA	8B4D 08	MOV ECX,DWORD PTR SS:[EBP+8]
00417EDD	83C1 3C	ADD ECX,3C
00417EE0	FF15 24114000	CALL DWORD PTR DS:[<&msvbum60.__vbaStrCopy>]
00417EE6	8D4D 00	LEA ECX,DWORD PTR SS:[EBP-30]
00417EE9	FF15 78114000	CALL DWORD PTR DS:[<&msvbum60.__vbaFreeStr>]
00417EEF	C745 FC 00000000	MOV DWORD PTR SS:[EBP-4],0

خوب، باز هم برنامه را با F8 ادامه دهید، تا به اینجا برسید:

004183FA	50	PUSH EAX
004183FB	8D4D 90	LEA ECX,DWORD PTR SS:[EBP-70]
004183FE	51	PUSH ECX
004183FF	6A 10	PUSH 10
00418401	FF15 24104000	CALL DWORD PTR DS:[<&msvbm60.__vbaFreeVarList>]
00418407	83C4 44	ADD ESP,44
0041840A	0FBF95 7CFEFFFF	MOVSX EDX,WORD PTR SS:[EBP-184]
00418411	85D2	TEST EDX,EDX
00418413	0F84 55030000	JE 0041876E
00418419	C745 FC 0E000000	MOV DWORD PTR SS:[EBP-4],0E
00418420	833D F0E44100 00	CMP DWORD PTR DS:[41E4F0],0
00418427	75 1C	JNZ SHORT 00418445
00418429	68 F0E44100	PUSH 0041E4F0
0041842E	68 64704100	PUSH 00417064
00418433	FF15 14114000	CALL DWORD PTR DS:[<&msvbm60.__vbaNew2>]
00418439	C785 20FEFFFF F0E44100	MOV DWORD PTR SS:[EBP-1E0],0041E4F0
00418443	EB 0A	JMP SHORT 0041844F
00418445	C785 20FEFFFF F0E44100	MOV DWORD PTR SS:[EBP-1E0],0041E4F0
0041844F	8B85 20FEFFFF	MOV EAX,DWORD PTR SS:[EBP-1E0]
00418455	8B08	MOV ECX,DWORD PTR DS:[EAX]
00418457	898D 7CFEFFFF	MOV DWORD PTR SS:[EBP-184],ECX
0041845D	8D55 B4	LEA EDX,DWORD PTR SS:[EBP-4C]
00418460	52	PUSH EDX
00418461	8B85 7CFEFFFF	MOV EAX,DWORD PTR SS:[EBP-184]
00418467	8B08	MOV ECX,DWORD PTR DS:[EAX]
00418469	8B95 7CFEFFFF	MOV EDX,DWORD PTR SS:[EBP-184]
0041846F	52	PUSH EDX

این یک پرش بسیار بلند است. بر روی آن کلید Enter را بزنید تا ببینیم به کجا می رود:

14000	LEA EAX,DWORD PTR SS:[EBP-34]	
	PUSH EAX	
	PUSH 3	
	CALL DWORD PTR DS:[<&msvbm60.__vbaFreeStrList>]	msvbm60.__vbaFreeStrList
	ADD ESP,10	
14000	LEA ECX,DWORD PTR SS:[EBP-4C]	
	CALL DWORD PTR DS:[<&msvbm60.__vbaFreeObj>]	msvbm60.__vbaFreeObj
04000	LEA ECX,DWORD PTR SS:[EBP-60]	
	CALL DWORD PTR DS:[<&msvbm60.__vbaFreeVar>]	msvbm60.__vbaFreeVar
10000000	MOV DWORD PTR SS:[EBP-4],10	
04000	CALL DWORD PTR DS:[<&msvbm60.__vbaEnd>]	msvbm60.__vbaEnd
12000000	MOV DWORD PTR SS:[EBP-4],12	
44100 00	CMP DWORD PTR DS:[41E4F0],0	
	JNZ SHORT 0041879A	
100	PUSH 0041E4F0	
100	PUSH 00417064	
14000	CALL DWORD PTR DS:[<&msvbm60.__vbaNew2>]	msvbm60.__vbaNew2
FFFFFF F0E44100	MOV DWORD PTR SS:[EBP-1FC],0041E4F0	
FFFFFF F0E44100	MOV DWORD PTR SS:[EBP-1FC],0041E4F0	
FFFFFF	MOV ECX,DWORD PTR SS:[EBP-1FC]	
FFFFFF	MOV EDX,DWORD PTR DS:[ECX]	
FFFFFF	MOV DWORD PTR SS:[EBP-184],EDX	
	LEA EAX,DWORD PTR SS:[EBP-30]	
	PUSH EAX	
	PUSH 65	

می بینید؟...این پرش تابع vbaEnd را رد می کند. یعنی این پرش باید حتما انجام شود، چرا که اگر انجام نشود به تابع vbaEnd برمی خوریم و پروسه بسته می شود. پس این پرش را هم JMP کنید و برنامه را با F8 ادامه دهید:

0041840A	0FBF95 7CFEFFFF	MOVSX EDX,WORD PTR SS:[EBP-184]
00418411	85D2	TEST EDX,EDX
00418413	0F84 55030000	JNE 0041876E
00418418	90	NOP
00418419	C745 FC 0E000000	MOV DWORD PTR SS:[EBP-4],0E
00418420	833D F0E44100 00	CMP DWORD PTR DS:[41E4F0],0
00418427	75 1C	JNZ SHORT 00418445
00418429	68 F0E44100	PUSH 0041E4F0
0041842E	68 64704100	PUSH 00417064
00418433	FF15 14114000	CALL DWORD PTR DS:[<&msvbm60.__vbaNew2>]
00418439	C785 20FEFFFF F0E44100	MOV DWORD PTR SS:[EBP-1E0],0041E4F0
00418443	EB 0A	JMP SHORT 0041844F
00418445	C785 20FEFFFF F0E44100	MOV DWORD PTR SS:[EBP-1E0],0041E4F0
0041844F	8B85 20FEFFFF	MOV EAX,DWORD PTR SS:[EBP-1E0]
00418455	8B08	MOV ECX,DWORD PTR DS:[EAX]
00418457	898D 7CFEFFFF	MOV DWORD PTR SS:[EBP-184],ECX
0041845D	8D55 B4	LEA EDX,DWORD PTR SS:[EBP-4C]
00418460	52	PUSH EDX
00418461	8B85 7CFEFFFF	MOV EAX,DWORD PTR SS:[EBP-184]
00418467	8B08	MOV ECX,DWORD PTR DS:[EAX]

آنقدر با F8 جلو بروید تا به این خط برسید:

00418A88	51	PUSH ECX
00418A89	8D55 B4	LEA EDX,DWORD PTR SS:[EBP-4C]
00418A8C	52	PUSH EDX
00418A8D	6A 02	PUSH 2
00418A8F	FF15 2C104000	CALL DWORD PTR DS:[<&msvbm60.__vbaFreeObjList>]
00418A95	83C4 0C	ADD ESP,0C
00418A98	0FBF85 54FEFFFF	MOVSX EAX,WORD PTR SS:[EBP-1AC]
00418A9F	85C0	TEST EAX,EAX
00418AA1	0F84 55030000	JE 00418DFC
00418AA7	C745 FC 10000000	MOV DWORD PTR SS:[EBP-4],10
00418AAE	833D F0E44100 00	CMP DWORD PTR DS:[41E4F0],0
00418AB5	75 1C	JNZ SHORT 00418AD3
00418AB7	68 F0E44100	PUSH 0041E4F0
00418AB8	68 64704100	PUSH 00417064
00418ABC	FF15 14114000	CALL DWORD PTR DS:[<&msvbm60.__vbaNew2>]
00418AC7	C785 E4FDFFFF F0E44100	MOV DWORD PTR SS:[EBP-21C],0041E4F0
00418AD1	EB 0A	JMP SHORT 00418AD5
00418AD3	C785 E4FDFFFF F0E44100	MOV DWORD PTR SS:[EBP-21C],0041E4F0
00418ADD	8B8D E4FDFFFF	MOV ECX,DWORD PTR SS:[EBP-21C]
00418AE3	8B11	MOV EDX,DWORD PTR DS:[ECX]
00418AE5	8995 7CFEFFFF	MOV DWORD PTR SS:[EBP-184],EDX
00418AEB	8D45 B4	LEA EAX,DWORD PTR SS:[EBP-4C]
00418AEE	50	PUSH EAX
00418AEF	8B8D 7CFEFFFF	MOV ECX,DWORD PTR SS:[EBP-184]

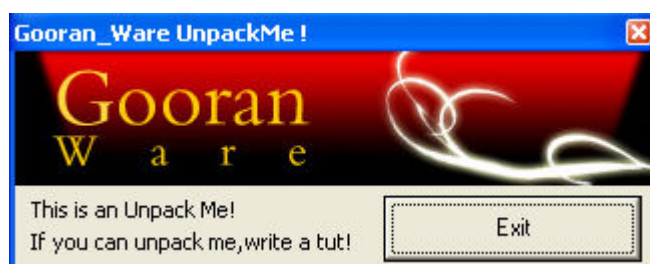
این هم یک پرش دیگر، باز هم بر روی آن کلید Enter را بزنید:

114000	LEA ECX,DWORD PTR SS:[EBP-4C]	msvbm60.__vbaFreeObj
	CALL DWORD PTR DS:[<&msvbm60.__vbaFreeObj>]	
104000	LEA ECX,DWORD PTR SS:[EBP-60]	msvbm60.__vbaFreeVar
15000000	CALL DWORD PTR DS:[<&msvbm60.__vbaFreeVar>]	
104000	MOV DWORD PTR SS:[EBP-4],15	
00000000	CALL DWORD PTR DS:[<&msvbm60.__vbaEnd>]	msvbm60.__vbaEnd
	MOV DWORD PTR SS:[EBP-10],0	
	WAIT	
4100	PUSH 00418ECC	
0000	JMP 00418EE2	
	LEA EDX,DWORD PTR SS:[EBP-48]	
	PUSH EDX	
	LEA EAX,DWORD PTR SS:[EBP-44]	
	PUSH EAX	
	LEA ECX,DWORD PTR SS:[EBP-40]	

می بینید این پرش هم تابع vbaEnd را رد می کند. پس این پرش باید JMP شود:

00	CALL DWORD PTR DS:[<&msvbm60.__vbaFreeObjList>]	msvbm60.__vbaFreeObjList
FFFF	ADD ESP,0C	
	MOVSX EAX,WORD PTR SS:[EBP-1AC]	
	TEST EAX,EAX	
	JMP 00418DFC	
	NOP	
000000	MOV DWORD PTR SS:[EBP-4],13	
00 00	CMP DWORD PTR DS:[41E4F0],0	
	JNZ SHORT 00418AD3	
	PUSH 0041E4F0	
	PUSH 00417064	
00	CALL DWORD PTR DS:[<&msvbm60.__vbaNew2>]	msvbm60.__vbaNew2
FF F0E44100	MOV DWORD PTR SS:[EBP-21C],0041E4F0	
	JMP SHORT 00418AD0	

حالا برنامه را با F9 اجرا کنید:



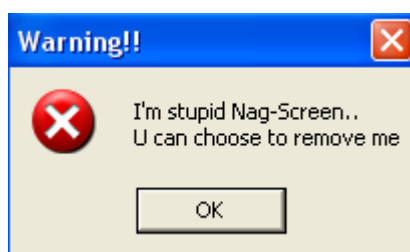
می بینید؟ برنامه بدون هیچ مشکلی اجرا می شود. ما سه پرش را در فایل آنپک شده دستکاری کردیم:

۱. پرش اول تست می کرد، اگر تابع IsDebuggerPresent مقدار یک را برمی گرداند، پروسه بسته می شد.
۲. پرش دوم تست می کرد، اگر مقدار CRC32 مخالف "352C9C25" بود، پیغام Don't send را می داد.
۳. پرش سوم تست می کرد، اگر سایز فایل مخالف 68608 Bytes بود، پیغام Don't send را می داد.

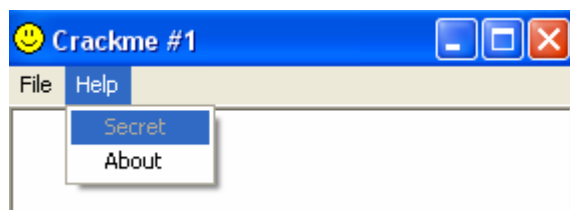
حالا تغییرات را ذخیره کنید. تا کار آنپک این فایل هم تمام شود.

Inline Patching (Level 2)

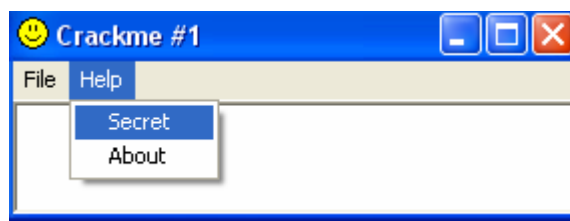
من در بخش ساختار زبان برنامه نویسی گفتم معمولا بعد از نزدیکی های Entry point در زبان C++ تابع GetVersion (یا GetVersionEx در نسخه های جدیدتر) می آید. در نزدیکی های Entry point در زبان دلفی معمولا GetModuleHandleA و در زیر Entry point در VB تابع ThunRTMain می آید... خوب، حالا بیا فرض کنیم که یک فایل پک شده داریم که در C++ نوشته شده است... من می توانم پیام و تابع GetVersion در فایل Kernel32.dll را دستکاری کنم (Hook کنم)... یعنی پیام و خط اول این تابع را به یک پرش تبدیل کنم و این پرش به جایی برود که در آنجا کدهایی قرار دادم. کدهایی که برنامه را کرک می کند. به این ترتیب می توانم بدون آنپک کردن فایل، حتی بدون داشتن OEP و فقط با دانستن اینکه برنامه در چه زبانی نوشته شده، برنامه را کرک کنم. خوب، برای اینکه این موضوع را روشن تر کنم، بذارید مثالی بزنم. در ابتدای این مقاله من EZIP را آنپک کردم. حالا می خواهم بدون اینکه فایل را آنپک کنم با تزریق کدهایی به آن برنامه را کامل کرک کنم. فایل اصلی پک شده با EZIP را باز کنید:



همانطور که می بینید یک Nag Screen نمایش داده شده... من باید این Nag Screen را از بین ببرم. از طرفی:



در منوی Help گزینه Secret وجود دارد که Disable هست. من باید این گزینه را هم دوباره فعال کنم. من قبلا با Hook کردن تابع GetVersion این کار را انجام دادم. حالا پچی که در Attachment این فایل هست را نصب کنید و دوباره برنامه را اجرا کنید:



می بینید؟ خبری از Nag Screen نیست و گزینه Secret هم فعال شده ☺ خوب، فایل پچ شده را در OllyDBG باز کنید، تا به بررسی تغییراتی که انجام دادم بپردازیم:

Address	Hex dump	Disassembly	Comment
0040653E	<ModuleE	PUSHAD	
0040653F	68 09654000	PUSH 00406509	ASCII "kernel32.dll"
00406544	FF15 CCF04000	CALL DWORD PTR DS:[<kernel32.GetModuleHandleA>]	kernel32.GetModuleHandleA
0040654A	50	PUSH EAX	
0040654B	50	PUSH EAX	
0040654C	50	PUSH 00406516	ASCII "VirtualProtect"
00406551	50	PUSH EAX	
00406552	FF15 C8F04000	CALL DWORD PTR DS:[<kernel32.GetProcAddress>]	kernel32.GetProcAddress
00406558	A3 A06E4000	MOV DWORD PTR DS:[406EA0],EAX	
0040655D	58	POP EAX	
0040655E	68 25654000	PUSH 00406525	ASCII "VirtualAlloc"
00406563	50	PUSH EAX	
00406564	FF15 C8F04000	CALL DWORD PTR DS:[<kernel32.GetProcAddress>]	kernel32.GetProcAddress
0040656A	A3 A46E4000	MOV DWORD PTR DS:[406EA4],EAX	
0040656F	58	POP EAX	
00406570	68 32654000	PUSH 00406532	ASCII "GetVersion"
00406575	50	PUSH EAX	
00406576	FF15 C9F04000	CALL DWORD PTR DS:[<kernel32.GetProcAddress>]	kernel32.GetProcAddress
0040657C	A3 A86E4000	MOV DWORD PTR DS:[406EA8],EAX	
00406581	6A 40	PUSH 40	
00406583	68 00300000	PUSH 3000	
00406588	68 00100000	PUSH 1000	
0040658D	6A 00	PUSH 0	
0040659F	FF15 A46E4000	CALL DWORD PTR DS:[406EA4]	
0040659E	A3 AC6E4000	MOV DWORD PTR DS:[406EAC],EAX	
0040659A	BE 006E4000	MOV ESI,00406E00	
0040659F	8BF8	MOV EDI,EAX	
004065A1	B9 13000000	MOV ECX,13	
004065A6	EB 04	REP MOVSB R/E PTR ES:[EDI],R/E PTR DS:[ESI]	

اینجا قبلا کدی قرار نداشت. من با تزریق این کدها تابع GetVersion را Hook کردم. همچنین Entry point فایل را از خط 4070BE به اینجا منتقل کردم.

خوب، بهتر است قبل از اجرای کدها نگاهی به تابع GetVersion بیندازیم (CTRL + G را بزنید و تایپ کنید GetVersion):

7C8111D3	EB BD	JMP SHORT 7C8111E2
7C8111D5	90	NOP
7C8111D6	90	NOP
7C8111D7	90	NOP
7C8111D8	90	NOP
7C8111D9	90	NOP
7C8111DA	64:A1 18000000	MOV EAX,DWORD PTR FS:[18]
7C8111E0	8B48 30	MOV ECX,DWORD PTR DS:[EAX+30]
7C8111E3	8B81 B0000000	MOV EAX,DWORD PTR DS:[ECX+B0]
7C8111E9	0FB791 AC000000	MOVZX EDX,WORD PTR DS:[ECX+AC]
7C8111F0	83F0 FE	XOR EAX,FFFFFFFF
7C8111F3	C1E0 0E	SHL EAX,0E
7C8111F6	0BC2	OR EAX,EDX
7C8111F8	C1E0 08	SHL EAX,8
7C8111FB	0B81 A8000000	OR EAX,DWORD PTR DS:[ECX+A8]
7C811201	C1E0 08	SHL EAX,8
7C811204	0B81 A4000000	OR EAX,DWORD PTR DS:[ECX+A4]
7C81120A	C3	RET
7C81120B	90	NOP
7C81120C	90	NOP
7C81120D	90	NOP
7C81120E	90	NOP
7C811210	8BFF	MOV EDI,EDI
7C811212	55	PUSH EBP
7C811213	8BEC	MOV EBP,ESP
7C811215	83EC 24	SUB ESP,24
7C811218	57	PUSH EDI
7C811219	6A 07	PUSH 7
7C81121B	33C0	XOR EAX,EAX
7C81121D	2145 FC	AND DWORD PTR SS:[EBP-4],EAX
7C811220	59	POP ECX
7C811221	8D7D E0	LEA EDI,DWORD PTR SS:[EBP-20]
7C811224	C745 DC 20000000	MOV DWORD PTR SS:[EBP-24],20

همانطور که می بینید اولین دستور در تابع GetVersion برابر Mov EAX,Dword ptr FS:[18] است. خوب، حالا برنامه را با کلید F9 اجرا کنید و بعد نگاهی به تابع GetVersion بیندازید:

F84	JMP 00A10000
	NOP
000000	MOV ECX,DWORD PTR DS:[EAX+30]
C0000000	MOV EAX,DWORD PTR DS:[ECX+B0]
	MOVZX EDX,WORD PTR DS:[ECX+AC]
	XOR EAX,FFFFFFFF
	SHL EAX,0E
	OR EAX,EDX
	SHL EAX,8
000000	OR EAX,DWORD PTR DS:[ECX+A8]
	SHL EAX,8
000000	OR EAX,DWORD PTR DS:[ECX+A4]
	RET
	NOP
	NOP
	NOP
	NOP

می بینید؟ دستوری که قبلا Mov EAX,Dword ptr FS:[18] بوده، الان شده JMP 00A10000... پس کدها توانسته اند تابع GetVersion را Hook کنند. حالا ببینم این پرش به کجا می رود؟... بر روی این پرش کلید Enter را بزنید تا ببینیم این پرش به کجا می رود:

Address	Hex dump	Disassembly
00A10000	64:A1 18000000	MOV EAX,DWORD PTR FS:[18]
00A10006	C605 40104000 50	MOV BYTE PTR DS:[401040],50
00A1000D	C605 8A104000 00	MOV BYTE PTR DS:[40108A],0
00A10014	- E9 C711E07B	JMP kernal32.7C8111E0
00A10019	0000	ADD BYTE PTR DS:[EAX],AL
00A1001B	0000	ADD BYTE PTR DS:[EAX],AL
00A1001D	0000	ADD BYTE PTR DS:[EAX],AL
00A1001F	0000	ADD BYTE PTR DS:[EAX],AL
00A10021	0000	ADD BYTE PTR DS:[EAX],AL
00A10023	0000	ADD BYTE PTR DS:[EAX],AL
00A10025	0000	ADD BYTE PTR DS:[EAX],AL
00A10027	0000	ADD BYTE PTR DS:[EAX],AL

می بینید؟ دستور Mov EAX,Dword ptr FS:[18] که در تابع GetVersion قرار داشت را به اینجا منتقل کردم و بعد با دو دستور زیر، دو بایت از برنامه را دستکاری کردم تا به این ترتیب برنامه کرک شود:

Mov Byte ptr ds: [401040], 50
Mov byte ptr ds: [40108a], 0

دستور اول Nag-Screen را از بین می برد و دستور دوم گزینه Secret را فعال می کند. و بعد از این دو دستور، دستور JMP 7C8111E0 را نوشتم تا دوباره به تابع GetVersion برگردیم و ادامه دستورات این تابع اجرا شود و مشکلی برای برنامه پیش نیاید © خوب... حالا با روشی که من برای کرک کردن این برنامه استفاده کردم، آشنا شدید. بهتر است برنامه را ری استارت کنید تا دوباره کدها را بررسی کنیم:

Address	Hex dump	Disassembly
0040653E <ModuleLea	60	PUSHAD
0040653F	68 09654000	PUSH 00406509
00406544	FF15 CCF04000	CALL DWORD PTR DS:[&KERNEL32.GetModuleHandleA]
0040654A	50	PUSH EAX
0040654B	50	PUSH EAX
0040654C	68 16654000	PUSH 00406516
00406551	50	PUSH EAX
00406552	FF15 C8F04000	CALL DWORD PTR DS:[&KERNEL32.GetProcAddress]
00406558	A3 A06E4000	MOV DWORD PTR DS:[406EA0],EAX
0040655D	58	POP EAX
0040655E	68 25654000	PUSH 00406525
00406563	50	PUSH EAX
00406564	FF15 C8F04000	CALL DWORD PTR DS:[&KERNEL32.GetProcAddress]
0040656A	A3 A46E4000	MOV DWORD PTR DS:[406EA4],EAX
0040656F	58	POP EAX
00406570	68 32654000	PUSH 00406532
00406575	50	PUSH EAX
00406576	FF15 C8F04000	CALL DWORD PTR DS:[&KERNEL32.GetProcAddress]
0040657C	A3 A86E4000	MOV DWORD PTR DS:[406EA8],EAX
00406581	6A 40	PUSH 40
00406583	68 00300000	PUSH 3000
00406588	68 00100000	PUSH 1000
0040658D	6A 00	PUSH 0
0040658F	FF15 A46E4000	CALL DWORD PTR DS:[406EA4]
00406595	A3 AC6E4000	MOV DWORD PTR DS:[406EAC],EAX
0040659A	BE B06E4000	MOV ESI,00406EB0
0040659F	8BF8	MOV EDI,EAX
004065A1	B9 13000000	MOV ECX,13
004065A6	F3:A4	REP MOVSB BYTE PTR ES:[EDI],BYTE PTR DS:[ESI]
004065A8	A1 A86E4000	MOV EAX,DWORD PTR DS:[406EA8]
004065AD	68 FA6F4000	PUSH 00406FFA
004065B2	6A 40	PUSH 40
004065B4	60 06	PUSH 6

اولین دستوری که اجرا کردم، تابع GetModuleHandleA بود که برای به دست آوردن فایل Kernel32.dll با آن با تابع GetProcAddress آدرس توابعی که لازم دارم را به دست آوردم. (چون فایل پک شده این توابع را نداشت. من باید خودم آدرس این توابع را بگیرم.)

بعد از گرفتن آدرس توابع، در خط 40658F تابع VirtualAlloc را اجرا کردم... چرا که باید در Memory یک سکشن درست کنم تا کدها را آنجا قرار بدهم. و بعد از آن با دستوری که در خط 4065A6 قرار دارد، (REP MOVSB BYTE PTR ES:[EDI],BYTE PTR DS:[ESI]) کدهای خودم (همان دو خط کدی که برنامه را کرک می کرد) را از این سکشن به سکشنی که در Memory ساختم منتقل کردم. و بعد در خط 4065B7 تابع VirtualProtect را اجرا کردم. چرا که به طور پیشفرض برنامه ها حق دستکاری فایل Kernel32.dll را ندارند. من با این تابع کاری کردم که بتوانم تابع GetVersion را دستکاری کرد.

برای اطلاعات بیشتر راجع به این توابع (VirtualAlloc و VirtualProtect و GetProcAddress) می توانید به سایت MSDN.microsoft.com یا فایل راهنمای توابع API در OllyDBG مراجعه کنید.

بعد از اجرای تابع VirtualProtect همه چیز آماده هست تا تابع GetVersion را دستکاری کنم. پس با دستوراتی که از خط 4065BD تا خط 4065CE قرار دارد، دستور اول تابع GetVersion را کپی می کنم در سکشنی که ساختم.

و بعد با دستوراتی که از خط 4065D0 تا خط 4065E7 قرار دارد، اولین دستور تابع GetVersion را پاک می کنم و جای آن پرشی می گذارم که به سکشنی که در Memory ساختم می رود.

بعد هم با دستوراتی که از خط 4065EB تا خط 406605 قرار دارد، یک پرش بعد از کرک شدن برنامه (با آن دو دستور) می سازم تا بعد از کرک شدن، برنامه دوباره به تابع GetVersion مراجعه کند و مشکلی در اجرای برنامه پیش نیاید.

البته من این کدها را می توانستم خیلی کوتاهتر بنویسم و بایت های کمتری را مصرف کنم ولی خوب... خیلی سریع نوشتم، و حوصله دوباره نویسی هم ندارم ☹

پند نکته:

۱. روش Hook کردن بیشتر در پروتکتورهای صورت می گیرد که آنپک کردن آن سخت است. (مثل TheMida(WinLicense)، ExeCryptor، Armadillo، Asprotect و...) و گرنه وقتی می شود فایل را به راحتی آنپک کرد، چه دلیلی وجود دارد که این همه کد بنویسیم؟

۲. این روش همیشه هم به همین راحتی نیست. ☹ بزرگترین مشکلی که در Inline patch کردن پروتکتورها وجود دارد این است که اکثراً پروتکتورها از integrity check (همانند CRC32 Check) استفاده می کنند، به این ترتیب اگر شما کوچکترین کدی به فایل پک شده تزریق کنید، پروتکتور متوجه می شود و اجازه اجرا شدن برنامه را نمی دهد ☹ ... در این موارد معمولاً تابع API که در نزدیکی Integrity Check قرار دارد را Hook می کنند، یا بررسی می کنند ببینند چه تابعی CRC را چک می کند؟ و آن تابع را Hook می کنند ☹

۳. حتماً لازم نیست که با استفاده از تابع VirtualAlloc یک سکشن در Memory بسازیم. می شود کدها را در داخل خود فایل ریخت. ولی ممکن است این کار باعث شود برنامه در هر سیستمی اجرا نشود. (در ضمن این روش در برخی از توابع API جواب نمی دهد.)

۴. اگر فضای خالی برای تزریق کد پیدا نکردید، می توانید از نرم افزار ToPo استفاده کنید و سکشنی جدید به فایل اضافه کنید.

۵. اگر پکر مورد نظر از Self-modifying کد استفاده نمی کند. همان روش Inline patch معمولی بهترین راه است ☹

PeBundle

توضیحی در مورد پکر: PeBundle که توسط Jeremy Collake نوشته شده، برنامه ای است که کار اصلی آن این است که می تواند فایل های DLL مورد نیاز یک برنامه را در یک فایل ذخیره کند... یعنی فرض کنید شما برنامه ای نوشته اید که ۶ فایل DLL و یک فایل exe دارد، این برنامه می تواند تمامی این هفت فایل را در درون فایل exe ذخیره کند، و به اصطلاح Bundle کند. شما در برنامه PeBundle می توانید انتخاب کنید که آیا فایل های DLL بعد از اجرا شدن فایل Exe در دیسک ذخیره شود یا در Memory ؟ اگر برنامه نویس گزینه دیسک را انتخاب کرده باشد، فایل DLL در یکی از این چند جا ذخیره می شود.

System Directory (C:\Windows\system32)

Runtime Folder

Executable's Folder

Windows folder

Temp

Custom path (مسیر انتخابی)

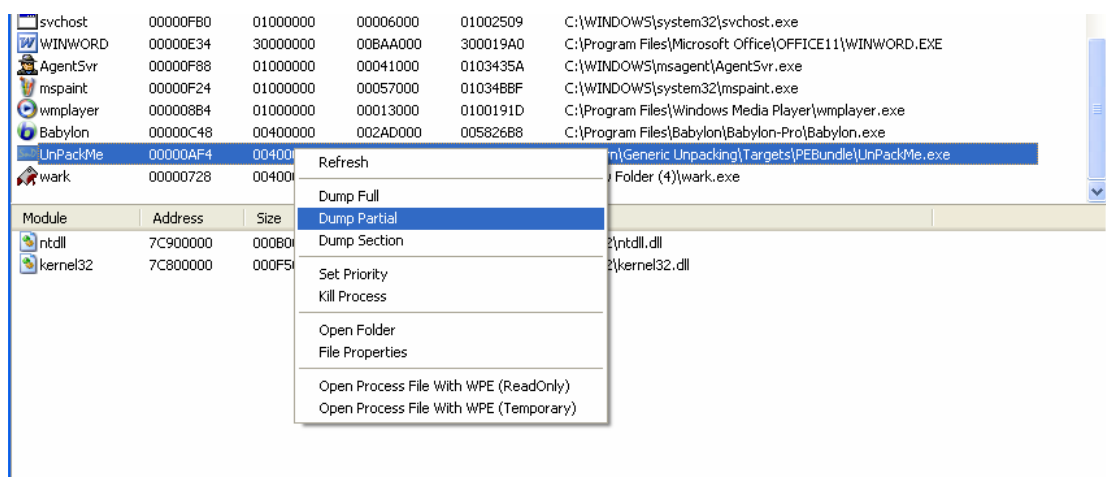
پس همیشه این جاها را بگردید شاید فایل DLL را پیدا کردید. می توانید از نرم افزارهایی مثل FileMon برای اینکه بفهمید فایل DLL کجا ریخته شده استفاده کنید.

در این مثال، برنامه نویس گزینه Memory را تیک زده، یعنی فایل فقط در حافظه قرار می گیرد. فایل مورد نظر را در OllyDBG باز کنید و کلید ALT + M را بزنید تا وارد Memory Map شوید:

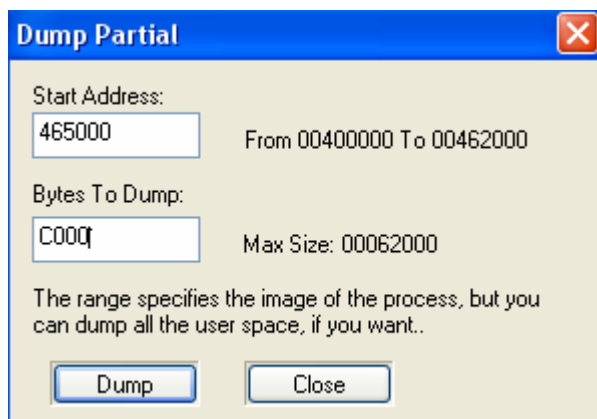
00320000	00006000	00320000	(itself)		
00330000	00041000	00330000	(itself)		
00380000	00001000	00380000	(itself)		
00390000	00001000	00390000	(itself)		
003A0000	00001000	003A0000	(itself)		
003B0000	00001000	003B0000	(itself)		
00400000	00001000	UnPackMe	00400000	(itself)	
00401000	0004A000	UnPackMe	00400000		PE header
0044B000	0000C000	UnPackMe	00400000		code
00457000	00009000	UnPackMe	00400000		data
00460000	00003000	UnPackMe	00400000		
00463000	00002000	UnPackMe	00400000		
00465000	00001000	UnPackMe	00400000		resources
00466000	00004000	UnPackMe	00400000		
0046A000	00001000	UnPackMe	00400000		.pe
0046B000	00004000	UnPackMe	00400000		.text
0046F000	00001000	UnPackMe	00400000		.rdata
00470000	00001000	UnPackMe	00400000		.data
00471000	00002000	UnPackMe	00400000		.rsrc
00473000	00001000	UnPackMe	00400000		.reloc
00474000	00005000	UnPackMe	00400000		pebundle
00479000	00002000	UnPackMe	00400000		
0047B000	00002000	UnPackMe	00400000		.pe
0047D000	00002000	UnPackMe	00400000		.text
0047F000	00001000	UnPackMe	00400000		.rdata
00480000	00002000	UnPackMe	00400000		.data
00483000	00004000	UnPackMe	00400000		.rsrc
00487000	00001000	UnPackMe	00400000		.reloc
00488000	00004000	UnPackMe	00400000		pebundle
0048C000	00002000	UnPackMe	00400000		
0048E000	00001000	UnPackMe	00400000		.pe
0048F000	00002000	UnPackMe	00400000		.text
00491000	00001000	UnPackMe	00400000		.rdata
00492000	00005000	UnPackMe	00400000		.data
00497000	00002000	UnPackMe	00400000		.rsrc
00499000	00002000	UnPackMe	00400000		.reloc
0049B000	00002000	UnPackMe	00400000		pebundle
0049D000	00001000	UnPackMe	00400000		
0049E000	00002000	UnPackMe	00400000		.pe
004A1000	00005000	UnPackMe	00400000		.text
004A6000	00002000	UnPackMe	00400000		.rdata
004A8000	00002000	UnPackMe	00400000		.data
004AA000	00001000	UnPackMe	00400000		.rsrc
004AB000	00001000	UnPackMe	00400000		.reloc
004AC000	00002000	UnPackMe	00400000		pebundle
004AE000	00001000	UnPackMe	00400000		
004AF000	00004000	UnPackMe	00400000		.pe
004B3000	00001000	UnPackMe	00400000		.text
004B4000	00004000	UnPackMe	00400000		.rdata
004B8000	00001000	UnPackMe	00400000		.data
004B9000	00001000	UnPackMe	00400000		.rsrc
004BA000	00002000	UnPackMe	00400000		.reloc
004BF000	000021000	kernel32	7C800000	(itself)	pebundle
7C800000	00001000	kernel32	7C800000		SFX, imports
7C801000	000083000	kernel32	7C800000		
7C804000	00005000	kernel32	7C800000		PE header
7C809000	00006000	kernel32	7C800000		code, imports, exports
					data
					resources

فایل های DLL در سکشن هایی جدید به فایل exe اضافه می شوند. در تصویر بالا، همانطور که می بینید ۶ فایل DLL در درون فایل قرار دارند. (سکشن های هر فایل DLL در درون مستطیل قرار گرفتند). من اولین فایل DLL را از داخل فایل exe بیرون می آورم. ۵. فایل دیگر با شما ☺

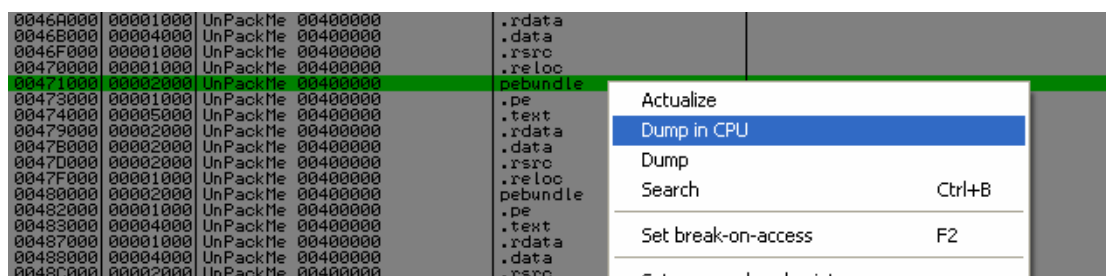
در تصویر بالا اولین مستطیل قرمز (اولین فایل DLL) از خط 465000 شروع می شود. (محل شروع سکشن .pe) و در خط 471000 (محل شروع سکشن Pebundle) تمام می شود. چون محتویات فایل DLL دستکاری نشده است، من می توانم این قسمت از فایل را دامپ بگیرم و فایل را بیرون بیاورم. برنامه WARK را باز کنید، همانند تصویر بر روی پروسه مورد نظر کلیک راست کرده و گزینه Dump partial را بزنید:



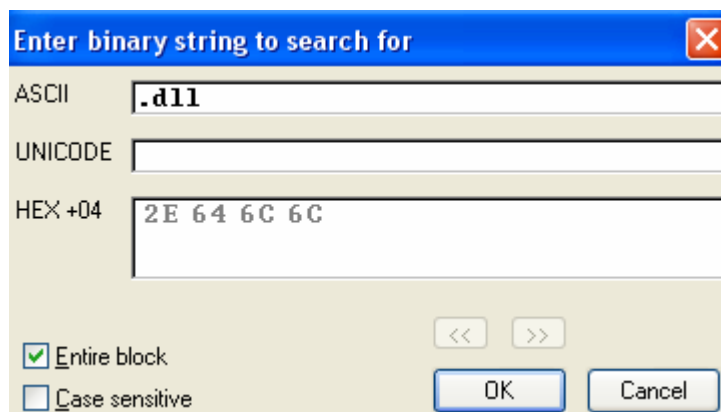
حالا محل شروع فایل DLL را در Start Address می نویسیم: 465000 و در قسمت Bytes to dump سایز فایل DLL را می نویسیم (با ماشین حساب ویندوز، انتهای فایل را منهای ابتدای آن کنید).



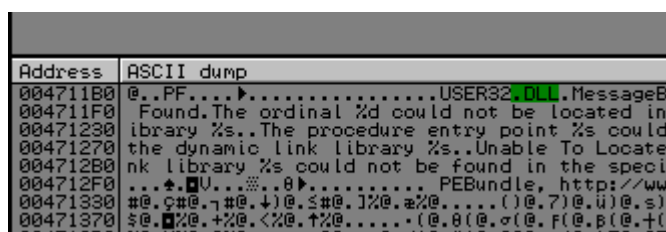
حالا گزینه دامپ را بزیند و فایل را با یک نام دلخواه ذخیره کنید. (مثلا Gooran.dll)
بعد از دامپ کردن فایل با RoleX یا پلاگین PeTools، Relocation های فایل DLL را دوباره بسازید.
حالا باید بررسی کنیم نام واقعی این فایل چیست؟... برای اینکار در Memory Map همانند تصویر بر روی سکشن PeBundle مربوط به این فایل کلیک راست کرده و گزینه Dump in CPU را بزیند:



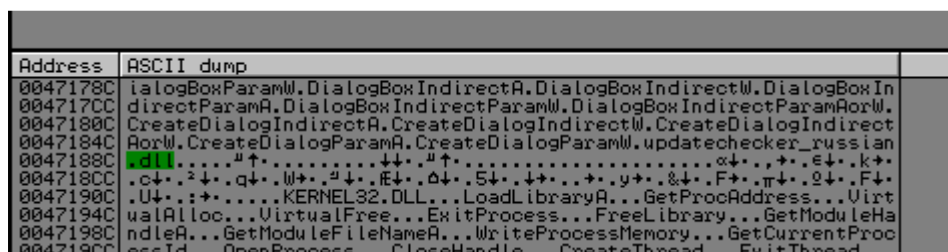
حالا در پنجره دامپ کلیدهای CTRL + B را بزنید:



همانند تصویر تایپ کنید .dll. (چون نام اصلی فایل در داخل این سکشن نوشته می شود، ما به دنبال مقدار .dll می گردیم تا نام فایل را پیدا کنیم). کلید OK را بزنید:



اینجا نوشته شده USER32.dll پس اینجا جایی نیست که ما می خواهیم. کلیدهای CTRL + L را بزنید تا دوباره جستجو انجام شود:



اینجا نوشته شده : updatechecker_russian.dll پس نام اصلی فایل ما همین است ☺

خوب، حالا ۵ فایل DLL دیگر را هم به این ترتیب از داخل فایل بیرون بیاورید:



حالا نوبت فایل exe است. برای آنپک کردن فایل exe باید OEP را پیدا کنیم. برای یافتن OEP هم می توانیم از رجیستر ESP استفاده کنیم ☺

دو بار کلید F8 را بزنید تا مقدار رجیستر ESP تغییر کند، بعد بر روی مقدار رجیستر ESP یک Hardware breakpoint on access می گذاریم و بعد برنامه با F9 اجرا می کنیم:



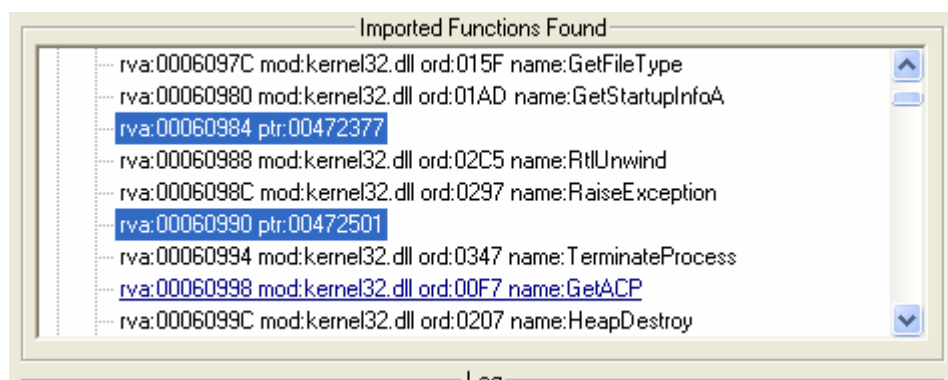
سه بار کلید F8 را بزنید:

Address	Hex dump	Disassembly
004AC000	> 9C	PUSHAD
004AC001	. 60	PUSHAD
004AC002	. E8 02000000	CALL 004AC009
004AC007	. 33C0	XOR EAX,EAX
004AC009	. 8BC4	MOV EAX,ESP
004AC00B	. 83C0 04	ADD EAX,4
004AC00E	. 93	XCHG EAX,EBX
004AC00F	. 8BE3	MOV ESP,EBX
004AC011	. 8B5B FC	MOV EBX,DWORD PTR DS:[EBX-4]
004AC014	. 81EB 07204000	SUB EBX,00402007
004AC01A	. 87DD	XCHG EBP,EBX
004AC01C	. 01AD BB2F4000	ADD DWORD PTR SS:[EBP+402FBB],EBP
004AC022	. 01AD E5304000	ADD DWORD PTR SS:[EBP+4030E5],EBP
004AC028	. 01AD 5E304000	ADD DWORD PTR SS:[EBP+40305E],EBP
004AC02E	. 01AD 92314000	ADD DWORD PTR SS:[EBP+403192],EBP
004AC034	. 01AD 42314000	ADD DWORD PTR SS:[EBP+403142],EBP

دوباره همین کار را بکنید. یعنی بعد از اجرا شدن دستور PushAD بر روی مقدار رجیستر ESP، Hardware Breakpoint بگذارید. آنقدر این عمل را انجام دهید تا به OEP برسیم:

004271AD	90	NOP
004271AE	90	NOP
004271AF	90	NOP
004271B0	> 55	PUSH EBP
004271B1	. 8BEC	MOV EBP,ESP
004271B3	. 6A FF	PUSH -1
004271B5	. 68 600E4500	PUSH 00450E60
004271BA	. 68 C8924200	PUSH 004292C8
004271BF	. 64:A1 00000000	MOV EAX,DWORD PTR FS:[0]
004271C5	. 50	PUSH EAX
004271C6	. 64:8925 00000000	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 SS:[EBP-18],ESP
004271D6	. FF15 DC0A4600	CALL DWORD PTR DS:[460ADC]
004271DC	. 33D2	XOR EDX,EDX
004271DE	. 8AD4	MOV DL,AH
004271E0	. 8915 34F64500	MOV DWORD PTR DS:[45F634],EDX

حالا از فایل دامپ بگیریید و ImportREC را باز کنید:



همانطور که می بینید V تا از توابع ما Invalid هستند. (Redirect شده اند) من یکی از آنها را پیدا می کنم، بقیه با شما ☺ همانطور که در تصویر بالا می بینید در خط 60984 (بر حسب RVA) مقدار 472377 قرار دارد که یک تابع API نیست. در OllyDBG کلیدهای CTRL + G را بزنید و تایپ کنید 472377 و کلید OK را بزنید تا به این خط بروید. حال کلیک راست کرده، و گزینه New origin here را بزنید تا دستورات این قسمت را اجرا کنیم و ببینیم این دستورات در واقع به کدام تابع API می رود.

00472374	00	DB 00
00472375	00	DB 00
00472376	00	DB 00
00472377	> C8 040000	ENTER 4,0
00472378	. 53	PUSH EBX
0047237C	. E8 00000000	CALL 00472381
00472381	. 5B	POP EBX
00472382	. 81EB 81334000	SUB EBX,00403381
00472388	. 80BB 6F324000 00	CMP BYTE PTR DS:[EBX+40326F],0
0047238F	. 74 77	JE SHORT 00472408
00472391	. 83BB 6F334000 00	CMP DWORD PTR DS:[EBX+40336F],0
00472398	. 75 65	JNZ SHORT 004723FF
0047239A	. FF93 49254000	CALL DWORD PTR DS:[EBX+402549]
004723A0	. 8945 FC	MOV [LOCAL.1],EAX
004723A3	. 6A 04	PUSH 4
004723A5	. 68 00100000	PUSH 1000
004723AA	. 68 00010000	PUSH 100
004723AF	. 6A 00	PUSH 0
004723B1	. FF93 09294000	CALL DWORD PTR DS:[EBX+402909]
004723B7	. 8983 6F334000	MOV DWORD PTR DS:[EBX+40336F],EAX
004723BD	. 85C0	TEST EAX,EAX
004723BF	. 74 47	JE SHORT 00472408
004723C1	. 57	PUSH EDI
004723C2	. 56	PUSH ESI

برنامه را با F8 ادامه دهید، پرسی که در 47238F قرار دارد در کامپیوتر من انجام شده است، بنابراین به این خط رسیدم:

004723FD	. 5E	POP ESI
004723FE	. 5F	POP EDI
004723FF	> 8B83 6F334000	MOV EAX,DWORD PTR DS:[EBX+40336F]
00472405	. 5B	POP EBX
00472406	. C9	LEAVE
00472407	. C3	RET
00472408	> FF93 49254000	CALL DWORD PTR DS:[EBX+402549]
0047240E	. 5B	POP EBX
0047240F	. C9	LEAVE
00472410	. C3	RET
00472411	. C8 040000	ENTER 4,0
00472415	. 53	PUSH EBX
00472416	. E8 00000000	CALL 0047241B
0047241B	. 5B	POP EBX
0047241C	. 81EB 1B344000	SUB EBX,0040341B
00472422	. 80BB 6F324000 00	CMP BYTE PTR DS:[EBX+40326F],0

حالا کلید F7 را بزنید تا به داخل تابع بروید. باز هم به جایی مثل خط 472377 رسیدیم:

Address	Hex dump	Disassembly
00481377	> C8 040000	ENTER 4,0
0048137B	. 53	PUSH EBX
0048137C	. E8 00000000	CALL 00481381
00481381	. 5B	POP EBX
00481382	. 81EB 81334000	SUB EBX,00403381
00481388	. 80BB 6F324000 00	CMP BYTE PTR DS:[EBX+40326F],0
0048138F	. 74 77	JE SHORT 00481408
00481391	. 83BB 6F334000 00	CMP DWORD PTR DS:[EBX+40336F],0
00481398	. 75 65	JNZ SHORT 004813FF
0048139A	. FF93 49254000	CALL DWORD PTR DS:[EBX+402549]
004813A0	. 8945 FC	MOV [LOCAL.1],EAX
004813A3	. 6A 04	PUSH 4
004813A5	. 68 00100000	PUSH 1000
004813AA	. 68 00010000	PUSH 100
004813AF	. 6A 00	PUSH 0
004813B1	. FF93 09294000	CALL DWORD PTR DS:[EBX+402909]
004813B7	. 8983 6F334000	MOV DWORD PTR DS:[EBX+40336F],EAX
004813BD	. 85C0	TEST EAX,EAX
004813BF	. 74 47	JE SHORT 00481408
004813C1	. 57	PUSH EDI
004813C2	. 56	PUSH ESI

حالا دوباره همین عمل را انجام می دهیم. یعنی چند بار F8 می زنیم تا به دستوری شبیه [Call Dword ptr DS:[EBX + xxxxx]] برسیم:

004813FF	> 8B83 6F334000	MOV EAX,DWORD PTR DS:[EBX+40336F]
00481405	. 5B	POP EBX
00481406	. C9	LEAVE
00481407	. C3	RET
00481408	> FF93 49254000	CALL DWORD PTR DS:[EBX+402549]
0048140E	. 5B	POP EBX
0048140F	. C9	LEAVE
00481410	. C3	RET
00481411	. C8 040000	ENTER 4,0
00481415	. 53	PUSH EBX
00481416	. E8 00000000	CALL 0048141B
0048141B	. 5B	POP EBX
0048141C	. 81EB 1B344000	SUB EBX,0040341B
00481422	. 80BB 6F324000 00	CMP BYTE PTR DS:[EBX+40326F],0

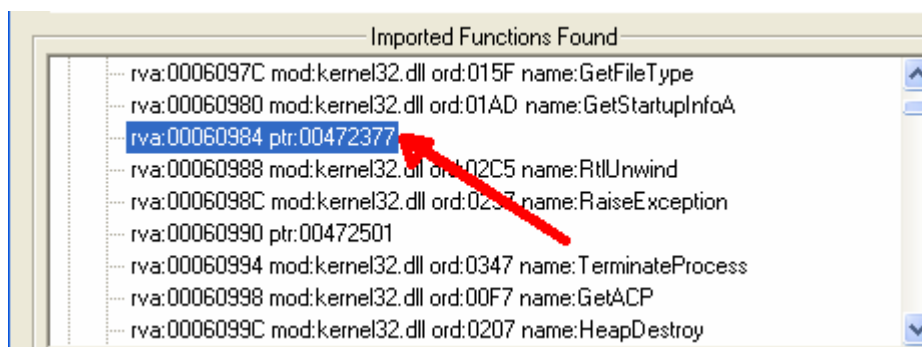
این بار هم F7 می زنیم و به درون این تابع می رویم. باز هم به جایی شبیه خط 472377 رسیدیم:

Address	Hex dump	Disassembly
00490377	> C8 040000	ENTER 4,0
0049037B	. 53	PUSH EBX
0049037C	. E8 00000000	CALL 00490381
00490381	. 5B	POP EBX
00490382	. 81EB 81334000	SUB EBX,00403381
00490388	. 80BB 6F324000 00	CMP BYTE PTR DS:[EBX+40326F],0
0049038F	. 74 77	JE SHORT 00490408
00490391	. 83BB 6F334000 00	CMP DWORD PTR DS:[EBX+40336F],0
00490398	. 75 65	JNZ SHORT 004903FF
0049039A	. FF93 49254000	CALL DWORD PTR DS:[EBX+402549]
004903A0	. 8945 FC	MOV [LOCAL.1],EAX
004903A3	. 6A 04	PUSH 4
004903A5	. 68 00100000	PUSH 1000

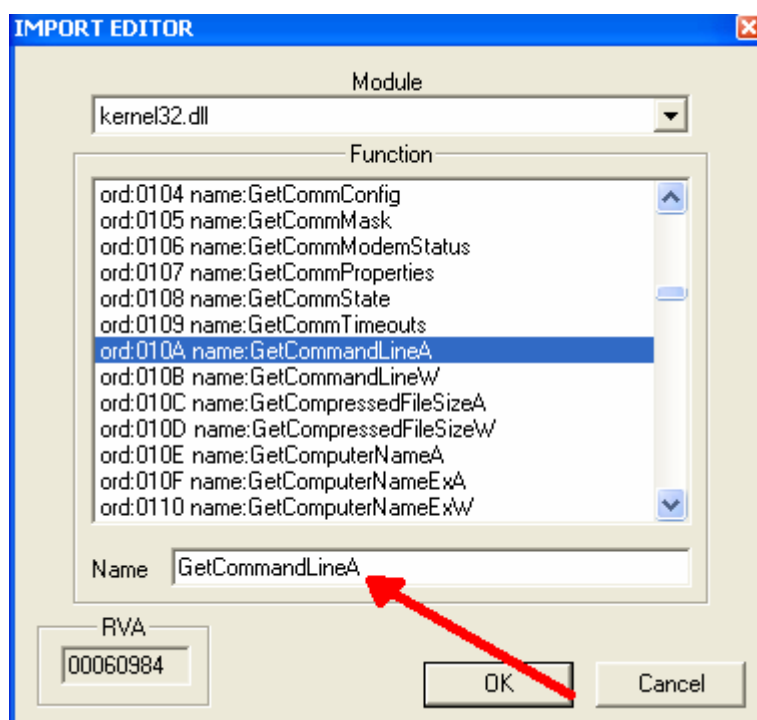
آنقدر این عمل را انجام دهید تا بالاخره به اینجا برسید:

8B83 6F334000	POP EDI	
5B	MOV EAX,DWORD PTR DS:[EBX+40336F]	
C9	POP EBX	
C3	LEAVE	
FF93 49254000	RET	
FF93 49254000	CALL DWORD PTR DS:[EBX+402549]	kernel32.GetCommandLineA
5B	POP EBX	
C9	LEAVE	
C3	RET	
C8 040000	ENTER 4,0	
53	PUSH EBX	
E8 00000000	CALL 0048B41B	
5B	POP EBX	
81EB 1B344000	SUB EBX,0040341B	

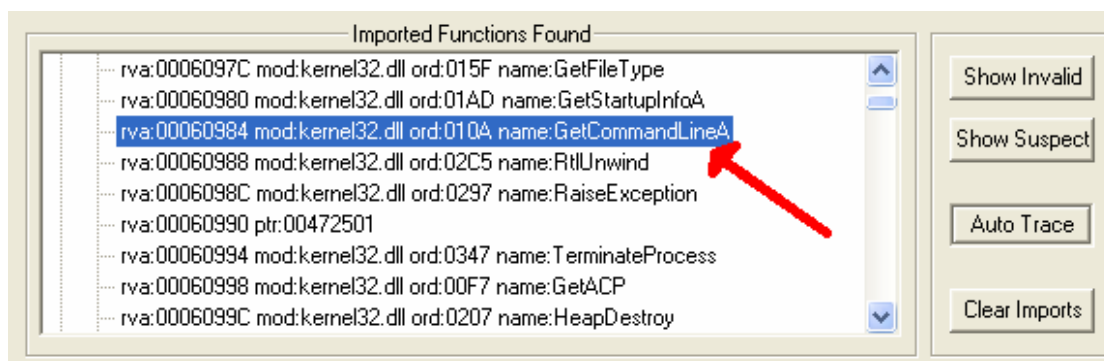
همانطور که می بینید در نهایت خط 472377 به تابع GetCommandLineA می رود. پس دوباره به ImportREC برگردید:



بر روی تابع Invalid مورد نظر دوبار کلیک کرده تا تصویر زیر ظاهر شود:



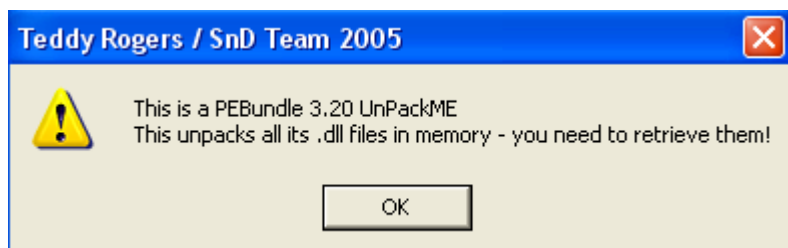
همانند تصویر در قسمت Name تایپ کنید: GetCommandLineA و کلید OK را بزنید:



خوب، یکی از توابع API را درست کردم. ۶ تای دیگر را خودتان درست کنید: (برای اطلاعاتان لیست توابع صحیح را نوشتم، شما با اندکی تلاش می توانید اینها را به دست بیاورید.)

Rva: 00060984=GetCommandLineA
 Rva: 00060990=ExitProcess
 Rva: 00060AD8=GetModuleFileNameA
 Rva: 00060B3C=LoadLibraryA
 Rva: 00060B40=GetProcAddress
 Rva: 00060B44=FreeLibrary
 Rva: 00060B9C=GetModuleHandleA

خوب، بعد از درست کردن این ۷ تابع، فایل دامپ خود را فیکس می کنید، اگر فایل های DLL را هم درست درآورده باشید، برنامه بدون هیچ مشکلی اجرا می شود:

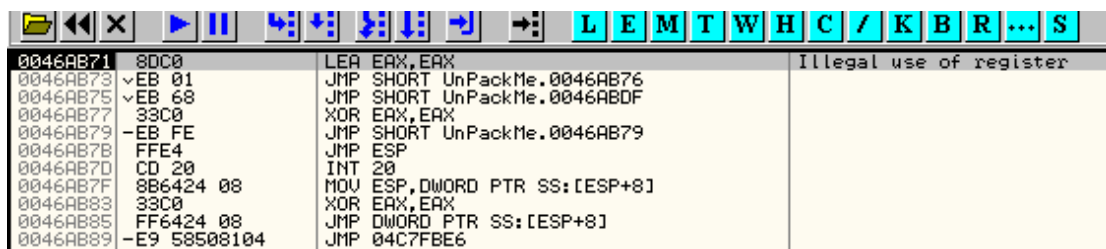


اگر برنامه پیغام زیر را داد، یعنی یکی از فایل های DLL را درست Extract نکرده اید:

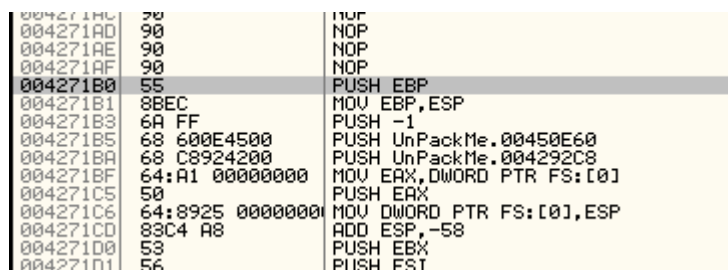


Telock

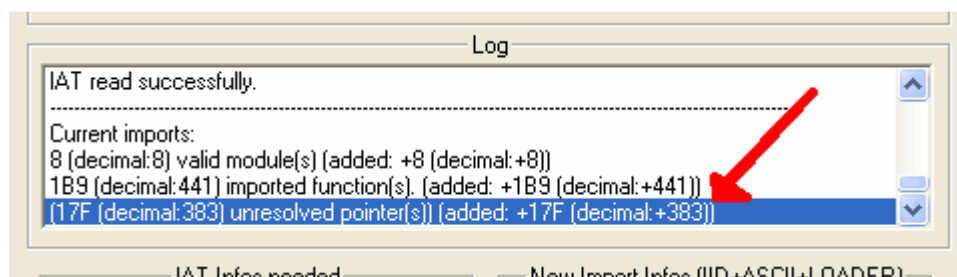
فایل مورد نظر را در OllyDBG باز کنید. (این فایل از چند آنتی دیباگ استفاده می کند. برای همین آن را در SLV modified by OllyDBG باز کنید. در این OllyDBG برنامه بی هیچ مشکلی اجرا می شود.) برای یافتن OEP هم می توانیم از Memory breakpoint استفاده کنیم. ابتدا تیک تمامی خطاها را بردارید. (کلیدهای ALT + O را بزنید و در قسمت Exceptions تمامی تیک ها را بردارید). کلیدهای Shift + F9 را بزنید تا به آخرین خطای اتفاق افتاده در برنامه برسید:



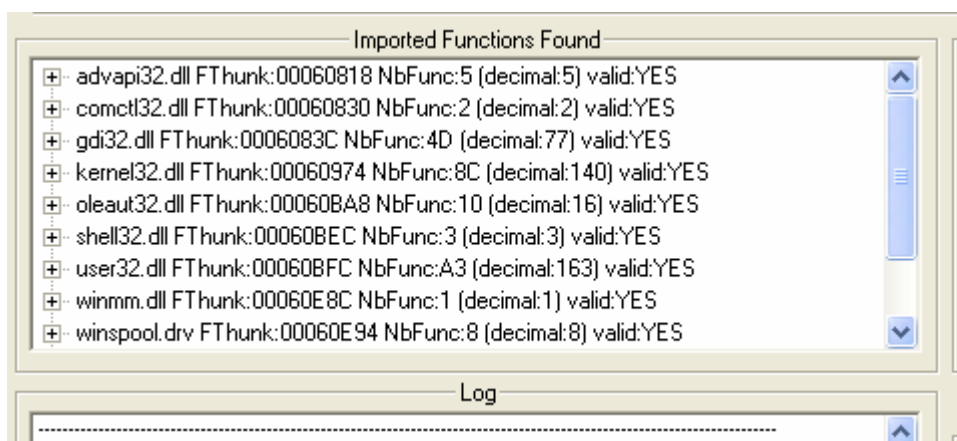
اینجا دیگر کدهای ما Decrypt شده است و می توانیم Memory breakpoint بگذاریم. بر روی سکشن text. یک Memory breakpoint on access بگذارید. و بعد هم کلید F9 را بزنید:



حالا از فایل دامپ بگیری و ImportREC را باز کنید:



همانطور که می بینید ۳۸۳ تابع Invalid داریم. برای فیکس کردن این توابع می توانیم از قابلیت (trap flag) Trace level3 استفاده کنیم. ابتدا برنامه را به طور کامل در OllyDBG باز کنید (کلید F9 را بزنید). و بعد بر روی توابع Invalid کلیک راست کرده و گزینه trap flag را بزنید تا تابع valid شود. (اگر از سیستم ضعیفی استفاده می کنید. بهتر است توابع Invalid را قسمت قسمت فیکس کنید، چون ممکن است ImportREC هنگ کند. در ضمن یادتان باشد که دوباره تیک خطاها را OllyDBG بزنید.)



خوب... حالا که تمامی توابع Valid شد، فایل دامپ شده را فیکس کنید.

ArmProtector

برای یافتن OEP در این پکر می توانیم از Memory breakpoint استفاده کنیم. این پکر آدرس های خودش را در ثبات های Debug می نویسد، به همین دلیل Hardware Breakpoint به خوبی جواب نمی دهد. البته ما نیازی به Hardware breakpoint نداریم، ولی خوب... گفتم که گفته باشم.

ابتدا تیک تمامی خطاها را بردارید، تا به آخرین خطای برنامه برسید. (در ضمن آنالیز OllyDBG را از کدها بردارید):

Address	Hex dump	Disassembly
0046A5F1	8B00	MOV EAX, DWORD PTR DS:[EAX]
0046A5F3	EB 01	JMP SHORT 0046A5F6
0046A5F5	84D4	TEST AH, DL
0046A5F7	011A	ADD DWORD PTR DS:[EDX], EBX
0046A5F9	05 00000000	ADD EAX, 0
0046A5FE	0000	ADD BYTE PTR DS:[EAX], AL
0046A600	0000	ADD BYTE PTR DS:[EAX], AL
0046A602	0000	ADD BYTE PTR DS:[EAX], AL
0046A604	0000	ADD BYTE PTR DS:[EAX], AL
0046A606	0000	ADD BYTE PTR DS:[EAX], AL
0046A608	0000	ADD BYTE PTR DS:[EAX], AL
0046A60A	0000	ADD BYTE PTR DS:[EAX], AL
0046A60C	0000	ADD BYTE PTR DS:[EAX], AL
0046A60E	0000	ADD BYTE PTR DS:[EAX], AL
0046A610	0000	ADD BYTE PTR DS:[EAX], AL
0046A612	8611	XCHG BYTE PTR DS:[ECX], DL
0046A614	C683 00000000 00	MOV BYTE PTR DS:[EBX], 0

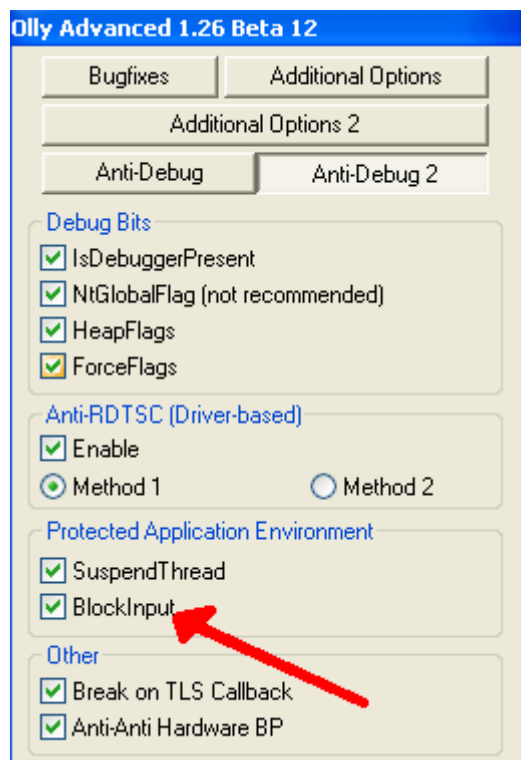
حالا هم همانند مثال های قبل یک Memory Breakpoint بر روی سکشن text. بگذارید و بعد برنامه را با F9 اجرا کنید:

004271AD	90	NOP
004271AE	90	NOP
004271AF	90	NOP
004271B0	55	PUSH ESP
004271B1	8BEC	MOV EBP, ESP
004271B3	6A FF	PUSH -1
004271B5	68 600E4500	PUSH 00450E60
004271B8	68 C8924200	PUSH 004292C8
004271BF	64:A1 00000000	MOV EAX, DWORD PTR FS:[0]
004271C5	50	PUSH EAX

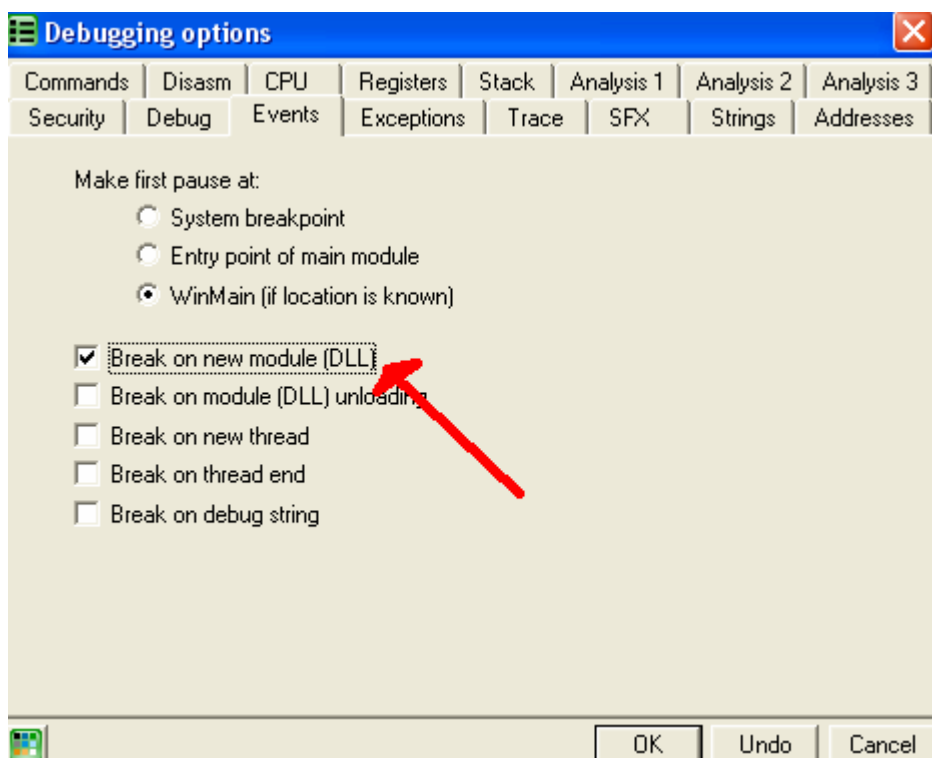
به همین راحتی به OEP رسیدیم. حالا دامپ بگیرید و فیکس کنید ☺

Yoda's Protector

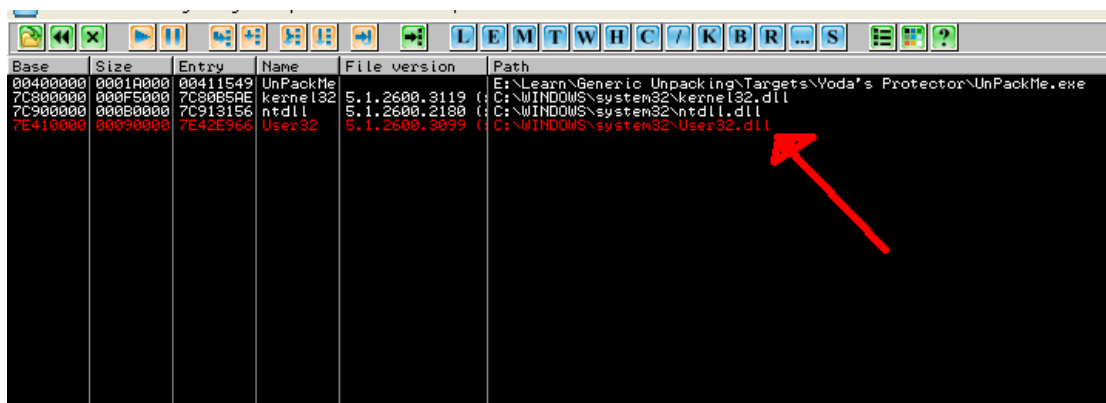
فایل مورد نظر را در OllyDBG باز کنید. چون Yoda's Protector در صورت شناسایی Debugger تابع BlockInput را به چون ما می اندازد. (این تابع موس و کیبرد را از کار می اندازد. هیچکدام از دکمه های کیبرد عمل نمی کنند، مگر ALT + CTRL + Del) به همین دلیل ما باید این تابع را از بین ببریم. در منوی Plugins گزینه OllyAdvanced (این پلاگین را باید داشته باشید) و سپس گزینه Options را بزنید و در قسمت Anti-Debug 2 تیک گزینه BlockInput را بزنید:



کلید OK را بزنید. حالا برای یافتن OEP می توانیم از همین تابع استفاده کنیم. چون این تابع در فایل User32.dll قرار دارد. ما باید اول ببینیم در چه موقعی فایل user32.dll اجرا می شود و بعد بر روی این تابع BP بگذاریم. کلیدهای ALT + O را بزنید و وارد قسمت Events شوید و گزینه Break on new module را تیک بزنید:

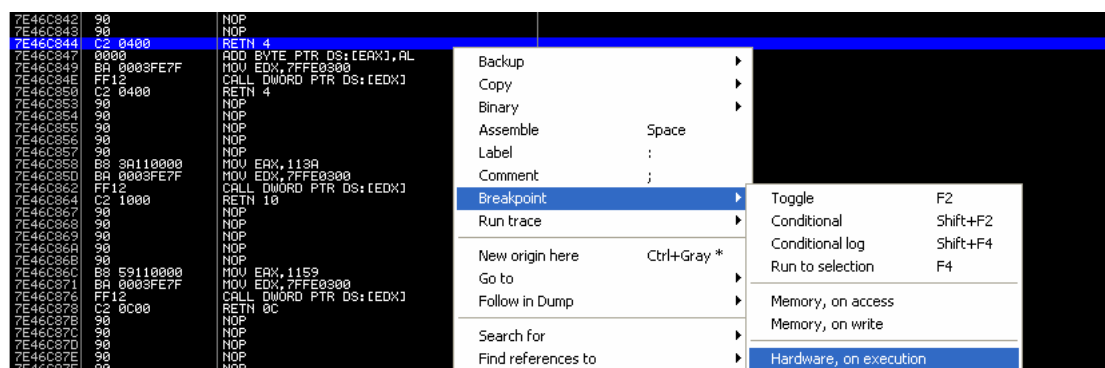


کلید OK را بزنید و دو بار کلید F9 را بزنید تا فایل User32.dll اجرا شود:

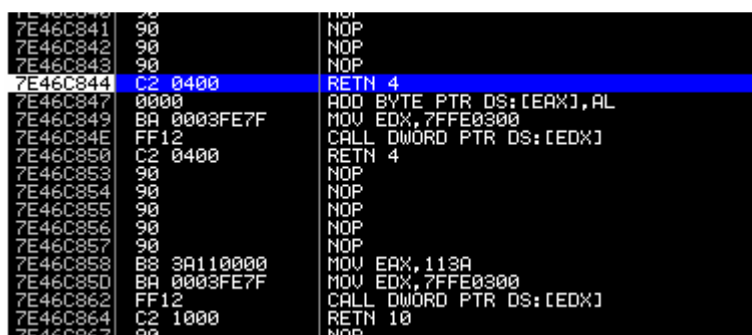


خوب، حالا کلیدهای ALT + C را بزنید تا وارد پنجره اصلی (CPU) شوید و بعد کلیدهای CTRL + G را بزنید و تایپ کنید: BlockInput

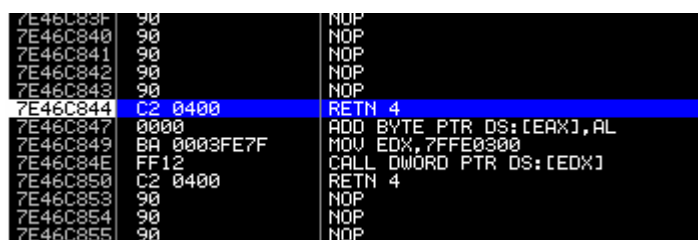
کلید OK را بزنید و بر روی این تابع یک Hardware BP on execution بگذارید:



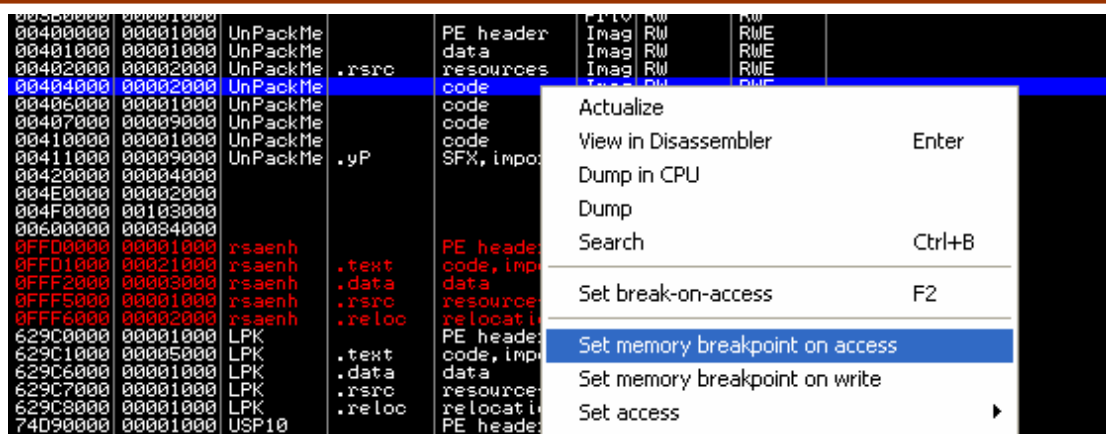
حالا دیگر نیازی به گزینه Break on new module نداریم. (کلیدهای ALT + O را بزنید و در قسمت Events تیک گزینه Break on new modules را بردارید.)



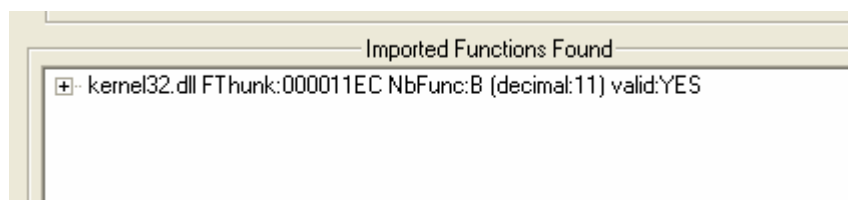
در تابع BlockInput متوقف شدیم. یکبار دیگر F9 بزنید:



خوب، دوباره در تابع BlockInput متوقف شدیم. اینجا جایی است که کدهای ما Decrypt شده و ما می توانیم با Memory BP گذاشتن روی سکشن Code (اولین سکشن محتوی Code زیر PE Header) OEP را پیدا کنیم. کلیدهای ALT + M را بزنید و همانند تصویر Memory BP بگذارید:



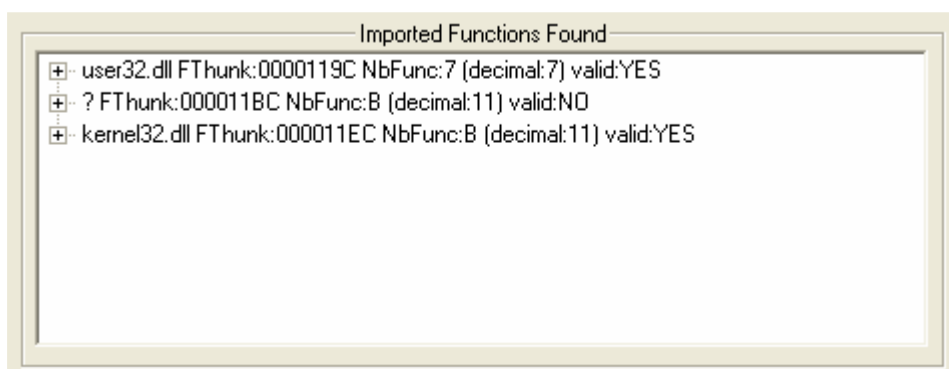
حال کلید F9 را بزنید تا به OEP برسید. (کلیدهای CTRL + A را بزنید تا این سکشن آنالیز شود) بعد از فایل دامپ بگیرید. حالا ImportREC را باز کنید. OEP را به برنامه بدهید و سایر مراحل لازم را انجام دهید:



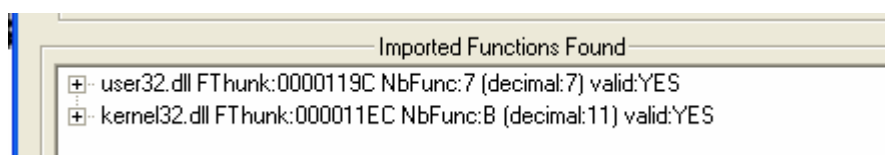
همانطور که می بینید ImportREC نتوانسته IAT را به درستی به دست بیاورد. پس باید خودمان شروع IAT را پیدا کنیم:

Start of IAT = 401198 (RVA: 1198)
End of IAT = 401218 (RVA: 1218)
Size = 80

این اطلاعات را در ImportREC وارد کنید و Get Imports را بزنید:



حالا گزینه show Invalid را بزنید، بر روی توابع Invalid کلیک راست کرده و گزینه Cut Thunk را بزنید:



حالا هم فایل دامپ خود را فیکس کنید:

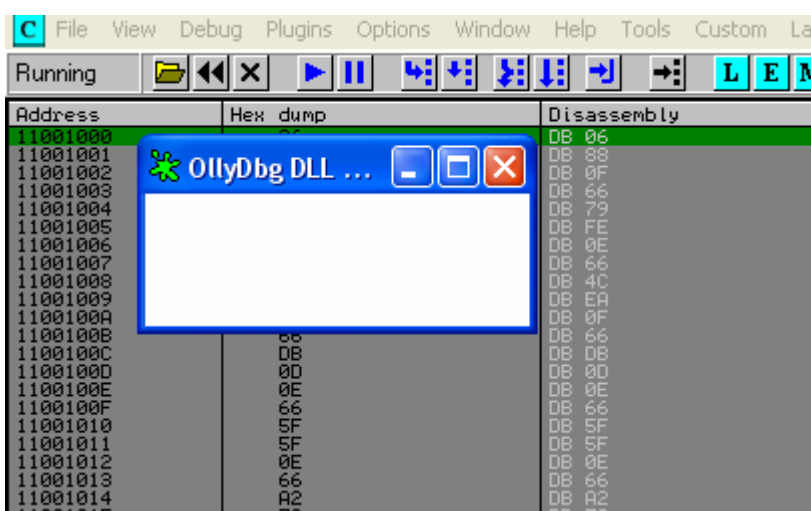


Yoda's Protector OCX

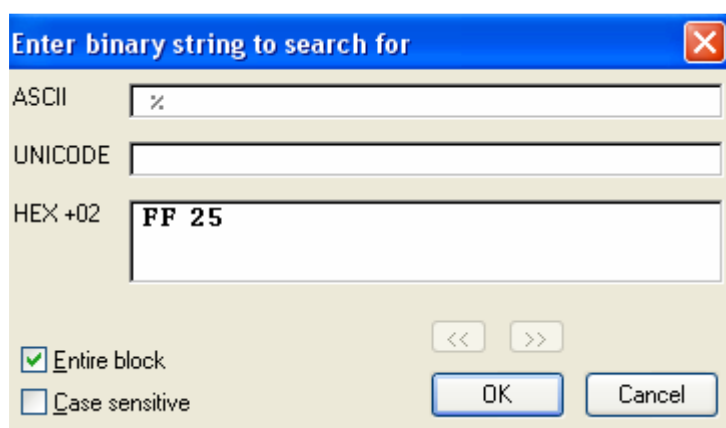
فایل مورد نظر (KewlButtonz.ocx) را در OllyDBG باز کنید:

Address	Hex dump	Disassembly
110188D0 <Module Entry>	60	PUSHAD
110188DE	EB 01	JMP SHORT 110188E1
110188E0	E8	DB E8
110188E1	90	NOP
110188E2	EB 01	JMP SHORT 110188E5
110188E4	E9	DB E9
110188E5	EB 01	JMP SHORT 110188E8
110188E7	E9	DB E9
110188E8	EB 01	JMP SHORT 110188EB
110188EA	E8	DB E8
110188EB	EB 01	JMP SHORT 110188EE
110188ED	C2	DB C2
110188EE	90	NOP
110188EF	EB 01	JMP SHORT 110188F2

برای اجرا شدن فایل OCX چند بار کلید F9 را بزنید تا فایل LoadDll.exe اجرا شود:



ما الان در سکشن اصلی فایل KewlButtonz.ocx هستیم. کلیدهای CTRL + A را بزنید تا این سکشن آنالیز شود. ☺
حالا چه طور OEP را پیدا کنیم؟ برای اینکار چون برنامه در ویژوال بیسیک نوشته شده می توانیم از ساختار این زبان برنامه نویسی استفاده کنیم. کلیدهای CTRL + B را بزنید و تایپ کنید FF25 :



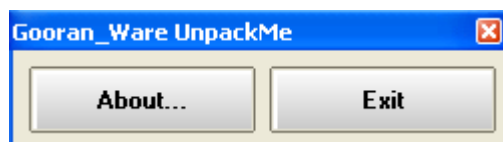
کلید OK را بزنید:

110015EC	00	DB 00
110015ED	00	DB 00
110015EE	00	DB 00
110015EF	00	DB 00
110015F0	FF25 74100011	JMP DWORD PTR DS:[11001074]
110015F6	FF25 B4100011	JMP DWORD PTR DS:[11001084]
110015FC	FF25 C4100011	JMP DWORD PTR DS:[110010C4]
11001602	FF25 50100011	JMP DWORD PTR DS:[11001050]
11001608	FF25 40100011	JMP DWORD PTR DS:[11001040]
1100160E	FF25 E0100011	JMP DWORD PTR DS:[110010E0]
11001614	FF25 18100011	JMP DWORD PTR DS:[11001018]
1100161A	FF25 F4100011	JMP DWORD PTR DS:[110010F4]
11001620	FF25 58100011	JMP DWORD PTR DS:[11001058]
11001626	FF25 F0100011	JMP DWORD PTR DS:[110010F0]
1100162C	FF25 E4100011	JMP DWORD PTR DS:[110010E4]
11001632	FF25 C0100011	JMP DWORD PTR DS:[110010C0]
11001638	FF25 90100011	JMP DWORD PTR DS:[11001090]
1100163E	FF25 BC100011	JMP DWORD PTR DS:[110010BC]
11001644	FF25 28100011	JMP DWORD PTR DS:[11001028]
1100164A	FF25 04100011	JMP DWORD PTR DS:[11001004]
11001650	FF25 24110011	JMP DWORD PTR DS:[11001124]

همانطور که می بینید به JUMP Table رسیدیم. همانطور که می دانید درست زیر Jump Table، OEP قرار دارد. پس:

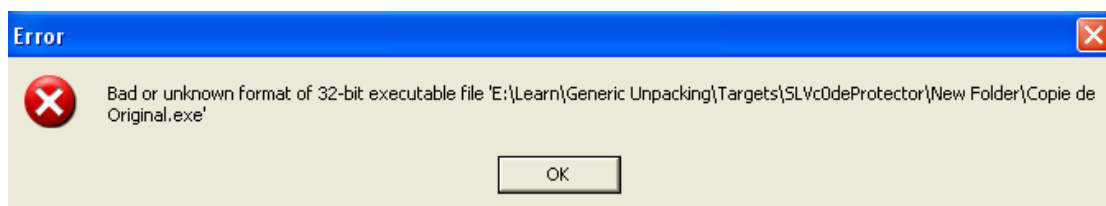
110017C4	FF25 08110011	JMP DWORD PTR DS:[11001108]
110017CA	FF25 04110011	JMP DWORD PTR DS:[11001104]
110017D0	FF25 FC100011	JMP DWORD PTR DS:[110010FC]
110017D6	FF25 0C110011	JMP DWORD PTR DS:[1100110C]
110017DC	58	POP EAX
110017DD	68 102E0111	PUSH 11012E10
110017E2	68 142E0111	PUSH 11012E14
110017E7	52	PUSH EDX
110017E8	E9 E9FFFFFF	JMP 110017D6
110017ED	00	DB 00
110017EE	00	DB 00
110017EF	00	DB 00
110017F0	58	DB 58
110017F1	00	DB 00
110017F2	00	DB 00
110017F3	00	DB 00
110017F4	30	DB 30

خط 17DC همان OEP ماست. حالا بر روی OEP کلیک راست کرده و گزینه new origin here را بزنید و از فایل دامپ بگیرید. حالا ImportREC را باز کنید و فایل دامپ را فیکس کنید (یادتان نرود، باید گزینه Pick DLL را بزنید و بعد فایل خودتان رو انتخاب کنید) بعد هم با RoleX یا PeTools باید Relocation های فایل را درست کنید. حالا فایل Program.exe را اجرا کنید. می بینید که فایل بی هیچ مشکلی اجرا می شود:

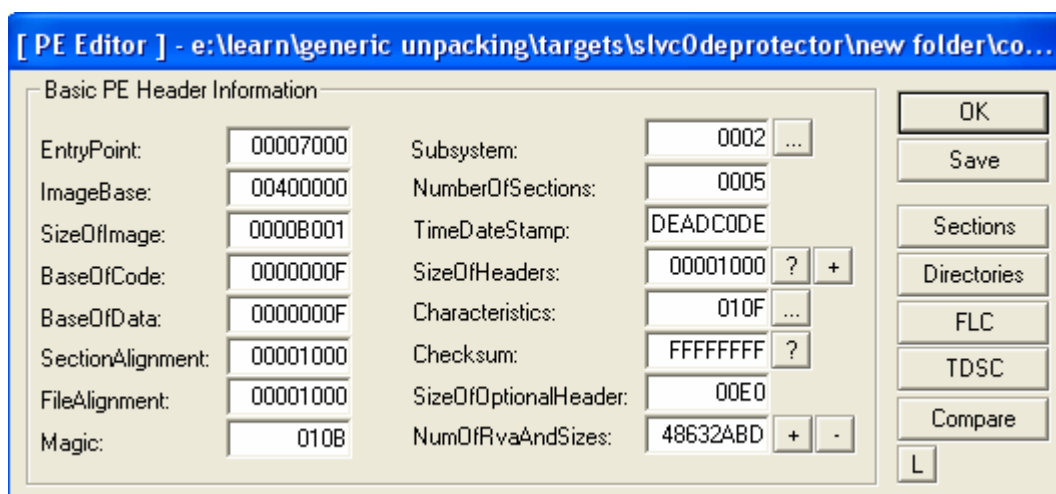


SLVc0deProtector

خوب، فایل مورد نظر را در OllyDBG باز کنید (یک OllyICE که فقط پلاگین های OllyDump و HideDBG، OllyDump رو داشته باشد):



همانطور که می بینید، PE Header فایل دستکاری شده است. به همین دلیل به درستی در OllyDBG اجرا نمی شود ⑥
فایل را در LordPE باز کنید، تا مشکلات فایل را بررسی کنیم:



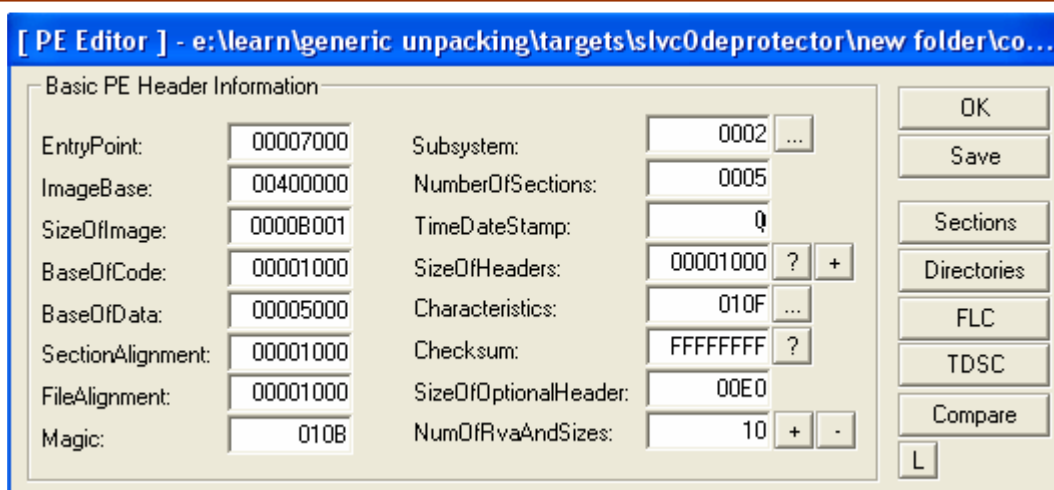
۱. اولین مشکلی که توی فایل می بینیم مقدار NumOfRvaAndSizes هست. همانطور که قبلا اشاره کردم، این مقدار معمولا ۱۰ است.

۲. مورد بعدی TimeDateStamp می باشد که نمایشگر تاریخ ساخت فایل است. در اینجا این مقدار برابر DEADCODE است. که قاعدتا اشتباه است. این مقدار را هم صفر کنید. (البته صفر هم درست نیست، ولی بهتر از اون DEADCODE هست.)

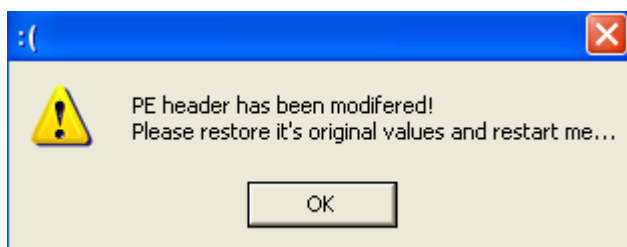
۳. مقدار BaseOfCode برابر F است. BaseOfCode نمایشگر محل شروع سکشنی است که در آن کدهای برنامه قرار دارد. گزینه Sections را بزنید:

[Section Table]					
Name	VOffset	VSize	ROffset	RSize	Flags
.text	00001000	00002A2E	00001000	00003000	E0000020
.rdata	00004000	000007E8	00004000	00001000	C0000040
.data	00005000	000004DC	00005000	00001000	C0000040
.rsrc	00006000	000001A8	00006000	00001000	40000040
::ICU::	00007000	00004000	00007000	00002FA3	A0000020

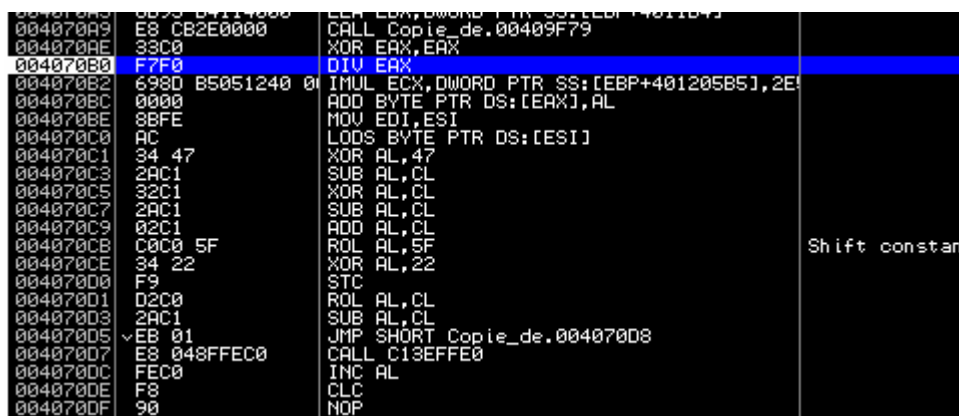
همانطور که می بینید محل شروع سکشن .text برابر ۱۰۰۰ است. پس BaseOfCode در واقع باید ۱۰۰۰ باشد.
در ضمن محل شروع سکشن .data برابر ۵۰۰۰ است. پس BaseOfData هم در واقع باید ۵۰۰۰ باشد.
۴. بهتر است مقدار Checksum رو هم صفر کنید. ولی اگر نکردید هم مشکلی پیش نمی آید.



حالا تغییرات را ذخیره کنید و فایل را در OllyDBG باز کنید. می بینید که فایل بی هیچ مشکلی در OllyDBG باز می شود. اما اگر فایل را با خود ویندوز اجرا کنید:



همانطور که می بینید برنامه متوجه شده است که ما PE Header فایل را دستکاری کرده ایم ② خوب در داخل OllyDBG ابتدا پلاگین HideDBG را فعال کنید. (در منوی Plugins گزینه HideDBG و سپس گزینه Hide All را بزنید.) و بر روی تابع FindWindowA یک Hardware Breakpoint بگذارید. (چون فایل ما از این تابع برای شناسایی OllyDBG استفاده می کند.) در ضمن تیک تمامی خطاها را بردارید و برنامه را اجرا کنید، به خاطر خطا در اینجا متوقف شدیم:



یکبار دیگر Shift + F9 را بزنید، تا چند بار در تابع FindWindowA متوقف شویم. بعد از آن به خاطر خطا در اینجا متوقف می شویم:


```

0040776B F7F2 DIU EDX
0040776D 0F31 RTSC
0040776F 8BC8 MOV ECX,EAX
00407771 65:0F31 RTSC
00407774 2BC1 SUB EAX,ECX
00407776 EB 03 JMP SHORT Copie_de.0040777B
00407778 EB 03 JMP SHORT Copie_de.0040777D
0040777A DFEB FUCOMIP ST,ST(3)
0040777C FB STI
0040777D 3D 00001000 CMP EAX,1000000
00407782 F3: PREFIX REP:
00407783 -0F87 760F3678 JA 787686FF
00407789 8D85 1C234000 LEA EAX,DWORD PTR SS:[EBP+40231C]
0040778F 8038 01 CMP BYTE PTR DS:[EAX],1
00407792 0F85 C6010000 JNZ Copie_de.0040779E
00407798 E9 4A010000 JMP Copie_de.00407797
0040779D 0000 ADD BYTE PTR DS:[EAX],AL
0040779F 0000 ADD BYTE PTR DS:[EAX],AL
004077A1 0000 ADD BYTE PTR DS:[EAX],AL
004077A3 0000 ADD BYTE PTR DS:[EAX],AL
004077A5 0000 ADD BYTE PTR DS:[EAX],AL
004077A7 0000 ADD BYTE PTR DS:[EAX],AL
004077A9 0000 ADD BYTE PTR DS:[EAX],AL

```

همانطور که می بینید در اینجا چند دستور آنتی دیباگ قرار دارد. تابع RDTSC که زمان اجرای یک دستور را محاسبه می کند یکی از آنهاست. بر روی خط 407798 BP بگذارید و برنامه را با Shift + F9 اجرا کنید، بعد از آن چند بار کلید F8 را بزنید تا به اینجا برسید:

```

0040791E 74 2D JE SHORT Copie_de.0040794D
00407920 6A 00 PUSH 0
00407922 8B85 B52C4000 MOV EAX,DWORD PTR SS:[EBP+402CB5]
00407928 50 PUSH EAX
00407929 EB 03 JMP SHORT Copie_de.0040792E
0040792B EB 03 JMP SHORT Copie_de.00407930
0040792D DFEB FUCOMIP ST,ST(3)
0040792F FB STI
00407930 8D85 A2194000 LEA EAX,DWORD PTR SS:[EBP+4019A2]
00407936 8D95 AB194000 LEA EDX,DWORD PTR SS:[EBP+4019AB]
0040793C 6A FF PUSH -1
0040793E 6A 00 PUSH 0
00407940 6A 30 PUSH 30

```

این پرش شرطی چک می کند، آیا نام فایل تغییر کرده است یا خیر؟...این پرش باید همواره انجام شود. پس وقتی به این پرش رسیدید. مقدار Z-Flag را تغییر دهید تا این پرش انجام شود:

```

Registers (FPU)
EAX 004077B8 Copie_de.004077B8
ECX 00000010
EDX 00140608
EBX 00000000
ESP 0012FFA8
EBP 00005EFF
ESI 004077B9 Copie_de.004077B9
EDI 00408379 ASCII "opie de Original.exe"
EIP 0040791E Copie_de.0040791E
C 1 ES 0023 32bit 0(FFFFFFFF)
P 1 CS 001B 32bit 0(FFFFFFFF)
A 1 SS 0023 32bit 0(FFFFFFFF)
Z 1 DS 0023 32bit 0(FFFFFFFF)
O 1 FS 003B 32bit 7FFDF000(FFF)
D 0 GS 0000 NULL
O 0
O 0 LastErr ERROR_FILE_NOT_FOUND (00000002)
EFL 000002D7 (NO,B,E,BE,S,PE,L,LE)
ST0 zero 0.0
ST1 empty -UNORM BDEC 01050104 00000000
ST2 empty 0.0
ST3 empty 0.0
ST4 empty 0.0
ST5 empty 0.0
ST6 empty 0.0
ST7 empty 0.0
FST 3800 Cond 0 0 0 0 Err 0 0 0 0 0 0 0 0
FCW 027F Prec NEAR,53 Mask 1 1 1 1 1 1

```

حالا برنامه را با F8 ادامه دهید تا به یک پرش شرطی دیگر برسید:

```

0793C 6A 00407943 52 PUSH EDX
0793E 6A 00407944 6A 00 PUSH 0
07940 6A 00407946 FF95 CB2C4000 CALL DWORD PTR SS:[EBP+402CCB]
07942 50 0040794C C3 RETN
07943 52 0040794D 803F 2E CMP BYTE PTR DS:[EDI],2E
07944 6A 00407950 74 0C JE SHORT Copie_de.0040795E
07946 FF9 00407952 ^EB CC JMP SHORT Copie_de.00407920
0794C C3 00407954 ^EB 01 JMP SHORT Copie_de.00407957
0794D 803 00407956 68 F3EB0169 PUSH 6901EBF3
07950 74 0040795B ^EB 01 JMP SHORT Copie_de.0040795E
07952 ^EB 0040795D 68 F7D4EB02 PUSH 2EBD4F7
07954 ^EB 00407962 12B4EB 0168F3EB ADC DH,BYTE PTR DS:[EBX+EBP*8+EBF3]
07956 68 00407969 0169 EB ADD DWORD PTR DS:[ECX-15],EBP
07958 ^EB 0040796C 0168 F7 ADD DWORD PTR DS:[EAX-9],EBP
0795D 68 0040796F D4 E8 RAM 0E8
07962 12B 00407971 1C 00 SBB AL,0
07969 016 00407973 0000 ADD BYTE PTR DS:[EAX],AL
0796C 016 00407975 38B0 927E0111 CMP BYTE PTR DS:[EAX+11017E92],DH
0796F D4 0040797B F0:B0 524CA2A6 LOCK MOV EBP,A6A24C52
07971 1C 00407981 A1 8B9E6AEB MOV EAX,DWORD PTR DS:[EB6A9E8B]

```

این پرش هم باید انجام شود.(مقدار Z-flag را در اینجا برابر یک کنید)...اگر این پرش انجام نشود.پیغام File name changed می شود
حالا برنامه را با F9 اجرا کنید،نگاهی به پنجره Stack بیندازید:

```

0012FF9C 00145681 CALL to FindWindowA from 0014567B
0012FFA0 00144527 Class = "OlllyDbg"
0012FFA4 00000000 Title = NULL
0012FFA8 00000000
0012FFAC FFFF0000
0012FFB0 0012FFC4
0012FFB4 00000000
0012FFB8 00000000
0012FFBC 00000000

```

می بینید؟برنامه با تابع FindWindowA به دنبال OlllyDbg می گردد ☺ خوب...بر رو نام OlllyDBG کلیک راست کرده و گزینه Follow in dump را بزنید:

Right-click menu options:

- Search for address
- Search for binary string Ctrl+B
- Go to ESP *
- Go to expression Ctrl+G
- Follow in Dump
- Appearance

```

5681 CALL to FindWindowA from
4527 Class = "OlllyDbg"
0000 Title = NULL
0000
0000
FFC4
0000
0000
0000
9FC3 Copie_de.00409FC3

```

Address	Hex dump	ASCII
00144527	4F 6C 6C 79 44 62 67 00 30 6C 6C 79 44 62 67 00	OlllyDbg.OlllyDbg.
00144537	54 53 70 79 57 69 6E 64 6F 77 00 32 39 39 34 46	TSpYWindow.2994F
00144547	44 32 30 00 54 44 65 44 65 4D 61 69 6E 46 6F 72	D20.TDeMainFor
00144557	6D 00 4F 57 4C 5F 57 69 6E 64 6F 77 00 54 49 64	m.OwL_Window.Tid
00144567	61 57 69 6E 64 6F 77 00 49 6D 70 6F 72 74 20 52	alWindow.Import R
00144577	45 43 6F 6E 73 74 72 75 63 74 6F 72 20 76 31 2E	EConstructor v1.
00144587	36 20 46 49 4E 41 4C 20 28 43 29 20 32 30 30 31	6 FINAL (C) 2001
00144597	20 32 30 30 33 20 4D 61 63 68 54 2F 75 43 46 00	-2003 MackT/ucf.
001445A7	52 65 73 6F 75 72 63 65 20 48 61 63 6B 65 72 00	Resource Hacker.
001445B7	5B 20 4C 6F 72 64 50 45 20 44 65 6C 75 78 65 20	[LordPE Deluxe
001445C7	5D 20 62 79 20 79 6F 64 61 00 50 45 69 44 20 76	] by yoda.PEId v
001445D7	30 2E 39 33 00 50 45 20 54 6F 6F 6C 73 20 76 31	0.93.PE Tools v1
001445E7	2E 35 20 58 6D 61 73 20 45 64 69 74 69 6F 6E 20	.5 Xmas Edition
001445F7	20 20 62 79 20 45 4F 4F 73 2F 5F 75 69 6F 48 5D	- by N50y4FrieC7

Hardware breakpoint 1 at USER32.FindWindowA

همانطور که می بینید در اینجا دو مقدار با نام OlllyDBG نوشته شده است.ما برای اینکه برنامه نتواند پنجره OlllyDBG را پیدا کند.می توانیم نام این دو مقدار را عوض کنیم(برای این کار در قسمت ASCII نام ها را انتخاب کنید و کلیدهای CTRL + E را بزنید و آن قسمت از دامپ را ویرایش کنید)

Address	Hex dump	ASCII
00144527	70 6C 6C 79 44 62 67 00 70 6C 6C 79 44 62 67 00	IllyDbg. IllyDbg.
00144537	54 53 70 79 57 69 6E 64 6F 77 00 32 39 39 34 46	TSpvWindow.2994F
00144547	44 32 30 00 54 44 65 44 65 40 61 69 6E 46 6F 72	D20.TDeMainFor
00144557	60 00 4F 57 4C 5F 57 69 6E 64 6F 77 00 54 49 64	m.OwL_Window.Tid
00144567	61 57 69 6E 64 6F 77 00 49 6D 70 6F 72 74 20 52	aWindow.Import R

همانطور که می بینید من نام OllyDBG را به pllyDBG تغییر دادم. تا برنامه نتواند چیزی پیدا کند. خوب، برنامه را با F9 ادامه دهید، در چند جا متوقف می شویم، برنامه به غیر از ollyDBG به دنبال برنامه دیگری هم می گردد. (مثل ImportREC) ☹
بعد از اینکه تمامی اینها را رد کردیم. به این پیغام می رسیم:



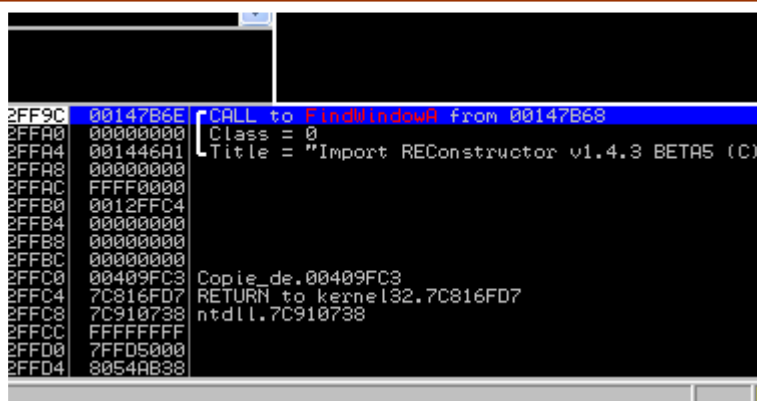
حالا چه جوری این پیغام را از بین ببرم؟ برنامه را در OllyDBG متوقف کنید. (کلید F12 را بزنید.) و کلید ALT + F9 را بزنید تا وارد حالت Back to user شوید. و بعد هم کلید OK را بزنید تا پیغام داده شده برود، به این ترتیب در اینجا متوقف می شویم:

00143ECA	FF95 E52C4000	CALL DWORD PTR SS:[EBP+402CE5]	
00143ED0	B8 BD2A6348	MOV EAX,48632ABD	
00143ED5	8B57 74	MOV EDX,DWORD PTR DS:[EDI+74]	
00143ED8	3BC2	CMP EAX,EDX	
00143EDA	74 7F	JE SHORT 00143F5B	
00143EDC	6A 00	PUSH 0	
00143EDE	8B85 B52C4000	MOV EAX,DWORD PTR SS:[EBP+402CB5]	
00143EE4	50	PUSH EAX	
00143EE5	8D85 EE204000	LEA EAX,DWORD PTR SS:[EBP+4020EE]	
00143EEB	8D95 99204000	LEA EDX,DWORD PTR SS:[EBP+402099]	
00143EF1	6A FF	PUSH -1	
00143EF3	6A 00	PUSH 0	
00143EF5	6A 30	PUSH 30	
00143EF7	50	PUSH EAX	
00143EF8	52	PUSH EDX	
00143EF9	6A 00	PUSH 0	
00143EFB	FF95 CB2C4000	CALL DWORD PTR SS:[EBP+402CCB]	
00143F01	C3	RETN	
00143F02	6950 45 2068656	IMUL EDX,DWORD PTR DS:[EAX+45],61656820	Super
00143F09	64:	PREFIX FS:	Super
00143F0A	65:72 20	JB SHORT 00143F2D	
00143F0D	68 61732062	PUSH 62207361	Super
00143F12	65:	PREFIX GS:	Super
00143F13	65:6E	OUTS DX,BYTE PTR ES:[EDI]	I/O
00143F15	206D 6F	AND BYTE PTR SS:[EBP+6F],CH	
00143F18	64:6966 65 7265	IMUL ESP,DWORD PTR FS:[ESI+65],21646572	
00143F20	0A0D 506C6561	OR CL,BYTE PTR DS:[61656C50]	
00143F26	73 65	JNB SHORT 00143F8D	
00143F28	2072 65	AND BYTE PTR DS:[EDX+65],DH	

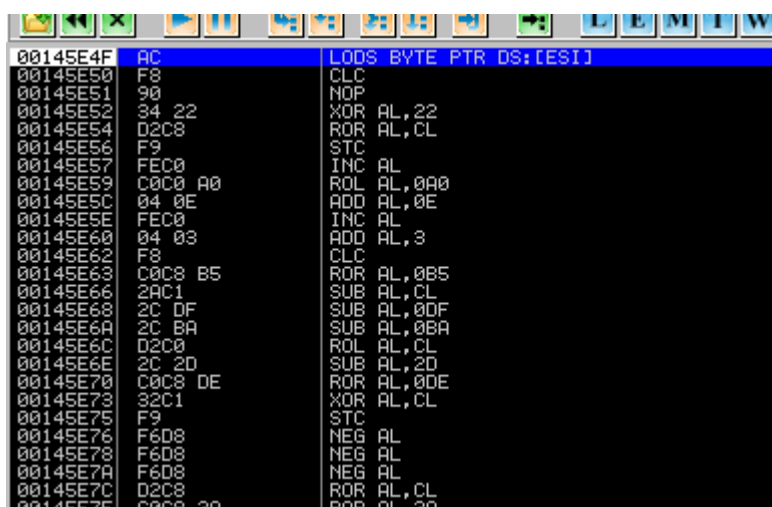
پرشی که در خط 143EDA (ممکن است آدرس این خطوط در سیستم شما فرق داشته باشد) تصمیم گیرنده است. یعنی اگر این پرش انجام شود، برنامه چنین پیغامی را نمی دهد. پس بر روی این پرش Hardware breakpoint on execution بگذارید و برنامه را ری استارت کنید. حالا تمامی کارهایی که قبلا انجام دادیم را دوباره انجام دهید. تا دوباره به این پرش برسیم:

00143ECA	FF95 E52C4000	CALL DWORD PTR SS:[EBP+402CE5]	
00143ED0	B8 BD2A6348	MOV EAX,48632ABD	
00143ED5	8B57 74	MOV EDX,DWORD PTR DS:[EDI+74]	
00143ED8	3BC2	CMP EAX,EDX	
00143EDA	74 7F	JE SHORT 00143F5B	
00143EDC	6A 00	PUSH 0	
00143EDE	8B85 B52C4000	MOV EAX,DWORD PTR SS:[EBP+402CB5]	
00143EE4	50	PUSH EAX	
00143EE5	8D85 EE204000	LEA EAX,DWORD PTR SS:[EBP+4020EE]	
00143EEB	8D95 99204000	LEA EDX,DWORD PTR SS:[EBP+402099]	
00143EF1	6A FF	PUSH -1	
00143EF3	6A 00	PUSH 0	
00143EF5	6A 30	PUSH 30	
00143EF7	50	PUSH EAX	
00143EF8	52	PUSH EDX	
00143EF9	6A 00	PUSH 0	
00143EFB	FF95 CB2C4000	CALL DWORD PTR SS:[EBP+402CCB]	

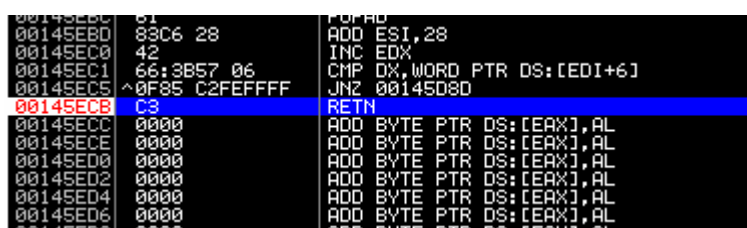
حالا مقدار Z-flag را تغییر دهید تا این پرش انجام شود. حالا کلید F9 را بزنید، می بینید برنامه دوباره می خواهد با تابع FindWindowA پنجره OllyDBG را پیدا کند، اما ما قبلا نام OllyDBG را دستکاری کردیم، پس مشکلی به وجود نمی آید. ☺ چند بار کلید F9 را بزنید تا به آخرین تابع FindWindowA برسیم:



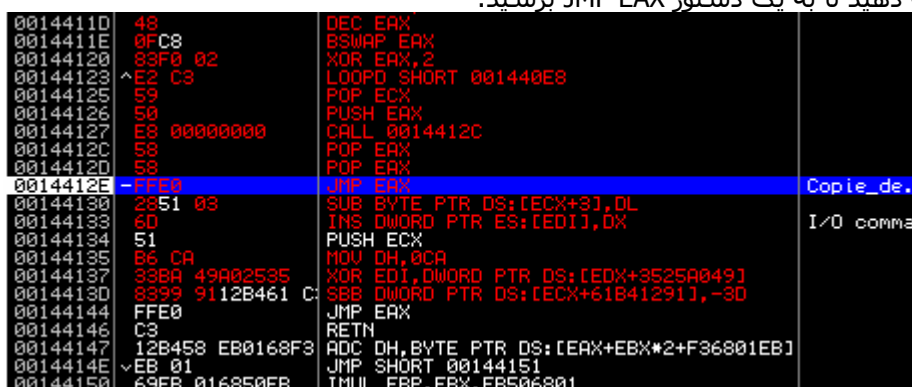
اینجا دیگر آخرین جایی است که برنامه ما از تابع FindWindowA استفاده می کند. حالا بر روی سکشن text. برنامه یک Memory Breakpoint بگذارید. و بعد برنامه را با F9 اجرا کنید:



اینجا جایی است که سکشن text. Decrypt می شود. پس با موس به پایین بروید و بر روی RET که در آخرین کدها قرار دارد، Breakpoint بگذارید. بعد Memory Breakpoint که گذاشته بودید را پاک کنید. (کلیک راست کرده، گزینه Breakpoint و سپس گزینه Remove memory Breakpoint را بزنید). و بعد برنامه را با F9 اجرا کنید تا در جایی که breakpoint گذاشتیم متوقف شویم:



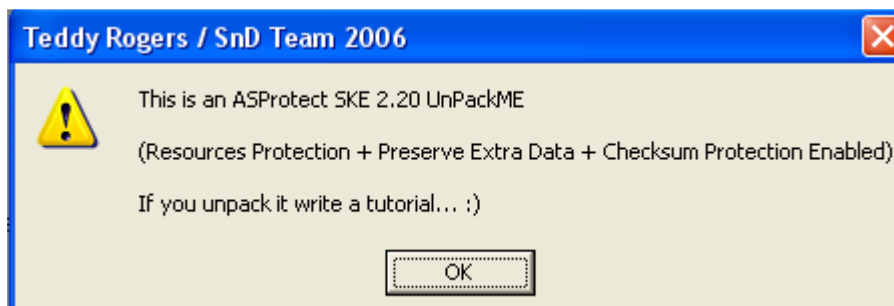
حالا با F8 برنامه را ادامه دهید تا به یک دستور JMP EAX برسید:



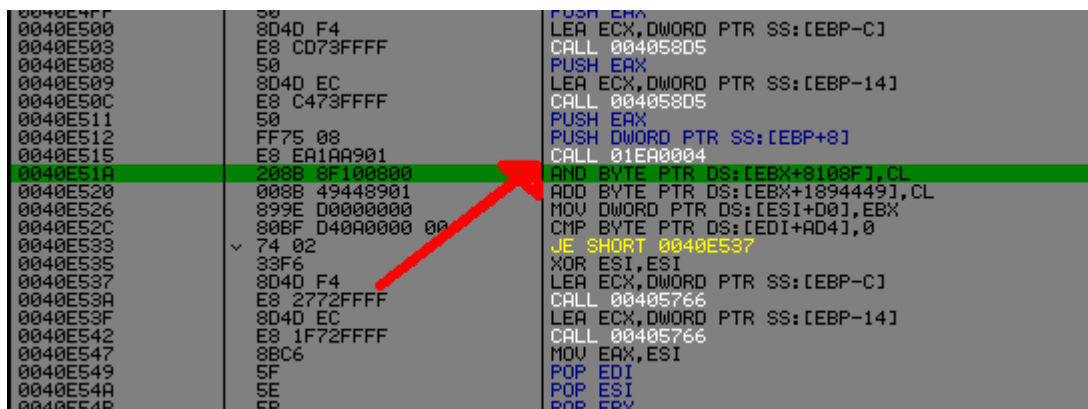
این پرش ما را به OEP می برد ©
حالا از فایل دامپ بگیرید. (ترجیحا با خود ImportREC این کار را کنید).
و بعد با ImportREC فایل را فیکس کنید. (تیک گزینه Use PE Header from disk را هم باید بزنید).
فایل آنپک شده را اجرا کنید، می بینید که بی هیچ مشکلی اجرا می شود ©

Asprotect

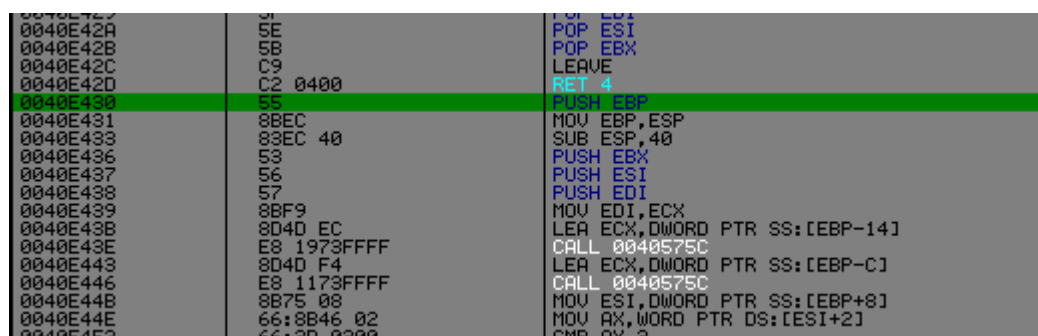
فایل مورد نظر را در OllyDBG باز کنید. برای یافتن OEP می توانیم از روش Back trace استفاده کنیم. یعنی جایی از برنامه را پیدا کنیم که مطمئن باشیم، آنجا بعد از OEP است و بعد هم آنقدر به عقب برویم تا به OEP برسیم ☺ ابتدا برنامه را به طور کامل اجرا کنید:



حالا کلید F12 را بزنید تا برنامه متوقف شود و بعد از آن کلید ALT + F9 را بزنید تا وارد حالت Back to user شوید. بعد هم کلید OK را بزنید تا این پیغام از بین برود و برنامه در اینجا متوقف می شود:



تابع MessageBoxA از داخل تابع بالایی صدا زده شده. خوب، با موس به بالا بروید تا ابتدای این Routine را پیدا کنید:

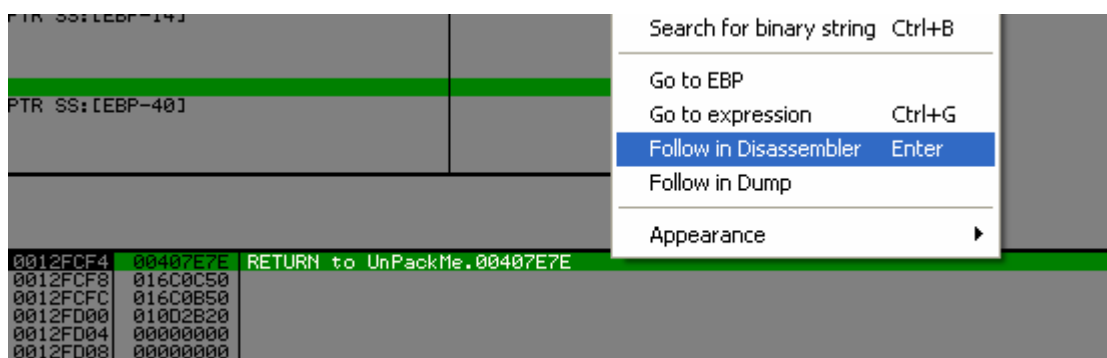


اینجا ابتدای Routine است. بر روی آن Hardware breakpoint on execution بگذارید و برنامه را ری استارت کنید و دوباره اجرا کنید:



ما در جایی که Breakpoint گذاشتیم متوقف شدیم. حالا باید ببینیم این Routine از کجا صدا زده شده؟

در پنجره دامپ همانند تصویر کلیک راست کرده و گزینه Follow in disassembler را بزنید:



00407E63	75 20	JNZ SHORT 00407E85
00407E65	8B86 D40C0000	MOV EAX,DWORD PTR DS:[ESI+CD4]
00407E68	8B9E D40A0000	MOV BYTE PTR DS:[ESI+AD4],BL
00407E71	50	PUSH EAX
00407E72	8BCE	MOV ECX,ESI
00407E74	0FB710	MOVZX EDX,WORD PTR DS:[EAX]
00407E77	FF9496 F80C0000	CALL DWORD PTR DS:[ESI+EDX*4+CF8]
00407E7E	8BF8	MOV EDI,EAX
00407E80	F7DF	NEG EDI
00407E82	1BFF	SBB EDI,EDI
00407E84	47	INC EDI
00407E85	FF86 CC0A0000	INC DWORD PTR DS:[ESI+ACC]
00407E8B	8B86 CC0A0000	MOV EAX,DWORD PTR DS:[ESI+ACC]
00407E91	3B86 C80A0000	CMP EAX,DWORD PTR DS:[ESI+AC8]
00407E97	73 08	JNB SHORT 00407EA1
00407E99	3B9E D40A0000	CMP BYTE PTR DS:[ESI+AD4],BL
00407E9F	74 03	JE SHORT 00407EA4
00407EA1	6A 01	PUSH 1
00407EA3	5F	POP EDI
00407EA4	3BFB	CMP EDI,EBX
00407EA6	74 98	JE SHORT 00407E40
00407EA8	3B9E D40A0000	CMP BYTE PTR DS:[ESI+AD4],BL
00407EAE	75 07	JNZ SHORT 00407EB7
00407EB0	C745 FC 01000000	MOV DWORD PTR SS:[EBP-4],1
00407EB7	8B06	MOV EAX,DWORD PTR DS:[ESI]

حالا هم دوباره همان عمل را انجام دهید. یعنی با موس به ابتدای Routine بروید و بر روی آن Hardware breakpoint on execution بگذارید.

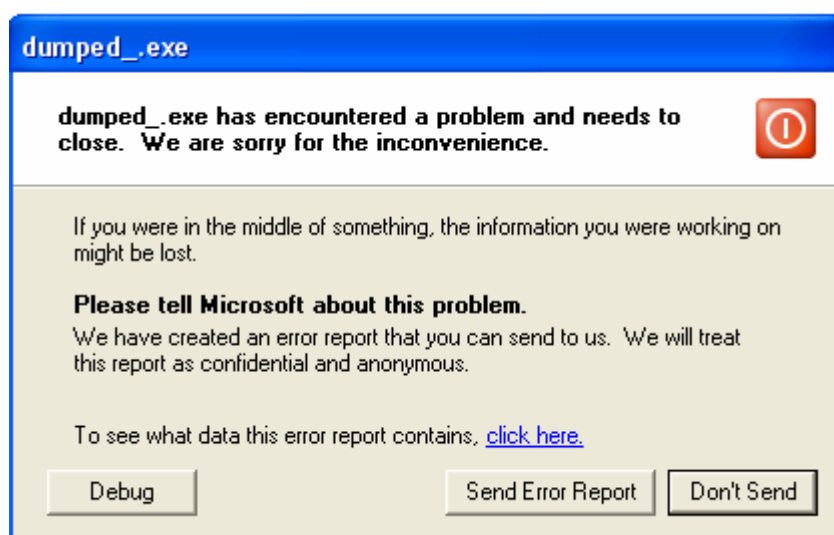
00407DA3	C9	LEAVE
00407DA4	C3	RET
00407DA5	55	PUSH ESP
00407DA6	8BEC	MOV EBP,ESP
00407DA8	51	PUSH ECX
00407DA9	53	PUSH EBX
00407DAA	56	PUSH ESI
00407DAB	8BF1	MOV ESI,ECX
00407DAD	33DB	XOR EBX,EBX
00407DAF	57	PUSH EDI
00407DB0	53	PUSH EBX
00407DB1	33FF	XOR EDI,EDI
00407DB3	8B9E D40A0000	MOV BYTE PTR DS:[ESI+AD4],BL
00407DB9	8B9E D50A0000	MOV BYTE PTR DS:[ESI+AD5],BL
00407DBF	895D FC	MOV DWORD PTR SS:[EBP-4],EBX
00407DC2	E8 16520100	CALL 0041CFDD
00407DC7	59	POP ECX
00407DC8	E8 C24F0100	CALL 0041CD8F
00407DCD	399E D8070000	CMP DWORD PTR DS:[ESI+7D8],EBX
00407DD3	0F84 E4000000	JE 00407EB0
00407DD9	399E B80A0000	CMP DWORD PTR DS:[ESI+AB8],EBX
00407DDF	0F84 D8000000	JE 00407EB0
00407DE5	8B06	MOV EAX,DWORD PTR DS:[ESI]
00407DE7	8BCE	MOV ECX,ESI

چند بار این عمل را انجام دهید. (یعنی ابتدای Routine را پیدا می کنید. Hardware breakpoint می گذارید، برنامه را ری استارت می کنید و با استفاده از پنجره دامپ جایی که این Routine صدا زده شده را پیدا کنید.) تا بالاخره به اینجا می رسید:

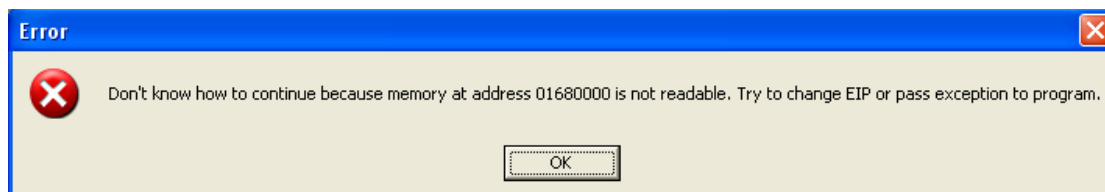
004271AC	90	NOP
004271AD	90	NOP
004271AE	90	NOP
004271AF	90	NOP
004271B0	55	PUSH EBP
004271B1	8BEC	MOV EBP,ESP
004271B3	6A FF	PUSH -1
004271B5	68 600E4500	PUSH 00450E60
004271B9	68 C8924200	PUSH 004292C8
004271BF	64:A1 00000000	MOV EAX,DWORD PTR FS:[0]
004271C5	50	PUSH EAX
004271C6	64:8925 00000000	MOV 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 SS:[EBP-18],ESP
004271D6	FF15 DC0A4600	CALL DWORD PTR DS:[460ADC]
004271DC	33D2	XOR EDX,EDX
004271DE	8AD4	MOV DL,AH
004271E0	8915 34E64500	MOV DWORD PTR DS:[45E634],EDX
004271E6	8BC8	MOV ECX,EAX
004271E8	81E1 FF000000	AND ECX,0FF
004271EE	890D 30E64500	MOV DWORD PTR DS:[45E630],ECX
004271F4	C1E1 08	SHL ECX,8
004271F7	03CA	ADD ECX,EDX
004271F9	890D 2CE64500	MOV DWORD PTR DS:[45E62C],ECX

اینجا OEP ماست...از کجا فهمیدم؟...خوب چون چند خط بعد از این تابع GetVersion وجود دارد و در ضمن اگر بر روی این خط هم یک Hardware breakpoint بگذارید.و بخواهید با پنجره Dump جایی که این تابع صدا زده شده را پیدا کنید.می بینید که در پنجره Dump چیزی نیست ☺

حالا بر روی OEP، Hardware bp، بگذارید و دوباره برنامه را اجرا کنید و وقتی در OEP متوقف شدید،از فایل دامپ بگیرید. و بعد با ImportREC فایل را فیکس می کنید.حالا فایل دامپ شده را اجرا کنید:



فایل دامپ شده اجرا نمی شود ☹ اصولا در مورد هر پکری به این مشکل برخوردید و دیدید که فایل شما اجرا نمی شود.همیشه فایل آنپک شده را با OllyDBG باز می کنید تا ببینید مشکل چیست؟...چرا فایل درست جواب نمی دهد؟...خطا در چه خطی اتفاق می افتد؟...چه طور می شود مشکل را حل کرد؟
یک OllyDBG دیگر باز کنید و فایل آنپک شده را در آن اجرا کنید(کلید F9):



در این پیغام نوشته شده که فایل آنپک شده قصد داشته به خط 01680000 برود.(یا این خود را بخواند.) اما چنین خطی اصلا وجود ندارد ☹ به همین دلیل برنامه دچار خطا می شود.
خوب،حالا چه کار کنیم؟به چند خط زیر OEP در فایل اصلی(پک شده) نگاه کنید:

00427225	E8 38010000	CALL 00427360
0042722A	83C4 04	ADD ESP,4
0042722D	C745 FC 00000000	MOV DWORD PTR SS:[EBP-4],0
00427234	E8 872B0000	CALL 00429DC0
00427239	E8 12110000	CALL 00428350
0042723E	E8 BD0D2501	CALL 01680000
00427243	21A3 D8EB4500	AND DWORD PTR DS:[EBX+45EBD8],E
00427249	E8 32940000	CALL 00430680
0042724E	A3 10E64500	MOV DWORD PTR DS:[45E610],EAX
00427253	85C0	TEST EAX,EAX
00427255	74 09	JE SHORT 00427260
00427257	A1 D8EB4500	MOV EAX,DWORD PTR DS:[45EBD8]
0042725C	85C0	TEST EAX,EAX
0042725E	75 0A	JNZ SHORT 0042726A
00427260	6A FF	PUSH -1
00427262	E8 490B0000	CALL 00427DB0
00427267	83C4 04	ADD ESP,4
0042726A	E8 61910000	CALL 004303D0
0042726F	E8 6C900000	CALL 004302E0

می بینید چند خط زیر OEP در فایل اصلی ما دستور Call 01680000 وجود دارد. بر روی آن کلید Enter را بزنید:

Address	Hex dump	Disassembly
01680000	3E:EB 02	JMP SHORT 01680005
01680003	CD20 509CF3EB	UxDJump EBF39C50
01680009	02CD	ADD CL,CH
0168000B	2023	AND BYTE PTR DS:[EBX],AH
0168000D	C683 EC20F3EB 02	MOV BYTE PTR DS:[EBX+EBF3],
01680014	CD20 81C0BED	UxDJump ED0BC881
0168001A	7F F1	JG SHORT 01680000
0168001C	B8 06924200	MOV EAX,429206
01680021	034424 38	ADD EAX,DWORD PTR SS:[ESP
01680025	8D4424 0F	LEA EAX,DWORD PTR SS:[ESP
01680029	83E8 0F	SUB EAX,0F
0168002C	55	PUSH EBP
0168002D	8F40 14	POP DWORD PTR DS:[EAX+14]
01680030	33E9	XOR EBP,ECX
01680032	36:EB 01	JMP SHORT 01680036
01680035	6956 8F 401813F3	IMUL EDX,DWORD PTR DS:[ES
0168003C	8958 0C	MOV DWORD PTR DS:[EAX+C],
0168003F	F2:	PREFIX REPNE:
01680040	EB 01	JMP SHORT 01680043
01680042	F3:	PREFIX REP:
01680043	83EB 81	SUB EBX,-7F
01680046	65:EB 01	JMP SHORT 01680040

می بینید؟ این آدرس در فایل پک شده وجود دارد. ولی در فایل آپیک شده وجود ندارد (در واقع Asprotect قبل از رسیدن به OEP این آدرس ها را تهیه کرده و این سکشن ها را ساخته است).

با اندکی تلاش متوجه موضوعی خواهید شد. ببینید ما در تعمیر IAT به مشکلی برخوردیم... چرا؟... چون Asprotect با IAT کاری نداشته... و آنها را همونجوری که بود دوباره ساخته... اما؟... توابع API برنامه را دستکاری کرده و برخی از آنها را به دستور CALL 01680000 تبدیل کرده ☺

در واقع دستوری که در خط 42723E قرار دارد (CALL 1680000) باید یک همچین دستوری باشد:

Call DWORD PTR DS:[xxxxxx]

خوب، حالا برای حل این مشکل چه کار کنیم؟... یک راه حل این است که ما باییم و سکشنی را که در آن خط 1680000 قرار دارد، به فایل آپیک شده با نرم افزارهایی مثل LoadPe اضافه کنیم، این روش در مورد این فایل جواب نمی دهد. (در برخی از فایل های Asprotect جواب می دهد ولی در این یکی نه ☺)

خوب، بذارید من این خط را (42723E) فیکس کنم. بر روی آن کلیک راست کرده و گزینه new origin here را بزنید.... ما باید ببینیم که این خط به کجا می رود؟... برای این کار خودتان یک راه حل پیدا کنید ☺ ... من این کارو کردم:

بعد از اینکه بر روی این خط new origin here گذاشتم، کلیدهای ALT + M را می زنم و وارد Memory Map می شوم... حالا بر روی سکشن text از فایل Kernel32.dll یک Memory Breakpoint می گذارم:

77F61000	0006C000	SHLWAPI	77F60000	.text	code, imports, exports	Imag
77FCD000	00001000	SHLWAPI	77F60000	.data	data	Imag
77FCE000	00002000	SHLWAPI	77F60000	.rsrce	resources	Imag
77FD0000	00006000	SHLWAPI	77F60000	.reloc	relocations	Imag
7C800000	00001000	kernel32	7C800000	(itself)	PE header	Imag
7C801000	00003000	kernel32	7C800000	.text	code, imports, exports	Imag
7C804000	00005000	kernel32	7C800000	.data	data	Imag
7C809000	00006000	kernel32	7C800000	.rsrce	resources	Imag
7C80F000	00006000	kernel32	7C800000	.reloc	relocations	Imag
7C900000	00001000	ntdll	7C900000	(itself)		Imag
7C901000	00007B000	ntdll	7C900000	.text	code, imports, exports	Imag
7C97C000	00005000	ntdll	7C900000	.data	data	Imag
7C981000	0002C000	ntdll	7C900000	.rsrce	resources	Imag
7C9AD000	00003000	ntdll	7C900000	.reloc	relocations	Imag
7C9C0000	00001000	SHELL32	7C9C0000	(itself)		Imag
7C9C1000	0001FC000	SHELL32	7C9C0000	.text	code, imports, exports	Imag
7C9BD000	00010000	SHELL32	7C9C0000	.data	data	Imag
7C9BD000	0005E0000	SHELL32	7C9C0000	.rsrce	resources	Imag
7D1BA000	00018000	SHELL32	7C9C0000	.reloc	relocations	Imag
7DF70000	00001000	ole32	7DF70000	(itself)		Imag
7DF71000	00013000	ole32	7DF70000	.text	code, imports, exports	Imag
7DF84000	00002000	ole32	7DF70000	.data	data	Imag
7DF86000	0000B000	ole32	7DF70000	.rsrce	resources	Imag
7DF91000	00001000	ole32	7DF70000	.reloc	relocations	Imag

حالا برنامه را با F9 اجرا کنید:

Address	Hex dump	Disassembly
7C809920 GetCurre	64:A1 10000000	MOV EAX,DWORD PTR FS:[10]
7C809926	8B40 20	MOV EAX,DWORD PTR DS:[EAX+20]
7C809929	C3	RET
7C80992A	90	NOP
7C80992B	90	NOP
7C80992C	90	NOP
7C80992D	90	NOP
7C80992E	90	NOP
7C80992F LocalFre	6A 18	PUSH 18
7C809931	68 7099807C	PUSH 7C809970

حالا سه بار کلید F9 را بزنید تا به اینجا برسیم:

Address	Hex dump	Disassembly
801D77 LoadLibr	8BFF	MOV EDI,EDI
801D79	55	PUSH EBP
801D7A	8BEC	MOV EBP,ESP
801D7C	837D 08 00	CMP DWORD PTR SS:[EBP+8],0
801D80	53	PUSH EBX
801D81	56	PUSH ESI
801D82	74 14	JE SHORT 7C801D98
801D84	68 C0E0807C	PUSH 7C80E0C0
801D89	FF75 08	PUSH DWORD PTR SS:[EBP+8]
801D8C	FF15 9C13807C	CALL DWORD PTR DS:[<ntdll._strcmpi>]
801D92	85C0	TEST EAX,EAX
801D94	59	POP ECX
801D95	59	POP ECX
801D96	74 12	JE SHORT 7C801DAA
801D98	6A 00	PUSH 0
801D9A	6A 00	PUSH 0
801D9C	FF75 08	PUSH DWORD PTR SS:[EBP+8]
801D9F	E8 ABFFFFFF	CALL LoadLibraryExA

اینجا تابع LoadLibraryA است. حالا Memory Breakpoint که گذاشته بودیم را پاک کنید و برنامه را با F8 آنقدر ادامه دهید، تا به اینجا برسید:

Address	Hex dump	Disassembly
01006BB2	84C0	TEST AL,AL
01006BB4	8B45 F4	MOV EAX,DWORD PTR SS:[EBP-C]
01006BB7	8B80 E0000000	MOV EAX,DWORD PTR DS:[EAX+E0]
01006BB0	0345 E4	ADD EAX,DWORD PTR SS:[EBP-1C]
01006BC0	8945 FC	MOV DWORD PTR SS:[EBP-4],EAX
01006BC3	57	PUSH EDI
01006BC4	6A 00	PUSH 0
01006BC6	8D4D E0	LEA ECX,DWORD PTR SS:[EBP-20]
01006BC9	8B45 F4	MOV EAX,DWORD PTR SS:[EBP-C]
01006BCC	8B40 3C	MOV EAX,DWORD PTR DS:[EAX+3C]
01006BCF	8B55 FC	MOV EDX,DWORD PTR SS:[EBP-4]
01006BD2	E8 211FFFFFFF	CALL 00FF8AF8
01006BD7	8945 FC	MOV DWORD PTR SS:[EBP-4],EAX
01006BDA	8B45 E0	MOV EAX,DWORD PTR SS:[EBP-20]
01006BDD	8B00	MOV EAX,DWORD PTR DS:[EAX]
01006BDF	E8 D8E7FFFF	CALL 010053BC
01006BE4	8BD0	MOV EDX,EAX
01006BE6	02D3	ADD DL,BL
01006BE8	8B4D FC	MOV ECX,DWORD PTR SS:[EBP-4]
01006BEB	8B45 F4	MOV EAX,DWORD PTR SS:[EBP-C]
01006BEE	E8 7D040000	CALL 01007070
01006BF3	8945 FC	MOV DWORD PTR SS:[EBP-4],EAX
01006BF6	8B45 F4	MOV EAX,DWORD PTR SS:[EBP-C]
01006BF9	8B40 24	MOV EAX,DWORD PTR DS:[EAX+24]
01006BFC	8B55 F4	MOV EDX,DWORD PTR SS:[EBP-C]

توجه کنید که این آدرس ها در سیستم شما فرق دارد. بر روی جایی که الان هستید، یک Hardware breakpoint بگذارید، حالا نگاهی به رجیسترها بیندازید:

```

Registers (FPU)
EAX 7C812F1D kernel32.GetCommandLineA
ECX 0012FB0C
EDX 00000000
EBX 00000040
ESP 0012FB18
EBP 0012FB4C
ESI 0012FBB4
EDI 397AEAE6
EIP 01006BC0

C 1 ES 0023 32bit 0(FFFFFFFF)
P 1 CS 001B 32bit 0(FFFFFFFF)
A 0 SS 0023 32bit 0(FFFFFFFF)
Z 0 DS 0023 32bit 0(FFFFFFFF)
S 0 FS 003B 32bit 7FDD0000(FFF)
T 0 GS 0000 NULL
D 0
O 1 LastErr ERROR_FILE_NOT_FOUND (00000002)
EFL 00000A07 (O,B,NE,BE,NS,PE,L,LE)
ST0 empty -UNORM BB90 01050104 00000000
ST1 empty 0.0
ST2 empty 0.0
ST3 empty 0.0
ST4 empty 0.0
ST5 empty 0.0
ST6 empty 1.000000000000000000000000
ST7 empty 96.0000000000000000000000

3 2 1 0 E S P U O Z D I
FST 4020 Cond 1 0 0 0 Err 0 0 1 0 0 0 0 (EQ)
FCW 027F Prec NEAR,53 Mask 1 1 1 1 1 1

```

متغیر EAX الان نشاندهنده خطی است که تابعی که در خط (42723E) قرار دارد در واقع به آنجا می رود ☺ پس این خط در واقع به GetCommandLine می رود ☺
دوباره به آدرس 42723E بروید و جای دستور CALL 1680000 بنویسید:

Call Dword ptr DS: [460984]

5H 10	PUSH 10	
E8 36010000	CALL 00427360	
83C4 04	ADD ESP,4	
C745 FC 00000000	MOV DWORD PTR SS:[EBP-4],0	
E8 872B0000	CALL 00429DC0	
E8 12110000	CALL 00428350	
FF15 84094600	CALL DWORD PTR DS:[460984]	kernel32.GetCommandLineA
A3 08EB4500	MOV DWORD PTR DS:[45EBD8],EAX	
E8 32940000	CALL 00430680	
A3 10E64500	MOV DWORD PTR DS:[45E610],EAX	
85C0	TEST EAX,EAX	
74 09	JE SHORT 00427260	
A1 08EB4500	MOV EAX,DWORD PTR DS:[45EBD8]	
85C0	TEST EAX,EAX	
75 0A	JNZ SHORT 0042726A	

حواستان باشد که توابع را باید به این ترتیب درست کنید. یعنی از IAT استفاده کنید. الان آدرس تابع GetCommandLineA در خط 460984 قرار دارد(در IAT برنامه که از خط 460814 شروع می شود)،نگاهی به IAT بیندازید:

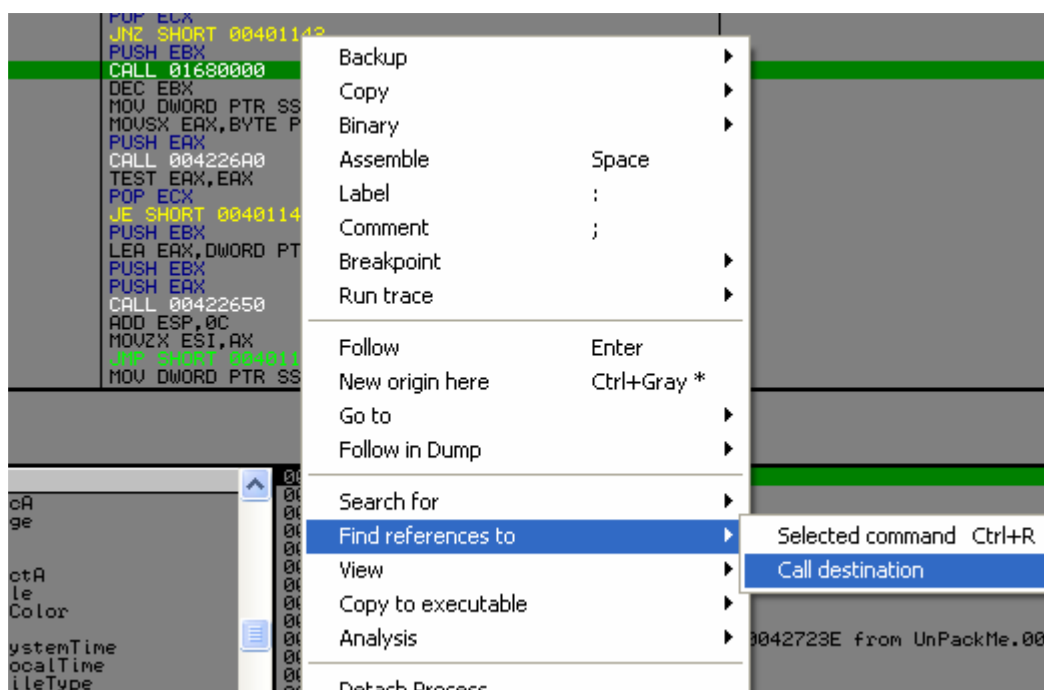
Address	Value	ASCII	Comment
00460954	77F450A9	rPw	GDI32.StartDocA
00460958	77F2F116	.+zW	GDI32.StartPage
0046095C	77F2D0B1	Wz	GDI32.EndPage
00460960	77F2E041	AzW	GDI32.EndDoc
00460964	77F18C0E	AtzW	GDI32.GetObjectA
00460968	77F1E649	IpzW	GDI32.Rectangle
0046096C	77F193F9	.dzW	GDI32.GetTextColor
00460970	00000000	...	
00460974	7C80176B	k*Q:	kernel32.GetSystemTime
00460978	7C80A7D4	*Q:	kernel32.GetLocalTime
0046097C	7C810E51	QW:	kernel32.GetFileType
00460980	7C801EEE	eAQ:	kernel32.GetStartupInfoA
00460984	7C812F1D	#/u:	kernel32.GetCommandLineA
00460988	7C937A40	@zd:	ntdll.RtlUnwind
0046098C	7C812A09	.*u:	kernel32.RaiseException
00460990	7C81CDDA	=u:	kernel32.ExitProcess
00460994	7C801E16	.AQ:	kernel32.TerminateProcess
00460998	7C809915	30Q:	kernel32.GetACP
0046099C	7C810EF8	*W:	kernel32.HeapDestroy
004609A0	7C812BB6	tl+u:	kernel32.HeapCreate
004609A4	7C809AE4	2uQ:	kernel32.VirtualFree
004609A8	7C809A51	QUQ:	kernel32.VirtualAlloc
004609AC	7C8214E3	tlle:	kernel32.GetDriveTypeA
004609B0	7C8350BF	~Pa:	kernel32.GetTimeZoneInformation
004609B4	7C838DE8	la:	kernel32.LCMapStringA
004609B8	7C80CCA8	dzQ:	kernel32.LCMapStringW
004609BC	7C862E2A	*.AQ:	kernel32.UnhandledExceptionFilter
004609C0	7C81DF77	wu:	kernel32.FreeEnvironmentStringsA
004609C4	7C814AE7	~Ju:	kernel32.FreeEnvironmentStringsW
004609C8	7C81CF5B	[u:	kernel32.GetEnvironmentStringsA
004609CC	7C812F08	u/u:	kernel32.GetEnvironmentStringsW
004609D0	7C84467D	~Fa:	kernel32.SetUnhandledExceptionFilter
004609D4	7C838A0C	.eAQ:	kernel32.GetStringTypeA
004609D8	7C80A490	dzQ:	kernel32.GetStringTypeW
004609DC	7C80BCCF	=uQ:	kernel32.IsBadCodePtr
004609E0	7C80D262	bmQ:	kernel32.GetLocaleInfoA
004609E4	7C811562	bsu:	kernel32.GetLocaleInfoW

Command:

خوب...حالا به نظرتان چند تا از توابع فایل توسط Asprotect دستکاری شده است؟
 کلیدهای CTRL + F را بزنید و تایپ کنید: CALL 1680000

00401106	80B5 F4FFFFFF	LEA ESI,DWORD PTR SS:[EBP-100]
00401108	E8 65160200	CALL 00422770
0040110C	59	POP ECX
0040110E	85C0	TEST EAX,EAX
0040110F	59	POP ECX
00401111	75 32	JNZ SHORT 00401143
00401112	53	PUSH EBX
00401117	E8 E9EE2701	CALL 01680000
00401118	4B	DEC EBX
0040111B	8945 F8	MOV DWORD PTR SS:[EBP-8],EAX
00401122	0FB85 F4FFFFFF	MOVSX EAX,BYTE PTR SS:[EBP-10]
00401123	50	PUSH EAX
00401128	E8 78150200	CALL 004226A0
0040112A	85C0	TEST EAX,EAX
0040112B	59	POP ECX
0040112D	74 1D	JE SHORT 0040114A
0040112E	53	PUSH EBX
00401134	8D85 F4FFFFFF	LEA EAX,DWORD PTR SS:[EBP-100]
00401135	53	PUSH EBX
00401136	50	PUSH EAX

بفرما ! یک تابع دیگر پیدا کردیم.بذارید ببینیم چند تا از این دستورها هست?...همانند تصویر کلیک راست کرده و گزینه Find references to را بزنید:



00401112	CALL 01680000	(Initial CPU selection)
004011E8	CALL 01680000	
00401CE9	CALL 01680000	
00401DC0	CALL 01680000	
00405CC6	CALL 01680000	
00407D31	CALL 01680000	
00407F11	CALL 01680000	
00407F68	CALL 01680000	
00408CF8	CALL 01680000	
00408CFF	CALL 01680000	
0040954C	CALL 01680000	
00409557	CALL 01680000	
00409DEB	CALL 01680000	
0040AEC6	CALL 01680000	
0040AEE0	CALL 01680000	
0040AEF7	CALL 01680000	
0040DAEC	CALL 01680000	
0040E515	CALL 01680000	
0040F4C5	CALL 01680000	
0040F7AF	CALL 01680000	
0040F7DA	CALL 01680000	
0040F7F3	CALL 01680000	
0040F7FD	CALL 01680000	
0040F834	CALL 01680000	
0040F8BF	CALL 01680000	
0040F8D7	CALL 01680000	
0040F92F	CALL 01680000	
0040F967	CALL 01680000	
0040FBF4	CALL 01680000	
004100C4	CALL 01680000	
00410134	CALL 01680000	
00410196	CALL 01680000	
004101F6	CALL 01680000	
00417967	CALL 01680000	
00417F32	CALL 01680000	
0041A71F	CALL 01680000	
0041B797	CALL 01680000	
0041BB87	CALL 01680000	
0041BB9A	CALL 01680000	
0041C04A	CALL 01680000	
0041CD71	CALL 01680000	
0041DDBC	CALL 01680000	
0041DF0F	CALL 01680000	
0042081B	CALL 01680000	
00423310	CALL 01680000	
00423E5C	CALL 01680000	
00425306	CALL 01680000	
0042535E	CALL 01680000	
00425B2B	CALL 01680000	
00425B3F	CALL 01680000	
00425B53	CALL 01680000	
00425BF1	CALL 01680000	
00425C9C	CALL 01680000	
004272D5	CALL 01680000	
004272F6	CALL 01680000	
00427E07	CALL 01680000	
004280BB	CALL 01680000	
0042828D	CALL 01680000	

می بینید؟...این همه از این دستورها داریم که باید همه آنها را فیکس کنیم...همانطور که می بینید آنیک کردن این فایل اصلا کار سختی نیست...فقط وقت زیادی را از ما می گیرد.بذارید یک تابع دیگر را هم فیکس کنیم،اولین تابع که در خط 401112 قرار دارد را انتخاب می کنیم.بر روی آن کلیک راست کرده و گزینه new origin here را بزنید و بعد کلید F9 را بزنید(حتما باید توی اون خطی که قبلا بهتون گفته بودم،Hardware breakpoint،اون Memory BP را هم باید پاک کنید) :

Registers (FPU)	
EAX	7C80B6A1 kernel32.GetModuleHandleA
ECX	0012FDB0
EDX	00000000
EBX	00000087
ESP	0012FDBC
EBP	0012FDF0
ESI	00F2F27C
EDI	0B00F7F8
EIP	01006BC0
C 1	ES 0023 32bit 0(FFFFFFFF)
P 0	CS 001B 32bit 0(FFFFFFFF)
A 1	SS 0023 32bit 0(FFFFFFFF)
Z 0	DS 0023 32bit 0(FFFFFFFF)
S 0	FS 003B 32bit 7FFDF000(FFF)
T 0	GS 0000 NULL

باز هم مثل قبل مقدار رجیستر EAX تابع اصلی ماست.پس این خط را هم اینگونه فیکس می کنیم:

00401112	CALL 004221F0	
00401118	POP ECX	
0040111C	TEST EAX,EAX	
00401120	POP ECX	
00401124	JNZ SHORT 00401143	
00401126	PUSH EBX	
00401128	CALL DWORD PTR DS:[460B9C]	kernel32.GetModuleHandleA
0040112A	MOV DWORD PTR SS:[EBP-8],EAX	
0040112C	MOVX EAX,BYTE PTR SS:[EBP-10C]	
0040112E	PUSH EAX	
00401130	CALL 004226A0	
00401132	TEST EAX,EAX	
00401134	POP ECX	
00401136	JE SHORT 0040114A	
00401138	PUSH EBX	

اگر بخواهیم تمام این Call ها را فیکس کنیم...دو سه ساعتی از ما وقت می گیرد ☹ پس بهتر است یک اسکریپت بنویسم(این اسکریپت به هیچ وجه یک اسکریپت برای Asproctect نیست،فقط برای اینکه کارم سریعتر برای این فایل پیش برود این اسکریپت را نوشتم) :

این اسکریپت تمامی دستورات CALL 1680000 را پیدا می کند و آنها را درست می کند. (البته باید اون Hardware Breakpoint را هم بگذارید ☺)

```
var a // یک متغیر تعریف می کنم
var b // یک متغیر دیگر
var c // یک متغیر دیگر
```

mov a, 401000 // مقدار متغیر a را برابر 401000 می گیرم، تا از ابتدا به دنبال دستور 1680000 Call باشیم

Loop:

findop a, #E8????? 01# // با این دستور به دنبال دستور Call 1680000 می گردم

cmp \$RESULT, 0 // آیا چنین دستوری پیدا کردم؟

je End // اگر پیدا نکردم، اسکریپت تمام می شود

mov eip, \$RESULT // این هم همانند گزینه new origin here می باشد

Mov a, \$RESULT // مقدار متغیر a را برابر با جایی که دستور 1680000 Call می دهم

Run // برنامه را اجرا می کنیم

Mov b, eax // مقدار b را برابر EAX میگیرم. چون اگر HW BP گذاشته باشید، در جایی متوقف میشویم که EAX برابر تابع اصلی است

Mov c, 460814 // متغیر C را برابر ابتدای IAT می گیرم

jmp Loop2 // به قسمت Loop2 می روم

Loop2: // در این قسمت به دنبال تابع در IAT می گردم

Cmp [c], b // آیا در این قسمت از IAT تابع اصلی وجود دارد؟

Je Loop3 // اگر دارد به قسمت Loop3 برو

Add c, 4 // در غیر این صورت به Dword بعدی IAT برو

Jmp Loop2 // این حلقه همچنان ادامه پیدا می کند

Loop3: // در این خط تابع اصلی را جایگزین CALL 1680000 می کنم

Mov [a], 15FF // با این خط و خط بعدی دستور CALL 1680000 را تغییر می دهم

Mov [a+2], c

Add a, 6 // به مقدار متغیر a شش واحد اضافه می کنم

Jmp loop // و این حلقه ادامه پیدا می کند

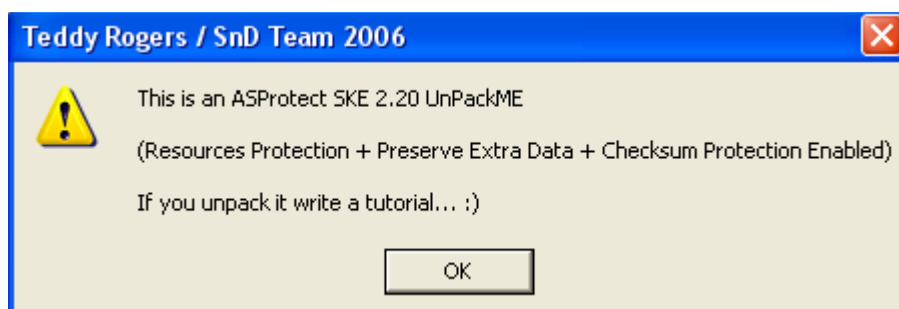
End: // وقتی تمامی CALL 1680000 ها درست شد، اسکریپت به این خط می رسد

Ret // اسکریپت تمام می شود.

دو تا تابع را فیکس کردیم، بقیه را با این اسکریپت درست می کنیم ☺
اسکریپت را اجرا کنید و اندکی منتظر بمانید تا:



حالا بر روی OEP یک New origin here بگذارید و بعد از فایل دوباره دامپ بگیرید و بعد فایل دامپ را با ImportREC فیکس کنید و فایل دامپ شده را اجرا کنید:



این بار فایل بی هیچ مشکلی اجرا می شود ☺

راه مل دیگر:

راه حل دیگر این است که از یک اسکریپت برای Asprotect که توسط VolX نوشته شده، استفاده کنیم. این اسکریپت بسیار هوشمند عمل کرده و خیلی خوب جواب می دهد ☺
 فایل پک شده را در OllyDBG باز کنید. و اسکریپت را اجرا کنید، تا هم OEP را پیدا کند. هم CALL ها را فیکس کند. هم از فایل دامپ بگیرد و خیلی کارهای دیگر انجام دهد ☺

004271AD	90	NOP
004271AE	90	NOP
004271AF	90	NOP
004271B0	55	PUSH EBP
004271B1	8BEC	MOV EBP,ESP
004271B3	6A FF	PUSH -1
004271B5	68 60E4500	PUSH 00450E60
004271B8	68 C8924200	PUSH 004292C8
004271BF	64:A1 00000000	MOV EAX,DWORD PTR FS:[0]
004271C5	50	PUSH EAX
004271C6	64:8925 00000000	MOV DWORD PTR FS:[0],ESP
004271CD	83C4 A8	ADD ESP,-58
004271D0	53	PUSH EBX

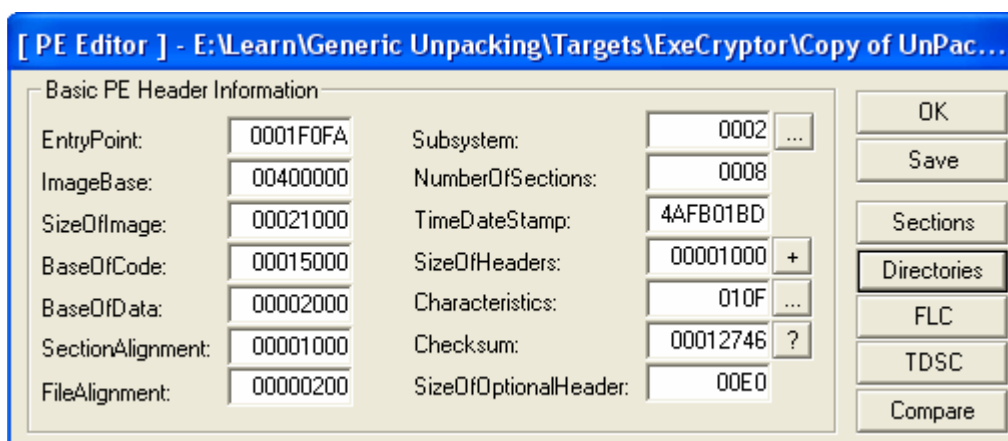
همانطور که می بینید اسکریپت در OEP متوقف شده است ☺
 خوب، اسکریپت از فایل دامپ هم گرفته و با نام de_UnpackMe.exe ذخیره کرده است. حالا تنها کاری که لازم است انجام دهید این است که با ImportREC این فایل دامپ را فیکس کنید ☺

ExeCryptor

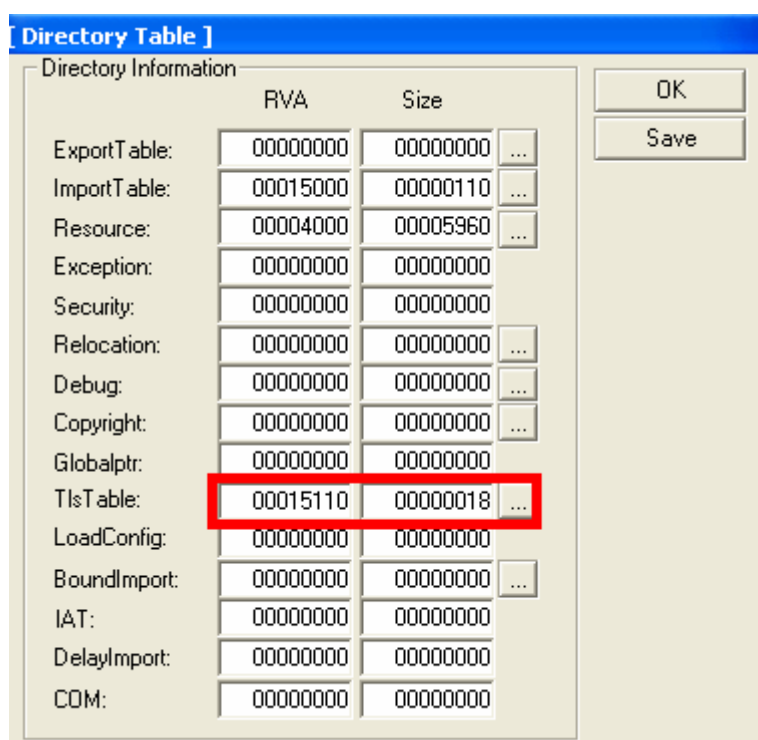
فایل مورد نظر را در OllyDBG باز کنید:



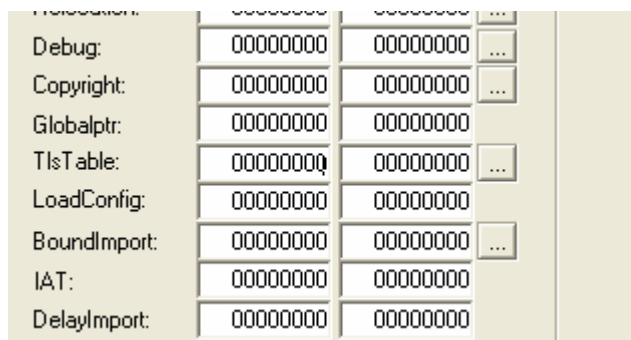
همانطور که می بینید با باز کردن فایل در OllyDBG پروسه بسته شده است...چه طور ممکنه؟...ببینید ما یک Function داریم به نام TLS Callback Function... TLS با یک برنامه می تواند برای هر Thread یک متغیر مخصوص به آن Thread بسازد. خوب، TLS Callback Function قبل از EntryPoint فایل اجرا می شود...یک پکر مثل ExeCryptor می آید و از TLS Callback Function نهایت سو استفاده را می کند، یعنی در این Function چند روش شناسایی آنتی دیباگ قرار می دهد. و اگر متوجه بشود که فایل در حال دیباگ شدن است، خودش را می بندد ② البته فقط این روش نیست...خیلی کارهای دیگر برای باز نشدن یا درست باز نشدن فایل در OllyDBG (یا هر دیباگر دیگری) می شود کرد. مثل PE Header Trick که قبلا توضیح دادم. خوب، حالا چه کار کنیم؟...یک راه حل این است که ما کلا TLS Callback Function را از بین ببریم، برنامه را در LordPE باز کنید:



گزینه Directories را بزنید:



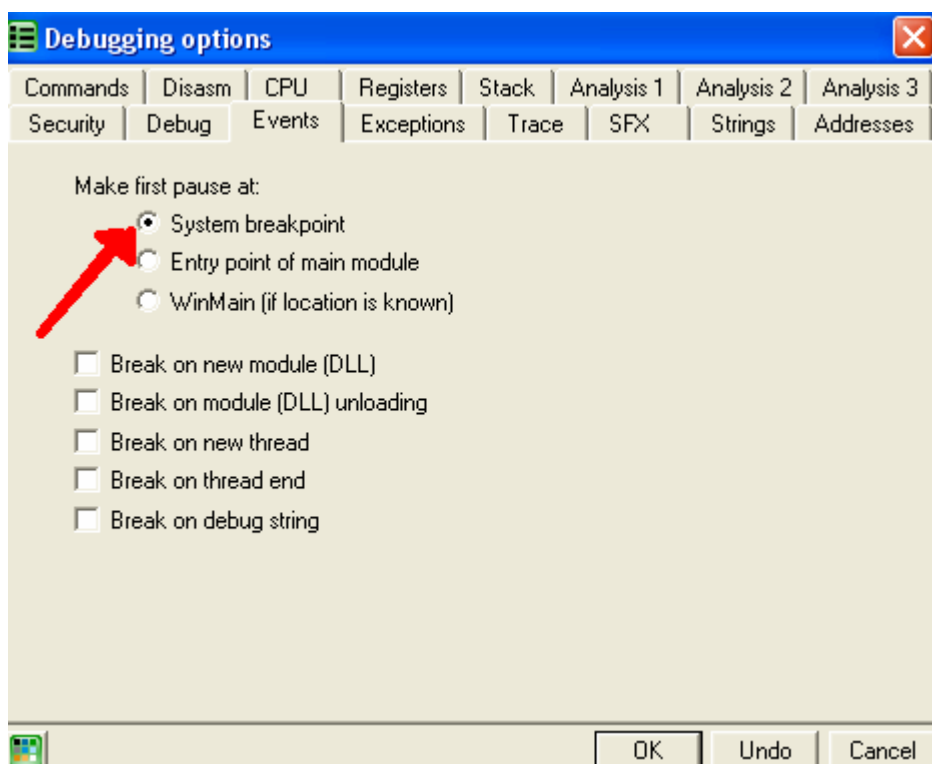
همانطور که می بینید مقدار TlsTable صفر نیست... یعنی این برنامه از TLS Callback Function استفاده می کند. ما اگر این دو را صفر کنیم. به طور کامل TLS Callback Function از بین می رود و برنامه در OllyDBG اجرا می شود. این دو مقدار را صفر کنید و گزینه Save را بزنید.



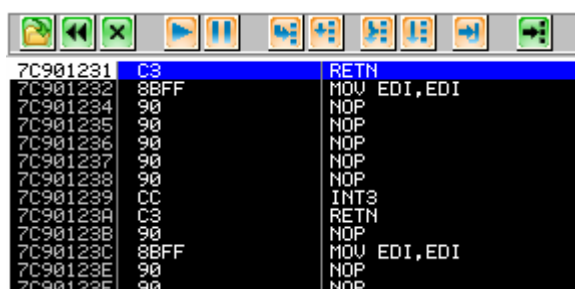
راه حل دیگر:

روش قبل برای از بین بردن TLS Callback Function روش خوبیست... اما همیشه جواب نمی دهد. چرا که ExeCryptor، قابلیت Integrity Check هم دارد. اگر ExeCryptor بفهمد که TLS Callback Function از بین رفته است، اجازه اجرای برنامه را نمی دهد. اول ما باید به OllyDBG بگوییم که به جای متوقف شدن در EntryPoint، در System breakpoint متوقف شود. یعنی قبل از جایی که TLS Callback Function اجرا می شود.

در منوی Options گزینه Debugging Options را بزنید. و وارد قسمت Events شوید و بعد گزینه System Breakpoint را تیک بزنید.



حالا دوباره فایل را در OllyDBG باز کنید:

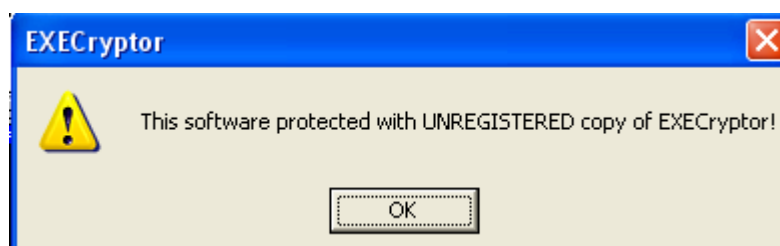


این بار در System BP متوقف شدیم نه در Entry point فایل، خوب برای رد کردن آنتی دیباگی که در TLS CallBack Function قرار دارد، در منوی View گزینه Breakpoints را بزنید:

Address	Module	Active	Disassembly
0041F0FA	UnPackMe	One-shot	CALL UnPackMe.0041EFF6

همانطور که می بینید ExeCryptor یک Breakpoint بر روی خودش گذاشته که با این Breakpoint وجود یا عدم وجود دیباگر را چک می کند. ما می توانیم با Delete کردن این Breakpoint قابلیت شناسایی دیباگر را از بین ببریم ☺ پس بر روی Breakpoint کلیک کنید و دکمه Delete را بزنید. با پاک کردن این Breakpoint فایل بی هیچ مشکلی در OllyDBG باز می شود.

خوب، برای یافتن OEP کلید F9 را بزنید، تا این پیام که مربوط به رجیستر نبودن برنامه است مواجه شوید:



این پیام موقعی نمایش داده شده است که کدهای سکشن اصلی برنامه کامل Decrypt شده است. پس می توانیم بر روی سکشن text، یک Memory Breakpoint بگذاریم. حالا کلید OK را بزنید تا این پیام برود و در اینجا که OEP ماست، متوقف شویم:

00401000	6A 30	PUSH 30	
00401002	68 00304000	PUSH UnPackMe.00403000	ASCII "Gooran_Ware"
00401007	68 00304000	PUSH UnPackMe.00403000	ASCII "This is an ExeCryptor (Unregistered Version) UnpackMe!<img alt='\"
0040100C	6A 00	PUSH 0	
0040100E	E8 07000000	CALL UnPackMe.0040101A	JMP to user32.MessageBoxA
00401013	6A 00	PUSH 0	
00401015	E9 06000000	CALL UnPackMe.00401020	JMP to kernel32.ExitProcess
0040101A	FF25 00204000	JMP DWORD PTR DS:[402000]	user32.MessageBoxA
00401020	FF25 00204000	JMP DWORD PTR DS:[402000]	kernel32.ExitProcess
00401026	0000	ADD BYTE PTR DS:[EAX],AL	
00401028	0000	ADD BYTE PTR DS:[EAX],AL	
0040102A	0000	ADD BYTE PTR DS:[EAX],AL	
0040102C	0000	ADD BYTE PTR DS:[EAX],AL	
0040102E	0000	ADD BYTE PTR DS:[EAX],AL	
00401030	0000	ADD BYTE PTR DS:[EAX],AL	

همانطور که می بینید برنامه ما در MASM نوشته شده و کدهای خیلی کمی دارد. بعد از این دیگر مشکلی نیست، دامپ می گیرید و فیکس می کنید ☺

PeSpin

برای یافتن OEP در PeSpin می توانیم از رجیستر ESP استفاده کنیم. دو بار کلید F8 را بزنید تا دستور PushAD اجرا شود و بعد بر روی مقدار رجیستر ESP یک Hardware Breakpoint on access بگذارید، و برنامه را با F9 اجرا کنید:

Address	Hex dump	Disassembly
0046CADA	F3:	PREFIX REP:
0046CADB	F7C2 08401401	TEST EDX,1144008
0046CAE1	EB 01	JMP SHORT 0046CAE4
0046CAE3	08F7	OR BH,DH
0046CAE5	C147 4F 28	ROL DWORD PTR DS:[EDI+4F],28
0046CAE9	95	XCHG EAX,EBP
0046CAEA	81E9 7DEDF1A3	SUB ECX,A3F1ED7D
0046CAF0	42	INC EDX
0046CAF1	8BD0	MOV EDX,EAX
0046CAF3	C7C1 269A0C67	MOV ECX,670C9A26
0046CAF9	F3:	PREFIX REP:
0046CAFA	0FC9	BSWAP ECX
0046CAFC	8BD0	MOV EDX,EAX
0046CAFE	0FBCC8	BSF ECX,EAX
0046CB01	89C1	MOV ECX,EAX
0046CB03	BA F9633585	MOV EDX,853563F9
0046CB08	D1C2	ROL EDX,1
0046CB0A	C7C1 0218798B	MOV ECX,8B7918D2
0046CB10	C7C1 389310DD	MOV ECX,DD1D9338
0046CB16	F3:	PREFIX REP:
0046CB17	89C2	MOV EDX,EAX
0046CB19	F3:	PREFIX REP:
0046CB1A	81C2 1442C7B7	ADD EDX,B7C74214
0046CB20	8D0D E5348F15	LEA ECX,DWORD PTR DS:[158F34E5]
0046CB26	8D0D C07CF980	LEA ECX,DWORD PTR DS:[80F97CC0]
0046CB2C	F7C2 8DE5E071	TEST EDX,71E0E58D
0046CB32	F7C2 90707F7A	TEST EDX,7A7F7090
0046CB38	41	INC ECX
0046CB39	D3E1	SHL ECX,CL
0046CB3B	0BD0	OR EDX,EAX
0046CB3D	0FC1D2	XADD EDX,EDX

یکبار دیگر F9 را بزنید:

Address	Hex dump	Disassembly
004271B0	> 55	PUSH EBP
004271B1	8BEC	MOV EBP,ESP
004271B3	6A FF	PUSH -1
004271B5	E9 BE8FFDFF	JMP 00400173
004271BA	E9 C48FFDFF	JMP 00400163
004271BF	64:A1 00000000	MOV EAX,DWORD PTR FS:[0]
004271C5	50	PUSH EAX
004271C6	64:8925 00000000	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 SS:[EBP-18],ESP
004271D6	FF15 ECF54600	CALL DWORD PTR DS:[46F5EC]
004271DC	33D2	XOR EDX,EDX
004271DE	8AD4	MOV DL,AH
004271E0	8915 34E64500	MOV DWORD PTR DS:[45E634],EDX
004271E6	8BC8	MOV ECX,EAX
004271E8	81E1 FF000000	AND ECX,0FF
004271EE	890D 30E64500	MOV DWORD PTR DS:[45E630],ECX
004271F4	C1E1 08	SHL ECX,8
004271F7	03CA	ADD ECX,EDX

اینبار به OEP رسیدیم. به پرشی که چند خط زیر OEP قرار دارد (JMP 00400178)، با دقت نگاه کنید. ببینید در این پرش چه اتفاقی می افتد... بر روی آن کلید Enter را بزنید تا ببینیم این پرش به کجا می رود؟

Address	Hex dump	Disassembly
00400178	68 600E4500	PUSH 00450E60
0040017D	E9 38700200	JMP 00402715A
00400182	66:68 C892	PUSH 92C8
00400186	42	INC EDX
00400187	00E9	ADD CL,CH
00400189	3270 02	XOR DH,BYTE PTR DS:[EAX+2]
0040018C	0028	ADD BYTE PTR DS:[EAX],CH
0040018E	E9 0D920200	JMP 004293A3
00400193	38E9	CMP CL,CH
00400195	C7	???
00400196	71 02	JNO SHORT 0040019A
00400198	008A E951A002	ADD BYTE PTR DS:[EDX+2A051E9],CL
0040019E	0070 E9	ADD BYTE PTR DS:[EAX-17],DH
004001A1	BB 7102003A	MOV EBX,3A000271

خوب، در اینجا دستور Push 00450E60 اجرا می شود، حالا بر روی پرشی که زیر 00450E60 قرار دارد، کلید Enter را بزنید.

004271B1	. 8BEC	MOV EBP,ESP
004271B3	. 6A FF	PUSH -1
004271B5	.- E9 BE8FFDFF	JMP 00400173
004271B8	.- E9 C48FFDFF	JMP 00400183
004271BF	. 64:A1 00000000	MOV EAX,DWORD PTR FS:[0]
004271C5	. 50	PUSH EAX
004271C6	. 64:8925 00000000	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 SS:[EBP-18],ESP
004271D6	. FF15 ECF54600	CALL DWORD PTR DS:[46F5EC]
004271DC	. 33D2	XOR EDX,EDX
004271DE	. 8AD4	MOV DL,AH
004271E0	. 8915 34E64500	MOV DWORD PTR DS:[45E634],EDX
004271E6	. 8BC8	MOV ECX,EAX
004271E8	. 81E1 FF000000	AND ECX,0FF
004271EE	. 890D 30E64500	MOV DWORD PTR DS:[45E630],ECX
004271F4	. C1E1 08	SHL ECX,8
004271F7	. 03CA	ADD ECX,EDX
004271F9	. 890D 2CE64500	MOV DWORD PTR DS:[45E62C],ECX
004271FF	. C1E8 10	SHR EAX,10
00427202	. A3 28E64500	MOV DWORD PTR DS:[45E628],EAX
00427207	. E8 828FFDFF	CALL 0040018E
0042720C	. 85C0	TEST EAX,EAX
0042720E	. 75 0A	JNZ SHORT 0042721A

می بینید؟...دوباره به نزدیکی های OEP برگشتیم...این یعنی چی؟...یعنی پکر ما تعدادی از کدهای برنامه را منتقل کرده به یک آدرس دیگر(کدها را به PE Header منتقل کرده است)...به این قابلیت پکرها می گویند: Code Redirection...خوب،حالا ما چه کار کنیم؟...ما باید تمام این Code Redirection ها را درست کنیم ☺ مثلا پرشی که در خط 4271B5 بود...همانطور که دیدید،دستور Push 00450E60 را اجرا می کرد و بعد به سر جای خودش برمی گشت.پس این دستور در واقع Push 00450E60 است:

004271AE	90	NOP
004271AF	90	NOP
004271B0	> 55	PUSH EBP
004271B1	. 8BEC	MOV EBP,ESP
004271B3	. 6A FF	PUSH -1
004271B5	. 68 600E4500	PUSH 00450E60
004271B8	.- E9 C48FFDFF	JMP 00400183
004271BF	. 64:A1 00000000	MOV EAX,DWORD PTR FS:[0]
004271C5	. 50	PUSH EAX
004271C6	. 64:8925 00000000	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 SS:[EBP-18],ESP
004271D6	. FF15 ECF54600	CALL DWORD PTR DS:[46F5EC]
004271DC	. 33D2	XOR EDX,EDX
004271DE	. 8AD4	MOV DL,AH

PeSpin فقط با پرش (JMP) کدها را Redirect نمی کند،بلکه با CALL هم این کار را می کند.به خط 427207 نگاه کنید:

. 03CA	ADD ECX,EDX
. 890D 2CE64500	MOV DWORD PTR DS:[45E62C],ECX
. C1E8 10	SHR EAX,10
. A3 28E64500	MOV DWORD PTR DS:[45E628],EAX
. E8 828FFDFF	CALL 0040018E
. 85C0	TEST EAX,EAX
. 75 0A	JNZ SHORT 0042721A
. 6A 1C	PUSH 1C
. E8 7D8FFDFF	CALL 00400194
. 83C4 04	ADD ESP,4
. E8 7B8FFDFF	CALL 0040019A
. 85C0	TEST EAX,EAX
. 75 0A	JNZ SHORT 0042722D
. 6A 10	PUSH 10
. E8 768FFDFF	CALL 004001A0
. 83C4 04	ADD ESP,4
. C745 FC 00000000	MOV DWORD PTR SS:[EBP-4],0

خوب،بر روی آن کلید Enter را بزنید:

Address	Hex dump	Disassembly
0040018E	.- E9 0D920200	JMP 004293A0
00400193	38E9	CMPL,CH
00400195	C7	???
00400196	71 02	JNO SHORT 0040019A
00400198	008A E951A002	ADD BYTE PTR DS:[EDX+2A051E9],
0040019E	0070 E9	ADD BYTE PTR DS:[EAX-17],DH
004001A1	BB 7102003A	MOV EBX,3A000271
004001A6	.- E9 159C0200	JMP 004293B0
004001AB	7C E9	JL SHORT 00400196
004001AD	9F	LAHF
004001AE	8102 006FE9C9	ADD DWORD PTR DS:[EDX],C9E96F
004001B4	04 03	ADD AL,3
004001B6	00F5	ADD CH,DH
004001B8	.- E9 F37B0200	JMP 004273B0

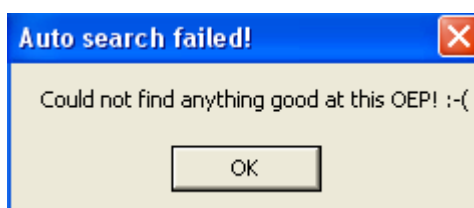
می بینید؟...این CALL به جای اینکه مستقیم به خط مورد نظرش برود.از یک پرش به عنوان واسطه استفاده می کند و این پرش در واقع جایی است که این CALL باید برود.خوب پس این CALL در واقع به خط 004293A0 می رود.پس این مورد را هم تصحیح کنید:

004271F4	. C1E1 08	SHL ECX,8
004271F7	. 03CA	ADD ECX,EDX
004271F9	. 890D 2CE64500	MOV DWORD PTR DS:[45E62C],ECX
004271FF	. C1E8 10	SHR EAX,10
00427202	. A3 28E64500	MOV DWORD PTR DS:[45E628],EAX
00427207	. E8 94210000	CALL 004293A0
0042720C	. 85C0	TEST EAX,EAX
0042720E	. 75 0A	JNZ SHORT 0042721A
00427210	. 6A 1C	PUSH 1C
00427212	. E8 7D8FFDFF	CALL 00400194
00427217	. 83C4 04	ADD ESP,4
0042721A	. E8 7B8FFDFF	CALL 0040019A
0042721F	. 85C0	TEST EAX,EAX
00427221	. 75 0A	JNZ SHORT 0042722D
00427223	. 6A 10	PUSH 10
00427225	. E8 768FFDFF	CALL 00400190

خوب، حالا باید از OEP (4271B0) شروع کنیم و تمام JUMP ها و CALL هایی که Redirect شده اند را درست کنیم ☺ می دانم که کار وقت گیری است. اما خوب، چاره ی دیگری نیست. جز اینکه بخوایم یک اسکریپت بنویسیم. خوب حالا از OEP شروع کنید، تا به اینجا برسید:

00428134	. 3D FF000000	CMP EAX,0FF
00428139	. 72 E9	JNB SHORT 00428124
0042813B	. 55	PUSH EBP
0042813C	. 892D 6CE74500	MOV DWORD PTR DS:[45E76C],EBP
00428142	. E8 79010000	CALL 004282C0
00428147	. 83C4 04	ADD ESP,4
0042814A	. A3 70E74500	MOV DWORD PTR DS:[45E770],EAX
0042814F	. EB 0C	JMP SHORT 0042815D
00428151	. 8935 6CE74500	MOV DWORD PTR DS:[45E76C],ESI
00428157	. 8935 70E74500	MOV DWORD PTR DS:[45E770],ESI
0042815D	. 33D2	XOR EDX,EDX
0042815F	. 6A 19	PUSH 19
00428161	. 8915 78E74500	MOV DWORD PTR DS:[45E778],EDX

این دیگر آخرین کدی بود که Redirect شده بود. حالا باید از فایل دامپ بگیریم. حالا ImportREC را باز کنید، OEP را بدهید و گزینه IAT Auto-search را بزنید:

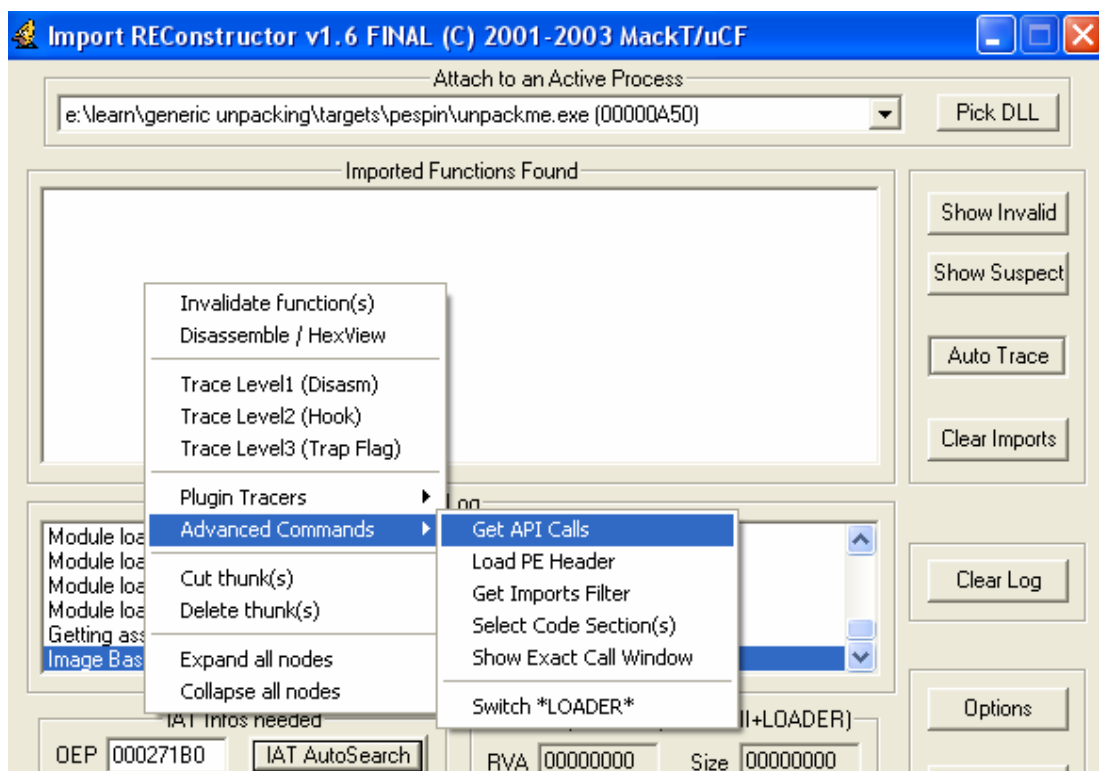


ImportREC نمی تواند IAT را پیدا کند ☺ پس بهتر است خودمان ببینیم چه بلایی سر IAT آمده؟ چند خط زیر OEP تابع GetVersion وجود دارد (خط 4271D6)... بر روی این خط کلیک راست کرده، گزینه Follow in dump و سپس گزینه Memory Address را بزنید:

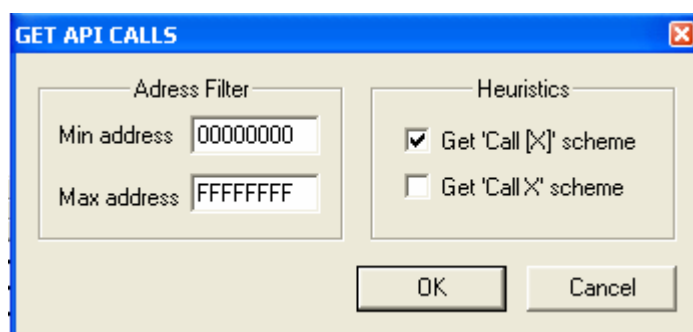
Address	Value	ASCI	Comment
0046F5EC	7C8111DA	r4u!	kernel32.GetVersion
0046F5F0	8097C600	.FuC	
0046F5F4	B219007C	!..%	
0046F5F8	AC007C85	ai.%	
0046F5FC	007C8217	%ai.	kernel32.DeleteFileA
0046F600	7C831EAB	%ai.	
0046F604	8286EE00	.eae	
0046F608	3F79007C	!..?	
0046F60C	03007C87	q!..	
0046F610	007C81DC	ui!	
0046F614	7C871321	!!q!	kernel32.AllocConsole
0046F618	81B58B00	.!ui	
0046F61C	0A49007C	!..I.	
0046F620	20007C87	q!..	
0046F624	007C8099	0q!..	
0046F628	7C80929C	aeq!	kernel32.GetTickCount
0046F62C	870A7100	.q.q	
0046F630	2442007C	!..B\$	
0046F634	2B007C80	q!..+	
0046F638	007C81BC	ui!	
0046F63C	7C873009	.0q!	kernel32.FillConsoleOutputCharac
0046F640	87305400	.T0q	
0046F644	3C1A007C	!..+<	
0046F648	EE007C87	q!..e	
0046F64C	007C8361	ai!	
0046F650	7C80BC69	i!q!	kernel32.SizeofResource
0046F654	87349D00	.4q	
0046F658	34E3007C	!..T4	
0046F65C	2C007C87	q!..	
0046F660	007C873B	q!..	
0046F664	7C801D77	w#q!	kernel32.LoadLibraryA
0046F668	80ADA000	.aiC	
0046F66C	ABDE007C	!..%	
0046F670	39007C80	q!..9	
0046F674	007C812F	ui!	
0046F678	7C81CF25	%ui	kernel32.WriteConsoleA
0046F67C	871A9D00	.!C	

Command:

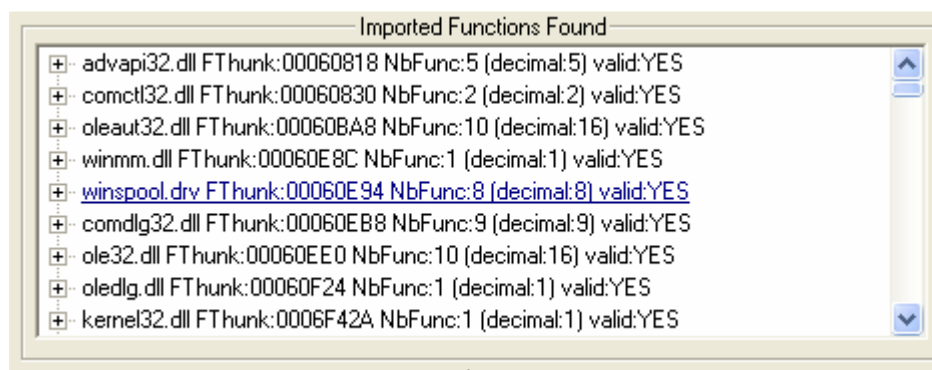
همانطور که می بینید PeSpin، IAT را کامل به هم ریخته و آدرس توابع را با فاصله نوشته است. حالا چه کار کنیم؟... ما می توانیم به ImportREC بگوییم که تمام Call هایی که به توابع API می روند را در برنامه پیدا کند و به این ترتیب آدرس تمامی توابع API را پیدا کند. بر روی ImportREC کلیک راست کنید، گزینه Advanced commands و سپس گزینه Get API Calls را بزنید:



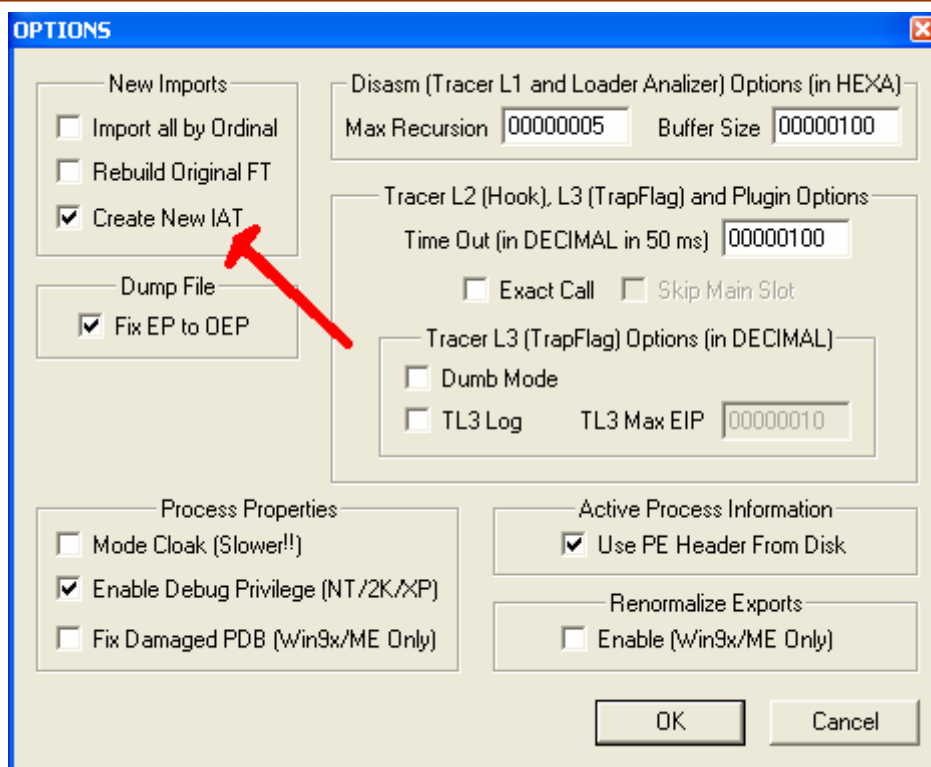
و بعد کلید OK را بزنید:



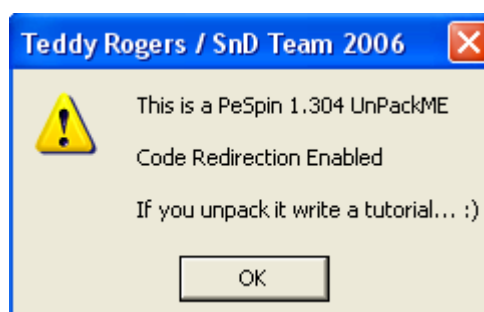
حالا گزینه Show Invalid را بزنید و بر روی توابع Invalid کلیک راست کرده و گزینه Cut Thunk را بزنید:



حالا گزینه Options را بزنید و مطمئن شوید که گزینه Create New IAT تیک دارد. (چون PeSpin، IAT را کامل به هم ریخته، ما باید یک سکشن جدید بسازیم و در آن سکشن Import ها را قرار دهیم):

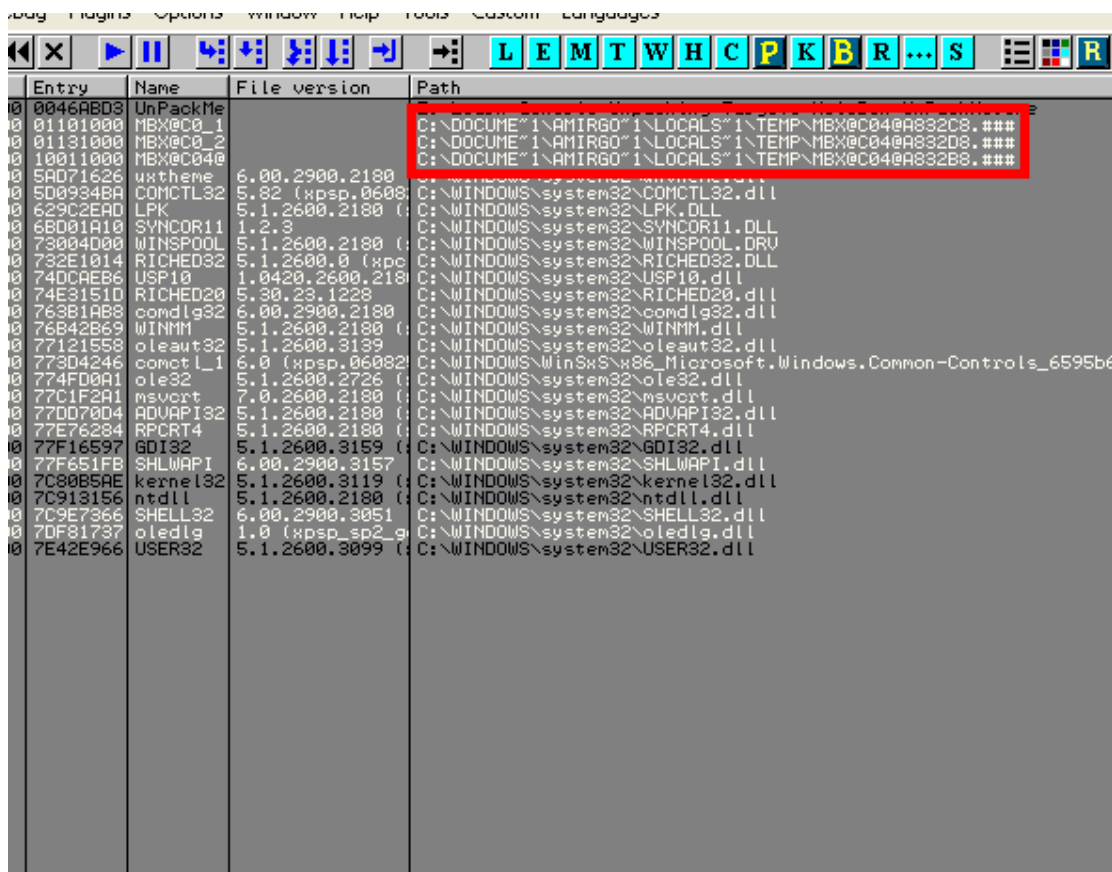


خوب، حالا فایل دامپ خود را فیکس کنید :



MoleBox

MoleBox یکی دیگر از پکرهایی است که قابلیت Bundle کردن فایل ها را دارد. در این مثال ما باید علاوه بر آپک کردن فایل Exe، فایل های DLL را هم از آن استخراج کنیم... حالا از کجا بفهمیم چند فایل DLL در درون این فایل قرار دارد؟
فایل مورد نظر را در OllyDBG باز کنید، کلید F9 را بزنید تا برنامه اجرا شود، حالا در منوی View گزینه Executable modules را بزنید:



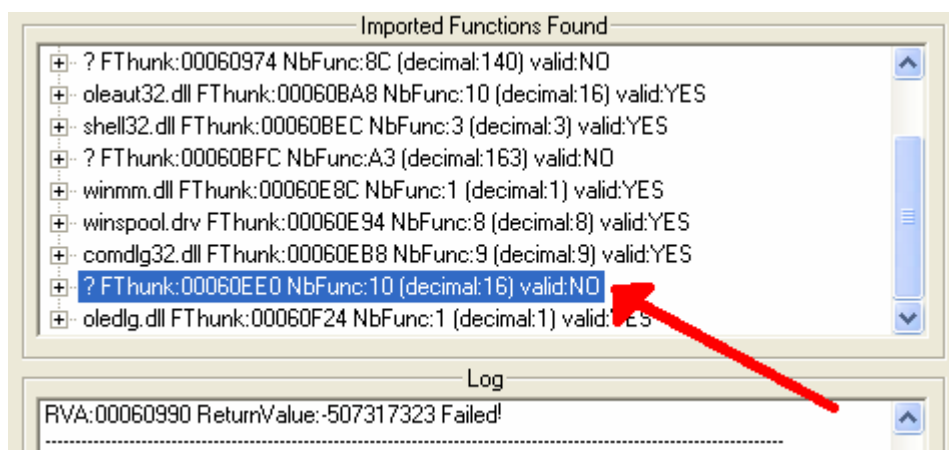
همانطور که می بینید، سه فایل DLL در TEMP قرار دارد که این فایل ها از داخل خود فایل استخراج شده و Load شده اند. یافتن OEP بسیار راحت است، به راحتی می توانیم از رجیستر ESP برای یافتن OEP استفاده کنیم ☺ دو بار کلید F8 را بزنید تا مقدار رجیستر ESP تغییر کند، بعد بر روی آن یک Hardware bp on access بگذارید و برنامه را با F9 اجرا کنید:

Address	Hex dump	Disassembly
0046A7B1	. 58	POP EAX
0046A7B2	. 58	POP EAX
0046A7B3	. FFD0	CALL EAX
0046A7B5	. E8 05CB0000	CALL 004772BF
0046A7BA	. CC	INT3
0046A7BB	. CC	INT3
0046A7BC	. CC	INT3
0046A7BD	. CC	INT3
0046A7BE	. CC	INT3
0046A7BF	. CC	INT3
0046A7C0	. 0000	ADD BYTE PTR DS:[EAX],AL
0046A7C2	. 0000	ADD BYTE PTR DS:[EAX],AL
0046A7C4	. 90	NOP

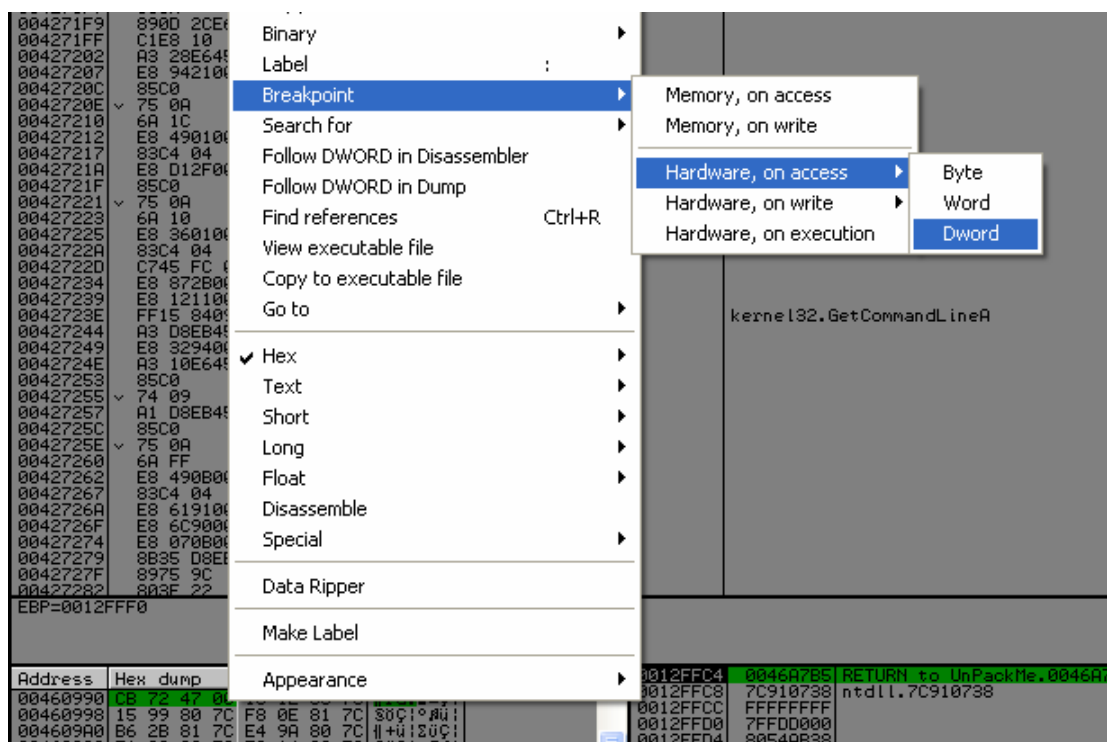
دستور CALL EAX که چند خط پایینتر قرار دارد، ما را به OEP می برد. پس سه بار F7 بزنید تا به OEP برسید:

Address	Hex dump	Disassembly
004271B0	55	PUSH EBP
004271B1	8BEC	MOV EBP,ESP
004271B3	6A FF	PUSH -1
004271B5	68 600E4500	PUSH 00450E60
004271B8	68 C8924200	PUSH 004292C8
004271BF	64:A1 00000000	MOV EAX,DWORD PTR FS:[0]
004271C5	50	PUSH EAX
004271C6	64:8925 00000000	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 SS:[EBP-18],ESP
004271D6	FF15 DC0A4600	CALL DWORD PTR DS:[460ADC]
004271DC	33D2	XOR EDX,EDX
004271DE	8AD4	MOV DL,AH
004271E0	8915 34E64500	MOV DWORD PTR DS:[45E634],EDX
004271E6	8BC8	MOV ECX,EAX

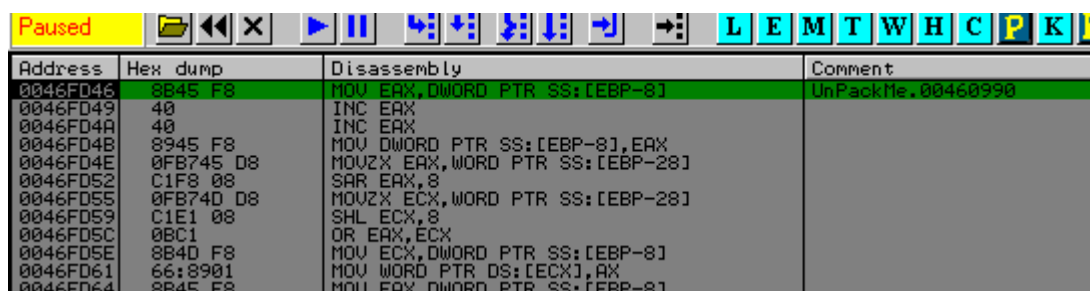
از فایل دامپ بگیرید و ImportREC را باز کنید:



همانطور که می بینید، چند تابع Invalid داریم ☹ برای یافتن این توابع باید در داخل کدهای پکر بگردیم و ببینیم کجا این توابع دستکاری می شود و ما آنجا را اصلاح کنیم (یافتن Magic Jump)
بر روی آدرس یکی از توابع Invalid (مثلا خط ۴۶۰۹۹۰) یک Hardware bp on write بگذارید و برنامه را ری استارت کنید:



و اجرا کنید:



هنوز به جایی که توابع Redirect می شوند، نرسیدیم. (می توانید به خط ۴۶۰۹۹۰ در دامپ نگاه کنید، می بینید که در این آدرس هنوز تابع Redirect شده نوشته نشده است).
سه بار دیگر F9 بزنید:

001 02	MOV ECX,2	
0B55 EC	MOV EDX,DWORD PTR SS:[EBP-14]	
02	PUSH EDX	
0F15 28E74700	CALL DWORD PTR DS:[47E728]	kernel32.GetProcAddress
0B4D E0	MOV ECX,DWORD PTR SS:[EBP-20]	
0901	MOV DWORD PTR DS:[ECX],EAX	
0B 2C	JMP SHORT 00471E36	
0B55 F4	MOV EDX,DWORD PTR SS:[EBP-C]	
0B02	MOV EAX,DWORD PTR DS:[EDX]	
05 FFFF0000	AND EAX,0FFFF	
0945 D0	MOV DWORD PTR SS:[EBP-30],EAX	
0B4D D0	MOV ECX,DWORD PTR SS:[EBP-30]	
01	PUSH ECX	
0B55 EC	MOV EDX,DWORD PTR SS:[EBP-14]	
02	PUSH EDX	
0F15 28E74700	CALL DWORD PTR DS:[47E728]	kernel32.GetProcAddress
0945 D4	MOV DWORD PTR SS:[EBP-2C],EAX	
037D D4 00	CMP DWORD PTR SS:[EBP-2C],0	
04 08	JE SHORT 00471E36	
0B45 E0	MOV EAX,DWORD PTR SS:[EBP-20]	
0B4D D4	MOV ECX,DWORD PTR SS:[EBP-2C]	
0908	MOV DWORD PTR DS:[EAX],ECX	
0B55 F0	MOV EDX,DWORD PTR SS:[EBP-10]	
01E2 FF000000	AND EDX,0FF	
05D2	TEST EDX,EDX	

الان اگر نگاهی به مقدار رجیستر EAX بیندازید، متوجه می شوید که این آدرس از IAT در واقع تابع ExitProcess بوده، پس یکبار دیگر F9 بزنید:

Address	Hex dump	ASCII
00460990	CB 72 47 00 34 17 06 00	TrG.4
00460998	48 17 06 00 17 06 00	H.R
004609A0	60 17 06 00 AE 17 06 00	'n
004609A8	7C 17 06 00 8C 17 06 00	!i
004609B0	9C 16 06 00 AE 17 06 00	6.i

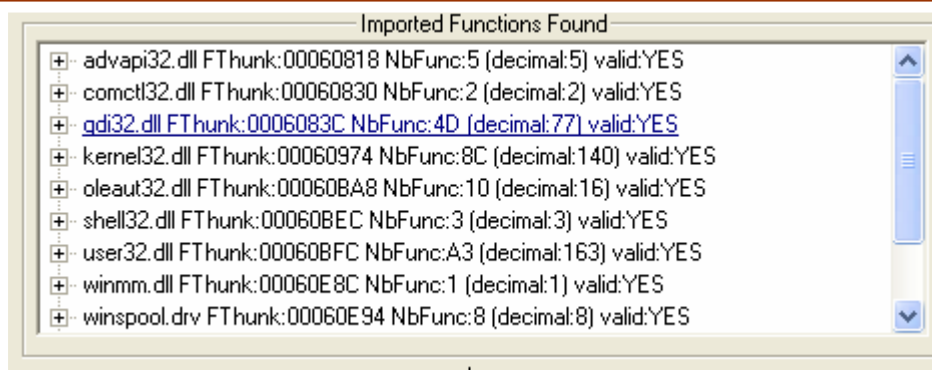
می بینید؟...تابع ExitProcess به خط 4772CB منتقل شده ☹ نگاهی به پنجره Disassembler هم بندازید:

000	TEST EAX,EAX	
05 0A	JNZ SHORT 0047258A	
09 0B0000EF	MOV ECX,EF00000B	
03 9D2F0000	CALL 00475527	
0B4D 08	MOV ECX,DWORD PTR SS:[EBP+8]	
0B55 F8	MOV EDX,DWORD PTR SS:[EBP-8]	
0B02	MOV EAX,DWORD PTR DS:[EDX]	
0901	MOV DWORD PTR DS:[ECX],EAX	
0B4D F4	LEA ECX,DWORD PTR SS:[EBP-C]	
0B55 F0	PUSH ECX	
02	MOV EDX,DWORD PTR SS:[EBP-10]	
04 04	PUSH EDX	
0945 08	PUSH 4	
01	MOV EAX,DWORD PTR SS:[EBP+8]	
0F15 ACE74700	PUSH EAX	
0945 FC 010000	CALL DWORD PTR DS:[47E7AC]	kernel32.VirtualProtect
0B45 FC	MOV DWORD PTR SS:[EBP-4],1	
0B55	MOV EAX,DWORD PTR SS:[EBP-4]	
01	MOV ESP,EBP	
03	POP EBP	
09	RET	
0BEC	PUSH EBP	
	MOV EBP,ESP	

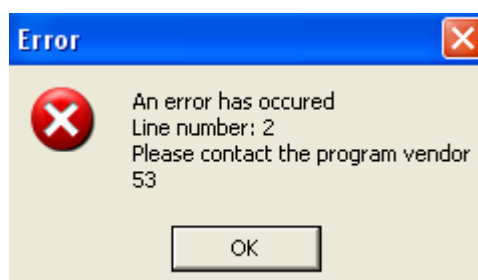
دستور MOV DWORD PTR DS:[ECX],EAX، توابع را Redirect می کند. پس بر روی این دستور یک Hardware BP بگذارید(تمامی Hardware bp های دیگر را پاک کنید.) و برنامه را دوباره اجرا کنید تا به اینجا برسید و بعد این خط را NOP کنید:

00472580	07 0B0000EF	MOV ECX,EF00000B
00472585	E8 9D2F0000	CALL 00475527
0047258A	8B4D 08	MOV ECX,DWORD PTR SS:[EBP+8]
0047258D	8B55 F8	MOV EDX,DWORD PTR SS:[EBP-8]
00472590	8B02	MOV EAX,DWORD PTR DS:[EDX]
00472592	90	NOP
00472593	90	NOP
00472594	8D4D F4	LEA ECX,DWORD PTR SS:[EBP-C]
00472597	51	PUSH ECX
00472598	8B55 F0	MOV EDX,DWORD PTR SS:[EBP-10]
0047259B	52	PUSH EDX
0047259C	6A 04	PUSH 4
0047259E	8B45 08	MOV EAX,DWORD PTR SS:[EBP+8]
004725A1	50	PUSH EAX
004725A2	FF15 ACE74700	CALL DWORD PTR DS:[47E7AC]
004725A8	C745 FC 010000	MOV DWORD PTR SS:[EBP-4],1
004725AF	8B45 FC	MOV EAX,DWORD PTR SS:[EBP-4]
004725B2	8BE5	MOV ESP,EBP
004725B4	5D	POP EBP
004725B5	C3	RET

حالا Hardware BP که در اینجا گذاشته بودید، پاک کنید. برنامه را اجرا کنید و دوباره ImportREC را باز کنید:

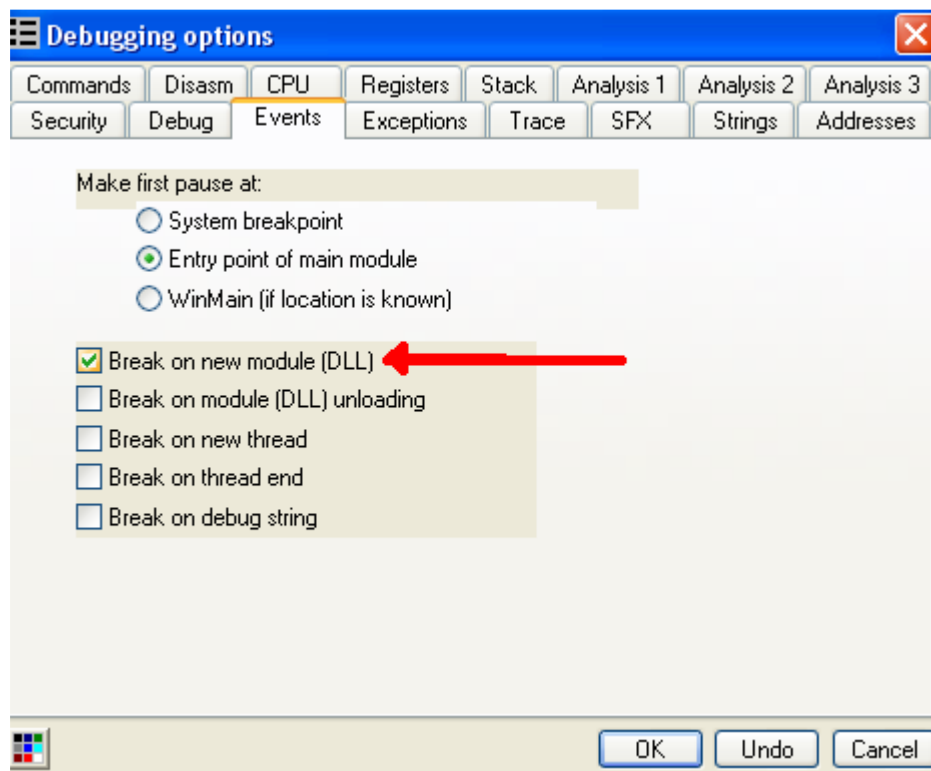


این بار تمامی توابع Valid هستند ☺ حالا فایل دامپ خود را فیکس کنید:



فایل آنپک شده است. اما به خاطر اینکه فایل های DLL را استخراج نکردیم و برنامه این فایل ها را ندارد، دچار خطا می شود. پس باید این فایل ها را استخراج کنیم.

برای استخراج DLL اول باید دستور CALL ی را پیدا کنیم که به OEP فایل های DLL می رود. برنامه را ری استارت کنید. کلیدهای ALT + O را بزنید و وارد قسمت Events شوید. حالا تیک گزینه Break on new module را بزنید:



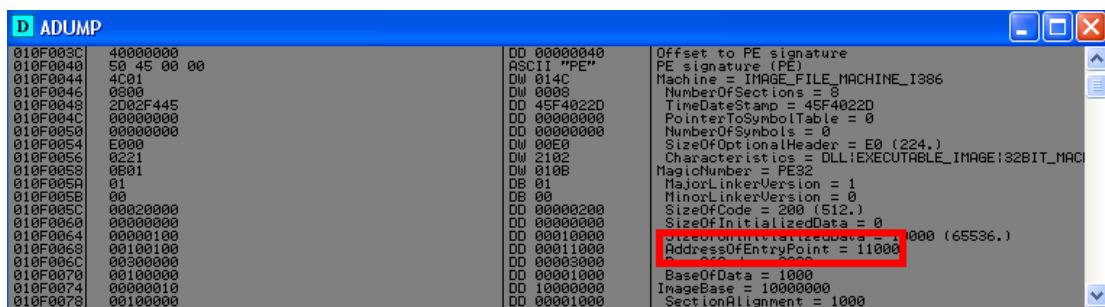
به این ترتیب اگر یک فایل DLL اجرا شود، OllyDBG متوقف می شود. چند بار کلید F9 را بزنید تا فایل های DLL که می خواهیم استخراج کنیم، اجرا شوند:

خوب، حالا باید Entry point فایل DLL را پیدا کنیم. کلیدهای ALT + M را بزنید تا وارد Memory Map شوید:

برای یافتن Entry Point می توانیم مشخصات PE Header فایل را در OllyDBG نگاه کنیم. برای دیدن PE Header یک فایل در Memory Map کافی است بر روی PE Header دو بار کلیک کنید. (در تصویر بالا بر روی قسمت سبز رنگ دو بار کلیک کنید).

همانطور که ما در PeTools ، LordPE یا PEID مشخصات PE Header فایل را می دیدیم، در OllyDBG هم می توانیم این مشخصات را ببینیم.

در این پنجره به پایین بروید تا به اینجا برسید:



پس Entry point فایل ما بر حسب RVA برابر 11000 است ☺
حالا کلیدهای ALT + C را بزنید تا وارد پنجره CPU شوید و کلیدهای CTRL + G بزنید و به Entry Point فایل بروید:



Address	Hex dump	Disassembly
10011000	58	POP EAX
10011001	68 DCC4BECA	PUSH CABEC4DC
10011006	68 08D0AE00	PUSH 0AED008
10011008	68 6C000000	PUSH 6C
10011010	50	PUSH EAX
10011011	68 372D4700	PUSH 472D37
10011016	C3	RET
10011017	90	NOP
10011018	90	NOP
10011019	90	NOP
1001101A	90	NOP
1001101B	90	NOP

اینجا Entry Point فایل ماست، به خط 10011011 نگاه کنید. دستور Push 472D37، مقدار 472D37 را وارد حافظه Stack می کند و با دستور RET به خط 472D37 می رود. (همانطور که قبلا گفتیم **Push xxx + RET = JMP xxx**)
پس بر روی خط 472D37 یک BP بگذارید و برنامه را اجرا کنید.

00472D28	8B45 E8	MOV EAX, DWORD PTR SS:[EBP-20]
00472D29	8B4D E8	MOV ECX, DWORD PTR SS:[EBP-18]
00472D2C	8908	MOV DWORD PTR DS:[EAX], ECX
00472D2E	EB 95	JMP SHORT 00472D35
00472D30	E9 50FFFFFF	JMP 00472D35
00472D35	C9	LEAVE
00472D36	C3	RET
00472D37	55	PUSH ESP
00472D38	8BEC	MOV EBP, ESP
00472D3A	6A FF	PUSH -1
00472D3C	68 98B44700	PUSH 0047B498
00472D41	68 4C954600	PUSH 0046954C
00472D46	64:A1 00000000	MOV EAX, DWORD PTR FS:[0]
00472D4C	50	PUSH EAX
00472D4D	64:8925 00000000	MOV DWORD PTR FS:[0], ESP
00472D54	51	PUSH ECX
00472D55	51	PUSH ECX
00472D56	83EC 7C	SUB ESP, 7C
00472D59	53	PUSH EBX
00472D5A	56	PUSH ESI
00472D5B	57	PUSH EDI
00472D5C	8965 E8	MOV DWORD PTR SS:[EBP-18], ESP
00472D5F	8365 E4 00	AND DWORD PTR SS:[EBP-1C], 0
00472D63	8365 FC 00	AND DWORD PTR SS:[EBP-4], 0

حالا بر روی RET بالای سر آن یک BP بگذارید و برنامه را اجرا کنید، و یک بار کلید F8 را بزنید:

00472E95	74 14	JE SHORT 00472EAB
00472E97	FF75 D4	PUSH DWORD PTR SS:[EBP-2C]
00472E9A	FF75 BC	PUSH DWORD PTR SS:[EBP-44]
00472E9D	FF75 C0	PUSH DWORD PTR SS:[EBP-40]
00472EA0	FF75 DC	PUSH DWORD PTR SS:[EBP-24]
00472EA3	E8 C2FDFFFF	CALL 00472C6A
00472EA8	83C4 10	ADD ESP,10
00472EAB	6A 5C	PUSH 5C
00472EAD	FF75 0C	PUSH DWORD PTR SS:[EBP+C]
00472EB0	E8 4B68FFFF	CALL 00469700
00472EB5	59	POP ECX
00472EB6	59	POP ECX
00472EB7	8945 E0	MOV DWORD PTR SS:[EBP-20],EAX
00472EBA	837D E0 00	CMP DWORD PTR SS:[EBP-20],0
00472EBE	75 08	JNZ SHORT 00472EC9
00472EC0	8B45 0C	MOV EAX,DWORD PTR SS:[EBP+C]
00472EC3	8945 E0	MOV DWORD PTR SS:[EBP-20],EAX
00472EC6	EB 07	JMP SHORT 00472ECF
00472EC8	8B45 E0	MOV EAX,DWORD PTR SS:[EBP-20]

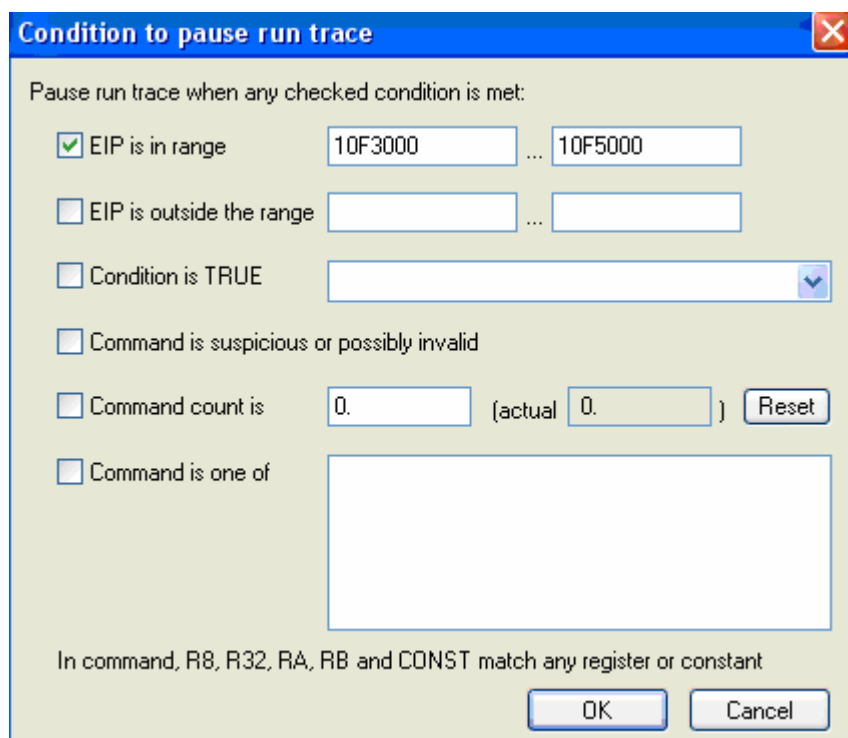
ما الان در کدهای پکر هستیم، حالا برای یافتن OEP می توانیم از Run Trace استفاده کنیم. Run Trace یکی از قابلیت های بسیار خوب OllyDBG است. OllyDBG با Run Trace با توجه به شرطی که شما داده اید، کدها را اجرا می کند و لیست تمامی خط هایی که اجرا شده را در منوی View، قسمت Run Trace نگهداری می کند. برای اجرای یک Run Trace ابتدا سایز و شروع سکشن اصلی این فایل DLL را از Memory Map بگیرید:

00FE0000	00023000	00FE0000	(itself)		
010E0000	00010000	010E0000	(itself)		
010F0000	00001000	MBX0B0_1	010F0000	(itself)	
010F1000	00001000	MBX0B0_1	010F0000		.idata
010F2000	00001000	MBX0B0_1	010F0000		.edata
010F3000	00002000	MBX0B0_1	010F0000		.text
010F5000	00010000	MBX0B0_1	010F0000		.text
010F6000	00009000	MBX0B0_1	010F0000		.data
010FF000	00000000	MBX0B0_1	010F0000		.bss
01100000	00001000	MBX0B0_1	010F0000		IMPORTS
					.reloc

همانطور که می بینید محل شروع سکشن text. در این فایل DLL برابر 10F3000 است و سایز آن هم ۲۰۰۰ است. پس:

End of code section=Start + Size= 10F3000 + 2000 = 10F5000

البته ممکن است این مقادیر در سیستم شما فرق داشته باشد. در اینجا مقدار ImageBase (محل شروع PE Header) برابر 10F0000 است. این مقدار را به خاطر داشته باشید. چون بعدا به دردمان خواهد خورد. حالا برای اجرا Run Trace کلیدهای CTRL + T را بزنید و در قسمت EIP in range همانند تصویر بنویسید:



ما می خواهیم ببینیم که در چه موقعی وارد سکشن text. در این فایل DLL می شود. پس نوشتم هر وقت مقدار EIP بین 10F3000 و 10F5000 باشد. عملیات Trace تمام شود. کلید OK را بزنید و برای اجرای Trace کلیدهای CTRL + F11 را بزنید. (کلید CTRL + F11 همانند کلید F7 در Trace عمل می کند و کلید CTRL + F12 همانند کلید F8) خوب، بعد از مدتی که عملیات Tracing انجام شد، به اینجا می رسیم (برای دیدن لیست تمامی دستوراتی که اجرا شده، در منوی View گزینه Run Trace را بزنید) :

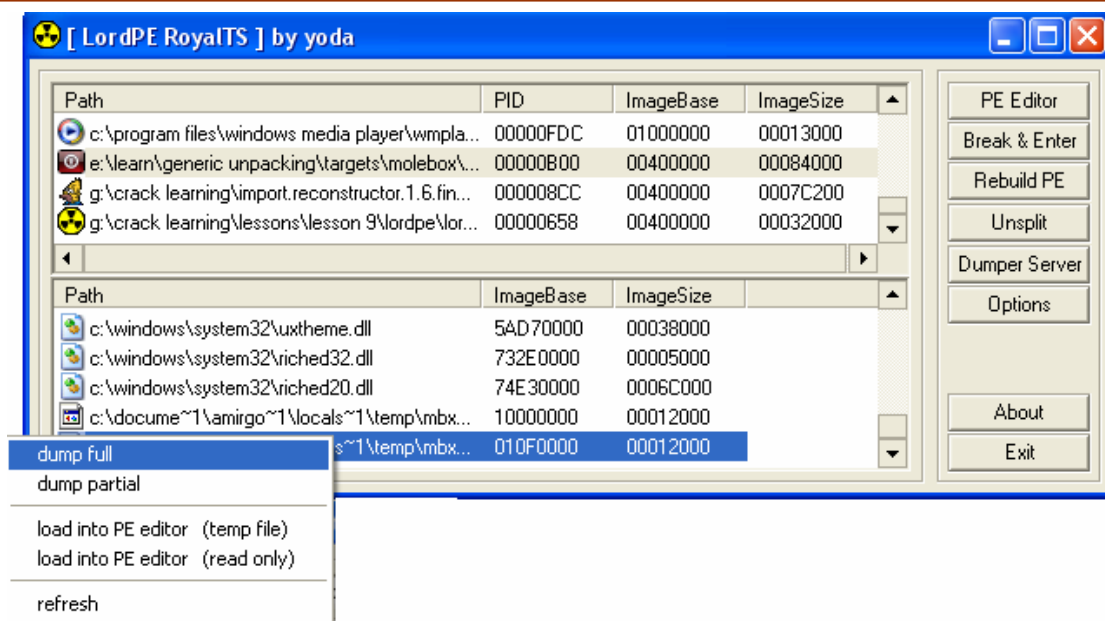
Address	Hex dump	Disassembly
010F3000	55	PUSH ESP
010F3001	89E5	MOV EBP,ESP
010F3003	8D55 0C	LEA EDI,DWORD PTR SS:[EBP+C]
010F3006	8B02	MOV EAX,DWORD PTR DS:[EDI]
010F3008	3D 01000000	CMP EAX,1
010F300D	75 19	JNZ SHORT 010F3028
010F300F	E8 F1040000	CALL 010F3505
010F3014	8D55 08	LEA EDI,DWORD PTR SS:[EBP+8]
010F3017	8B02	MOV EAX,DWORD PTR DS:[EDI]
010F3019	A3 07E00F01	MOV DWORD PTR DS:[10FE007],EAX
010F301E	B8 01000000	MOV EAX,1
010F3023	E9 11000000	JMP 010F3039
010F3028	3D 00000000	CMP EAX,0
010F302D	75 0A	JNZ SHORT 010F3039
010F302F	E8 510A0000	CALL 010F3A85
010F3034	E8 65050000	CALL 010F359E
010F3039	89EC	MOV ESP,EBP
010F303B	5D	POP EBP
010F303C	C2 0C00	RET 0C
010F303F	9B	WAIT
010F3040	DBE2	FCLEX
010F3042	D92D 00500F01	FLDCW WORD PTR DS:[10F5000]
010F3048	C3	RET
010F3049	C3	RET

اینجا OEP فایل DLL ماست. چون در MoleBox، OEP تمامی DLL ها توسط یک CALL فراخوانده می شود، ما اگر آن CALL را پیدا کنیم، می توانیم OEP سایر DLL ها را هم پیدا کنیم ☺
پس در پنجره Stack کلیک راست کرده و گزینه Follow in disassembler را بزنید تا بفهمیم از کجا به اینجا آمدیم:

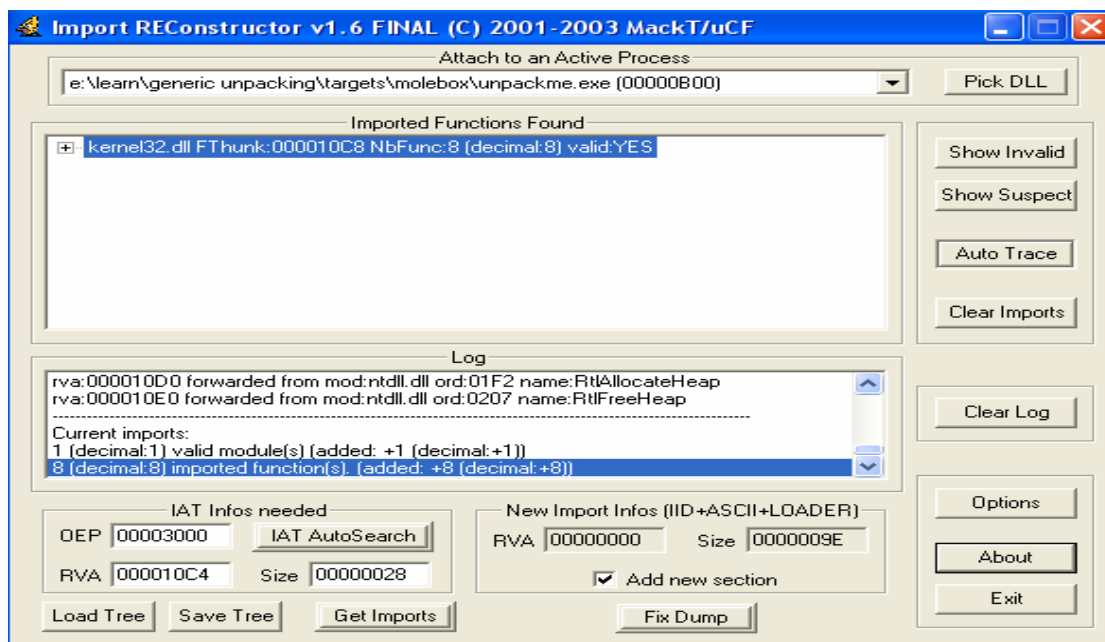
The screenshot shows a disassembler window with a context menu open. The menu options are: 'Go to EBP', 'Go to expression Ctrl+G', 'Follow in Disassembler Enter' (highlighted), 'Follow in Dump', and 'Appearance'. Below the menu, a list of instructions is visible. The instruction 'CALL DWORD PTR SS:[EBP-64]' is highlighted with a red box.

Address	Hex dump	Disassembly
012F340	00473112	RETURN to Unl
012F344	010F0000	ASCII "M2L"
012F348	00000001	
012F34C	00000000	
012F350	F4200001	
012F354	F360F400	
012F358	FFFF1000	
00473107	8B45 18	MOV EAX,DWORD PTR SS:[EBP+18]
0047310A	50	PUSH EAX
0047310B	8B45 14	MOV EAX,DWORD PTR SS:[EBP+14]
0047310E	50	PUSH EAX
0047310F	FF55 9C	CALL DWORD PTR SS:[EBP-64]
00473112	A3 28E94700	MOV ESP,DWORD PTR DS:[47E960]
00473117	66:61	POPAW
0047311F	8365 FC 00	AND DWORD PTR SS:[EBP-4],0
00473123	E8 05000000	CALL 0047312D
00473128	E9 46010000	JMP 00473223

پس این CALL که در خط 473112 قرار دارد، همان CALL ی است که به OEP تمامی DLL ها می رود. تمامی Breakpoint ها را پاک کنید (همچنین تیک گزینه Break on new module در قسمت تنظیمات را بردارید.) و فقط بر روی این دستور hardware bp on execution بگذارید، و برنامه را دوباره اجرا کنید.
کلا سه بار در این خط متوقف می شویم. که هر سه بار این دستور به OEP یکی از این فایل های DLL می رود. (هر سه فایل با LordPE (یا نرم افزارهای مشابه) دامپ بگیرید: OEP، DLL آنها برابر 3000 است. البته بر حسب RVA)



و با ImportREC فیکس کنید:



dumped_.exe



dumped_.dll

حالا فایل DLL آنپک شده را در LordPE باز کنید:

[PE Editor] - e:\learn\generic unpacking\targets\molebox\unpack\dumped_.dll

Basic PE Header Information

EntryPoint:	00003000	Subsystem:	0002 ...
ImageBase:	10000000	NumberOfSections:	0009
SizeOfImage:	00013000	TimeDateStamp:	45F4022D
BaseOfCode:	00003000	SizeOfHeaders:	00001000 ? +
BaseOfData:	00001000	Characteristics:	2102 ...
SectionAlignment:	00001000	Checksum:	00000000 ?
FileAlignment:	00001000	SizeOfOptionalHeader:	00E0
Magic:	010B	NumOfRvaAndSizes:	00000010 + -

OK
Save
Sections
Directories
FLC
TDSC
Compare
L

همانطور که می بینید، ImageBase فایل آنپک شده ما 1000000 است، در حالی که وقتی این فایل در Memory بوده، ImageBase آن برابر 10F0000 بوده است. پس این مورد را درست کنید:

[PE Editor] - e:\learn\generic unpacking\targets\molebox\unpack\dumped_.dll

Basic PE Header Information

EntryPoint:	00003000	Subsystem:	0002 ...
ImageBase:	010F0000	NumberOfSections:	0009
SizeOfImage:	00013000	TimeDateStamp:	45F4022D
BaseOfCode:	00003000	SizeOfHeaders:	00001000 ? +
BaseOfData:	00001000	Characteristics:	2102 ...
SectionAlignment:	00001000	Checksum:	00000000 ?
FileAlignment:	00001000	SizeOfOptionalHeader:	00E0
Magic:	010B	NumOfRvaAndSizes:	00000010 + -

OK
Save
Sections
Directories
FLC
TDSC
Compare
L

خوب، حالا باید بفهمیم نام واقعی این فایل چیست؟... برای این کار گزینه Directories را بزنید:

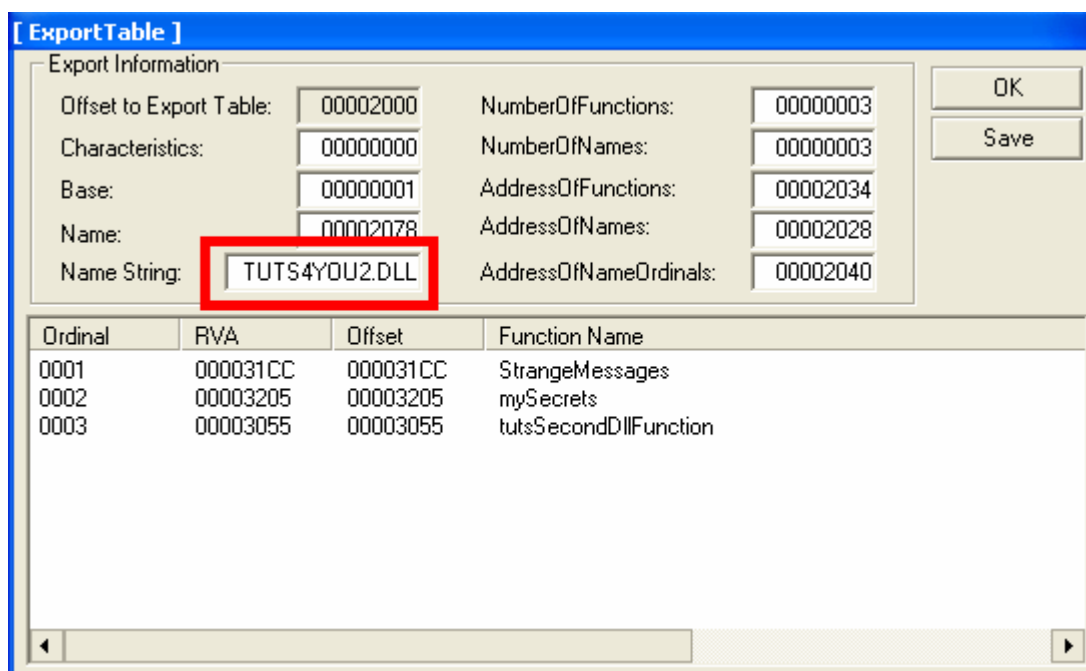
Directory Table]

Directory Information

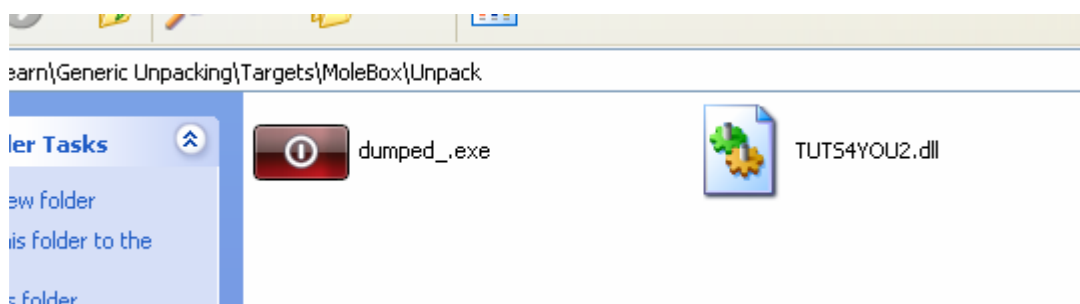
	RVA	Size			
ExportTable:	00002000	00000086	...	L	H
ImportTable:	00012000	00000014	...	L	H
Resource:	00000000	00000000	...	L	H
Exception:	00000000	00000000		L	H
Security:	00000000	00000000			H
Relocation:	00011020	00000008	...	L	H
Debug:	00000000	00000000	...	L	H
Copyright:	00000000	00000000	...	L	H
Globalptr:	00000000	00000000			
TlsTable:	00000000	00000000	...	L	H
LoadConfig:	00000000	00000000		L	H
BoundImport:	00000000	00000000	...	L	H
IAT:	00000000	00000000			H
DelayImport:	00000000	00000000		L	H
COM:	00000000	00000000	...	L	H
Reserved:	00000000	00000000			H

OK
Save

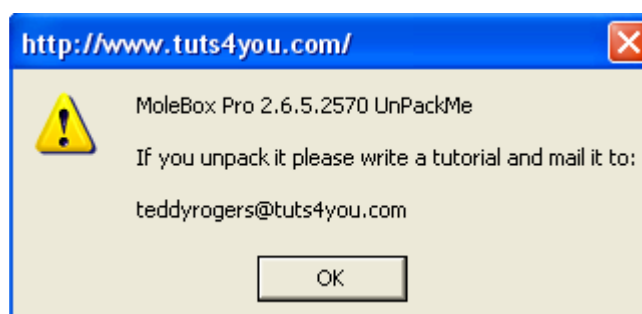
دکمه کنار Export Table را بزنید:



همانطور که می بینید، فایل های دیگر با نام TUTS4You2.dll این فایل را صدا می زنند، پس نام اصلی این فایل TUTS4YOU2.DLL است ☺ حالا تغییرات را در ذخیره کنید. و نام فایل را هم درست کنید:

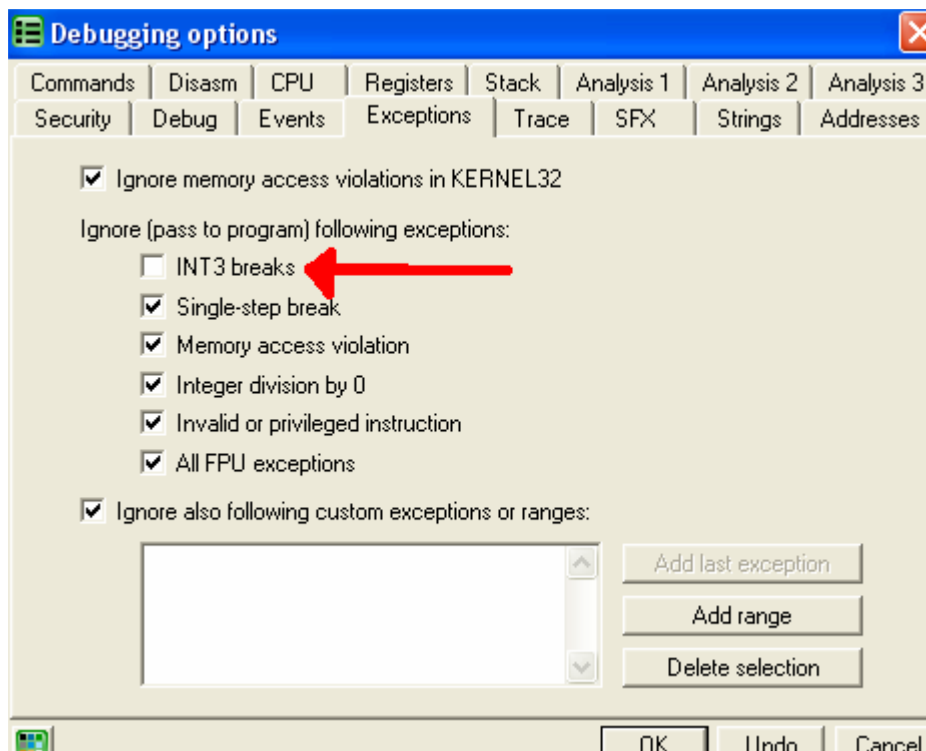


همانطور که می بینید یکی از DLL ها استخراج شده است. دو تای دیگر هم مثل همین است. البته دیگر لازم نیست که Run Trace اجرا کنیم. چون CALL ی که به OEP ها می رود را پیدا کردیم. حالا کافیه دو فایل DLL دیگر را هم به همین ترتیب استخراج کنیم تا فایل ما بی هیچ مشکلی اجرا شود ☺

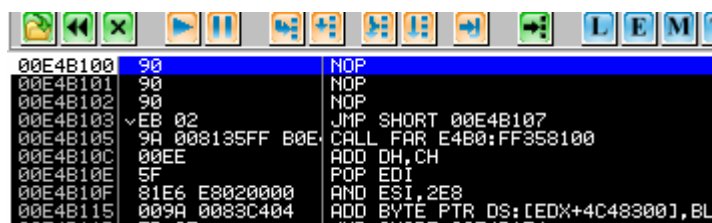


Enigma

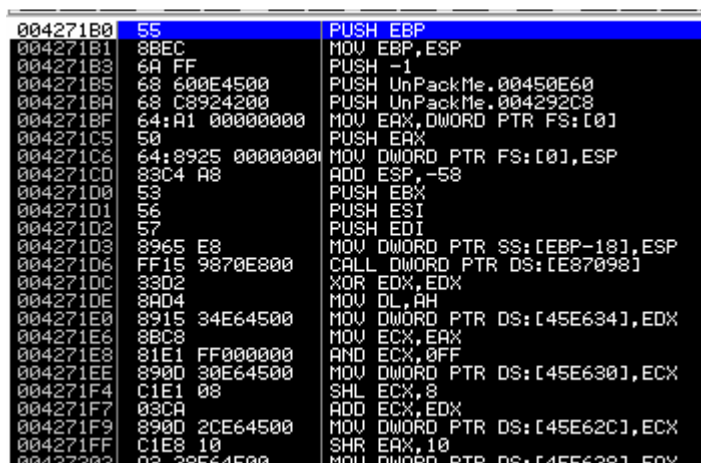
فایل مورد نظر را OllyDBG (ترجیحا در OllyIce باز کنید) باز کنید، برای یافتن OEP می توانیم از Memory Breakpoint استفاده کنیم. ابتدا کلیدهای ALT + O را بزنید و در قسمت Exceptions تیک گزینه INT3 Breaks را بردارید:



و بعد برنامه را با F9 اجرا کنید. (ممکن است برنامه مقداری کند اجرا شود. چون پکر برای کند کردن دیباگ برنامه، خطای Access violation on write را به وجود می آورد. در ضمن پلاگین HideDBG فراموش نشود، چون این پکر از چند تابع برای شناسایی دیباگر استفاده می کند.)
چند بار کلید F9 را بزنید تا به آخرین خطایی که در برنامه قبل از اجرا شدن اتفاق می افتد، برسید:



حالا بر روی سکشنی که محتوی Code است (اولین سکشن زیر PE Header) یک Memory breakpoint on access بگذارید و بعد برنامه را با F9 اجرا کنید:



خوب، به OEP رسیدیم ☺ حالا باید دامپ بگیریم و بعد ImportREC را باز می کنیم. OEP را بدهید و گزینه IAT Auto-search را بزنید. همانطور که می بینید ImportREC نتوانسته است چیزی پیدا کند ☹ پس بهتر است ببینیم چه بلایی سر IAT آمده است؟... بر روی یکی از CALL هایی که به تابع API می روند. (مثل خط 4271D6) کلیک راست کرده و گزینه Follow in Dump و سپس گزینه Memory Address را بزنید:

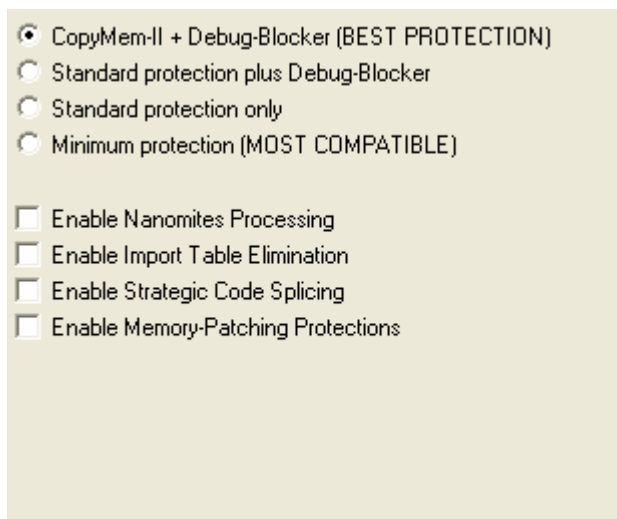
Address	Value	ASCII	Comment
00E87080	E783487C	!Hāγ	
00E87084	FE828600	.āē■	
00E87088	82307C80	Q!0ē	
00E8708C	CF8600E7	γ.āē	
00E87090	EC7C80B4	γ.āē	
00E87094	8600E782	ēγ.ā	
00E87098	7C81110A	γ!ū!	kernel32.GetVersion
00E8709C	00E78390	ēāγ.	
00E870A0	8097C686	āγ!C	
00E870A4	E783BC7C	!āγ	
00E870A8	B2198600	.ā!ā	
00E870AC	81A87C85	ā!ā	
00E870B0	AC8600E7	γ.āē	
00E870B4	0C7C8217	ēē!	
00E870B8	8600E783	ēγ.ā	
00E870BC	7C831EAB	γ!ā!	kernel32.DeleteFileA
00E870C0	00E7840C	.āγ.	
00E870C4	8286EE86	āēāē	
00E870C8	E784207C	!āγ	
00E870CC	3F798600	.āγ?	
00E870D0	84347C87	Q!4ā	
00E870D4	038600E7	γ.āē	
00E870D8	487C810C	■!H	
00E870DC	8600E784	ēγ.ā	
00E870E0	7C871321	γ!ā!	kernel32.AllocConsole
00E870E4	00E7846C	!āγ.	
00E870E8	81B58B86	ā!ā	
00E870EC	E784807C	!Cāγ	
00E870F0	0A498600	.āI.	
00E870F4	84947C87	Q!ā	
00E870F8	208600E7	γ.ā	
00E870FC	A87C8099	Q!ā	
00E87100	8600E784	ēγ.ā	
00E87104	7C80929C	āēQ!	kernel32.GetTickCount
00E87108	00E784E4	2āγ.	
00E8710C	870A7186	āq.γ	
00E87110	E784FC7C	!āγ	
00E87114	24428600	.āBē	
00E87118	85107C80	Q!ā	
00E8711C	2B8600E7	γ.ā+	
00E87120	247C81BC	γ!ā!	
00E87124	8600E785	ēγ.ā	
00E87128	7C878009	!Q!	kernel32.FillConsoleOutputCharacterA
00E8712C	00E78538	8āγ.	
00E87130	67305486	5Tā	

می بینید؟... این پکر هم همان کاری را کرده که PeSpin کرده... پس می توانیم همان کاری را بکنیم که در مورد PeSpin کردیم. یعنی در ImportREC کلیک راست کرده، گزینه Advanced Commands و سپس گزینه Get API Calls را بزنید. کلید OK را بزنید و بعد هم توابع Invalid را از بین ببرید (Cut Thunk) را بزنید و بعد هم فایل دامپ را فیکس کنید (یادتان باشد تیک گزینه Create New IAT را بزنید). حالا فایل آپک شده را اجرا کنید:

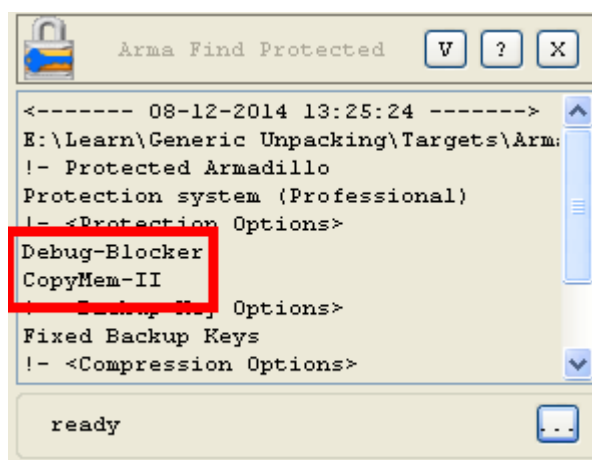


Armadillo

در آنپک کردن Armadillo اول باید بفهمیم که برنامه نویس کدام یک از قابلیت های Armadillo را فعال کرده است؟... اگر با برنامه نویسی باهوشی طرف باشیم که تمامی گزینه ها را فعال کرده باشد... کاری سختی را در پیش داریم ولی اگر فقط گزینه Standard protection تیک داشته باشد، زیاد مشکلی نخواهیم داشت.



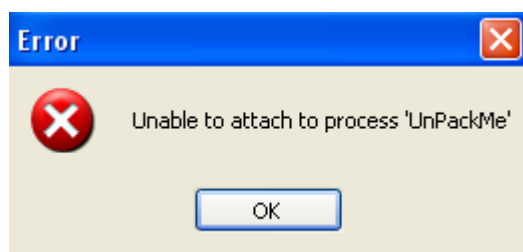
خوب، حالا از کجا بفهمیم که برنامه نویس کدام یکی از این گزینه ها را تیک زده است؟ برای این کار می توانیم از نرم افزار Armadillo Find Protected استفاده کنیم. این برنامه را اجرا می کنید و فایل را به برنامه می دهید:



همانطور که می بینید برنامه نویس هم از قابلیت Debug-Blocker و هم از قابلیت Copy-Mem II استفاده کرده. (یعنی در برنامه Armadillo تیک گزینه Best Protection را زده) خوب، برای آنپک کردن این فایل با این Protection اگر بخواهیم به طور کامل دستی این کار را بکنیم، راه تقریباً وقت گیری را در پیش داریم. (چون باید یک کدهایی را هم تزریق کنیم و با آن کدها قابلیت CopyMem II را از بین ببریم).

قابلیت Debug-Blocker در Armadillo چیست؟

در قابلیت Debug-Blocker، Armadillo دو پروسه برای فایل می سازد. یکی از پروسه ها که به آن پروسه پدر می گویند وظیفه مراقبت از پروسه فرزند را دارد. به این ترتیب امکان Attach کردن به پروسه فرزند وجود ندارد. (حتی با پلاگین OllyAdvanced ©)

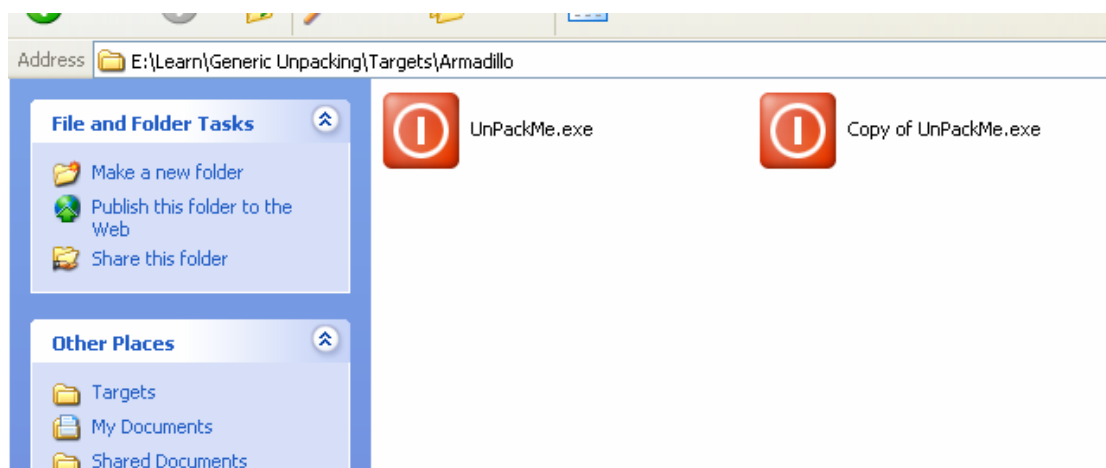


قابلیت CopyMem II در Armadillo چیست؟

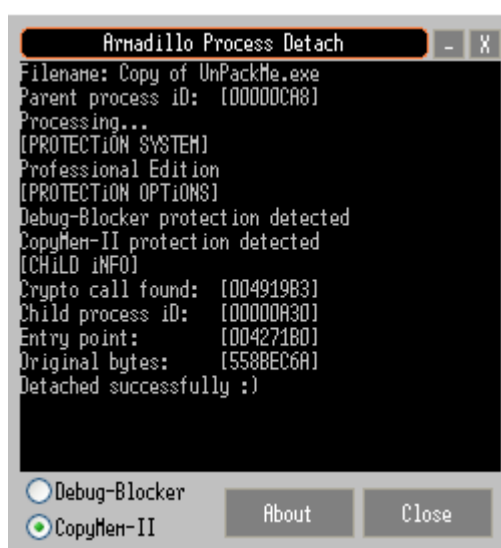
در قابلیت CopyMem II هم دو پروسه ایجاد می شود. اما اینبار پروسه پدر وظیفه Decrypt کردن پروسه فرزند را برعهده دارد. بگذارید مراحل انجام CopyMem II را بنویسم:

۱. پروسه پدر، پروسه فرزند را اجرا می کند.
۲. پروسه فرزند از OEP اجرا می شود. اما چون اطلاعات Encrypt شده است، پروسه فرزند دچار خطا می شود. پروسه پدر با استفاده از تابع WaitForDebugEvent متوجه این خطا می شود.
۳. پروسه فرزند متوقف می شود.
۴. پروسه پدر بررسی می کند آیا نوع خطای اتفاق افتاده همان چیزی بوده که مورد انتظار است؟ اگر خطا درست باشد، پروسه پدر هزار بایت از بایت های پروسه فرزند را با استفاده از تابع WriteProcessMemory Decrypt می کند.
۵. پروسه پدر هزار بایت قبلی را دوباره Encrypt می کند ☹
۵. پروسه فرزند از حالت توقف خارج می شود
۶. پروسه فرزند تا زمانی که خطای دیگری اتفاق بیفتد اجرا می شود و بعد دوباره این حلقه ادامه پیدا می کند.

برای از بین این دو قابلیت می توانیم از نرم افزار Arma-Detach که توسط یکی از اعضای تیم RESURRECTION نوشته شده استفاده کنیم. ابتدا یکی کپی از فایل مورد نظر بگیرید:



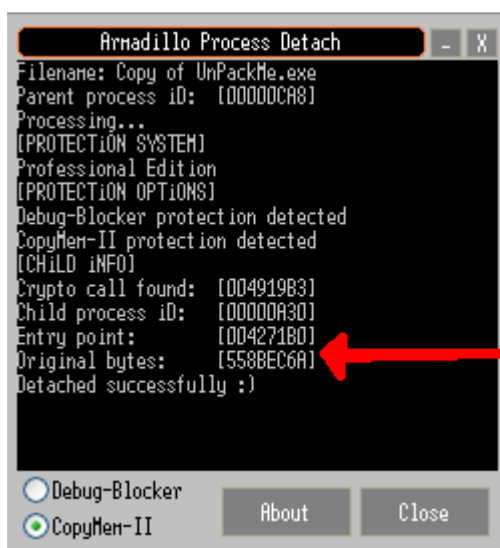
خوب، حالا برنامه Arma-Detach را باز کنید، تیک گزینه CopyMem II را بزنید و فایل کپی شده (Copy of Unpackme.exe) را به داخل برنامه Drag کنید :



خوب، برنامه توانسته با موفقیت قابلیت CopyMem II را از بین ببرد. همچنین توانسته Entry point پروسه فرزند (که همان OEP ماست) را به دست بیاورد. (004271B0) همانطور که در تصویر بالا می بینید، Process ID پروسه فرزند من در کامپیوتر من برابر A30 است. خوب، حالا می توانم به پروسه فرزند Attach کنم. (در OllyDBG در منوی File گزینه Attach را می زنم و به PID فرزند A30 Attach می کنم ☺) Arma-Detach را همینطور باز نگه دارید و Entry point پروسه فرزند یعنی خط 4271B0 بروید:

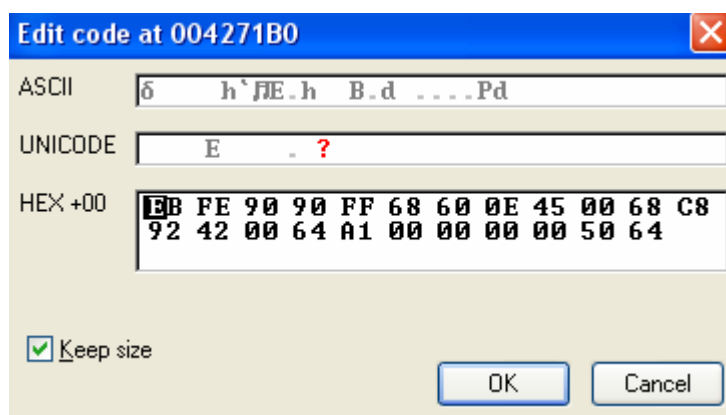
Address	Hex dump	Disassembly
004271B0	- EB FE	JMP SHORT 004271B0
004271B2	90	NOP
004271B3	90	NOP
004271B4	FF 68 60	JMP FAR FWORD PTR DS:[EAX+60]
004271B7	0E	PUSH CS
004271B8	45	INC EBP
004271B9	00 68 C8	ADD BYTE PTR DS:[EAX-38],CH
004271BC	92	XCHG EAX,EDX
004271BD	42	INC EDX
004271BE	00 64 A1 00	ADD BYTE PTR DS:[ECX],AH
004271C2	00 00	ADD BYTE PTR DS:[EAX],AL
004271C4	00 50 64	ADD BYTE PTR DS:[EAX+64],DL
004271C7	89 25 00 00 00 00	MOV DWORD PTR DS:[0],ESP
004271CD	83 C4 A8	ADD ESP,-58
004271D0	53	PUSH EBX
004271D1	56	PUSH ESI

همانطور که می بینید،Arma-Detach برای اینکه از اجرا شدن کامل پروسه جلوگیری کند در Entry point بایت های EBFE را گذاشته.ما باید این بایت ها را به سر جایش باز گردانیم،یک بار دیگر نگاهی به Arma-Detach بیندازید:



همانطور که می بینید Arma-Detach می گوید،در ابتدا بایت های 558BEC6A در Entry point قرار داشته.پس ما باید این بایت ها را به سر جایش برگردانیم،چند خط ابتدای پروسه فرزند را انتخاب کرده و کلیدهای CTRL + E را بزنید تا این بایت ها را Edit کنیم:

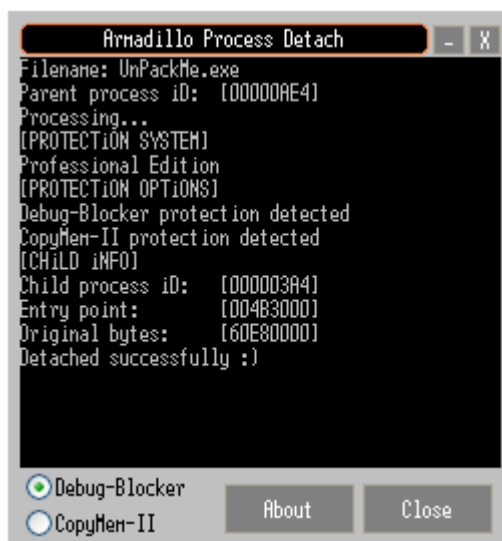
Address	Hex dump	Disassembly
004271B0	- EB FE	JMP SHORT 004271B0
004271B2	90	NOP
004271B3	90	NOP
004271B4	FF 68 60	JMP FAR FWORD PTR DS:[EAX+60]
004271B7	0E	PUSH CS
004271B8	45	INC EBP
004271B9	00 68 C8	ADD BYTE PTR DS:[EAX-38],CH
004271BC	92	XCHG EAX,EDX
004271BD	42	INC EDX
004271BE	00 64 A1 00	ADD BYTE PTR DS:[ECX],AH
004271C2	00 00	ADD BYTE PTR DS:[EAX],AL
004271C4	00 50 64	ADD BYTE PTR DS:[EAX+64],DL
004271C7	89 25 00 00 00 00	MOV DWORD PTR DS:[0],ESP
004271CD	83 C4 A8	ADD ESP,-58
004271D0	53	PUSH EBX
004271D1	56	PUSH ESI
004271D2	57	PUSH EDI
004271D3	89 65 E8	MOV DWORD PTR SS:[EBP-18],ESP
004271D6	FF 15 DC 0A 46 00	CALL DWORD PTR DS:[460ADC]
004271DC	33 D2	XOR EDX,EDX



حالا بایت های اصلی(558BEC6A) را بنویسید و کلید OK را بزنید.

Address	Hex dump	Disassembly
004271B0	55	PUSH EBP
004271B1	8BEC	MOV EBP,ESP
004271B3	6A FF	PUSH -1
004271B5	68 600E4500	PUSH 00450E60
004271B8	68 C8924200	PUSH 004292C8
004271BF	64:A1 00000000	MOV EAX,DWORD PTR FS:[0]
004271C5	50	PUSH EAX
004271C6	64:8925 00000000	MOV DWORD PTR FS:[0],ESP
004271CD	83C4 A8	ADD ESP,-58

خوب، حالا باید قابلیت Debug-Blocker را از بین ببریم. برای این کار یک Arma-Detach دیگر باز کنید و این بار تیک گزینه Debug-Blocker را بزنید و فایل اصلی (Unpackme.exe) را به آن بدهید:



خوب، حالا یک پروسه دیگر برای از بین بردن قابلیت Debug-Blocker ساخته شده، یک OllyDBG دیگر باز کنید و به این پروسه جدید (همانطور که می بینید در کامپیوتر من این پروسه 3A4 است) Attach کنید. حالا هم همانند قبل به Entry point این پروسه بروید: (004B3000) و بعد بایت های اصلی را جایگزین EBFE کنید (60E8):

Address	Hex dump	Disassembly
004B3000	60	PUSHAD
004B3001	E8 00000000	CALL 004B3006
004B3006	5D	POP EBP
004B3007	50	PUSH EAX
004B3008	51	PUSH ECX
004B3009	0FCA	BSWAP EDX
004B300B	F7D2	NOT EDX
004B300D	9C	PUSHFD
004B300E	F7D2	NOT EDX
004B3010	0FCA	BSWAP EDX
004B3012	EB 0F	JMP SHORT 004B3023
004B3014	B9 EB0FB8EB	MOV ECX,EBB80FEB
004B3019	07	POP ES
004B301A	B9 EB0F90EB	MOV ECX,EB900FEB
004B301F	08FD	OR CH,BH
004B3021	EB 0B	JMP SHORT 004B302E
004B3023	F2:	PREFIX REPNE:
004B3024	EB F5	JMP SHORT 004B3018
004B3026	EB F6	JMP SHORT 004B301C

خوب، حالا باید در داخل این پروسه جدید OEP را به دست بیاوریم... برای این کار می توانیم از روشی مخصوص Armadillo استفاده کنیم. یعنی استفاده از تابع CreateThread ©

در پلاگین Command Bar تایپ کنید: HE CreateThread (تا بر روی تابع CreateThread یک Hardware BP on execution بگذاریم) و بعد برنامه را با F9 اجرا کنید. (مراقب تابع OutputDebugStringA باشید. Armadillo از این تابع برای بستن OllyDBG استفاده می کند) پس با پلاگین OllyAdvanced یا پلاگین های دیگر این تابع را از بین ببرید) در ضمن چون Armadillo همانند Enigma از خطای Access violation on write برای کند کردن دیباگ برنامه استفاده می کند، بهتر است به جای OllyDBG از OllyICE استفاده کنید. (اگر دیدید که OllyDBG جواب نمی دهد.)

2F6D8	00B9BF53	CALL to CreateThread from 00B9BF
2F6DC	00000000	pSecurity = NULL
2F6E0	00000000	StackSize = 0
2F6E4	00B9C82B	ThreadFunction = 00B9C82B
2F6E8	00000000	pThreadParam = NULL
2F6EC	00000000	CreationFlags = 0
2F6F0	0012F6FC	pThreadId = 0012F6FC
2F6F4	004C9E08	UnPackMe.004C9E08
2F6F8	00000000	
2F6FC	00000001	
2F700	0012F774	
2F704	00BAF774	RETURN to 00BAF774 from 00B9BEA4
2F708	00000000	
2F70C	00000004	
2F710	00000000	

Paused

خوب، در تابع CreateThread متوقف شدیم، کلیدهای ALT + F9 را بزنید:

00B9BF53	50	PUSH EAX	
00B9BF54	FF15 4C62B800	CALL DWORD PTR DS:[BB624C]	kern
00B9BF5A	5F	POP EDI	
00B9BF5B	5E	POP ESI	
00B9BF5C	C9	LEAVE	
00B9BF5D	C3	RETN	
00B9BF5E	55	PUSH EBP	
00B9BF5F	8BEC	MOV EBP,ESP	
00B9BF61	81EC 28010000	SUB ESP,128	
00B9BF67	56	PUSH ESI	
00B9BF68	57	PUSH EDI	
00B9BF69	BE 9CBCB800	MOV ESI,0BBBC9C	ASCII
00B9BF6E	8DBD 08FEFFFF	LEA EDI,DWORD PTR SS:[EBP-128]	
00B9BF74	A5	MOVS DWORD PTR ES:[EDI],DWORD PTR DS:[E	
00B9BF75	A5	MOVS DWORD PTR ES:[EDI],DWORD PTR DS:[E	
00B9BF76	66:A5	MOVS WORD PTR ES:[EDI],WORD PTR DS:[ESI	

چند بار کلید F8 را بزنید تا از دستور RET رد بشوید.

00BAF767	E8 5566FEFF	CALL 00B950C1	
00BAF76C	330B	XOR EBX,EBX	
00BAF76E	53	PUSH EBX	
00BAF76F	E8 30C7FEFF	CALL 00B9BEA4	
00BAF774	59	POP ECX	
00BAF775	BE 68FB8B00	MOV ESI,0BBFB68	
00BAF77A	8BCE	MOV ECX,ESI	
00BAF77C	E8 0F8EFDFF	CALL 00B88590	
00BAF781	84C0	TEST AL,AL	
00BAF783	75 09	JNZ SHORT 00BAF78E	
00BAF785	6A 01	PUSH 1	
00BAF787	8BCE	MOV ECX,ESI	
00BAF789	E8 210DFDFF	CALL 00B8D4AF	
00BAF78E	C705 E8C0B800 0	MOV DWORD PTR DS:[B8C0E8],0BB800C	
00BAF798	B9 00ECB800	MOV ECX,0BBECD0	
00BAF79D	E8 5E27FDFF	CALL 00B81F00	
00BAF7A2	53	PUSH EBX	
00BAF7A3	E8 5827FDFF	CALL 00B81F00	
00BAF7A8	59	POP ECX	

معمولاً دومین دستور CALL ECX زیر این خط به OEP می رود ☺

00BAF878	FF77 0C	PUSH DWORD PTR DS:[EDI+0C]	
00BAF87B	8B50 74	MOV EDX,DWORD PTR DS:[EAX+74]	
00BAF87E	3350 58	XOR EDX,DWORD PTR DS:[EAX+58]	
00BAF881	3350 08	XOR EDX,DWORD PTR DS:[EAX+8]	
00BAF884	2BCA	SUB ECX,EDX	
00BAF886	FFD1	CALL ECX	
00BAF888	8945 E4	MOV DWORD PTR SS:[EBP-1C],EAX	
00BAF88B	8B45 E4	MOV EAX,DWORD PTR SS:[EBP-1C]	
00BAF88E	8B4D F0	MOV ECX,DWORD PTR SS:[EBP-10]	
00BAF891	64:8900 00000000	MOV DWORD PTR FS:[0],ECX	
00BAF898	5F	POP EDI	
00BAF899	5E	POP ESI	
00BAF89A	5B	POP EBX	
00BAF89B	C3	RETN	

پس بر روی این خط یک BP بگذارید و برنامه را با F9 اجرا کنید و یک بار F8 را بزنید تا به OEP برسید:


```

004271A7 0E          PUSH CS
004271A8 DBE9       FUCOMI ST,ST(1)
004271AA 35 1E876966 XOR EAX,6669871E
004271AF 1E         PUSH DS
004271B0 42         INC EDX
004271B1 72 1A      JB SHORT UnPackMe.004271CD
004271B3 E4 E8      IN AL,0E8
004271B5 91         XCHG EAX,ECX
004271B6 96         XCHG EAX,ESI
004271B7 05 52 F9 9E ADC BYTE PTR DS:[EDX-7],9E
004271B8 46         INC ESI
004271B9 85 BB F6 EAB6F9 TEST DWORD PTR DS:[EBX+F9B6EAF6],ED
004271C2 F6         ???
004271C3 8E 17      MOV SS,WORD PTR DS:[EDI]
004271C5 A9 920732F9 TEST EAX,F9320792
004271C7 F6         ???
004271C8 8E 17      MOV SS,WORD PTR DS:[EDI]
004271CD 7A 32      JPE SHORT UnPackMe.00427201
004271CF 26 44      INC ESP
004271D1 AF        SCAS DWORD PTR ES:[EDI]
004271D2 A1 07721109 MOV EAX,DWORD PTR DS:[9117207]
004271D7 9B        WAIT
004271D8 CB        RETF
004271D9 F3        PREFIX REP:
004271DA B0 8E      MOV AL,8E
004271DB 24 28      AND AL,28

```

همانطور که می بینید دستوراتی که در نزدیکی OEP قرار دارند، Decrypt شده هستند ☺
 خوب، حالا ما دو تا پروسه داریم، که یکی از آنها OEP اش مشکل دارد و دیگری IAT آن مشکل دارد. من IAT پروسه دوم را کپی می
 کنم در پروسه اول، تا مشکلات پروسه اول حل شود ☺
 خوب، حالا در پروسه اول IAT را پیدا می کنیم. (بر روی یکی از CALL هایی که به توابع API می روند، کلیک راست کرده و گزینه
 Follow in dump و سپس گزینه Memory Address را می زنیم.)

حالا به پنجره دامپ نگاه کنید:

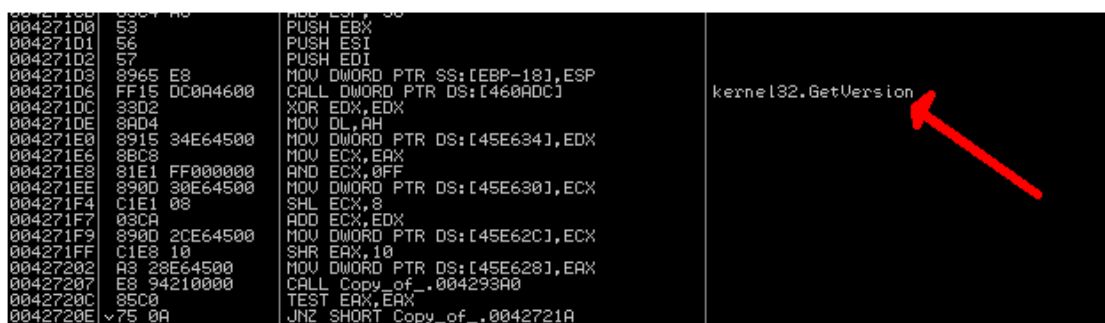
Address	Value	ASCII	Comment
00460814	00000000		
00460818	77DD6BF0	kk w	ADVAPI32.RegCloseKey
0046081C	77DD761B	+v w	ADVAPI32.RegOpenKeyExA
00460820	00B9A6A3	u w	
00460824	77DDEBE7	r w	ADVAPI32.RegSetValueExA
00460828	77DD7883	sk w	ADVAPI32.RegQueryValueExA
0046082C	00B96E1E	Am	
00460830	5D0965CF	=e j	COMCTL32.InitCommonControls
00460834	5D0A03D8	7# j	COMCTL32.ImageList_Destroy
00460838	00B96D97	Am	
0046083C	77F16AB1	z w	GDI32.GetClipboard
00460840	77F19536	6 w	GDI32.ExcludeClipRect
00460844	77F16A66	fj w	GDI32.IntersectClipRect
00460848	77F1ADC3	h w	GDI32.MoveToEx
0046084C	77F1D9BF	l w	GDI32.LineTo
00460850	77F18B74	ti w	GDI32.SetTextAlign
00460854	77F1B590	ei w	GDI32.SetPixelV
00460858	77F17D01	0 w	GDI32.GetViewportExtEx
0046085C	77F17C89	ei w	GDI32.GetWindowExtEx
00460860	77F45A37	7Z w	GDI32.PtVisible

حالا همانند مثال های قبل اطلاعات IAT را به دست می آوریم:

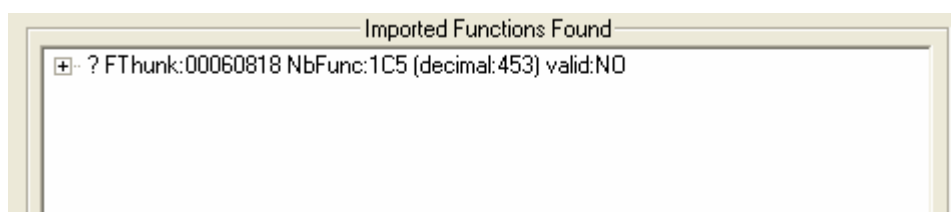
Start of IAT=460814 (RVA: 60814)
 End of IAT=460F2C (RVA: 60F2C)
 Size of IAT=718

خوب، همانطور که گفتم IAT پروسه دوم ما (Unpackme.exe) مشکلی ندارد. ما IAT پروسه دوم را در IAT پروسه اول (copy of unpackme.exe) کپی می کنیم.

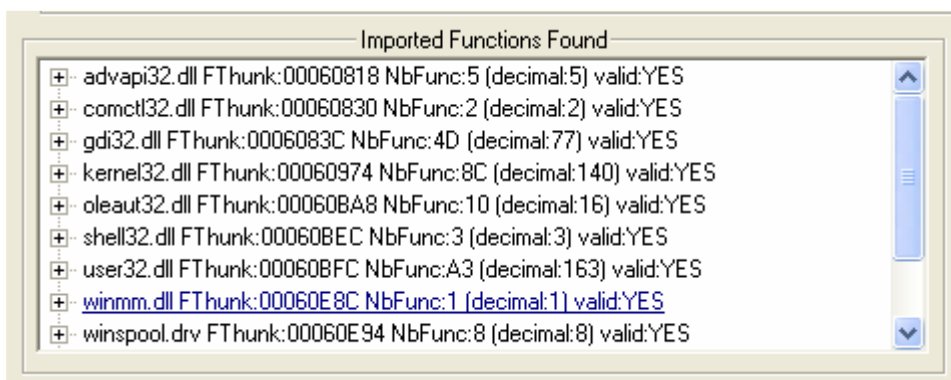
پس ما از خط 460814 تا خط 460F2C را از پروسه دوم کپی کرده و در پروسه اول (Copy of unpackme.exe) Paste می کنیم. (ابتدا در پنجره دامپ این خطوط را انتخاب می کنید، کلیک راست کرده و گزینه Binary Copy و سپس گزینه Binary Paste را می زنید، حالا در پروسه اول (Copy of unpackme.exe)، در پنجره دامپ این خطوط (460F2C تا 460814) را انتخاب می کنید و کلیک راست کرده گزینه Binary و سپس گزینه Binary Paste را می زنید.)



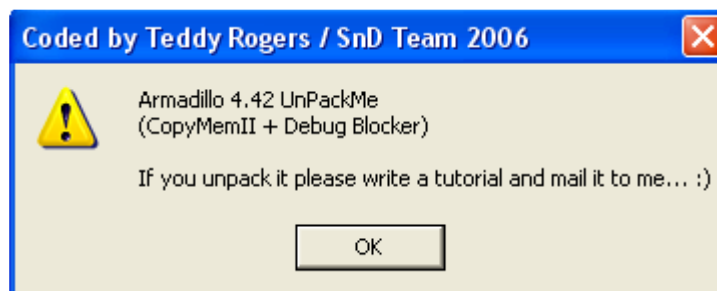
همانطور که می بینید حالا IAT درسته شده است. پس از پروسه اول (Copy of unpackme.exe) دامپ بگیرید و این پروسه را در ImportREC باز کنید، OEP را بدهید و سایر مراحل را انجام دهید:



چون بین توابع API مربوط به هر فایل DLL مقدار صفر وجود ندارد. ImportREC نمی تواند توابع را از هم جدا کند. گزینه Show invalid را بزنید و بر روی توابع Invalid کلیک راست کرده و گزینه Cut Thunk را بزنید:



حالا فایل دامپ خود را فیکس کنید، می بینید که فایل دامپ بی هیچ مشکلی اجرا می شود ☺



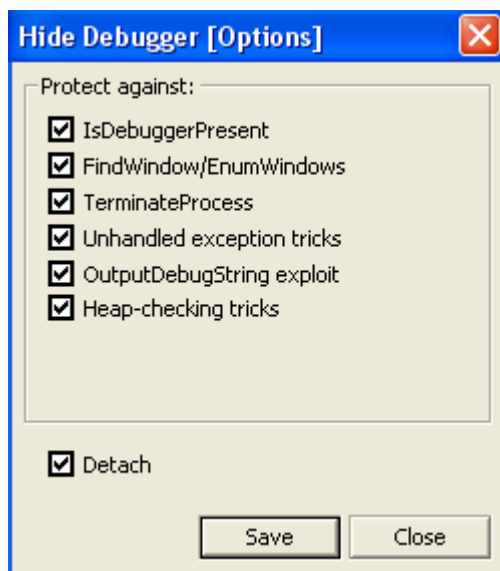
توضیح: اگر برنامه نویس علاوه بر قابلیت های CopyMemII و Debug Blocker تیک گزینه Import Table elimination را می زد. برای درست کردن IAT باید از برنامه ArmInline استفاده می کردیم ☺

TheMida

فایل مورد نظر را در OllyDBG باز کنید. همانطور که دیدید با باز کردن فایل در OllyDBG دیباگر بسته می شود. این به این خاطر است که برنامه کاری شبیه به ExeCryptor می کند. (البته ExeCryptor از TLS Callback Function استفاده می کرد، ولی این از یک تابع دیگر استفاده می کرد.)
برای حل این مشکل می توانید از یک OllyDBG (OLLYDBG 9in1.EXE) که برای TheMida ساخته شده، استفاده کنید. فایل مورد نظر را در آن OllyDBG باز کنید:

Address	Hex dump	Disassembly
0046A014 <Mod	BB 00000000	MOV EAX,0
0046A019	60	PUSHAD
0046A01A	0BC0	OR EAX,EAX
0046A01C	74 68	JE SHORT 0046A086
0046A01E	E8 00000000	CALL 0046A023
0046A023	58	POP EAX
0046A024	05 53000000	ADD EAX,53
0046A029	8038 E9	CMP BYTE PTR DS:[EAX],0E9
0046A02C	75 13	JNZ SHORT 0046A041
0046A02E	61	POPAD

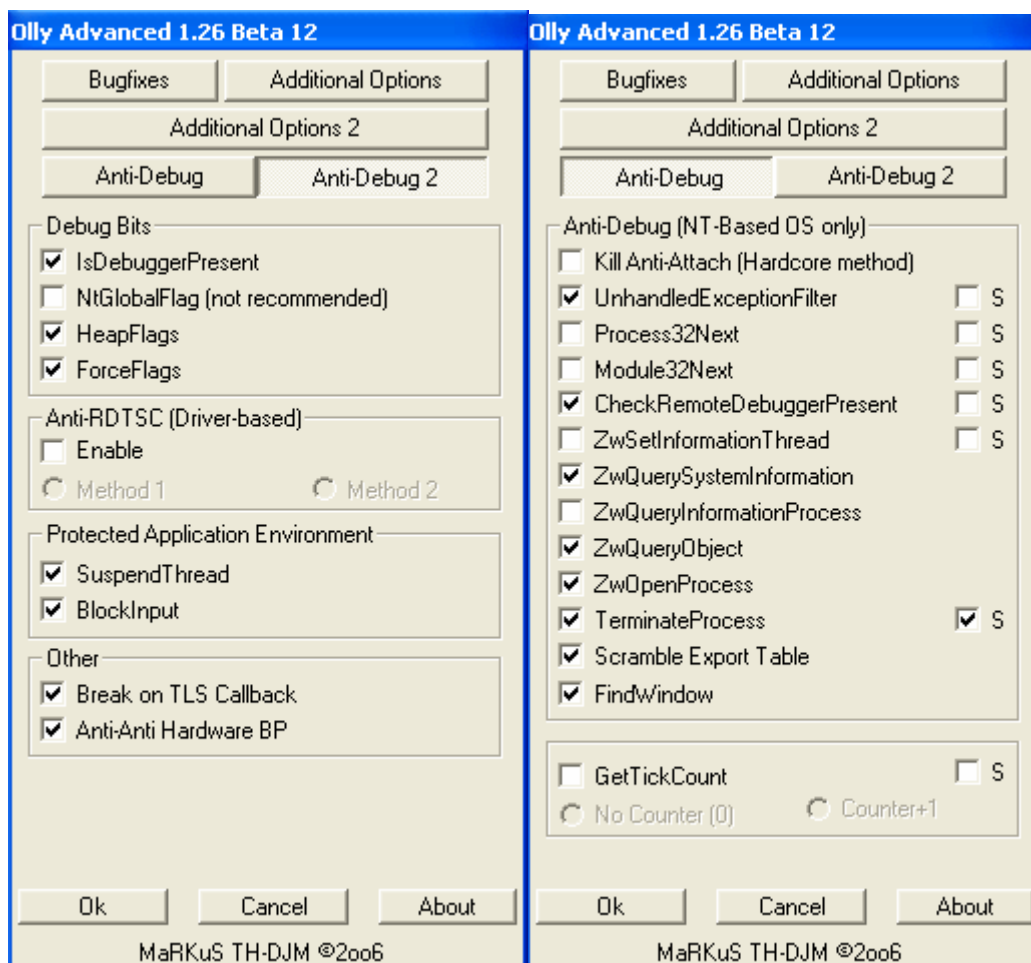
خوب، این پروتکتور از آنتی دیباگ های بسیار زیادی استفاده می کند. پس باید در مقابل آنها مقاوم باشیم. آخرین ورژن پلاگین HideDebugger را دانلود کرده و تمامی گزینه هایش را تیک بزنید:



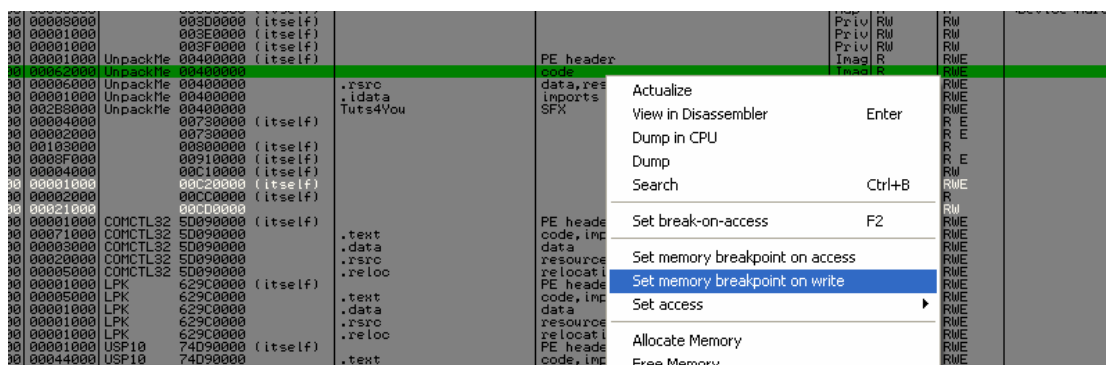
پلاگین Phantom را هم دانلود کنید:



در پلاگین Phantom به هیچ وجه گزینه Fake Windows Version را تیک نزنید. پلاگین OllyAdvanced را هم دانلود کنید و همانند تصویر زیر تنظیم کنید:



خوب، حالا دیگر فایل باید در OllyDBG اجرا شود. (برای تست کلید F9 را بزنید.) این فایلی که ما داریم IAT را Redirect که چه عرض کنیم... کلاً ناپود می کند... در ضمن Call هایی که به توابع API می روند هم به طور کامل Redirect شده اند... لذا ما اگر بخواهیم بعد از یافتن OEP، IAT و CALL ها را درست کنیم، کار بسیار سخت و طاقت فرسایی را در پیش خواهیم داشت. لذا باید قبل از یافتن OEP کاری کنیم که پکر، IAT و CALL ها را Redirect نکند. یعنی Magic Jump را پیدا کنیم. خوب، برای یافتن Magic Jump ها ابتدا بر روی سکشنی که محتوی code است، Memory Breakpoint on write بگذارید. چون می خواهیم محلی که عملیات Decrypt انجام می شود را پیدا کنیم:



برنامه را با F9 اجرا کنید:

	FFD0	CALL EAX
	8BF8	MOV EDI,EDX
	8BF1	MOV ESI,ECX
	8BD1	MOV EDX,ECX
	8BC8	MOV ECX,EAX
	F3:B4	REP MOVS BYTE PTR ES:[EDI],BYTE PTR DS:[ESI]
	C685 C1081B07 56	MOV BYTE PTR SS:[EBP+71B08C1],56
	68 396D1FD4	PUSH D41F6D39
	FFB5 35301B07	PUSH DWORD PTR SS:[EBP+71B3035]
	8D85 81012807	LEA EAX,DWORD PTR SS:[EBP+7280181]
	FFD0	CALL EAX
	68 00800000	PUSH 8000
	6A 00	PUSH 0
	52	PUSH EDX
	FFD0	CALL EAX
	8BC0	MOV EAX,EAX
	83BD A1331B07 00	CMP DWORD PTR SS:[EBP+71B33A1],0
√	75 09	JNZ SHORT 006F5E3A
	83BD B11B1B07 00	CMP DWORD PTR SS:[EBP+71B1BB1],0

اینجا چیز خاصی نیست، کلید F8 و سپس F9 را بزنید تا از اینجا رد شوید:

	MOV DWORD PTR SS:[ESP],ECX
F574	ADD DWORD PTR SS:[ESP],74F539EB
	POP DWORD PTR DS:[EAX]
74	SUB DWORD PTR DS:[EAX],74F539EB
	CLD
	LODS DWORD PTR DS:[ESI]
	JMP 007090E3
	POP EDX
	PUSHAD
	XOR EDI,DWORD PTR DS:[ECX]
	POPAD

حالا چهار بار F9 را بزنید:

	27D0	SUB EAX,EAX
	5B	POP EBX
	FC	CLD
	AB	STOS DWORD PTR ES:[EDI]
	F5	CMC
	AD	LODS DWORD PTR DS:[ESI]
	F9	STC
	C746 FC 8F62D23C	MOV DWORD PTR DS:[ESI-4],3CD2628F
	50	PUSH EAX
	B8 8F62D23C	MOV EAX,3CD2628F
	3146 FC	XOR DWORD PTR DS:[ESI-4],EAX
	58	POP EAX
√	0F85 07000000	JNZ 00709639
√	0F89 01000000	JNS 00709639
	F9	STC
^	E9 D2FAFFFF	JMP 00709110
	F9	STC
	89B5 DD2D1B07	MOV DWORD PTR SS:[EBP+71B20DD],ESI
	F8	CLC
	50	POPAD

حالا با F8 به جلو بروید تا به اینجا برسید:

	67:04 82	ADD AL,82
	0C 88	OR AL,88
	C785 7D181B07 00000000	MOV DWORD PTR SS:[EBP+71B187D],0
	60	PUSHAD
	60	PUSHAD
	81DB E0CB6B6A	SBB EBX,6A6BCBE0
	B3 E3	MOV BL,0E3
	61	POPAD
	61	POPAD
√	E9 0E000000	JMP 0070638C
	183CCB	SBB BYTE PTR DS:[EBX+ECX*8],BH
	3089 D790183D	XOR BYTE PTR DS:[ECX+3D1890D7],CL
	41	INC ECX
√	76 54	JBE SHORT 0070640E
	8059 C7 85	SBB BYTE PTR DS:[ECX-39],85
	B5 19	MOV CH,19

حالا کلیدهای CTRL + B را بزنید و تایپ کنید: 0F 00 ?? ?? ?? 39 85 و کلید OK را بزنید:

Enter binary string to search for

ASCII: 9 ????*

UNICODE: ???

HEX +07: 39 85 ?? ?? ?? 0? 0F|

☐ Entire block

☒ Case sensitive

<< >>

OK Cancel

3985 29121B07	CMP DWORD PTR SS:[EBP+71B1229],EAX
0F84 B5000000	JE 0070673A
F9	STC
0F80 19000000	JO 007066A5
E9 14000000	JMP 007066A5
9F	LAHF
4C	DEC ESP
BE 55312039	MOV ESI,39203155
C07F 33 2E	SAR BYTE PTR DS:[EDI+33],2E

پرشی که در زیر این خط قرار دارد،(JE 0070673A)، باید JMP شود:

F7D0	NOT EAX
91	XCHG EAX,ECX
91	XCHG EAX,ECX
F8	CLC
3985 29121B07	CMP DWORD PTR SS:[EBP+71B1229],EAX
E9 B6000000	JMP 0070673A
90	NOF
F9	STC
0F80 19000000	JO 007066A5
E9 14000000	JMP 007066A5
9F	LAHF
4C	DEC ESP
BE 55312039	MOV ESI,39203155
C07F 33 2E	SAR BYTE PTR DS:[EDI+33],2E
59	POP ECX
9A 964A5CD1 0FF7	CALL FAR F70F:D15C4A96
BA 68000000	MOV EDX,68
008B 042481C4	ADD BYTE PTR DS:[EBX+C4812404],CL
04 00	ADD AL,0
0000	ADD BYTE PTR DS:[EAX],AL

حالا دوباره کلیدهای CTRL + B را بزنید و تایپ کنید FF D3

7E	DB 7E
00	DB 00
25	DB 25
> 83BD D52E1B07 01	CMP DWORD PTR SS:[EBP+71B2ED5],1
0F84 D0000000	JE 00706EE7
60	PUSHAD
80D9 DD	SBB CL,0DD
BA 12F27A5A	MOV EDX,5A7AF212
61	POPAD
3B8D E1321B07	CMP ECX,DWORD PTR SS:[EBP+71B32E1]
0F84 BA000000	JE 00706EE7
0F88 0C000000	JS 00706E3F
E9 07000000	JMP 00706E3F
2F	DB 2F
22	DB 22
BA 94CFC69B	MOV EDX,9BC6CF94
> 3B8D F10E1B07	CMP ECX,DWORD PTR SS:[EBP+71B0EF1]
0F84 9C000000	JE 00706EE7
0F81 09000000	JNO 00706E5A
60	PUSHAD
0F81 00000000	JNO 00706E58
F9	STC
61	POPAD
> 3B8D 75281B07	CMP ECX,DWORD PTR SS:[EBP+71B2875]
0F84 81000000	JE 00706EE7
F9	STC
> 8D9D 1CA72F07	LEA EBX,DWORD PTR SS:[EBP+72FA71C]
60	PUSHAD
8AEA	MOV CH,DL
66:8BC8	MOV CX,AX
61	POPAD
FFD3	CALL EBX
60	PUSHAD
B8 7EBDA53D	MOV EAX,3DA5BD7E
60	PUSHAD
60	PUSHAD
61	POPAD
E9 0E000000	JMP 00706E92
F4	DB F4

بالای این خط چهار پرش وجود دارد که هر چهار تای آن به یک خط می روند. شما باید هر چهار تا را Nop کنید. (بهرتر است اول کدها را آنالیز کنید تا بهتر ببینید):

7E	DB 7E	
00	DB 00	
25	DB 25	
> 838D D52E1B07 01	CMP DWORD PTR SS:[EBP+71B2ED5],1	
90	NOP	
90	NOP	
90	NOP	
90	NOP	
90	NOP	
90	NOP	
60	PUSHAD	
8009 DD	SBB CL,000	
BA 12F27A5A	MOV EDI,5A7AF212	
61	POPAD	
388D E1321B07	CMP ECX,DWORD PTR SS:[EBP+71B32E1]	
90	NOP	
90	NOP	
90	NOP	
90	NOP	
90	NOP	
90	NOP	
0F88 0C000000	JNS 00706E3F	
E9 07000000	JMP 00706E3F	
2F	DB 2F	
22	DB 22	
BA 94CF69B	MOV EDI,9BC6CF94	
> 388D F10E1B07	CMP ECX,DWORD PTR SS:[EBP+71B0EF1]	
90	NOP	
90	NOP	
90	NOP	
90	NOP	
90	NOP	
90	NOP	
0F81 09000000	JNS 00706E5A	
60	PUSHAD	
0F81 00000000	JNS 00706E58	
F9	STC	
61	POPAD	
> 388D 75281B07	CMP ECX,DWORD PTR SS:[EBP+71B2875]	
90	NOP	
90	NOP	
90	NOP	
90	NOP	
90	NOP	
90	NOP	
F9	STC	
> 8D9D 1CA72F07	LEA EBX,DWORD PTR SS:[EBP+72FA71C]	
60	PUSHAD	
8AEA	MOV CH,DL	
66:8BC8	MOV CX,AX	

همانطور که دیدید ما پنج پرش را دستکاری کردیم، به این ترتیب IAT ما Redirect نمی شود. اما همچنان مشکلاتی با CALL هایی که به توابع API می روند، خواهیم داشت. برای یافتن OEP می توانیم از Memory Breakpoint استفاده کنیم. ابتدا باید بفهمیم که در کجا کدها به طور کامل Decrypt شده اند؟ برای این کار روی آخر تابع ZwFreeVirtualMemory یک BP بگذارید:

7C90DA45	90	NOP
7C90DA46	90	NOP
7C90DA47	90	NOP
7C90DA48 ZwFreeVirtualMemory	B8 53000000	MOV EAX,53
7C90DA49	BA 0003FE7F	MOV EDI,7FFE0300
7C90DA52	FF12	CALL DWORD PTR DS:[EDI]
7C90DA54	C2 1000	RET 10
7C90DA57	90	NOP
7C90DA58	90	NOP
7C90DA59	90	NOP

ممکن است سوال کنید چرا آخر این تابع BP گذاشتیم؟... برای اینکه برخی پروتکتورها در ابتدای توابع API به دنبال بایت CC می گردند. چون OillyDBG از این بایت برای گذاشتن BP استفاده می کند. پروتکتورها متوجه گذاشتن Breakpoint شده و در نتیجه دیباگر را شناسایی می کنند. لذا اگر با یک پروتکتور خوب طرف هستید، سعی کنید یا Hardware BP بگذارید، یا اگر بر روی تابع API، BP می گذارید، آخر آن بگذارید... البته می توانید از Memory Breakpoint هم استفاده کنید، ولی چون Memory BP سرعت پردازش فایل در دیباگر را پایین می آورد، معمولاً این کار را نمی کنند. برنامه را با F9 اجرا کنید:

Registers (FPU)	
EAX	00000000
ECX	0012FF28
EDX	7C90EB94 ntdll.KiFastS
EBX	006E4669 UnpackMe.006E
ESP	0012FF2C
EBP	0012FF48
ESI	7C90DA48 ntdll.ZwFreeV
EDI	0040E209 UnpackMe.0040
EIP	7C90DA54 ntdll.7C90DA5
C 0	ES 0023 32bit 0(FFFFF
P 1	CS 001B 32bit 0(FFFFF
D 0	DS 0023 32bit 0(FFFFF

می بینید؟... مقدار رجیسترها تغییر کرده اند (رنگشان سفید شده)... تا زمانی کلید F9 را بزنید که مقدار رجیسترها تغییر نکند (یعنی همه شان سیاه باشند، در ضمن آن Memory BP را هم پاک کنید):

90	NOP
90	NOP
90	NOP
90	NOP
90	NOP
115A C6	ADC DWORD PTR DS:[EDX-3A],EBX
B8 615FF9BF	MOV EAX,BFF95F61
6B00 07	IMUL EDX,EAX,-29
^ E1 CE	LOOPDE SHORT 0042718B
3A7A 65	CMP BH,BYTE PTR DS:[EDX+65]
AB	STOS DWORD PTR ES:[EDI]
0FA1	POP FS
6D	INS DWORD PTR ES:[EDI],DX
43	INC EBX
1F	POP DS
^ 73 A7	JNB SHORT 0042716F
BF 4933B0D3	MOV EDI,D3B03349
8B00	MOV EDX,EAX
B7 2C	MOV BH,2C
^ E2 3F	LOOPD SHORT 00427212
CC	INT3
68 36358C77	PUSH 778C3536
^ E0 79	LOOPDNE SHORT 00427254
1933	SBB DWORD PTR DS:[EBX],ESI
D28A D4891534	ROR BYTE PTR DS:[EDX+341589D4],CL
E6 45	OUT 45,AL
008B C881E1FF	ADD BYTE PTR DS:[EBX+FFE181C8],CL
0000	ADD BYTE PTR DS:[EAX],AL
0089 0D30E645	ADD BYTE PTR DS:[ECX+45E6300D],CL
00C1	ADD CL,AL
^ E1 08	LOOPDE SHORT 004271FF
03CA	ADD ECX,EDX
890D 2CE64500	MOV DWORD PTR DS:[45E62C],ECX
C1E8 10	SHR EAX,10
A3 28E64500	MOV DWORD PTR DS:[45E628],EAX
E8 94210000	CALL 004293A0
85C0	TEST EAX,EAX
^ 75 0A	JNZ SHORT 0042721A
6A 1C	PUSH 1C

ما الان در نزدیکی های OEP و در سکشن اصلی هستیم. اما چند بایت در نزدیکی OEP دزدیده شده است. برای یافتن بایت های دزدیده شده معمولا دو راه وجود دارد:

۱. در داخل پکر بگردیم و بایت ها را پیدا کنیم (کاری که در No Name Packer 2 کردیم)
۲. از ساختار زبان نویسی استفاده کنیم.

در اینجا همان روش دوم راحت تر است ☺ من اول باید بفهمم که برنامه در چه زبانی نوشته شده است. (تشخیص آن برای یک باتجربه بسیار آسان است. اگر هم نمی توانید تشخیص بدهید برنامه در چه زبانی نوشته شده، می توانید اول از فایل دامپ بگیرید و بعد با نرم افزارهایی مثل RDG Packer Detector تشخیص بدهید فایل در چه زبانی نوشته شده است.)

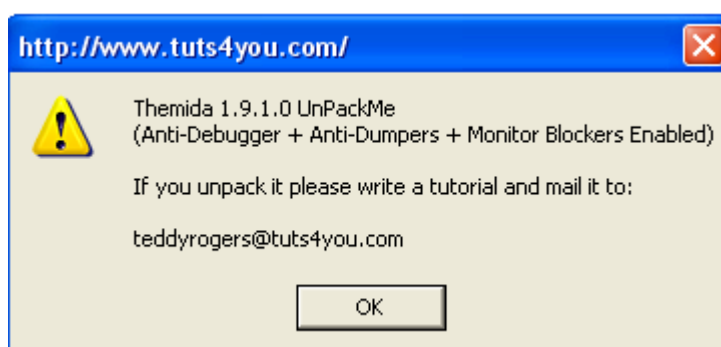
خوب، برنامه در زبان Visual C++ نوشته شده است. پس یک برنامه پیدا می کنم که در این زبان نوشته شده باشد و کدهای ابتدایی آن را به کدهای این فایل اضافه می کنم. (یعنی آن فایل (فرقی نمی کند فایل چی باشد، فقط در C++ ورژن ۵ نوشته شده باشد). را در OllyDBG باز کنیم و کدهای ابتدای آن را انتخاب کرده، کلیک راست می کنم، گزینه Binary و سپس Binary Copy را می زنم و بعد کدهایی که کپی کردم را در این فایل Binary Paste می کنم) ☺

90	NOP
90	NOP
90	NOP
90	NOP
90	NOP
55	PUSH EBP
8BEC	MOV EBP,ESP
6A FF	PUSH -1
68 600E4500	PUSH 00450E60
68 C8924200	PUSH 004292C8
64: A1 00000000	MOV EAX,DWORD PTR FS:[0]
50	PUSH EAX
64: 8925 00000000	MOV DWORD PTR FS:[0],ESP
83C4 A8	ADD ESP,-58
53	PUSH EBX
56	PUSH ESI
57	PUSH EDI
8965 E8	MOV DWORD PTR SS:[EBP-18],ESP
FF15 DC0A4600	CALL DWORD PTR DS:[460ADC]
33D2	XOR EDX,EDX
8AD4	MOV DL,AH
8915 34E64500	MOV DWORD PTR DS:[45E634],EDX
8BC8	MOV ECX,EAX
81E1 FF000000	AND ECX,0FF
890D 30E64500	MOV DWORD PTR DS:[45E630],ECX

البته بعد از Binary Copy/paste کردن باید مقداری کدها را با توجه به فایل خود دستکاری کنید. مثلا خط پنجم برای نصب SEH هست، ما باید در پنجره Stack بینم در کجا SEH نصب شده و آدرس آنجا را در این خط بدهم:

0012FF60	0012FF9C	
0012FF64	0000A042	
0012FF68	7C90EB94	ntdll.KiFastSystemCallRet
0012FF6C	006FA40A	UnpackMe.006FA40A
0012FF70	FFFD3D64	
0012FF74	65B78399	
0012FF78	00699F5A	UnpackMe.00699F5A
0012FF7C	0012FF20	
0012FF80	0012FF9C	
0012FF84	0012FFE0	Pointer to next SEH record
0012FF88	004292C8	SE handler
0012FF8C	00450E60	UnpackMe.00450E60
0012FF90	FFFFFFFF	
0012FF94	00699F5A	UnpackMe.00699F5A
0012FF98	1A6EB7C0	
0012FF9C	0012FFE0	
0012FFA0	00690675	UnpackMe.00690675
0012FFA4	7C910738	ntdll.7C910738
0012FFA8	FFFFFFFF	
0012FFAC	0012FF50	

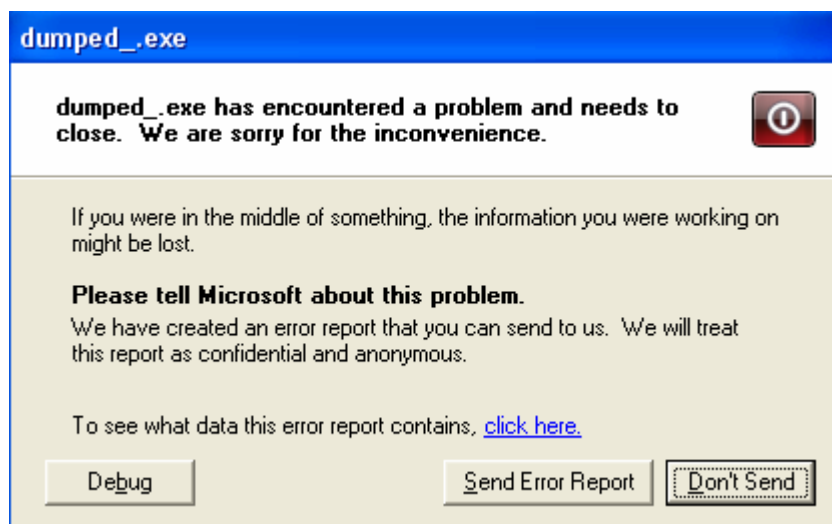
خوب، حالا بر روی OEP دزدیده شده برنامه (4271B0) کلیک راست کرده و گزینه New origin here را بزنید و بعد از فایل دامپ بگیرید.
حالا فایل را با ImportREC فیکس کنید و اجرا کنید:



بله، فایل آپیک شده در سیستم شما اجرا می شود ☺ اما این فایل را در OllyDBG باز کنید و نگاهی دقیق به آن بندازید:

75 0A	JNZ SHORT 0042722D	
6A 10	PUSH 10	
E8 36010000	CALL 00427360	
83C4 04	ADD ESP,4	
C745 FC 00000000	MOV DWORD PTR SS:[EBP-4],0	
E8 872B0000	CALL 00429DC0	
E8 12110000	CALL 00428350	
E8 DABC3E7C	CALL kernel32.GetCommandLineA	GetCommandLineA
90	NOP	
A3 D8EB4500	MOV DWORD PTR DS:[45EBD8],EAX	
E8 32940000	CALL 00430680	
A3 10E64500	MOV DWORD PTR DS:[45E610],EAX	
85C0	TEST EAX,EAX	
74 09	JE SHORT 00427260	
A1 D8EB4500	MOV EAX,DWORD PTR DS:[45EBD8]	

می بینید؟...در خط 42723E مستقیم نوشته شده: CALL GetCommandLineA... توابع API دیگر را نگاه کنید، می بینید که همه همینجوری هستند(به غیر از GetVersion که ما Binary paste کردیم ☺)
خوب، حالا چه اشکالی دارد اگر مستقیم یک تابع API را صدا بزنیم؟...همانطور که بارها گفتم، آدرس توابع API در سیستم های مختلف متفاوت است. و اگر شما یک تابع را مستقیم صدا بزنید، چون آدرس تابع در سیستم دیگران فرق دارد، برنامه فقط در سیستم شما اجرا می شود ☺...کافیست این فایل آپیک شده را در سیستم دیگر اجرا کنید:



حالا چه کار کنیم که این برنامه در هر سیستمی اجرا شود؟... من یکسری کد نوشتم که شما این کدها را در آدرس 460F2D فضای خالی در آنجا موجود است، کپی می کنید... این کدها به طور خودکار تمامی CALL ها را پیدا می کنند و آنها را درست می کنند ☺ (این کدها در داخل فایل Themida_codes.txt موجود است):

6C08F87D	DD oledlg.OleUIBusyA
00000000	DD 00000000
00	DB 00
B8 00104000	MOV EAX,00401000
8038 E8	CMP BYTE PTR DS:[EAX],0E8
74 0A	JE SHORT 00460F41
3D 2B0F4600	CMP EAX,00460F2B
74 3D	JE SHORT 00460F7B
40	INC EAX
EB F1	JMP SHORT 00460F32
8B58 01	MOV EBX,DWORD PTR DS:[EAX+1]
03D8	ADD EBX,EAX
83C3 05	ADD EBX,5
B9 14084600	MOV ECX,00460814
83C1 04	ADD ECX,4
81F9 280F4600	CMP ECX,00460F28
74 E5	JE SHORT 00460F3E
3919	CMP DWORD PTR DS:[ECX],EBX
75 F1	JNZ SHORT 00460F4E
8B58 01	MOV EBX,DWORD PTR DS:[EAX+1]
8078 FF 90	CMP BYTE PTR DS:[EAX-1],90
74 09	JE SHORT 00460F6F
8BF0	MOV ESI,EAX
66:C706 FF15	MOV WORD PTR DS:[ESI],15FF
EB 07	JMP SHORT 00460F76
50	PUSH EAX
48	DEC EAX
8BF0	MOV ESI,EAX
58	POP EAX
EB F2	JMP SHORT 00460F68
894E 02	MOV DWORD PTR DS:[ESI+2],ECX
EB C3	JMP SHORT 00460F3E
90	NOP
90	NOP
0000	ADD BYTE PTR DS:[EAX],AL
0000	ADD BYTE PTR DS:[EAX],AL
0000	ADD BYTE PTR DS:[EAX],AL
0000	ADD BYTE PTR DS:[EAX],AL
0000	ADD BYTE PTR DS:[EAX],AL

خوب...حالا باید این کدها را اجرا کنیم.بر روی خط 460F2D کلیک راست کرده و گزینه New origin here را بزنید و در خط 460F7B یک BP بگذارید که هر وقت اجرای این کدها تمام شد،ما متوجه بشویم ☺ برنامه را با F9 اجرا کنید:

460F68	8078 FF 90	CMP BYTE PTR DS:[EAX-1],90
460F64	74 09	JE SHORT 00460F6F
460F66	8BF0	MOV ESI,EAX
460F68	66:C706 FF15	MOV WORD PTR DS:[ESI],15FF
460F6D	EB 07	JMP SHORT 00460F76
460F6F	50	PUSH EAX
460F70	48	DEC EAX
460F71	8BF0	MOV ESI,EAX
460F73	58	POP EAX
460F74	EB F2	JMP SHORT 00460F68
460F76	894E 02	MOV DWORD PTR DS:[ESI+2],ECX
460F79	EB C3	JMP SHORT 00460F3E
460F7B	90	NOP
460F7C	90	NOP
460F7D	0000	ADD BYTE PTR DS:[EAX],AL
460F7F	0000	ADD BYTE PTR DS:[EAX],AL
460F81	0000	ADD BYTE PTR DS:[EAX],AL
460F83	0000	ADD BYTE PTR DS:[EAX],AL
460F85	0000	ADD BYTE PTR DS:[EAX],AL
460F87	0000	ADD BYTE PTR DS:[EAX],AL
460F89	0000	ADD BYTE PTR DS:[EAX],AL
460F8B	0000	ADD BYTE PTR DS:[EAX],AL
460F8D	0000	ADD BYTE PTR DS:[EAX],AL

خوب...این کدها هم اجرا شد،حالا دوباره نگاهی به CALL های برنامه بیندازید:

```

Disassembly
FF  CALL 0040E5A6
4600  CALL DWORD PTR DS:[<&kernel32.SetLastError>]
MOV ECX,EBX
FF  CALL 0040E5A6
0000  JMP SHORT 0041039E
MOV EAX,DWORD PTR DS:[ESI+80C]
PUSH 0
MOV EAX,DWORD PTR DS:[EAX+44]
4600  CALL DWORD PTR DS:[<&kernel32.SetLastError>]
0000 00 AND DWORD PTR DS:[EAX+D0],0
JMP SHORT 004103B1
0000 38 MOV BYTE PTR DS:[ESI+AD4],38
MOV EAX,[ARG.1]
POP EDI
POP ESI
POP EBX
LEAVE
RET 4
PUSH EBP
MOV EBP,ESP
SUB ESP,2C
PUSH ESI
MOV ESI,[ARG.1]
CMP WORD PTR DS:[ESI+2],1
JNZ SHORT 004103F5
PUSH 2
PUSH 0
LEA EAX,[LOCAL.11]
PUSH ESI
PUSH EAX
FF  CALL 0040E550
TEST EAX,EAX
JE SHORT 004103FC
MOV ECX,[LOCAL.11]
TEST ECX,ECX
JE SHORT 004103FC
MOV EAX,DWORD PTR DS:[ECX]
PUSH 1
CALL DWORD PTR DS:[EAX]
PUSH 0
4600  CALL DWORD PTR DS:[<&kernel32.SetLastError>]
JMP SHORT 004103B1
0000 38 MOV BYTE PTR DS:[ECX+AD4],38
MOV EAX,ESI
POP ESI
LEAVE
RET 4
PUSH EBP
MOV EBP,ESP
SUB ESP,2C

```

می بینید...تمامی CALL های برنامه درست شده و این بار از IAT استفاده می کنند ☺، حالا می توانید فایل آپک شده را در هر سیستمی تست کنید ☺

خوب، ممکن است سوال کنید که آن کدها چه طور عمل می کنند؟... برای اینکه با این کدها آشنایی بیشتری پیدا کنید، در داخل Attachments فایلی به نام TheMida_UDD.rar قرار دارد که شما این فایل را در پوشه UDD (یک فایل UDD است که OllyDBG از آن برای ذخیره اطلاعات یک فایل دیباگ شده، مثل Breakpoint ها، Bookmark ها و ... استفاده می کند.) استخراج کنید و به توضیحی که در جلوی کدها قرار دارد، نگاه کنید:

00000000	DD 00000000	
0	DB 00	
3 00104000	MOV EAX,00401000	Start of code section
338 E8	CMP BYTE PTR DS:[EAX],0E8	Is This is a call?
4 00	JE SHORT 00460F41	If it is a Call,Go to 460F41
0 2B0F4600	CMP EAX,00460F2B	Is it end on code section?
4 3D	JE SHORT 00460F7B	If it is end of code sections,Go to end of codes
0	INC EAX	Increase eax for search next byte
3 F1	JMP SHORT 00460F32	Go to next byte
358 01	MOV EBX,DWORD PTR DS:[EAX+1]	Find Call Destination
308	ADD EBX,EAX	Add eax to ebx for find Call Destination
3C3 05	ADD EBX,5	Add 5 to ebx,and now EBX has Destination API Address
9 14084600	MOV ECX,00460814	Mov ECX to start of IAT
3C1 04	ADD ECX,4	Add 4 to ecx for find next Address in IAT
1F9 280F4600	CMP ECX,00460F28	Is this end of IAT?
4 E5	JE SHORT 00460F3E	If it is end of IAT,go to next byte
319	CMP DWORD PTR DS:[ECX],EBX	Is this Address of IAT is Call Destination?
5 F1	JNZ SHORT 00460F4E	If it isn't Call Destination,go to next address in IAT
358 01	MOV EBX,DWORD PTR DS:[EAX+1]	Mov ebx for replace bytes
078 FF 90	CMP BYTE PTR DS:[EAX-1],90	Is before CALL exist nop?
4 09	JE SHORT 00460F6F	If before CALL exist nop,go to 460F6F
3F0	MOV ESI,EAX	ESI=EAX
6:C706 FF15	MOV WORD PTR DS:[ESI],15FF	Change CALL xxxxxxxx to CALL DWORD PTR DS:[xxxxxxxx]
3 07	JMP SHORT 00460F76	Go to 460F76,for change Call destination
0	PUSH EAX	Push EAX
8	DEC EAX	Decrease EAX
3F0	MOV ESI,EAX	ESI=EAX
8	POP EAX	Pop EAX
3 F2	JMP SHORT 00460F68	Go to 460F68,for change CALL
94E 02	MOV DWORD PTR DS:[ESI+2],ECX	Change CALL Destination to Address of API in IAT ;)
3 C3	JMP SHORT 00460F3E	Go to next byte
0	NOP	End of Codes
0	NOP	
000	ADD BYTE PTR DS:[EAX],AL	
000	ADD BYTE PTR DS:[EAX],AL	
000	ADD BYTE PTR DS:[EAX],AL	

Virtual Machine

Virtual Machine یکی از پروتکشن‌هایی است که امروزه اکثر پروتکتورها سعی می‌کنند این قابلیت را به پروتکتور خود اضافه کنند. زیر این روش کار آنیک شدن فایل را سخت می‌کند و تنها کسانی قادر به آنیک کردن آن می‌باشند که تجربه کافی در زمینه آنیکینگ داشته باشند ☺

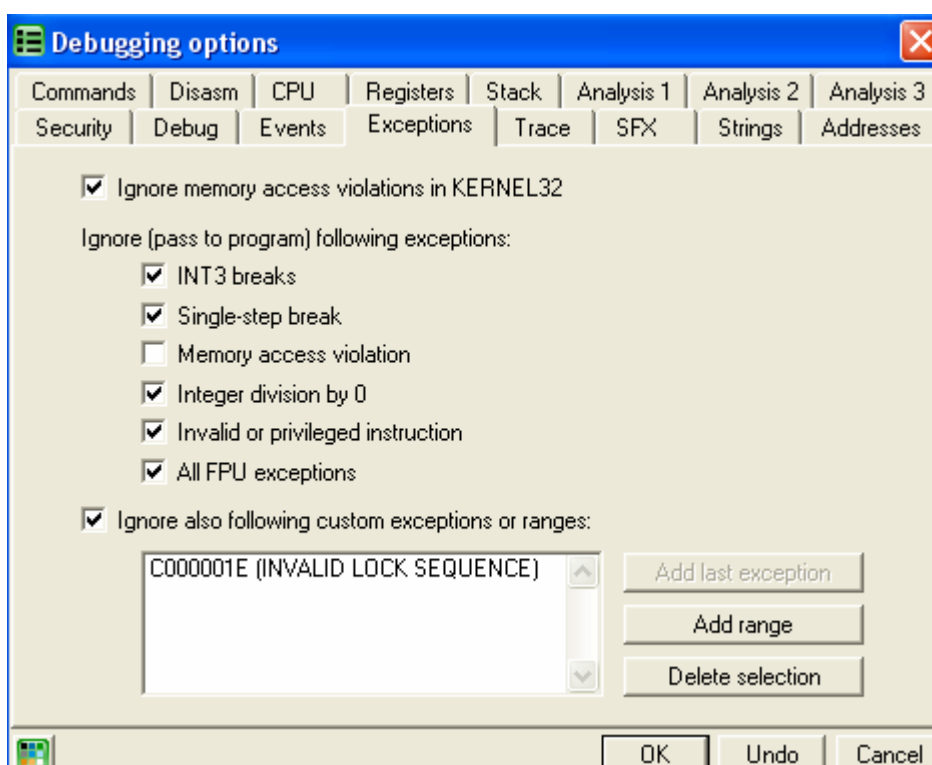
قاعده کلی Virtual Machine در این است که تعدادی از Function ها و کدهای برنامه توسط پروتکتور شبیه سازی (Emulate) می‌شود. در Virtual Machine قسمتی از کدهای برنامه (معمولا یک Function) از داخل سورس کد فایل استخراج می‌شود و به زبانی رمزگذاری می‌شود که فقط خود پروتکتور از آن سر در می‌آورد.

این قسمت از کدهای برنامه رمزگذاری شده می‌ماند، تا زمانی که برنامه نیاز به این کدها پیدا کند، در این موقع پروتکتور رمز کدها را باز می‌کند تا برنامه بتواند از کدهای آن قسمت استفاده کند.

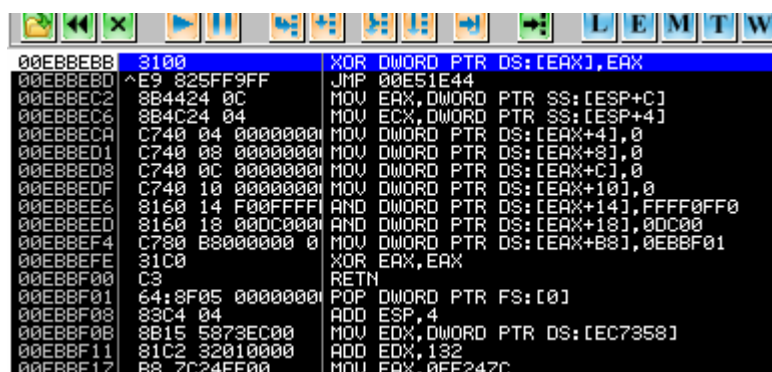
برای بررسی Virtual Machine من یک فایل را با Enigma Protector (تمامی پروتکشن‌های آن را غیر فعال کردم، فقط یکی از Function ها را با Virtual Machine، Emulate کردم ☺)

خوب، فایل مورد نظر را در OllyDBG (ترجیحا OllyICE) باز کنید. (البته برای آنالیز Virtual Machine معمولا از یک دیس اسمبلر مثل IDA استفاده می‌کنند. ولی ما اینجا از Olly استفاده کردیم)

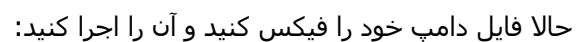
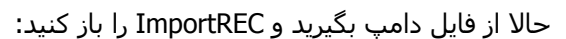
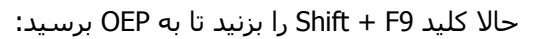
برای یافتن OEP می‌توانیم از Memory Breakpoint استفاده کنیم. ابتدا باید آخرین خطایی که در برنامه اتفاق می‌افتد را پیدا کنیم. کلیدهای ALT + O را بزنید و قسمت Exceptions را همانند تصویر زیر تنظیم کنید (فقط تیک گزینه Memory access violation را بردارید):

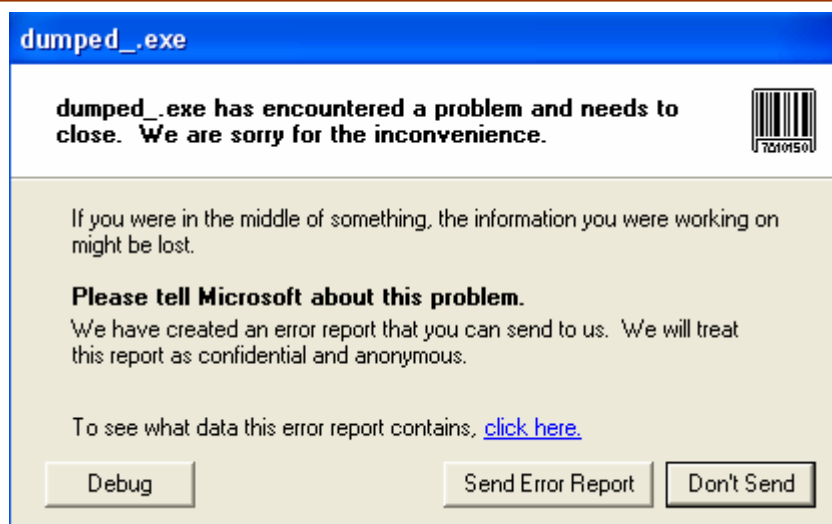


حالا برنامه را با F9 اجرا کنید و آنقدر با Shift + F9 برنامه را ادامه دهید، تا به آخرین خطای برنامه برسید:



خوب، حالا کلیدهای ALT + M را بزنید. و بر روی سکشنی محتوی code است، Memory BP بگذارید:

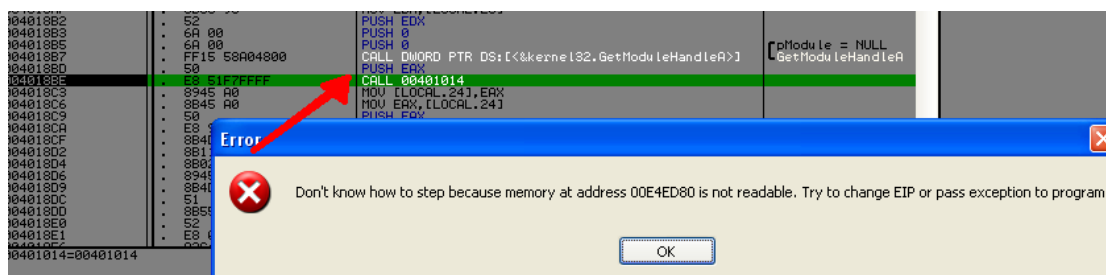




⊗ خوب، معمولاً در این موارد باید بررسی کنیم ببینیم چرا این فایل اجرا نمی شود؟... در چه خطی خطا می دهد؟... پس فایل آنپک شده را در OllyDBG (آن که فایل پک شده را در آن دیباگ کرده بودید، نبندید) باز کنید:

Address	Hex dump	Disassembly
004017B0 <ModuleEn	55	PUSH EBP
004017B1	8BEC	MOV EBP,ESP
004017B3	6A FF	PUSH -1
004017B5	68 10014200	PUSH 00420110
004017B8	68 94334000	PUSH 00403394
004017BF	64:A1 00000000	MOV EAX,DWORD PTR FS:[0]
004017C5	50	PUSH EAX
004017C6	64:8925 00000000	MOV DWORD PTR FS:[0],ESP
004017CD	83C4 A4	ADD ESP,-5C
004017D0	53	PUSH EBX
004017D1	56	PUSH ESI
004017D2	57	PUSH EDI
004017D3	8965 E8	MOV [LOCAL.6],ESP
004017D6	FF15 64A04800	CALL DWORD PTR DS:[<&kernel32.Get
004017DC	A3 D4364200	MOV DWORD PTR DS:[4236D4],EAX
004017E1	A1 D4364200	MOV EAX,DWORD PTR DS:[4236D4]
004017E6	C1E8 08	SHR EAX,8

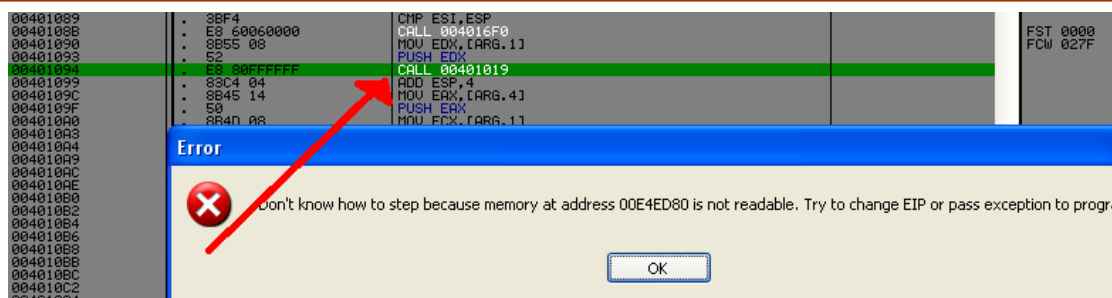
برنامه را با F8 ادامه دهید تا ببینیم در کجا برنامه دچار خطا می شود؟



می بینید؟... وقتی برنامه به خط 4018BE رسید، و آن خط اجرا شد، برنامه با خطا مواجه شد. پس خطا باید در داخل این CALL باشد. پس بر روی این خط BP گذاشته و برنامه را ری استارت می کنیم و دوباره با F9 اجرا می کنیم تا به این خط برسیم:

004018B5	6A 00	PUSH 0
004018B7	FF15 58A04800	CALL DWORD PTR DS:[<&kernel32.GetModuleHandleA]
004018BD	50	PUSH EAX
004018BE	E8 51F7FFFF	CALL 00401014
004018C3	8945 A0	MOV [LOCAL.24],EAX
004018C6	8B45 A0	MOV EAX,[LOCAL.24]
004018C9	50	PUSH EAX
004018CA	E8 91880000	CALL 00402160
004018CF	8B4D EC	MOV ECX,[LOCAL.5]
004018D2	8B11	MOV EDX,DWORD PTR DS:[
004018D4	8B02	MOV EAX,DWORD PTR DS:[
004018D6	8945 00	MOV [LOCAL.24],EAX

خوب، ما می دانیم خطا در داخل این CALL است. پس F7 را می زنیم و به داخل این CALL برویم و بعد با F8 ادامه می دهیم تا محل خطا را پیدا کنیم:



خوب...پس خطا در داخل این CALL اتفاق افتاده...پس بر روی این خط BP می گذاریم و برنامه را ری استارت کرده و دوباره با F9 برنامه را به اینجا می رسانیم. بعد F7 را می زنیم تا به داخل این CALL برویم:

00401004	CC	INT3
00401005	E9 86050000	JMP 004015C0
00401006	\$v E9 81020000	JMP 00401290
00401007	\$v E9 4C030000	JMP 00401360
00401008	\$v E9 27000000	JMP 00401340
00401019	\$v E9 72010000	JMP 00401190
0040101E	CC	INT3
0040101F	CC	INT3
00401020	CC	INT3
00401021	CC	INT3
00401022	CC	INT3
00401023	CC	INT3
00401024	CC	INT3
00401025	CC	INT3
00401026	CC	INT3

یکبار دیگر F8 را بزنید(و آنالیز OllyDBG را فایل بردارید) :

0040118B	CC	INT3
0040118C	CC	INT3
0040118D	CC	INT3
0040118E	CC	INT3
0040118F	CC	INT3
00401190	68 99F25C37	PUSH 375CF299
00401195	E9 E6DBA400	JMP 00E4ED80
0040119A	76 DD	JBE SHORT 00401179
0040119C	8F	???
0040119D	0A21	OR AH,BYTE PTR DS:[ECX]
0040119F	17	POP SS
004011A0	AD	LODS DWORD PTR DS:[ESI]
004011A1	36:6A DC	PUSH -24
004011A4	96	XCHG EAX,ESI
004011A5	9D	POPF
004011A6	46	INC ESI
004011A7	68 61DE8907	PUSH 789DE61
004011AC	27	DAA
004011AD	87DA	XCHG EDX,EBX
004011AF	62D4	BOUND EDX,ESP
004011B1	FD	STD

خطا در خط بعدی یعنی در 401195 اتفاق می افتد ☹ از کجا فهمیدم?...خب معلومه...چون نوشته شده است JMP 00E4ED80 و... اصلا آدرس 00E4ED80 در داخل فایل وجود ندارد. لذا فایل در این خط دچار خطا می شود...خوب، حالا بذارید ببینیم چرا وقتی فایل پک شده بود، خطا اتفاق نیفتاد و فایل اجرا شد?...پس در فایل پک شده (آن Olly را که نیستید؟) به این خط بروید:

0040118C	CC	INT3
0040118D	CC	INT3
0040118E	CC	INT3
0040118F	CC	INT3
00401190	E9 A69A5D55	PUSH 555D9AA6
00401195	E9 E6DBA400	JMP 00E4ED80
0040119A	76 DD	JBE SHORT Protecte.00401179
0040119C	8F	???
0040119D	0A21	OR AH,BYTE PTR DS:[ECX]
0040119F	17	POP SS
004011A0	AD	LODS DWORD PTR DS:[ESI]
004011A1	36:6A DC	PUSH -24
004011A4	96	XCHG EAX,ESI
004011A5	9D	POPF
004011A6	46	INC ESI
004011A7	68 61DE8907	PUSH 789DE61
004011AC	27	DAA
004011AD	87DA	XCHG EDX,EBX
004011AF	62D4	BOUND EDX,ESP
004011B1	FD	STD

در این خط BP بگذارید و برنامه(فایل پک شده) را با F9 اجرا کنید تا در اینجا متوقف شویم، حالا کلید F8 را بزنید:

00E4ED80	60	PUSHAD
00E4ED81	8B4424 20	MOV EAX,DWORD PTR SS:[ESP+20]
00E4ED85	50	PUSH EAX
00E4ED86	E8 09000000	CALL 00E4ED94
00E4ED8B	894424 20	MOV DWORD PTR SS:[ESP+20],EAX
00E4ED8F	61	POPAD
00E4ED90	C3	RETN
00E4ED91	8D40 00	LEA EAX,DWORD PTR DS:[EAX]
00E4ED94	55	PUSH EBP
00E4ED95	8BEC	MOV EBP,ESP
00E4ED97	83C4 F0	ADD ESP,-10
00E4ED9A	53	PUSH EBX
00E4ED9B	56	PUSH ESI
00E4ED9C	57	PUSH EDI
00E4ED9D	68 80D6ED00	PUSH 0EDD680
00E4EDA2	8D55 F0	LEA EDX,DWORD PTR SS:[EBP-10]
00E4EDA5	8D45 08	LEA EAX,DWORD PTR SS:[EBP+8]
00E4EDA8	B9 04000000	MOV ECX,4
00E4EAD0	E8 424FFFFF	CALL 00E43CF4
00E4EDB2	A1 A0D6ED00	MOV EAX,DWORD PTR DS:[EDD6A0]
00E4EDB7	E8 FC60FEFF	CALL 00E34EB8
00E4EDBC	8D00	MOV EDX,EAX
00E4EDBE	4A	DEC EDX
00E4EDBF	85D2	TEST EDX,EDX
00E4EDC1	7C 10	JL SHORT 00E4EDE0
00E4EDC3	42	INC EDX
00E4EDC4	33C0	XOR EAX,EAX
00E4EDC6	8BC8	MOV ECX,EAX
00E4EDC8	03C9	ADD ECX,ECX
00E4EDCA	8B35 A0D6ED00	MOV ESI,DWORD PTR DS:[EDD6A0]
00E4EDCB	8B3CCE	MOV ECX,DWORD PTR DS:[ESI+ECX*8]

می بینید؟...در فایل پک شده یک همچنین آدرسی وجود دارد. کدهایی که در مربع قرار دارند، کدهایی هستند که رمز Function را باز می کنند...بعد از اینکه کدها به طور کامل Decrypt شد، برنامه به خطی می رود که کدهای Decrypt شده در آنجا قرار دارد. پس چند بار کلید F8 را بزنید تا به RETN برسید:

00E4ED85	50	PUSH EAX
00E4ED86	E8 09000000	CALL 00E4ED94
00E4ED8B	894424 20	MOV DWORD PTR SS:[ESP+20],EAX
00E4ED8F	61	POPAD
00E4ED90	C3	RETN
00E4ED91	8D40 00	LEA EAX,DWORD PTR DS:[EAX]
00E4ED94	55	PUSH EBP
00E4ED95	8BEC	MOV EBP,ESP
00E4ED97	83C4 F0	ADD ESP,-10
00E4ED9A	53	PUSH EBX
00E4ED9B	56	PUSH ESI
00E4ED9C	57	PUSH EDI
00E4ED9D	68 80D6ED00	PUSH 0EDD680
00E4EDA2	8D55 F0	LEA EDX,DWORD PTR SS:[EBP-10]
00E4EDA5	8D45 08	LEA EAX,DWORD PTR SS:[EBP+8]

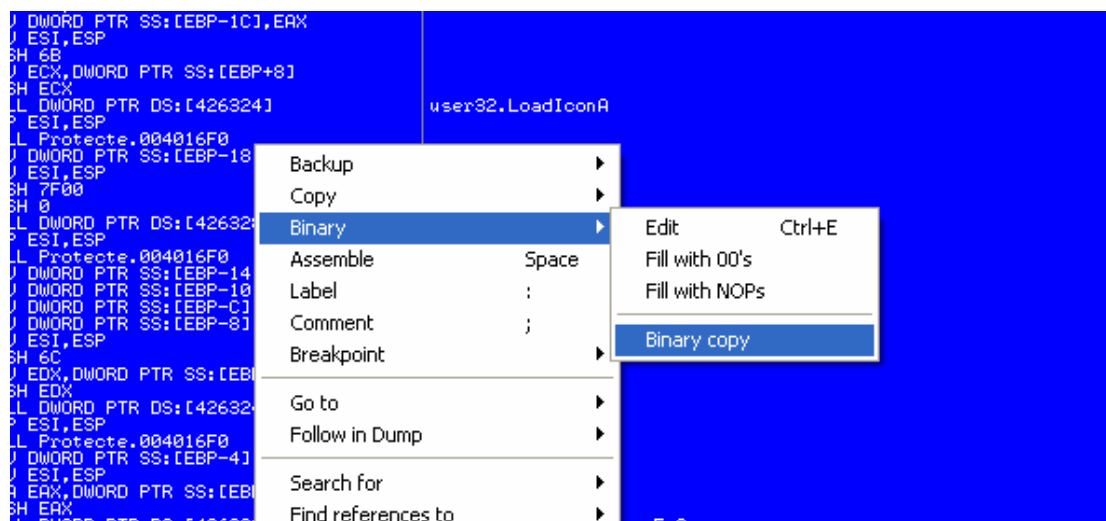
حالا یکبار F8 بزنید:

```

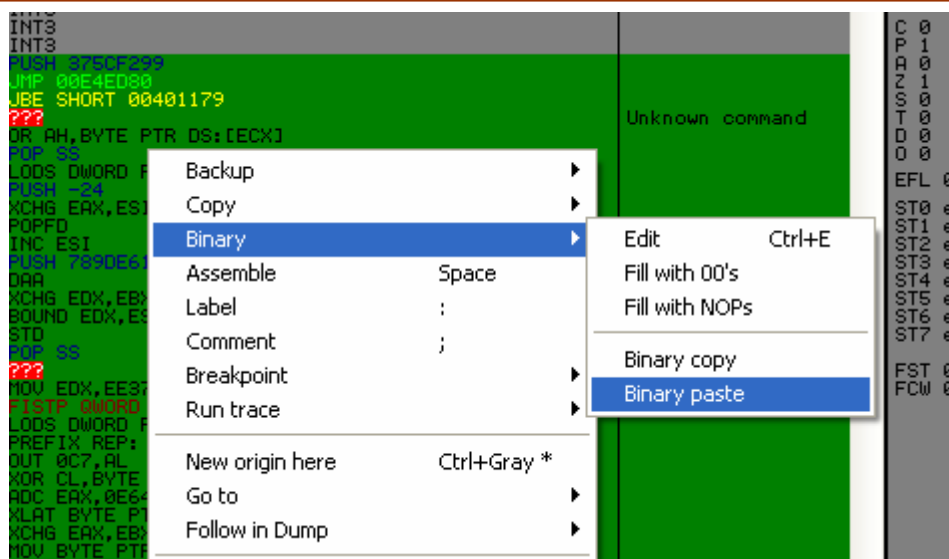
00F305EC 55          PUSH EBP
00F305ED 8BEC       MOV EBP,ESP
00F305EF 83EC 70    SUB ESP,70
00F305F2 53         PUSH EBX
00F305F3 56         PUSH ESI
00F305F4 57         PUSH EDI
00F305F5 8D7D 90    LEA EDI,DWORD PTR SS:[EBP-70]
00F305F8 B9 1C000000 MOV ECX,1C
00F305FD B8 CCCCCCCC MOV EAX,CCCCCCCC
00F30602 F3:AB     REP STOS DWORD PTR ES:[EDI]
00F30604 C785 D0FFFFFF 3 MOV DWORD PTR SS:[EBP-30],30
00F3060E C785 D4FFFFFF 0 MOV DWORD PTR SS:[EBP-2C],3
00F30618 C785 D8FFFFFF 0 MOV DWORD PTR SS:[EBP-28],40100F
00F30622 C785 DCFFFFFF 0 MOV DWORD PTR SS:[EBP-24],0
00F3062C C785 E0FFFFFF 0 MOV DWORD PTR SS:[EBP-20],0
00F30636 8B45 08    MOV EAX,DWORD PTR SS:[EBP+8]
00F30639 8985 E4FFFFFF MOV DWORD PTR SS:[EBP-1C],EAX
00F3063F 8BF4      MOV ESI,ESP
00F30641 6A 6B     PUSH 6B
00F30643 8B4D 08    MOV ECX,DWORD PTR SS:[EBP+8]
00F30646 51         PUSH ECX
00F30647 FF15 24634200 CALL DWORD PTR DS:[426324]
00F30649 8BF4      CMP ESI,ESP
00F3064F E8 9C104DFF CALL Protecte.004016F0
00F30654 8985 E8FFFFFF MOV DWORD PTR SS:[EBP-18],EAX
00F3065A 8BF4      MOV ESI,ESP
00F3065C 68 007F0000 PUSH 7F00
00F30661 6A 00     PUSH 0
00F30663 FF15 28634200 CALL DWORD PTR DS:[426328]
00F30665 8BF4      CMP ESI,ESP
00F3066B E8 80104DFF CALL Protecte.004016F0
00F30670 8985 ECFFFFFF MOV DWORD PTR SS:[EBP-14],EAX
00F30676 C785 F0FFFFFF 0 MOV DWORD PTR SS:[EBP-10],6
00F30680 C785 F4FFFFFF 6 MOV DWORD PTR SS:[EBP-C],6D
00F3068A C785 F8FFFFFF C MOV DWORD PTR SS:[EBP-8],4235C4
00F30694 8BF4      MOV ESI,ESP
00F30696 6A 6C     PUSH 6C
00F30698 8B55 E4    MOV EDX,DWORD PTR SS:[EBP-1C]
00F3069B 52         PUSH EDX
00F3069C FF15 24634200 CALL DWORD PTR DS:[426324]
00F306A2 8BF4      CMP ESI,ESP
00F306A4 E8 47104DFF CALL Protecte.004016F0
00F306A9 8985 FCFFFFFF MOV DWORD PTR SS:[EBP-4],EAX
00F306AF 8BF4      MOV ESI,ESP
00F306B1 8D45 D0    LEA EAX,DWORD PTR SS:[EBP-30]
00F306B4 50         PUSH EAX
00F306B5 FF15 2C634200 CALL DWORD PTR DS:[42632C]
00F306B8 8BF4      CMP ESI,ESP
00F306BD E8 2E104DFF CALL Protecte.004016F0
00F306C2 5F         POP EDI
00F306C3 5E         POP ESI
00F306C4 5B         POP EBX
00F306C5 83C4 70    ADD ESP,70
00F306C8 8BF4      CMP EBP,ESP
EBP=0012FF30

```

کدهایی که در اینجا قرار دارد، کدهای برنامه اصلی ما بوده... کدهایی که رمزگذاری شده و با آن چند خط رمزش باز شده... حالا برنامه می تواند از این کدها استفاده کند.
 خوب، برای حل مشکل اجرا نشدن برنامه ما می توانیم این کدها را به فایل آنپک شده خود اضافه کنیم. این کدها را انتخاب می کنید و همانند تصویر کلیک راست کرده و گزینه Binary و سپس گزینه Binary copy را می زنید:

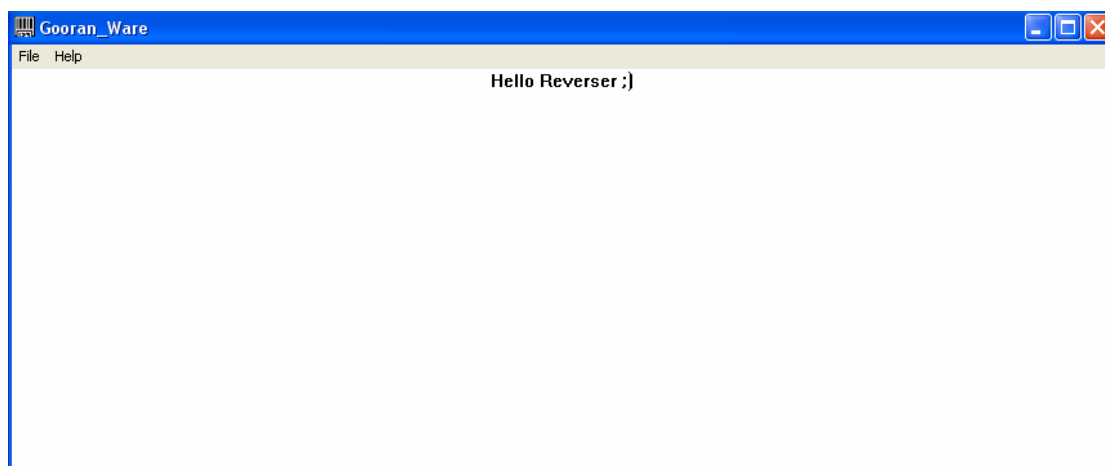


و حالا این کدها را در فایل آنپک شده Binary Paste کنید. (در خط 401190):



0040118F	CC	INT3
00401190	55	PUSH EBP
00401191	8BEC	MOV EBP,ESP
00401193	83EC 70	SUB ESP,70
00401196	53	PUSH EBX
00401197	56	PUSH ESI
00401198	57	PUSH EDI
00401199	8D7D 90	LEA EDI,DWORD PTR SS:[EBP-70]
0040119C	B9 1C000000	MOV ECX,1C
004011A1	B8 CCCCCCCC	MOV EAX,CCCCCCCC
004011A6	F3 AB	REP STOS DWORD PTR ES:[EDI]
004011A8	C785 00FFFFFF 30000000	MOV DWORD PTR SS:[EBP-30],80
004011B2	C785 04FFFFFF 03000000	MOV DWORD PTR SS:[EBP-2C],8
004011B8	C785 08FFFFFF 0F104000	MOV DWORD PTR SS:[EBP-20],0040100F
004011C6	C785 0CFFFFFF 00000000	MOV DWORD PTR SS:[EBP-24],0
004011D0	C785 00FFFFFF 00000000	MOV DWORD PTR SS:[EBP-20],0
004011DA	8B45 08	MOV EAX,DWORD PTR SS:[EBP+8]
004011DD	8985 E4FFFFFF	MOV DWORD PTR SS:[EBP-1C],EAX
004011E3	8BF4	MOV ESI,ESP
004011E5	6A 6B	PUSH 6B
004011E7	8B4D 08	MOV ECX,DWORD PTR SS:[EBP+8]
004011EA	51	PUSH ECX
004011EB	FF15 24634200	CALL DWORD PTR DS:[426324]
004011F1	3BF4	CMP ESI,ESP
004011F3	E8 D8F84EFF	CALL FF8F0AD0
004011F8	8985 E8FFFFFF	MOV DWORD PTR SS:[EBP-18],EAX
004011FE	8BF4	MOV ESI,ESP
00401200	68 007F0000	PUSH 7F00
00401205	6A 00	PUSH 0
00401207	FF15 28634200	CALL DWORD PTR DS:[426328]
0040120D	3BF4	CMP ESI,ESP
0040120F	E8 BCF84EFF	CALL FF8F0AD0
00401214	8985 ECFFFFFF	MOV DWORD PTR SS:[EBP-14],EAX
0040121A	C785 F0FFFFFF 06000000	MOV DWORD PTR SS:[EBP-10],6

خوب...حالا کدهایی رو که کپی کرده اید را با دقت نگاه کنید.چون اطلاعات Binary Copy شده،آدرس Call ها تغییر کرده.الان مثلا در تصویر بالا در خط 4011F3 نوشته شده : CALL FF8F0AD0،اما اگر به فایل پک شده نگاه کنید،می بینید نوشته شده : CALL 004016F0...پس این موارد کوچک را درست کنید و بعد کدها را یک دور با دقت نگاه کنید.کدها نباید هیچ تفاوتی با هم داشته باشند و بعد تغییرات را ذخیره کنید و این بار فایل را اجرا کنید:

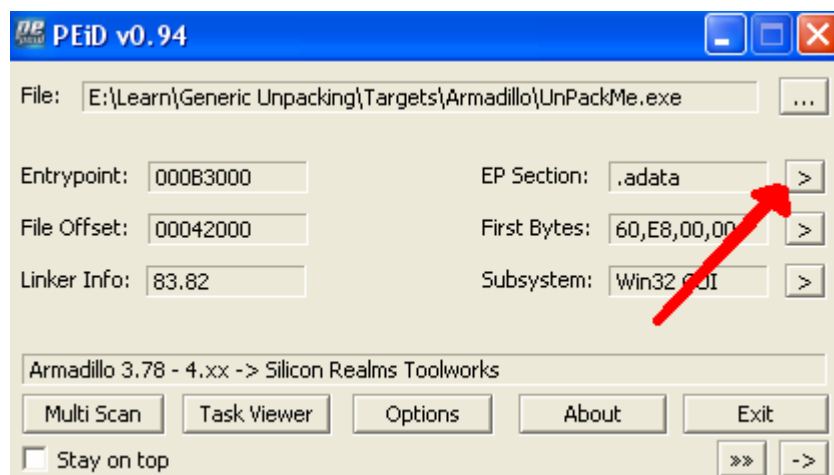


همانطور که دیدید ما یکی از Function های برنامه را که با Virtual Machine پروتکت شده بود را درست کردیم.خوب،حالا تصور کنید که یک فایل در شونصد جا این کار را کرده باشد،آنوقت باید هر شونصد جا را به همین ترتیب درست کنید ☹...فکر کنم حالا فهمیده باشید که چرا Virtual machine آنپک کردن فایل را سخت می کند.

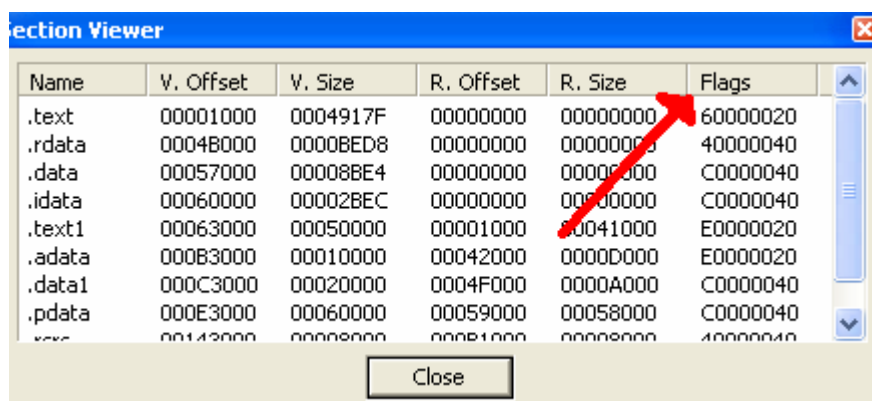
نکات ریز و درشتی که یادگیری آن لازم است

1. FLAG ها

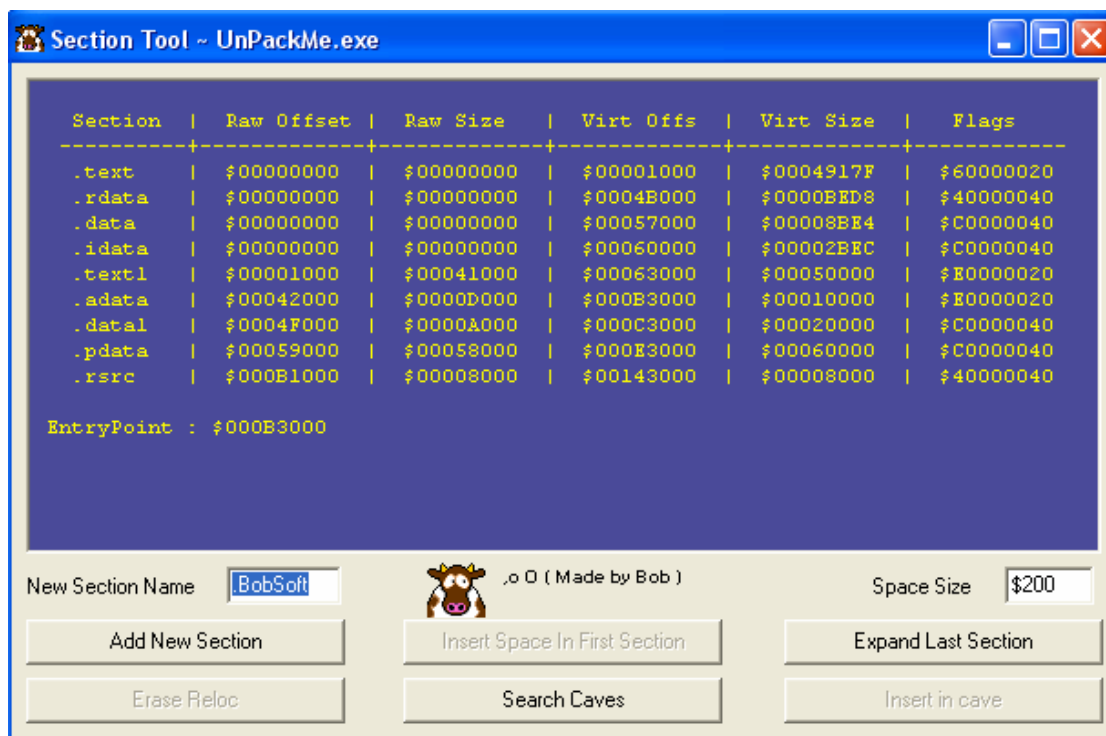
یک فایل در PEID باز کنید:



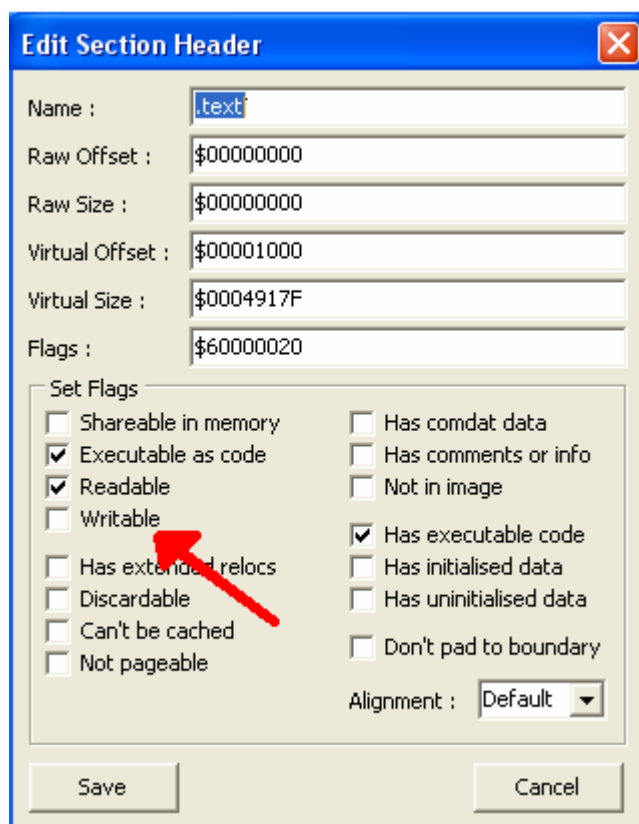
دکمه نشان داده شده را بزنید:



قبلا در مورد چهار قسمت V.Offset و ... توضیح دادم، اما در مورد Flags توضیحی ندادم. هر سگشن مشخصاتی مخصوص به خود دارد که در این مشخصات در **Flags** نشان داده می شود. این پنجره را ببندید و در قسمت Plugins گزینه Section tool را انتخاب کنید:



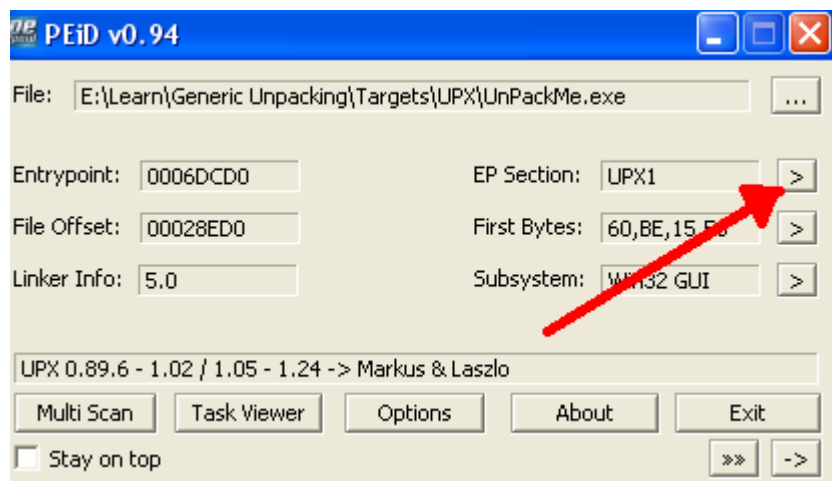
حالا بر روی هر سکشنی که کلیک راست کنید می توانید مشخصات آن سکشن را ببینید. بر مثال من بر روی سکشن .text کلیک راست می کنم:



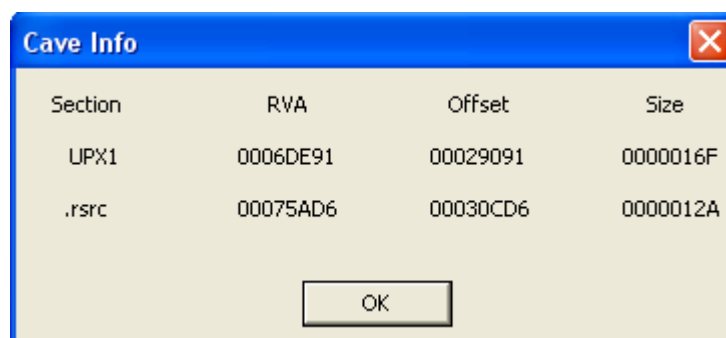
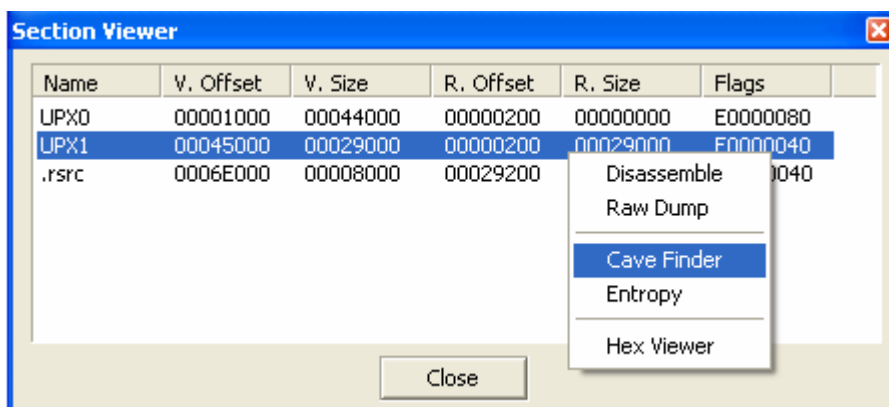
همانطور که می بینید، این سکشن Writable نیست. یعنی اگر شما در Inline Patch کدهایی بنویسید که این سکشن را تغییر دهد، برنامه دچار خطا می شود چرا که این سکشن Writable نیست. در موقع Inline Patch کردن باید حواستان به Flag سکشن مورد نظر باشد. و اگر لازم شد آن سکشن را Writable کنید. (یعنی در اینجا تیک گزینه Writable را بزنید و فایل را Save کنید.)

۲. یافتن فضای خالی برای Inline Patch کردن

در Inline Patch کردن ما به فضایی خالی نیاز داریم که کدهایمان را در آنجا تزریق کنیم. اینکه شما در OllyDBG جایی را پیدا کنید که بایت صفر وجود داشته باشد، دلیل بر این نیست که آنجا فضای خالی باشد، ممکن است آنجا فقط بایت صفر وجود داشته باشد برای اینکه سکشن پر شود. برای یافتن فضای خالی می توانیم از نرم افزار پر کاربرد PEID استفاده کنیم.



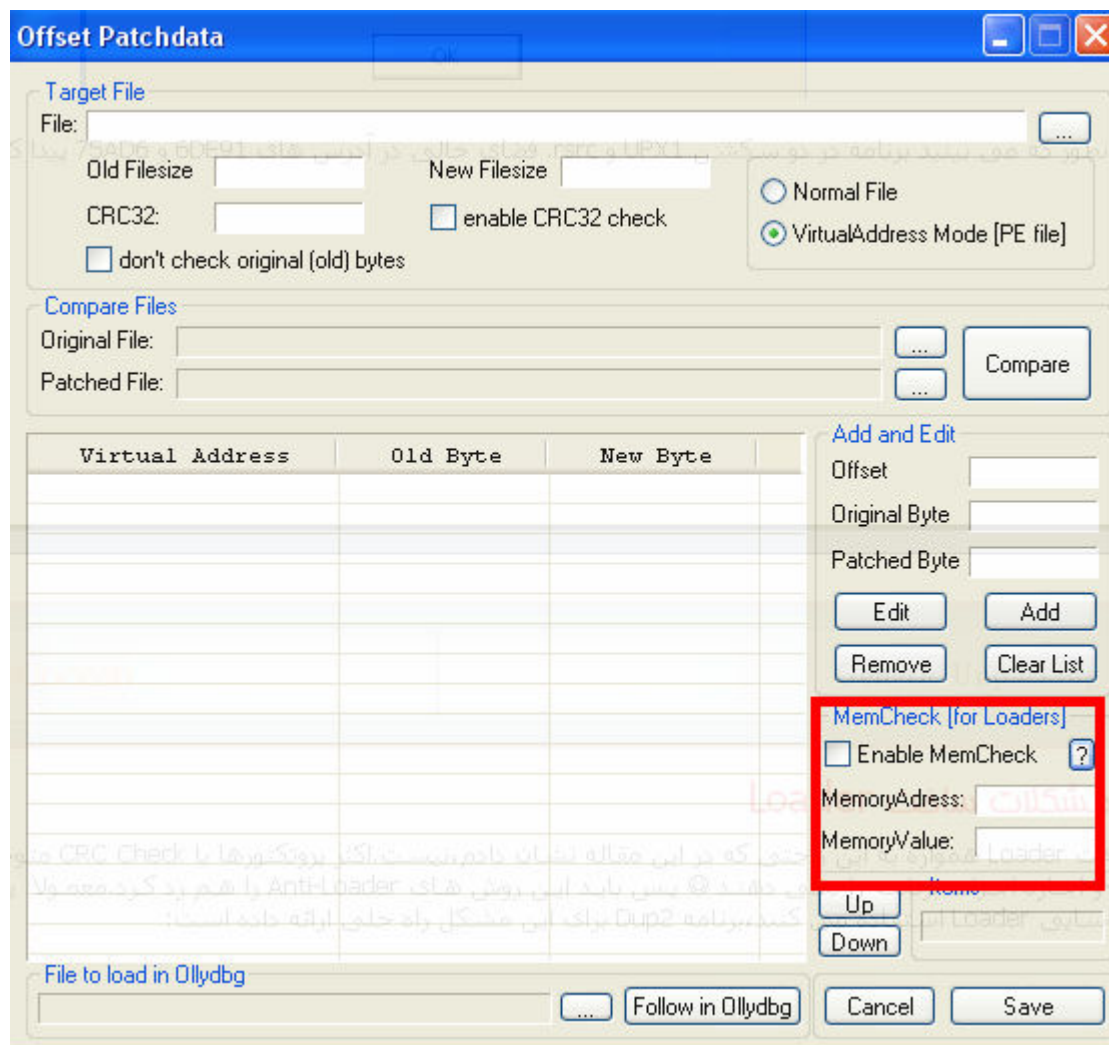
دکمه نشان داده شده را بزنید. حالا برای یافتن فضای خالی بر روی یکی از سکشن ها کلیک راست کرده و گزینه Cave Finder را بزنید:



همانطور که می بینید برنامه در دو سکشن UPX1 و .rsrc فضای خالی در آدرس های 6DE91 و 75AD6 پیدا کرده است ☺

۳. مشکلات ساخت Loader

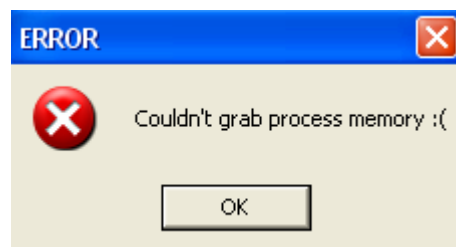
ساخت Loader همواره به این راحتی که در این مقاله نشان دادم، نیست. اکثر پروتکتورها با CRC Check متوجه وجود Loader خواهند شد و اجازه اجرای برنامه را نمی دهند ☹ پس باید این روش های Anti-Loader را هم رد کرد. معمولاً پکرها از CRC Check برای شناسایی Loader استفاده می کنند، برنامه Dup2 برای این مشکل راه حلی ارائه داده است:



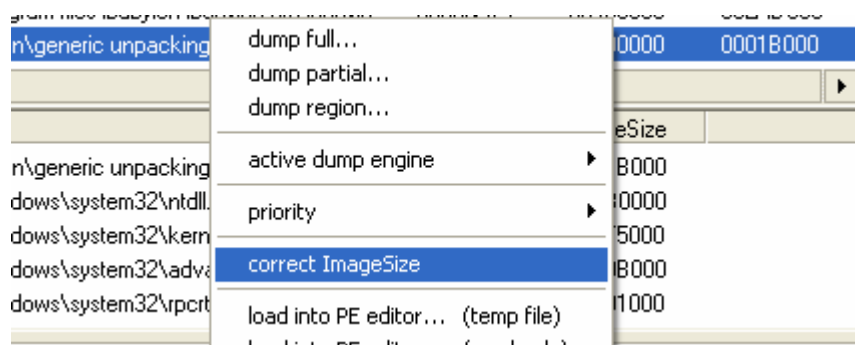
جایی که در تصویر بالا نشان دادم، قابلیت است که با آن می شود، Anti-Loader ها را رد کرد. شما باید مقدار Dword که بعد CRC Check نوشته می شود را به برنامه بدهید تا برنامه قابلیت Anti-Loader در پکر را رد کند ☺ البته ABEL Loader هم قابلیتی برای رد کردن Anti-Loader دارد. آن هم اینکه Caption برنامه هدف را به ABEL بدهیم. اگر هیچکدام از این روش ها جواب نداد، باید جایی که CRC Check انجام می شود را هم پچ کنید ☺

۴. مشکلات دامپ گرفتن از فایل

برخی از پکرها، قابلیت هایی برای سخت کردن دامپ گرفتن از برنامه دارند. مثلاً یک پکر می آید و سائز یکی از سکشن هایش (معمولاً سکشن آخری) افزایش می دهد تا نتوان یک دامپ درست از فایل گرفت:

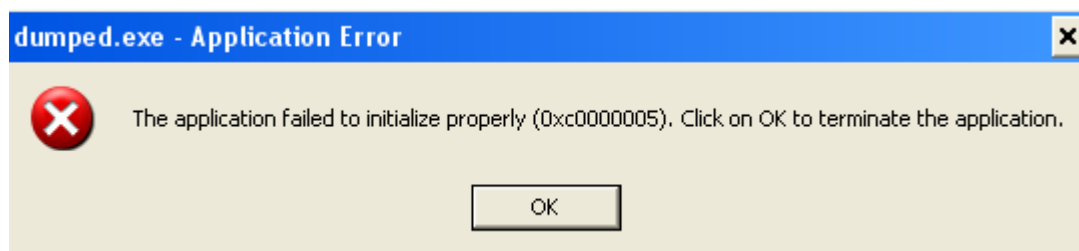


در این موارد در برنامه LordPE بر روی فایل مورد نظر کلیک راست کرده و گزینه Correct ImageSize را بزنید و دوباره برای دامپ گرفتن از فایل تلاش کنید:

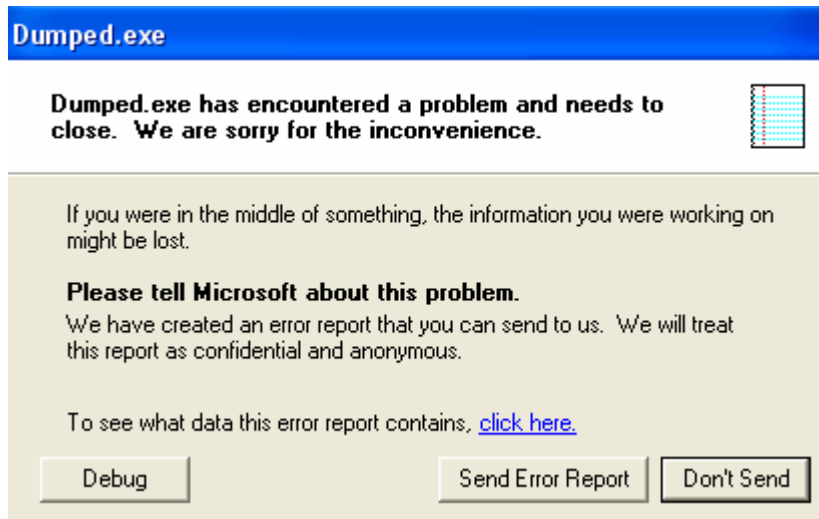


سعی کنید برای دامپ کردن فایل های DLL از OllyDump استفاده نکنید. بهتر است از LordPE یا PeTools استفاده کنید. یک چیز دیگر اینکه، سعی کنید برای دامپ کردن، برنامه های مختلفی در اختیار داشته باشید، اگر یکی جواب نداد، اون یکی جواب می ده ☺

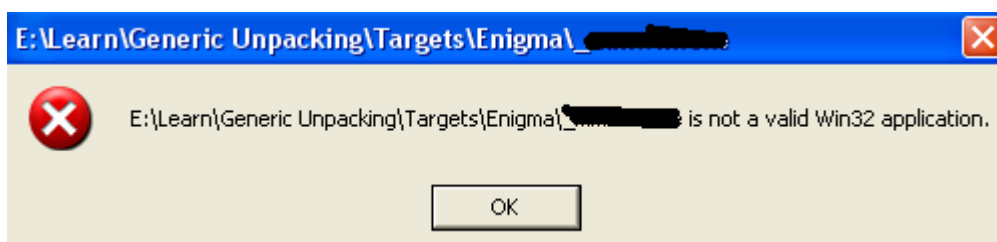
در ضمن، یک Dump سالم باید یک همچین پیغامی بدهد (به خاطر نبود Import Table همچین خطایی می دهد) :



یا Don't send بدهد:

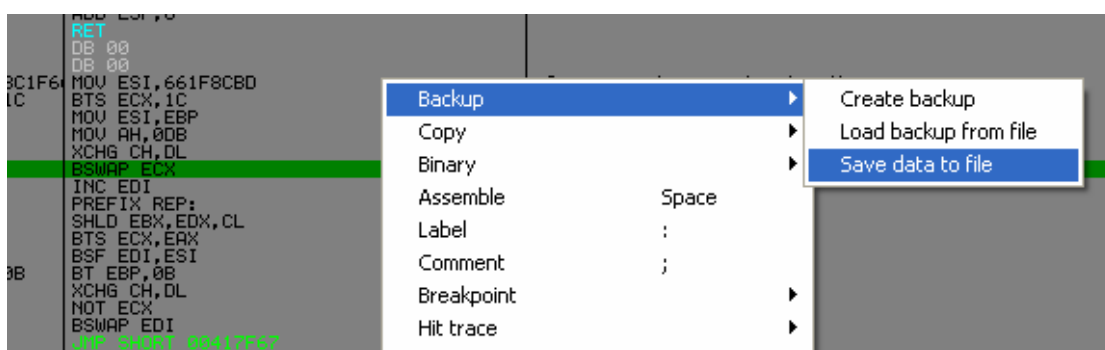


اگر فایل دامپ، همچین پیغامی بدهد یعنی فایل درست دامپ نشده:

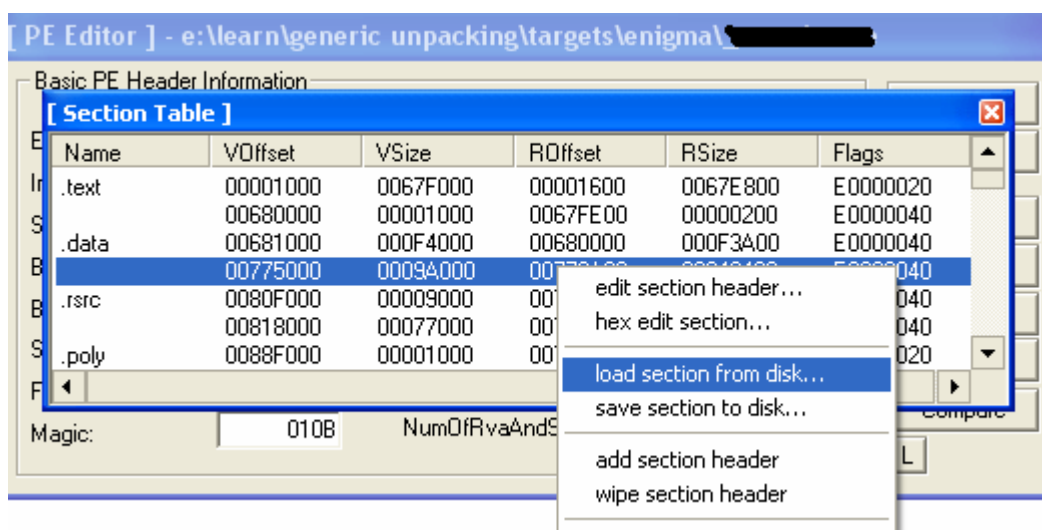


۵. اضافه کردن سکشن

گاهی اوقات یک پکر می آید و قسمتی از کدهای برنامه را منتقل می کند به سکشنی دیگر (Code Redirection)... به همین دلیل وقتی فایل را آنپک کردیم، فایل آنپک شده اجرا نمی شود. برای حل این مشکل می توانیم سکشنی که کدها به آنجا منتقل شده اند را به فایل آنپک شده اضافه کنیم ©
برای Save کردن یک سکشن در OlllyDBG کلیک راست کرده، گزینه Backup و سپس گزینه Save data to file را می زنیم:



بعد از ذخیره سکشن می توانید با LordPE آن سکشن را به برنامه اضافه کنید، برای این کار ابتدا فایل را در درون PE Editor باز می کنید و گزینه Sections را می زنید، حالا کلیک راست کرده و گزینه Load Section from disk را بزنید:



و بعد سکشنی که می خواهید اضافه کنید را به برنامه می دهید ©