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 سختی را انتخاب کردم،سعی کردم تنها تعداد کمی از Protection های آن را فعال کنم تا این مقاله برای شما زیاد سخت 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 سخت تری را آنپک می کنم.لیست نیمه کاملی از مباحثی که در قسمت دوم مطرح می شود:

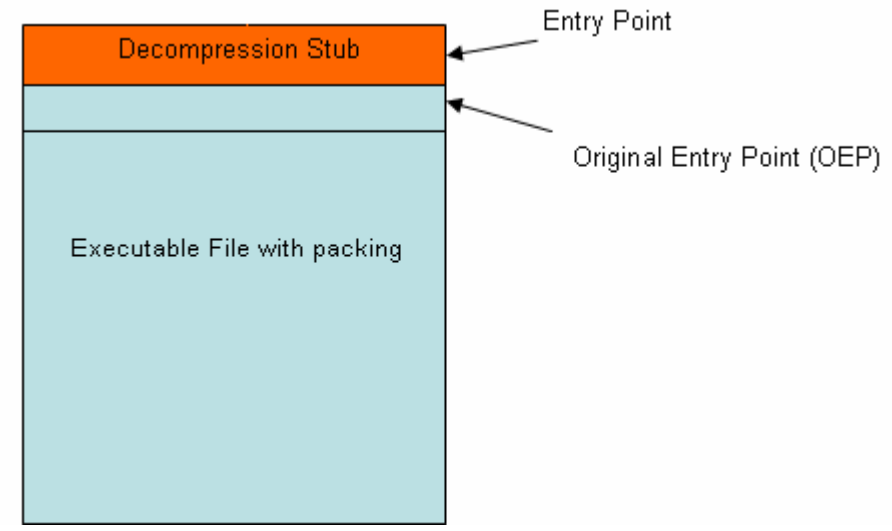
Analysising Virtual Machine in Asprotect – Unpack Armadillo with Nanomites – Unpack SDProtector 1.16 – How to write a simple unpacker in MASM – Analysis Protections in ActiveMark – Unpack Thinstall – Unpack .net Applications – How to write a plugin for ImportREC

در پایان از تمامی دوستانی که ما را در ساخت این مقاله سخت 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. mJo0T → 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 ؟

پـک کـردن یـک فایـل اجرایـي[1]،عملیـاتي اسـت کـه طـي آن یـک فایـل اجرایـي فـشرده شـده و برنامـه اي از طـرف پکـر کـه بـه آن Decompression Stub مي گوییم،به ابتداي آن تزریق مي شود.Decompression Stub که به ابتداي فایل ما تزریق مي شـود،وظیفه اجراي فایل ما را دارد.
وقتي ما فایل پک شده را اجرا مي کنیم،ابتدا Decompression Stub اجرا مي شود.فایل اصلي ما را در داخل خودش پیدا مي کنـد،و بعد شروع به باز کردن آن فایل و به اصـطلاح Decompress کـردن فایـل مـي کنـد.بعـد از اینکـه فایـل مـا بـه طـور کامـل Decompress شـد.آنوقت اختیار به دسـت فایل اصلي مي افتد و برنامه ما اجرا مي شود.این عملیات به طور کامل از دید یک کاربر معمـولي مخفـي مي ماند و کاربر فکر مي کند که این فایل همان فایل قبلي ما مي باشد.
Entry Point که مي دانید یعني چي؟...یعني جایي که برنامه ما از آنجا شروع به کار مـي کنـد.در مـورد Packer هـا مـا دو تـا Entry Point داریم.یکي Entry Point مربوط به Decompression Stub و دیگري Entry Point فایل اصلي مـا کـه بـه آن Original Entry Point مي گوییم:



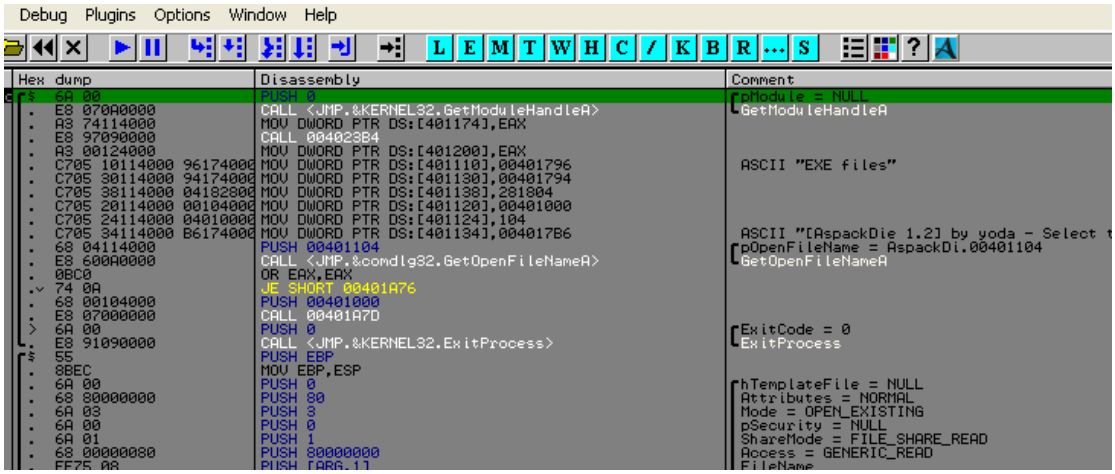
فرقي نمي کند که Packer ما چي باشد،هر چي باشد،به هر حال،در یک جا و در یک زماني،اختیار از دست Decompression Stub خارج شده و به دست فایل اصلي ما مي افتد.
اولین کاري که براي آنپک کردن باید بکنیم،این است که ببینیم در کجا و در چه خطي اختیـار از دسـت برنامـه مـا خـارج شـده و فایـل اصلي ما اجرا مي شود؟...کجا اولین دستور فایل اصلي ما اجرا مي شود؟...یا به زبان ساده،OEP فایل ما کجاست؟

[1] Executable Packing

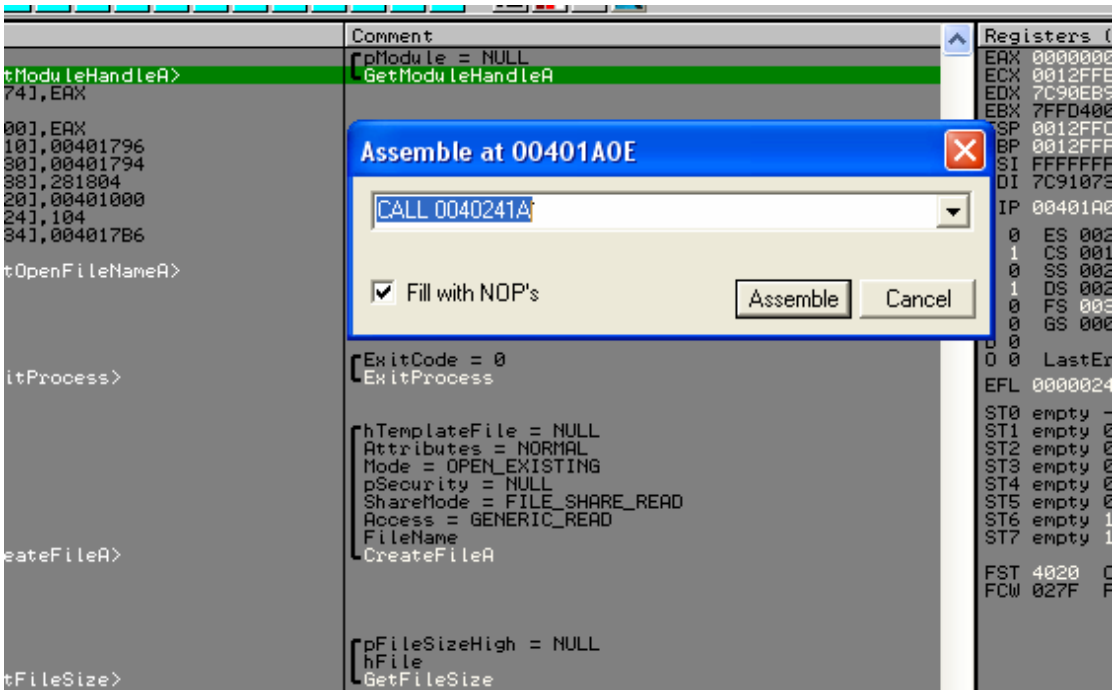
مقدمه ۲ : IAT چیست؟

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 هم گفته می شود در درون فایل های DLL ویندوز قرار دارند.اما برنامه نمی داند که این توابع در کجای فایل های DLL قرار دارد.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:

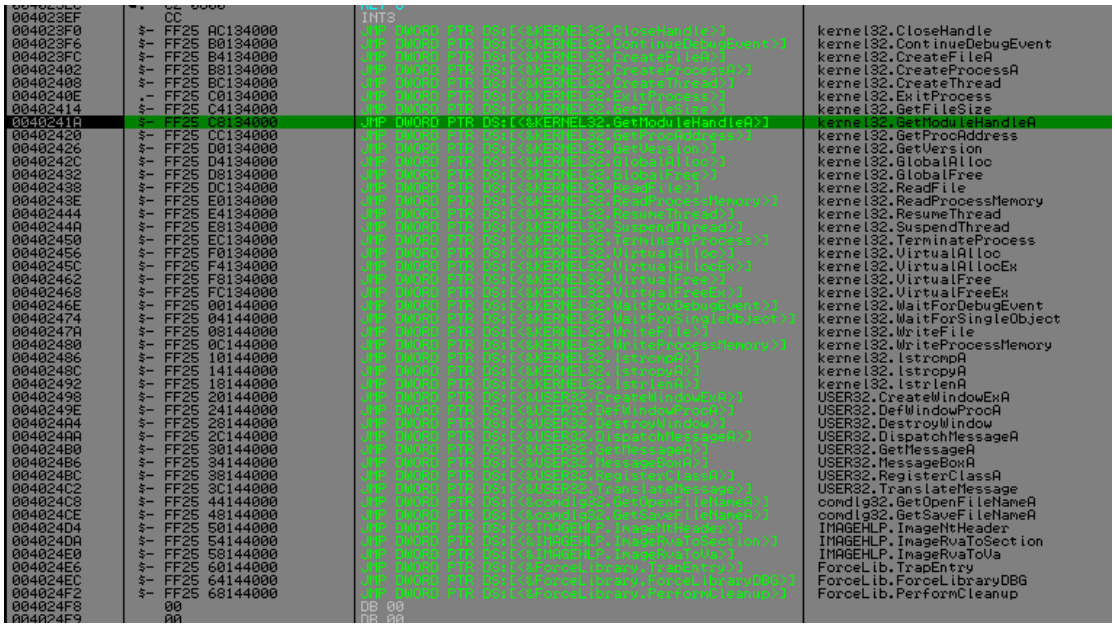
Targets/___Examples___/IAT/AspackDie.exe



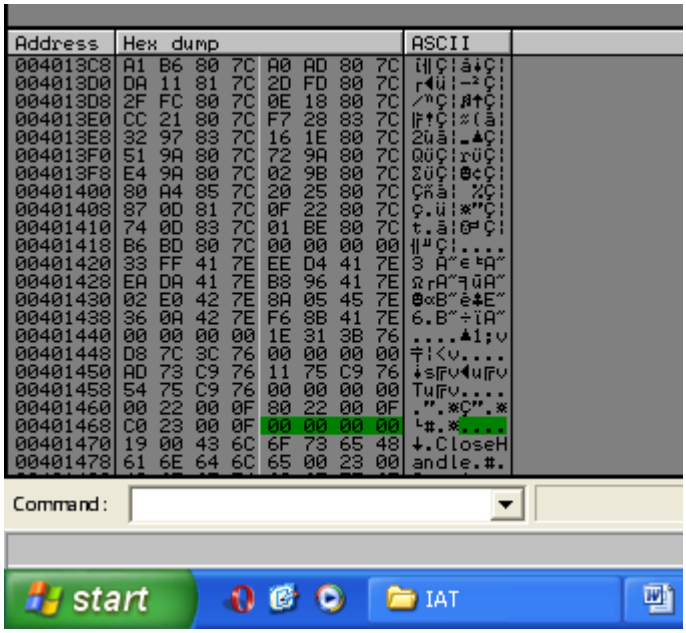
در خط دوم برنامه تابع GetModuleHandleA صدا زده شده،بذار ببینم که چه طور این کار انجام شده؟بر روی این تابع دوبار کلیک کنید:



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

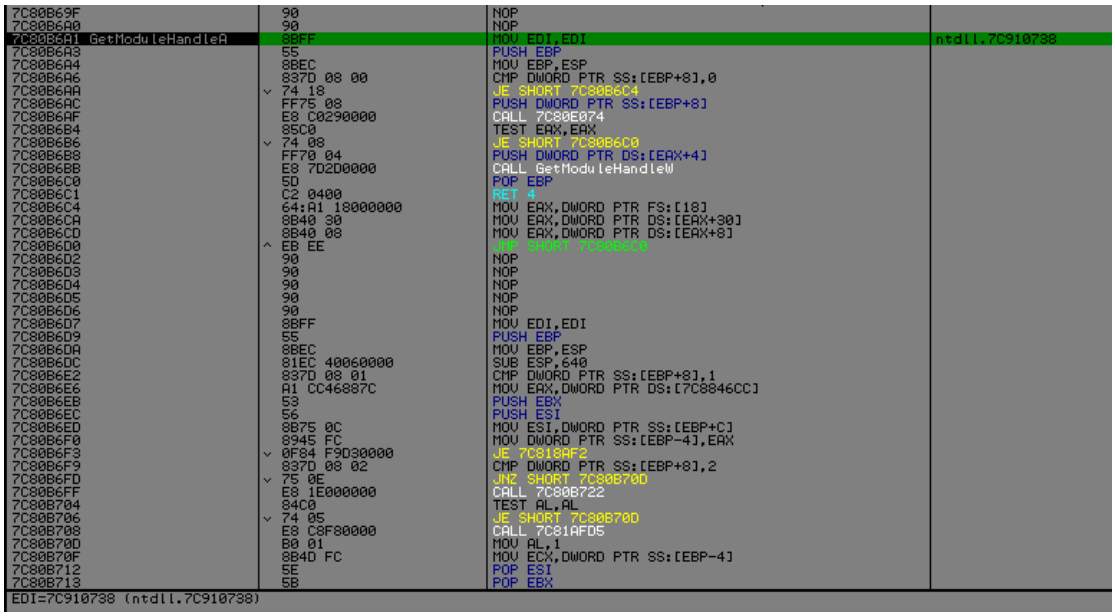


ما الان در Jump Table هستیم.تمامی توابع اول به اینجا منتقل می شوند و بعد با استفاده از این Jump ها به تابع مورد نظر می روند.البته وجود Jump Table الزامی نیست،یک برنامه می تواند مستقیم از IAT استفاده کند.حالا کافیست که فقط یکبار F7 بزنید تا وارد تابع GetModuleHandleA در فایل Kernel32.dll بشوید.خوب،برای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



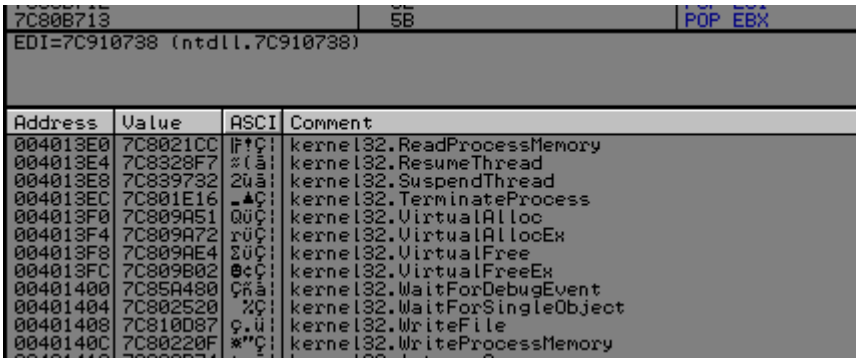
همانطور که می بینید الان محتویات پنجره Dump ما همان IAT ماست و آدرس توابع API ما در این آدرس قرار دارد.دقت کنید که در پنجره Dump اطلاعات را باید جفت جفت از آنور بخوانید،برای مثال در تصویر بالا در خط 004013C8 نوشته شده:
A1 B6 80 7C
ما باید خط بالا را اینجوری بخوانیم:
7C80B6A1

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

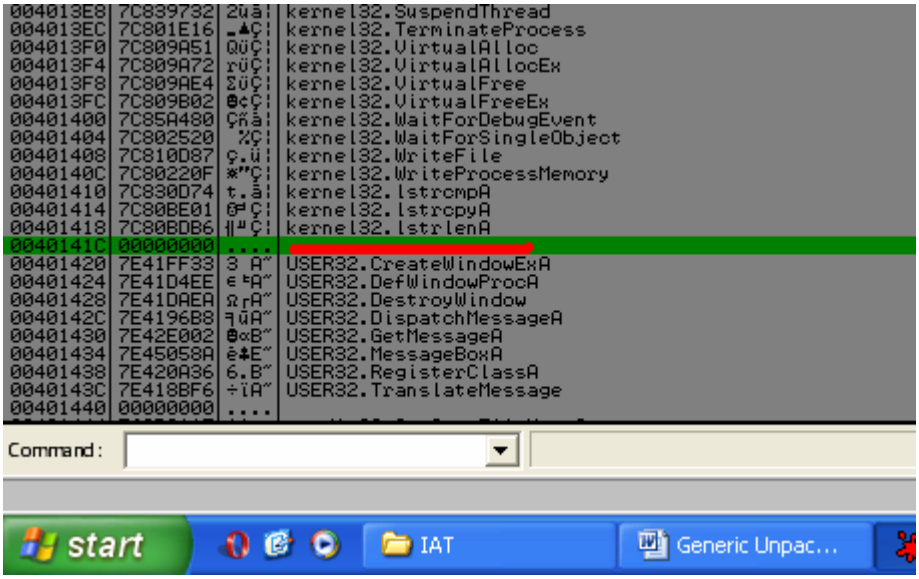


مـی بینیـد ایـن همـان تـابع GetModuleHandleA ماسـت کـه در کـامپیوتر مـن در آدرس 7C80B6A1 قـرار دارد.مطمئنـا تـابع GetModuleHandleA در سیستم شما در آدرسی متفاوت از کامپیوتر من قرار دارد.(به همین دلیل است که استفاده از IAT لازم است.)

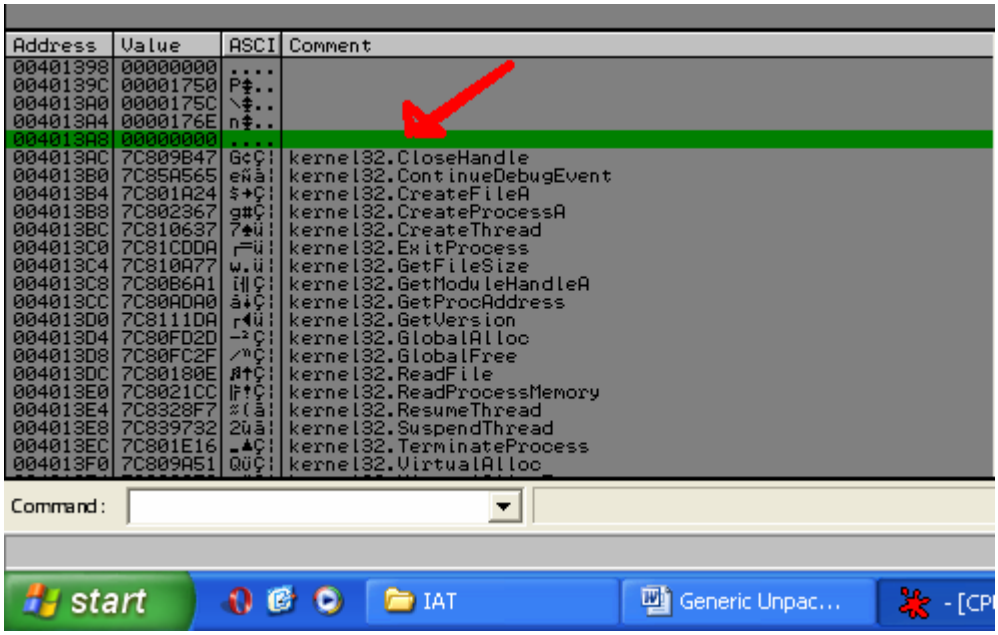
خوب،حالا می خوایم IAT را با دقت بیشتری بررسی کنیم.برای این کار بهتر است در پنجره Dump کلیک راست کرده گزینه Long و سپس گزینه Address with ASCII dump را بزنیم تا محتویات پنجره Dump را به این صورت ببینیم:



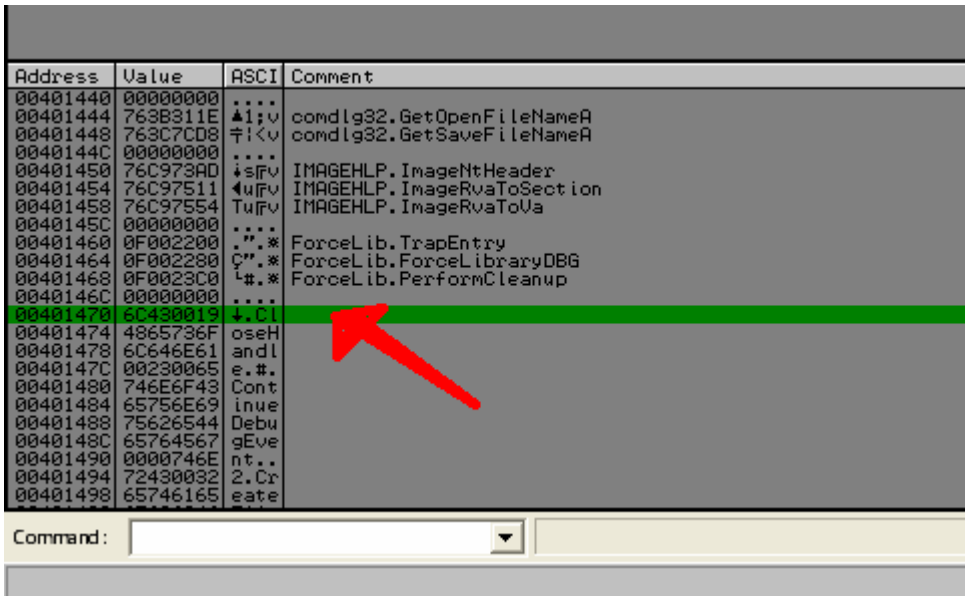
به تصویر زیر نگاه کنید،متوجه می شوید که توابع مربوط به هر DLL با یک 0000000 از هم جدا می شوند:



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



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



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

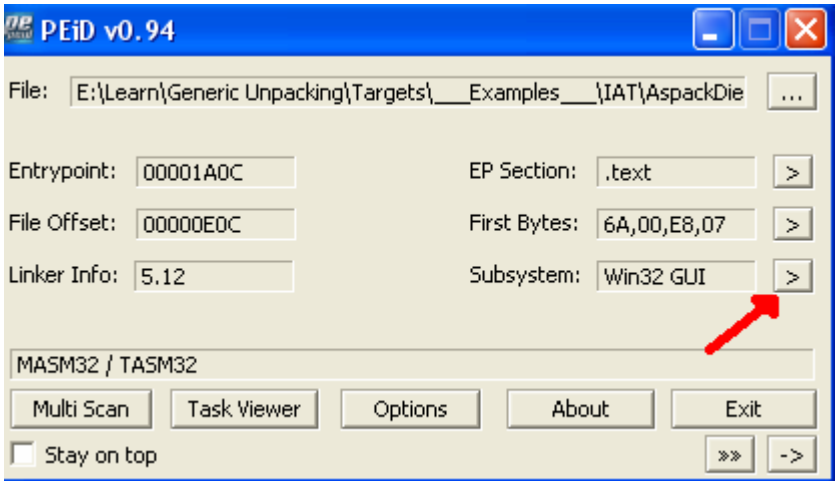
پس :

Start of IAT=4013A8
End of IAT=401470
Size of IAT=C8

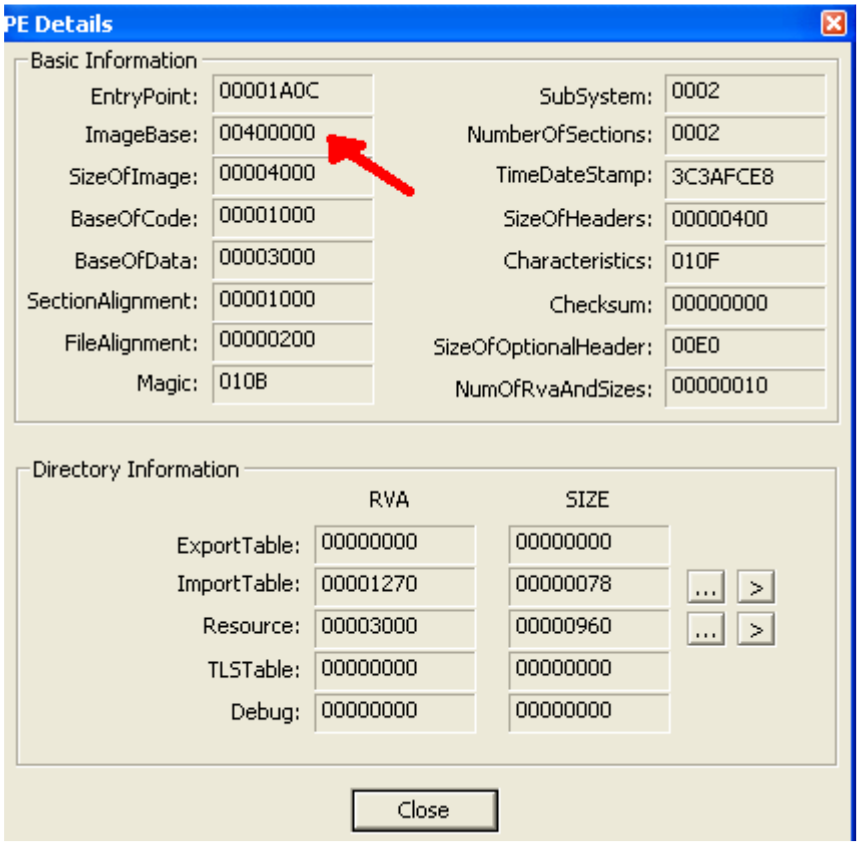
در بالا شروع IAT بر حسب VA محاسبه شده،ما باید آنها را بر مبنای RVA محاسبه کنیم.برای اینکار یک فرمول کلی داریم:

RVA=VA – ImageBase

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



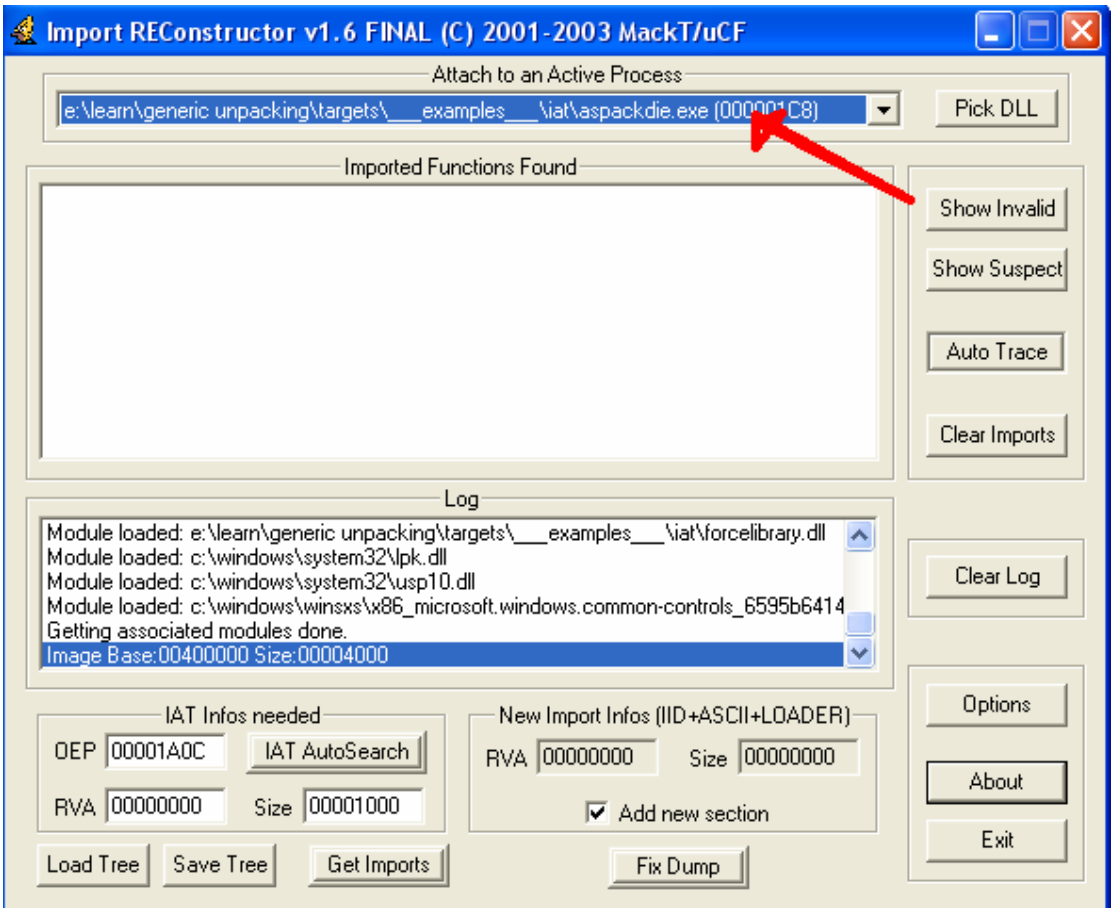
بر روی دکمه نشان داده شده کلیک کنید:



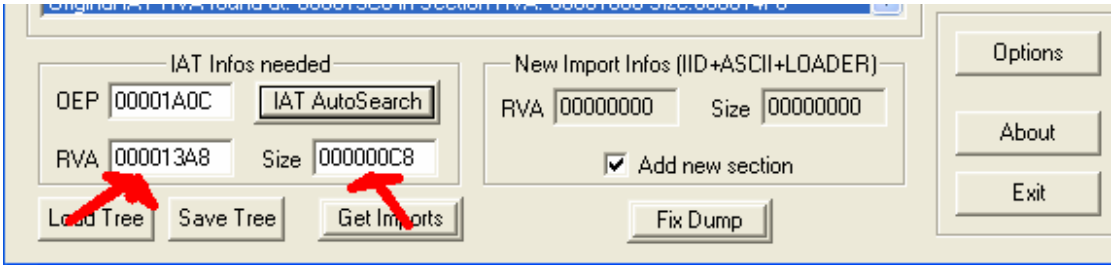
همانطور که می بینید ImageBase ما برابر 00400000 می باشد،لذا با توجه به فرمول بالا:

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

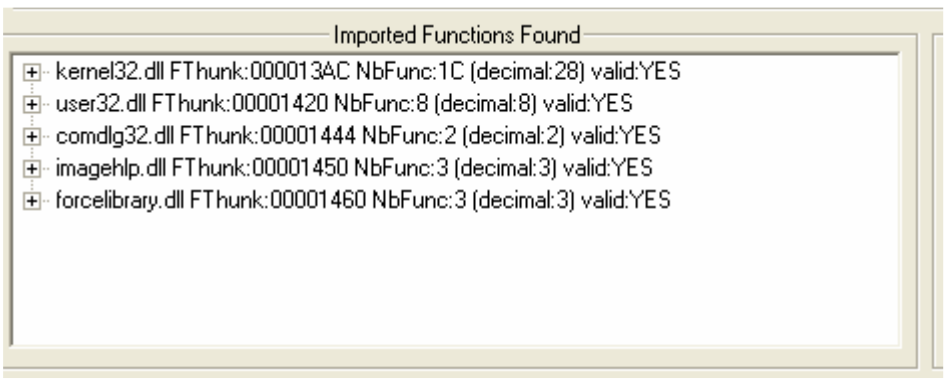
خوب،در مورد آنپک کردن فایل ما نیاز به برنامه ای داریم که IAT ما را دوباره بسازد برای این کار از ImportREC استفاده می کنیم،این برنامه را باز کنید تا اندکی با آن آشنایی پیدا کنیم:
اول از همه پروسه مورد نظرمان را انتخاب می کنیم:



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 قابلیتی 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.

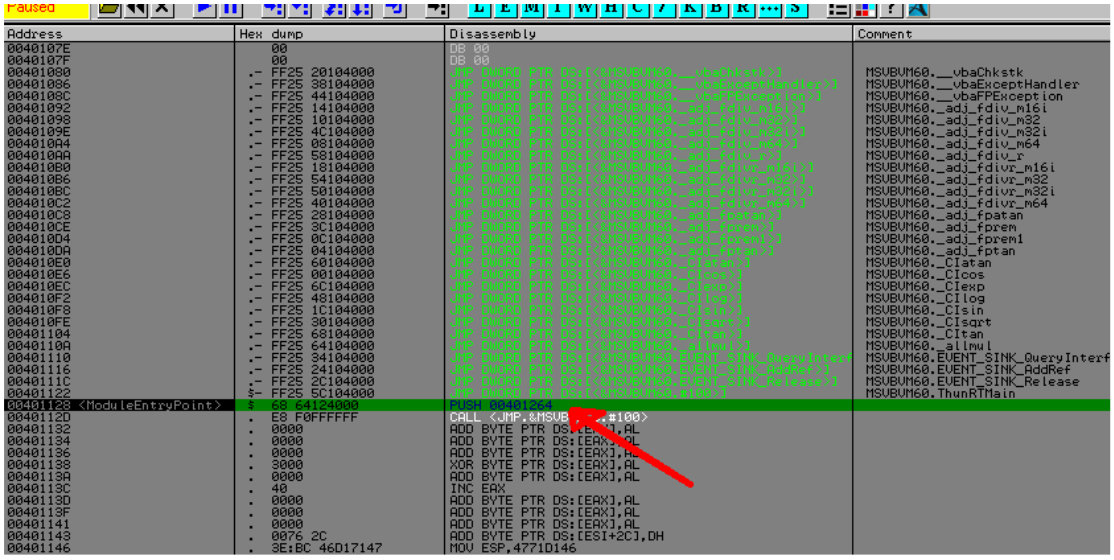
مقدمه ۳ : آشنایی اجمالی با ساختار زبان های برنامه نویسی

۳-۱ : ویژوال بیسیک

فایل زیر را در OllyDBG باز کنید:

Targets___Examples___\Language Structure\Visual Basic\VB.exe

برنامه هاي ويژوال بيسيک از توابع API به طور مستقيم استفاده نمي کنند،بلکه فايل MSVBVM60.dll به عنوان واسطه اين کار را انجام مي دهد.لذا IAT فايل در ويژوال بيسيک فقط شامل آدرس توابع فايل MSVBVM60.dll مي باشد:



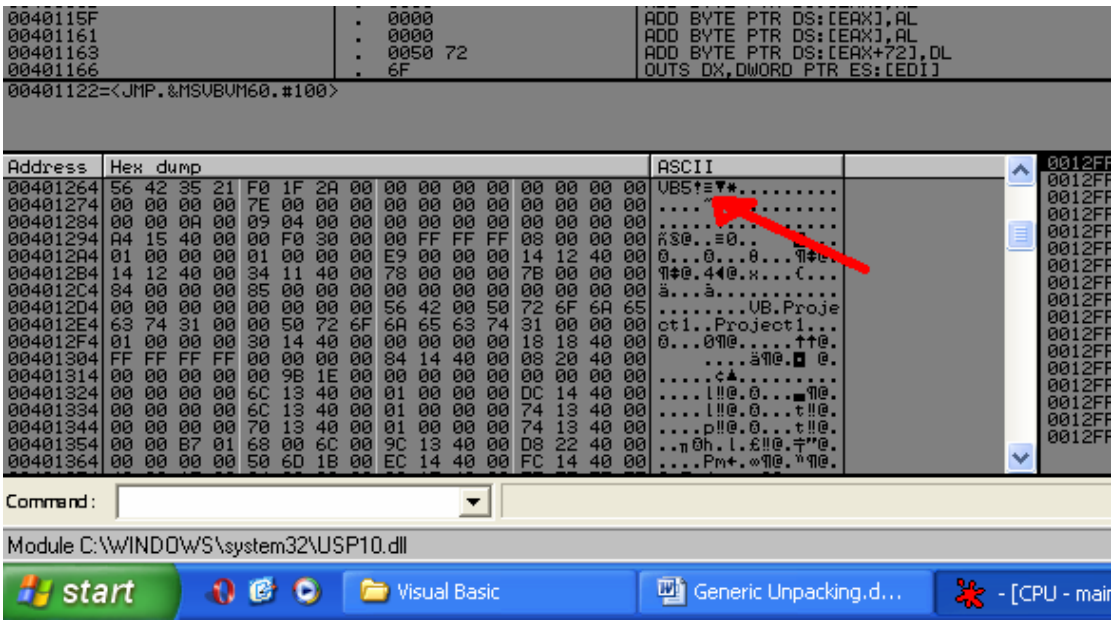
همانطور که مي بينيد در ويژوال بيسيک Entry Point فايل ما درست زير Jump Table قرار دارد.(گاهي اوقات دو بايت "00" بين Jump table و Entry Point فاصله وجود دارد.)
بعد از Entry Point دستور Call ThunRTMain وجود دارد.با اجرا شدن دستور Call ThunRTMain برنامه اجرا مي شود.در تمامي برنامه هاي ويژوال بيسيک Entry point ما يک دستور مثل اين هست:

Push xxxxxxxx

در مثال ما Entry Point برابر است با:

Push 401264

اگر در پنجره Dump به آدرس 401264 برويد،خواهيد ديد که در اين آدرس رشته "VB5!" وجود دارد.



اصولا آنپک کردن برنامه هاي ويژوال بيسيک آسانتر است،چرا که درست زير سر Entry Point، Jump Table(در مورد فايل پک شده OEP) قرار دارد.پس يافتن OEP در اينجور برنامه ها خيلي راحت تر است.از طرفي چون IAT ما فقط از MSVBVM60.dll تشکيل شده،لذا ساخت دوباره IAT هم آسانتر است.و همچنين اگر پکر ما تعدادي از بايت هاي ما را دزديده باشد،خيلي راحت تر مي شود آن بايت ها را حدس زد ☺

۳-۲ : دلفی

فایل زیر را در OllyDBG باز کنید:

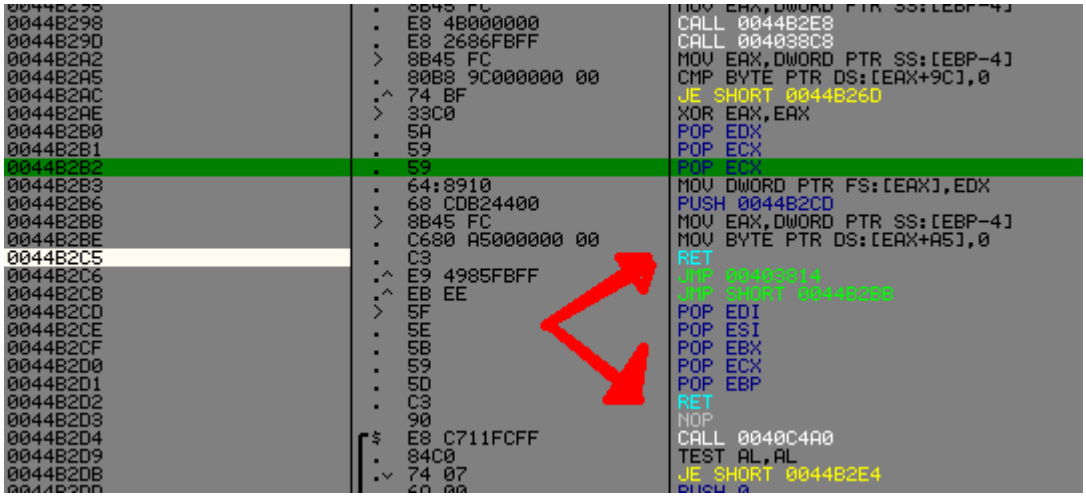
Targets___Examples___\Language Structure\Borland Delphi\Delphi.exe

```
0044CA94          90              NOP
0044CA95          C8              DB C8
0044CA96          44              DB 44
0044CA97          00              DB 00
0044CA98 <ModuleEntryPoint>  $ 55              PUSH EBP
0044CA99          .  8BEC         MOV EBP,ESP
0044CA9B          .  83C4 F0      ADD ESP,-10
0044CA9E          .  B8 B8C84400  MOV EAX,0044C8B8
0044CAA3          .  E8 2091FBFF  CALL 00405BC8
0044CAA8          .  A1 B8DF4400  MOV EAX,DWORD PTR DS:[44DFB8]
0044CAAD          .  8B00         MOV EAX,DWORD PTR DS:[EAX]
0044CAAF          .  E8 9CE6FFFF  CALL 0044B150
0044CAB4          .  8B0D 94E04400 MOV ECX,DWORD PTR DS:[44E094]
0044CABA          .  A1 B8DF4400  MOV EAX,DWORD PTR DS:[44DFB8]
0044CABF          .  8B00         MOV EAX,DWORD PTR DS:[EAX]
0044CAC1          .  8B15 F0C64400 MOV EDX,DWORD PTR DS:[44C6F0]
0044CAC7          .  E8 9CE6FFFF  CALL 0044B168
0044CACC          .  A1 B8DF4400  MOV EAX,DWORD PTR DS:[44DFB8]
0044CAD1          .  8B00         MOV EAX,DWORD PTR DS:[EAX]
0044CAD3          .  E8 10E7FFFF  CALL 0044B1E8
0044CAD8          .  E8 4372FBFF  CALL 00403D20
0044CADD          .  8D40 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
```

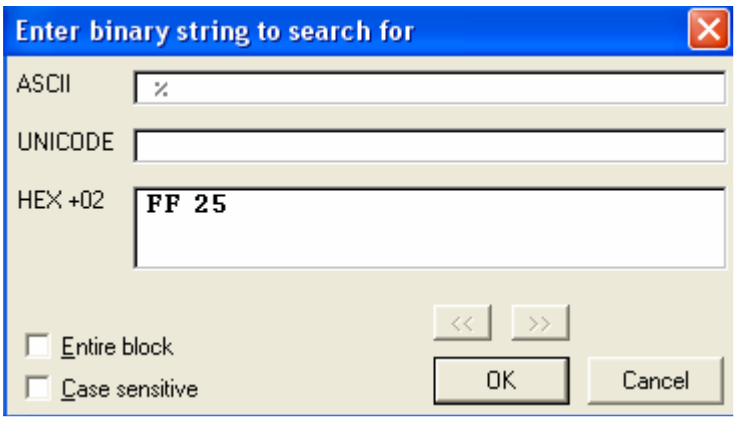
معمولا دو دستور اول در برنامه های دلفی این است:
Push EBP
MOV EBP, ESP
و چند خط بعد از آن تابع GetModuleHandleA اجرا می شود.در تصویر بالا در داخل تابعی که در خط 44CAA3 قرار دارد،تابع GetModuleHandleA قرار دارد.

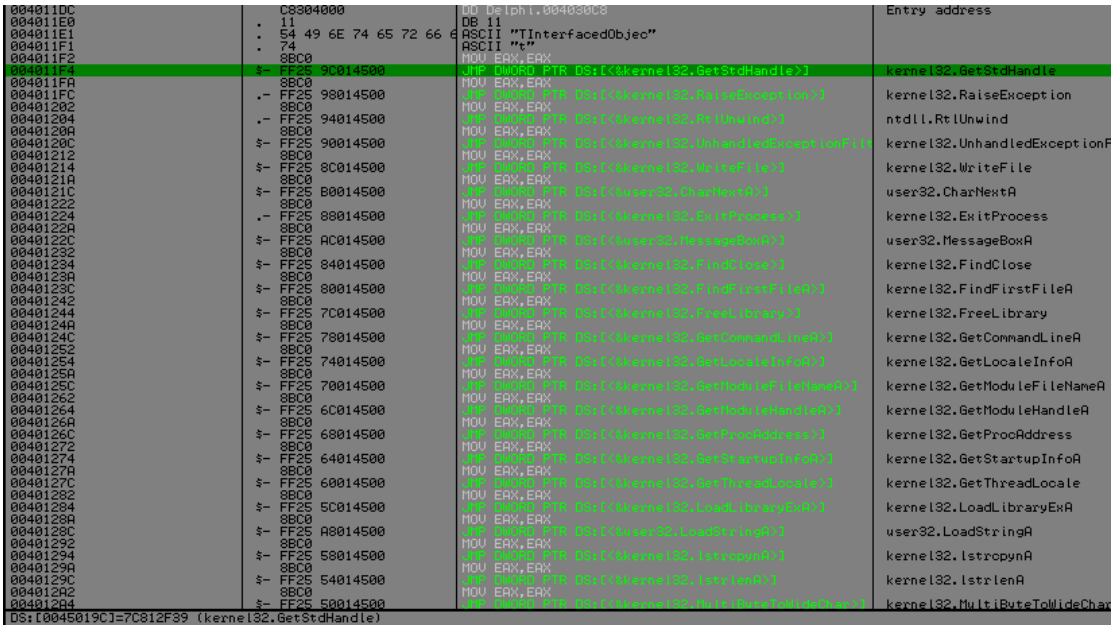
```
00405BB9          .^ 74 DB        JE SHORT 00405B96
00405BBB          .  C3           RET
00405BBC          $  B8 A4D04400  MOV EAX,0044D0A4
00405BC1          .  E8 06F8FFFF  CALL 004053CC
00405BC6          .  C3           RET
00405BC7          90              NOP
00405BC8          $  53           PUSH EBX
00405BC9          .  8BD8         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
00405BFB          .  BA A4D04400  MOV EDX,0044D0A4
00405C00          .  8BC3         MOV EAX,EBX
00405C02          .  E8 51DFFFFF  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          .  64:8920      MOV DWORD PTR FS:[EAX],ESP
00405C1D          .  FF05 68F64400 INC DWORD PTR DS:[44F668]

pModule = NULL
GetModuleHandleA
```

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



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



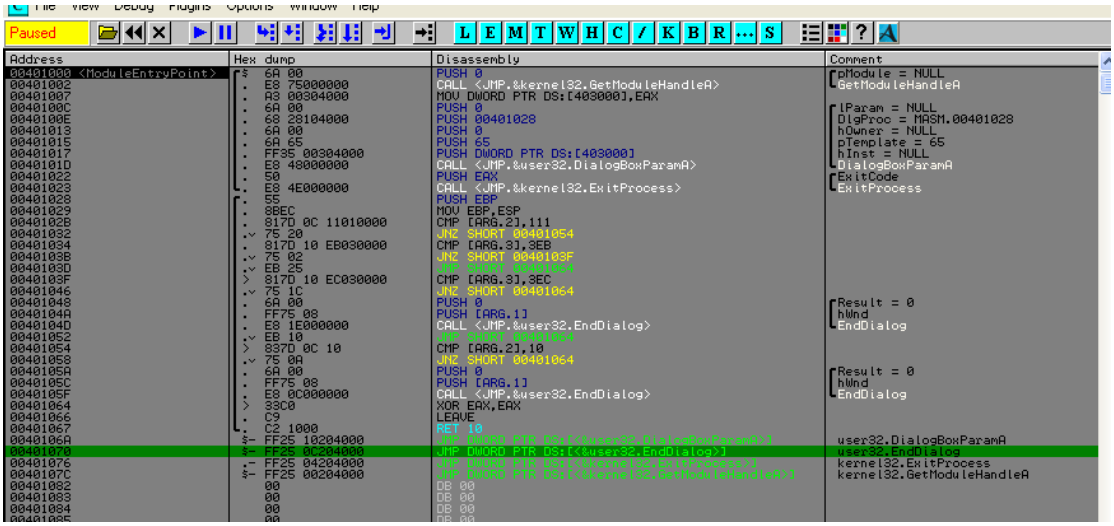


همانطور که می بینید در بین Jump ها یک دستور Mov Eax, Eax هم قرار دارد.

۳-۳ : MASM32 و TASM32

فایل زیر را در OllyDBG باز کنید:

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

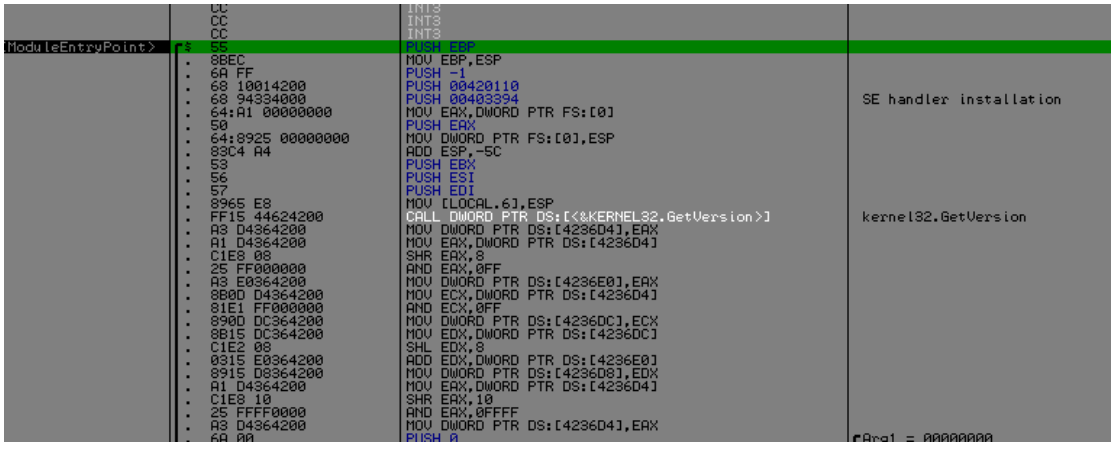


همانطور که می بینید برنامه هاي MASM یا TASM بسیار راحت خوانده می شوند و دیگر خبری از Call های تو در تو و خسته کننده نیست ☺...این خصوصیت برنامه های MASM هست که راحت تر از سایر زبان های برنامه نویسی خوانده می شوند.
معمولا Entry Point در برنامه های MASM برابر ابتدای سکشن text. است.

۳-۴ : ویژوال سی پلاس پلاس

فایل زیر را در OllyDBG باز کنید:

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



معمولا برنامه هاي كمپايل شده در سي پلاس پلاس با اين سه دستور شروع مي شوند:

Push EBP
MOV EBP, ESP
Push -1

و بعد از اين دستورها **معمولا** يک SEH(SE Handler) نصب مي شود.

توضيح اضافه: Structured Exception Handler كه به طور خلاصه به آن SEH گفته مي شود،تابعي است كه اگر خطايي در برنامه اتفاق بيفتد،برنامه به آن مراجعه مي كند،به اميد اينكه اين تابع بتواند خطا را برطرف نمايد.

در سي پلاس پلاس بعد از SEH **معمولا** توابع زير اجرا مي شوند :

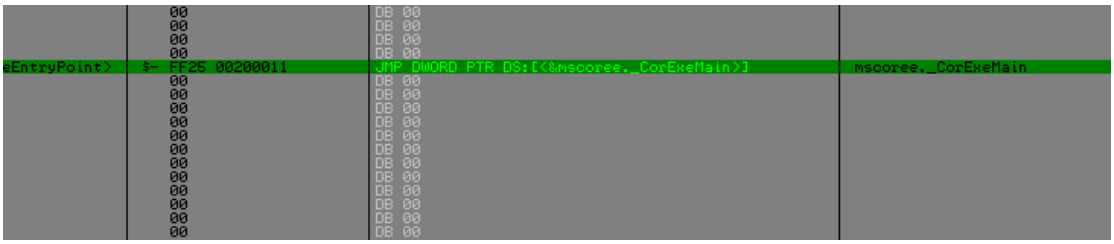
GetVersion يا GetVersionExA (براي به دست آوردن ورژن ويندوز)
GetCommandLineA (اشاره گري براي رشته Command-Line براي پروسه فعلي برمي گرداند.)
GetStartupInfo (Startup Info را بازگرداني مي كند.)
GetModuleHandleA (Handle براي ماژول مشخص شده برمي گرداند.)

۳-۵ : .net

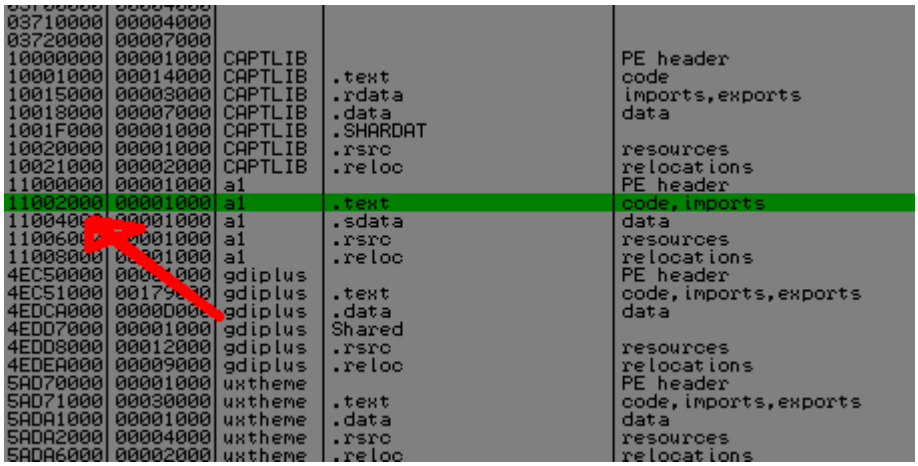
براي اجرا شدن برنامه هاي نوشته شده در VB.net يا VC.net نياز به .net Framework Microsoft وجود دارد.فايل زير را در OllyDBG باز كنيد:

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

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



همانطور كه مي بينيد در Entry Point برنامه ما تابع CorExeMain صدا زده شده،معمولا اين تنها Import ماست و IAT ما فقط يک تابع دارد و **معمولا** در برنامه هاي .net، IAT ما در شروع سكشني(Section) كه در آن Imports قرار دارد،وجود دارد.كافيست Memory Map را باز كنيد(كليد ALT + M)



ابتداي سكشني كه در آن Imports قرار دارد در اينجا برابر 11002000 هست.لذا شروع IAT ما هم همانجاست.

معمولا برنامه هاي نوشته شده در .net اگر پک بشوند.بسيار راحت آنپک مي شوند.كافيست فايل خودمان را در Memory Map پيدا كنيم و از آن قسمت از Memory با Lord PE يا PETools دامپ بگيريم.
البته اين را هم بگويم كه پكرها و پروتكتورهايي مخصوص برنامه هاي .net نوشته شده اند.اگر با چنين پكرهايي روبرو باشيم،آنپک كردن به اين راحتي نيست.ولي اگر پكر مخصوص برنامه هاي .net نباشيد،كافيست كه فايل را Dump بگيريم.همچنين يک نرم افزار براي آنپک كردن برنامه .net به اسم .net Generic Unpacker هست كه به طور خودكار فايل ما را در Memory پيدا مي كند و آن را دامپ مي كند. ☺

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

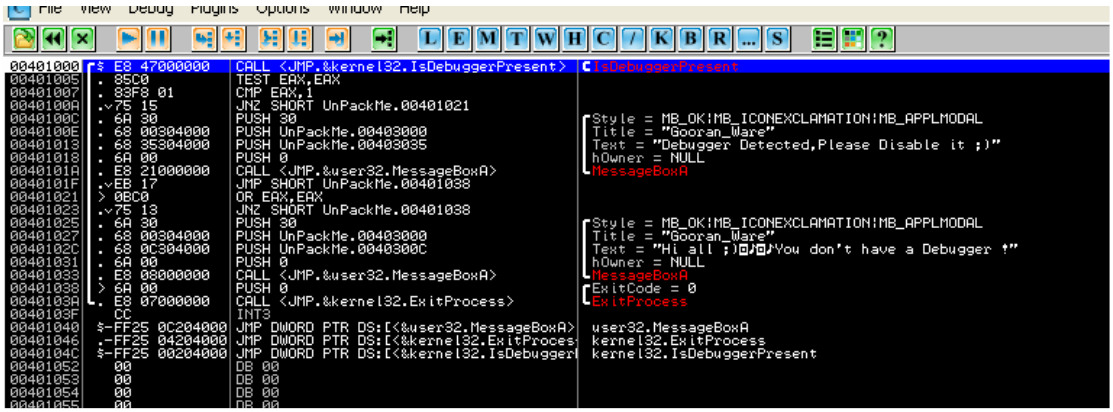
بعد از شناخته شدن دیباگر هر اتفاقی ممکن است بیفتد...هر اتفاقی که نویسنده پکر طراحی کرده است،می افتد:

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

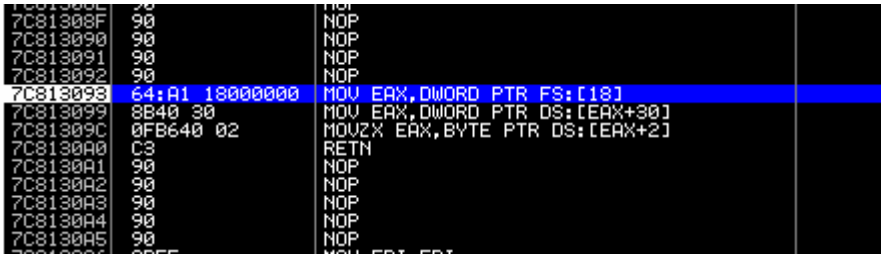
حتی من یک پکر را دیده بودم که اگر دیباگر را شناسایی می کرد،فایل Boot.ini ویندوز را دستکاری می کرد،طوری که ویندوز دیگر اجرا نمی شد.
یک پکر دیگر را هم دیده بودم که در صورت شناسایی دیباگر یک فایل Bat از درون خودش Extract می کرد و آن را اجرا می کرد،آن فایل Bat تمامی درایوها به غیر از درایو ویندوز را فرمت 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 OllyDBG بدون Plug-in باز کنید:

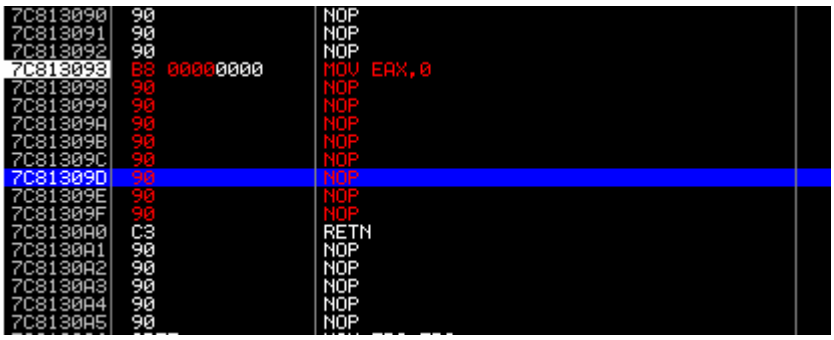
Targets___Examples___\Anti-Debug\Anti-Debug.exe



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،مثلا کدهای بالا را به این صورت تغییر دهیم:



همانطور که می بینید برنامه نتوانسته دیباگر را شناسایی کند:

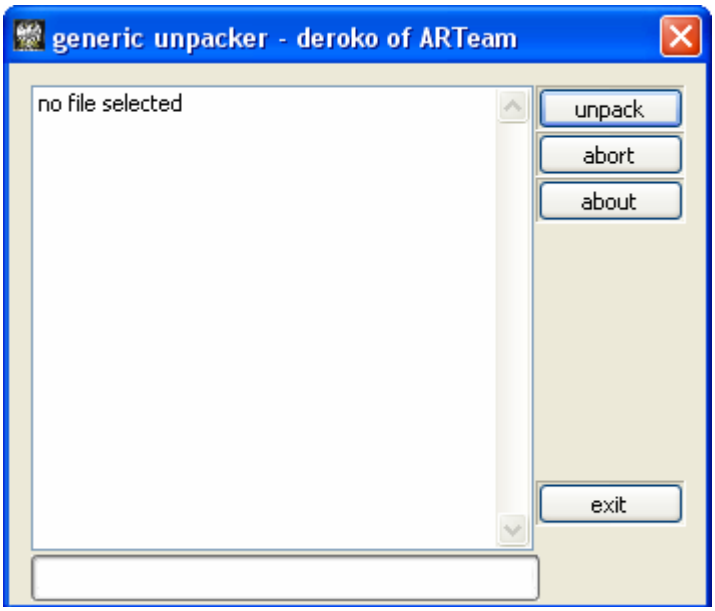
این تغییراتی که ما در فایل Kernel32.dll دادیم،ذخیره نمی شود،چرا که ویندوز اجازه دستکاری فایل های اصلی خود را نمی دهد.پلاگین هایی که برای شناسایی نشدن دیباگر نوشته شده اند هم **تقریبا** یک همچین کاری می کنند.یعنی هر وقت که OllyDBG باز شد،به طور خودکار این تغییرات را در فایل های DLL ویندوز انجام می دهند.بنابراین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

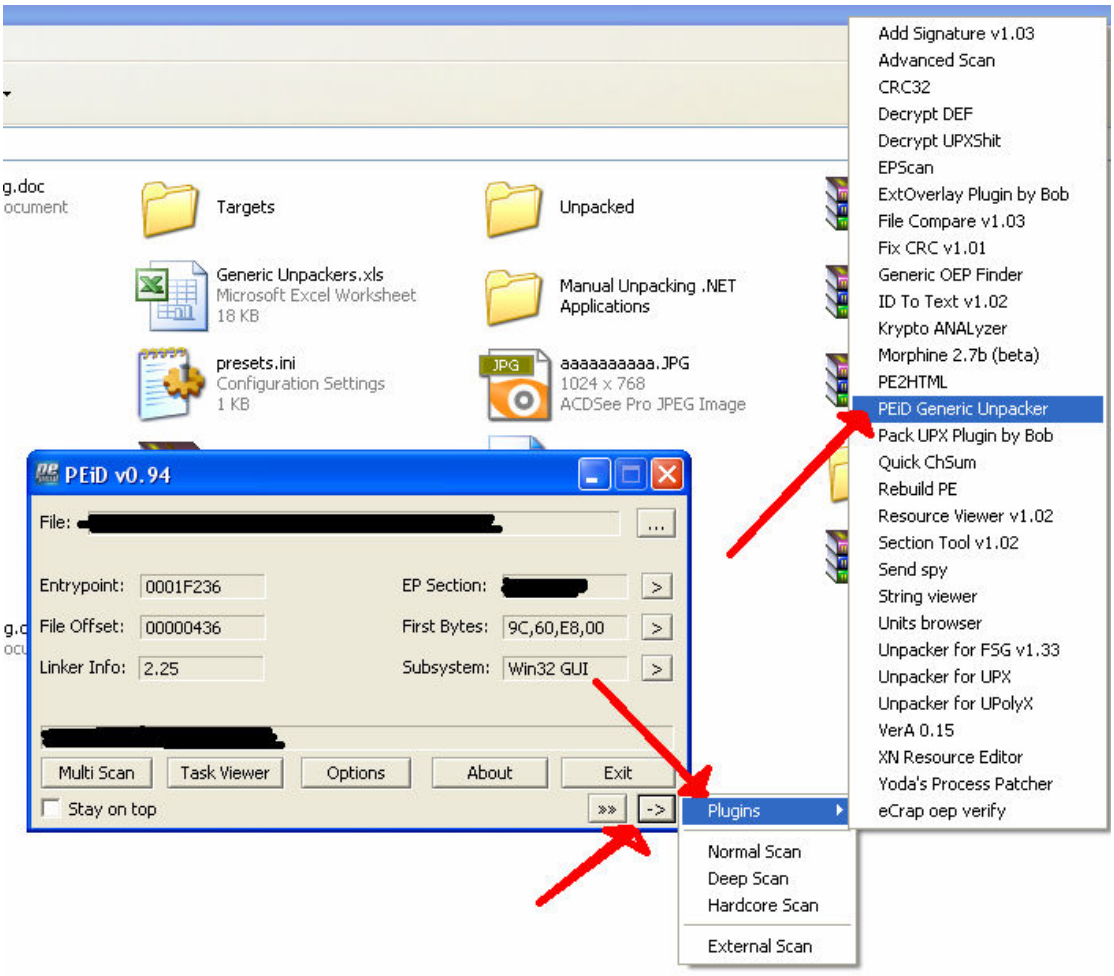
۵-۱ : Dereko Generic Unpacker

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 ☺

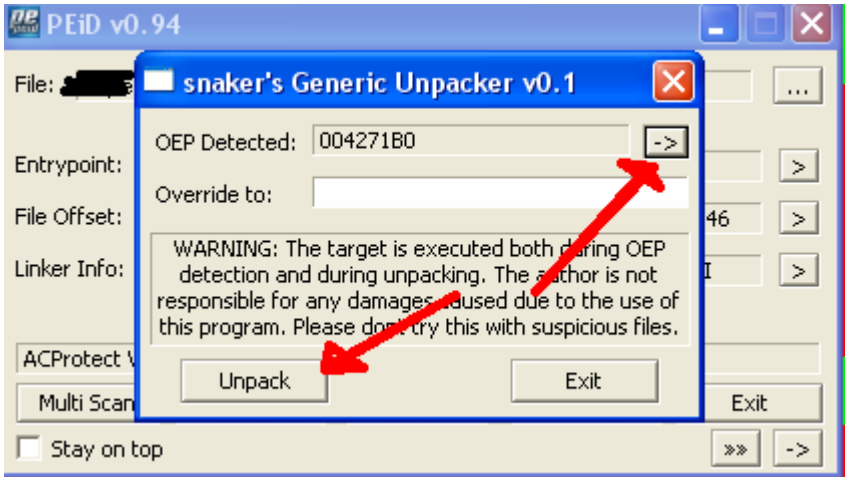


۵-۲ : PEID Generic Unpacker

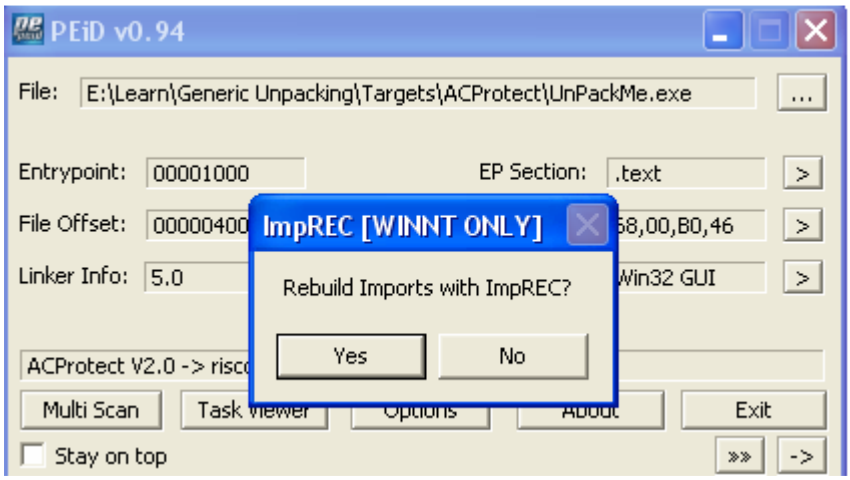
پلاگینی برای برنامه PEID نوشته شده که می تواند پکرهای ساده را آنپک کند،برای این کار بعد از باز کردن برنامه در PEID همانند تصویر عمل کنید:



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



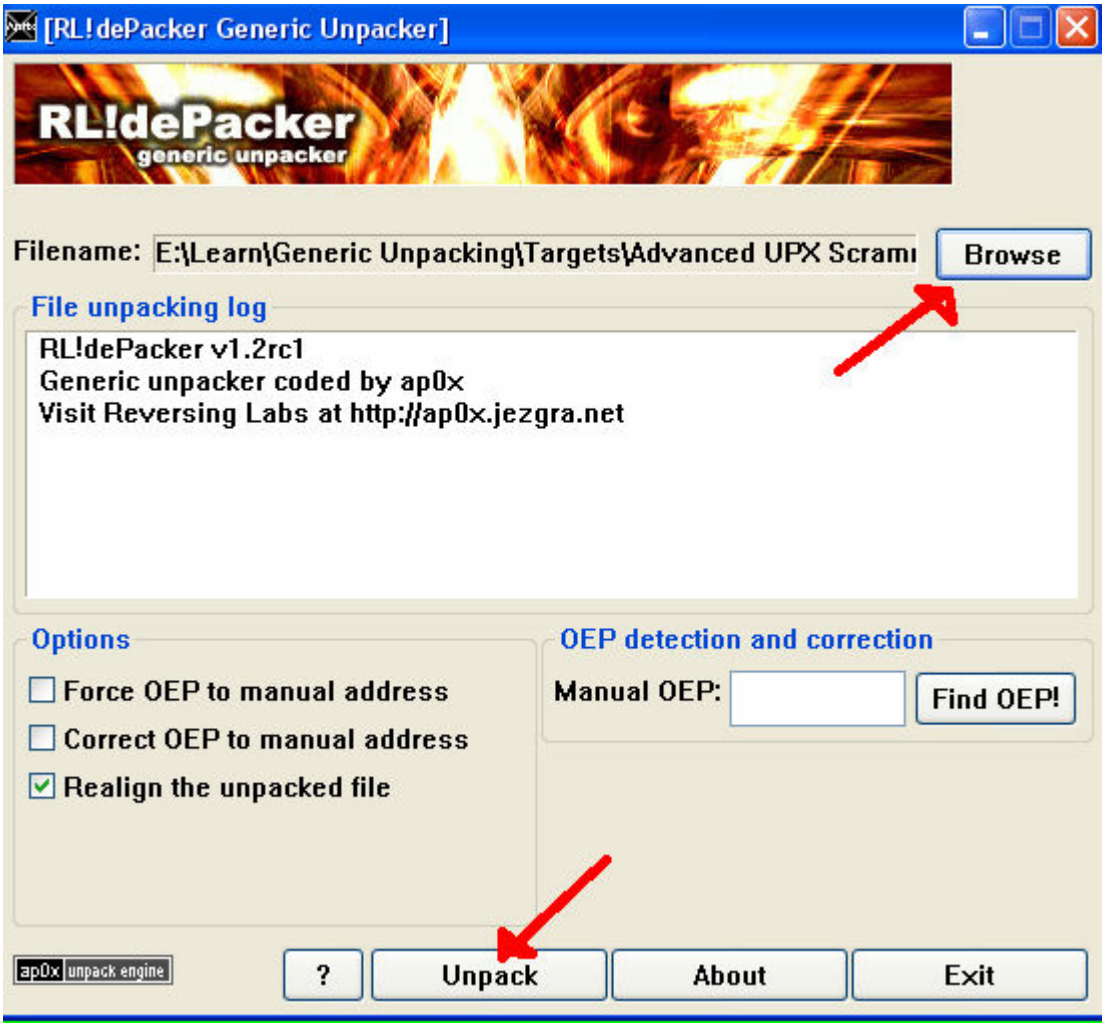
بعد از آن هم برنامه از شما می پرسد آیا می خواهید با استفاده از DLL برنامه ImportREC، IAT برنامه را دوباره بسازم؟شما گزینه Yes را بزنید تا فایل آنپک شود ☺
اگر بعد از زدن دکمه Yes پیغامی داده شده،یعنی برنامه آنپک نشد ☹



RL!dePacker : ۳-۵

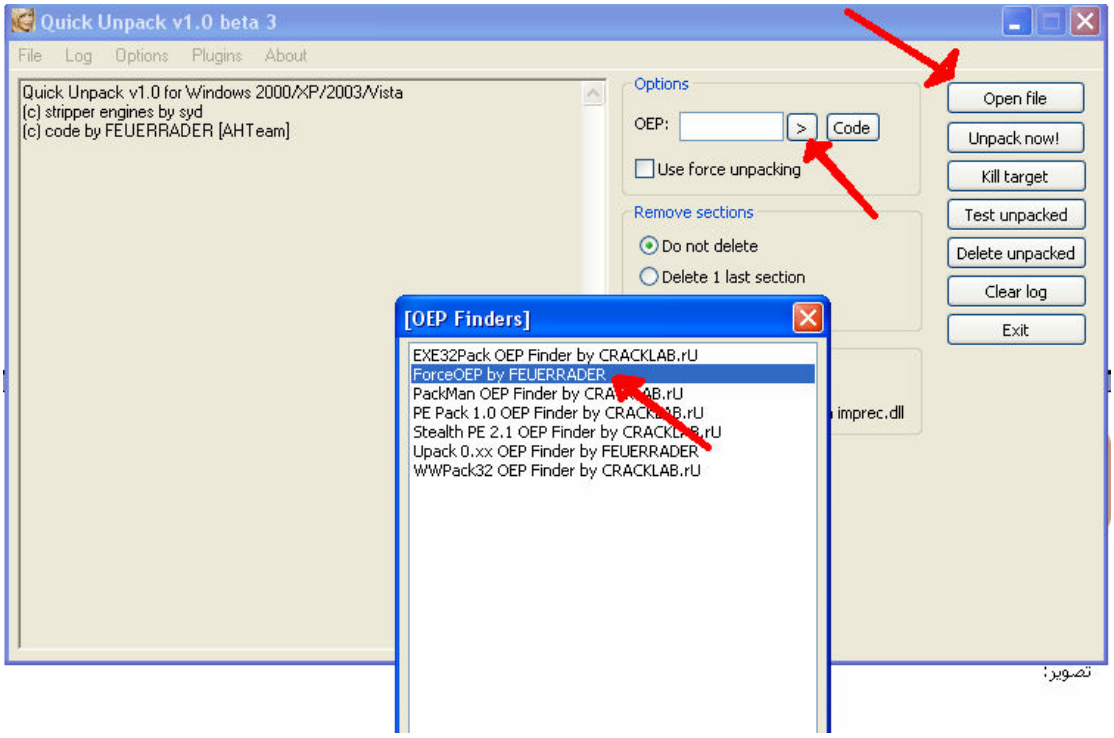
این هم یک Generic Unpacker دیگر هست که یه خرده قاطیه ☺...گاهی وقت ها یک فایل بهش می دهی،برات در عرض سه سوت آنپک می کنه...حالا اگر یک بار دیگر همون فایل رو بهش بدی،نمی تونه آنپک کنه ☹

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



۴-۵ : Quick Unpack

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.

۵-۵ : VMUnpacker

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 Scrammbler	✓	✓			
AHPacker	✓	✓			✓
ARM Protector		✓		✓	
Armadillo					
ASDPack	✓				
AsPack	✓		✓	✓	✓
Asprotect					
BamBam	✓	✓	✓	✓	✓
Crunch	✓			✓	
CrypKey SDK	✓	✓	✓	✓	
Enigma					
EP! ExePack					
Exe32Pack					✓

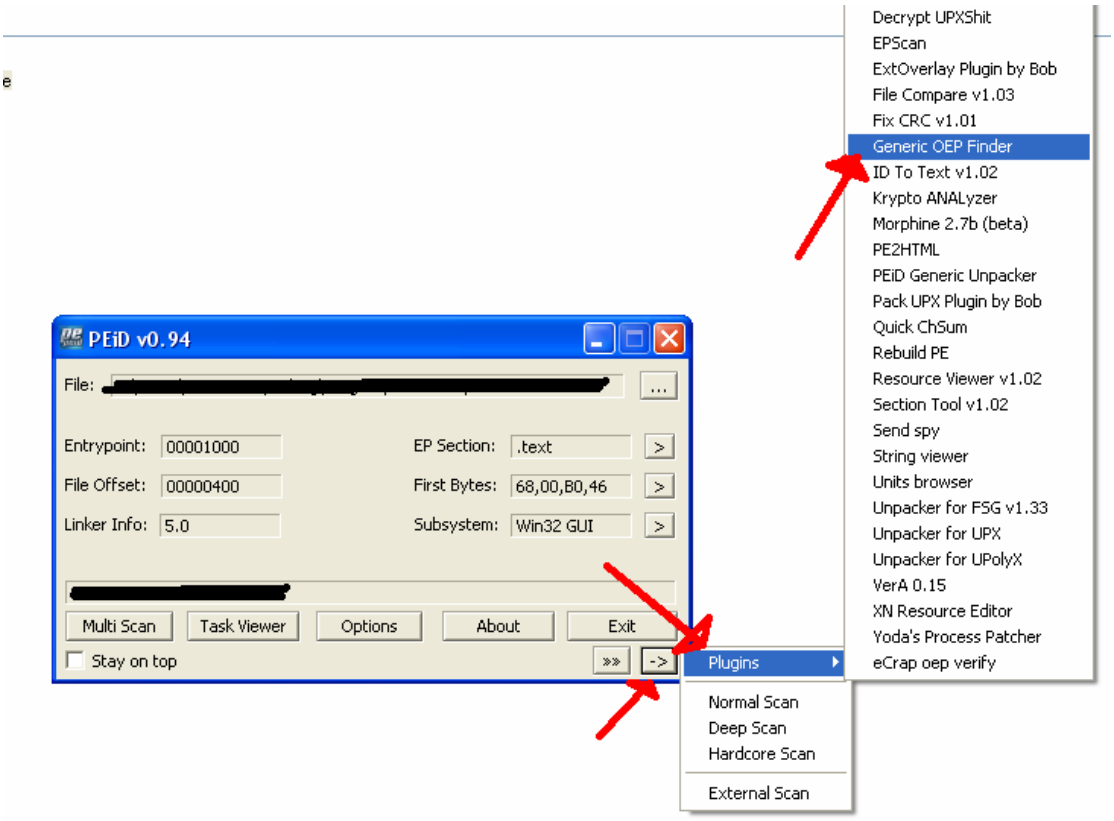
ExeCryptor	✓				
ExeShield			✓		
eXpressor	✓				
EZIP	✓	✓	✓		✓
Fearz Packer	✓				
FSG	✓			✓	✓
Fusion	✓			✓	
Gooran_Ware UnpackMe#1	✓				
JdPack	✓	✓			✓
KByS Packer	✓		✓	✓	✓
Mario Pack					
MEW	✓	✓	✓		✓
mJo0T MIV InfoViewer	✓	✓			
MoleBox					
Morphine	✓				
NeoLite	✓				
NoName Packer	✓	✓	✓	✓	
NoName 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 مي بينيد Dereko 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 هاي زير قرار دارند:

۲. PEID Generic Unpacker (۱۹ فایل آنپک کرد)
۳.VMUnpacker (۱۶ فایل آنپک کرد.)
۴.RL!DePacker و Quick Unpack (هر کدام ۱۵ فایل را آنپک کردند)

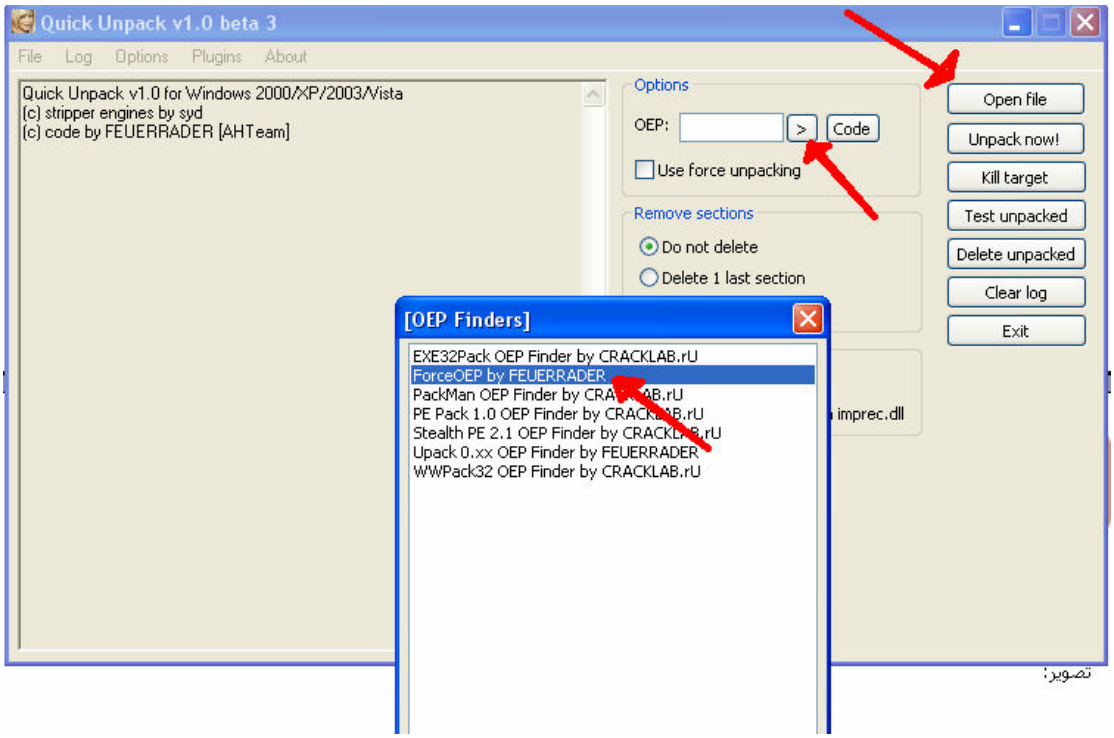
۷-۵ PEID OEP Finder :

يافتن OEP يكي از مهمترين مراحل در آنپک کردن یک فایل است.براي اين كار هم برنامه هايي وجود دارند كه مي 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:



۸-۵ : Quick Unpack

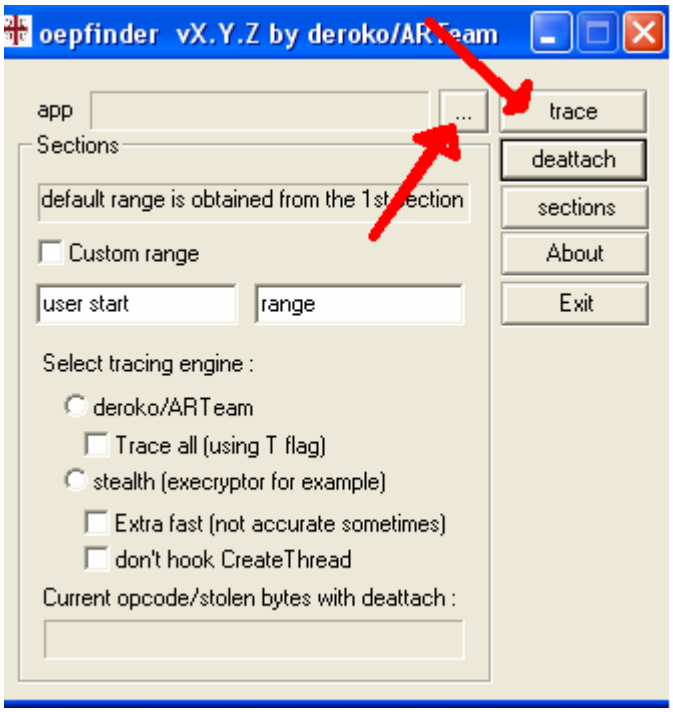
برای یافتن OEP در Quick Unpack همانطور که قبلا گفتم همانند این تصویر عمل می کنیم:



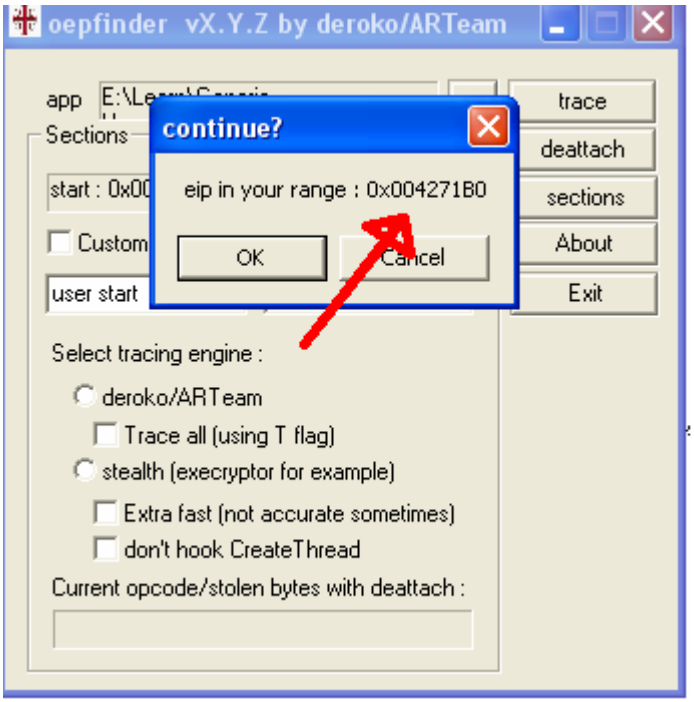
تصویر:

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 همچین 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 هاي قبلي سه OEP Finder به شما معرفي كرديم،اين هم نتيجه تست گرفته شده از اين OEP finder ها:

نام پکر	PEID OEP Finder	Quick Unpack	Dereko OEP Finder
Acprotect	✓	✓	
Advanced UPX Scrammbler			✓
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 مي بينيد PEID و Quick Unpack بسيار شبيه به هم عمل کردند و فقط PEID در يک مورد موفق تر عمل کرد.پس:
۱.PEID (۳۱ مورد درست)
۲.Quick Unpack (۳۰ مورد درست)
۳.Dereko (۲۵ مورد درست)

۵-۱۱ : آنپکرهای نرم افزاری

معمولا بعد از اينکه يک پکر يا پروتکتور منتشر مي شود،يک آنپکر مخصوص آن پکر نوشته مي شود.که کار ما را براي کرک کردن برنامه بسيار راحت مي کنند.براي به دست آوردن آنپکر مورد نظر خود مي توانيد از طريق گوگل به نتيجه برسيد،يا اينکه توي اين سايت ها جستجو کنيد:

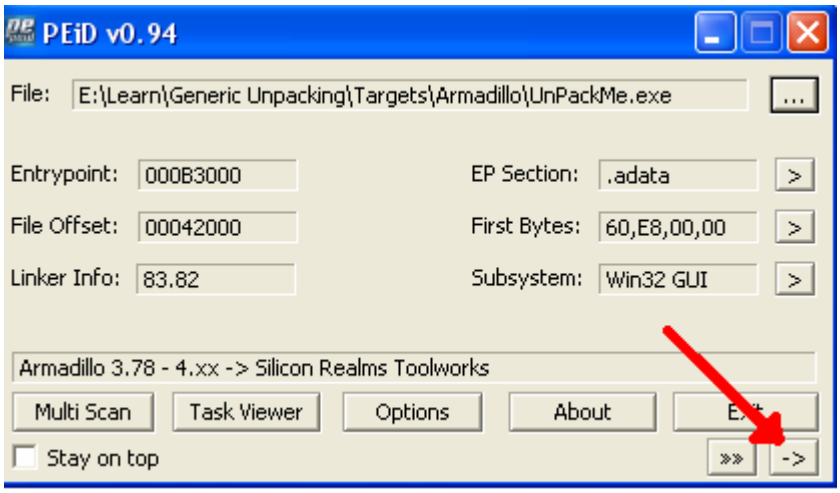
http://www.tuts4you.com
http://www.exetools.com/
http://www.woodmann.com/
http://www.unpack.cn/
http://forums.accessroot.com/
http://www.openrce.com/
/http://www.aoreteam.com
☺ http://forum.p30world.com/forumdisplay.php?f=133

مقدمه ۶ : شناسایی پکر

شناسايي پكر اولين قدم براي آنپک كردن يک فايل است.براي شناسايي يک پكر هم تا دلتون بخواد نرم افزار هست ☺
ولي خوب از بين اينها ما فقط پنج مورد را مورد بررسي قرار مي دهيم :

PEiD : ۶-۱

اين برنامه را همه مي شناسند.معروفترين نرم افزار براي شناسايي پكر ☺

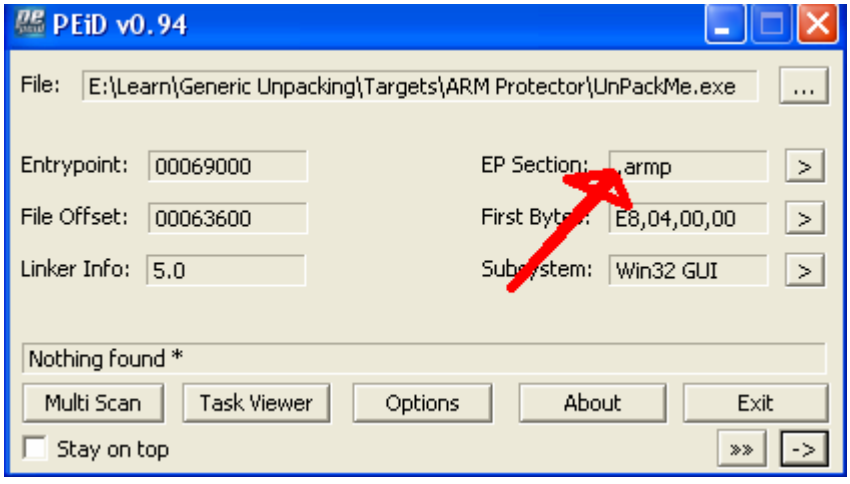


اگر برنامه موفق به شناسايي پكر نشد،مي توانيد با زدن دكمه نشان داده شده در بالا گزينه هاي زير را امتحان كنيد:

Deep Scan
Hardcore Scan
External Scan

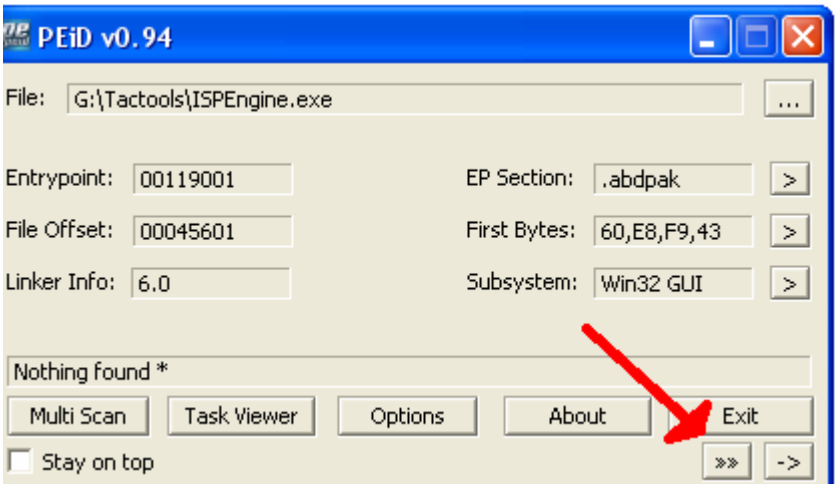
همچنين اين برنامه اطلاعات بسيار مفيدي درباره برنامه به ما مي دهد ☺

مثلا Entry point فايل ما(Entry point)
سكشني كه در آن Entry Point قرار دارد.(EP Section) و همچنين اگر بر روي دكمه كنار آن EP Section كليک كنيد،مي توانيد تمامي سكشن هاي برنامه را ببينيد.
و همچنين اولين دستورات برنامه را هم به ما نشان داده مي شود.(بر روي دكمه كنار First Bytes كليک كنيد)
اگر برنامه موفق به شناسايي پكر نشد،گاهي اوقات نام Entry point section مي تواند راهنماي خوبي براي شناسايي پكر باشد.مثلا تصوير زير را نگاه كنيد،برنامه نتوانسته پكر را نشان دهد.اما نام سكشن armp. هست.لذا **ممكن** است پكر ما ARM Protector باشد. ☺

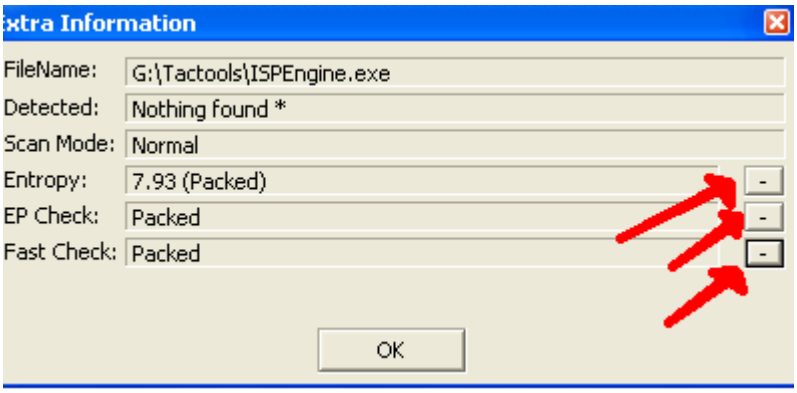


يک نكته را اشاره كنم،اينكه PEID مي گويد: Nothing found دليل نمي شود كه فايل ما حتما با يک پكر ناشناخته پک شده باشد...ممكن است فايل ما در يک زبان برنامه نويسي ناشناخته نوشته شده باشد.پس در اين موارد ابتدا بايد بفهميم كه آيا اصلا فايل پک شده است يا خير؟

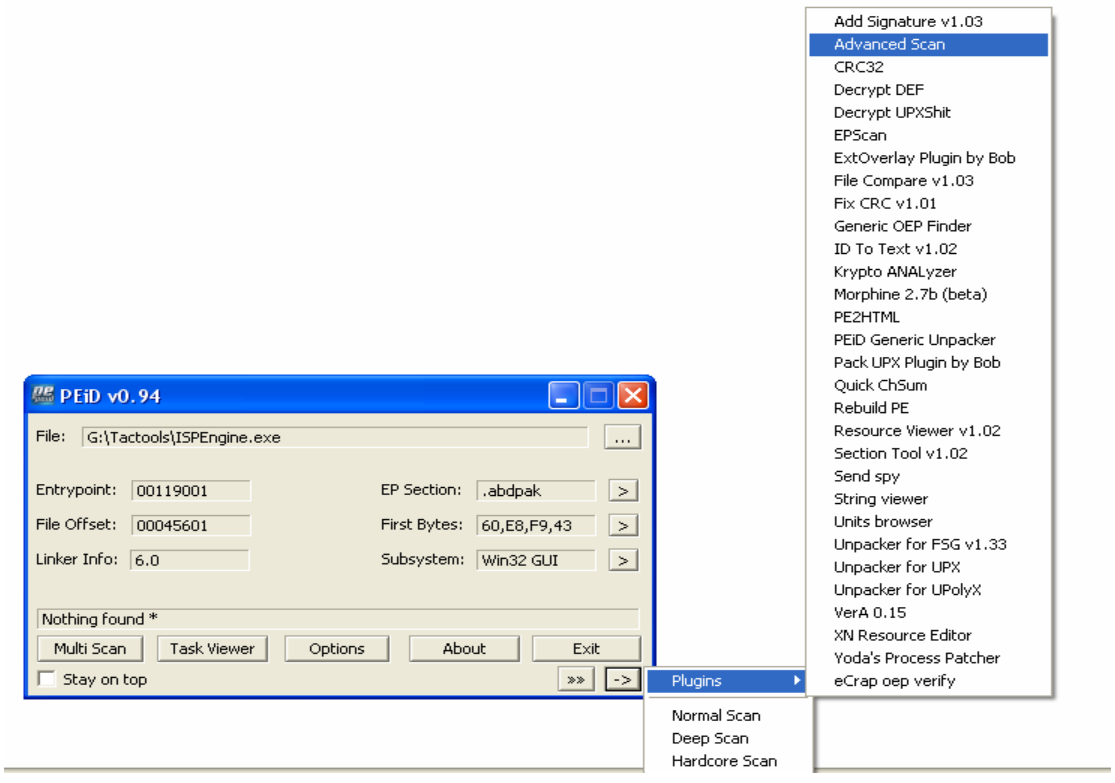
روشی هست که می توان فهمید آیا یک فایل پک شده است یا خیر؟...طبق تست هایی که من انجام دادم،این روش در % 78.6 موارد جواب می دهد.پس زیاد نمی شه بهش اعتماد کرد.برای این کار دکمه نشان داده شده در تصویر را بزنید:



حالا سه دکمه نشان داده شده را می زنیم:

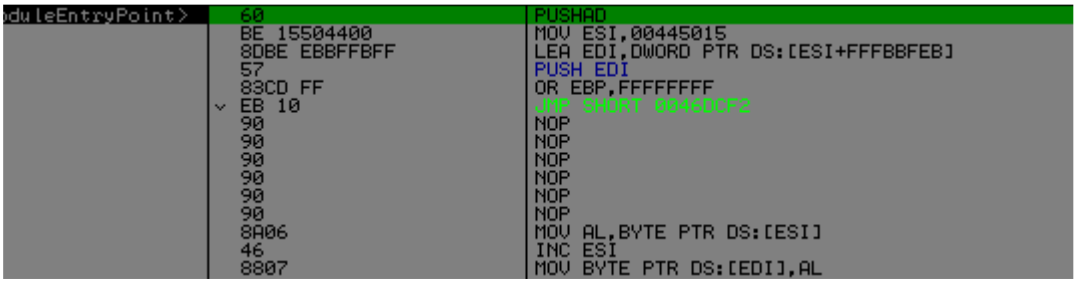


همانطور که می بینید هر سه دکمه به می گویند که فایل پک شده است...لذا فایل به احتمال بسیار زیاد پک شده است.
توصیه من این است که از پلاگین Advanced Scan هم استفاده کنید،تا لیست تمامی پکرهای ممکن را ببینید،همانند تصویر:

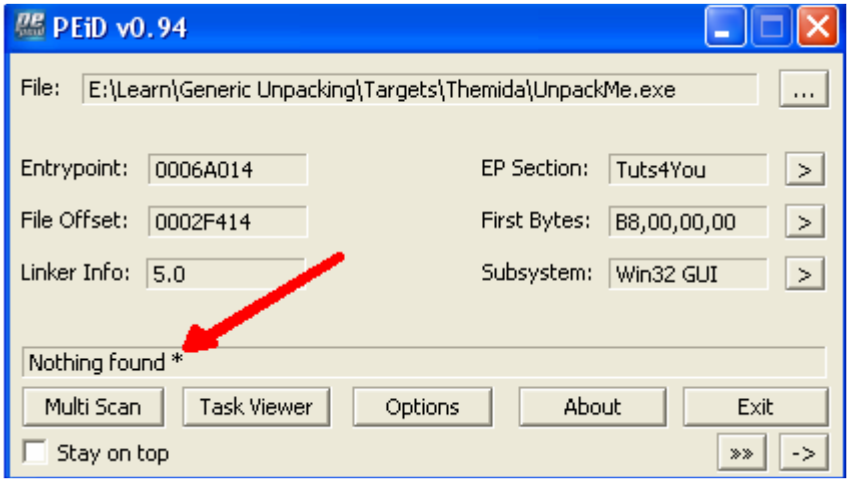


۲-۶ : اضافه کردن Signature

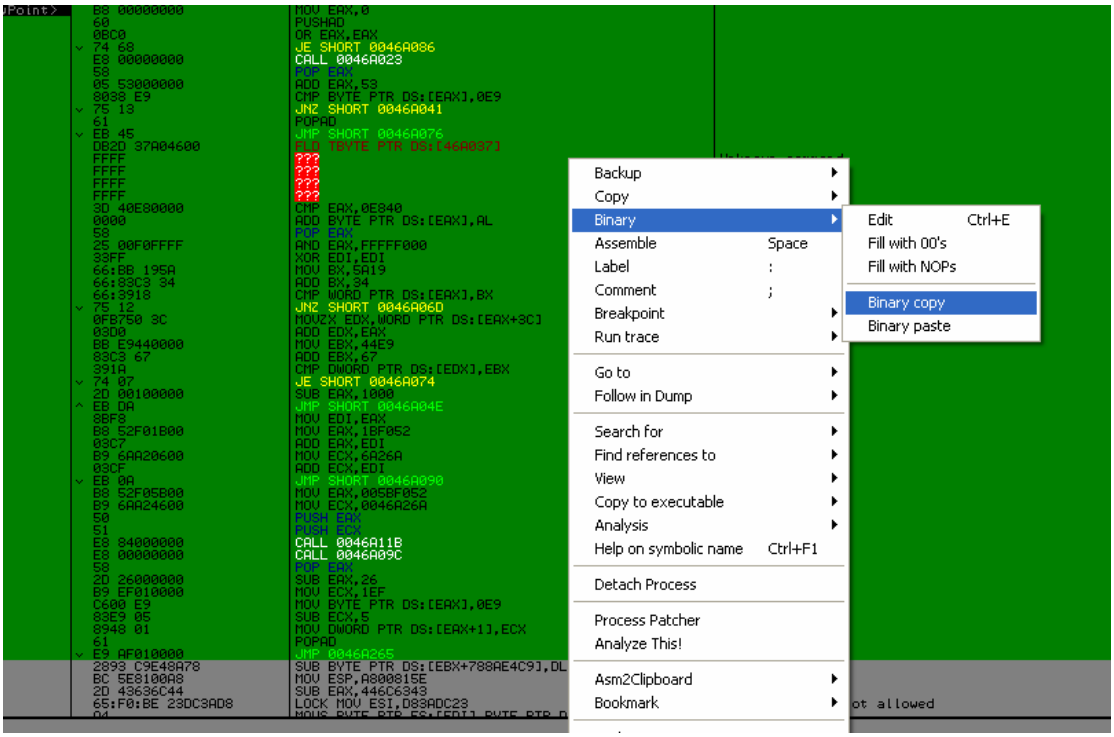
سوالي كه اينجا پيش مي آيد اين است كه PEID چه طور نوع پكر را تشخيص مي دهد؟...ساده است...PEID دستورات ابتداي فايل يا كل فايل را بررسي مي كند.براي مثال **اكثر** فايل هاي UPX در ابتداي فايلشان يک همچين دستوري دارند:



PEID مي آيد بررسي مي كند اگر ابتداي فايل دستوراتي شبيه اين وجود دارد،فايل را UPX تشخيص مي دهد.
PEID قابليت اضافه كردن اطلاعات به DataBase خودش را دارد.ما مي توانيم به PEID بگوييم كه اگر يک همچين دستوراتي در ابتداي فايلي پيدا شد،آن را به نام اين پكر شناسايي كند.براي مثال PEID نمي تواند Themidaرا تشخيص بدهد: ☹



من خود پكر(Themida) را دانلود مي كنم و حدود ۱۰ يا ۲۰ فايل را با آن پک مي كنم.و بعد دستورات ابتدايي(يا كل دستورات) اين فايل ها را با هم مقايسه مي كنيم،من يک فايل Themidaرا در OllyDBG باز كردم.همانند تصوير چند دستور اول را انتخاب مي كنيم و آنها را Binary Copy مي كنم.



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 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

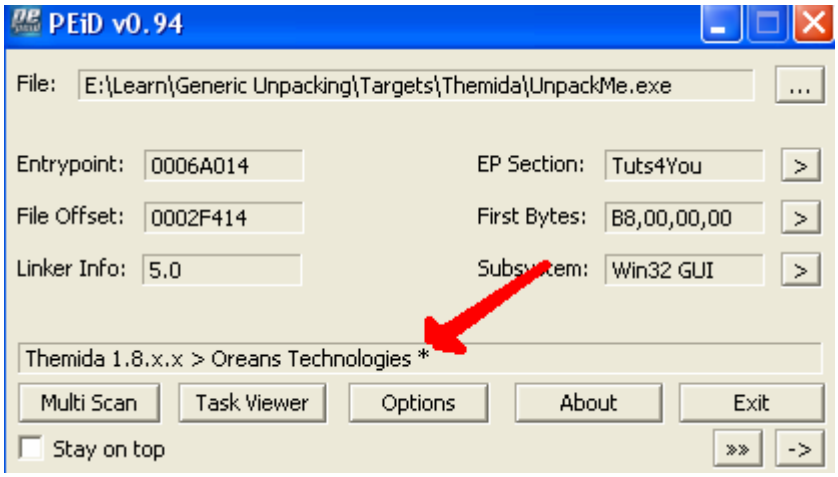
من این بایت هاي ابتدايي را اينجا Paste كردم،براي ده يا بيست فايل ديگر هم همين كار را مي كنم و تمام اطلاعات به دست آمده را با هم مقايسه مي كنم،در آخر به نتيجه زير مي رسم:

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 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 ؟؟ يعني اين بايت در فايل هاي مختلف پک شده با Themida متفاوت است.حالا مي توانيم اين Signature را به PEID اضافه كنيم،براي اينكار فايل userdb.txt كه در پوشه PEID قرار دارد را باز مي كنيم و اطلاعات زير را به آن اضافه مي كنيم:

]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 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 كه در بالا مي بينيد در داخل كروشه نام پكر را نوشتم.بعد از Signature=،Signature فايل را نوشتم و در كنار ep_only= بايد مشخص كنم كه آيا برنامه فقط بايد در ابتداي فايل دنبال اين بايت ها باشد،يا در كل فايل؟كه من نوشتم true،يعني فقط ابتداي فايل را بررسي كن!
حالا فايلمان را دوباره با PEID بررسي مي كنيم:

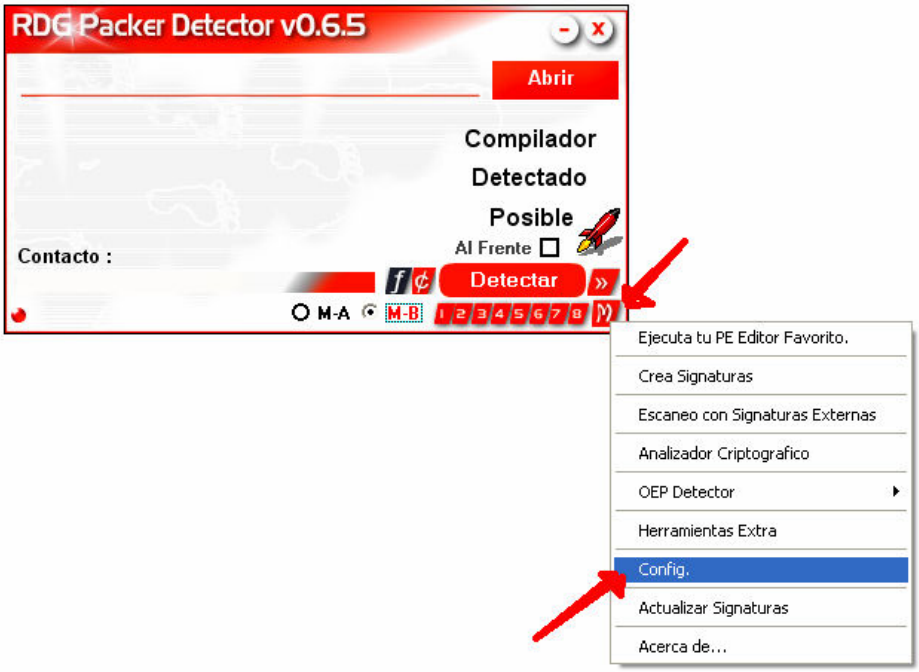


اينبار برنامه مي تواند Themida را تشخيص دهد.علامت * به اين معناست كه اين فايل با استفاده از Signature هاي خارجي(فايل Userdb.txt) شناسايي شده است.
فايل userdb.txt من حدود ۱۳۰۰ Signature در خود دارد.همچنين اگر پكر جديدي پيدا كردم سعي مي كنم Signature آن را اضافه كنم.براي دانلود آن از لينک زير استفاده كنيد:

http://subtitle.persiangig.com/userdb.rar

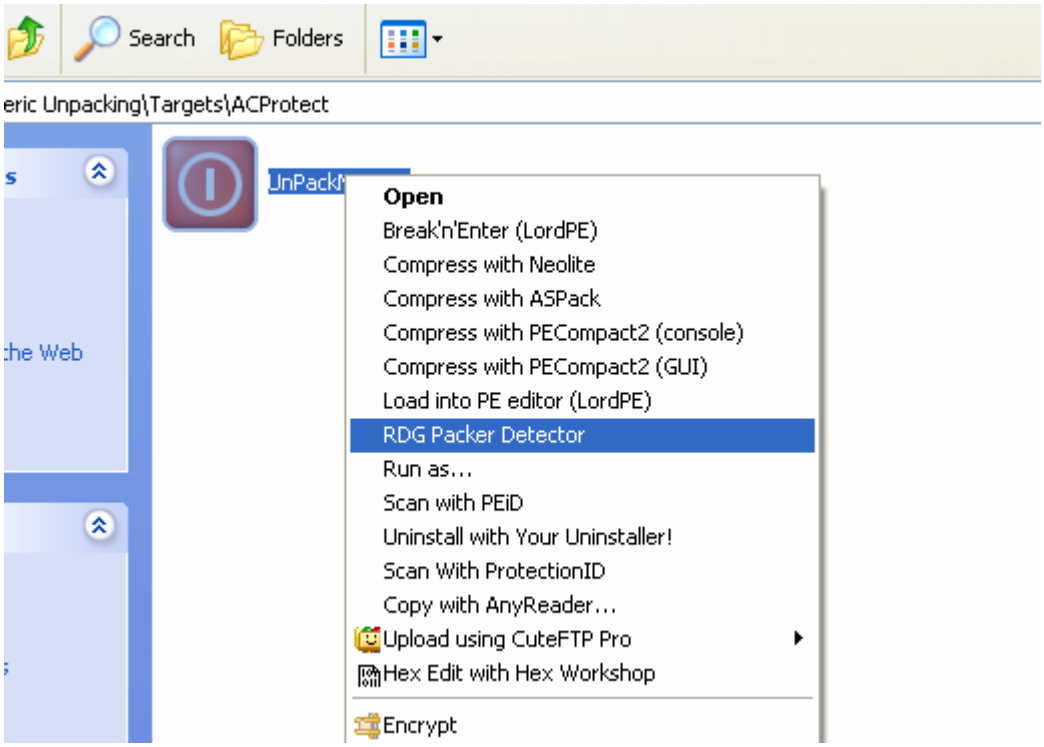
۳-۶ : RDG Packer Detector

اين هم برنامه ديگري براي شناسايي پكر است،بهتر است اين تنظيمات را در برنامه RDG Packer Detector انجام دهيد:



و بعد :

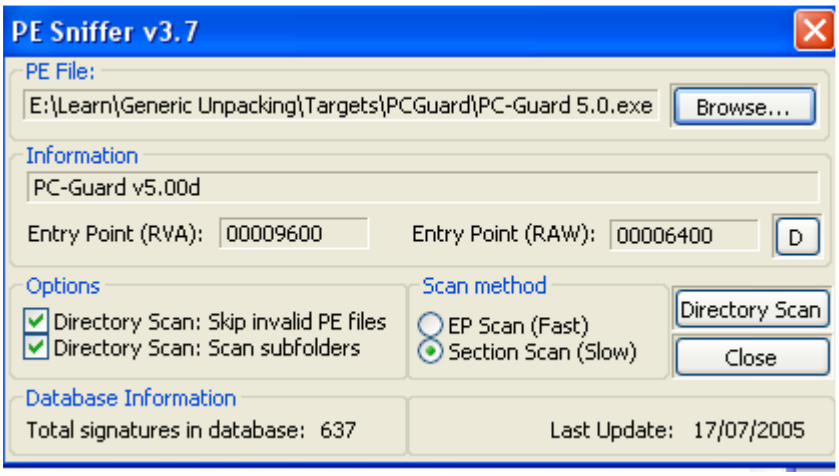
حالا براي بررسي يک فايل از طريق از اين برنامه کافيست بر روي فايل کليک راست کرده و گزينه RDG Packer Detector را بزنيد:



در قسمت Compilador نام کمپایلر(همان زبان برنامه نویسي) نوشته شده...در بالا برنامه نتوانسته زبان برنامه نویسي را تشخیص بدهد.در قسمت Detectado نام پکر نوشته مي شود.در قسمت Posible ،پروتکشن هایي که ممکن است فایل داشته باشد،نوشته مي شود.مثلا اگر پکري از تابع IsDebuggerPresent استفاده کرده باشد.آنجا نوشته مي شود.در قسمت Contacto نام سایت نویسنده پکر یا پروتکتور نوشته مي شود.
همچنین اگر برنامه موفق به شناسایي پکر نشد،یعني 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

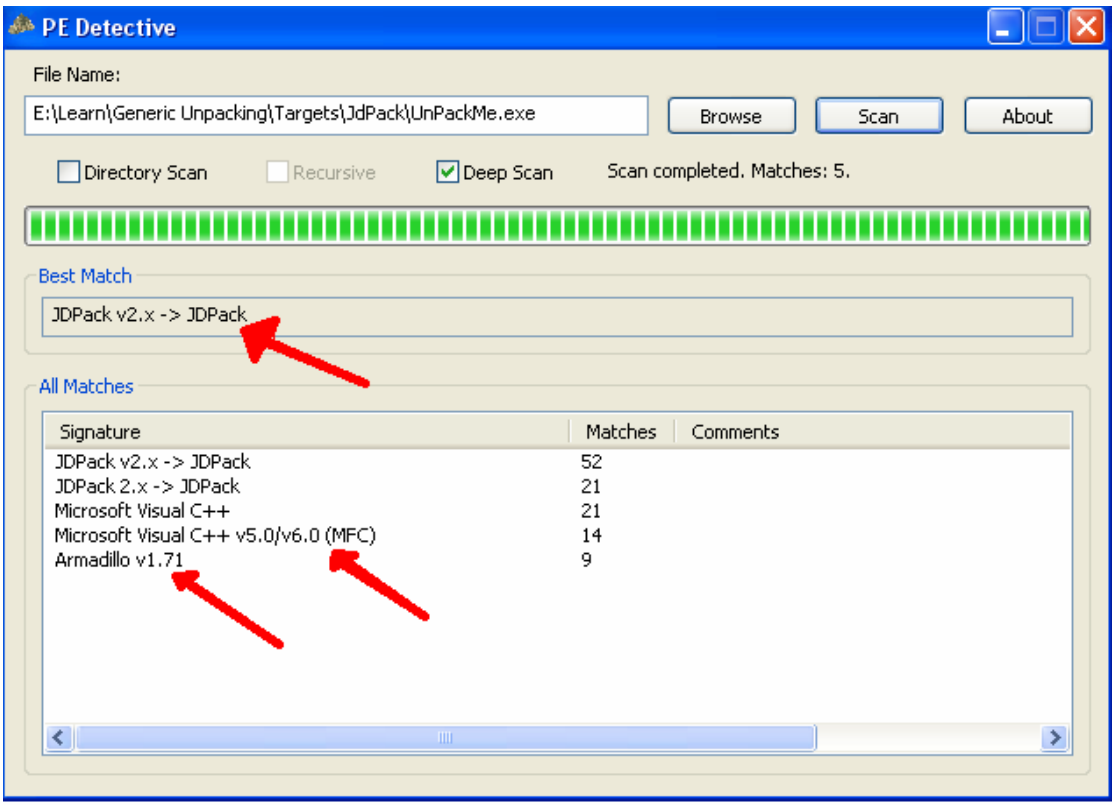
PeTools : ۴-۶

PeTools قابلیت هاي بسیار زیادي دارد که یکي از قابلیت هاي آن شناسایي نوع پکر مي باشد.براي شناسایي نوع پکر از طریق این برنامه در منوي Tools گزینه PE Sniffer را بزنید،و بعد گزینه Browse را بزنید و فایل خود را انتخاب کنید:



PE Detective : ۶-۵

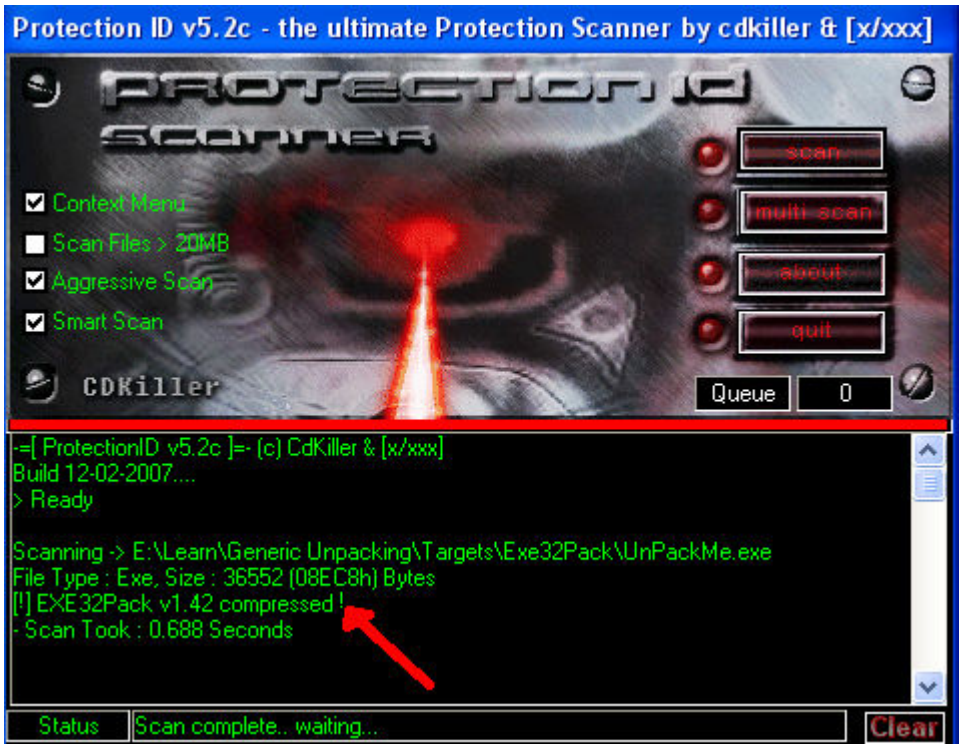
این برنامه به خاطر راحتي كاري كه دارد.يكي از برنامه هاي مورد علاقه من هست.براي بررسي يک فايل با استفاده از اين برنامه كافيست گزينه Browse را بزنيد فايل را به برنامه بدهيد و گزينه Scan را بزنيد:



همانطور كه مي بينيد اين برنامه مي گويد فايل داده شده به احتمال زياد JdPack مي باشد ولي ممكن است Visual C++ يا Armadillo 1.71 هم باشد.
با اين برنامه مي توانيد تمام فايل هاي يک پوشه را هم به طور يكسره بررسي كنيد.براي اين كار تيک گزينه Directory Scan را بزنيد.اگر مي خواهيد زير پوشه ها بررسي شوند گزينه Recursive را هم تيک بزنيد.

Protection ID : ۶-۶

اين هم يک برنامه ديگر،براي شناسايي پكر از طريق اين پكر برنامه را اجرا كنيد،گزينه Scanرا بزنيد و فايل را به برنامه بدهيد:



همانطور که مي بينيد فايل ما با Exe32Pack پک شده است.
اگر مي خواهيد با کليک راست بر روي فايل و انتخاب گزينه "Scan with protection ID" فايل خود را با اين برنامه بررسي کنيد.کافيست تيک گزينه Context menu را بزنيد.

۷-۶ : نتيجه تست

هر چند تست گرفته شده نشان دهنده قدرت اين ۵ برنامه نيست(چون اگر برنامه اي ديتابيسش آپديت باشد،مي تواند بيشتر پکرها را تشخيص دهد و در غير اين صورت خير).ولي به هر حال اين هم نتيجه تست گرفته شده از اين ۵ برنامه :

نام پکر	PEID	RDG	PeTools	PeDetective	Protection ID
Acprotect		✓		✓	✓
Advanced UPX Scrammbler		✓			
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:

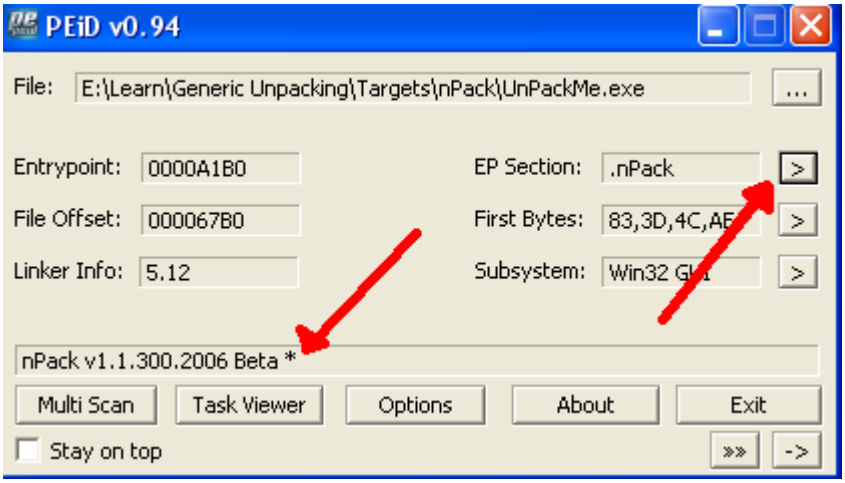
۲.PeDetective (۳۳ مورد صحیح)
۳.PEID (۲۷ مورد صحیح)
۴.Protection ID و PeTools (هر کدام ۲۳ مورد صحیح)

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

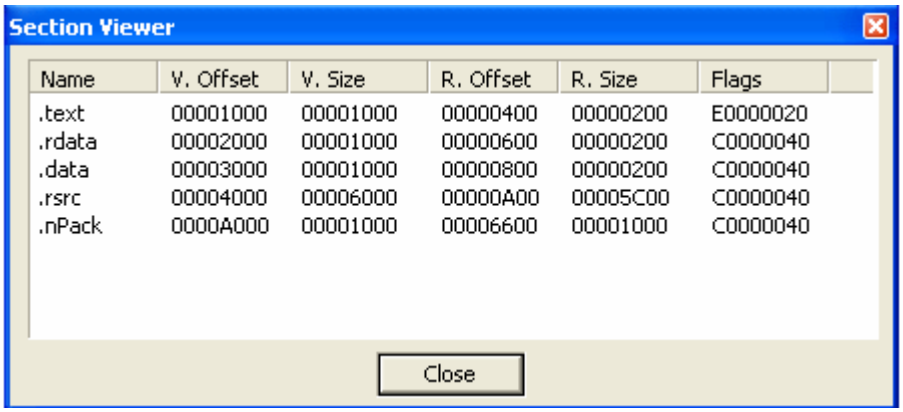
NPack

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 :



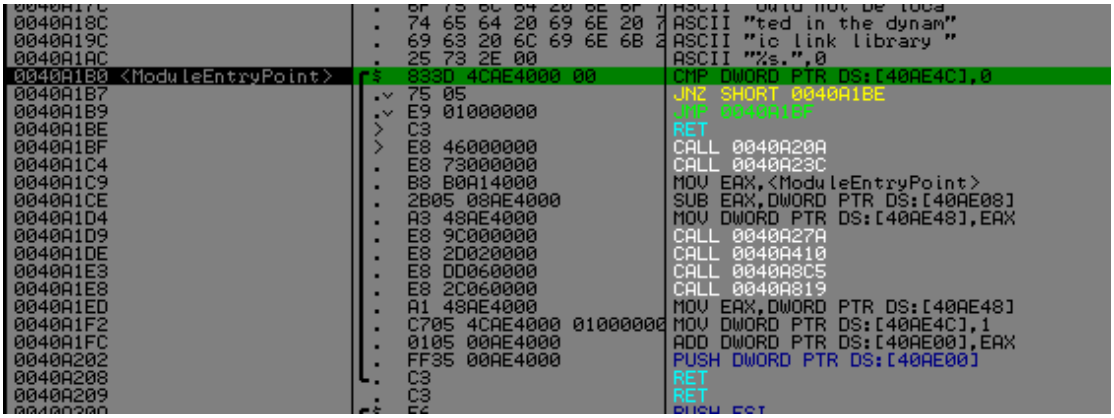
همانطور که می بینید ما ۵ سکشن داریم:

.text → 1000 تا 2000
.Rdata → 2000 تا3000
.data → 3000 تا 4000
.rsrc → 4000 تا A000
.nPack → A000 تا B000

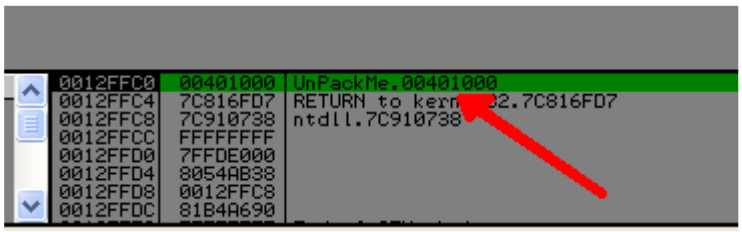
این اطلاعات بر مبنای RVA است،بهتر است آنها را به اضافه ImageBase کنیم تا اطلاعات را بر مبنای VA به دست اوریم:

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

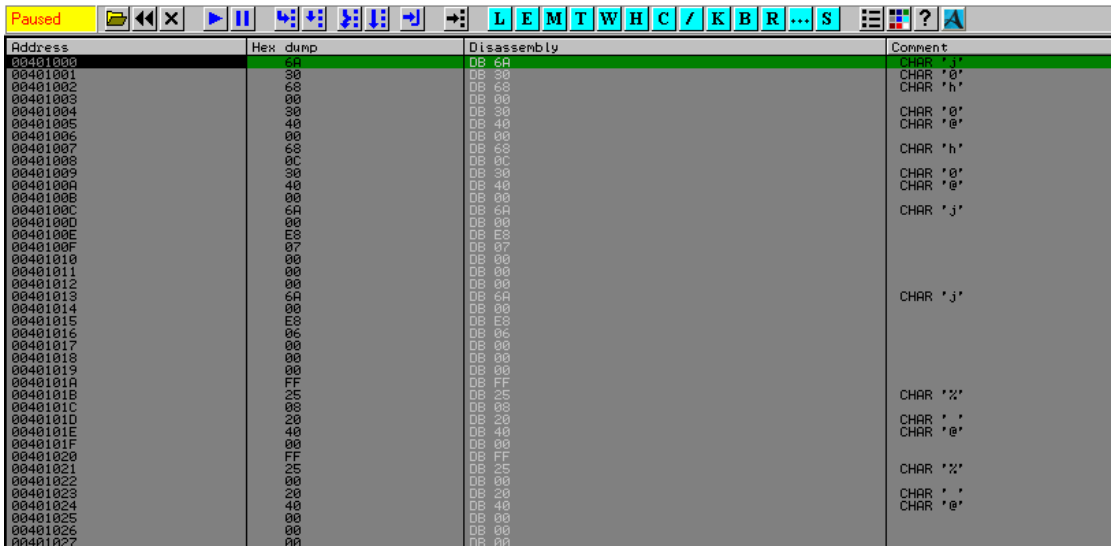
از بین این ۵ سکشن،دو سکشن Text و nPack برای ما اهمیت بیشتری دارند.چرا که در سکشن Text کدهای برنامه اصلی ما وجود دارد و در سکشن nPack کدهای پکر.
اطلاعات لازم را به دست آوردیم.فایل خود را در OllyDBG باز کنید:



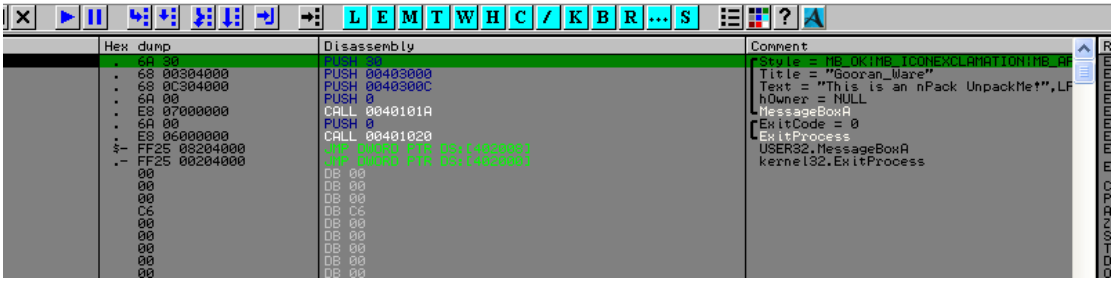
الان ما در خط 0040A1B0 هستیم.یعنی در سکشن nPack (با توجه به اطلاعات بالا)...ما باید ببینیم که در چه زمانی وارد سکشن اصلی برنامه یعنی Text Section می شویم؟...برنامه را آنقدر با کلید F8 جلو ببرید تا به خط 40A208 برسید...حالا نگاهی به پنجره Stack بیندازید:



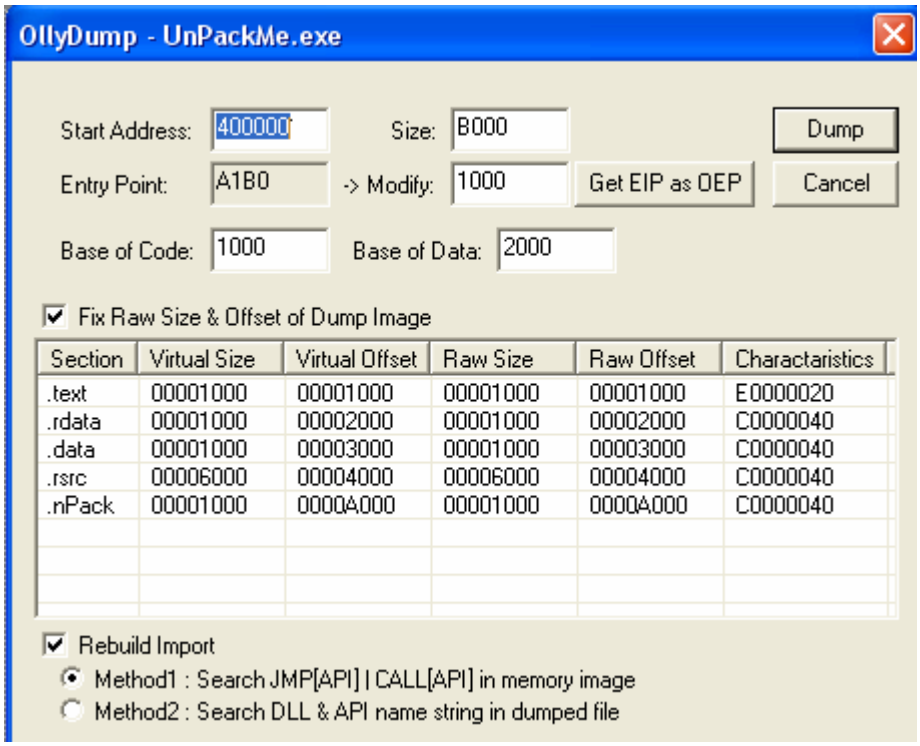
این یعنی اینکه این Return ما را به خط 401000 می برد.با توجه با اطلاعات بالا خط 401000 در کجا قرار دارد؟...درست است،در Text Section یعنی جایی که کدهای اصلی ما قرار دارد.پس با این Ret وارد Text Section می شویم.یعنی به OEP می رسیم.
حال کافیست یک بار دیگر کلید F8 را بزنید تا وارد Text Section بشوید:



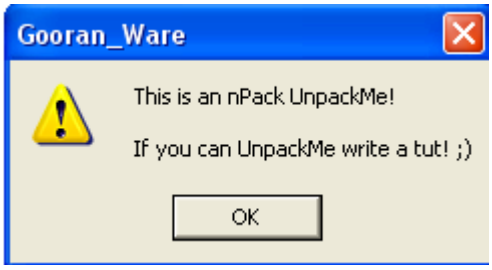
حالا چرا کدهای اصلی برنامه ما اینجوری است؟به خاطر اینکه OllyDBG به طور پیشفرض فقط کدهای سکشنی که در آن Entry Point قرار دارد را مورد بررسی قرار می دهد.و سکشن های دیگر را آنالیز نمی کند.کلید های CTRL + A را بزنید تا کدهای این قسمت بررسی شود:



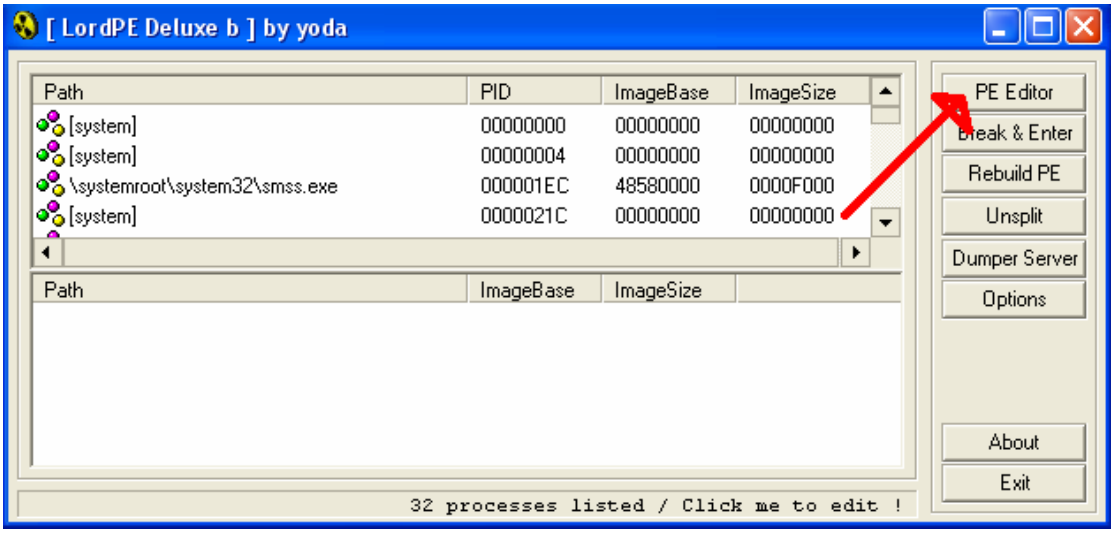
بله،برنامه ما در MASM نوشته شده و فقط با تابع MessageBoxA یک پیغام نشان می دهد و بعد با تابع ExitProcess بسته می شود.حالا کافیست با پلاگین OllyDump از فایلمان Dump بگیریم،برای این کار کلیک راست کرده و گزینه Dump Debugged Process را بزنید:



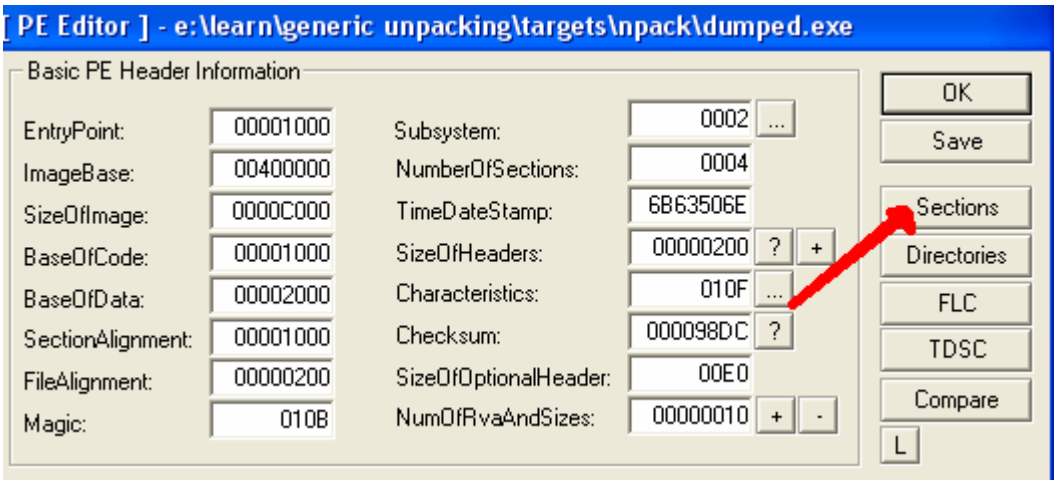
در تصویر بالا گزینه Rebuild Import را باید تیک بزنیم.چرا که خود OllyDump می تواند IAT ما را دوباره بسازد.و نیازی به استفاده از برنامه ای مثل ImportREC نیست.ولی خوب،اگر قرار بود IAT را با ImportREC بسازیم.باید تیک این گزینه را برمی داشتیم.
حالا گزینه Dump را بزنید و فایل دامپ شده را ذخیره کنید.و بعد فایل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



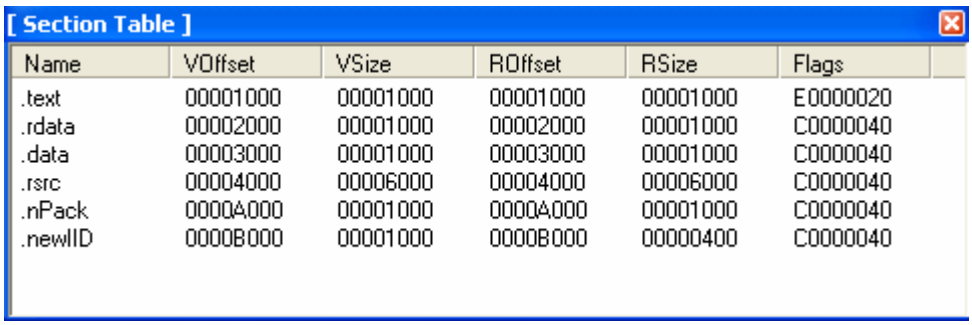
می بینید که فایل دامپ شده بی هیچ مشکلی اجرا می شود.
در مورد اینکه با چه دستوری وارد OEP می شویم.باید بگویم که هر دستوری ممکن است ما را به OEP ببرد.ممکن است یک Jump باشد.ممکن است یک Call باشد.یا اینکه مثل این مورد یک Ret ما را به OEP ببرد.
خوب،حالا باید سکشن nPack را از فایل آنپک شده خودمان پاک کنیم.چرا 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:



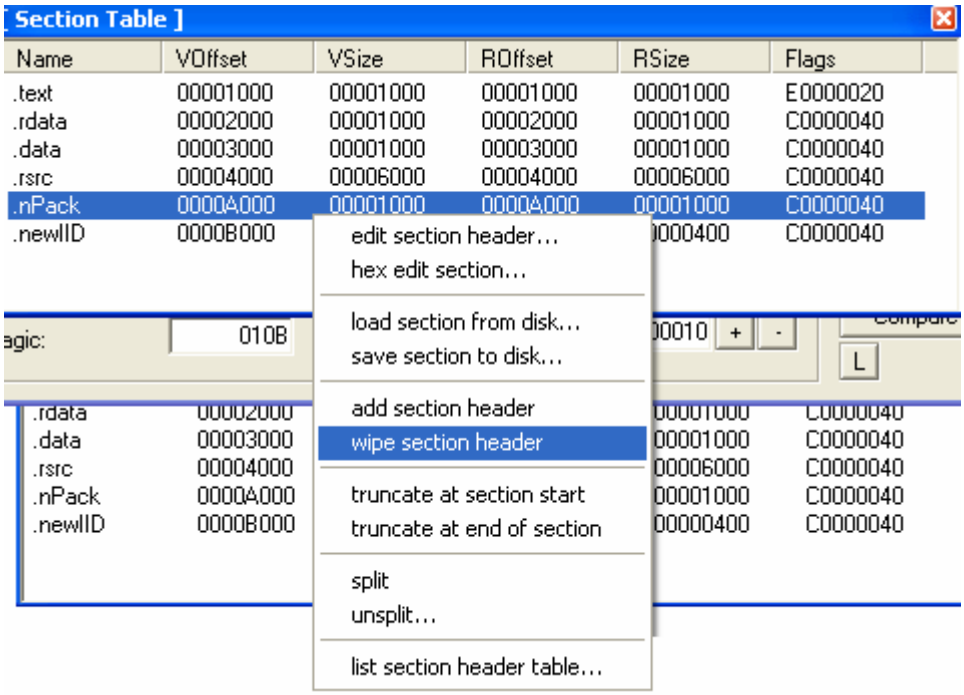
گزینه PE Editor را بزنید و فایل خود را انتخاب کنید:



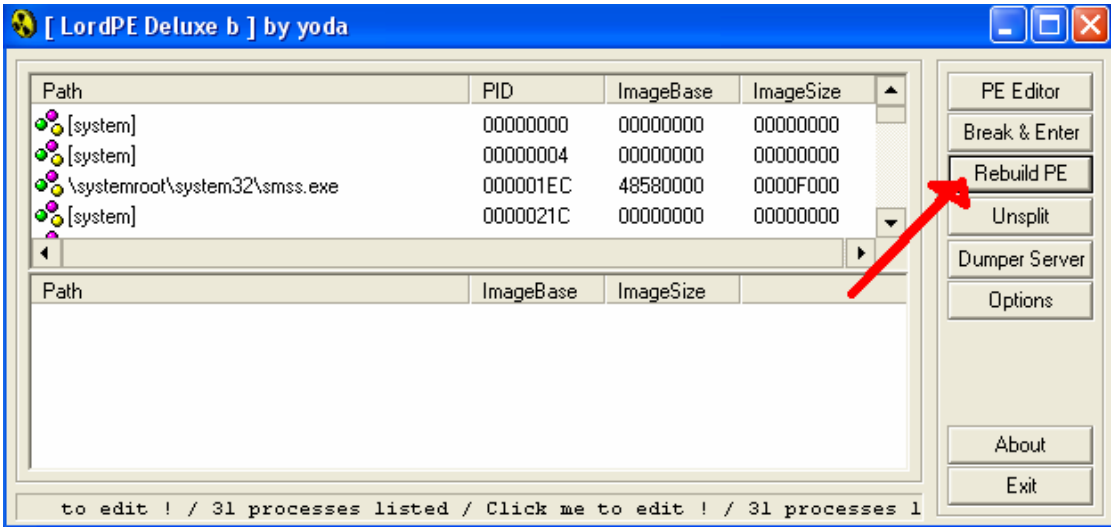
گزینه Sections را بزنید:



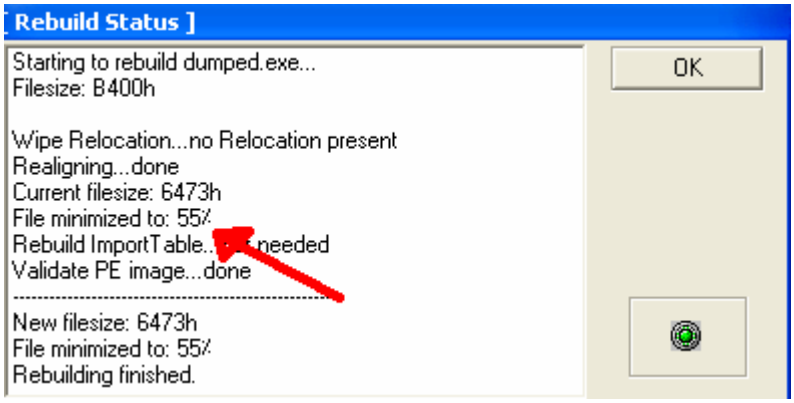
همانطور که می بینید یک سکشن جدید به نام newIID. توسط پلاگین OllyDump به فایل آنپک شده اضافه شده که حاوی Import های ماست.خوب،همانند تصویر بر روی سکشن nPack. کلیک راست کرده و گزینه Wipe Section Header را بزنید:



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



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



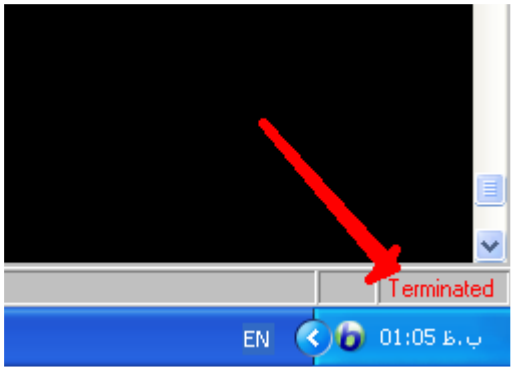
همانطور که می بینید فایل دوباره ساخته شده ما تنها ۵۵% حجم فایل اولیه را دارد.یعنی ۴۵% درصد از حجم فایل کم شد ☺

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

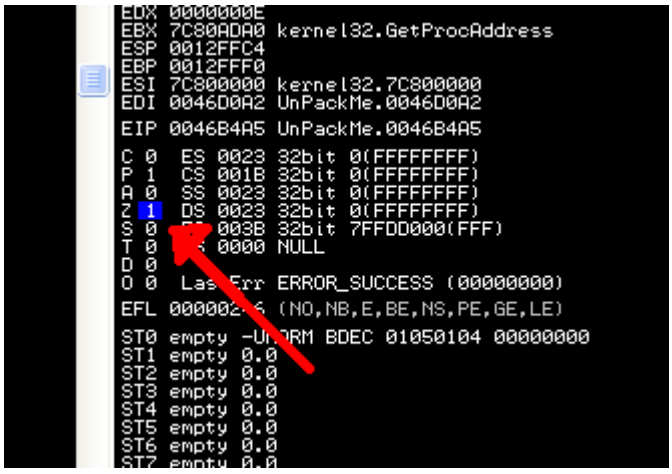
فایل را در داخل یک OllyDBG بدون پلاگین باز کند و برنامه را با کلید F9 اجرا کنید:



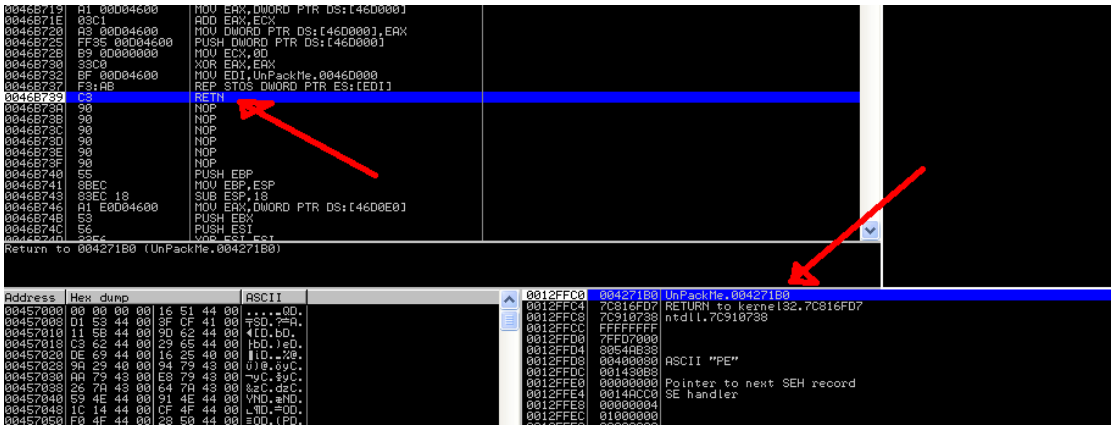
همانطور که می بینید برنامه Terminate شده...اما برنامه که خارج از OllyDBG بدون هیچ مشکلی اجرا می شد؟پس مشکل چیه؟بهتر است برنامه را Restart کرده و این بار با کلید F8 برنامه را ادامه دهیم تا به این خط برسیم:

```
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 D8D04600 CALL DWORD PTR DS:[46D0D8]   kernel32.IsDebuggerPresent
0046B4A2  83F8 01       CMP EAX,1
0046B4A5  75 08         JNZ SHORT UnPackMe.0046B4AF
0046B4A7  6A 00         PUSH 0
0046B4A9  FF15 D0D04600 CALL DWORD PTR DS:[46D0D0]   kernel32.ExitProcess
0046B4AF  6A 18         PUSH 18
0046B4B1  6A 40         PUSH 40
0046B4B3  FF15 D4D04600 CALL DWORD PTR DS:[46D0D4]   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
```

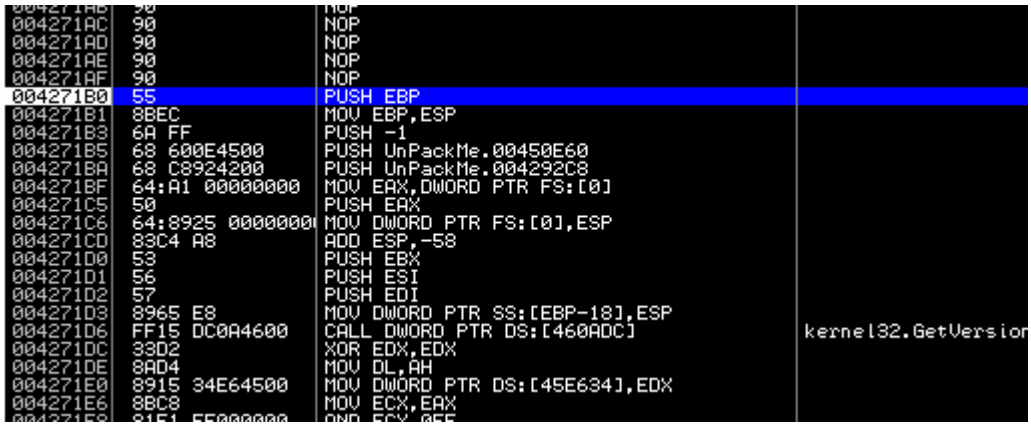
همانطور که می بینید برنامه در این خط از تابع IsDebuggerPresent استفاده می کند.و اگر مقدار بازگشتی این تابع برابر یک باشد.برنامه Terminate می شود.لذا حتما باید پرشی که در خط 0046BA45 قرار دارد.حتما باید انجام شود.در غیر این صورت به تابع ExitProcess می رسیم و از برنامه خارج می شویم ☹
پس دو بار دیگر کلید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



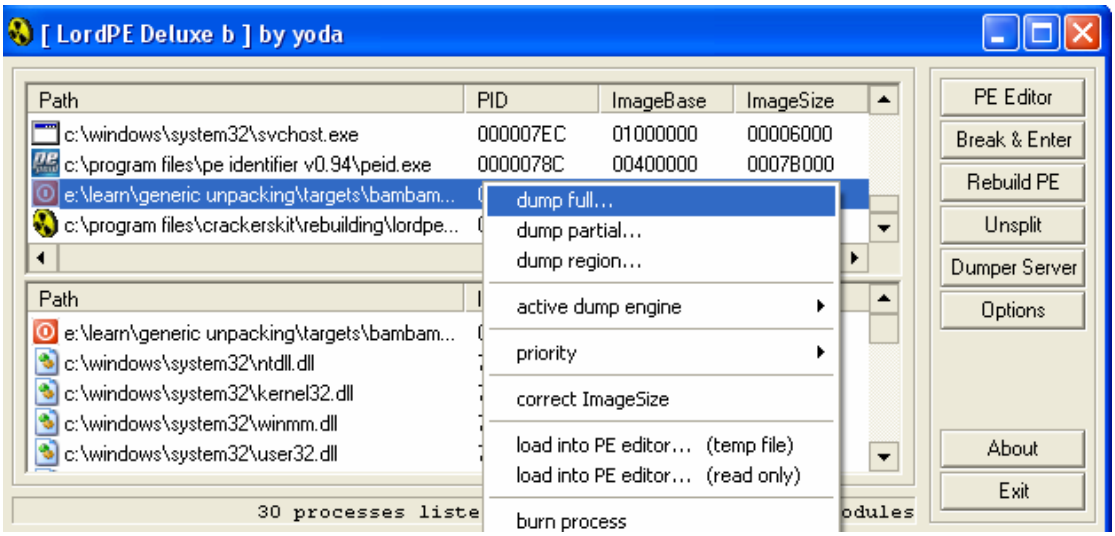
الان مقدار Z-flag در کامپیوتر من برابر ۱ است.یکبار بر روی آن کلیک می کنیم تا مقدار آن صفر شود.به این ترتیب این پرش،انجام می شود ☺
حالا اگر برنامه را با کلید F9 اجرا کنید.می بینید که برنامه بدون هیچ مشکلی اجرا می شود.حالا باید OEP را پیدا کنیم.برای یافتن OEP هم از همان روش قبلی استفاده می کنیم.یعنی آنقدر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



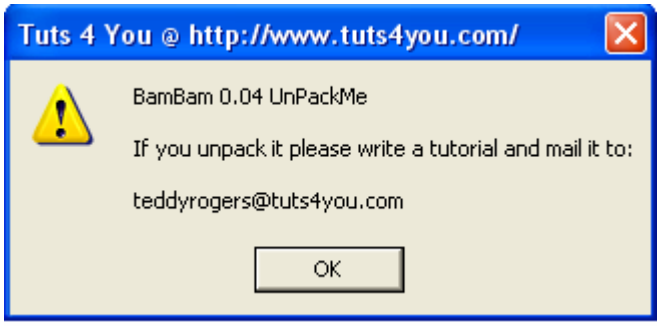
همانطور که می بینید این دستور ما را به خط 4271B0 می برد.یعنی از سکشن Bedrock به سکشن text می رویم.یعنی با این دستور به OEP می رویم☺



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 آنپک شده ما بدون هیچ مشکلی اجرا می شود ☺

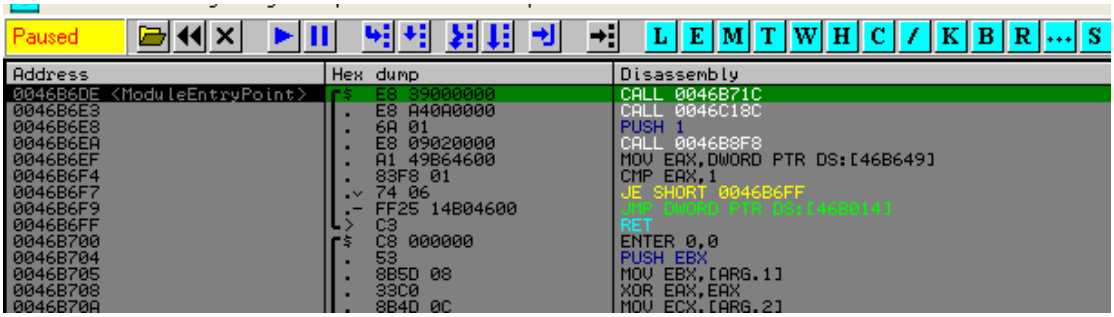

Tuts 4 You @ http://www.tuts4you.com/
BamBam 0.04 UnPackMe
If you unpack it please write a tutorial and mail it to:
teddyrogers@tuts4you.com
OK

CrypKey SDK

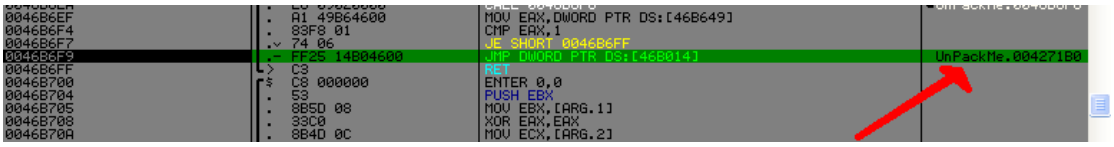
این بار هم برای یافتن OEP از همان روش استفاده می کنیم.یافتن OEP در این فایل بسیار راحت است.به طوری که با سه یا چهار بار زدن کلید F8 به OEP می رسیم ☺ از بین سکشن های برنامه،این دو سکشن برای ما اهمیت بیشتری دارد:

.text → 401000 → 44A17F
Have → 46B000 تا 46D000

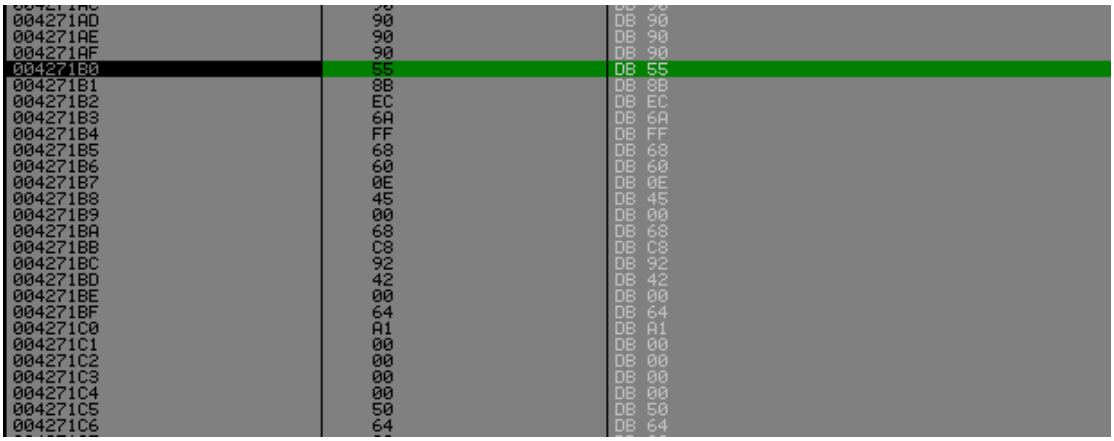
برنامه را در OllyDBG باز کنید:



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



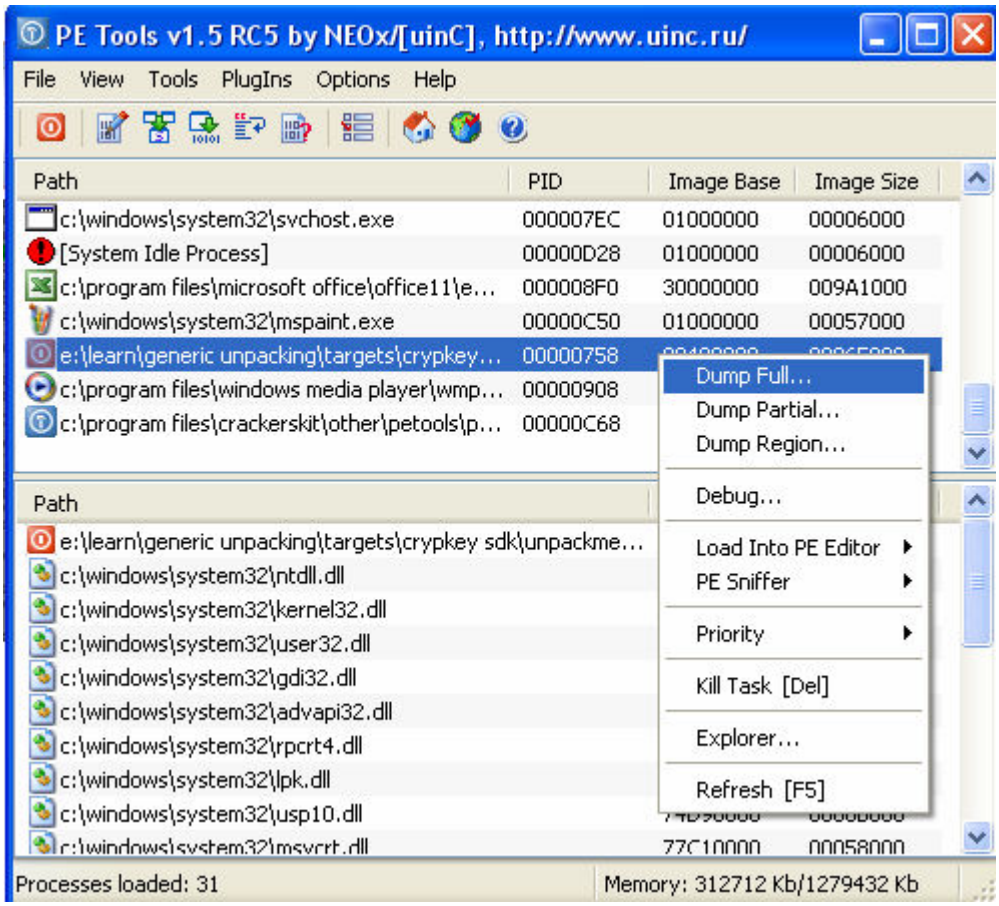
همانطور که می بینید این Jump ما را به خط 4271B0 می برد.یعنی از سکشن Have به سکشن text می رویم.پس این پرش به OEP می رود. ☺



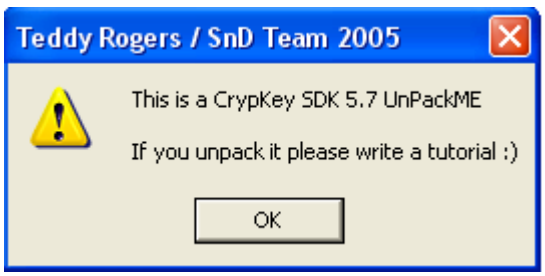
کلیدهای CTRL + A را بزنید،تا کدهای این سکشن آنالیز شود.

```
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 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
```

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 ☺

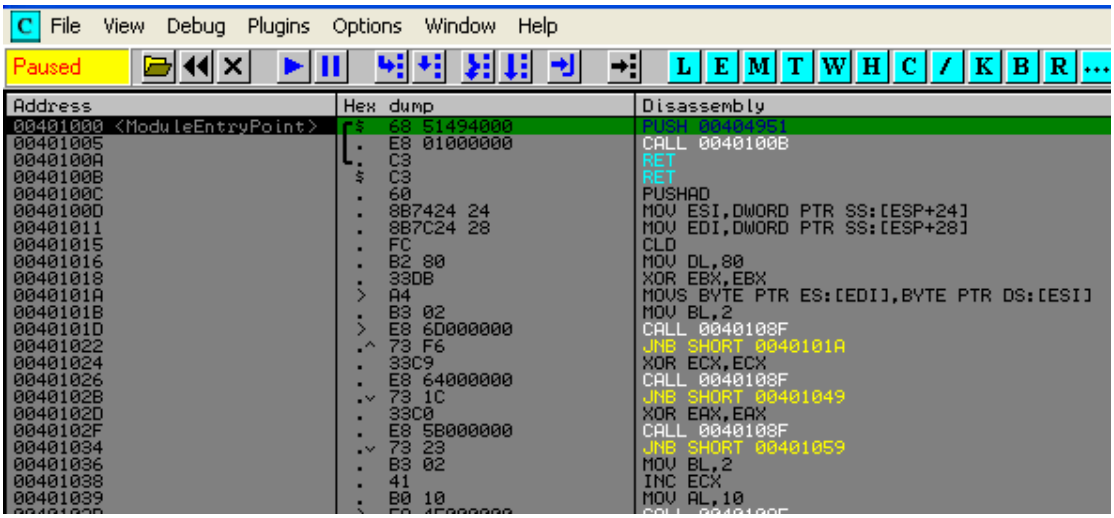


KByS Packer

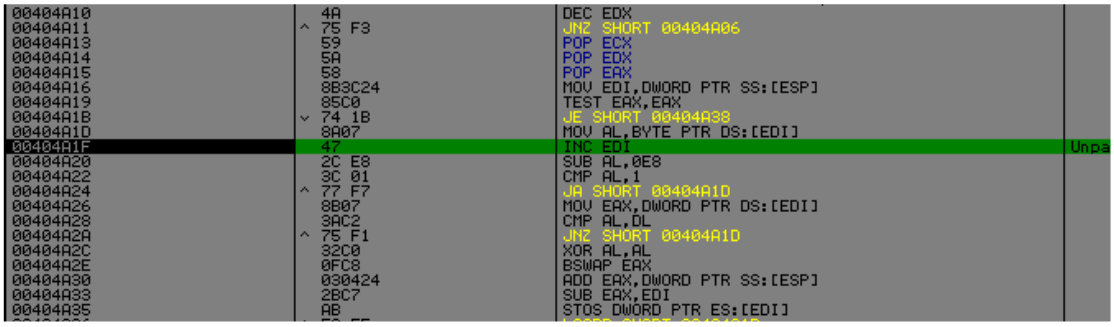
توضیحی کوتاه در مورد پکر:این پکر یکی از پکرهای مورد علاقه منه...که توسط یک نفر با نام مستعار(یا شـاید هـم واقعـی) Shoooo نوشته شده.برای پک کردن فایل با این برنامه تنها کافیست فایل مورد نظر خود را به داخل برنامه بکشید.(Drag کنید.)
این دو سکشن برای ما اهمیت بیشتری دارند:

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

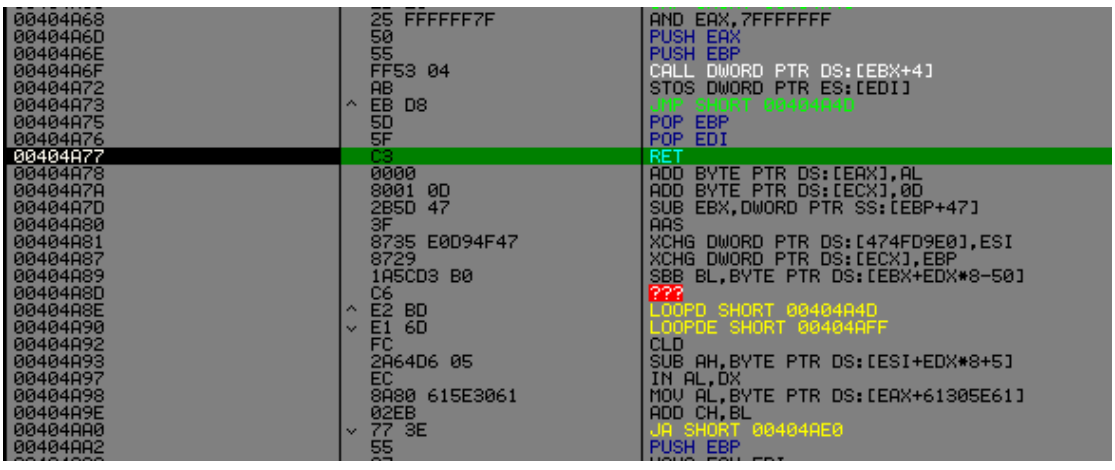
فایل مورد نظر را در OllyDBG باز کنید:



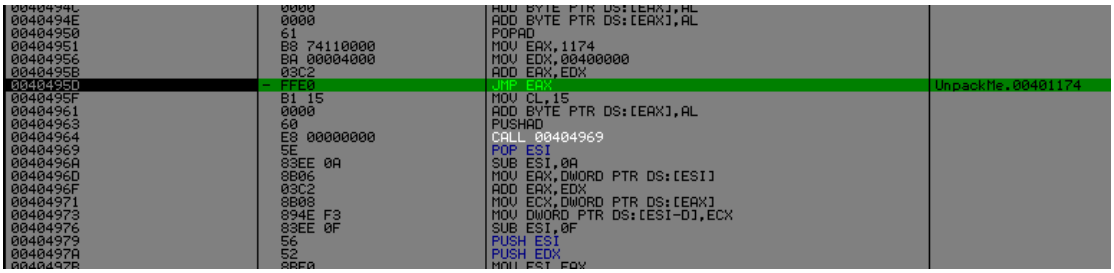
حالا هم همانند مثال های قبل با کلید F8 آنقدر برنامه را ادامه می دهیم.تا به OEP برسیم.با F8 به جلو بروید تا به اینجا برسید:



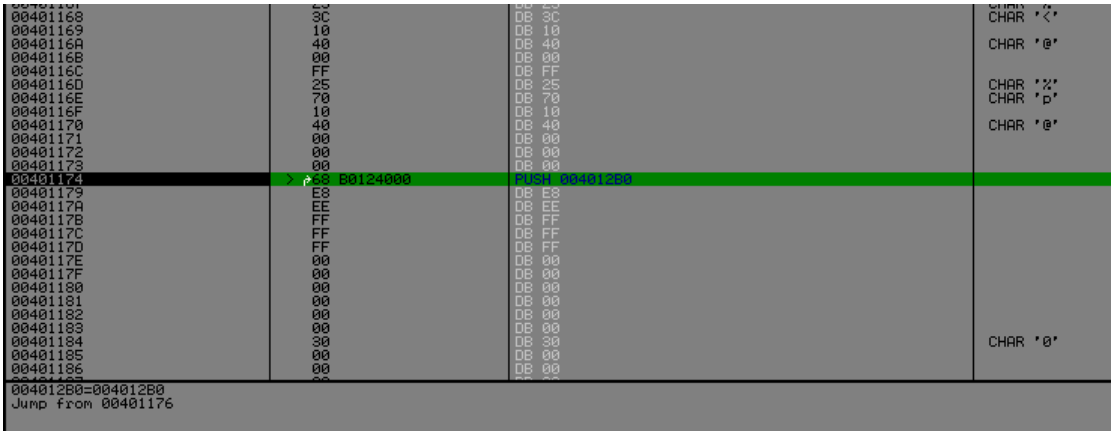
در اینجا ما در این حلقه گیر می کنیم.برای اینکه راحت تر این حلقه را رد کنیم.بر روی دستور RET که چند خط پایینتر در آدرس 00404A77 قرار دارد.یک Break Point می گذاریم و برنامه را با کلید F9 اجرا می کنیم:



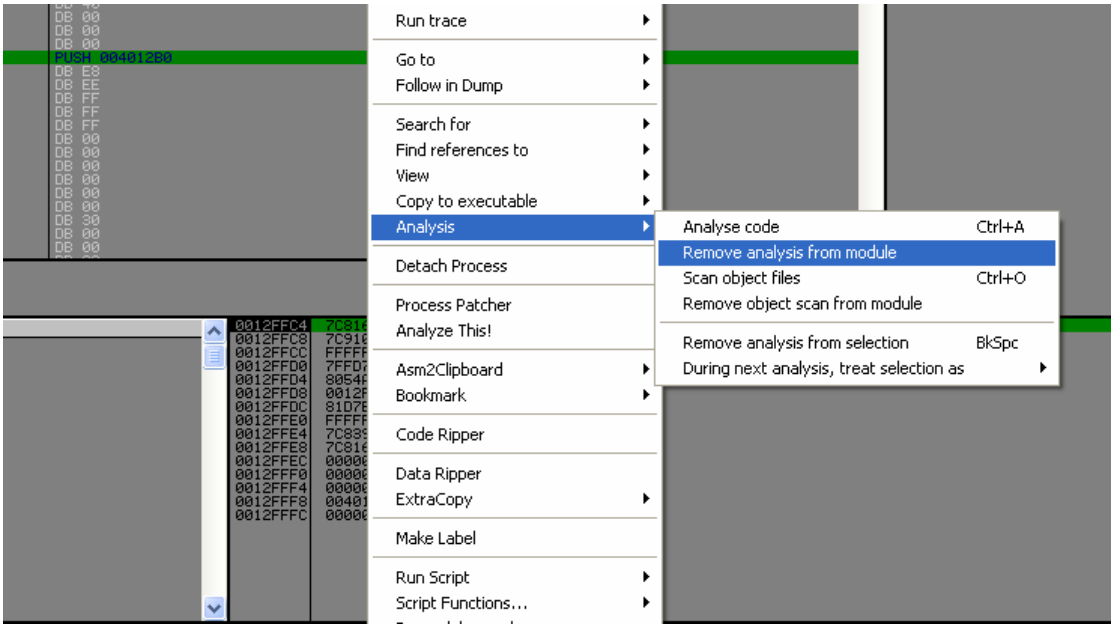
حالا هم با F8 برنامه را ادامه می دهیم تا به اینجا برسیم:

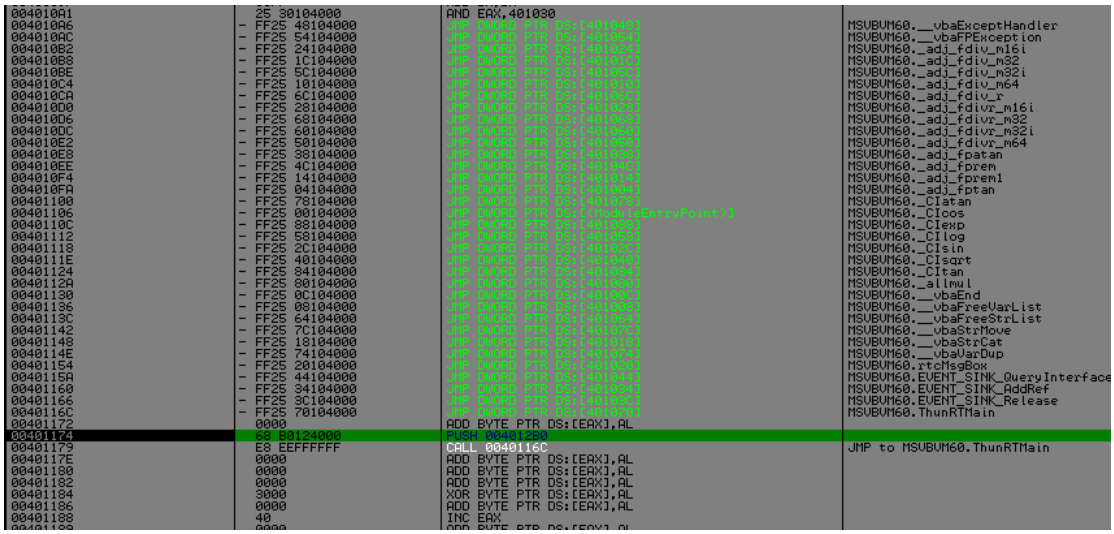


این دستور JMP EAX ما را از سکشن Shoooo به سکشن text یعنی خط 401174 می برد.پس ممکن است با این دستور به OEP برویم.یکبار کلید F7 را بزنید تا همه چیز روشن شود.



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

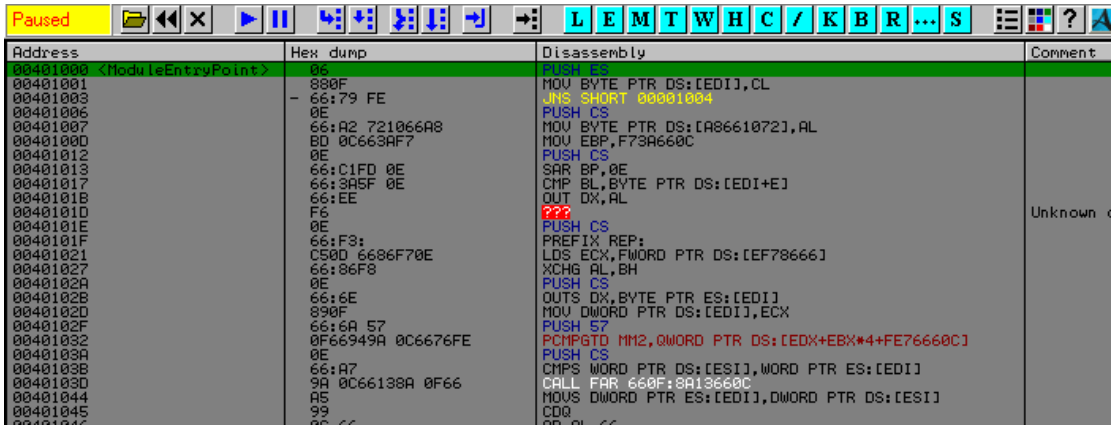




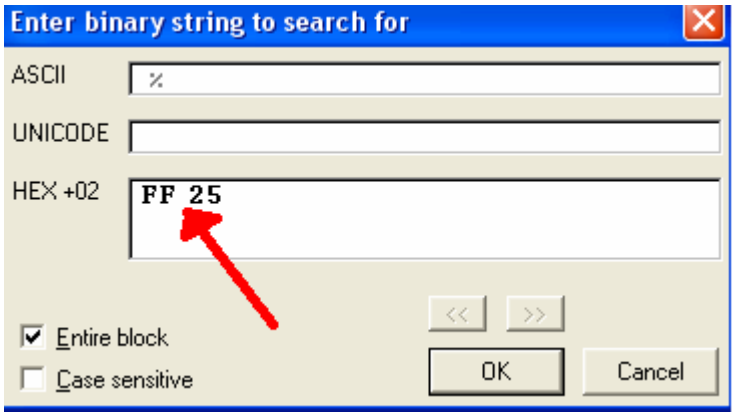
همانطور که می بینید در این برنامه Jump Table درست بالای OEP قرار دارد.این یعنی برنامه ما در ویژوال بیسیک نوشته شده.

راه حل دیگر:

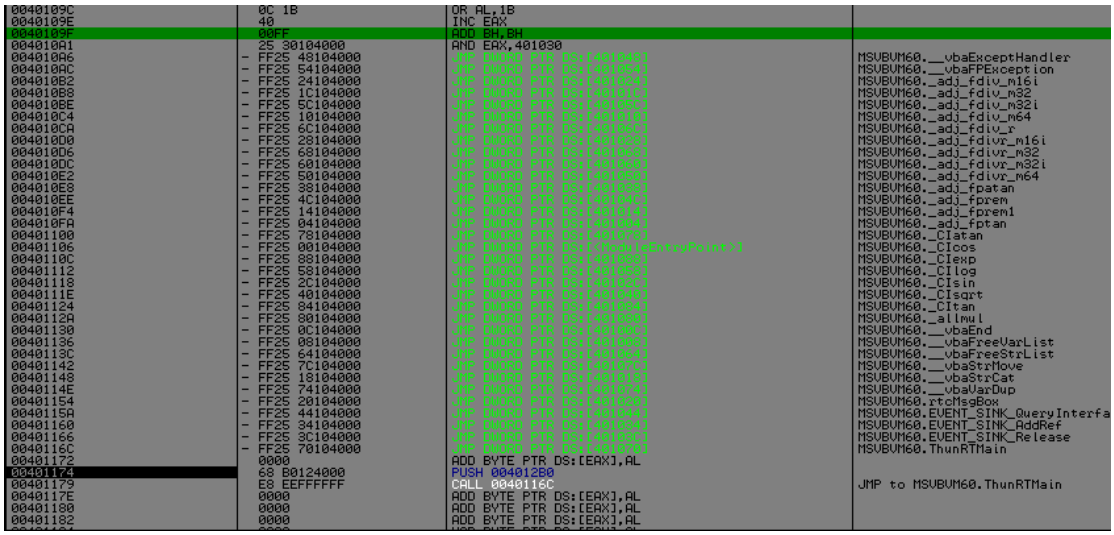
راه حل دیگر برای یافتن OEP این است که ما از خصوصیات زبان برنامه نویسی Visual Basic استفاده کنیم.یعنی اول برنامه را کامل با کلید F9 اجرا کنید و بعد کلیدهای CTRL + G را بزنیم و به ابتدای سکشن text یعنی 401000 برویم.و در آنجا به دنبال Jump Table باشیم.



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

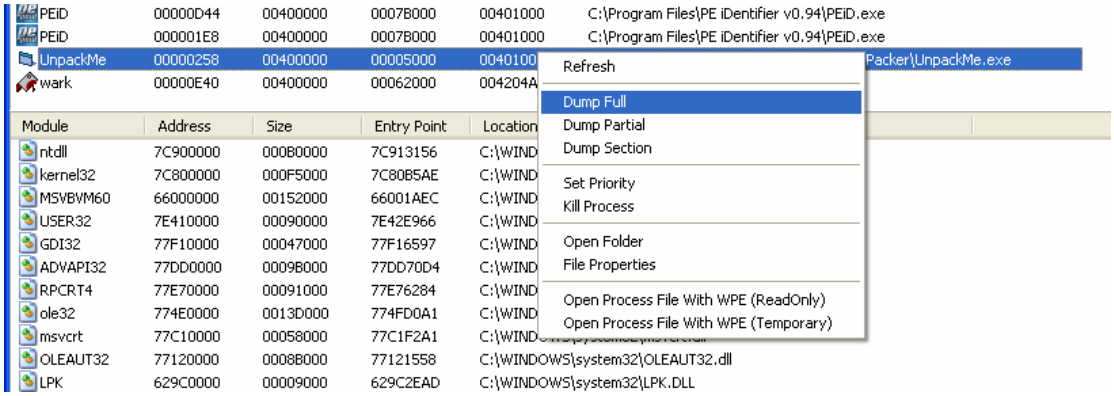


کلید OK را بزنید.

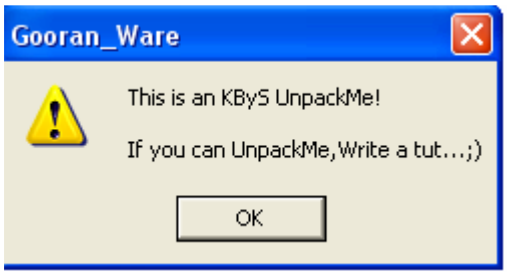


همانطور که می بینید به Jump Table رسیدیم و درست زیر Jump Table یک دستور Push وجود دارد که همان OEP ماست.

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:



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.

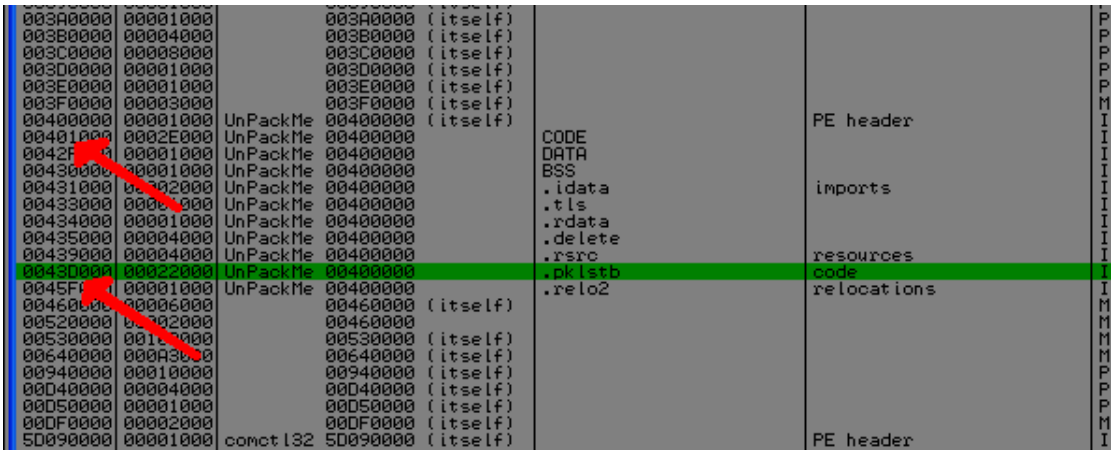


فایل آنپک شده ما بدون هیچ مشکلی اجرا می شود ☺

PKLite

این پکر هم بسیار راحت OEP آن پیدا می شود.

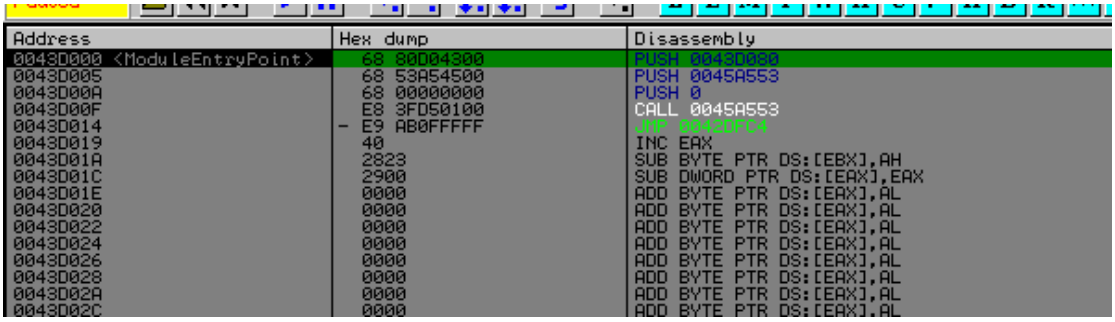
برنامه را در OllyDBG باز کنید و یک نگاهی به Memory Map (کلید های ALT + M)می اندازیم:



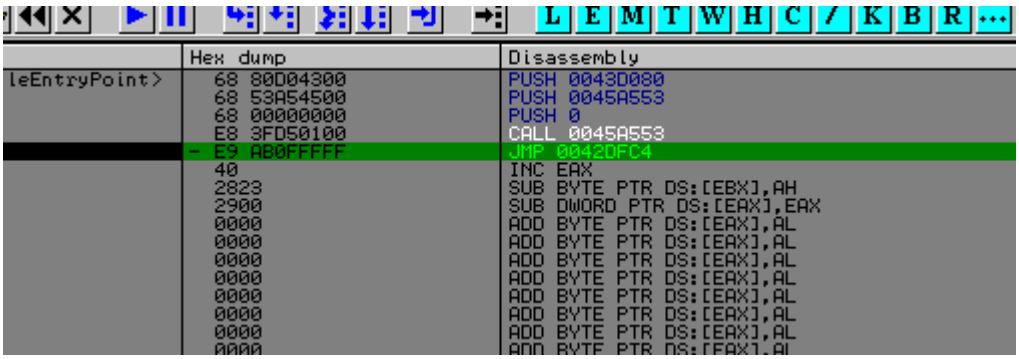
همانطور که در Memory Map می بینید دو سکشن CODE و pklstb در این آدرس قرار دارند:

.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

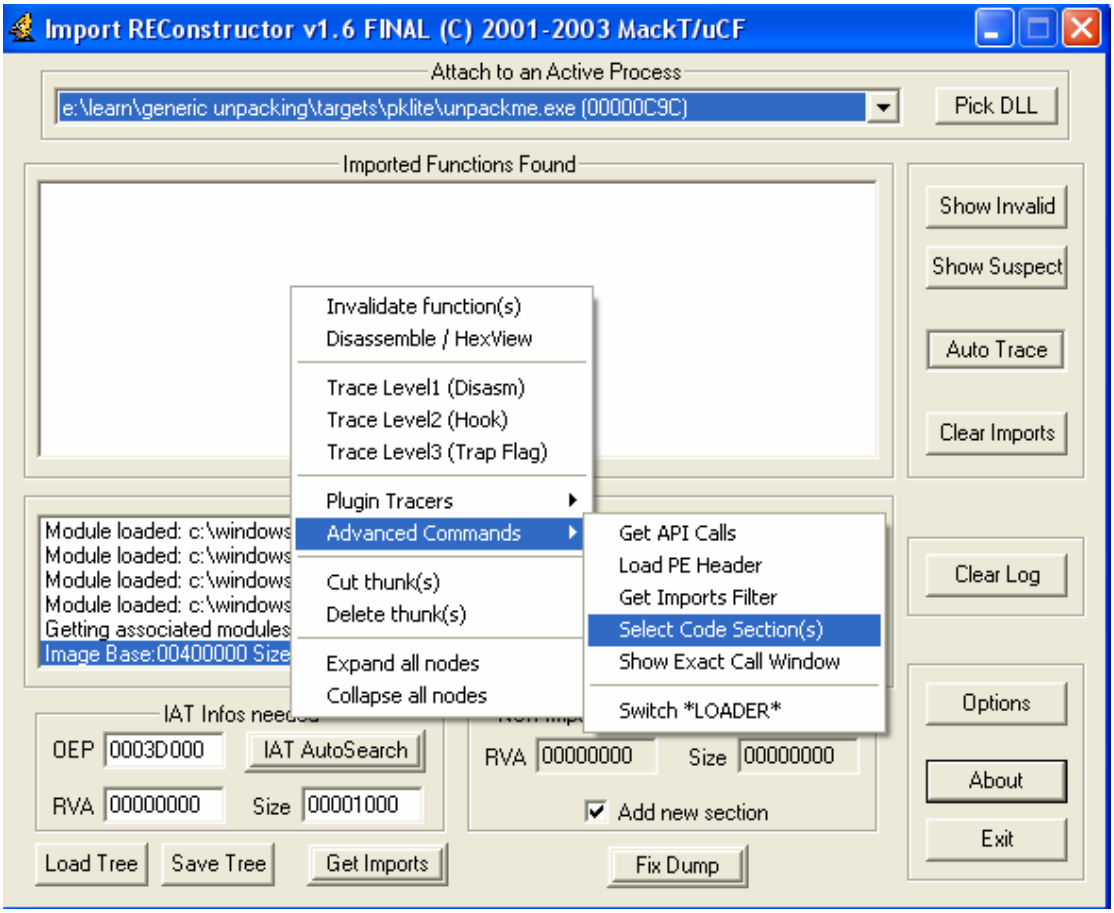


چند بار کلید F8 را بزنید تا به خط 0043D014 برسید:

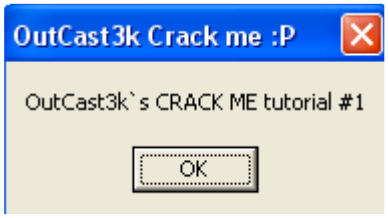


با این پرش از سکشن pklstb به سکشن CODE می رویم.یعنی این پرش احتمالا به OEP می رود. ☺

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 ☺

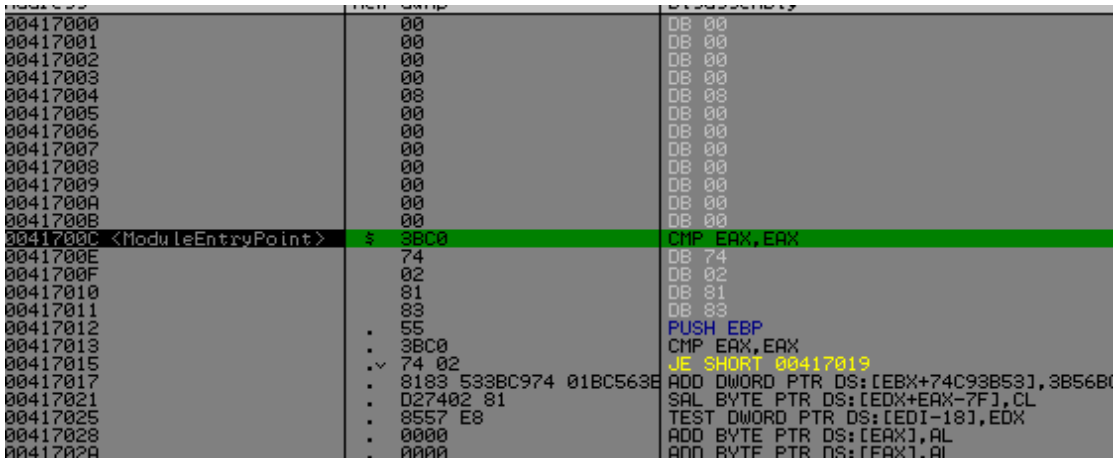


Exe32Pack

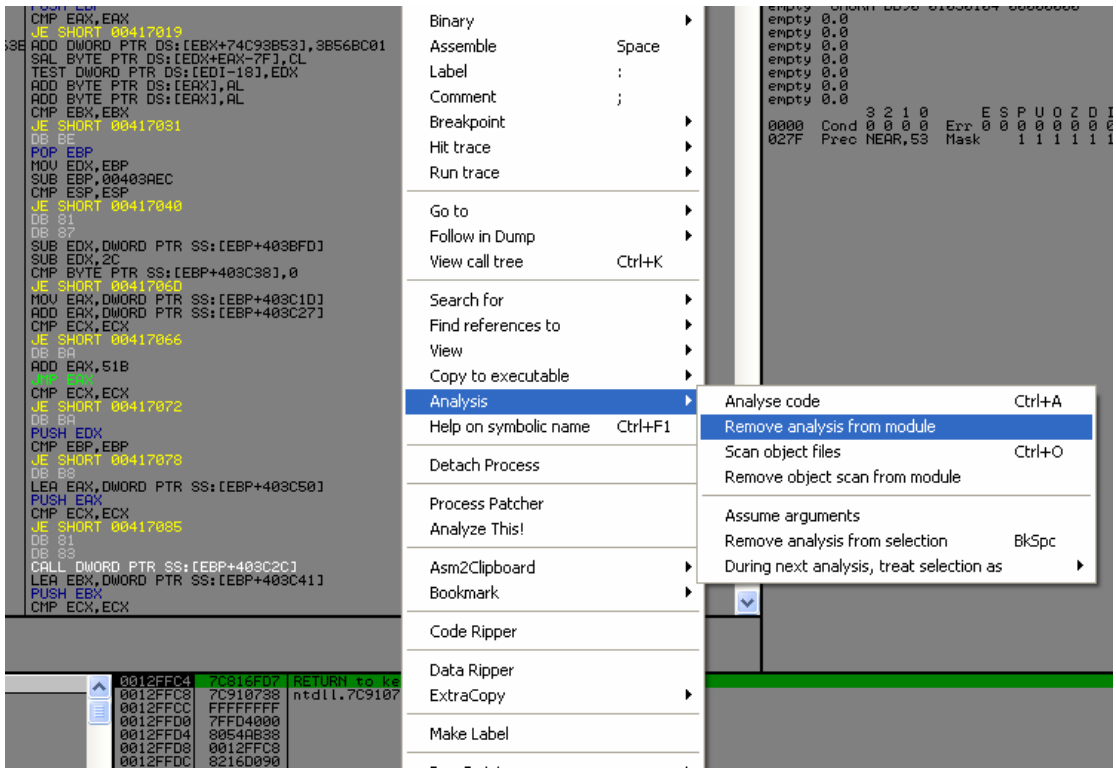
در مثال های قبل برای یافتن OEP آنقدر در برنامه به جلو رفتیم تا به OEP رسیدیم.اما در این مثال می خواهیم از روش راحت تری استفاده کنیم.روشی که در بیشتر **پکرها** جواب می دهد.
این دو سکشن از فایل برای ما اهمیت بیشتری دارند:

.text → 40A000 تا 40C000
.f → 417000 تا 418851

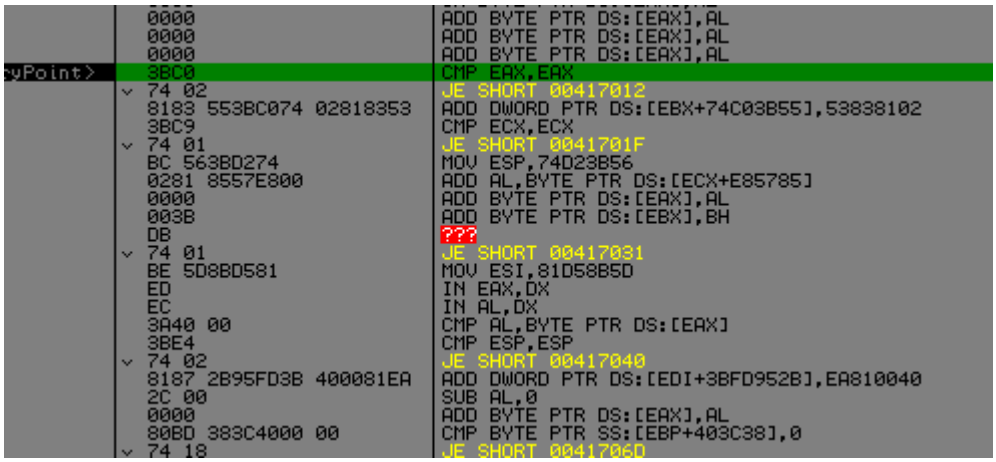
خوب،برنامه را در OllyDBG باز کنید:



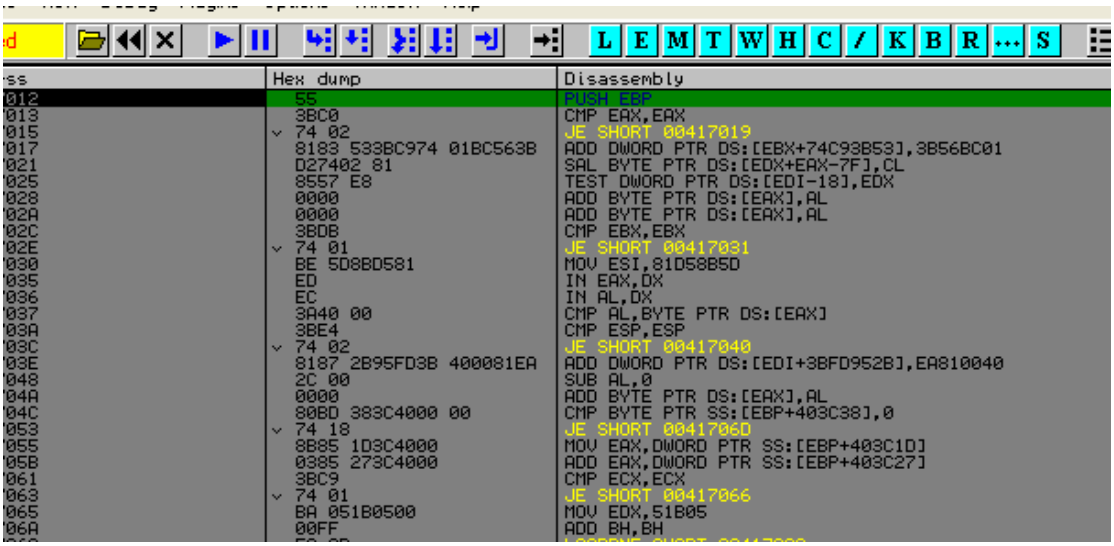
همانطور که می بینید کدهای آنالیز شده چندان جالب نیست ☹ اصولا بهتر است کدهای پکر را آنالیز نکنیم و وقتی به فایل اصلی خودمان رسیدیم،آن موقع آنالیز کنیم.برای ازبین بردن آنالیز این کدها همانند تصویر عمل کنید:



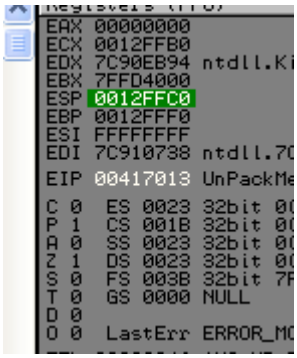
حالا کدهای ما بهتر می شود:



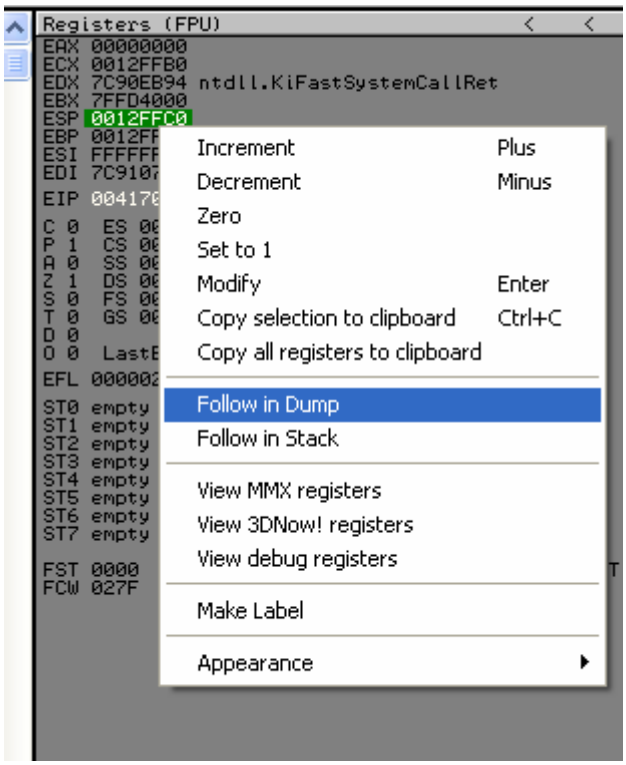
خوب،دو بار کلید F8 را بزنید تا به اینجا برسید:



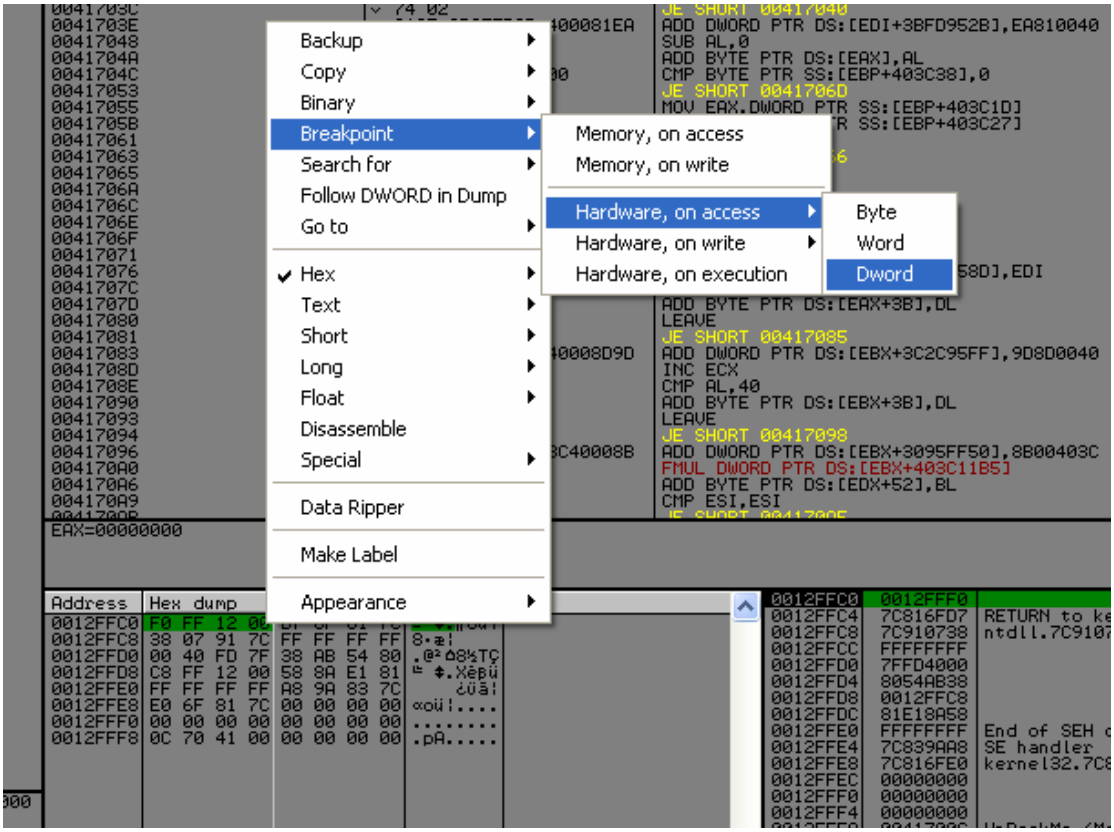
خوب قبل از هر کاری اول بگویم که رجیستر ESP چیست؟رجیستر ESP نگهدارنده خطی است که در Stack قرار است مورد استفاده قرار گیرد.(همانند رجیستر EIP که نشاندهنده خطی است که قرار است در پنجره Dissembler اجرا شود.)وقتی مقداری وارد Stack می شود یا از آن خارج می شود.مقدار رجیستر ESP تغییر می کند.
وقتی یک مقدار در 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؟اصلا چه نیازی هست؟...خوب،چون این اولین مقداری بوده که وارد Stack شده...**احتمالا** آخرین مقداری خواهد بود که از حافظه خارج می شود...یعنی در آخر کدهای پکر ما این مقدار از Stackخارج می شود...به این ترتیب من می توانم بفهمم که انتهای کدهای پکر ما کجاست و به نزدیک های فایل اصلی(OEP) برسم ☺
حالا یکبار کلید F8 را بزنید تا این خط را اجرا کنید:



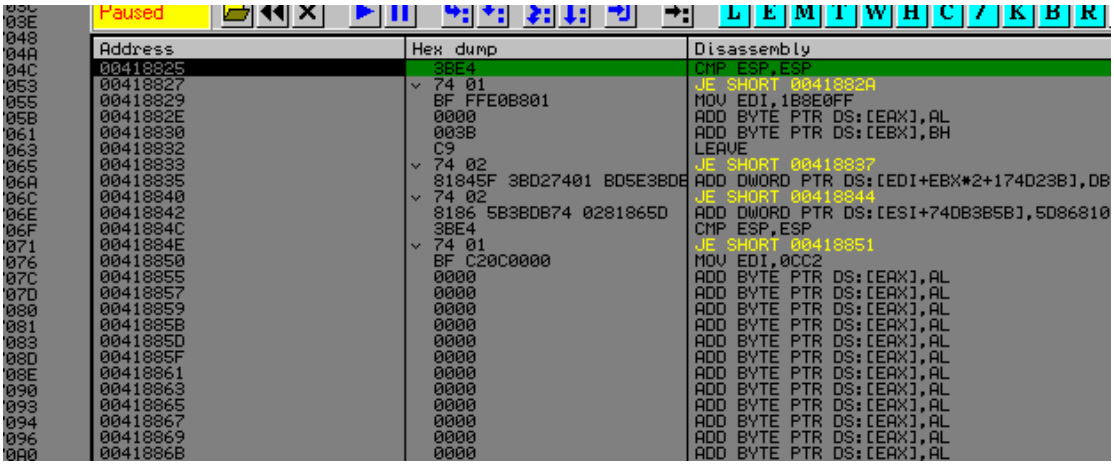
همانطور که می بینید مقدار رجیستر ESP تغییر کرده است.اگر من بر روی این رجیستر یک Hardware Breakpoint بگذارم،می توانم بفهمم که در کجا این مقدار از Stack خارج می شود.برای Hardware Breakpoint گذاشتن روی این آدرس،همانند تصویر عمل کنید:



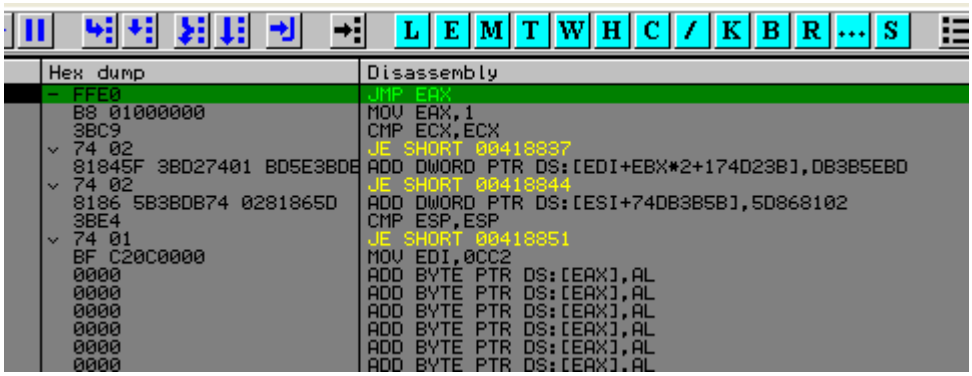
حالا در پنجره Dump بر روی آدرس مورد نظر(در اینجا 0012FFC0) Breakpoint hardware on access بگذارید،همانند تصویر:



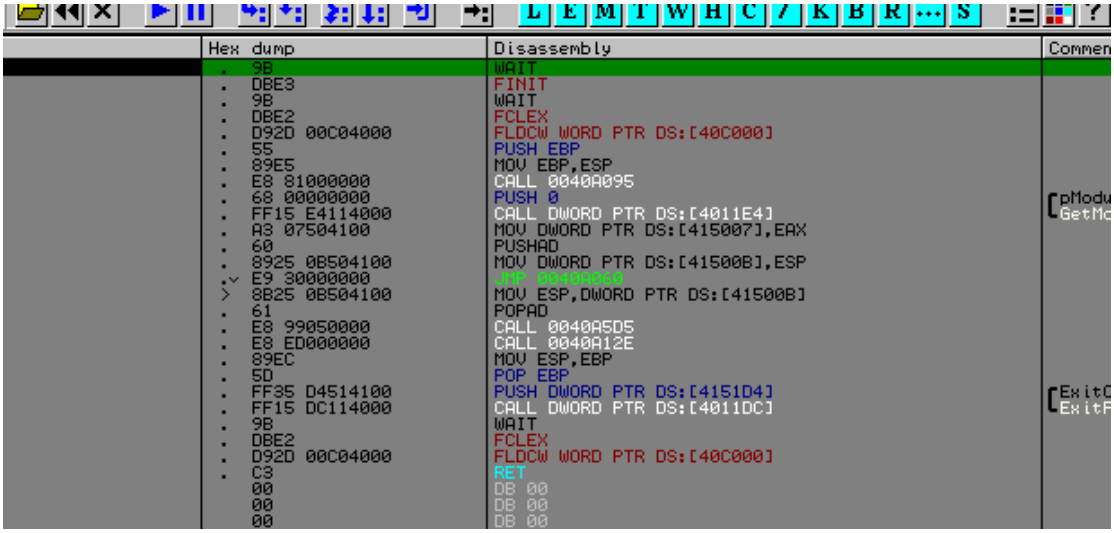
خوب...حالا برنامه را با کلید F9 اجرا کنید:



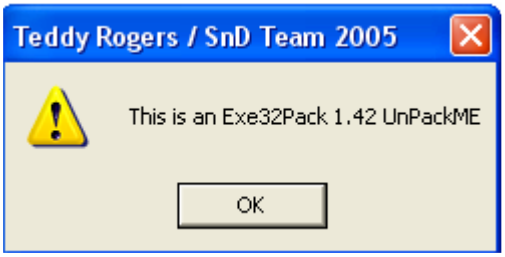
می بینید؟اینجا جایی است که آن مقدار از حافظه خارج شده است.بنابراین ما به آخر کدهای پکر رسیدیم.پس اگر چند بار کلید F8 را بزنیم و به طول کامل از کدهای پکر خارج می شویم و به OEP می رسیم.
دو بار کلید F8 را بزنید:



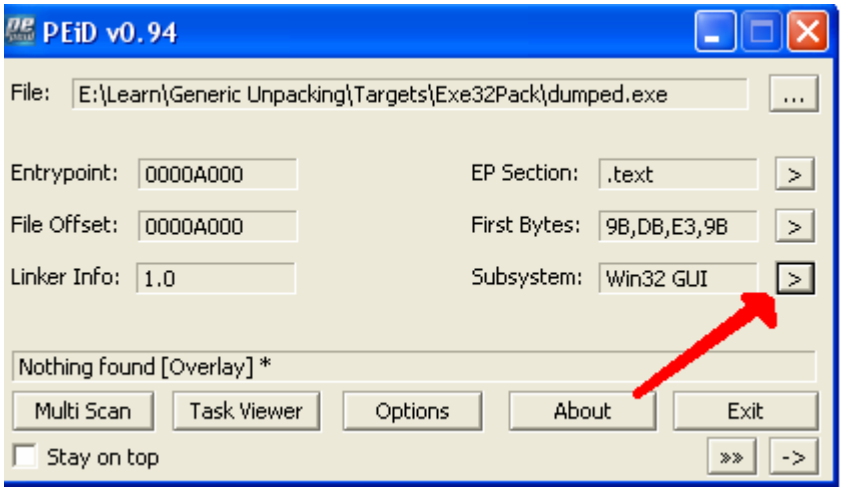
با این پرش از سکشن f. وارد سکشن text. می شویم.پس این پرش ما را به OEP می برد.یکبار کلید F7 را بزنید تا به OEP برسید،بعد هم کلیدهای CTRL + A را بزنید تا کدها آنالیز شود.



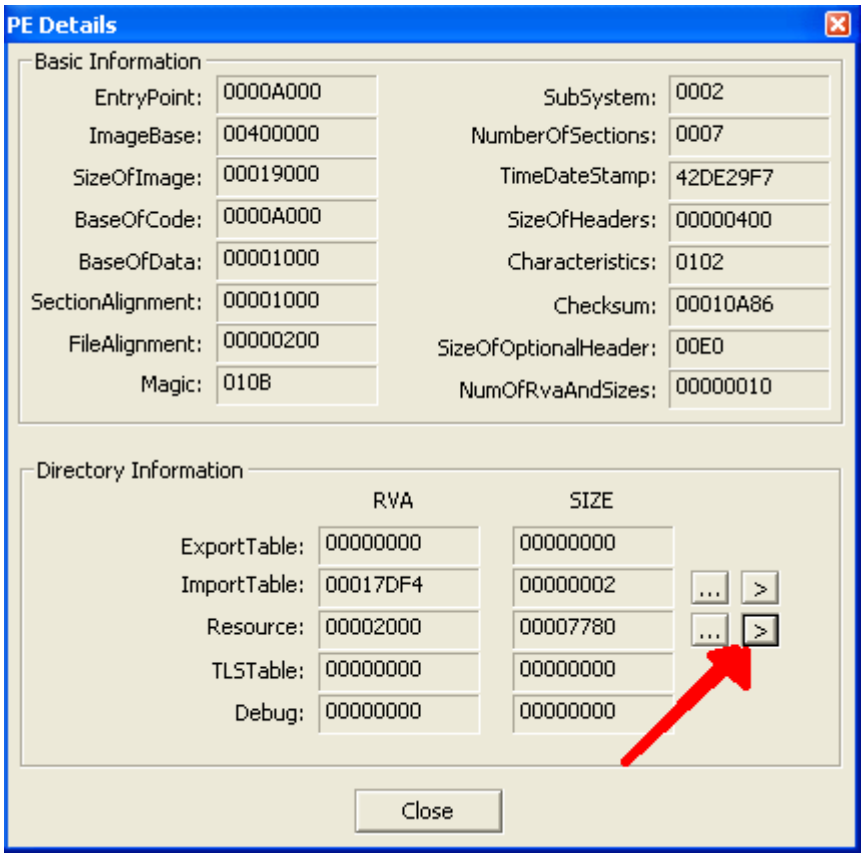
بله...ما به OEP رسیدیم...برنامه هم در MASM یا TASM نوشته شده.
حالا فایل را با OllyDump دامپ می گیریم:
کلیک راست ← Dump Debugged process
تیک گزینه Rebuild Import را بردارید و فایل را Dump کنید.حالا فایل دامپ شده را اجرا کنید:



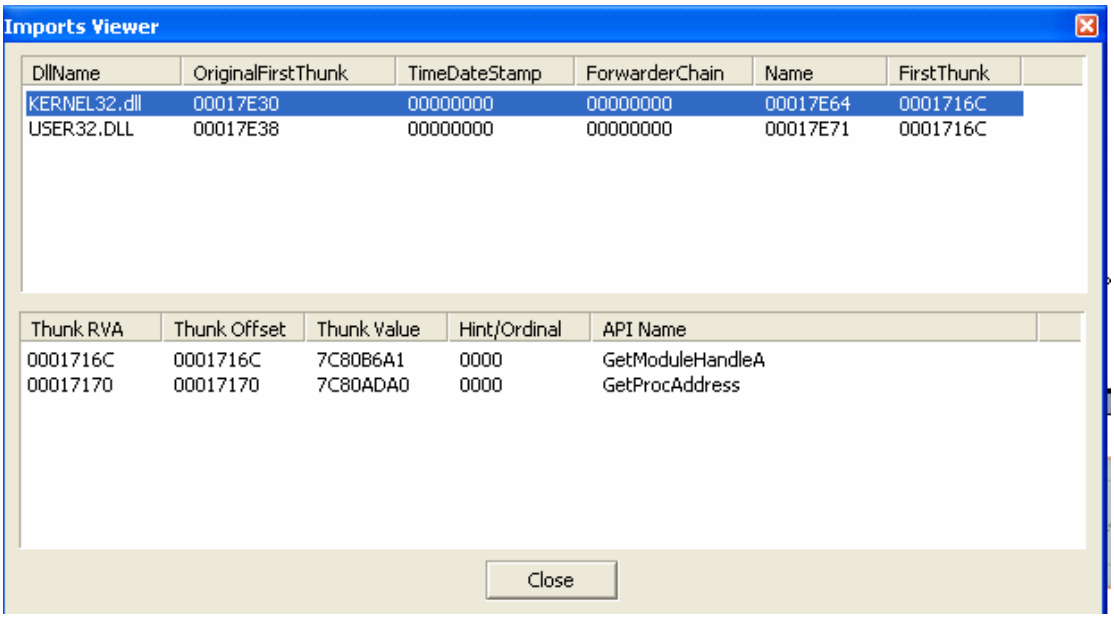
خوب،فایل آنپک شد...اما این فایل آنپک شده **فقط در سیستم شما اجرا می شود**،کافیست آن را در سیستم دیگر تست کنید،می بینید که اجرا نمی شود.چرا؟...چون این فایل فقط دامپ شده و اصلا IAT ندارد.پس قاعدتا نباید اجرا شود.دلیل اینکه این فایل در سیستم شما اجرا می شود،این است که آدرس توابع API هم به همراه فایل،دامپ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



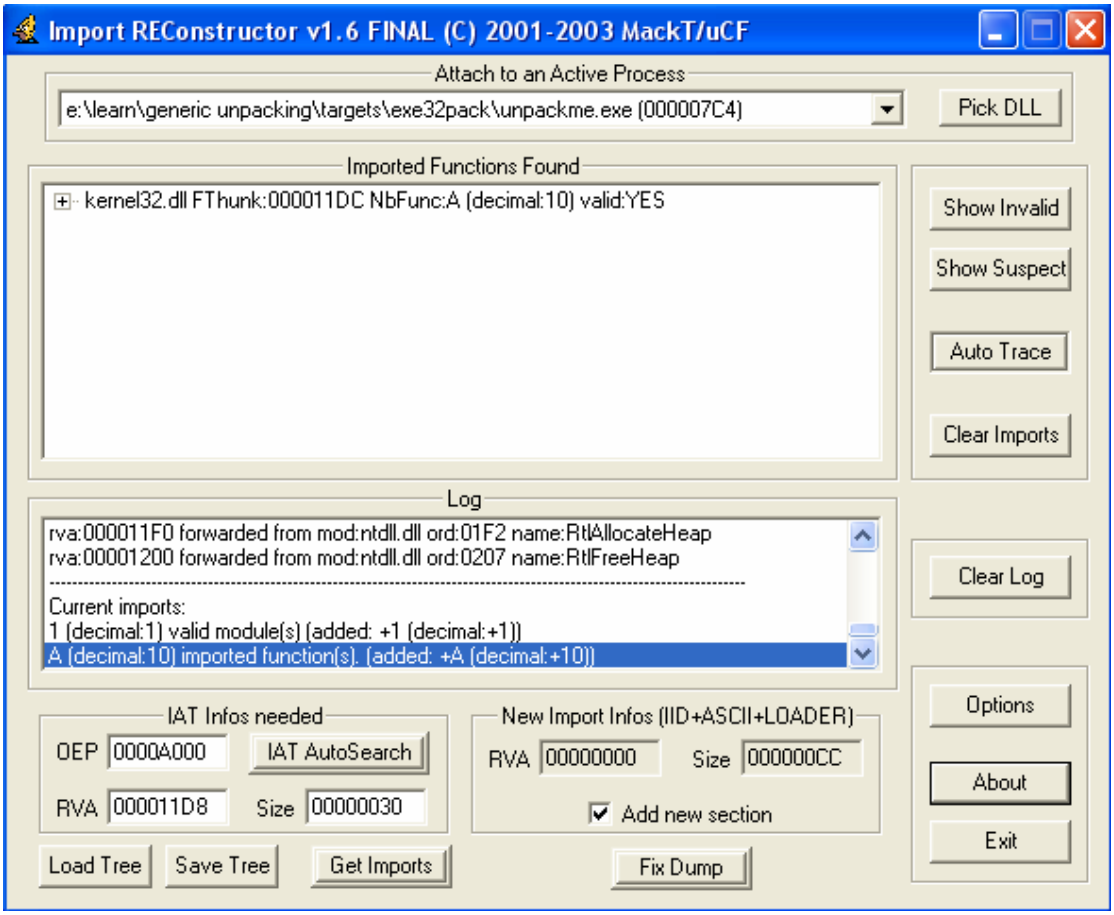
حالا این دکمه را بزنید:



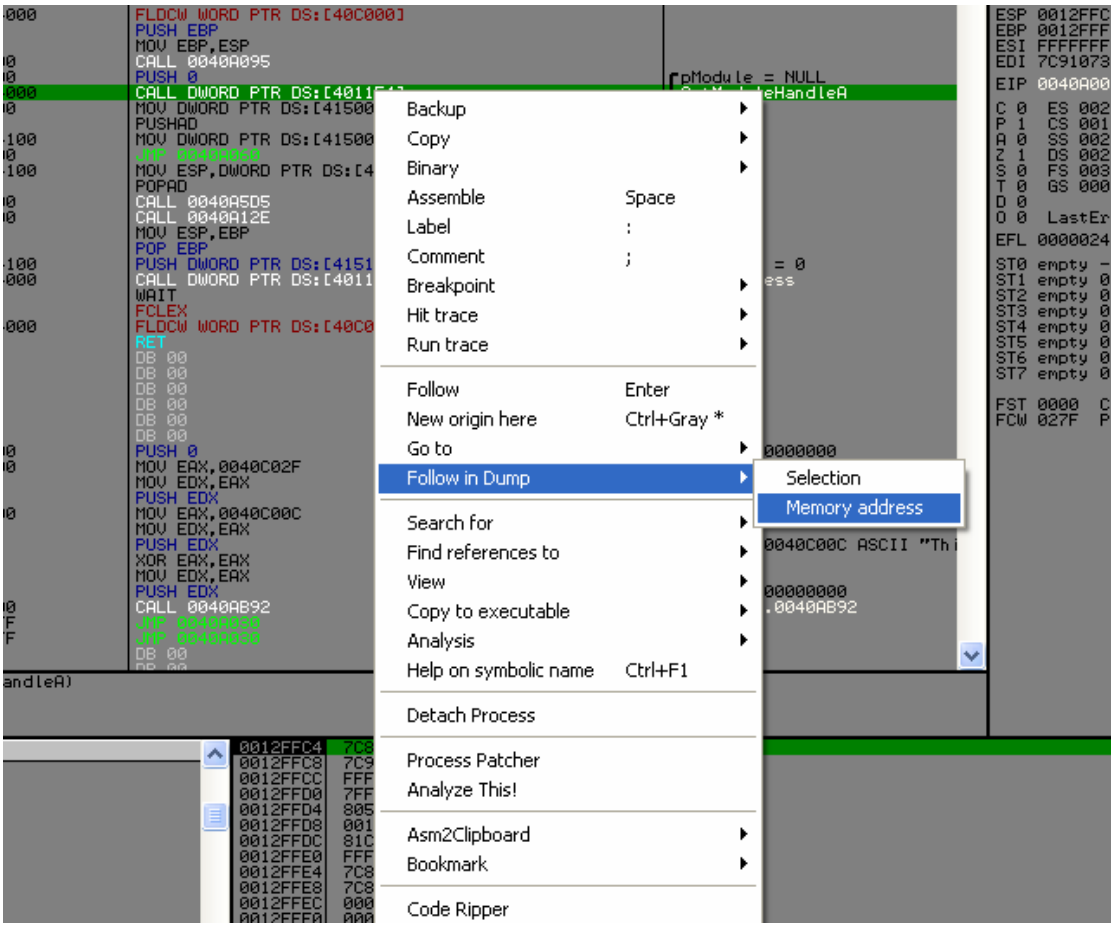
این IAT (در واقع Import Table) فایل ماست:

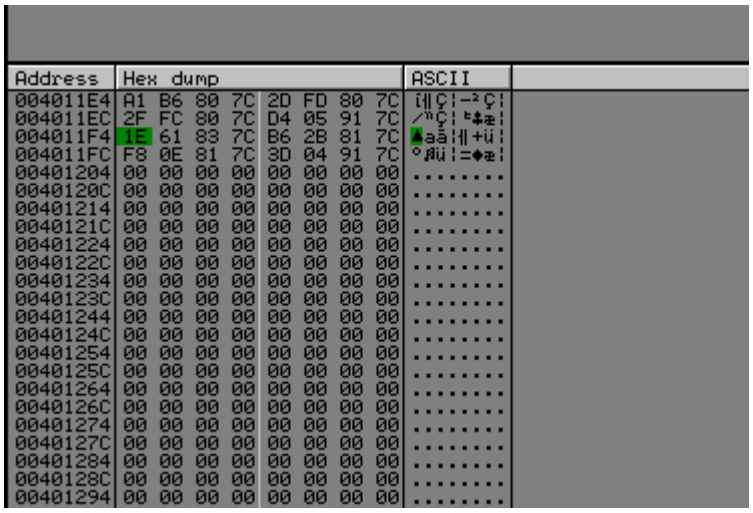


همانطور که می بینید فقط دو تابع GetModuleHandleA و GetProcAddress در فایل موجود است.و اصلا خبری از دیگر توابع API که برنامه استفاده کرده است(مثل MessageBoxA)،نیست.
پس لازم است IAT فایل را با ImportREC تعمیر کنیم.
برنامه ImportREC را باز کنید،پروسه مورد نظر را انتخاب کنید.OEP را بر حسب RVA به برنامه بدهید.(A000).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

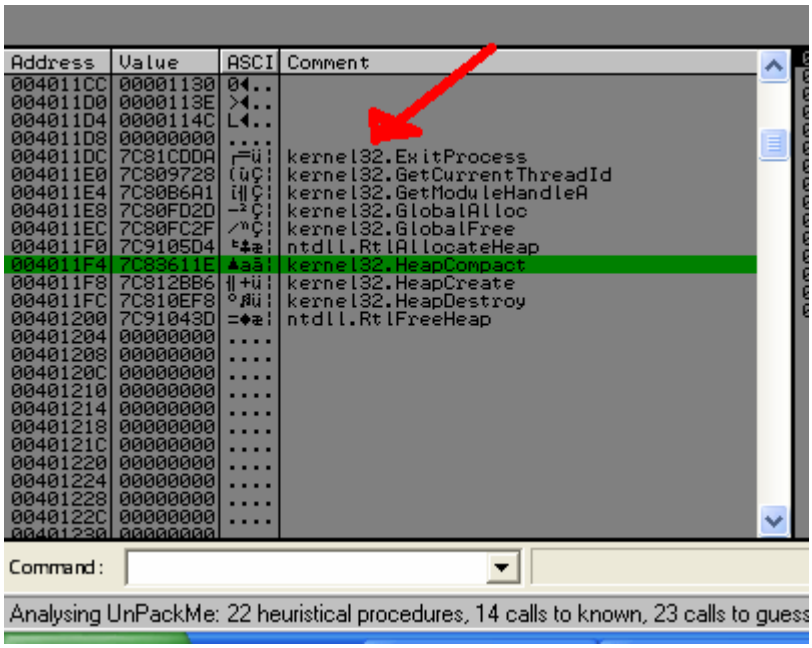


لعنتی ☹ ImportREC نتوانسته است IAT فایل مرا درست پیدا کند...از کجا فهمیدم؟...برنامه من از تابع MessageBoxA برای نمایش پیغام استفاده کرده بود.اما چنین تابعی در اینجا وجود ندارد.فقط ۱۰ تابع پیدا شده است که مربوط به فایل Kernel32.dll می باشد.
پس بهتر است خودمان محل شروع IAT برنامه و اندازه آن را به دست 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.

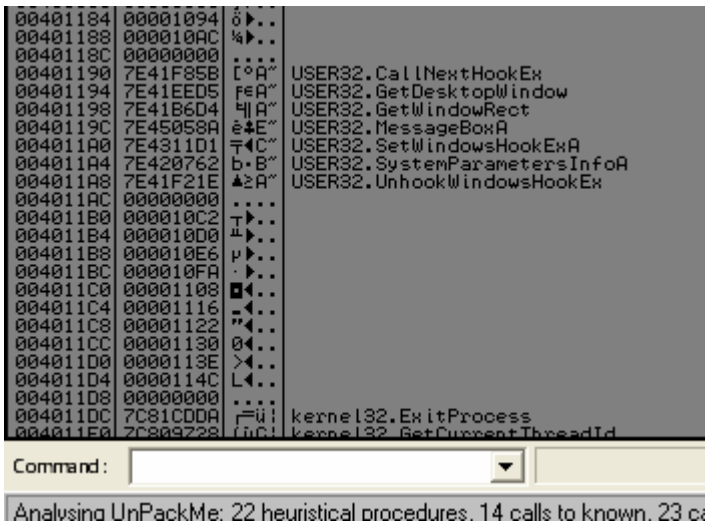




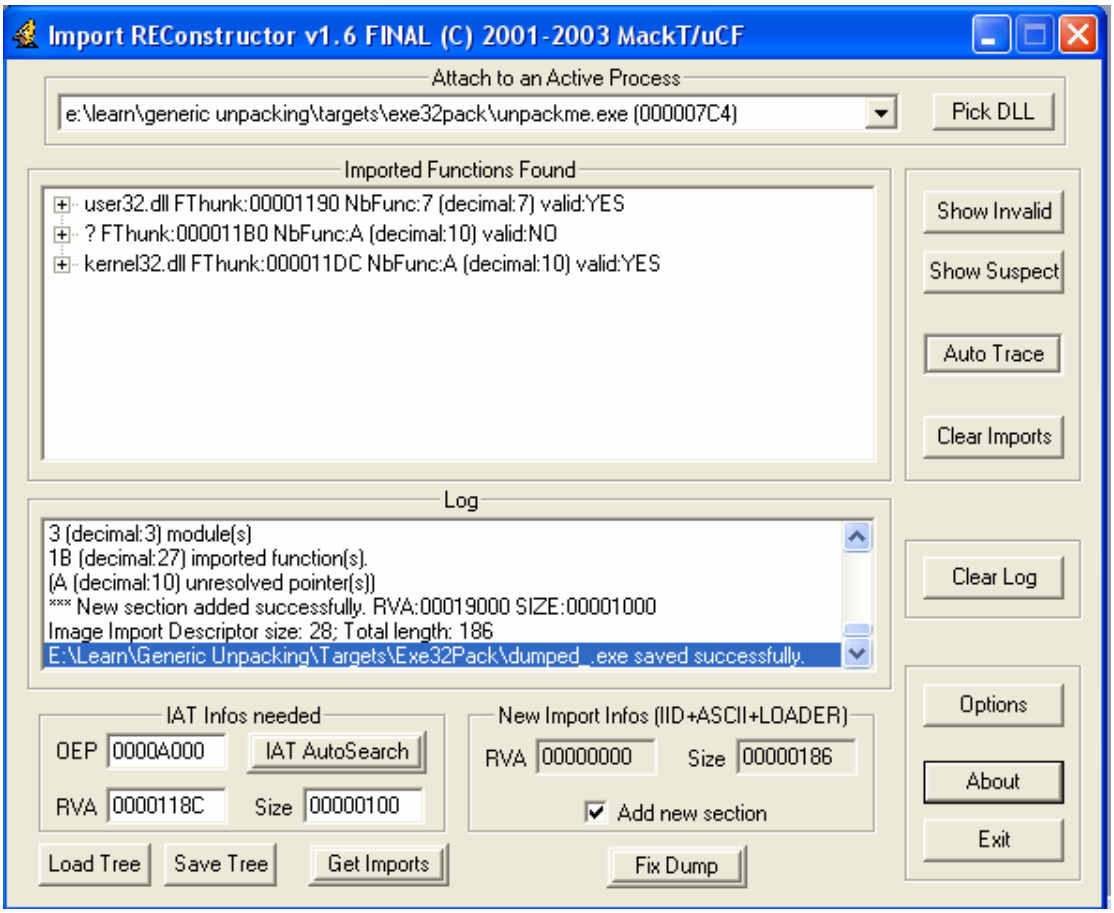
در پنجره Dump کلیک راست کرده گزینه Long و سپس گزینه Address with ASCII dump را بزنید:



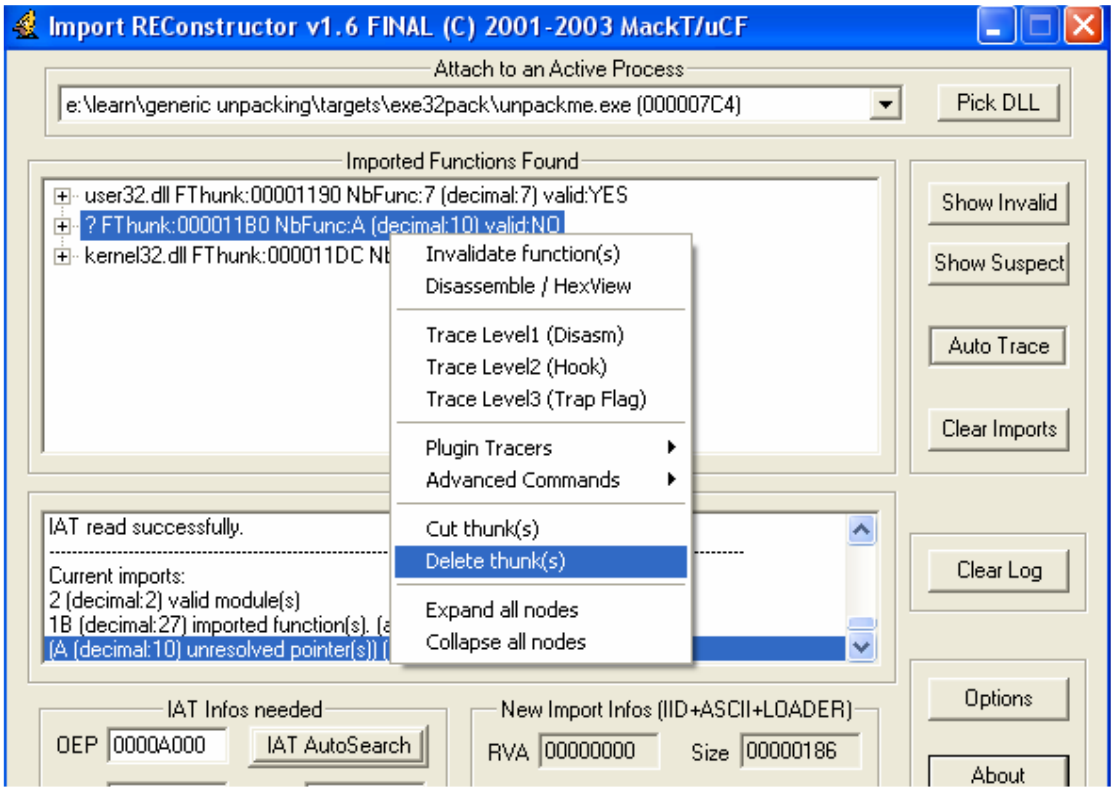
همانطور که می بینید به نظر ImportREC درست حدس زده است و IAT ما از خط 4011D8 شروع می شود.ولی اگر در پنجره Dump اندکی به بالا بروید:



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:



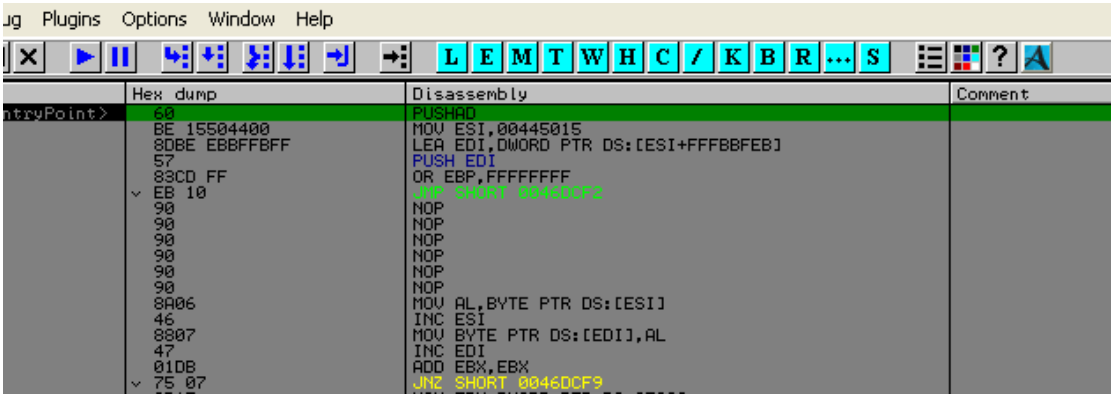
همانطور که انتظارش را داشتیم در بین توابع user32 و توابع kernel32 چند تابع Invalid وجود دارد.پس بهتر است این Thunk را کلا ازبین ببریم...هر چند وجود آن مشکلی ایجاد نمی کند و ما می توانیم در همین حالت که توابع Invalid هست،دامپ خودمان را فیکس کنیم...ولی بهتر است Invalid ها را از بین ببریم،همانند تصویر بر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



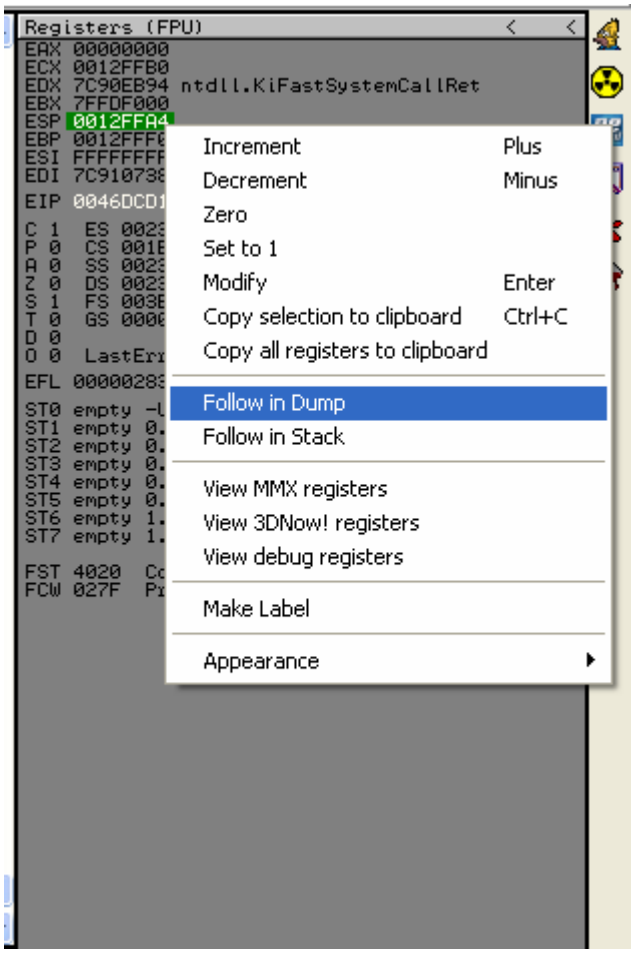
حالا دامپ خود را فیکس کنید،می بینید که فایل دامپ شده شما در هر سیستمی اجرا می شود ☺

UPX

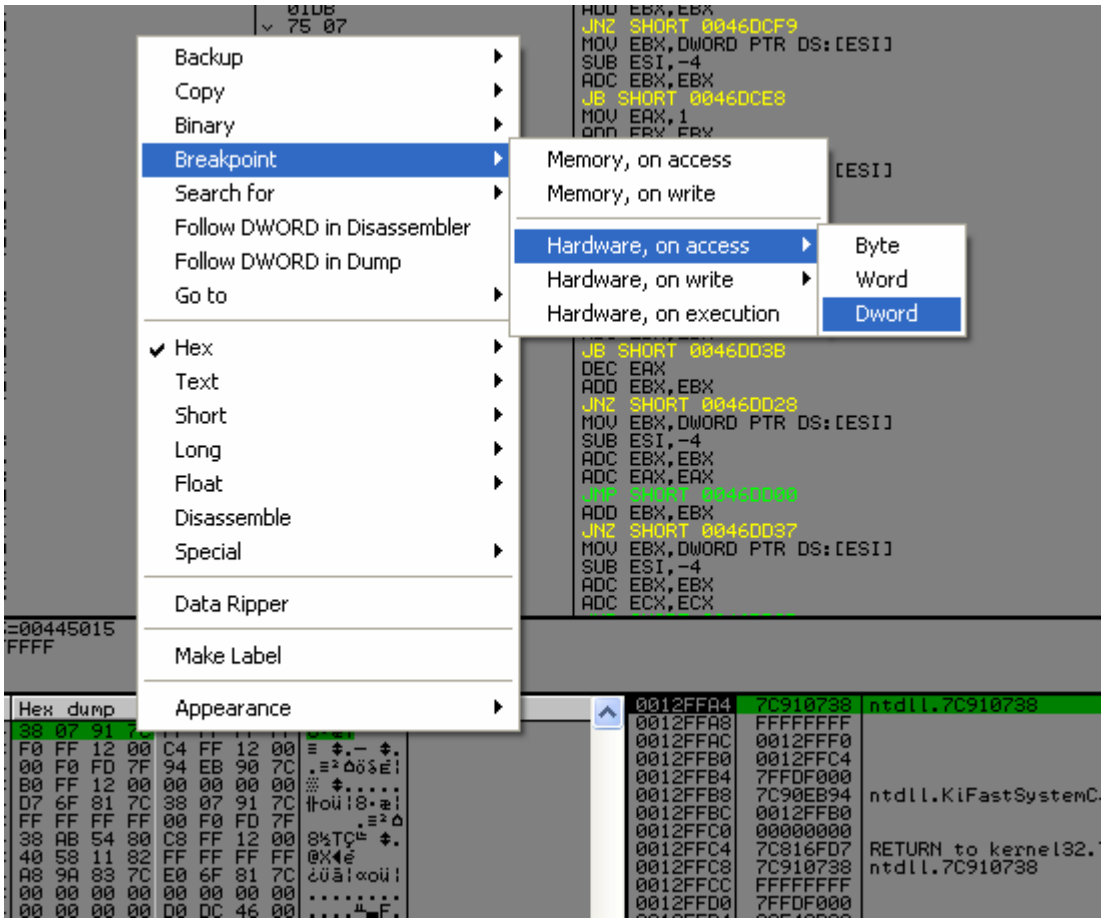
فایل را در OllyDBG باز کنید:



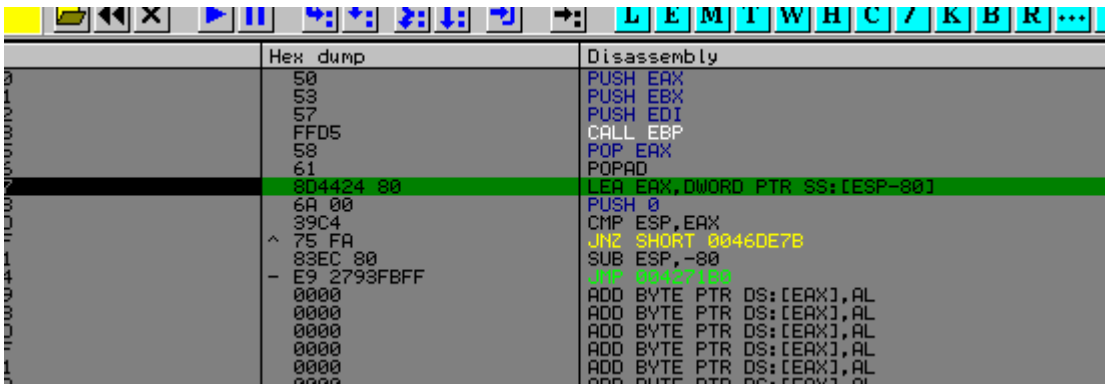
همانطور که می بینید در اولین خط از پکر ما دستور PushAD اجرا می شود.این دستور تمامی رجیسترهای ما را به غیر از EIP وارد حافظه Stack می کند.پس چون مقادیری وارد حافظه Stack می شود...مقدار ESP تغییر می کند و ما می توانیم با Hardware Breakpoint گذاشتن روی رجیستر ESP جایی که این مقادیر از حافظه خارج می شود،یعنی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



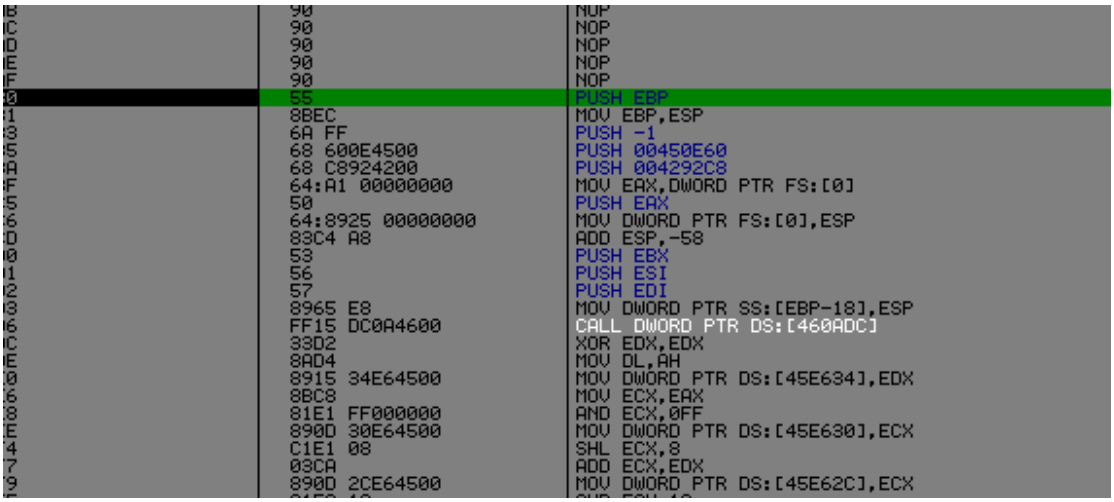
و بعد:



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



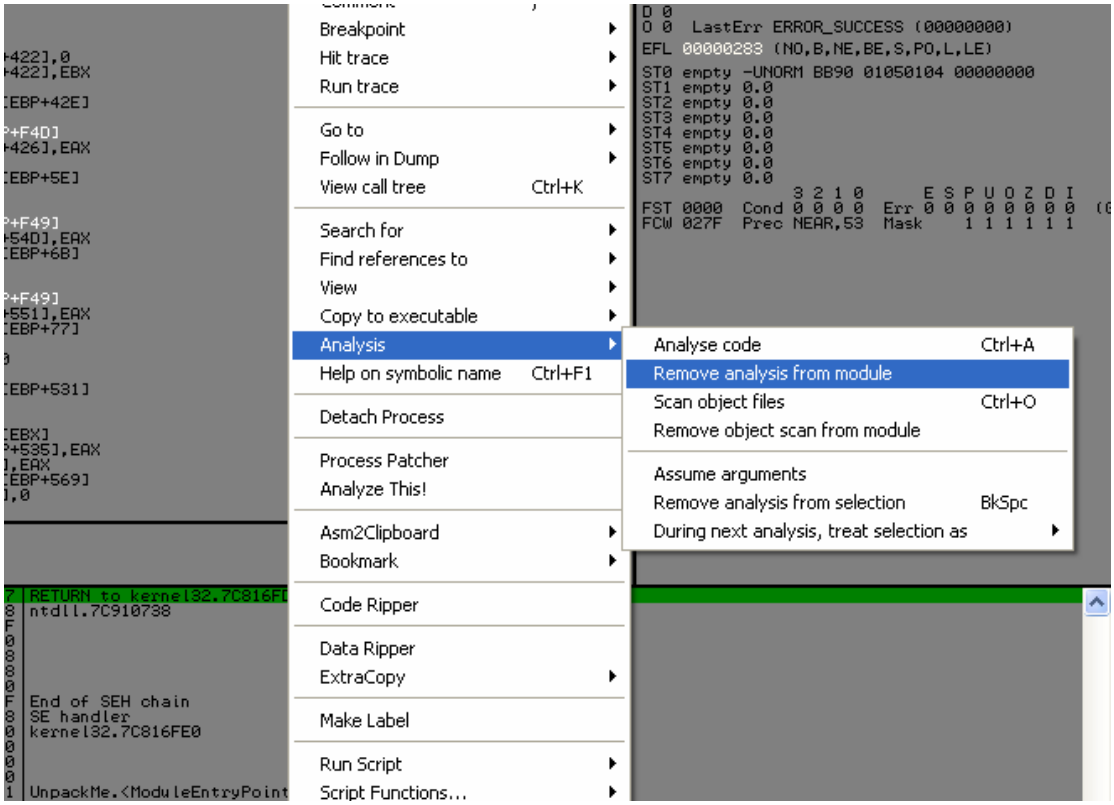
خوب به آخر کدهای پکر رسیدیم.در اینجا یک حلقه وجود دارد که این حلقه چند مقدار صفر را وارد حافظه Stack می کند و بعد با دستور JMP 004271B0 که آخرین دستور پکر ماست به OEP می ریم.
پس بر روی دستور JMP 004271B0 یک Breakpoint می گذاریم و با F9 برنامه را اجرا می کنیم و بعد کلید F7 را می زنیم تا به OEP برسیم:



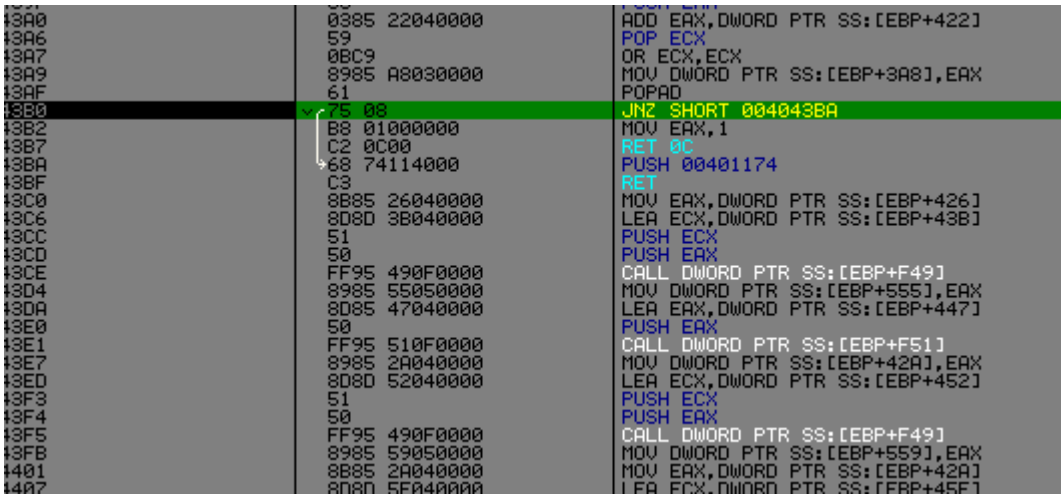
خوب،بعد از اینکه دیگر گمون نمی کنم نیازی به توضیح داشته باشد.فایل را دامپ می کنید و با ImportREC دامپ را فیکس می کنید.

AsPack

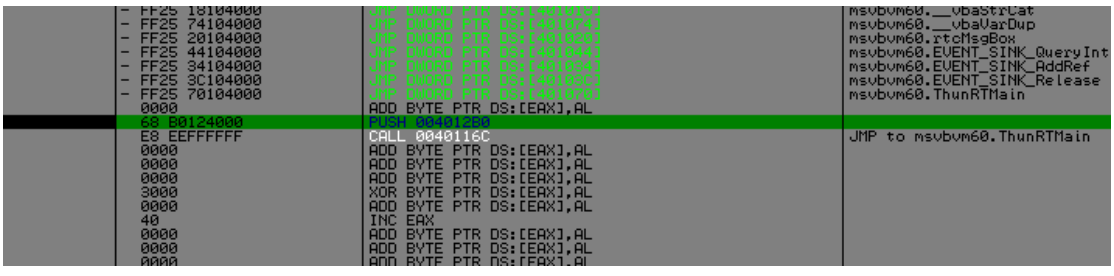
در مورد این پکر هم درست همان کاری را می کنیم که در مورد UPX کردیم.ابتدا آنالیز OllyDBG را از بین می بریم:



حالا هم اولین دستور ما PushAD می باشد.یک بار کلید F8 را بزنید تا این دستور اجرا شود و بعد بر روی مقدار رجیستر ESP یک Hardware Breakpoint می گذارید(در مثال های قبل توضیح دادم) و بعد هم برنامه را با F9 اجرا می کنید:



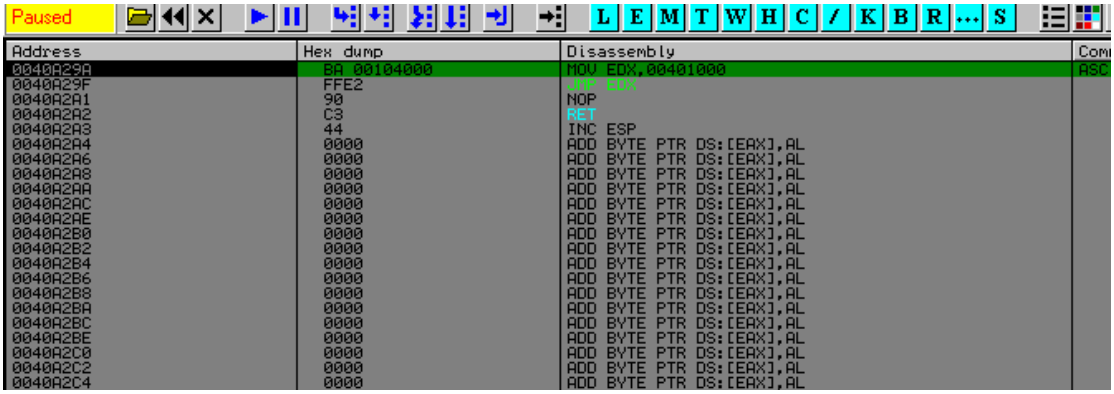
تبریک می گم،ما به آخر کدهای پکر رسیدیم.۳ بار کلید F8 را بزنید تا به OEP برسید:



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

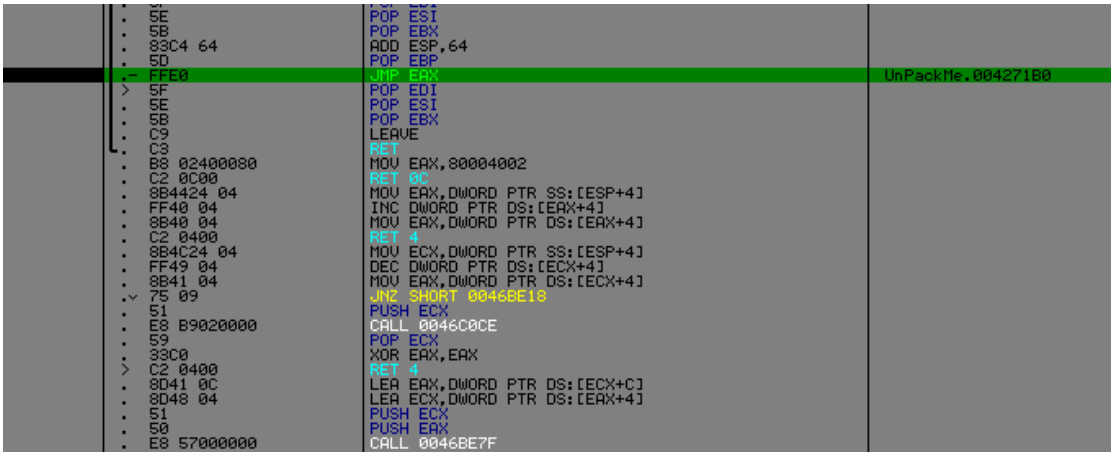
فایل مورد نظر را در OllyDBG باز می کنیم،اولین دستور PushAD است.یک بار کلید F8 را می زنیم تا این دستور اجرا شود.بر روی مقدار رجیستر ESP یک Hardware breakpoint on access می گذاریم و با F9 برنامه را اجرا می کنیم تا به اینجا برسیم:



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

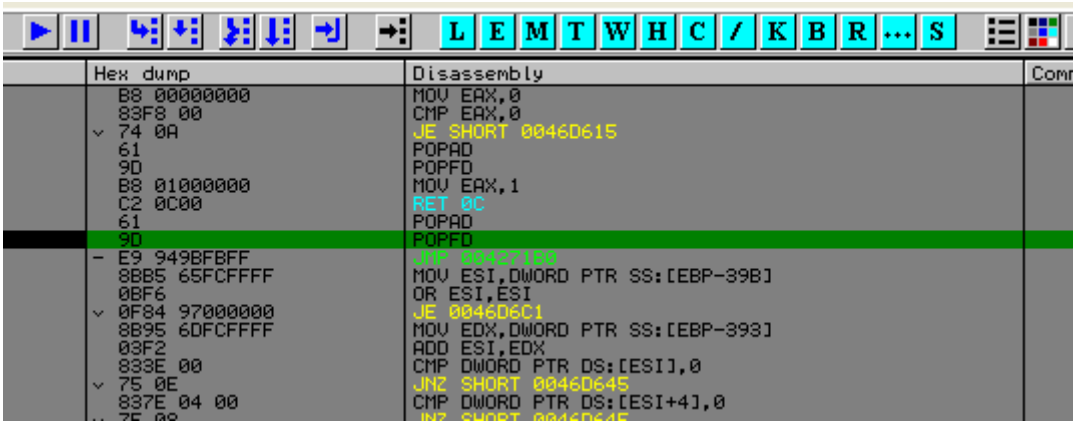
اولین دستور ما Push EBP است.یعنی مقدار EBP وارد Stack می شود.پس مقدار رجیستر ESP تغییر می کند.یک بار کلید F8 را می زنیم تا این دستور اجرا شود و بعد بر روی مقدار رجیستر ESP یک Hardware Breakpoint می گذاریم و با F9 برنامه را اجرا می کنیم،ابتدا پیغامی مربوط به رجیستر نبودن پکر داده می شود و بعد به اینجا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



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.اصولا 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:



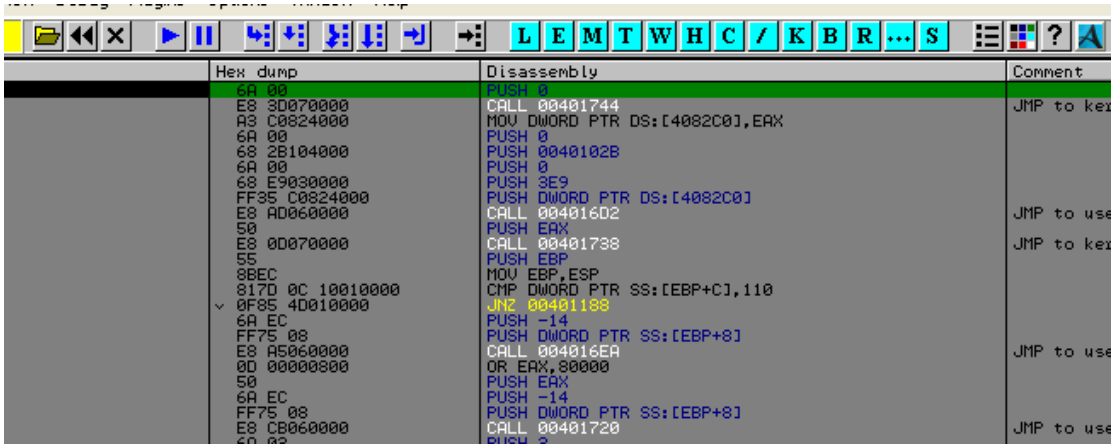
خوب،اینجا دیگر آخر کدهای پکر ماست و دستور JMP 004271B0 ما را به OEP می برد.حالا که OEP را پیدا کردیم،کافیست از فایل دامپ بگیریم و با ImportREC آن را فیکس کنیم.

NeoLite

در مورد این پکر هم خیلی راحت می شود OEP را پیدا کرد.کافیست که چند با کلید F8 را بزنیم تا به دستور JMP EAX برسیم...ممکن است قبل از رسیدن به این دستور پیغامی مبنی بر رجیستر نبودن نبود NeoLite داده شود که با زدن کلید No می توانید این پیغام 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.اصولا طبق تجربه ای که به دست آوردم...**معمولا** دستورهايي مثل:

JMP Register (JMP Eax, JMP EBX, JMP EBP....)
Call Register (Call EAX, Call EBX, Call ECX...)
Return

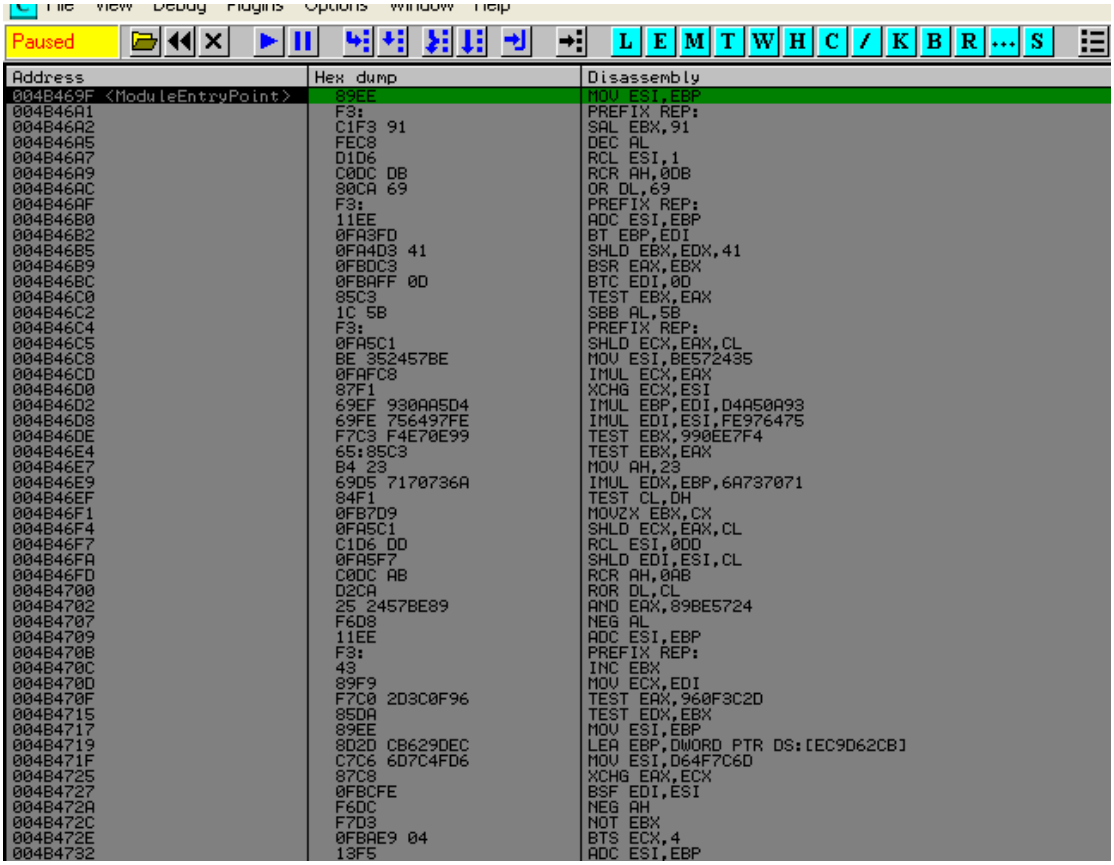
دستورهایی هستند که ممکن است ما را به OEP ببرند،در اینجا دستور JMP Eax ما را به OEP برد.
حالا یک بار کلید F8 را بزنید تا به OEP برسید.همانطور که می بینید برنامه ما در MASM نوشته شده:



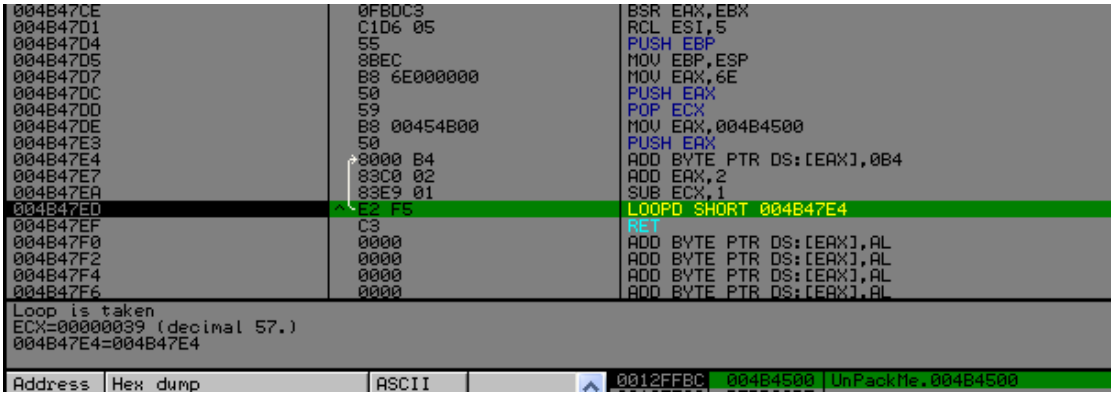
بعد از این دیگر کاری نداره.فایل را دامپ می کنید و با ImportREC فیکس می کنید.

UPolyX

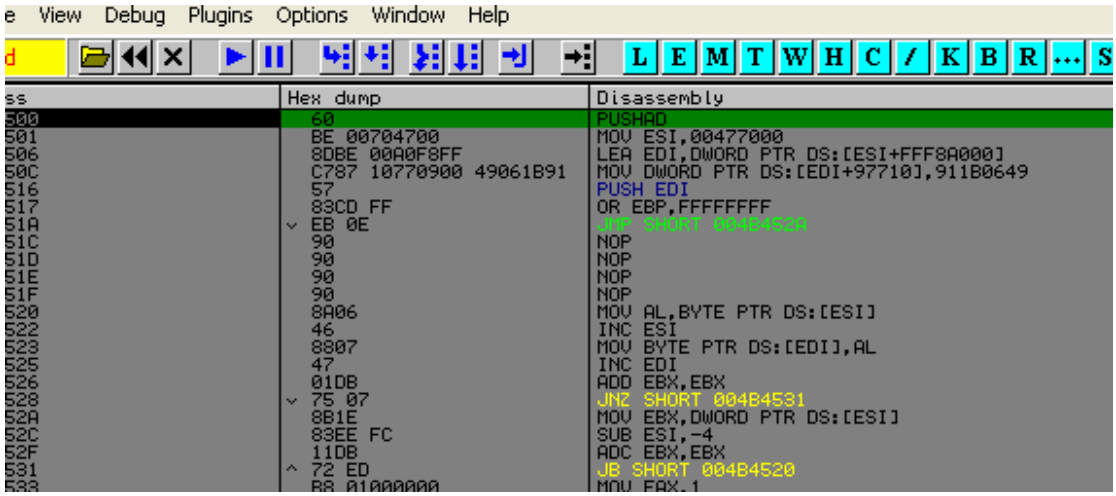
برنامه را در OllyDBG باز کنید.همانطور که می بینید در Entry point برنامه یکسری کدهای عجیب و غریب وجود دارد:



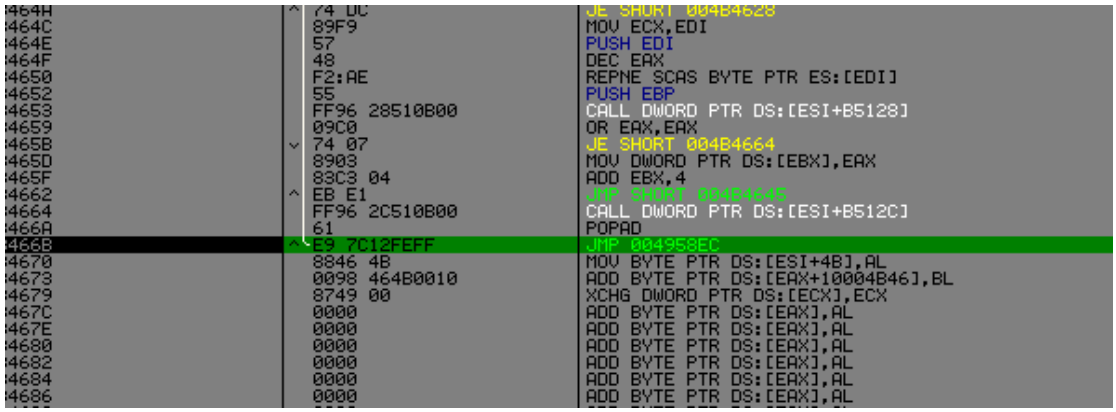
خوب برنامه را با F8 به جلو می بریم.تا به این حلقه برسیم:



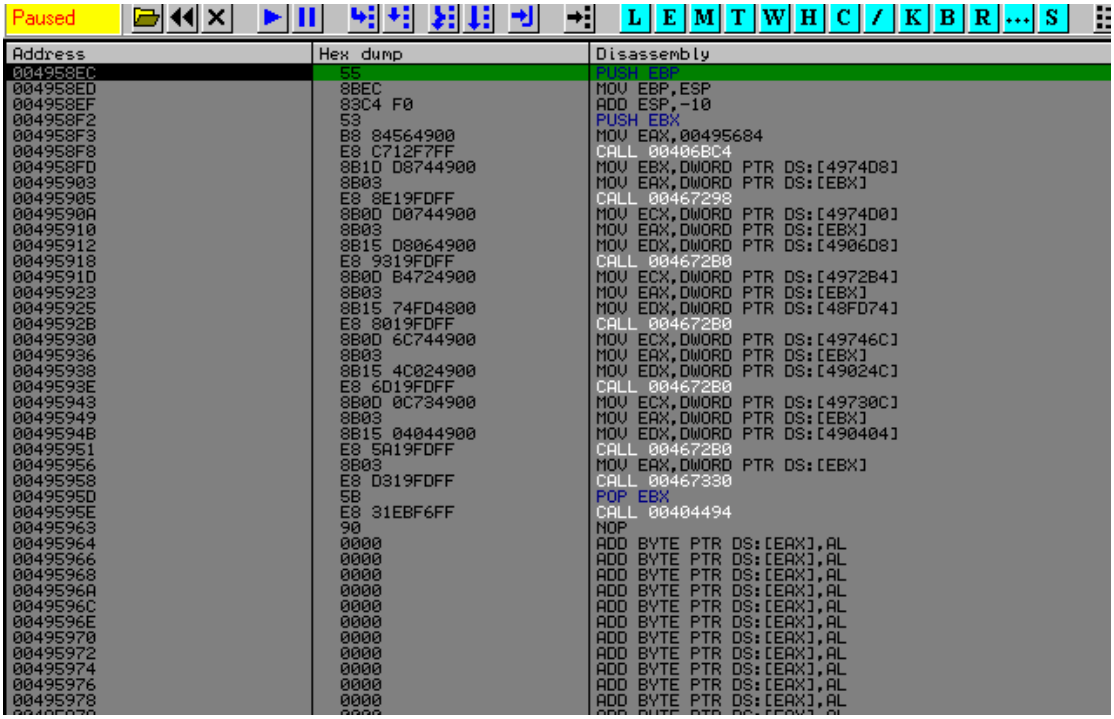
برای رد کردن این حلقه روی RET پایینی Breakpoint گذاشته و با F9 برنامه را اجرا کنید.حالا یک بار F8 را بزنید تا به اینجا برسید:



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:



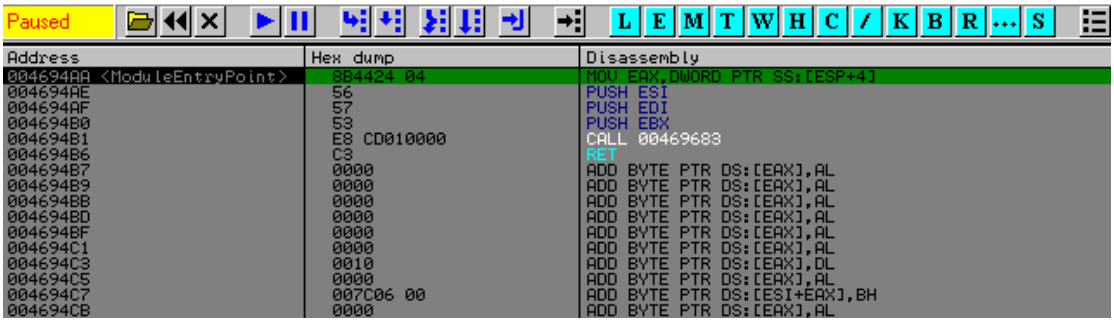
این پرش به OEP می رود.کافیست یک بار کلید F8 را بزنید تا به OEP برسید ☺



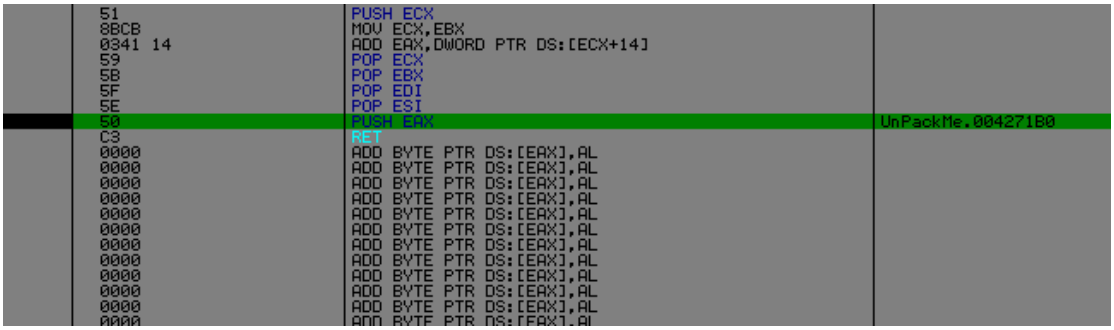
خوب،حالا به OEP رسیدیم،از کجا فهمیدم؟...به این کدها با دقت نگاه کنید.به نظر شما شبیه کدهایی که در Entry Point برنامه های دلفی قرار دارد نیست؟...(Call های فراوان...همچنین در درون اولین Call تابع GetModuleHandleA قرار دارد.)
پس اینجا OEP ماست و برنامه ما در دلفی نوشته شده است.بعد از این دیگر مشکلی نیست.فقط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

ASDPack

برنامه را در OllyDBG باز کنید:



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 کافیست 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.



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:

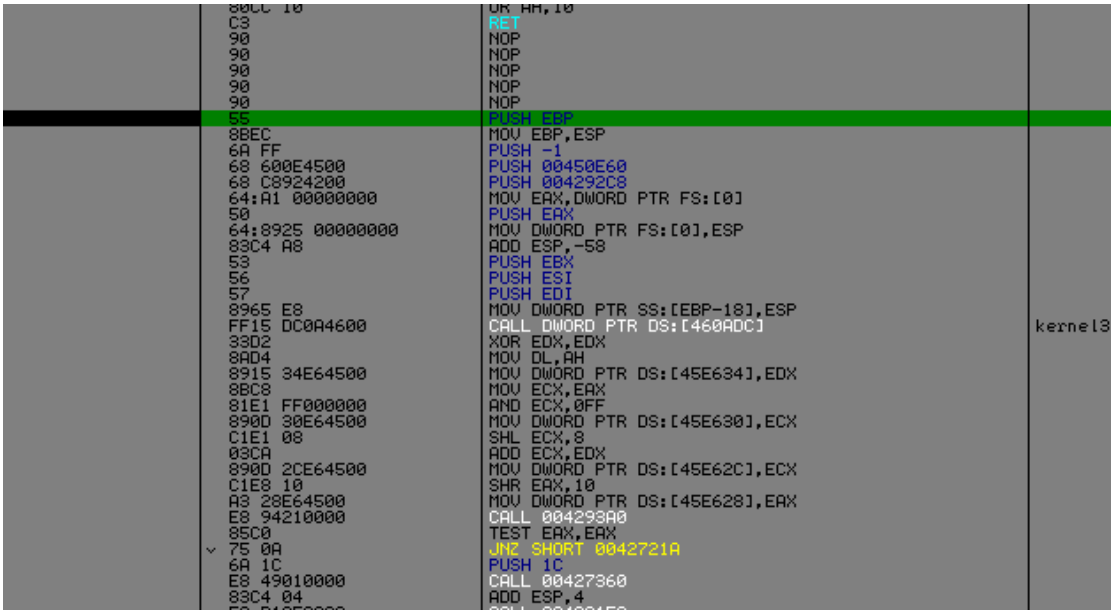
اگر قبل از دستور RET یک Push بیاید.دستور RET ما را به آدرسی می برد که Push شده است.

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

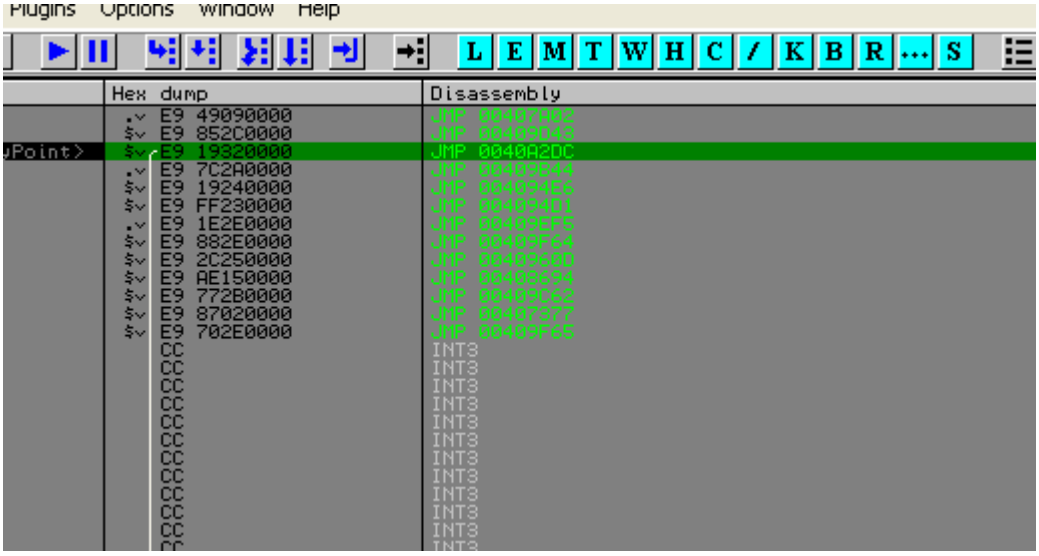
کافیست دو بار کلید F8 را بزنید.تا به OEP برسیم:



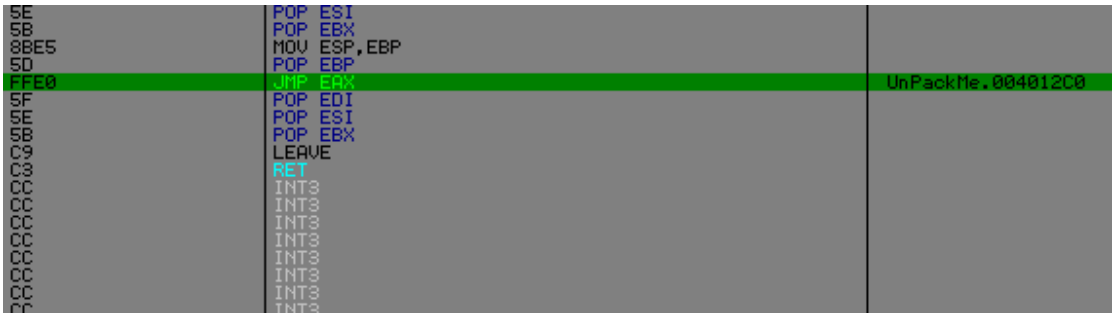
خوب...دیگر از این به بعد مشکلی نیست.کافی است فایل را دامپ کنید(توصیه من این است با LordPE این کار را بکنید.) و با ImportREC آن را فیکس کنید.

EZIP

برنامه را در OllyDBG باز کنید:



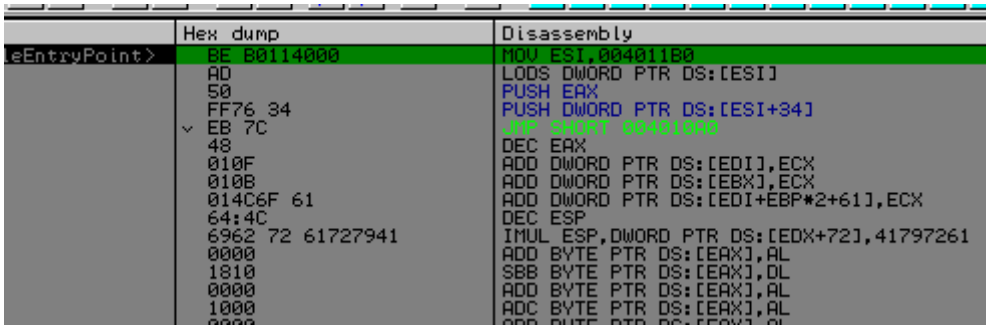
یک بار کلید F8 را بزنید تا به دستور Push EBP برسید.یک بار دیگر کلید F8 را بزنید تا رجیستر ESP تغییر کند و بعد روی مقدار رجیستر ESP یک Hardware breakpoint on access بگذارید و بعد کلید F9 را بزنید تا به این خط برسید:



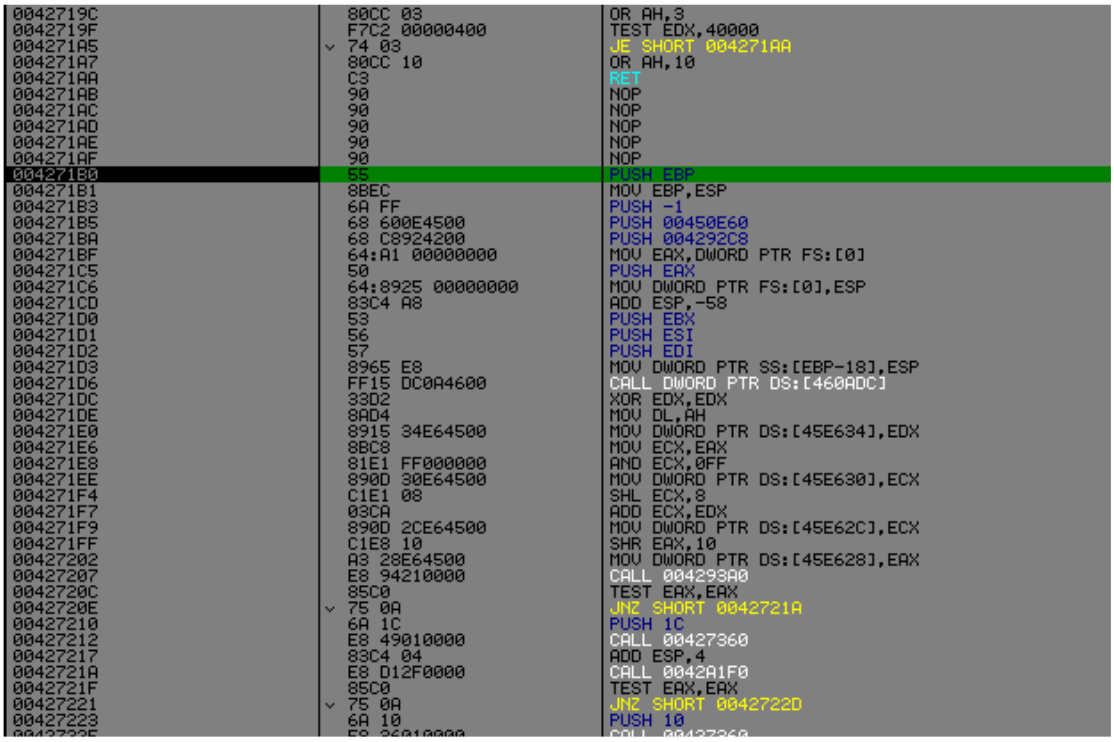
این پرش ما را درست به OEP می برد ☺
بعد از آن هم مشکلی وجود نخواهد داشت.فقط فایل را دامپ می کنید و با ImportREC فیکس می کنید.

WinUPack

برنامه را در OllyDBG باز کنید:



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 on access روی مقدار رجیستر ESP بگذارید و کلید F9 را بزنید:



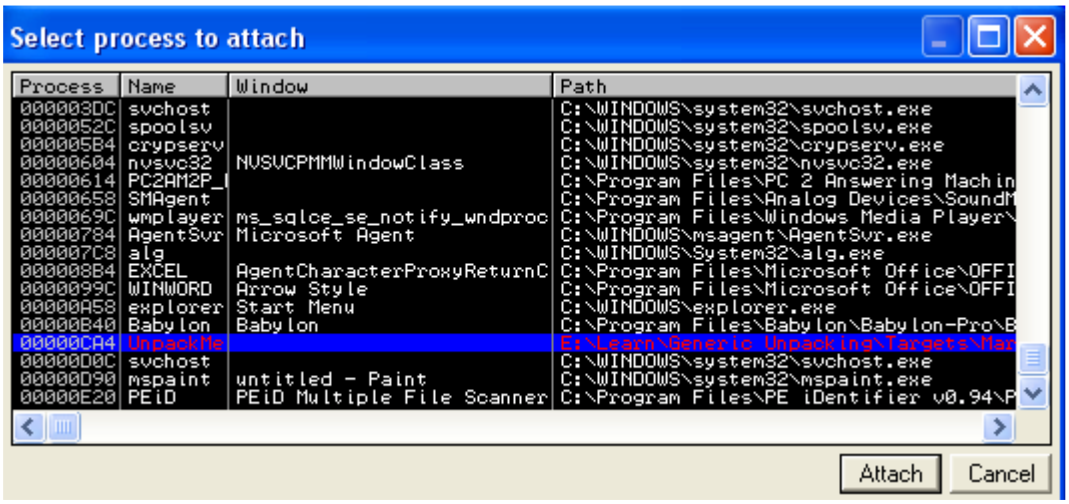
همانطور که می بینید این بار مستقیم به OEP رسیدیم. ☺
بعد از این هم مشکلی نیست.فایل را دامپ می کنید و بعد فیکس می کنید.

Mario Packer

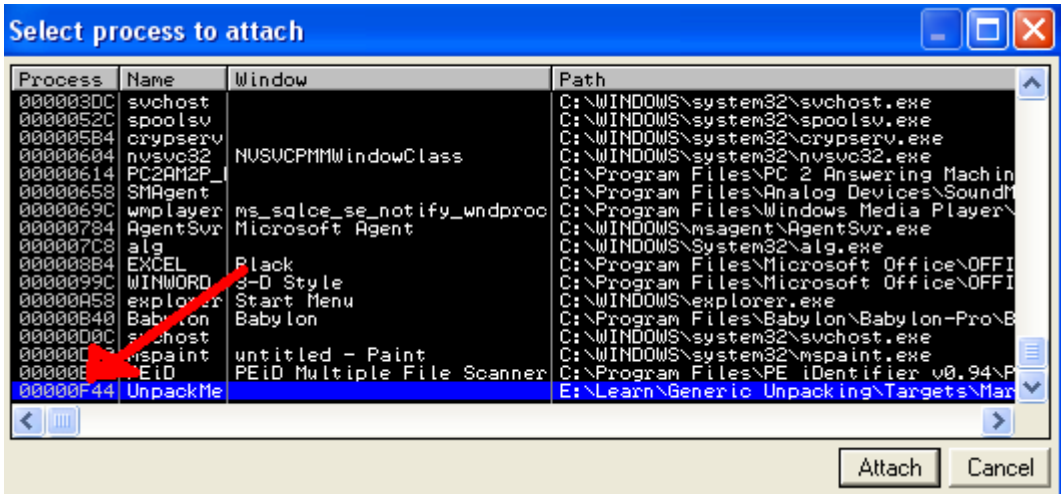
برنامه را در OllyDBG باز کنید،کلید F9 را بزنید تا برنامه به طور کامل اجرا شود:

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:



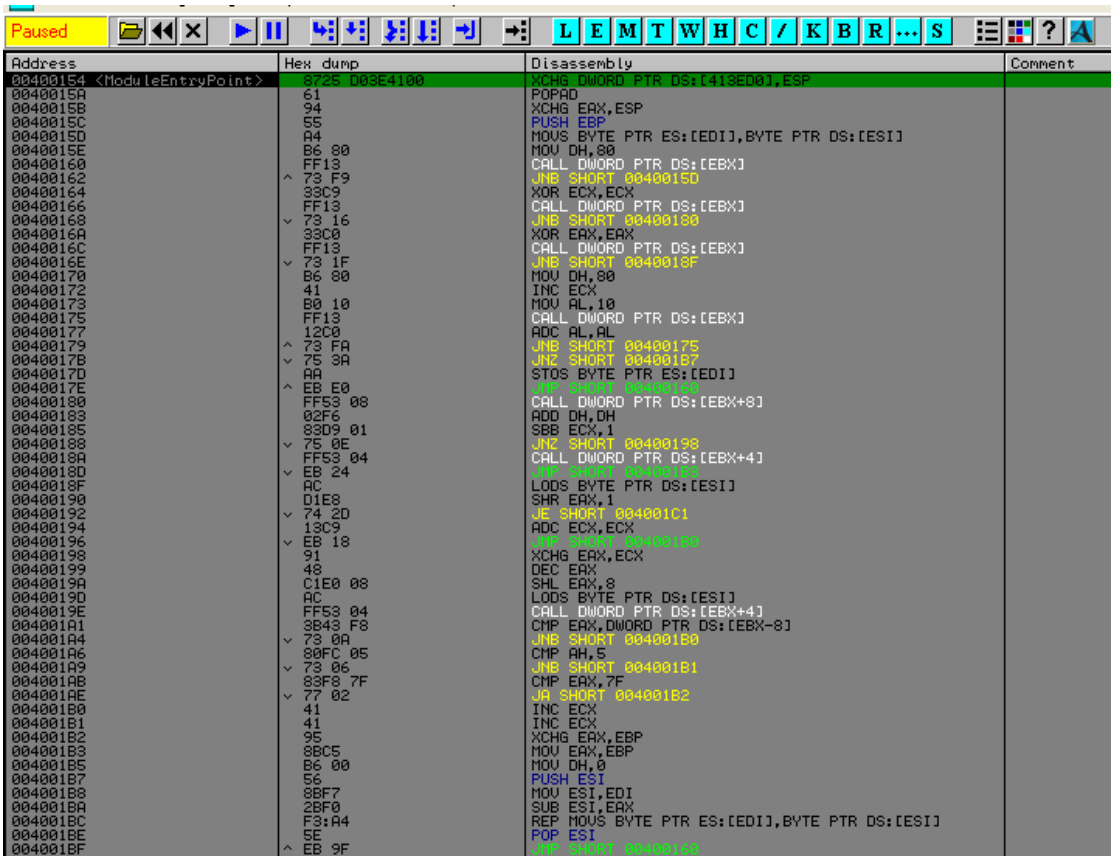
همانطور که در تصویر بالا می بینید،PID (Process ID) پروسه ما برابر CA4 می باشد.(در کامپیوتر من اینگونه است،در کامپیوتر شما PID فرق خواهد داشت)خوب،این صفحه را ببندید و دوباره برنامه را با F9 اجرا کنید.حالا همینطور که برنامه اجرا می باشد،دوباره در منوی File گزینه Attach را بزنید:



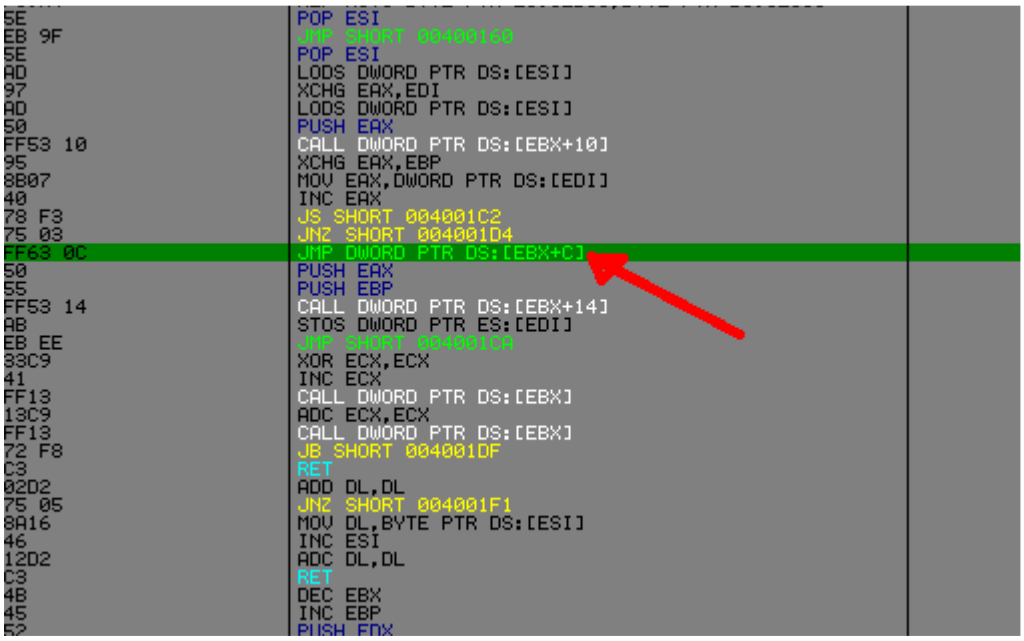
می بینید پروسه فایل ما حالا هست : F44 !
این یعنی چی؟...این یعنی برنامه ما بعد از این که اجرا می شود،فایل اصلی ما را با یک پروسه جدید به اجرا در می آورد...یعنی اینکه Mario Packer یک پکر نیست...بلکه یک Self-Extractor هست...اینجاست که باید فرق بین یک پکر با یک Self-Extractor را تشخیص بدین:
یک پکر کدهای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

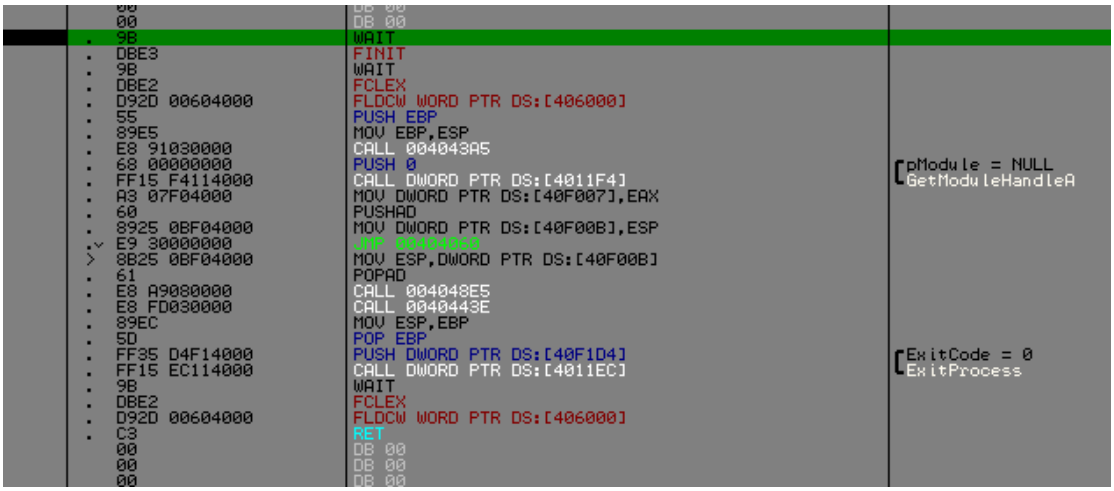
گاهی اوقات برای یافتن OEP بهتر است روش های خاصی را بلد بود.برای مثال در مورد پکر FSG چند خط زیر Entry point یک دستور مثل JMP DWORD PTR DS:[EBX+C] وجود دارد که درست می رود OEP...یادگیری این روش ها در مورد پکرهای معروف خیلی به ما کمک می کند ☺
خوب،برنامه را در OllyDBG باز کنید:



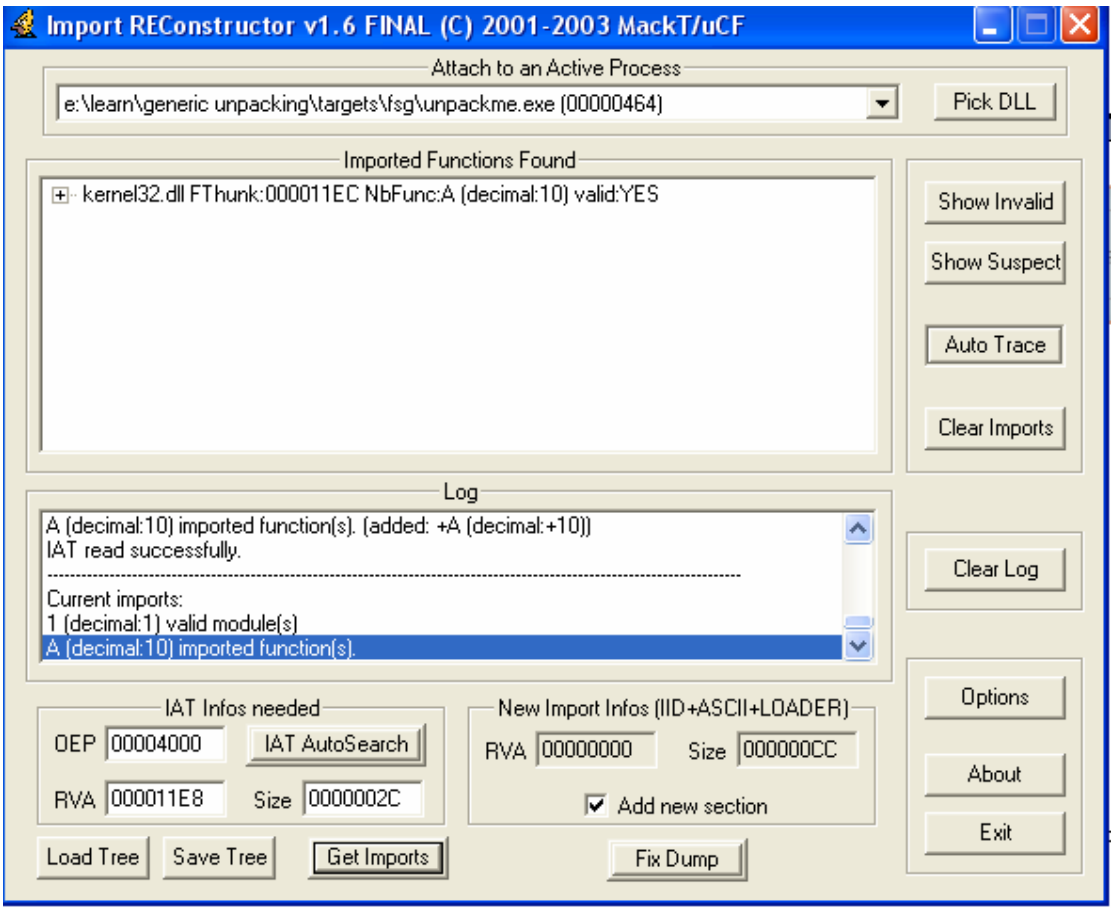
خوب،حالا با موس به سمت پایین بروید تا یک دستور شبیه JMP DWORD PTR DS:[EBX+C] پیدا کنید:



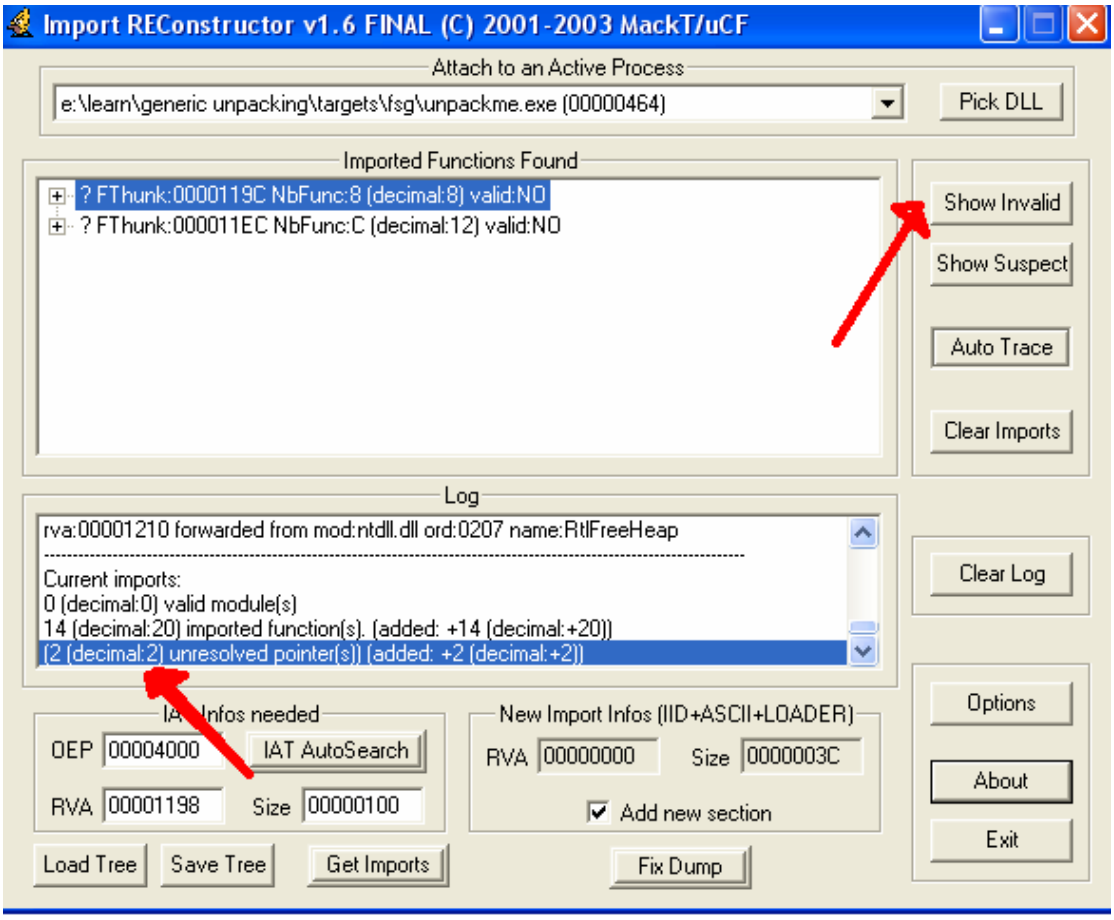
این دستور درست به OEP می رود.کافیست بر روی آن Breakpoint گذاشته و کلید F9 را بزنید تا به این خط برسید و بعد یک F8 به OEP نظر می رود:



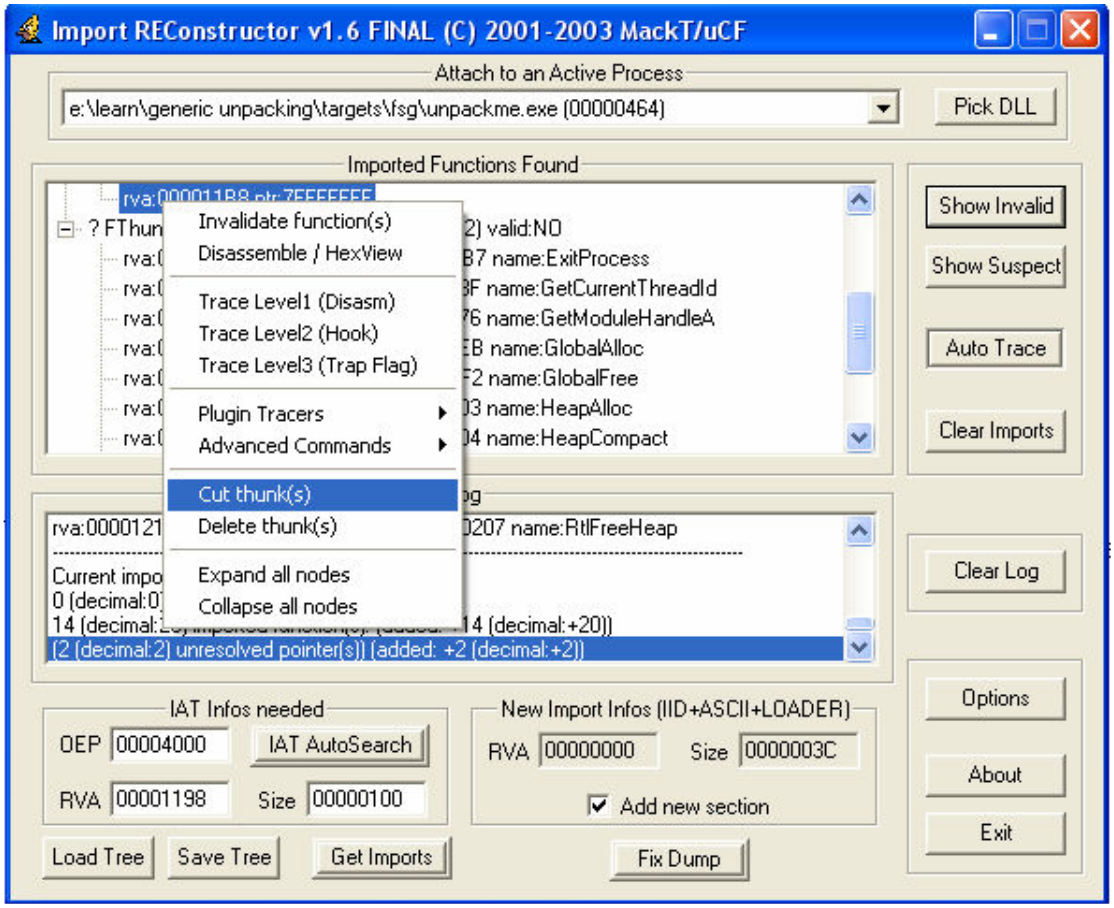
حالا از فایل دامپ بگیرید و بعد ImportREC را باز کنید،OEP را به برنامه بدهید(4000)...گزینه Auto-search IAT را بزنید.و بعد گزینه Get Imports را بزنید:



باز هم OllyDBG 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:



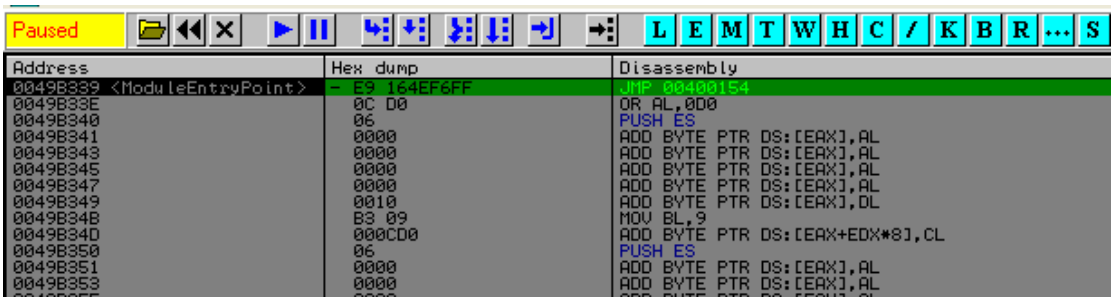
همانطور که می بینید دو تابع Invalid داریم،برای حذف این دو گزینه هم گزینه Show Invalid را بزنید و بعد همانند تصویر کلیک راست کرده و گزینه Cut Thunk را بزنید تا تمامی توابع Valid شود:



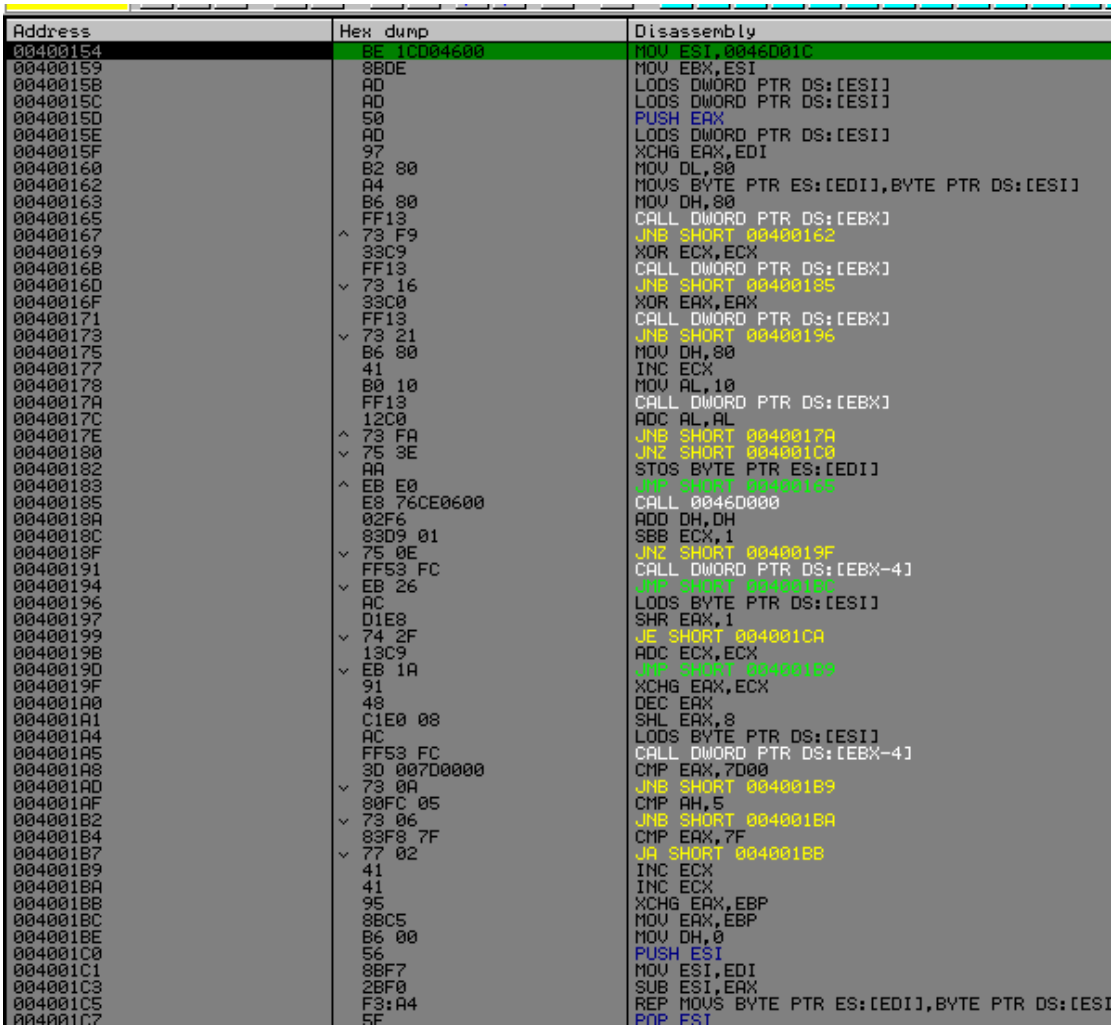
حالا هم فایل دامپ خودتان را فیکس کنید،می بینید که فایل آنپک شده اجرا می شود.

MEW

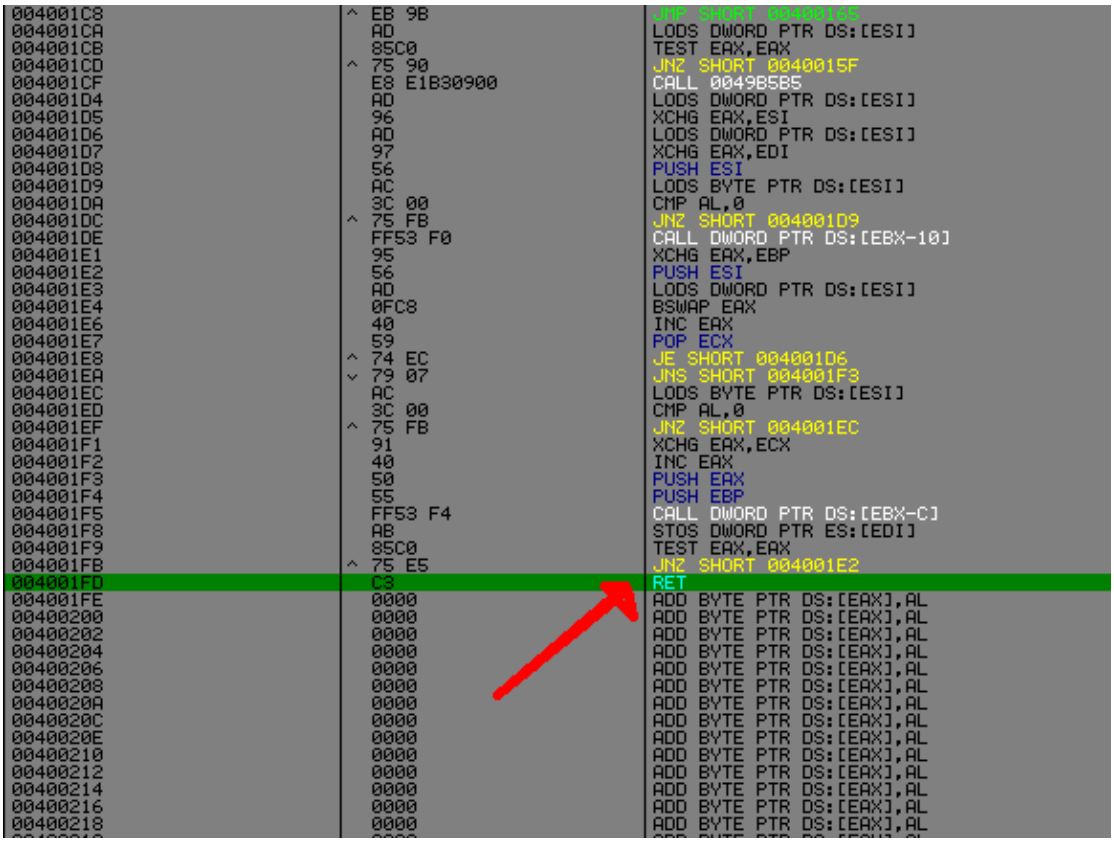
اصولا در یافتن OEP بهتر است همیشه در آخر کدها،یک Breakpoint (ترجیحا Hardware Breakpoint) بگذاریم تا بتوانیم OEP را پیدا کنیم.چرا که در معمولا در آخر کدهای یک پکر،اختیار از دست پکر خارج می شود و به دست فایل اصلی ما می افتد...در مورد MEW هم ما همین کار را می کنیم،برنامه را در OllyDBG باز کنید:



یکبار F8 بزنید:



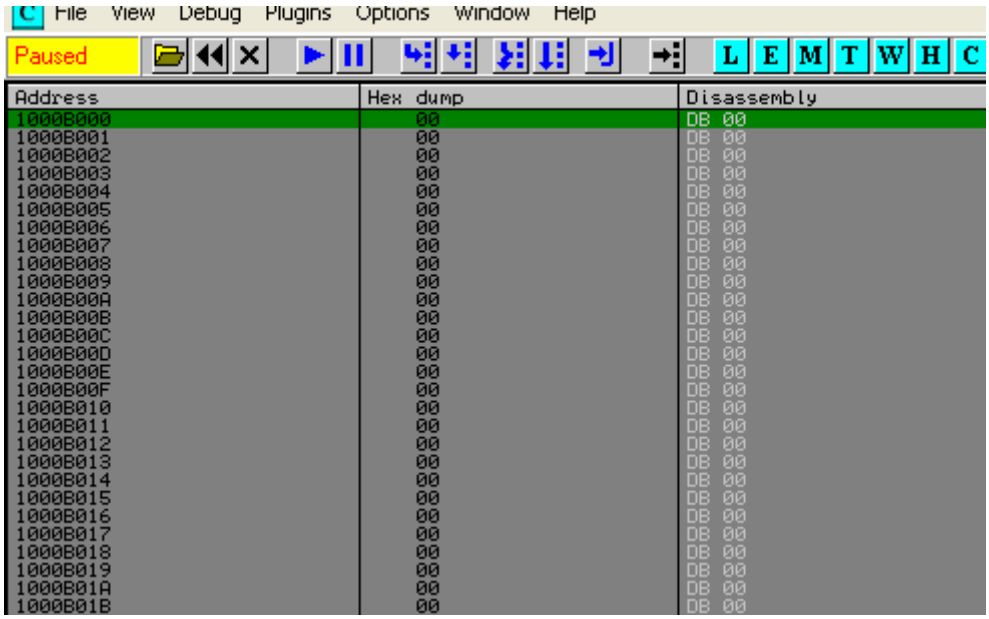
این کدهای اصلی پکر ماست.بهتر است انتهای کد را پیدا کنیم.برای این کار با Mouse به پایین بروید(Scroll down) تا به اینجا برسید:



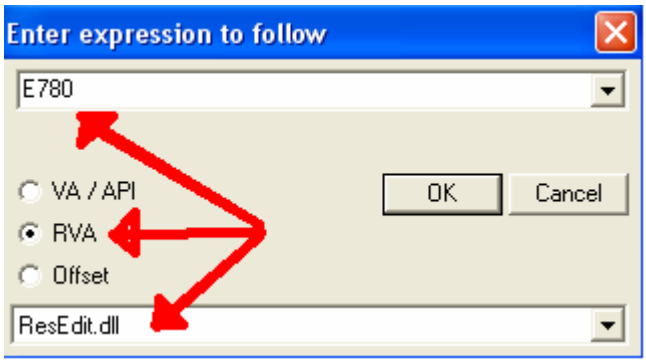
همانطور که می بینید بعد از این دستور Return دیگر دستور خاصی وجود ندارد،پس این RET ممکن است ما را به OEP ببرد.
پس بر روی آن یک Breakpoint بگذارید و برنامه را با F9 اجرا کنید،برنامه در جایی که Breakpoint گذاشتیم متوقف شد.حالا یک بار کلید F8 را بزنید تا به OEP برسیم.
بعد از یافتن OEP دیگر مشکل خاصی وجود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

UPX DLL

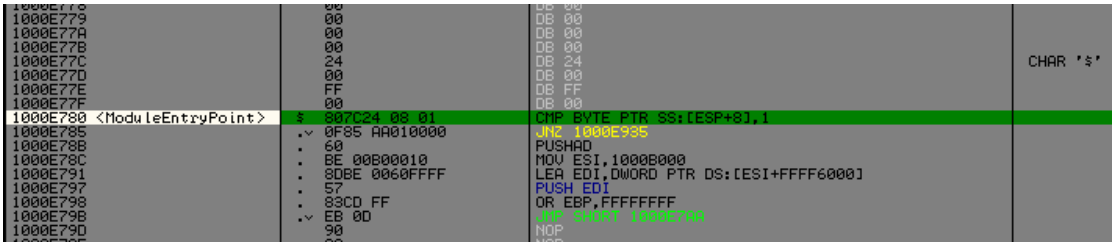
اصولا آنپک کردن یک فایل DLL **زیاد** فرقی با آنپک کردن یک Exe ندارد.اول از همه بهتر است ببینیم که ImageBase فایل ما چند است؟برای اینکار هم از PEiD یا نرم افزارهای مشابه استفاده می کنیم.(قبلا توضیح دادم).در این مثال(فایل ResEdit.dll) ImageBase ما برابر 10000000 است.OllyDBG برای اجرای یک فایل DLL از فایل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



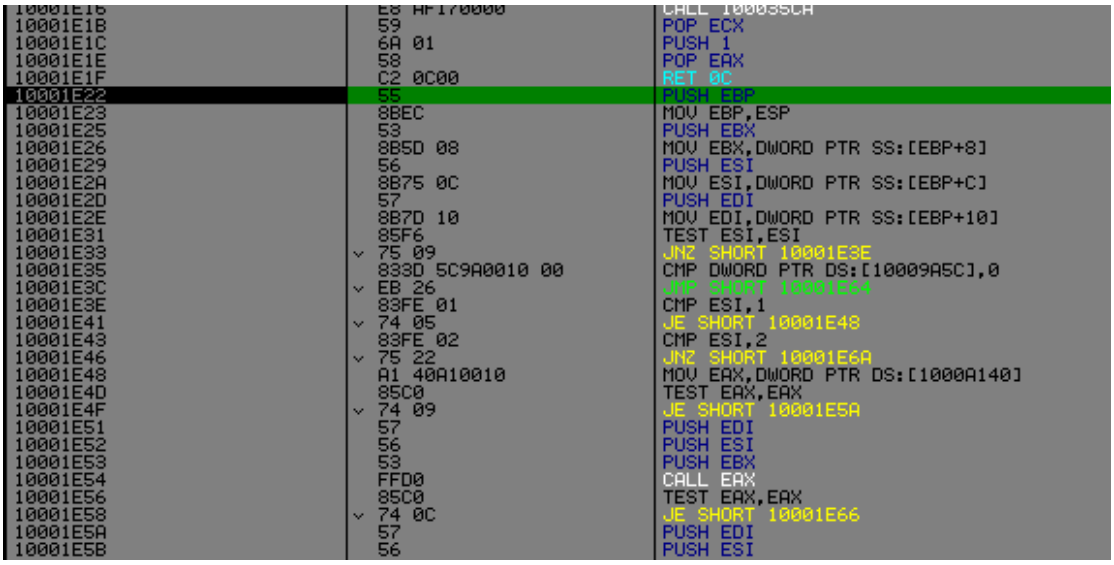
ابتدا به Entry Point فایل خود بروید.از آنجا که می دانیم Entry Point فایل ما بر حسب RVA برابر E780 است.(این اطلاعات را از PEiD گرفتم)
کلید CTRL + G را بزنید و همانند تصویر تیک گزینه RVA را بزنید.،فایل مورد نظر خود را انتخاب کنید(Resedit.dll) و بعد Entry point فایل را انتخاب کنید:



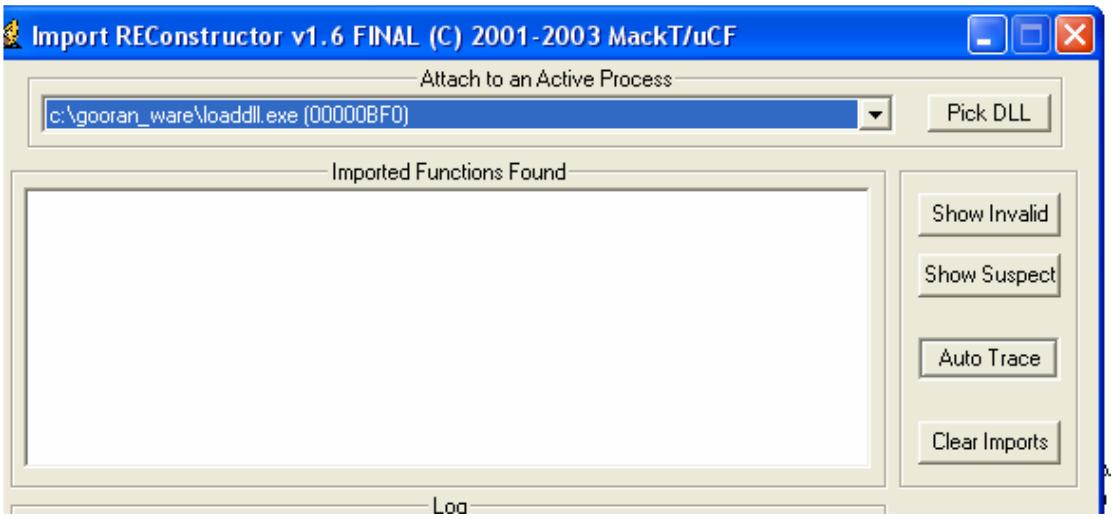
حالا بر روی Entry Point فایل Breakpoint بگذارید و کلید F9 را بزنید:



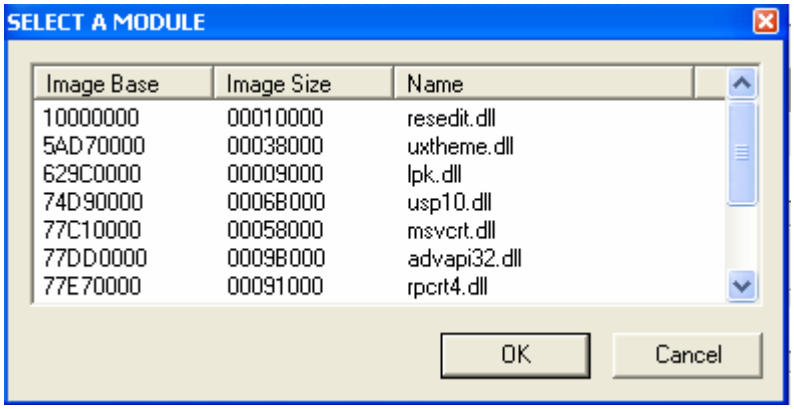
سه بار کلید F8 را بزنید تا به OEP برسید:



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 Loaddll.exe را در ImportREC را باز کنید:



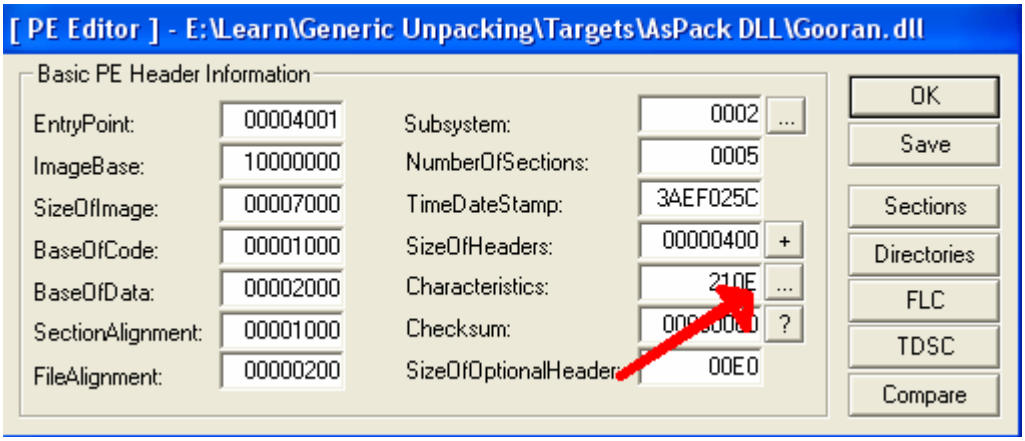
حالا گزینه Pick DLL را بزنید:



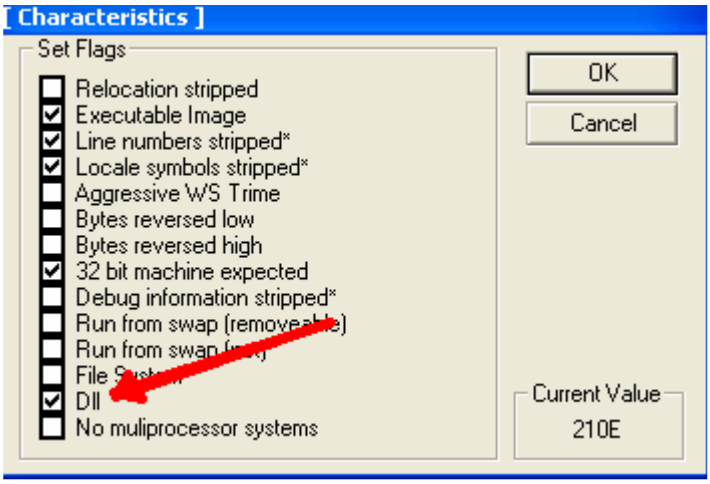
حالا هم فایل resedit.dll را انتخاب کرده و کلید OK را بزنید.از حالا دیگر مشکلی نیست.OEP را بر حسب RVA می دهید(1E22) و بعد فایل دامپ شده خودتان را فیکس کنید.

AsPack DLL

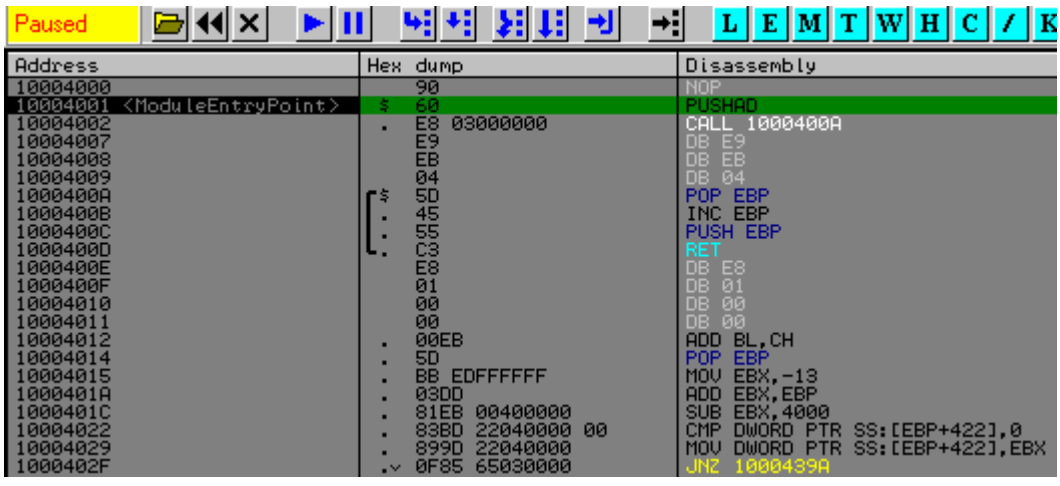
اول از همه در آنپک کردن یک فایل DLL خصوصا در مواردی که ImageBase آنها برابر 004000000 است.بهتر است مشخصات فایل را همانند یک فایل Exe کنیم.به این ترتیب OllyDBG خیال می کند که فایل Exe است و به طور مستقیم،بدون استفاده از Loaddll.exe فایل را اجرا می کند.فایل مورد نظر(Gooran.dll) را در LordPE باز کنید:



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



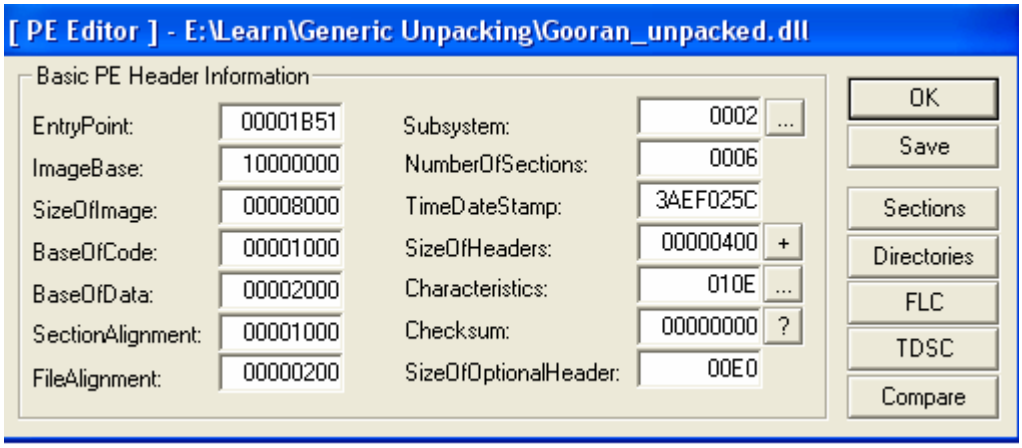
همانطور که می بینید گزینه DLL تیک دارد،ما با برداشتن تیک این گزینه کاری می کنیم که OllyDBG خیال کند این فایل DLL نیست ☺
تیک این گزینه را بردارید و گزینه OK را بزنید و بعد گزینه Save و بعد OK را بزنید.
حال فایل همانند یک فایل Exe در OllyDBG اجرا می شود.(بهتر است نام فایل را عوض کنید.بگذارید: Gooran.exe)
فایل را در OllyDBG باز کنید:



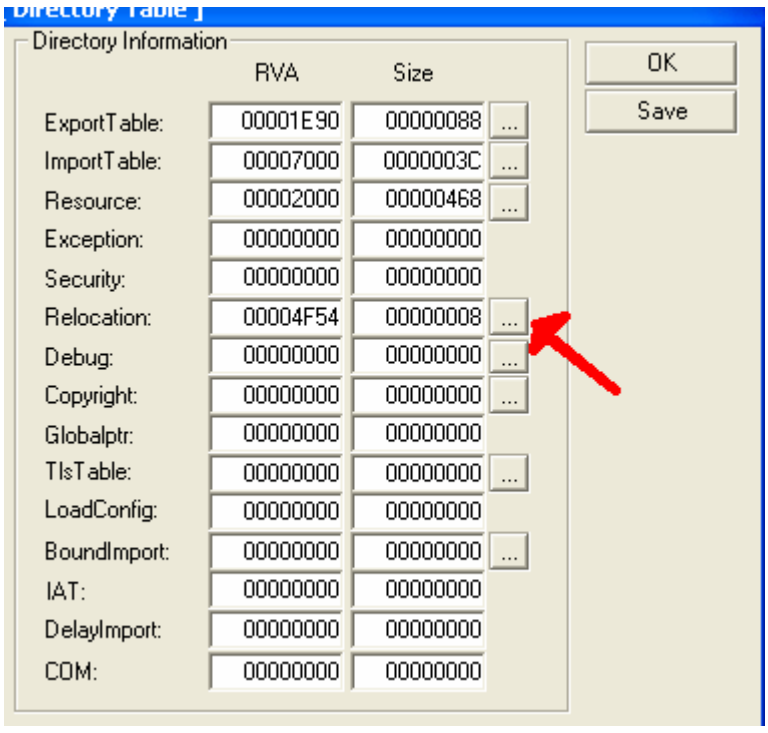
اولین دستور PushAD است.پس به راحتی می شود از طریق رجیستر ESP، OEP را به دست آورد.OEP ما اینجاست:

```
10001B4A   .  5E              POP ESI
10001B4B   >  6A 01           PUSH 1
10001B4D   .  58              POP EAX
10001B4E   L> C2 0C00         RET 0C
10001B51   r. 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   .v 75 09           JNZ SHORT 10001B6D
10001B64   .  833D F41B0010 00 CMP DWORD PTR DS:[10001BF4],0
10001B6B   .v EB 26           JMP SHORT 10001B93
10001B6D   >  83FE 01         CMP ESI,1
10001B70   .v 74 05           JE SHORT 10001B77
10001B72   .  83FE 02         CMP ESI,2
10001B75   .v 75 22           JNZ SHORT 10001B99
10001B77   >  A1 C41C0010     MOV EAX,DWORD PTR DS:[10001CC4]
10001B7C   .  85C0            TEST EAX,EAX
10001B7E   .v 74 09           JE SHORT 10001B89
10001B80   .  57              PUSH EDI
10001B81   .  56              PUSH ESI
10001B82   .  53              PUSH EBX
```

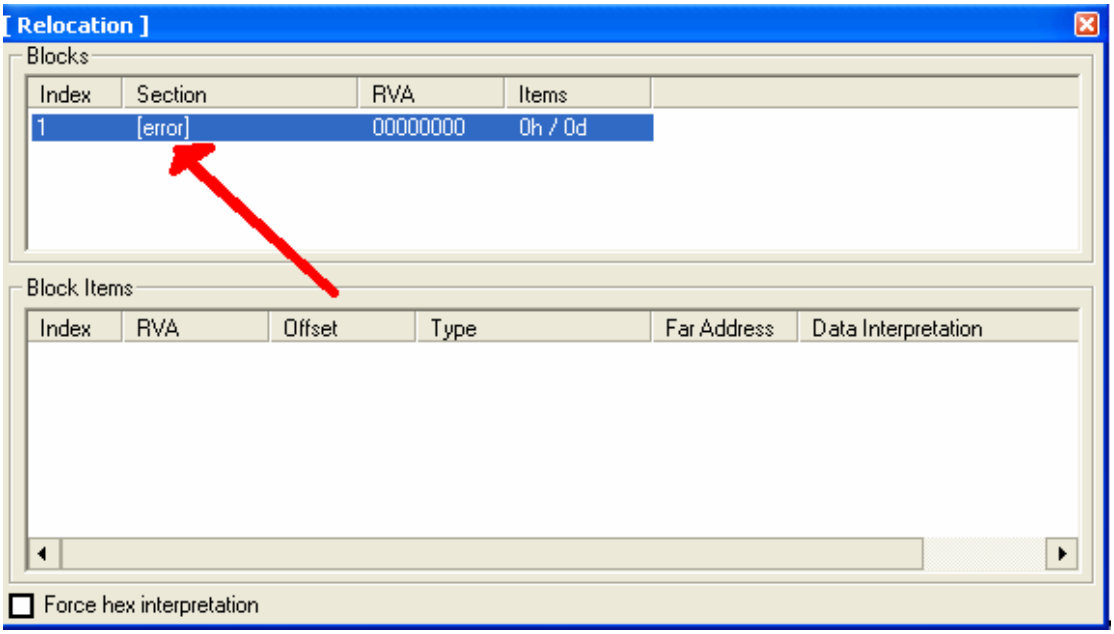
حالا از فایل دامپ می گیریم و بعد با ImportREC دامپ خودمان را فیکس می کنیم،حالا هم اسم فایل آنپک شده را عوض کنید،بذارید:
Gooran_Unpacked.dll
در ضمن،تیک آن گزینه DLL که قبلا از مشخصات فایل برداشته بودید را دوباره در فایل آنپک شده تیک بزنید تا مشکلی برای فایل پیش نیاید.
فایل های DLL یک سکشن به نام reloc. دارند که در آن Relocation های فایل DLL را نگه می دارند.فایل آنپک شده را در LordPE اجرا کنید:



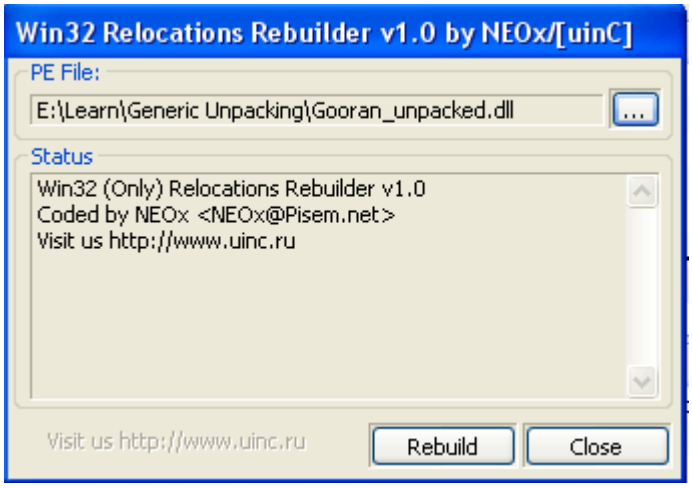
گزینه Directories را بزنید.



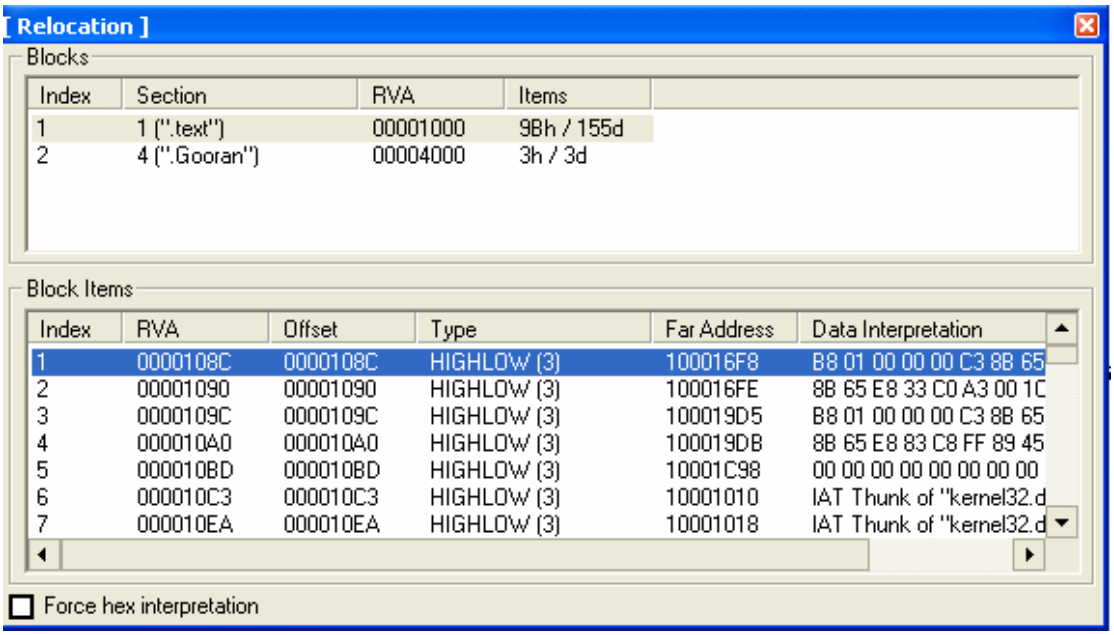
دکمه نشان داده شده در تصویر بالا را بزنید تا Relocation های فایل را ببینید:



همانطور که می بینید Relocation های فایل ازبین 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:

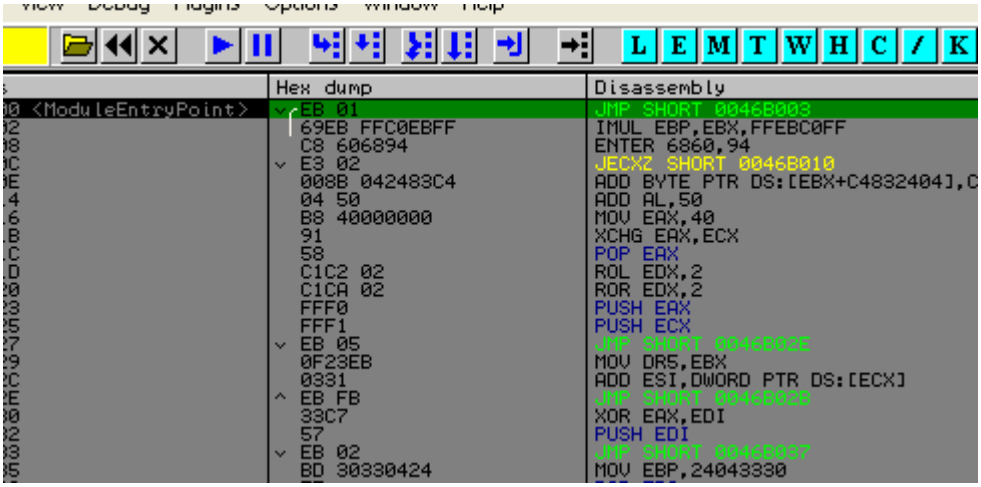


همانطور که می بینید Relocation های فایل درست شده و دیگر مشکلی به وجود نمی آید.اما...اگر در مواردی دیدید که PeTools نمی تواند این کار را انجام دهد.از نرم افزار تخصصی این کار یعنی RoleX استفاده کنید.

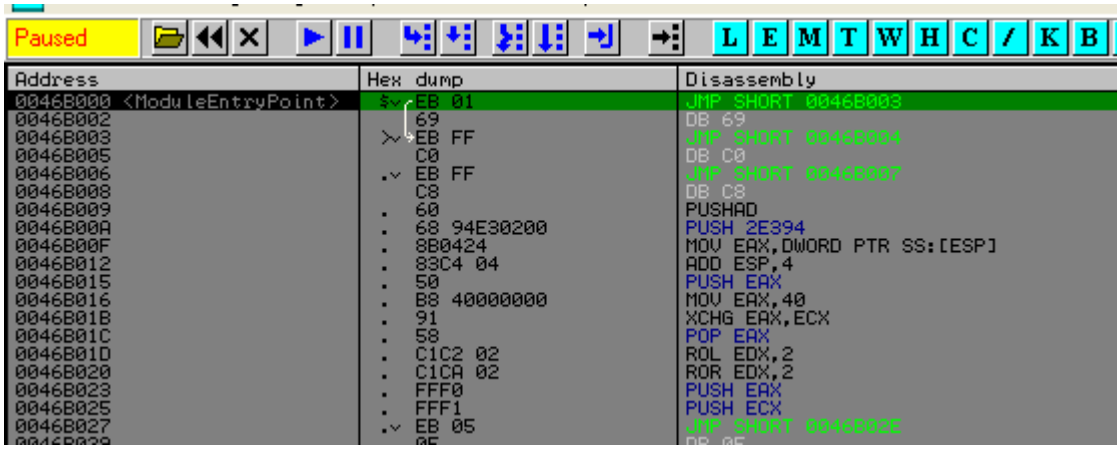
آموزش استفاده از RoleX :ابتدا باید ImageBase فایل DLL خود را تغییر بدهید.برای این کار می توانید از فایل Sample.exe(در داخل برنامه موجود است) استفاده کنید.به این ترتیب که فایل اصلی و یک کپی از فایل مورد نظر را در کنار فایل Sample.exe قرار دهید.تا Sample.exe هر دو فایل DLL را اجرا کند و بعد هم از دو فایلی که Load شده،دامپ می گیرید تا به این ترتیب،دو فایل DLL یکسان،با ImageBase متفاوت داشته باشید.(البته برای این کار از نرم افزار DLL Rebaser 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

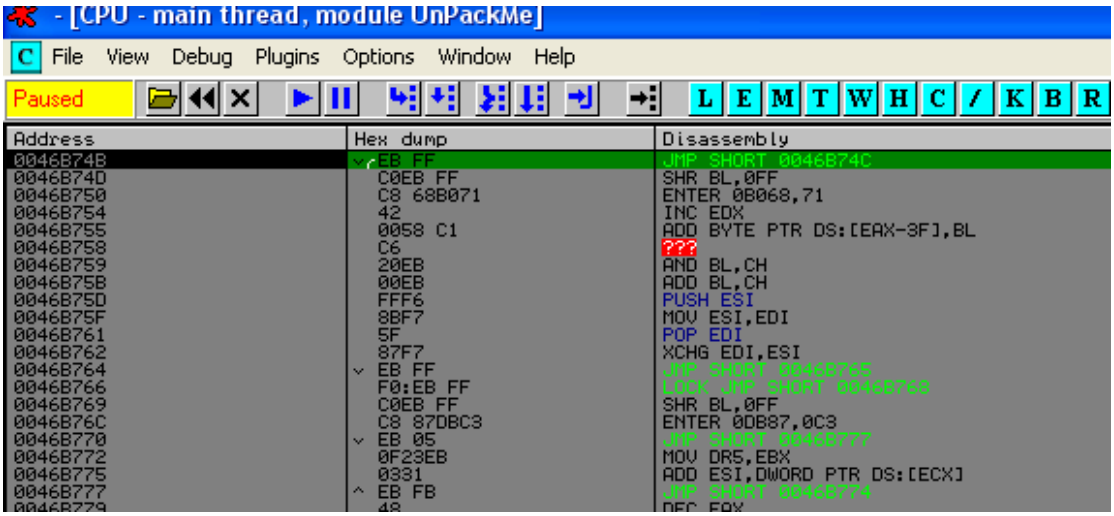
فایل مورد نظر را در OllyDBG باز کنید:



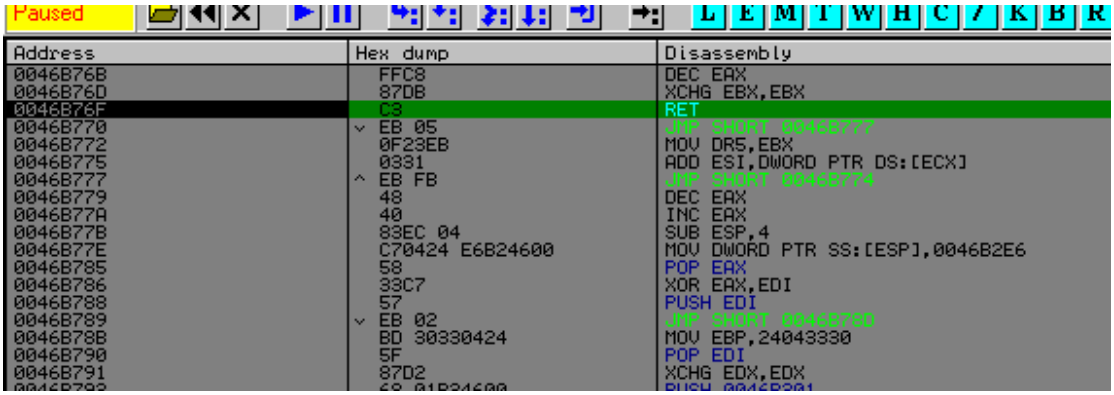
به دلیل قرار دادن پرش های بیخود در داخل کدهای پکر،کدهای پکر مقداری ناخوانا شده است (Junk code) ☹ اما مشکلی نیست.با پلاگین Analysis this اقدام به آنالیز کردن کدهای پکر بکنید.(کلیک راست کرده و گزینه Analysis This را بزنید.)



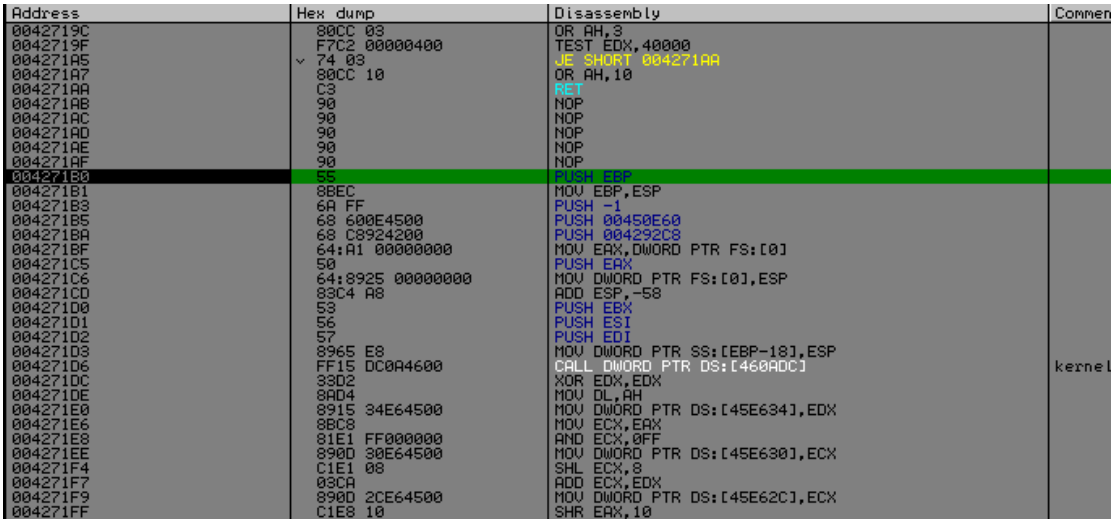
حالا کدها مقداری بهتر شد.با F8 به جلو بروید تا به دستور PushAD برسید و با اجرای دستور PushAD مقدار رجیستر ESP تغییر می کند.حالا هم کلید F9 را بزنید تا برنامه اجرا شود و به انتهای کدهای پکر برسیم.



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



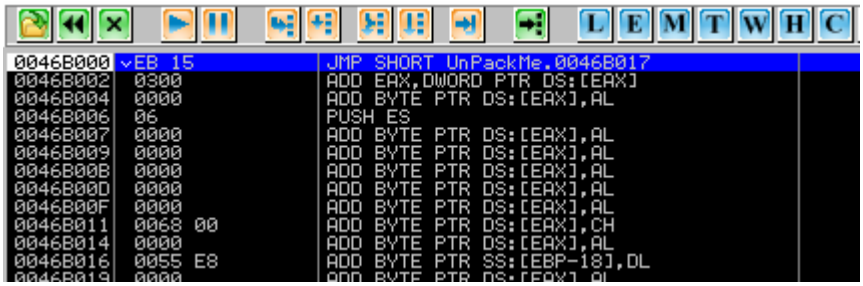
این دستور ما را از سکشن PolyEnE. به سکشن text. می برد.یعنی با این دستور به OEP می رویم،یک بار کلید F8 را بزنید:



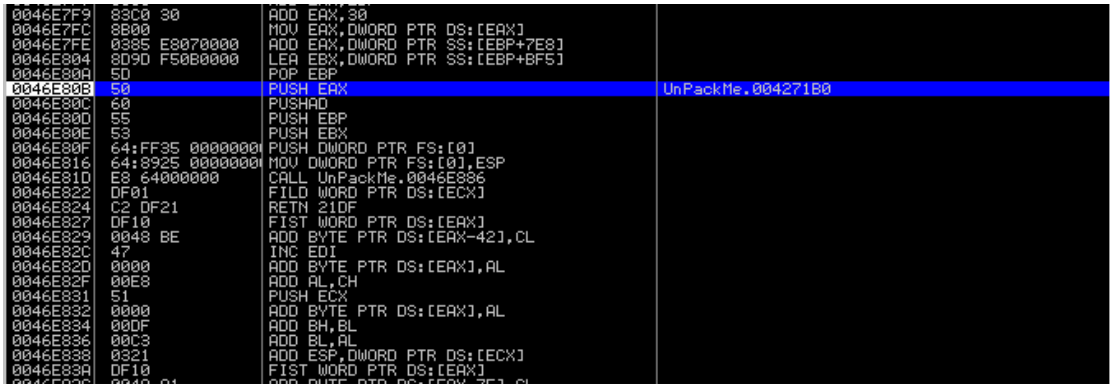
از این به بعد دیگر مشکلی نیست.دامپ می گیرید و فیکس می کنید.

Fusion

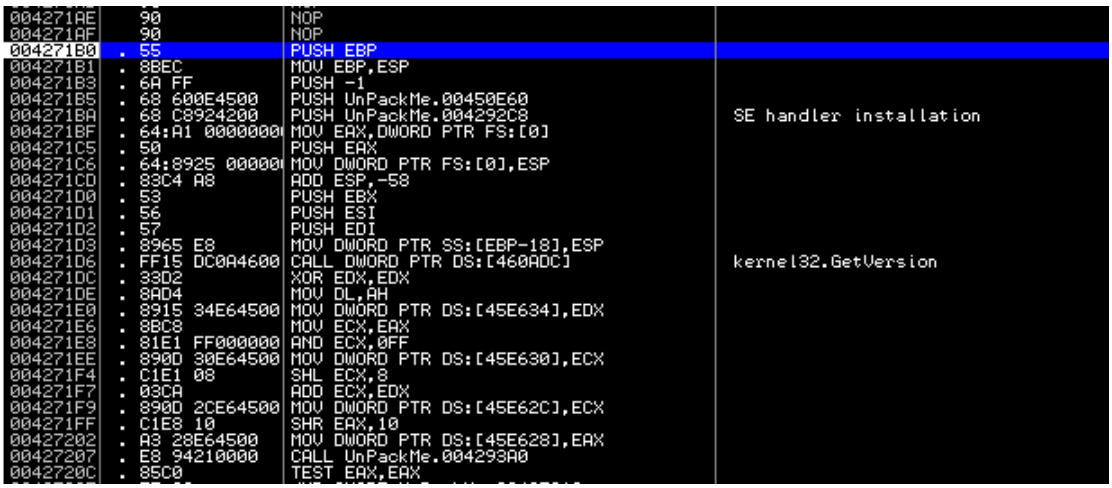
فایل را در OllyDBG باز کنید:



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:

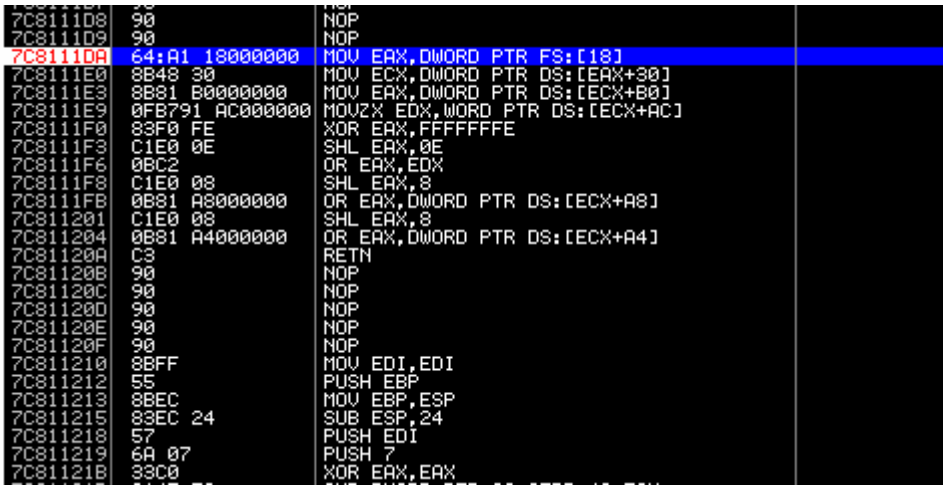


یکبار دیگر کلید F9 را بزنید.اینبار درست در OEP فرود می آیید.

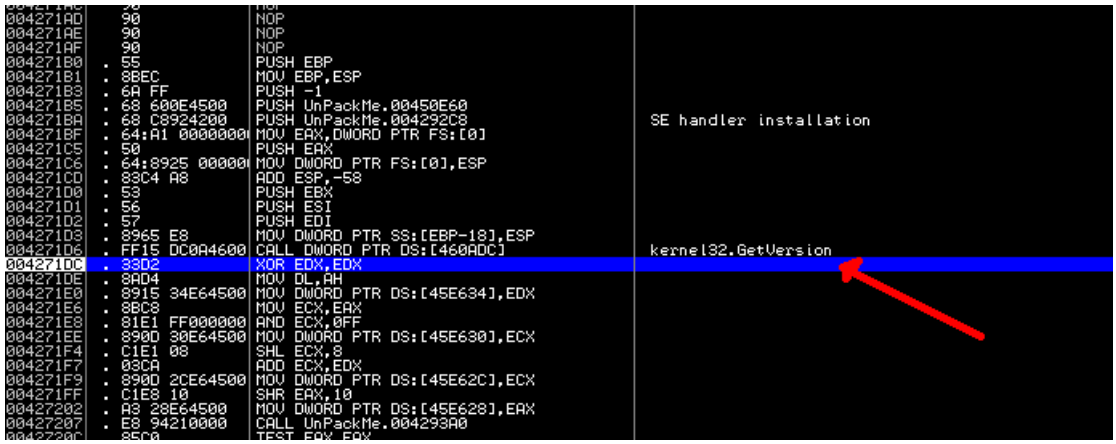


راه حل دیگر:

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 پک 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:



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

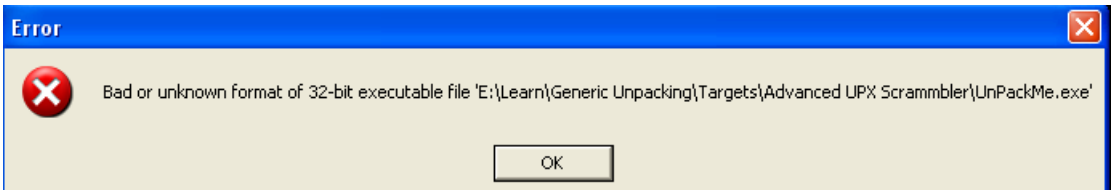


همانطور که می بینید چند خط بالاتر از خطی که الان هستیم،یعنی خط 004271B0،OEP ما قرار دارد ☺
همانطور که می بینید بلد بودن Language Structure کار ما را در یافتن OEP بسیار راحت می کند.و به نظر من یکی از راحت ترین راه ها برای یافتن OEP در مورد پکرهایی است که با ساختار آن آشنایی زیادی ندارید.

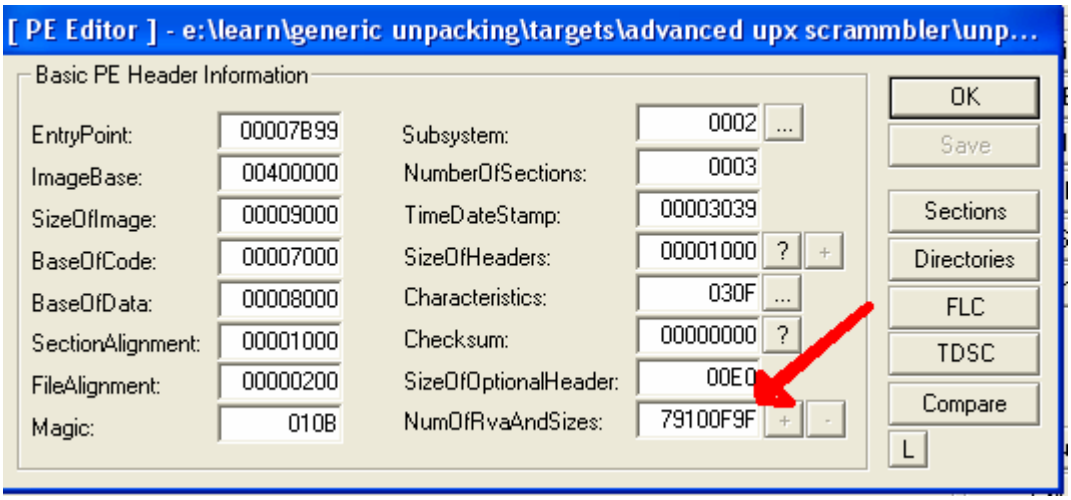
بعد از یافتن OEP مشکل خاصی نیست.دامپ می کنید و دامپ خود را فیکس می کند.(حواستان باشد اگر فایل دامپ شده در سیستم شما اجرا می شود...در سیستم دیگری اجرا نمی شود،**پس حتما باید IAT را فیکس کنید**.)

Advanced UPX Scrammbler

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



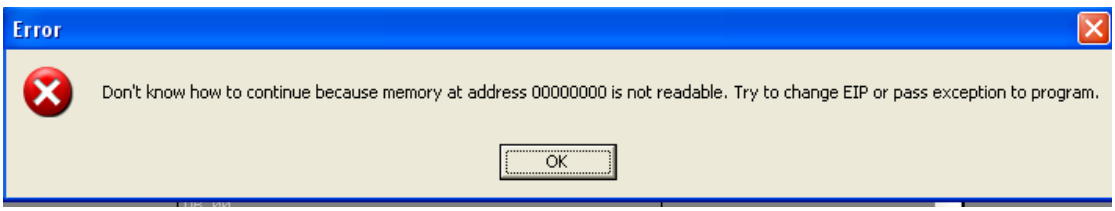
می بینید؟...OllyDBG نتوانسته است مشخصات فایل را درست تشخیص دهد؟...چرا؟...چون پکر ما مشخصات فایل را مقداری دستکاری کرده است(PE header trick)...شاید این دستکاری مشخصات فایل مشکلی در اجرا شدن فایل در ویندوز ایجاد نکند...ولی باعث ایجاد مشکلاتی در زمان اجرای فایل در OllyDBG می شود. ☹ به همین دلیل ما باید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



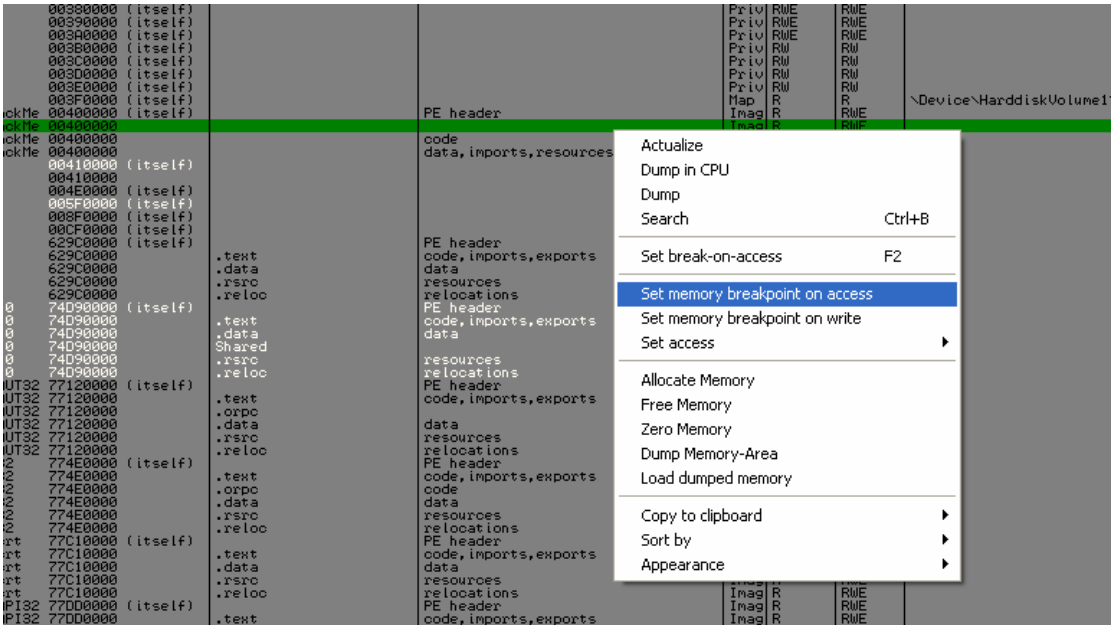
همانطور که می بینید مقدار NumofRvaandSizes برابر برابر 79100F9F است!...این مقدار در فایل های Exe برابر با ۱۰ است.(یا عددی در نزدیکی ۱۰)...لذا این مقدار غیرطبیعی است و به خاطر همین دستکاری،OllyDBG نمی تواند فایل را درست باز کند.
این مقدار را ۱۰ کنید تا فایل به درستی در Olly اجرا شود.البته به غیر از این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

SizeofImage
BaseofCode
BaseofData
TimeDateStamp
SizeofHeaders
Characteristics

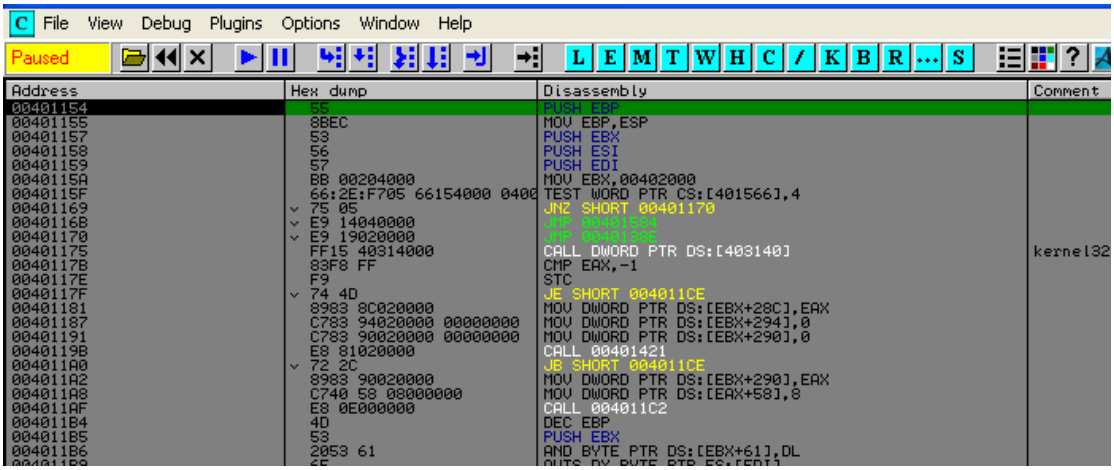
پس همیشه حواستان به این مقدارها باشد.
البته توجه داشته باشید که پلاگین OllyAdvanced این مشکل را در اکثر موارد حل می کند(نه همه موارد).یعنی فایل هایی که این مشکل را دارند،با این پلاگین به راحتی در OllyDBG باز می شوند.
یکی دیگر از راه های پراستفاده در یافتن OEP استفاده از Memory Breakpoint on Access می 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 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.



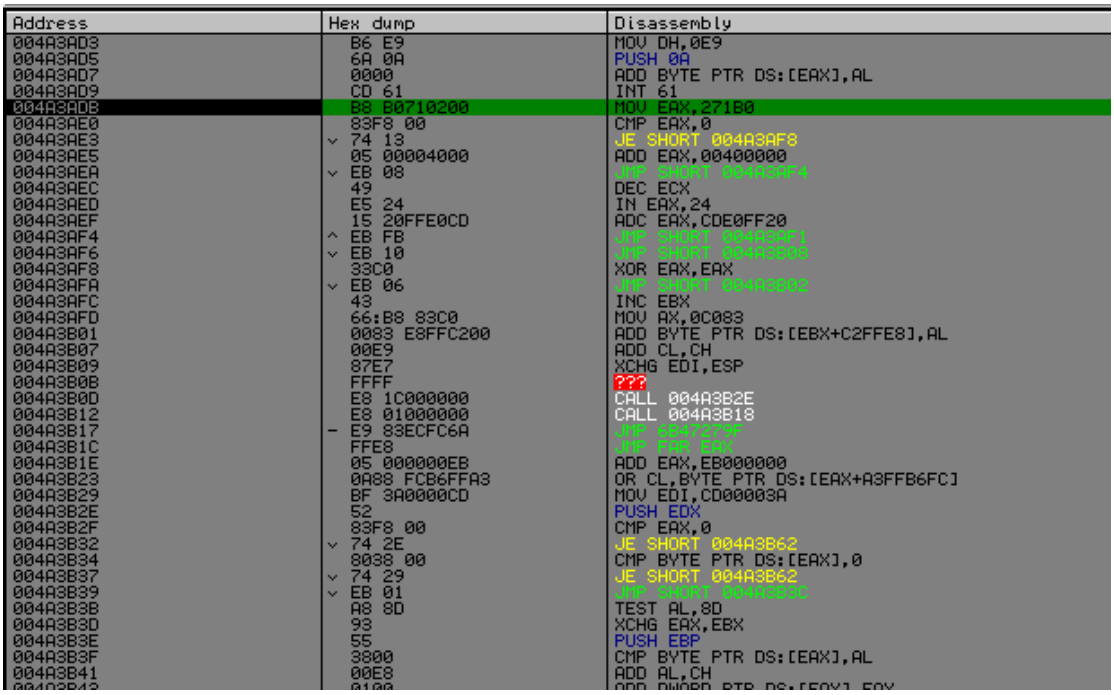
حالا کلید Shift + F9 را بزنید.(چون در برنامه خطا اتفاق افتاده برای ادامه برنامه باید کلید Shift را هم نگه دارید.)



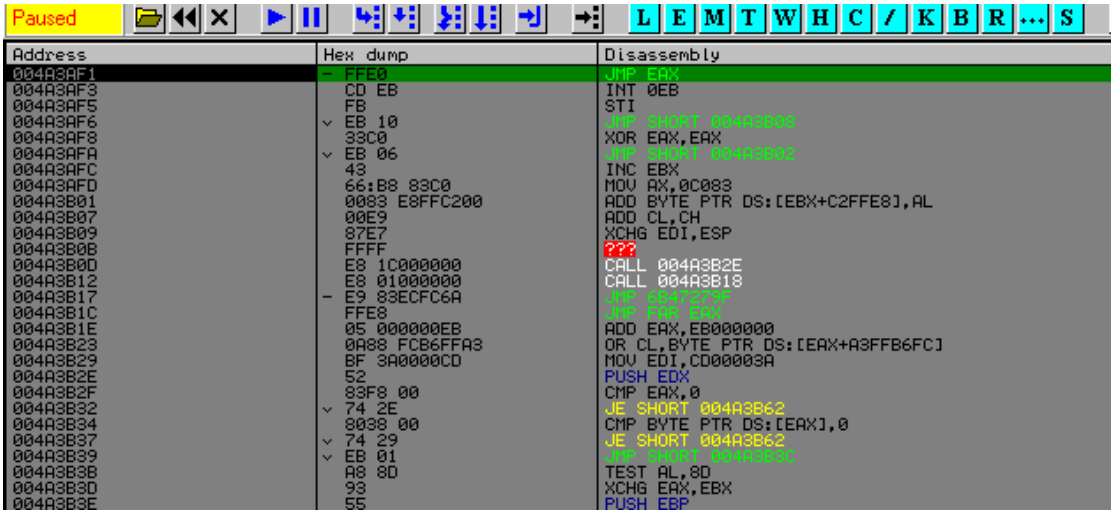
همانطور که می بینید وارد سکشن اصلی برنامه شدیم و به OEP رسیدیم...حالا از فایل دامپ بگیرید و آن را فیکس کنید.

Orien

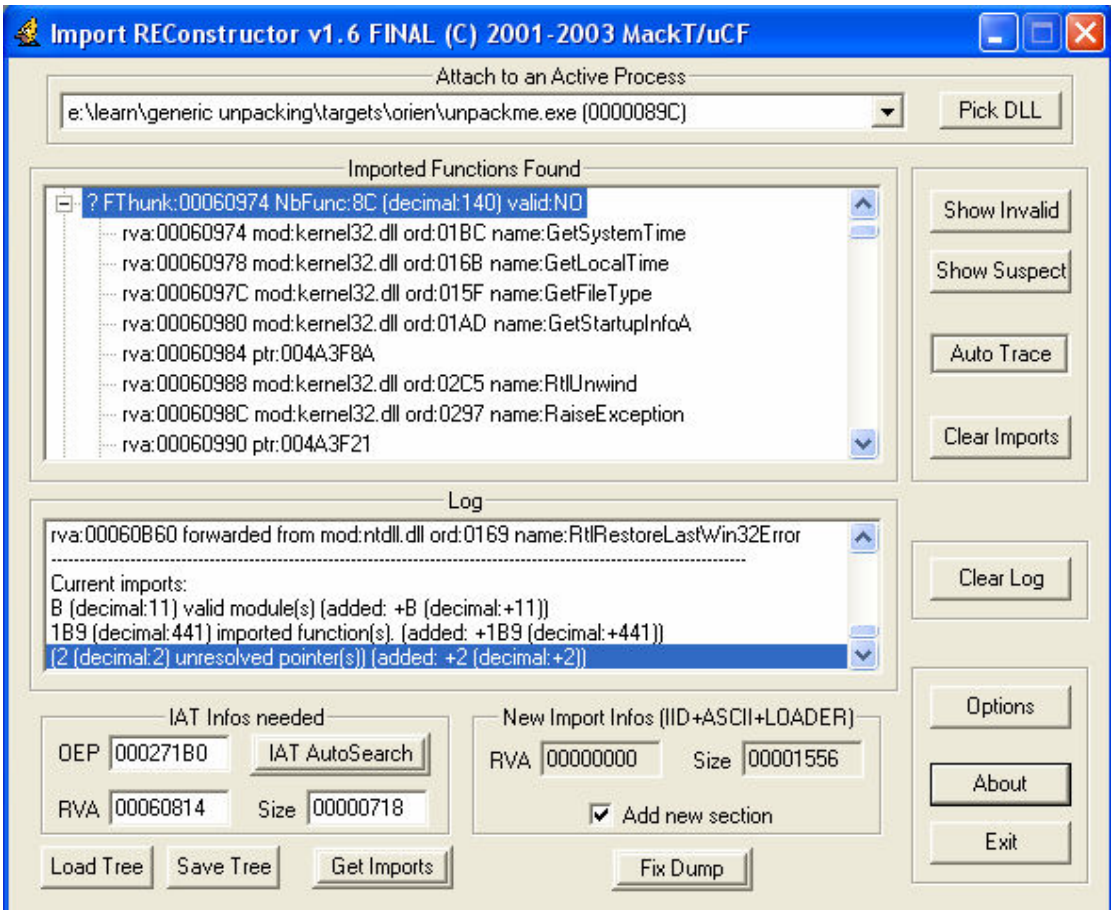
فایل مورد نظر را در OllyDBG باز کنید.دوبار کلید F8 را بزنید تا دستور PushAD اجرا شود.حالا بر روی مقدار رجیستر ESP یک Hardware Breakpoint on access بگذارید و بعد کلید F9 را بزنید:



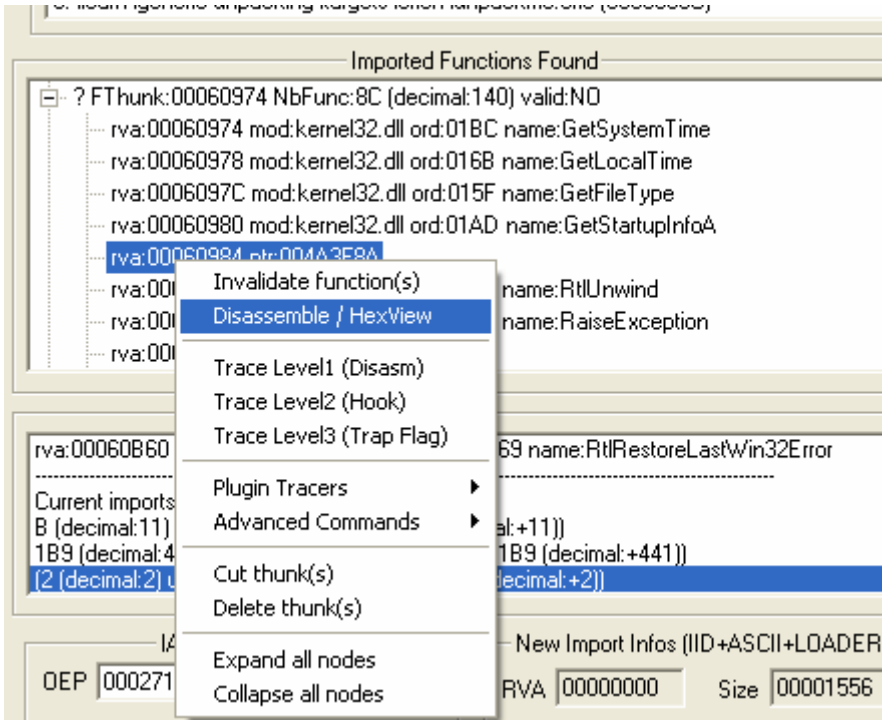
این دستور دیگر آخر کدهای پکر ماست.کافیست چند بار کلید F8 را بزنید تا به این خط برسید:

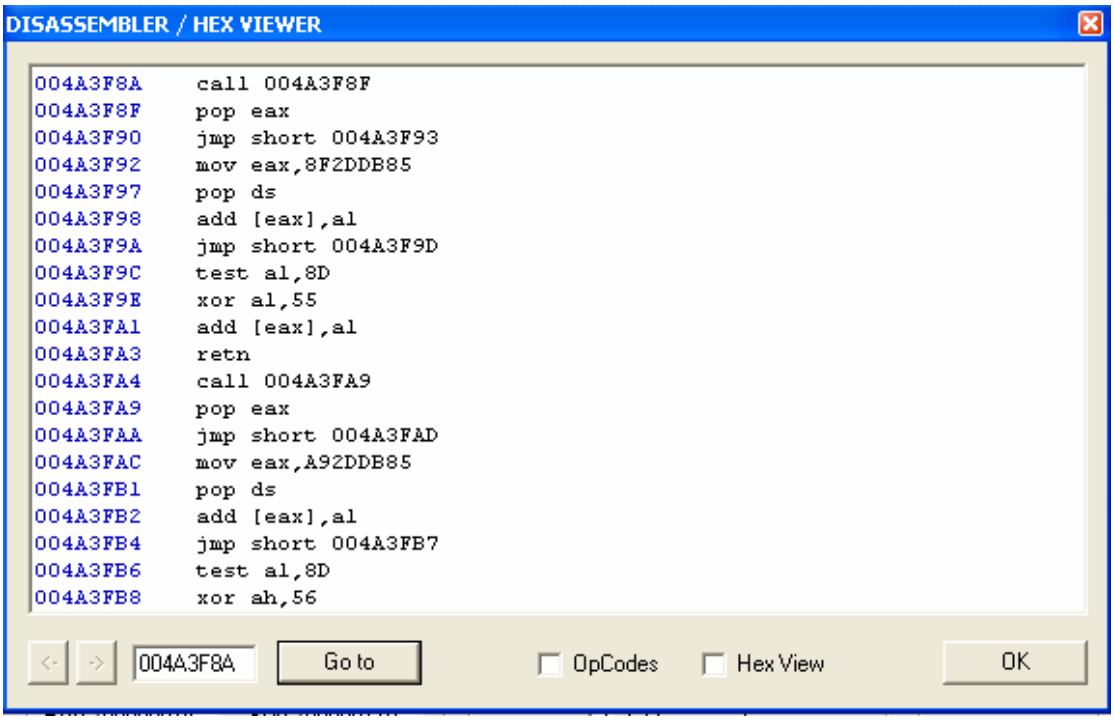


این دستور درست به OEP می رود.
حالا از فایل دامپ بگیرید و بعد ImportREC را باز کنید.فایل مورد نظر را به برنامه می دهید.و OEP بر حسب RVA را هم بنویسید:

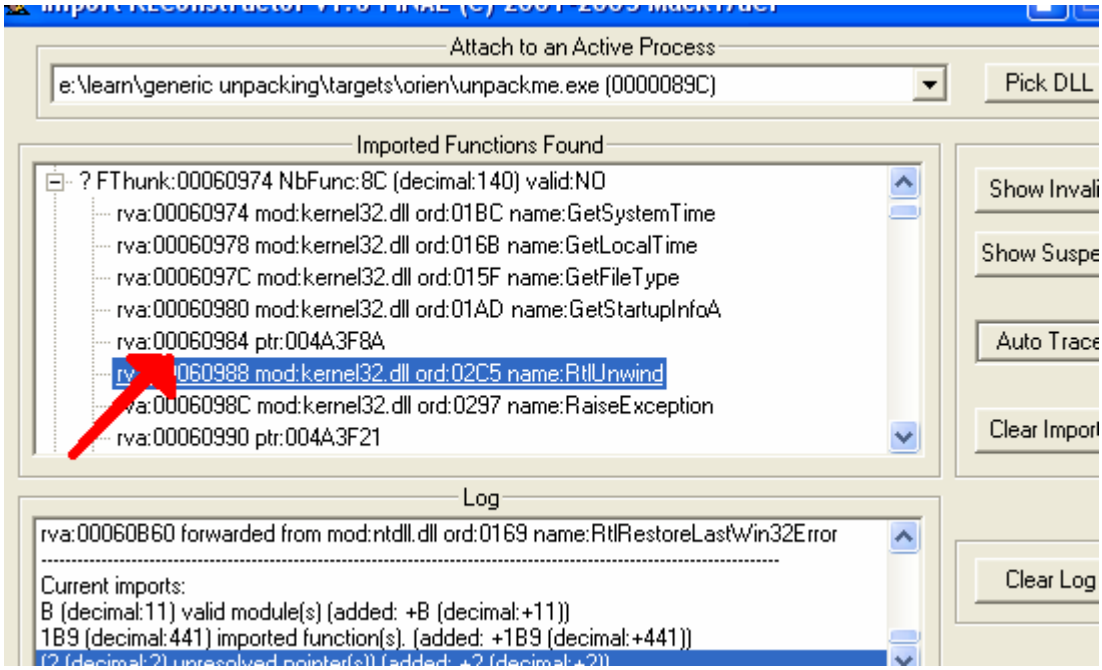


همانطور که می بینید دو تابع Invalid داریم ☹ ممکن است فکر کنید که این توابع بدردنخور هستند و می توانند بر روی آنها کلیک راست کرد و گزینه Cut Thunk را زد و آنها را از بین برد.ولی نه!...این طور نیست...از کجا فهمیدم؟...بر روی یک از توابع Invalid کلیک راست کنید و گزینه Disassemble/Hex View را بزنید:



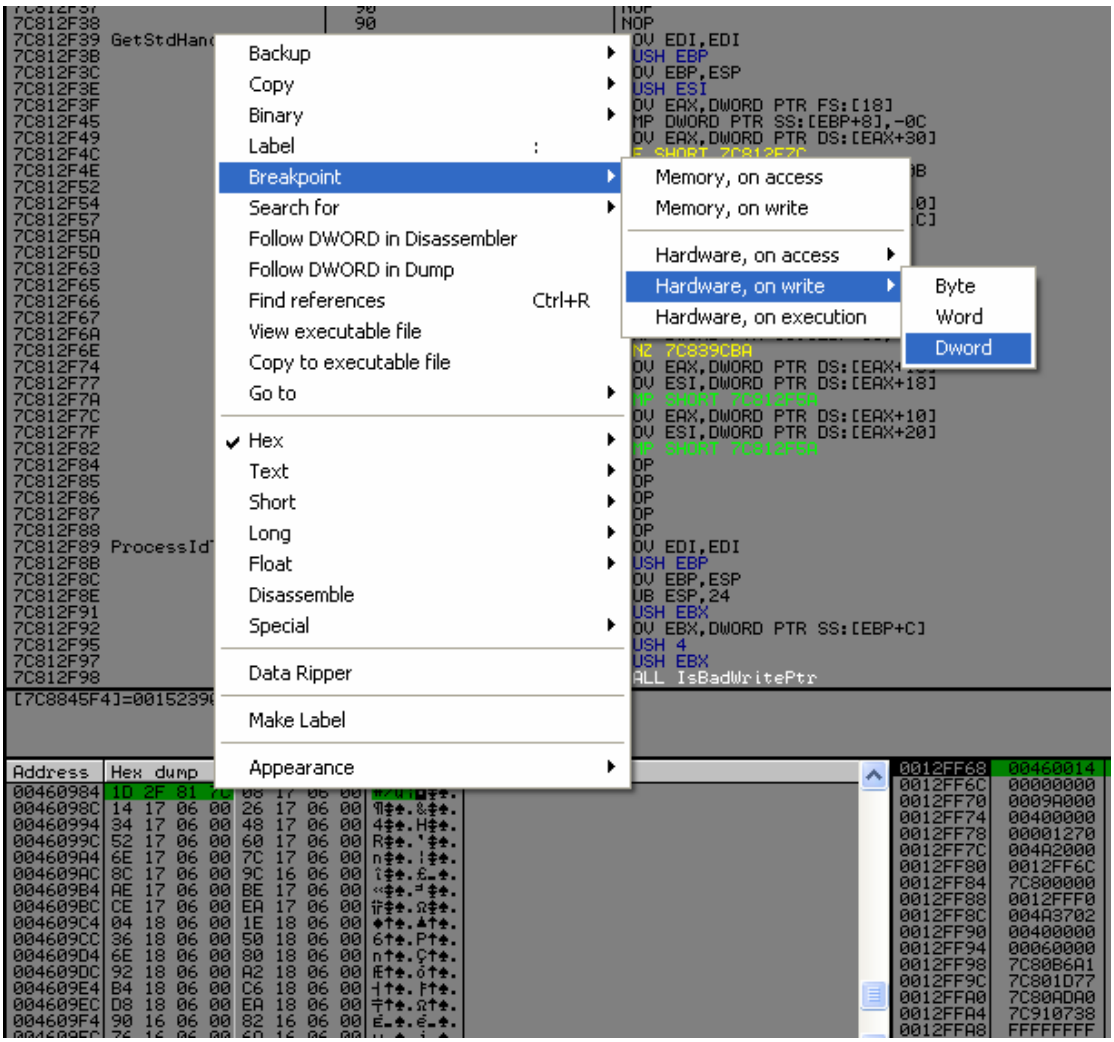


همانطور که در تصویر بالا می بینید دستورهای بالا دستوراتی هستند که مشخص و معنی دار هستند.(تابع دیگر هم همینطور است).لذا این دو تابع نمی توانند دو تابع بدردنخور باشند.
ببینید...این پکر از قابلیت Import Redirection استفاده می کند.**یعنی به جای اینکه بیاید آدرس واقعی این دو تابع را بنویسد،آدرس خطی از برنامه را می نویسد که به آدرس واقعی تابع می رود.**
خوب،حالا چه کار کنیم؟...راه حل های زیادی برای ازبین بردن Import Redirection وجود دارد.که یکی از معمولترین آنها این است که ما در درون کدهای پکر جستجو کنیم و ببینم 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.)

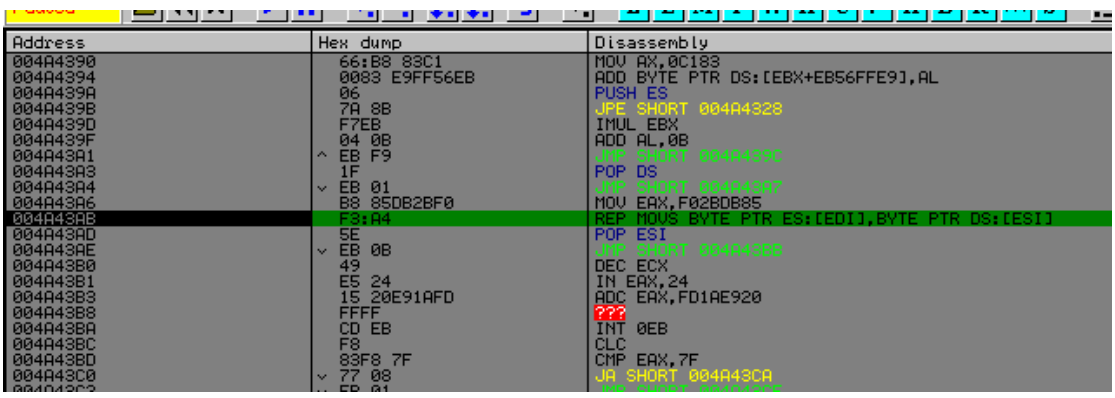


همانطور که در تصویر بالا می بینید یکی از توابع Invalid در خط 00060984 قرار دارد.برای اینکه Magic Jump را پیدا کنیم.باید بر روی این خط یک Hardware Breakpoint بگذاریم.
پس در OllyDBG در پنجره دامپ کلیک راست کرده،گزینه Go to و سپس گزینه Expression را بزنید و محل تابع Invalid را به برنامه بدهید،گزینه RVA را هم تیک بزنید:

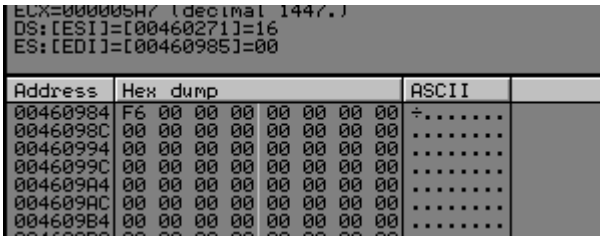
حالا همانند تصویر بر روی خط مورد نظر (460984) یک Hardware Breakpoint on write بگذارید...ممکن است سوال کنید چرا on write گذاشتم؟...خوب برای اینکه من می خوام ببینم این آدرس در کجای برنامه **نوشته شده است**.



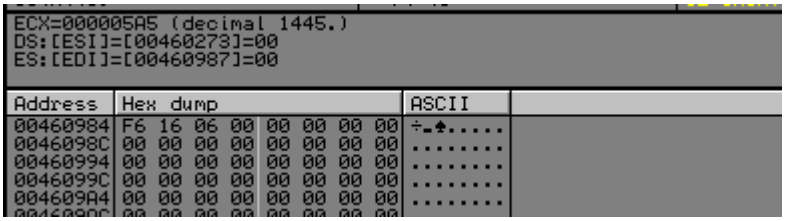
حالا برنامه را Re-Start کرده و با F9 اجرا کنید.من در اینجا متوقف شدم:



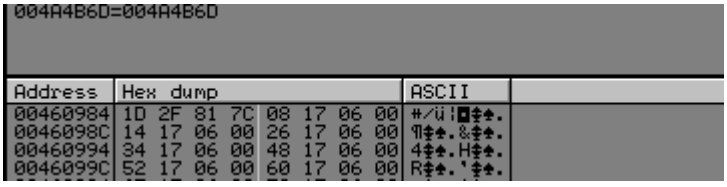
خوب،بهتر است ببینم که چه بلایی بر روی تابع مورد نظرم می آید؟برای این کار در منوی Debug گزینه Hardware Breakpoints را بزنید و گزینه Follow(جلوی خط 460984) را بزنید.



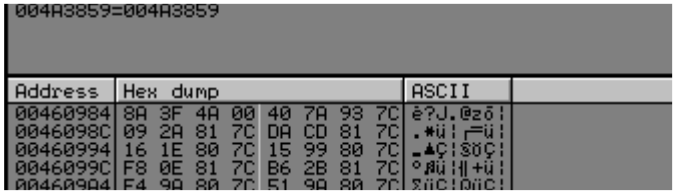
همانطور که در تصویر بالا مشاهده می کنید،هنوز آدرس در خط 460984 نوشته نشده است.پس یکبار دیگر کلید F9 را بزنید.



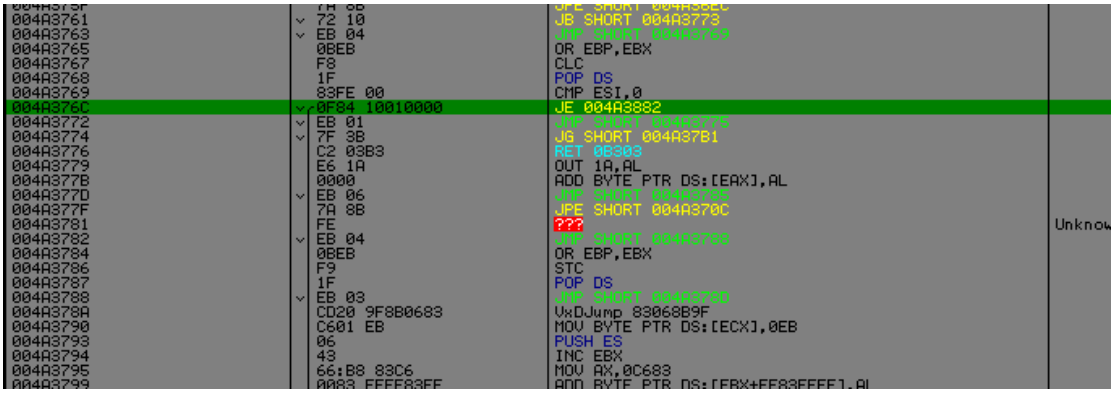
اینبار هم هنوز آدرسی در خط 460984 نوشته نشده است.یکبار دیگر کلید F9 را بزنید:



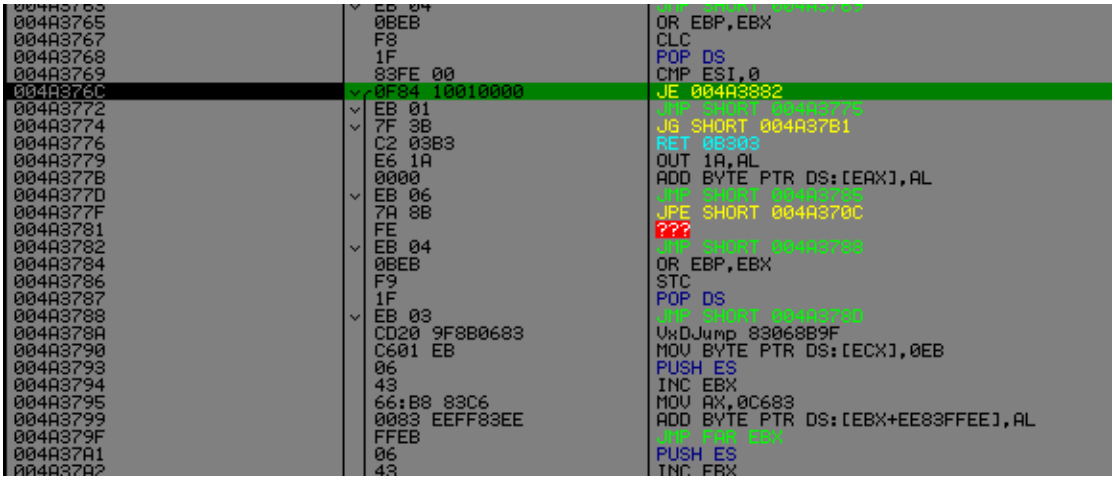
آها!...همانطور که می بینید در خط 460984 نوشته شده: 7C812F1D (باید بایت ها را از آنور بخوانید...بلد هستید که؟...قبلا توضیح دادم...در ضمن این آدرس در سیستم شما فرق دارد.)
این آدرس تابع GetCommandLineA دز سیستم من است،پس این آدرس در واقع GetCommandLineA بوده ☺
یکبار دیگر کلید F9 را بزنید:



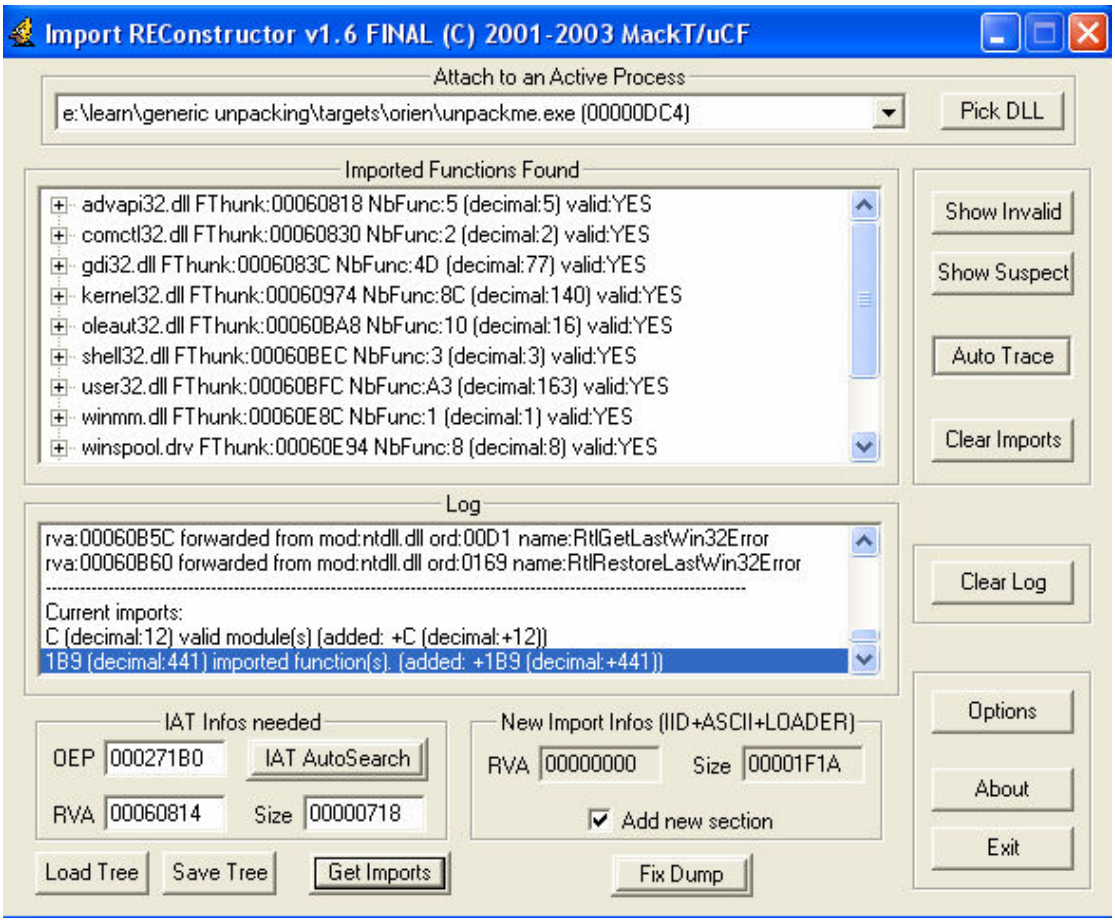
می بینید؟...آدرس GetCommandLineA پاک شده و جای آن خط 4A3F8A نوشته شده؟...حالا باید چه کار کنم؟...در پنجره Dissembler با موس اندکی به بالا بروید.تا به اینجا برسید:



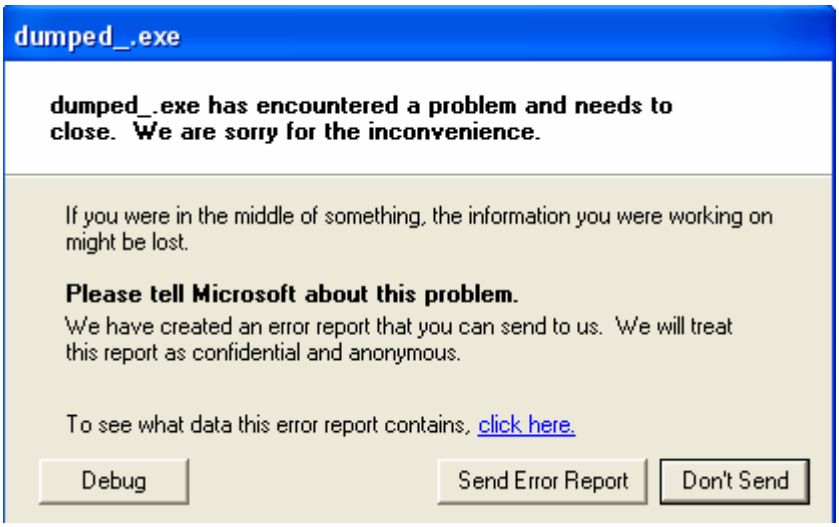
این یک پرش بسیار بلند است.به نظر من این پرش تصمیم گیرنده است...یعنی اگر پرش انجام می شود.این کدها اجرا نمی شود و در نتیجه تابع من Invalid نمی شود.پس:
تمامی Breakpoint ها را پاک می کنم و بعد بر روی این پرش یک Hardware Breakpoint on execution می گذارم.(کلیک راست کرده،گزینه Breakpoint و بعد گزینه Hardware 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:



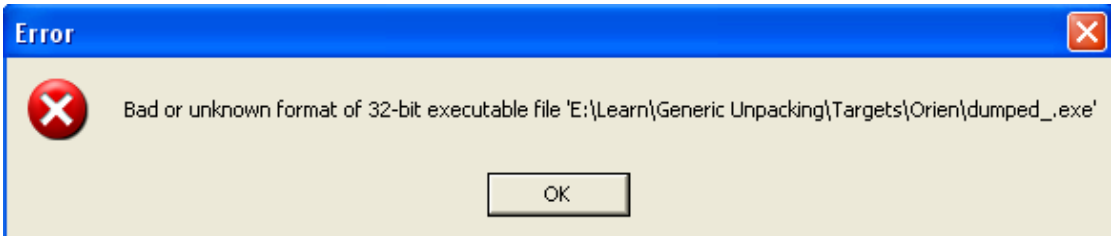
همانطورکه می بینید این پرش انجام نمی شود و کدهایی که آن دو تابع ما را Redirect می کنند اجرا می شود.پس باید کاری کنیم که این پرش همواره اجرا شود.(آن را JMP می کنیم.)...و بعد برنامه را اجرا می کنیم.
حالا دوباره با ImportREC فایل را باز می کنیم.OEP را می دهیم و گزینه IAT Auto-search را می زنیم و بعد Get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



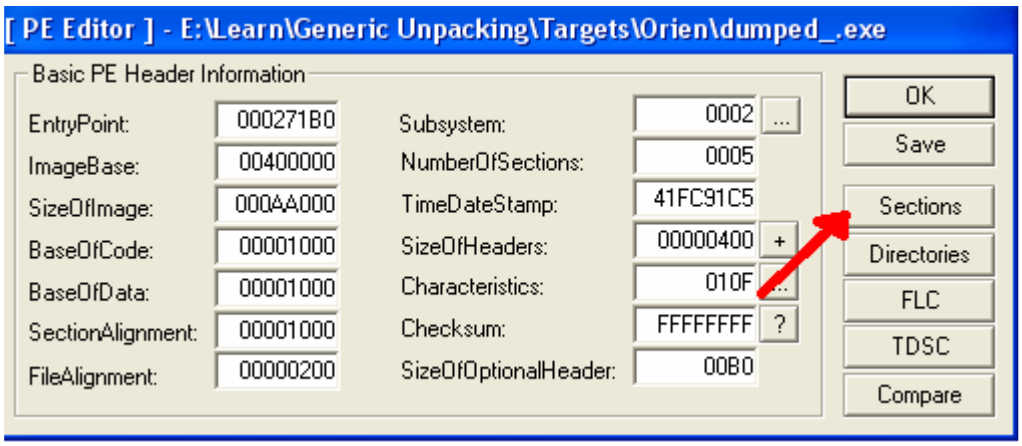
همانطور که می بینید این بار تمامی توابع ما Valid هست...چرا؟...چون آن کدهایی که دو تا از توابع ما را Invalid می کرد،اجرا نشده است.
حالا فایل دامپ شده خود را فیکس کنید.و آن را اجرا کنید:



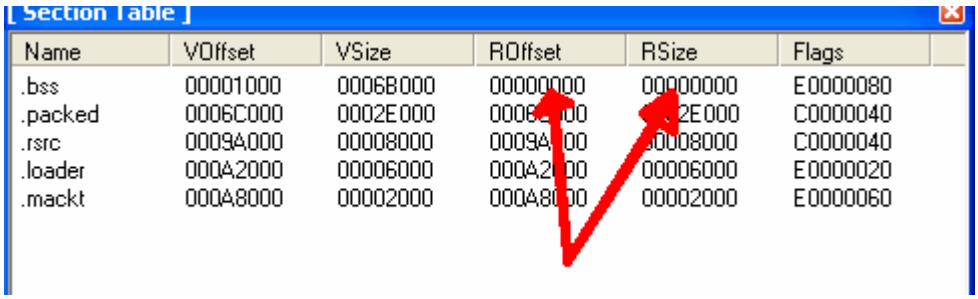
بهتر است فایل را در OllyDBG باز کنیم تا ببینم مشکل چیست و در کجا برنامه دچار مشکل می شود؟:



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:

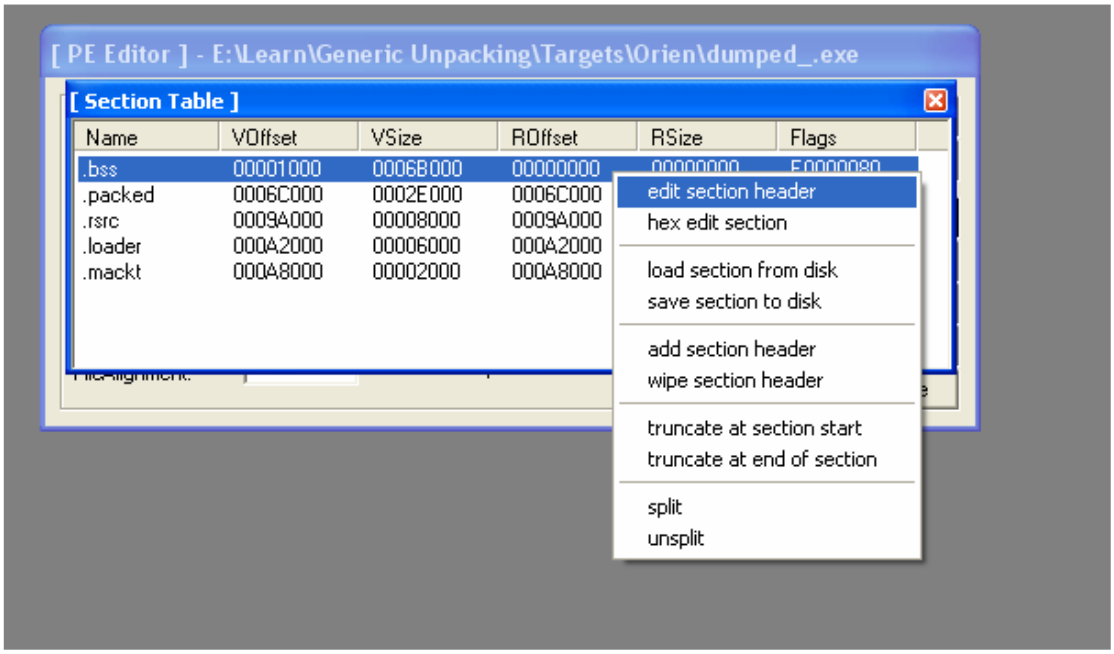


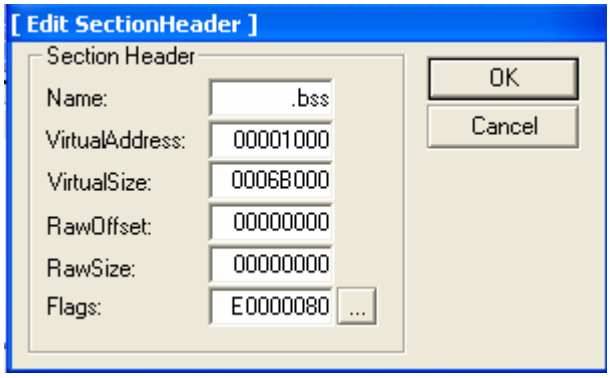
گزینه Sections را بزنید تا تمامی سکشن های فایل آنپک شده نمایش داده شود:



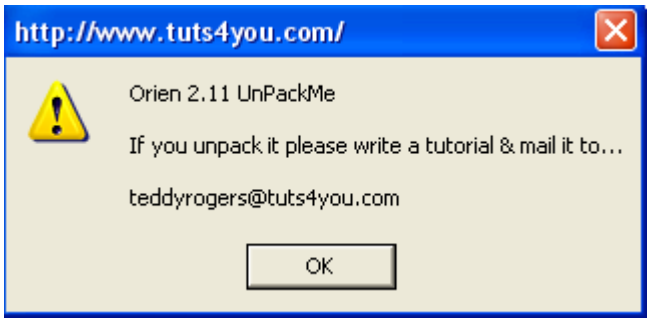
Voffset(مخفف Virtual offset) نشان دهنده آدرسی از حافظه است که وقتی این فایل اجرا می شود این سکشن در این آدرس قرار می گیرد.
ROffset(مخفف RAW Offset) هم نشان دهنده آدرسی است که این سکشن در داخل فایل موجود است
معمولا VOffset با ROffset برابر است.و VSize با RSize

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:

۱.STO --> این دستور همان کاری را می کند که کلید F8 انجام می دهد.(Step over)
۲.STI --> این دستور همان کاری را می کند که کلید F7 انجام می دهد.(Step into)
۳.Run --> این دستور همان کاری را می کند که کلید F9 می کنید.(Run Program)
۴.var --> از این دستور برای تعریف یک متغیر که در داخل اسکریپت مورد استفاده قرار می گیرد،استفاده می شود.
۵.Bphws --> با این دستور می توانید Hardware Breakpoint بگذارید.
۶.CMT --> با این دستور می توانید جلوی آدرس خاصی Comment بگذارید.
۷.an --> کدهای یک سکشن را آنالیز می کند.(CTRL + A)
۸.Msg --> از این دستور برای نمایش یک پیغام نمایش داده می شود.
۹.Ret --> با این دستور از اسکریپت خارج می شوید.
۱۰.MSGYN --> برای نمایش پیغامی که از کاربر سوال می کند،استفاده می شود.
۱۱.dpe --> با این دستور می توانید از فایل خود دامپ بگیرید.
۱۲.log --> با این دستور می توانید اطلاعاتی را به پنجره Log در پلاگین ODBGScript اضافه کنید.
۱۳.gpa --> با این دستور می توانید آدرس یک تابع خاص(مثل MessageboxA) را بگیرید.
۱۴.Rtu --> همان کاری را می کند که کلید های ALT + F9 می کنند(Return till user code)
۱۵.Findop --> این دستور می تواند دستور مشخص شده ای را در کد پیدا کند.
۱۶.bp --> Breakpoint می گذارد.
۱۷.Mov --> همان کاری که دستور Mov می کند.البته این دستور در اسکریپت نویسی کاربردهای زیادی دارد.مثلا برای تغییر یک دستور
۱۸.ASM --> با این دستور می تواند خط خاصی را تغییر دهید.

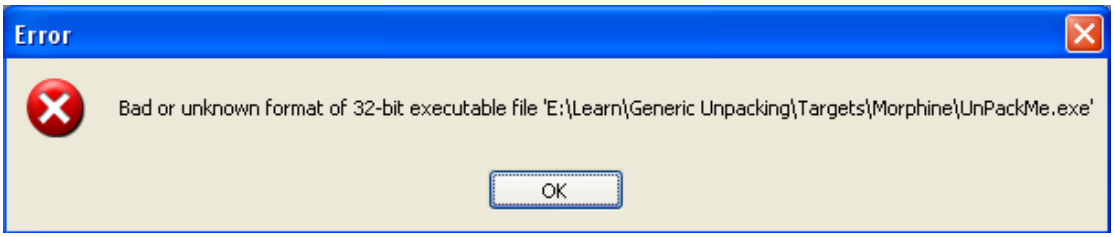
البته درباره این دستورات تنها یک توضیح کوچک داده شده،برای به دست آوردن لیست کامل دستورات و توضیحات بیشتر به فایلی که در Attachments به نام Script instroductions موجود است مراجعه کنید(صفحات ۸ تا ۲۶).در ضمن برخی از این دستورات ممکن است در ورژن ODBGScript شما پشتیبانی نشود.

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:

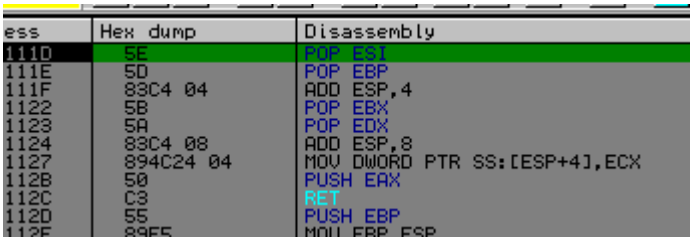
```
*/
Orien OEP Finder & Dumper
Author=AmirGooran
E-Mail=Amir7687@yahoo.com
/*

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

Sto // کلید F8را می زنم
Sto // کلید F8 را می زنم
Bphws ESP,"r" // یک Hardware breakpoint on access روی مقدار رجیستر ESP می گذارم
Run //F9 با این دستور برنامه را اجرا می کنم.همانند کلید

Loop: // یک حلقه ساختم تا ببینم دستور Jmp EAX کجاست؟
Sto // کلید F8 را می زنم
Findop eip, #FFE0# // به دنبال دستور jmp EAX می گردم
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 می زنم
Cmt eip,"This is OEP ;)" //This is oep در اینجا به OEP رسیدم،پس جلوی این آدرس می نویسم:
Mov Path,"C:\Dumped.exe"// متغیر Path را برابر با آدرس C:\dumped.exe می گیرم
Dpe Path, eip // از فایل دامپ می گیرم
Msg Path// آدرس جایی که فایل دامپ شده را گذاشتم را به صورت پیغام به کاربر نمایش می دهم
Ret// اسکریپت در این خط تمام می شود
```

Morphine

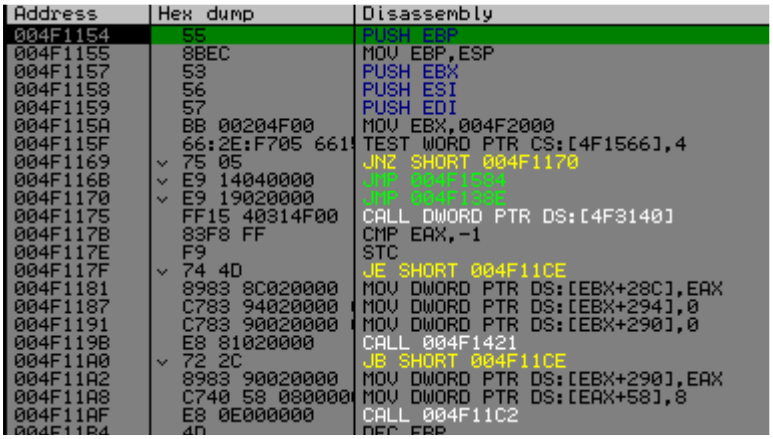
فایل مورد نظر را در OllyDBG باز کنید.این پکر یک سکشن جدید در Memory Map می سازد و در این سکشن فایل آنپک شده را قرار می دهد،برای یافتن OEP می توانیم از رجیستر ESP استفاده کنیم.
پنج بار کلید F8 را بزنید تا مقدار رجیستر ESP تغییر کند و بعد بر روی مقدار رجیستر ESP یک Hardware BP on access بگذارید،و کلید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



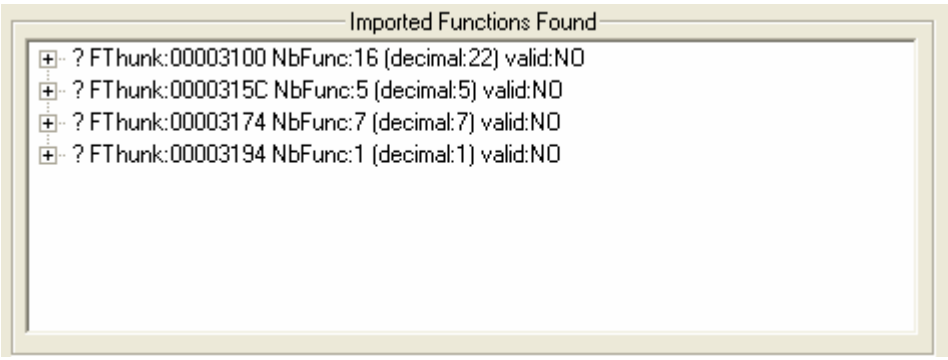
ممکن است سوال کنید چرا اینجا این پیغام داده شده؟...به خاطر اینکه مرفین الان فایل را Decrypt کرده و در یک سکشن جدید آن را ساخته است ☺ کلید OK را بزنید تا این پیغام برود.



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

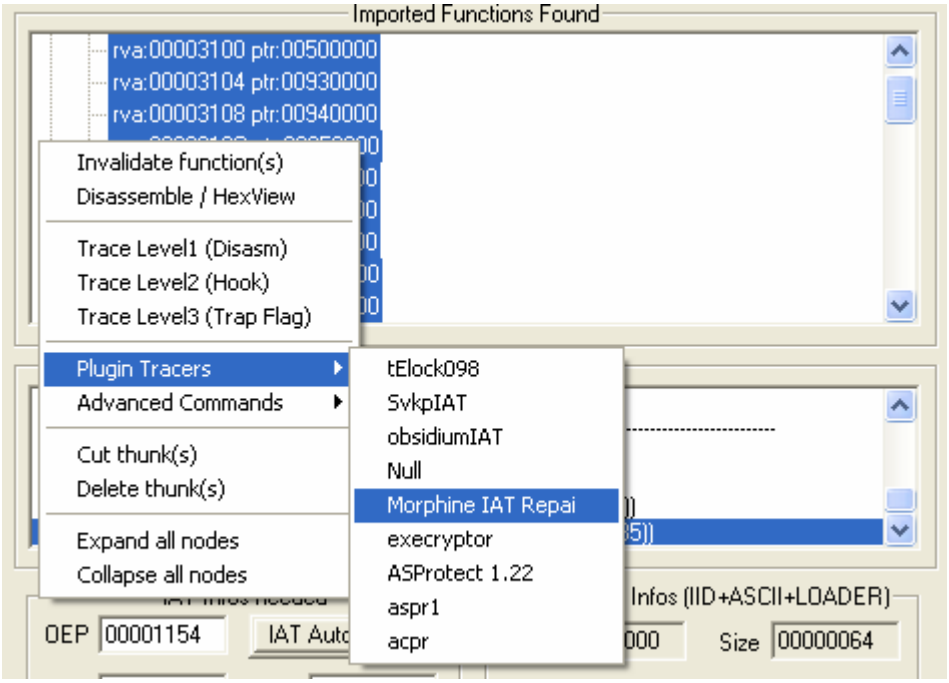


حالا از فایل دامپ بگیرید.(برای دامپ گرفتن از پلاگین OllyDump استفاده نکنید.)
و ImportREC را باز کنید(OEP بر حسب RVA برابر ۱۱۵۴ است.):

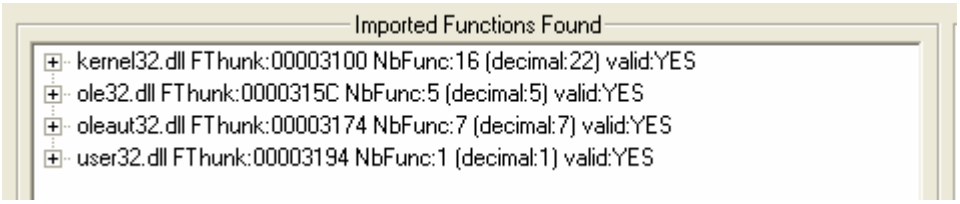


تمامی توابع Redirect شده اند.برای درست کردن آنها،می توانیم از پلاگین استفاده کنیم.یکی از قابلیت های خوب ImportREC هم همین است که می توان به آن پلاگین اضافه کرد. ☺ گزینه Show Invalid را بزنید و کلیک راست کرده و گزینه Plugin tracers و

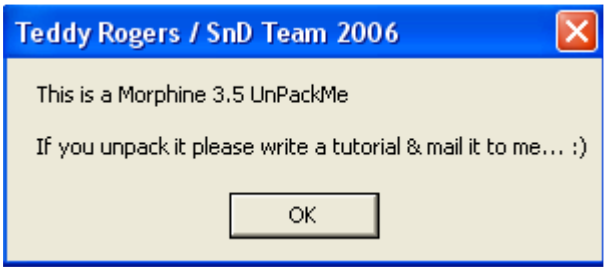
سپس گزینه Morphine IAT Repaier را بزنید.(البته برای درست کردن توابع راه های دیگری هم وجود دارد.مثلا این کار را دستی انجام بدهیم ☹)



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

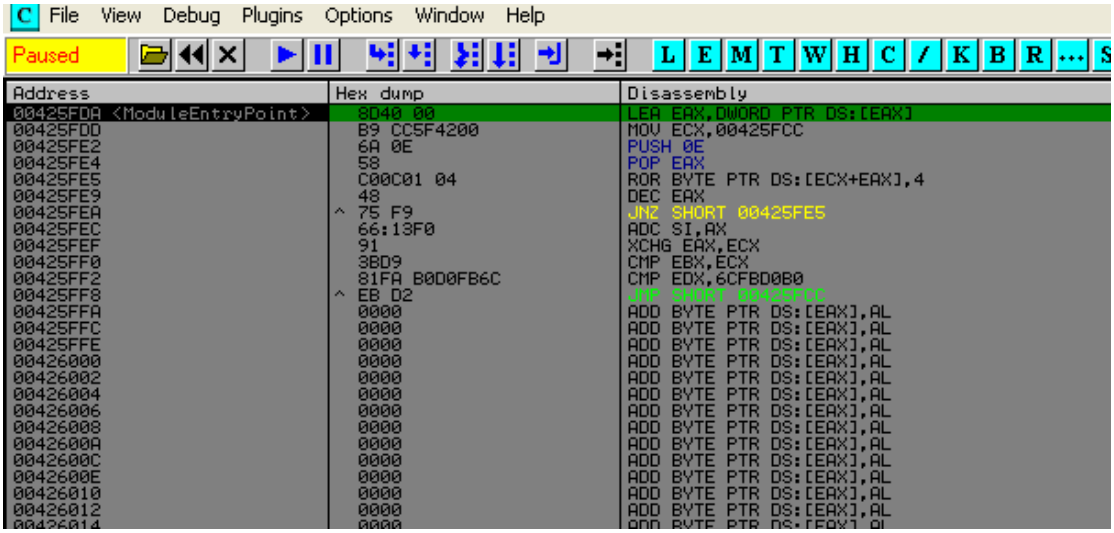


حالا فایل دامپ را فیکس کنید:

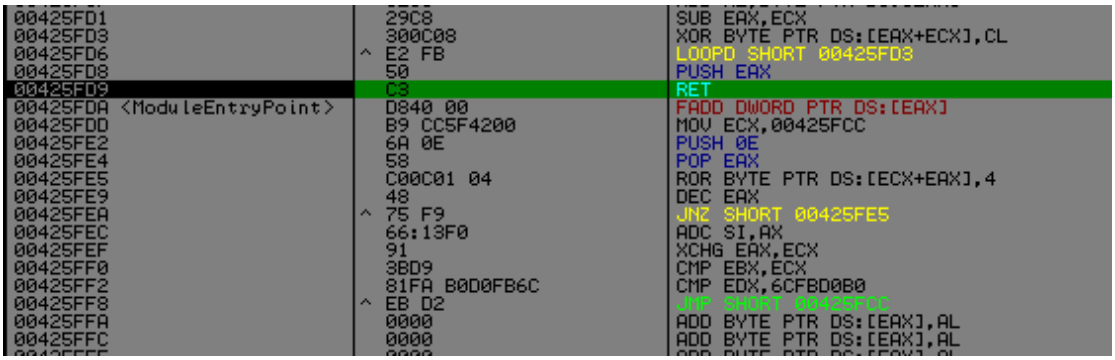


SPLayer

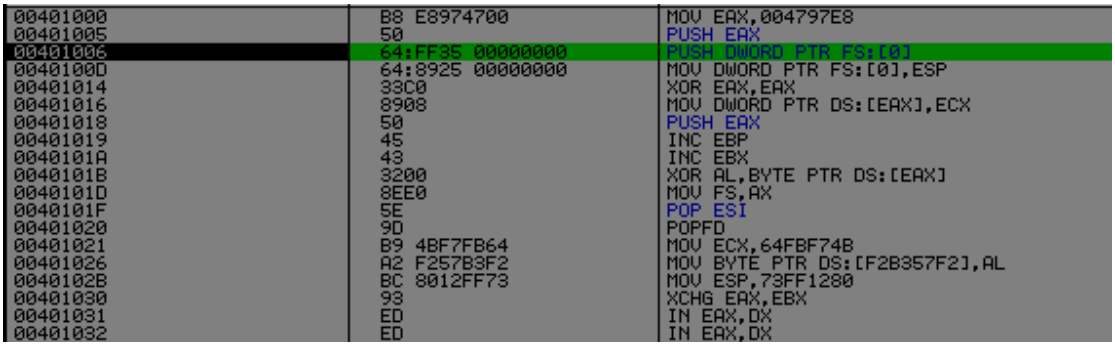
برنامه را در OllyDBG باز کنید:



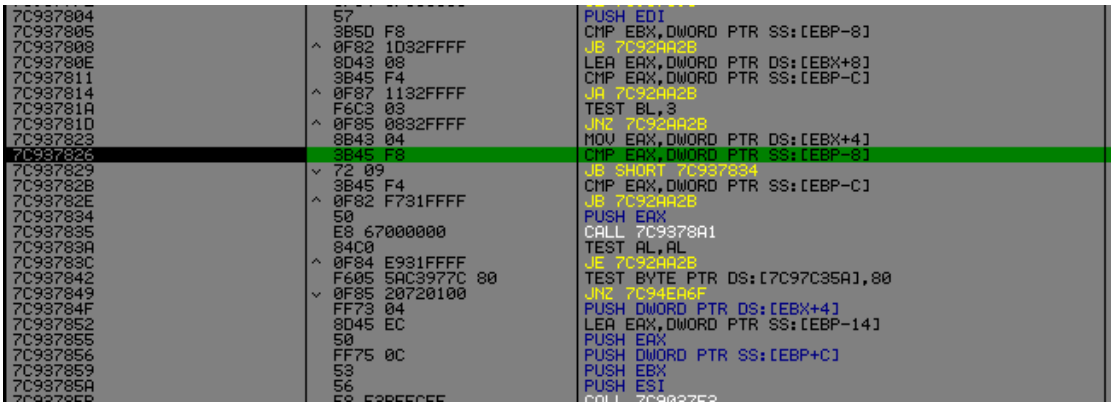
سه بار کلید F8 را بزنید تا مقدار رجیستر ESP تغییر کند و بعد هم روی مقدار رجیستر ESP یک Hardware breakpoint on access می گذاریم،و بعد کلید F9 را می زنم من در اینجا متوقف شدم:



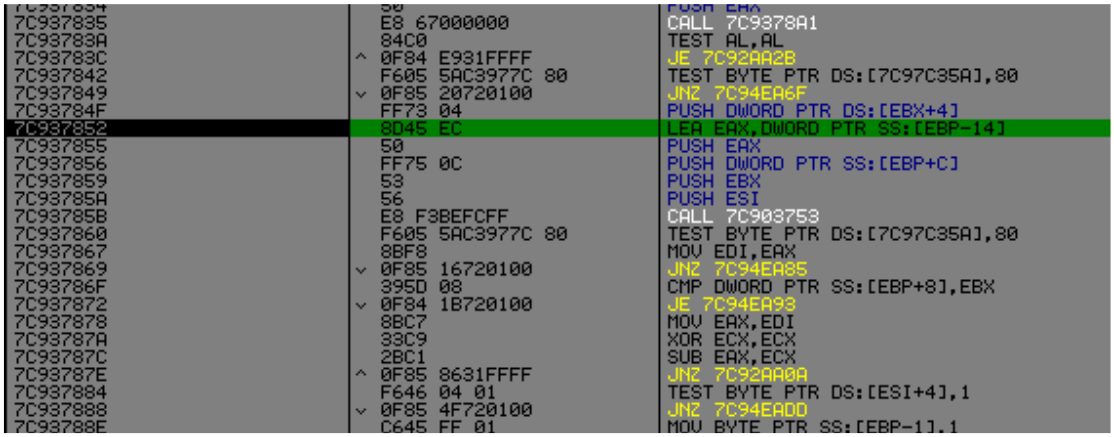
چیز جالبی توی این خط نیست،یکبار دیگر کلید F9 را می زنم:



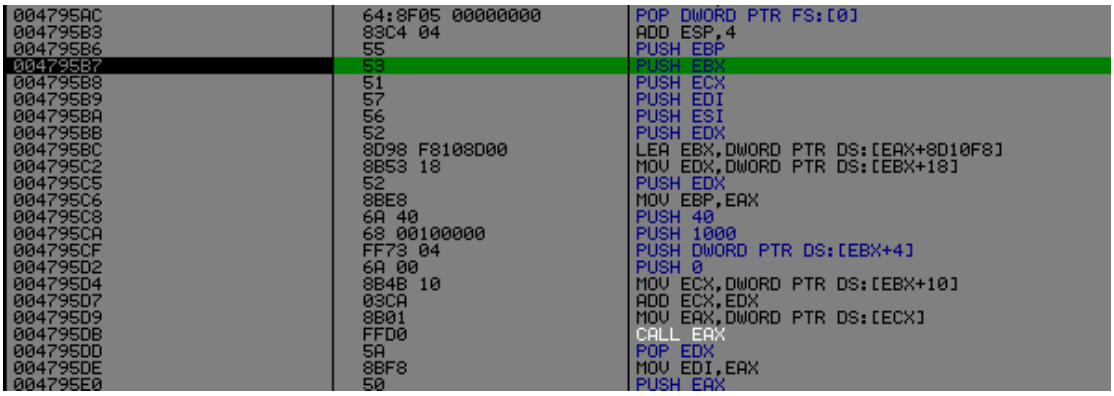
اینجا هم چیزی نیست،یکبار دیگر F9 را می زنم:



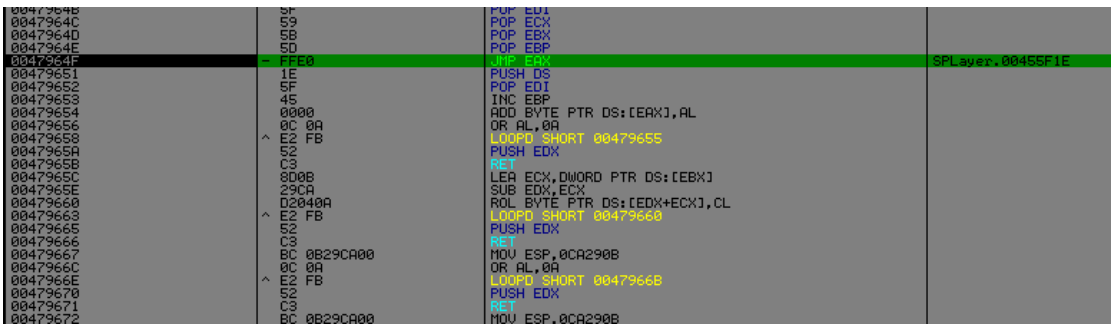
اینجا هم خبری نیست ☹ یکبار دیگر F9 را می زنم.



یکبار دیگر F9:



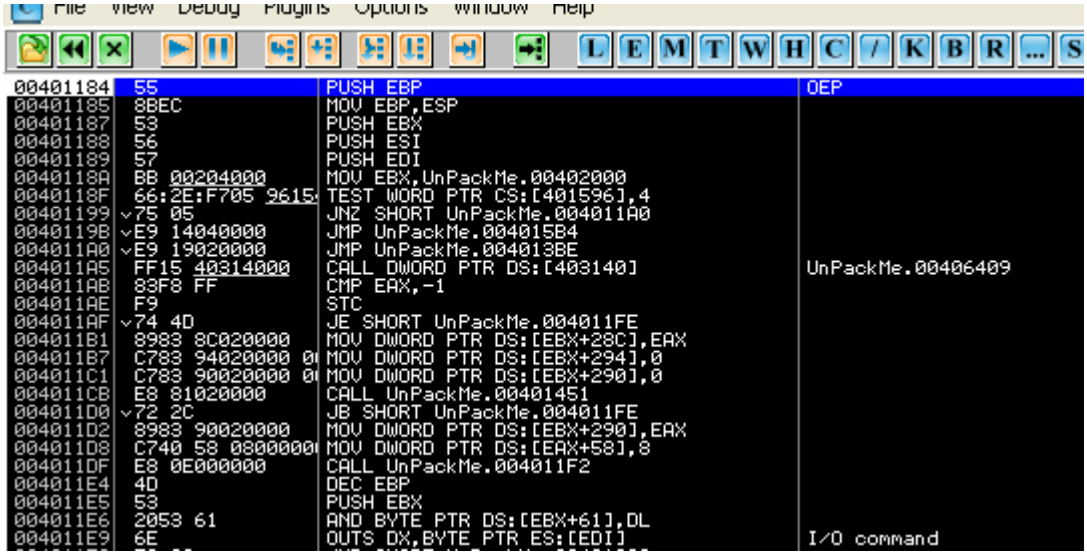
اینجا هم چیزی نیست.یکبار دیگر:



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

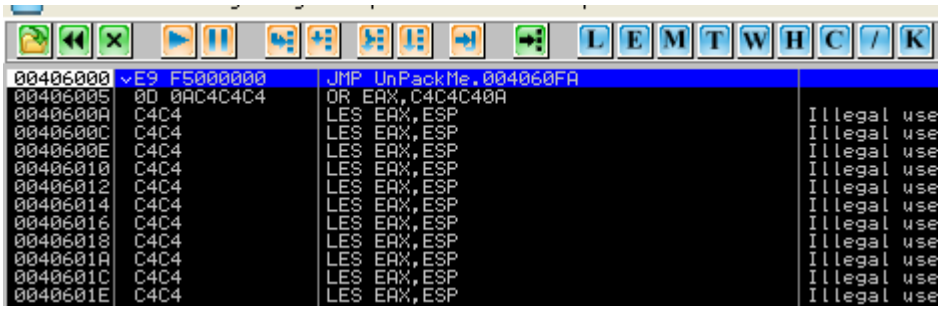
فایل مورد نظر را در OllyDBG باز کنید.برای یافتن OEP می توانیم از اسکریپت استفاده کنیم.در منوی PlugIns گزینه ODBGScript و سپس گزینه Run Script بزنید.اسکریپت مورد نظر(در Attachments موجود است) را انتخاب کنید:



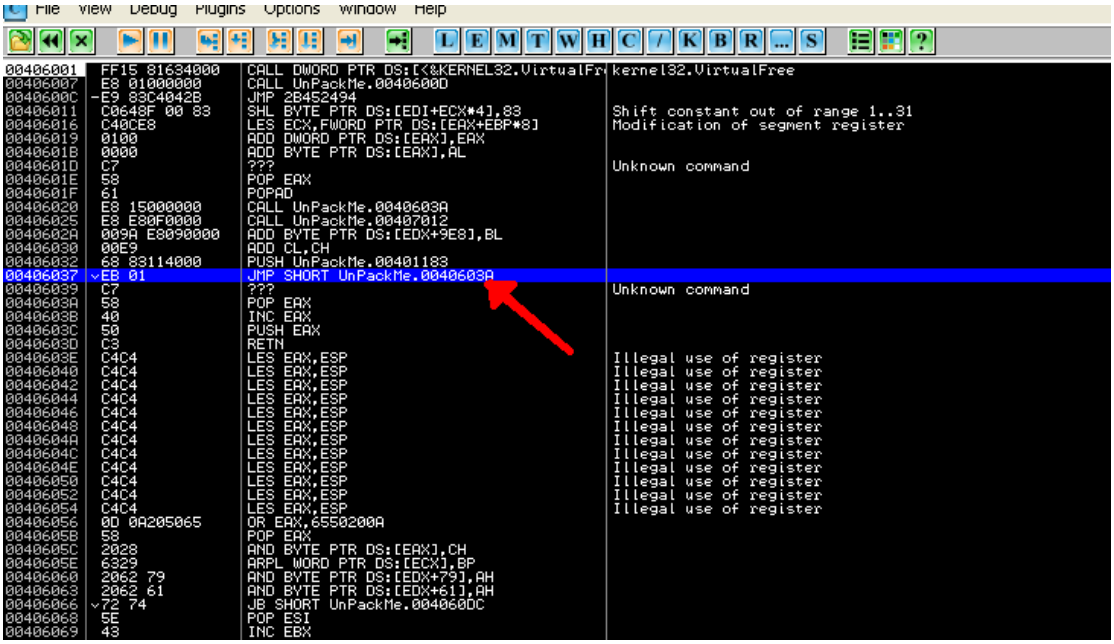
همانطور که دیدید اسکریپت توانسته OEP را با موفقیت به دست بیاورد. ☺

راه حل دیگر:

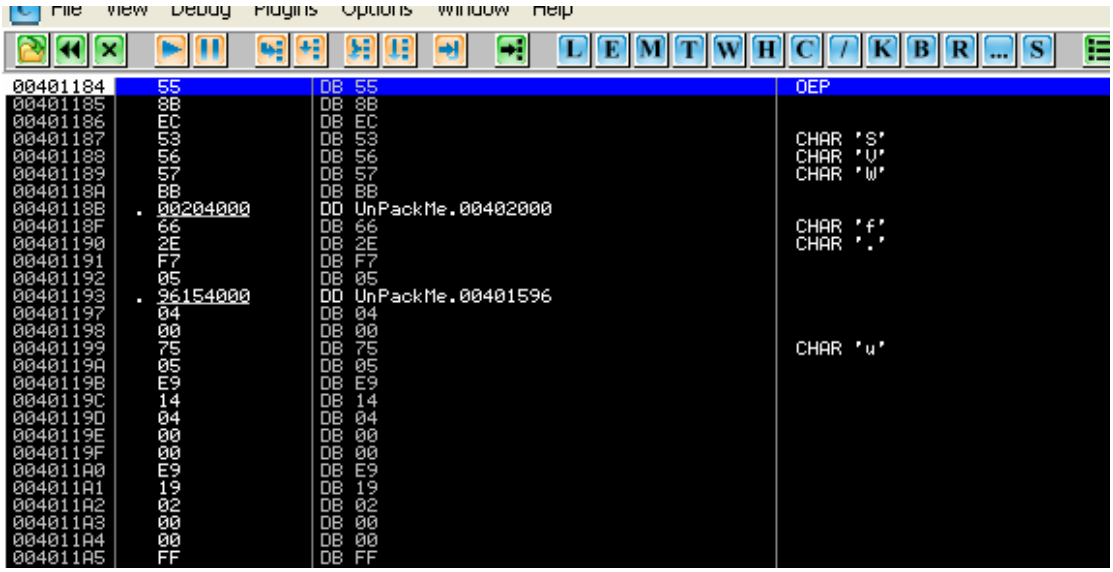
راه حل دیگر این است که به صورت دستی OEP را **با یک روش مخصوص PeX** پیدا کنیم.کاری که این اسکریپت می کند را ما به صورت دستی انجام می دهیم:



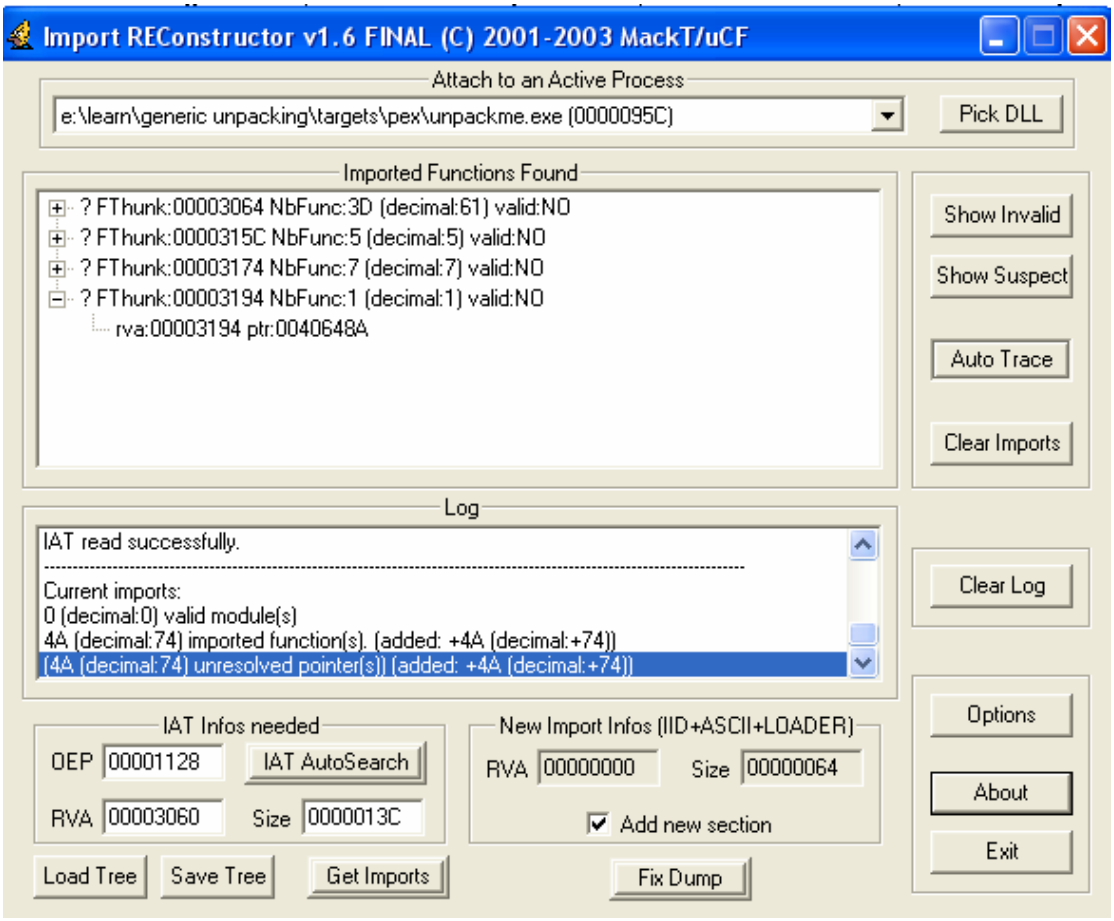
Entry Point ما خط 406000 هزار است.این مقدار را به اضافه یک می کنیم که می شود:406001
حالا کلید CTRL + G را می زنیم و به این آدرس می رویم: 406001
حالا بر روی این آدرس یک Hardware breakpoint on execution می گذاریم.(کلیک راست کرده گزینه Breakpoint و سپس گزینه Hardware on execution را می زنیم.) و بعد برنامه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



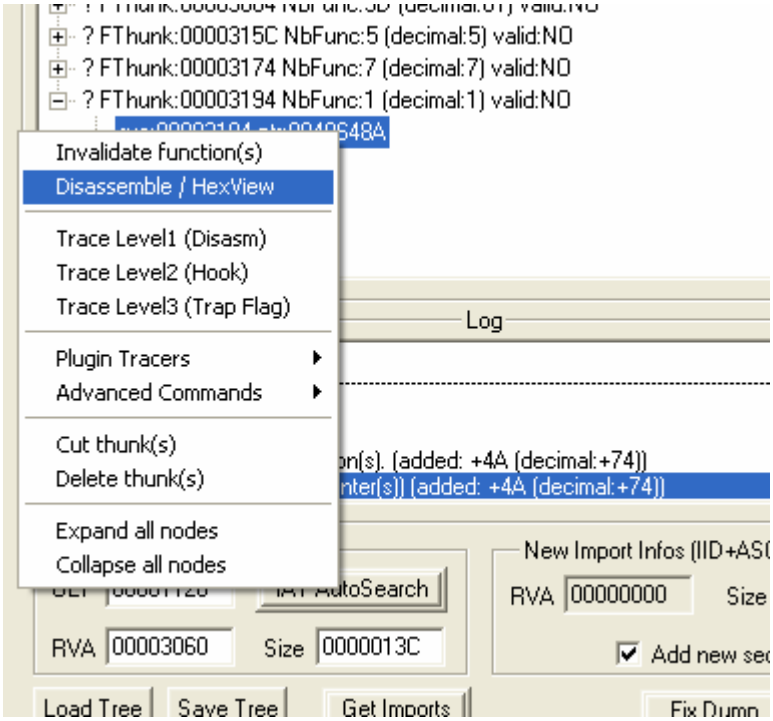
چند خط پایین تر یک دستور JMP (با فلش نشان داده شده) وجود دارد که بر روی آن Breakpoint می گذاریم و برنامه را با F9 اجرا می کنیم. برنامه ما در جایی که Breakpoint گذاشتیم متوقف می شود.حالا اگر چهار بار کلید F8 را بزنیم به OEP می رسیم:

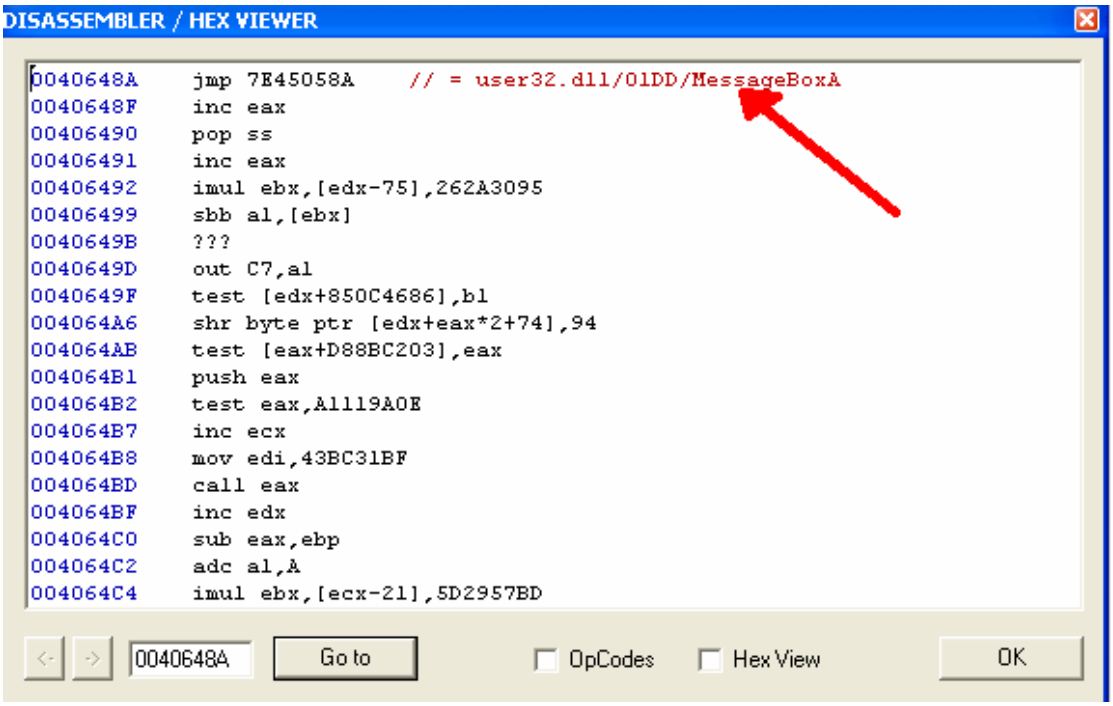


حالا از فایل دامپ بگیرید.و ImportREC را اجرا کنید.OEP را بر حسب RVA را به برنامه بدهید(من در تصاویر پایین OEP را اشتباه وارد کردم،باید وارد می کردم 1184...البته بعدا مشکل رو حل کردم،ولی وقتی این تصاویر رو گرفتم،OEP را اشتباه وارد کردم.شما اشتباه منو تکرار نکنید.) و ...:

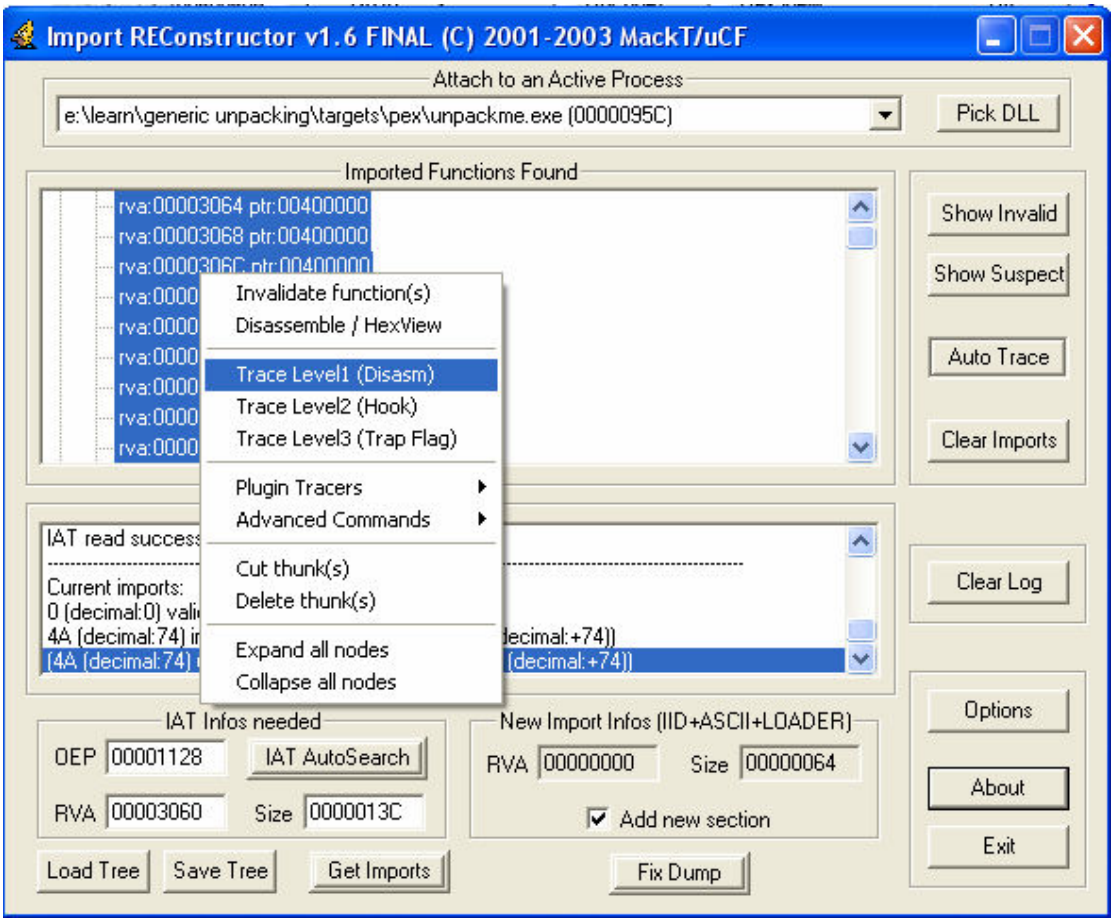


همانطور که می بینید تمامی توابع ما Invalid هستند.برخی از آنها واقعا Invalid هستند.اما برخی دیگر Redirect شده اند.همانند تصویر زیر بر روی آخرین تابع کلیک راست کرده و گزینه Disassemble/Hex View را بزنید:

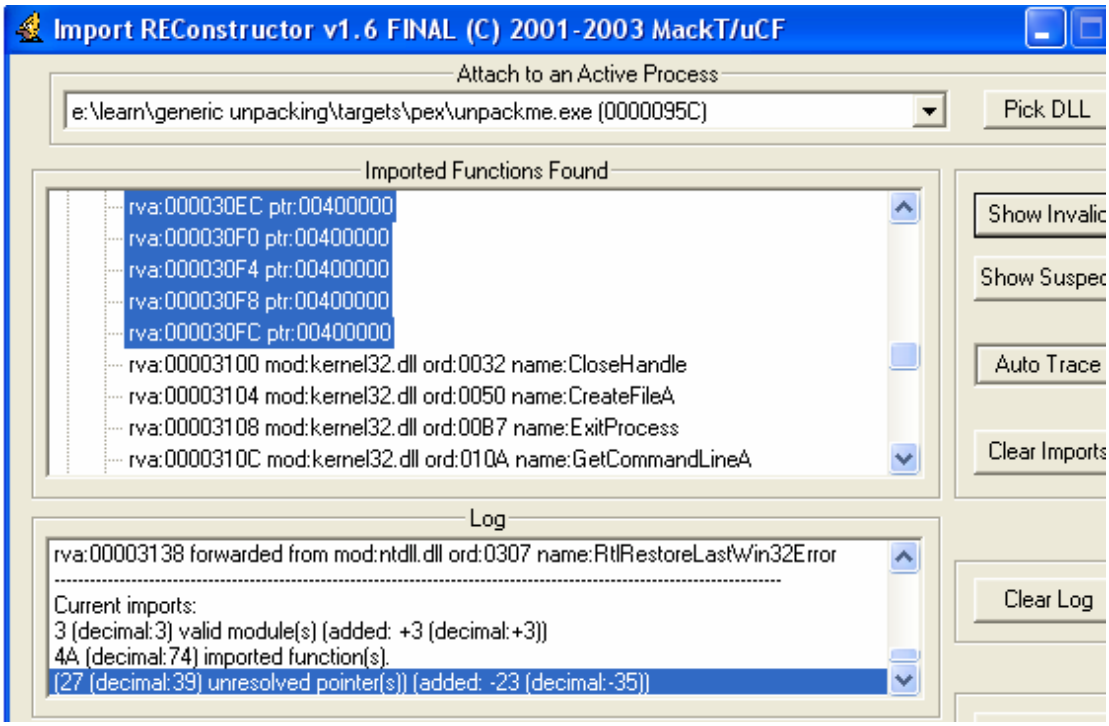




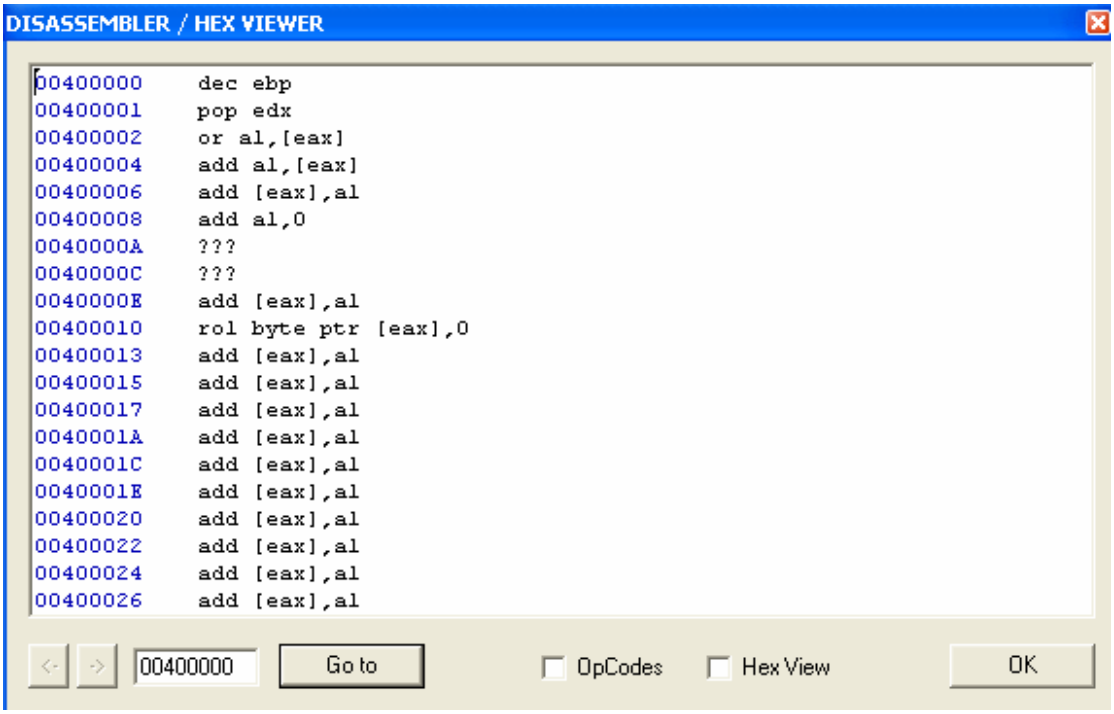
همانطور که می بینید این تابع ما به MessageBoxA می رود.پس این تابع Redirect شده است.خوب حالا چه طور توابع Redirect شده را فیکس کنیم؟
ما می توانیم از قابلیت ImportREC که می تواند توابعی که به سادگی Redirect شده اند را درست کند،استفاده کنیم.برای این کار گزینه Show Invalid را بزنید تا تمامی توابع مشخص شود،حالا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



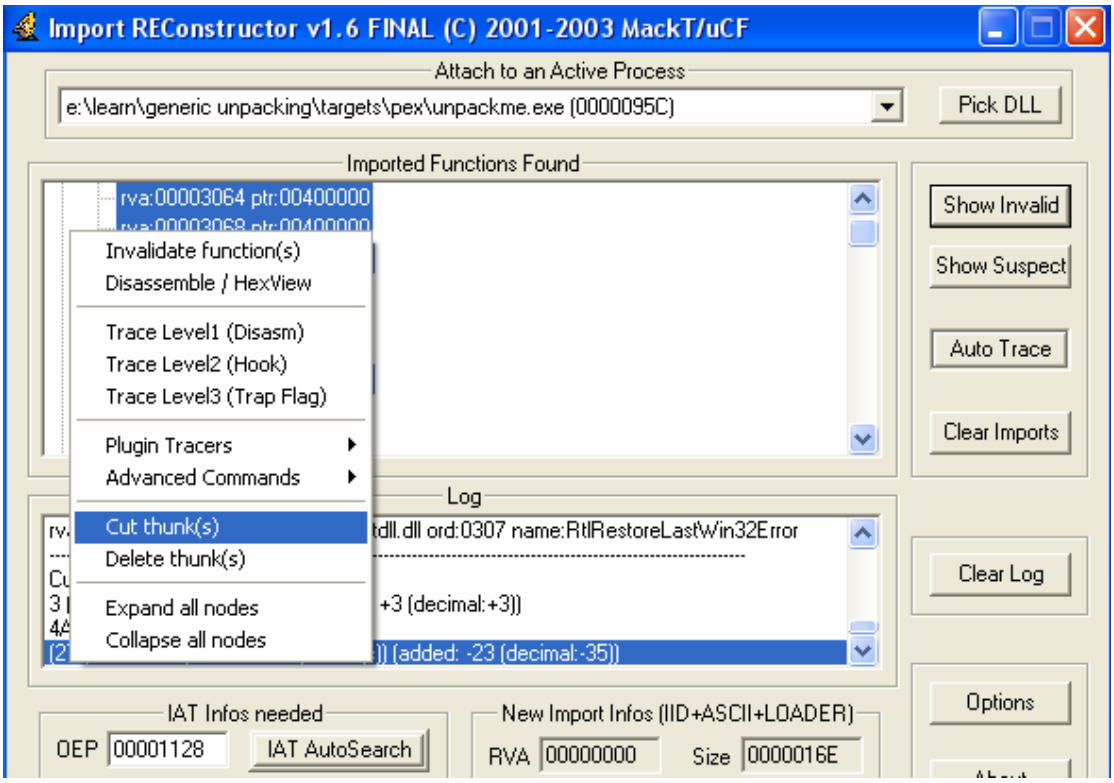
حالا دوباره گزینه Show Invalid را بزنید:



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:



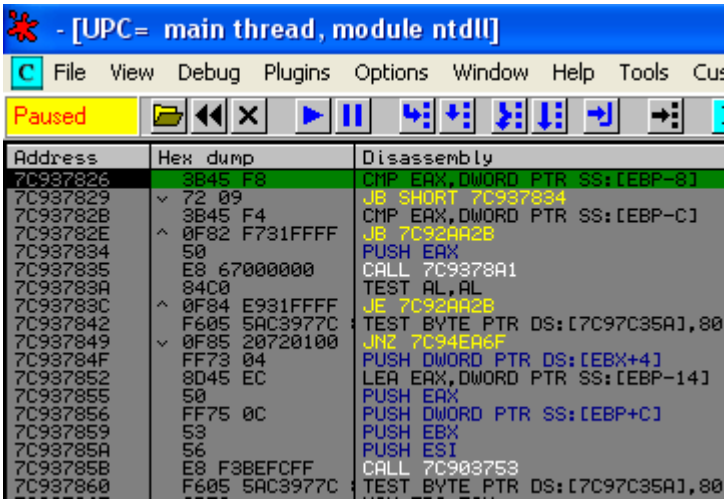
می بینید؟...کدهای این قسمت هیچ معنای خاصی ندارد...پس این تابع نمی تواند یک تابع واقعی باشد.پس می توانیم آنها را از بین ببریم،برای این کار کلیک راست کرده و گزینه Cut Thunk را می زنیم:



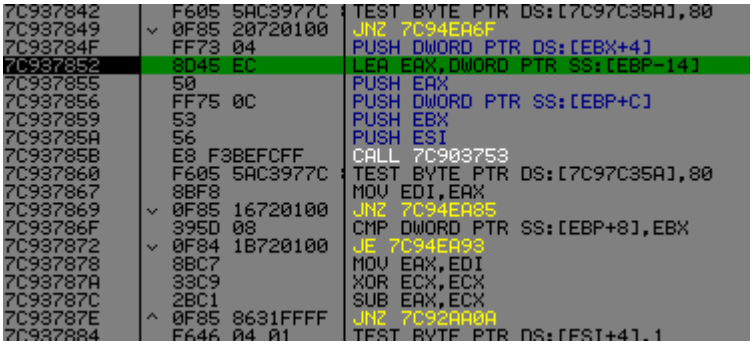
بعد از این هم فایل دامپ شده خودتان را فیکس می کنید.فایل دامپ شده بی هیچ مشکلی اجرا می شود.

PeCompact

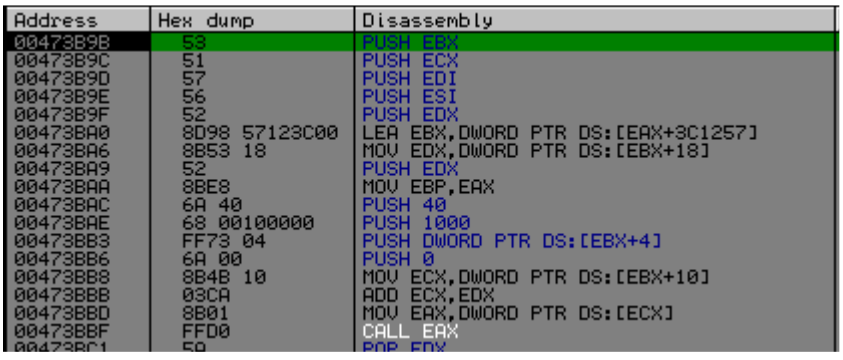
فایل مورد نظر را در OllyDBG باز کنید،برای یافتن OEP مـی تـوانیم از رجیـستر ESP اسـتفاده کنـیم.دو بـار کلیـد F8 را بزنیـد تـا مقـدار رجیستر ESP تغییر کند.بعد بر روي آن یک Hardware BP on access بگذارید و کلید F9 را بزنید:



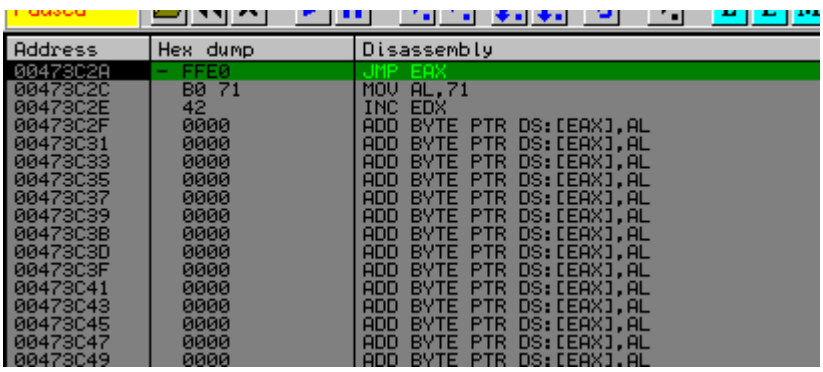
الان در فایل ntdll.dll هستیم،یکبار دیگر کلید F9 را بزنید:



هنوزم همانجا هستیم،یک F9 دیگر:



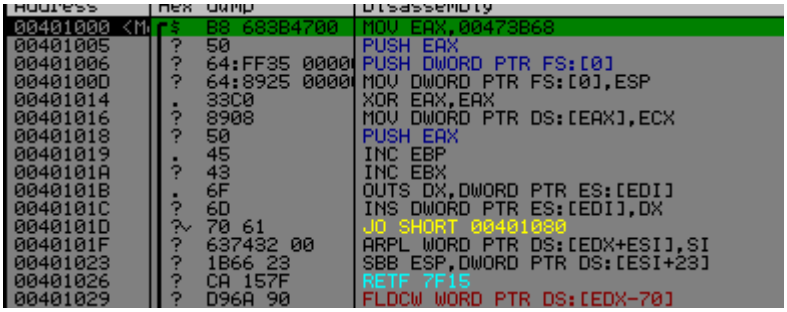
باز هم F9 :



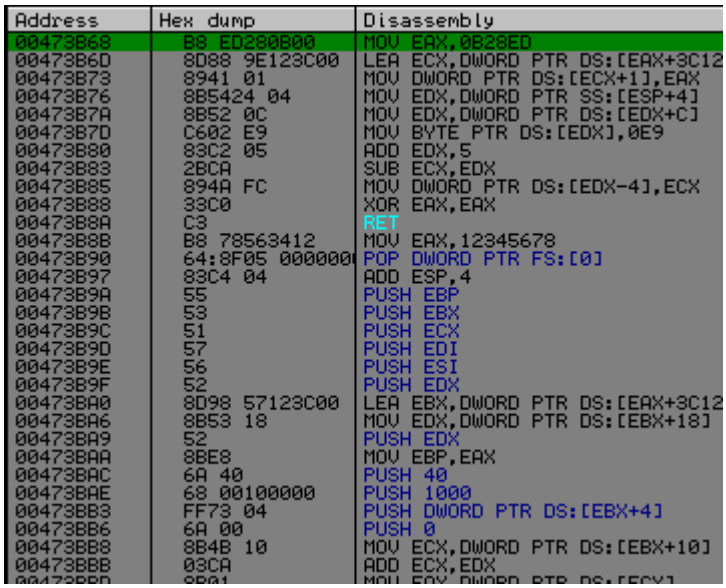
این پرش درست به OEP می رود.پس یکبار F8 بزنید تا به OEP برسید.

راه حل دیگر:

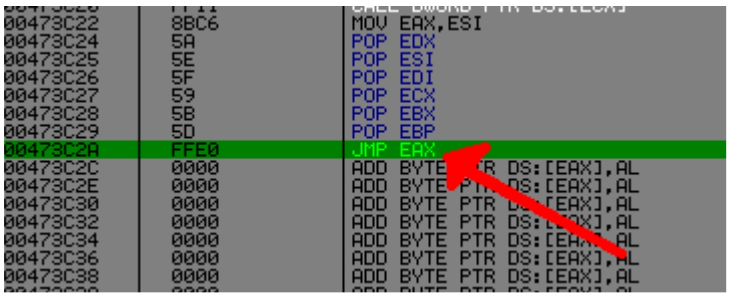
یک روش مخصوص PeCompact برای یافتن OEP وجود دارد.آن هم اینکه در اولین خط پکر یک دستور مثل MOV EAX,xxxxxxxx قرار دارد.اگر ما به آدرس xxxxxxxx برویم.چند خط پایینتر دستور JMP EAX که به OEP می رود را پیدا می کنیم.



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



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

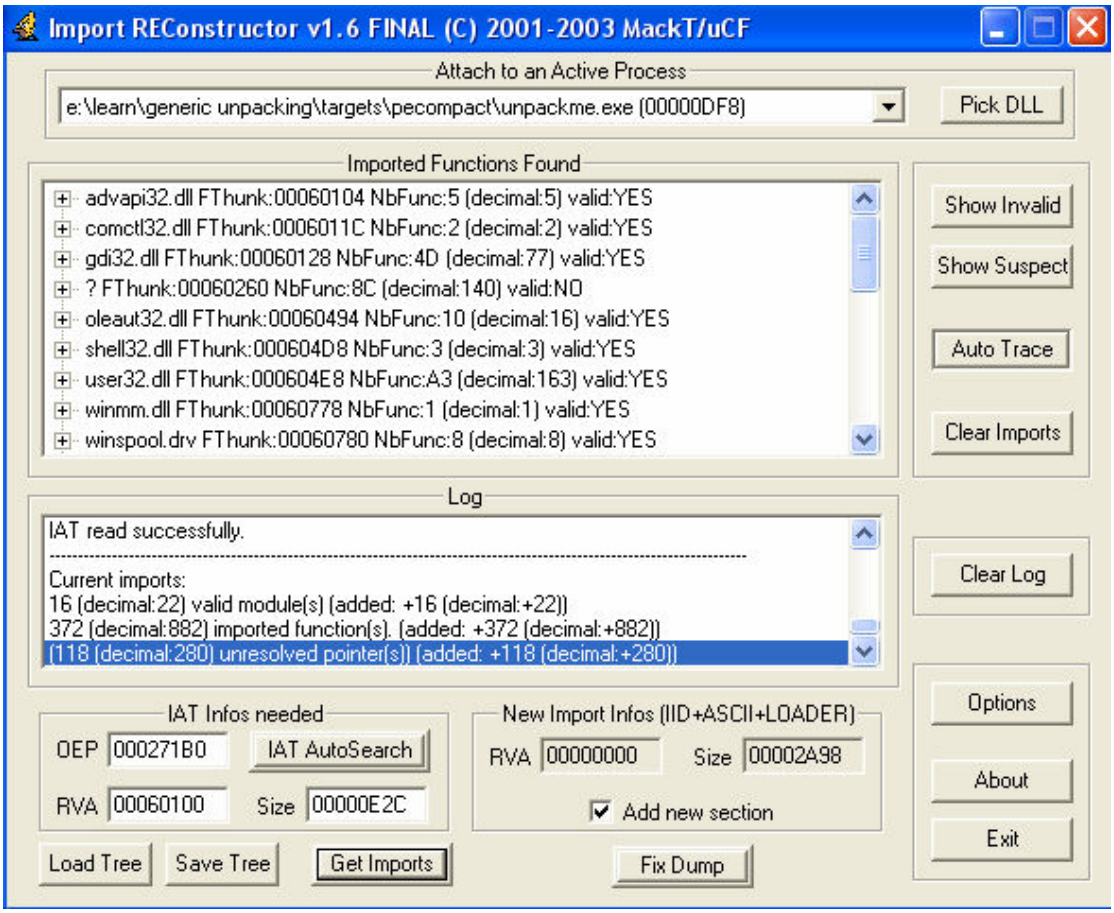


این دستور ما را به OEP می برد.پس بر روی آن BP بگذارید و برنامه را با F9 اجرا کنید:

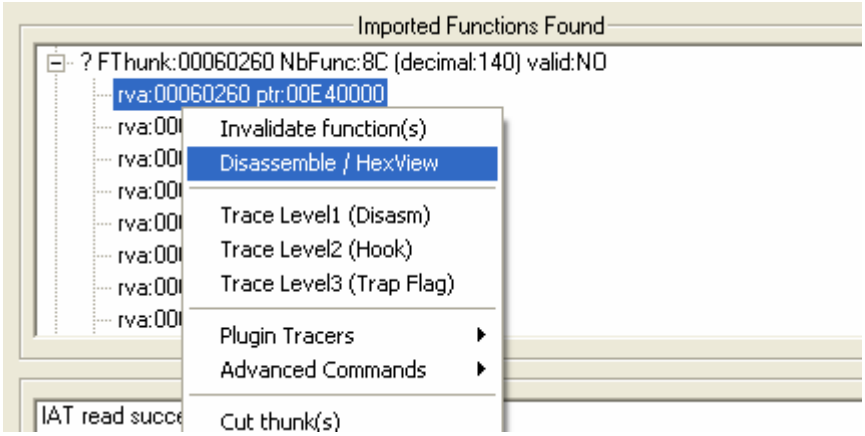
```
FF11    CALL DWORD PTR DS:[ECX]
8BC6    MOV EAX,ESI
5A      POP EDX
5E      POP ESI
5F      POP EDI
59      POP ECX
5B      POP EBX
5D      POP EBP
FFE0    JMP EAX                         UnPackMe.004271B0
B0 71   MOV AL,71
42      INC EDX
0000    ADD BYTE PTR DS:[EAX],AL
0000    ADD BYTE PTR DS:[EAX],AL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

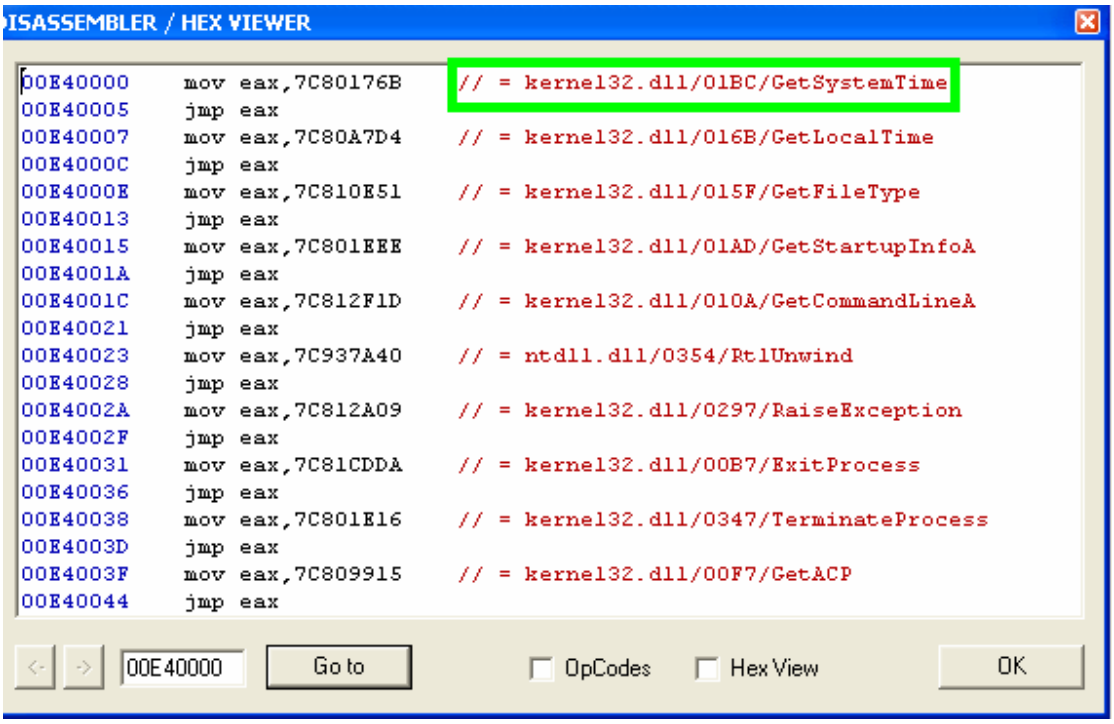
حالا با یک F8 به OEP می رسیم ☺

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

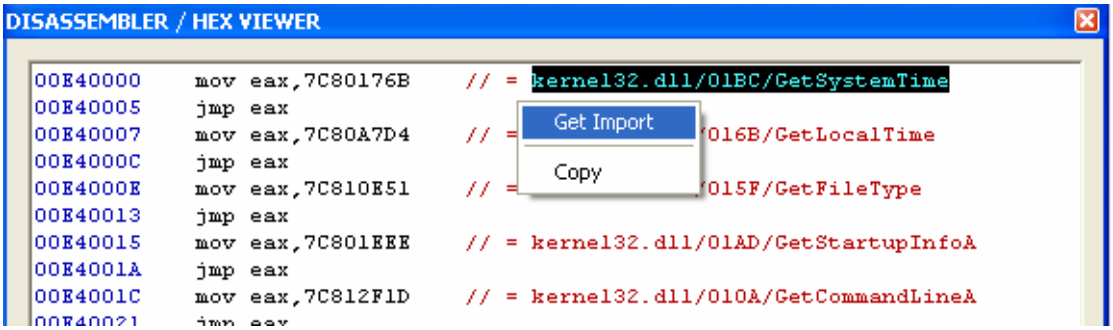


همانطور که می بینید تعدادی از توابع ما Invalid است.من یک تابع Invalid را درست می کنم،بقیه اش با شما...یکی از توابع Invalid را انتخاب کرده،کلیک راست کرده و گزینه Disassemble/hex view را بزنید:





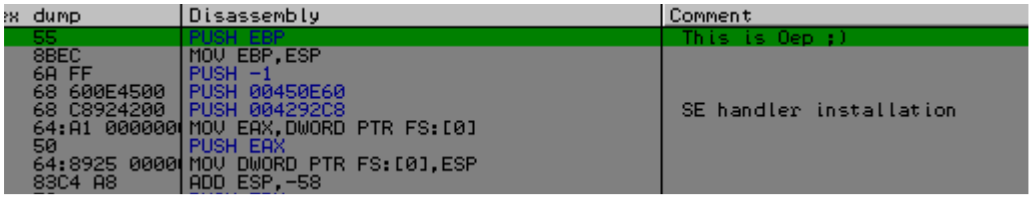
می بینید؟این آدرس از IAT در واقع به تابع GetSystemTime می رود.پس همانند تصویر کلیک راست کرده و گزینه GetImport را بزنید:



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 قابلیتی وجود دارد که می تواند تابع GetProceAddress 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.

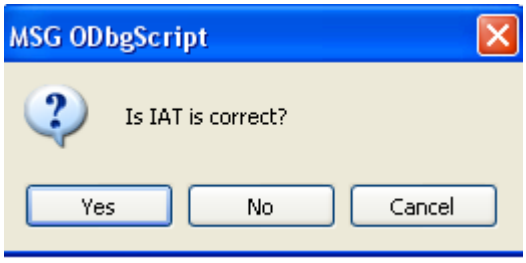
راه حل دیگر:

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 OllyDBG 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:

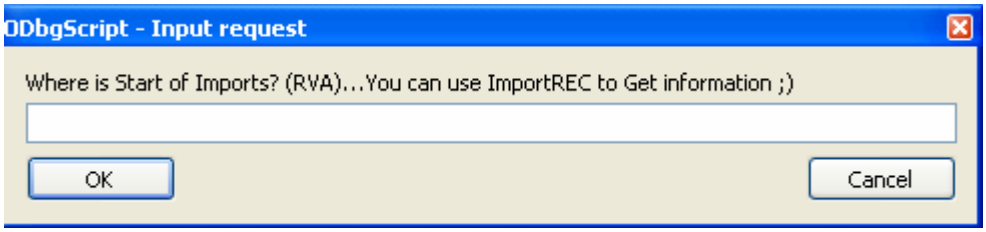


خوب،پس اسکریپت OEP را درست پیدا کرده.(اگر درست پیدا نشد،کلیدهای ALT + O را بزنید و در قسمت Exceptions تیک تمامی گزینه ها را بزنید.همچنین تمامی BP ها را پاک کنید.)

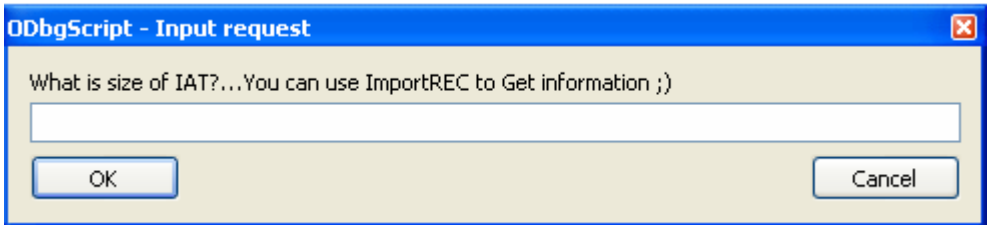
اسکریپت از شما سوال می کند:



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

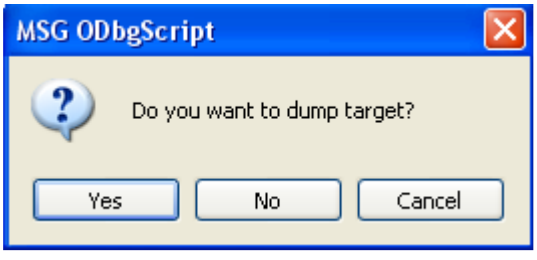


اسکریپت از ما می پرسد محل شروع IAT کجاست؟...همانطور که در ImportREC دیدیم،محل شروع بر حسب RVA برابر 60100 بوده.پس این مقدار را وارد کنید و کلید OK را بزنید:

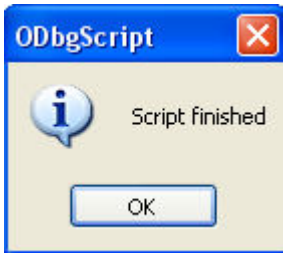


حالا سایز IAT چقدر بوده؟...ImportREC می گوید: E2C ،پس ما هم همین مقدار را وارد می کنیم:

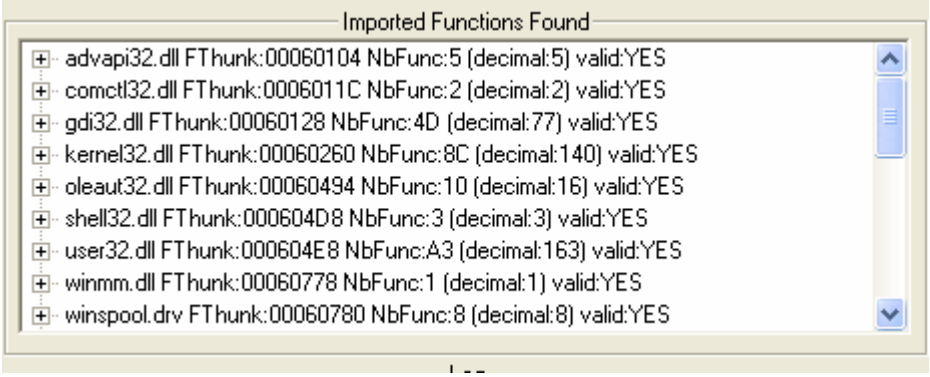
کلید OK را بزنید:



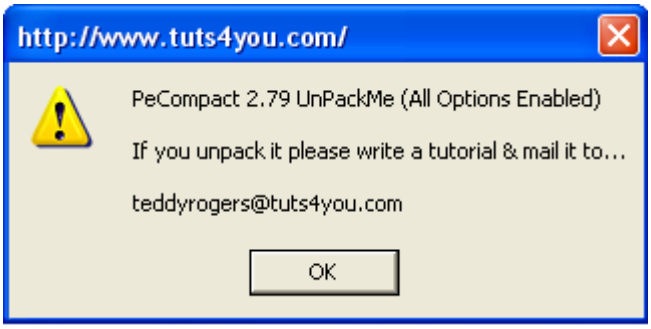
اگر می خواهید از فایل دامپ بگیرد،کلید Yes و در غیر این صورت کلید No را بزنید:



حالا دوباره توابع را در ImportREC وارد کنید:

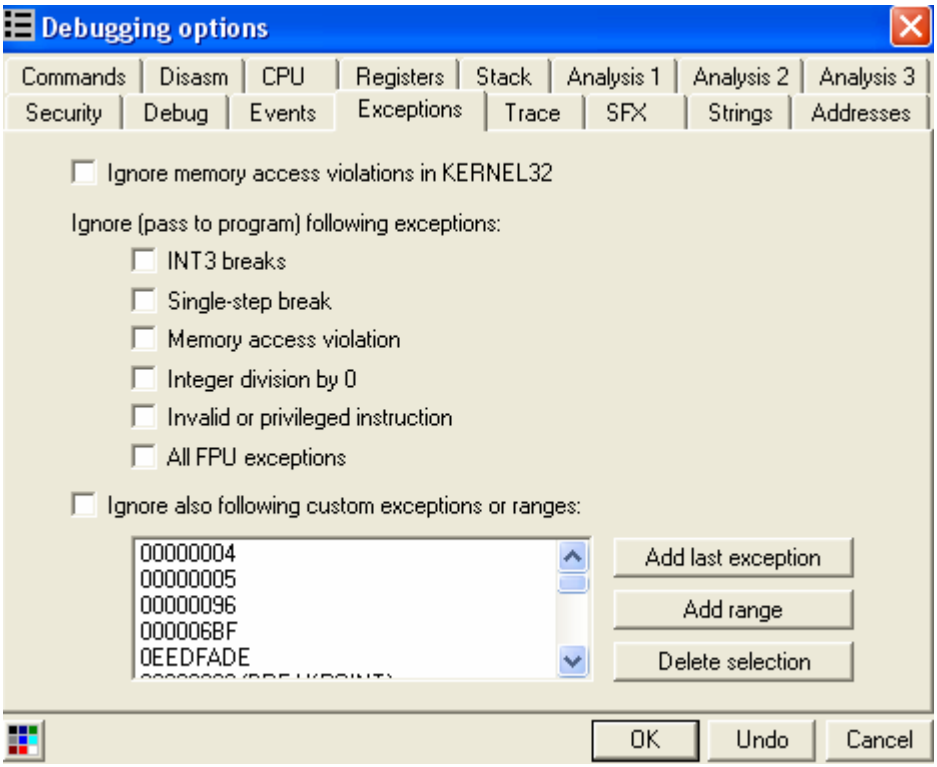


این بار تمامی توابع درست شده اند.حالا فایل دامپ خود را فیکس کنید:

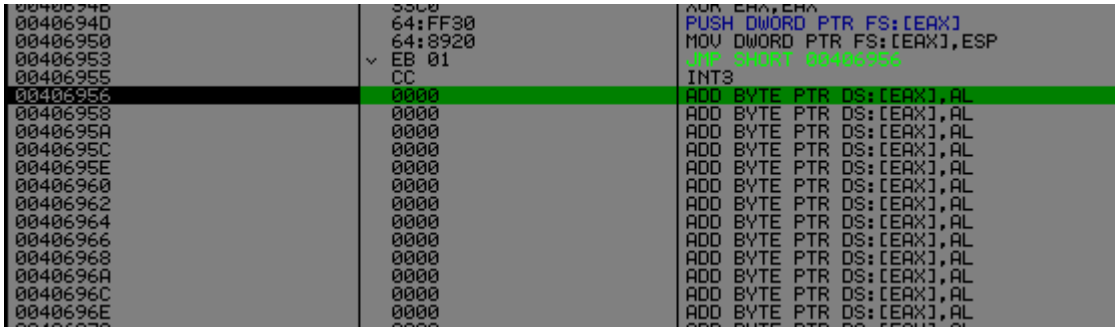


ExeShield

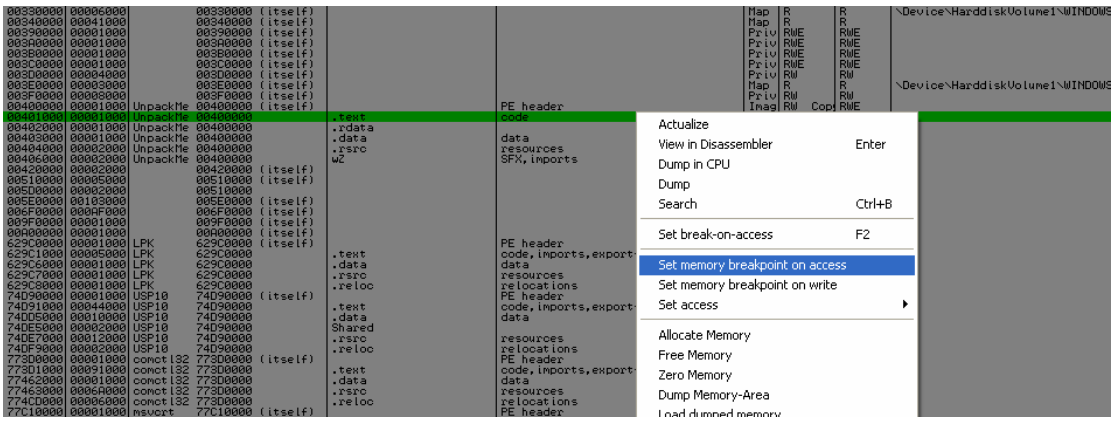
برای یافتن OEP در ExeShield هم می توانم به راحتی از Memory Breakpoint استفاده کنم.
اول باید ببینم که در کجا باید Memory Breakpoint بگذارم؟...خوب،اگه خطایی در برنامه اتفاق بیفتد،می توانم بعد از اینکه آخرین خطا در برنامه اتفاق افتاد،آنجا Memory Breakpoint بگذارم.پس اول باید به OllyDBG بگوییم اگر خطایی در برنامه اتفاق افتاد،خطا را رد نکن و در همانجا متوقف شو.در منوی Options گزینه Debugging Options را بزنید و وارد قسمت Exceptions شوید:



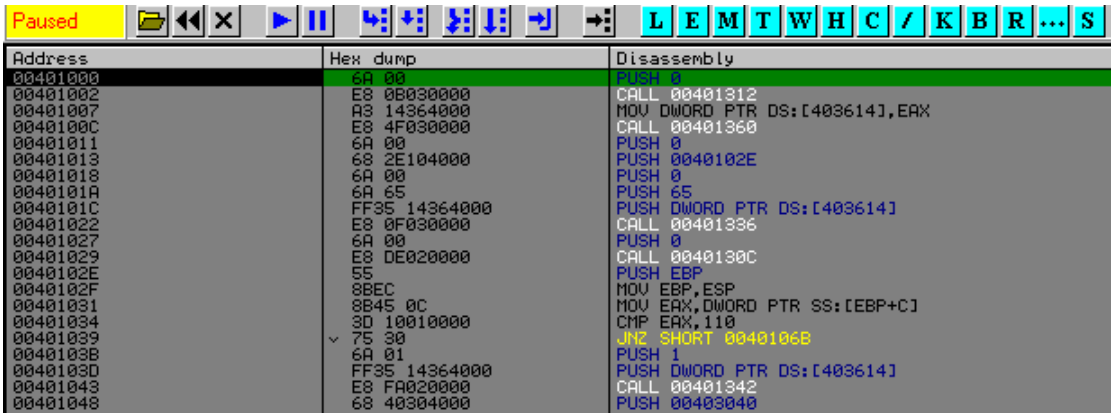
همانند تصویر تمامی تیک ها را بردارید تا برنامه در موقعی که خطا اتفاق افتاد،متوقف شود.حالا کلید OK را بزنید و برنامه را با F9 اجرا کنید:



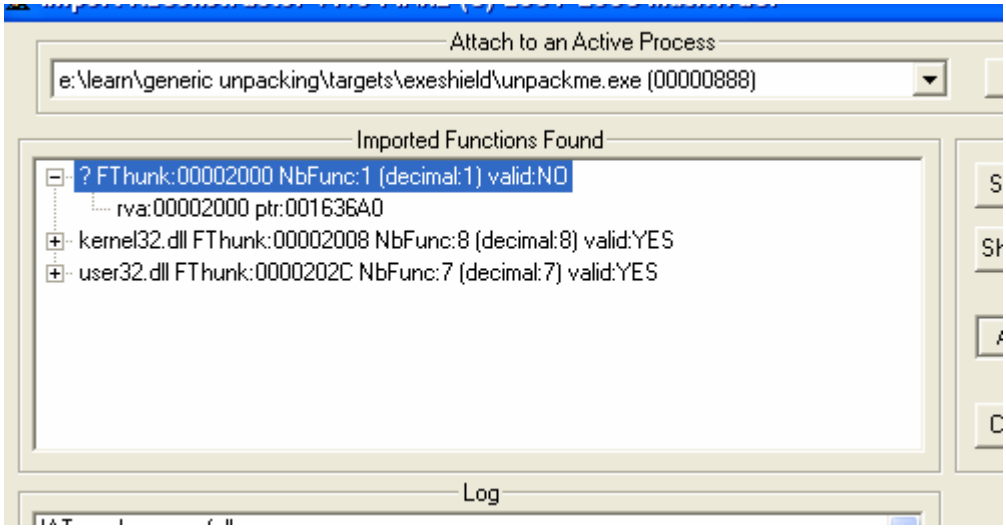
همانطور که می بینید برنامه ما در اینجا آدرسی را فرا می خواند که اصلا وجود ندارد به همین دلیل دچار خطا می شود.خوب حالا ما می توانیم بر روی سکشن اصلی برنامه Memory Breakpoint بگذاریم.کلید های ALT + M را بزنید و همانند تصویر یک Memory Breakpoint بر روی سکشن text. بگذارید:



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



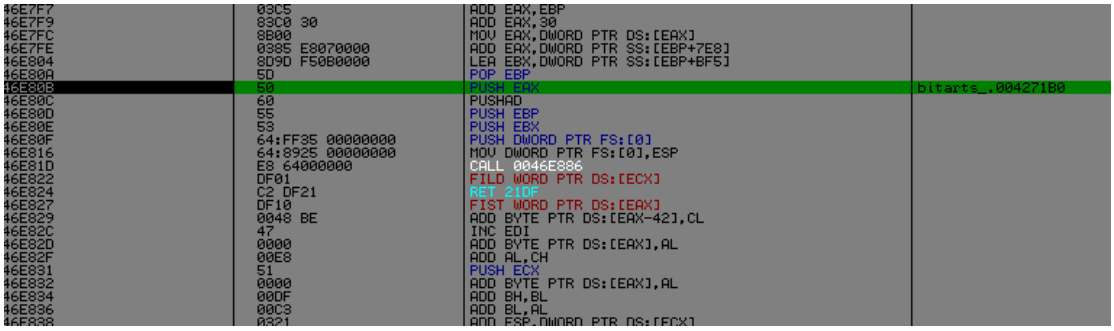
همانطور که می بینید ما درست در OEP فرود آمدیم ☺
حالا از فایل دامپ بگیرید و ImportREC را اجرا کرده،OEP را به برنامه بدهید و کارهای لازم را انجام دهید:



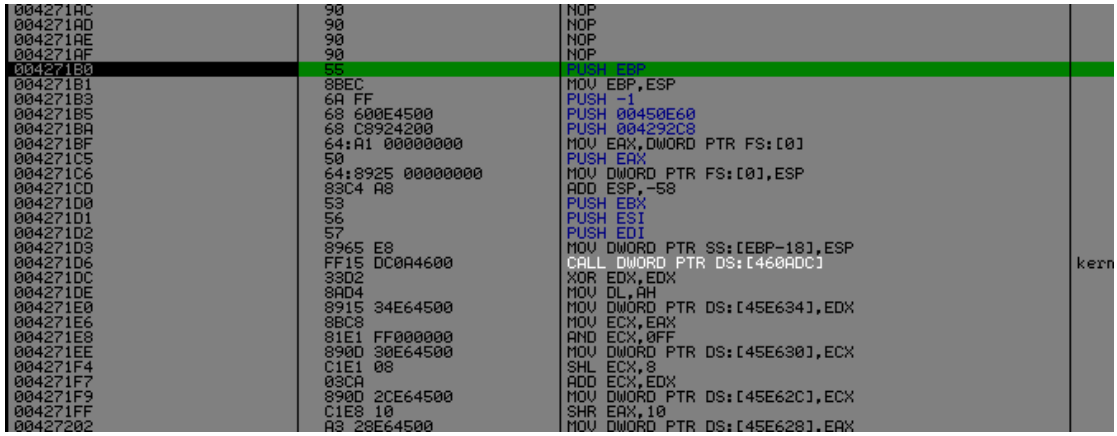
همانطور که می بینید یکی از توابع ما Redirect شده،ولی مشکلی نیست.چون کافیست بر روی آن کلیک راست کرده و گزینه Trace Level1 (disasm) را بزنیم تا تابع اصلی ما مشخص شود ☺
حالا هم فایل دامپ شده خودمان را فیکس می کنیم.فایل آنپک شده ما بی هیچ مشکلی اجرا می شود.

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:



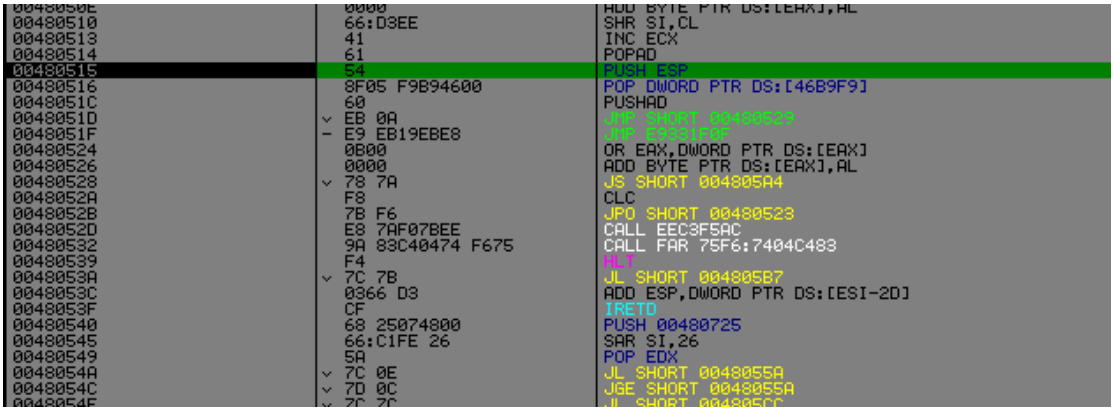
یکبار دیگر F9 را می زنم:



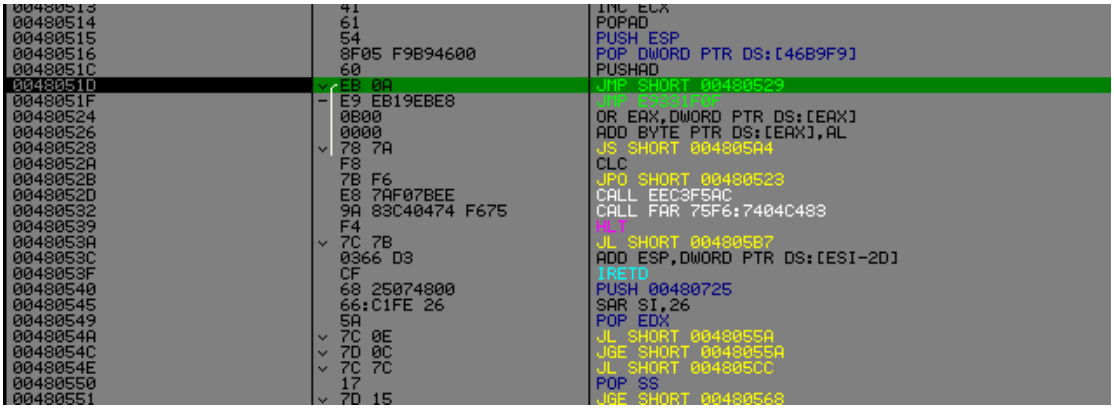
اینبار به OEP رسیدیم ☺
بعد از این دیگر مشکلی نیست،دامپ می گیرید و فیکس می کنید.

Acprotect

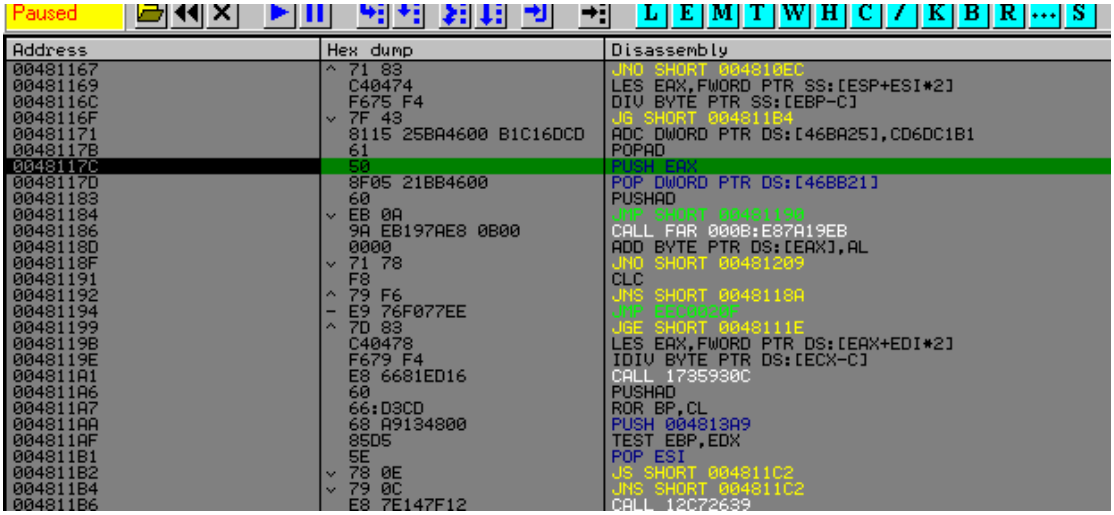
OEP در ورژن دوم Acprotect به راحتی پیدا می شود.کافیست بعد از دستور PushAD روی مقدار رجیستر ESP یک Hardware Breakpoint بگذاریم و بعد از چند بار زدن کلید F9 به پرشی می رسیم که ما را به OEP می برد. ☺
برنامه را در OllyDBG اجرا کنید،۵ بار کلید F8 را بزنید تا دستور PushAD اجرا شود و بعد بر روی مقدار رجیستر ESP یک Hardware breakpoint on access را بگذارید و بعد کلید F9 را بزنید:



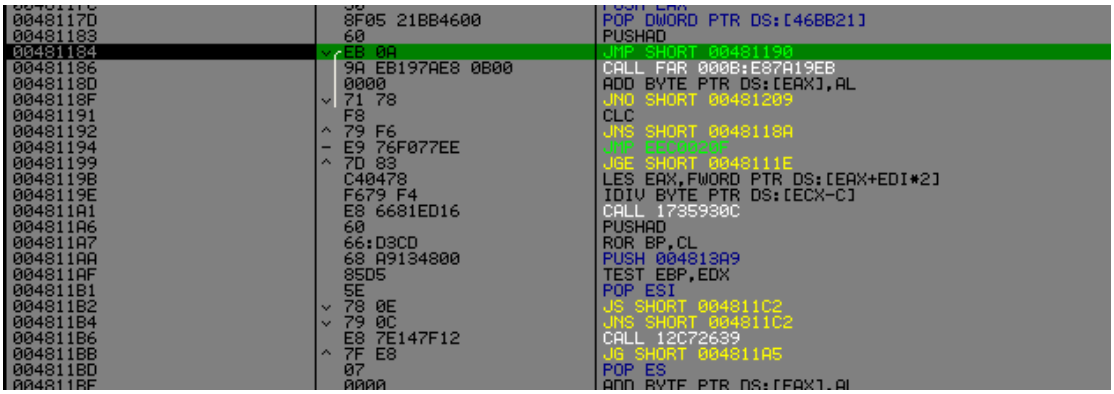
اینجا اتفاق خاصی نمی افتد،یکبار دیگر کلید F9 را بزنید:



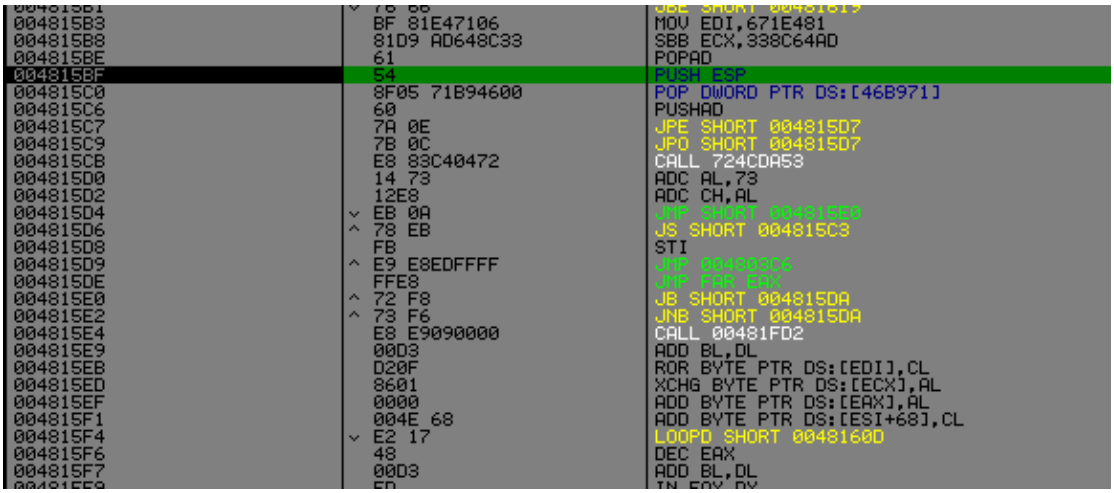
یکبار دیگر هم F9 بزنید:



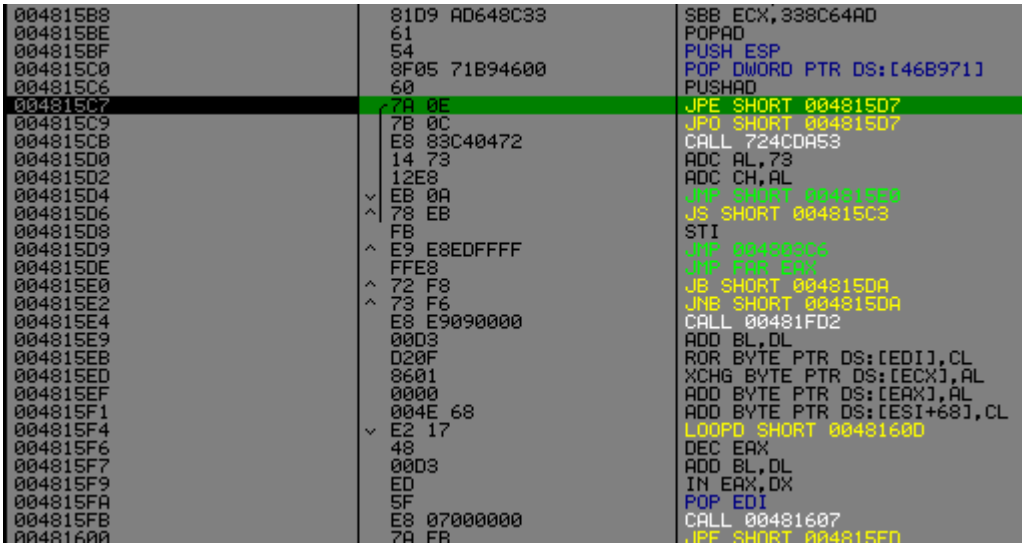
یک F9 دیگر:



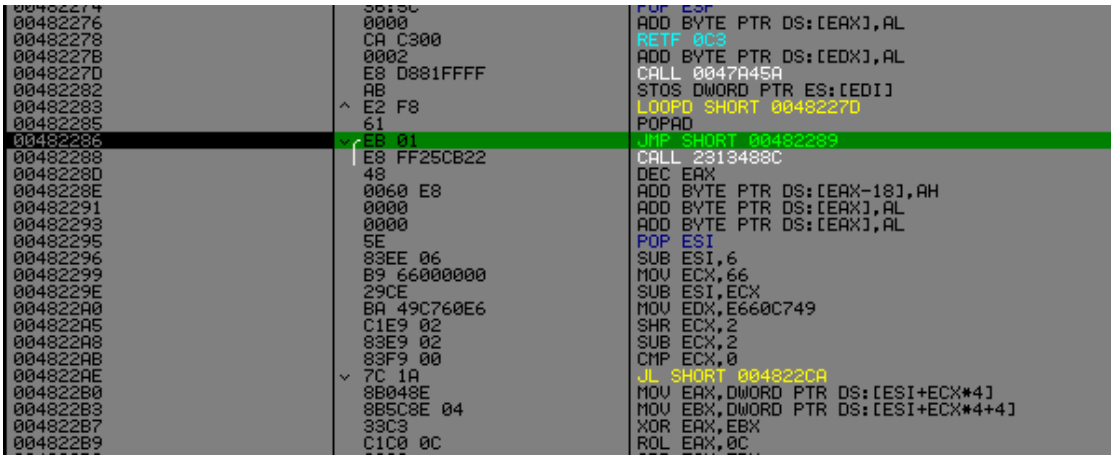
باز هم F9:



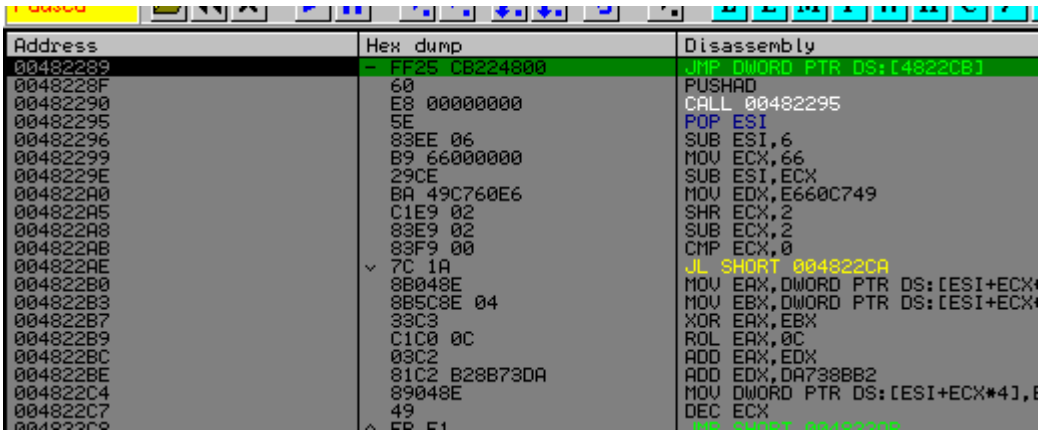
: F9



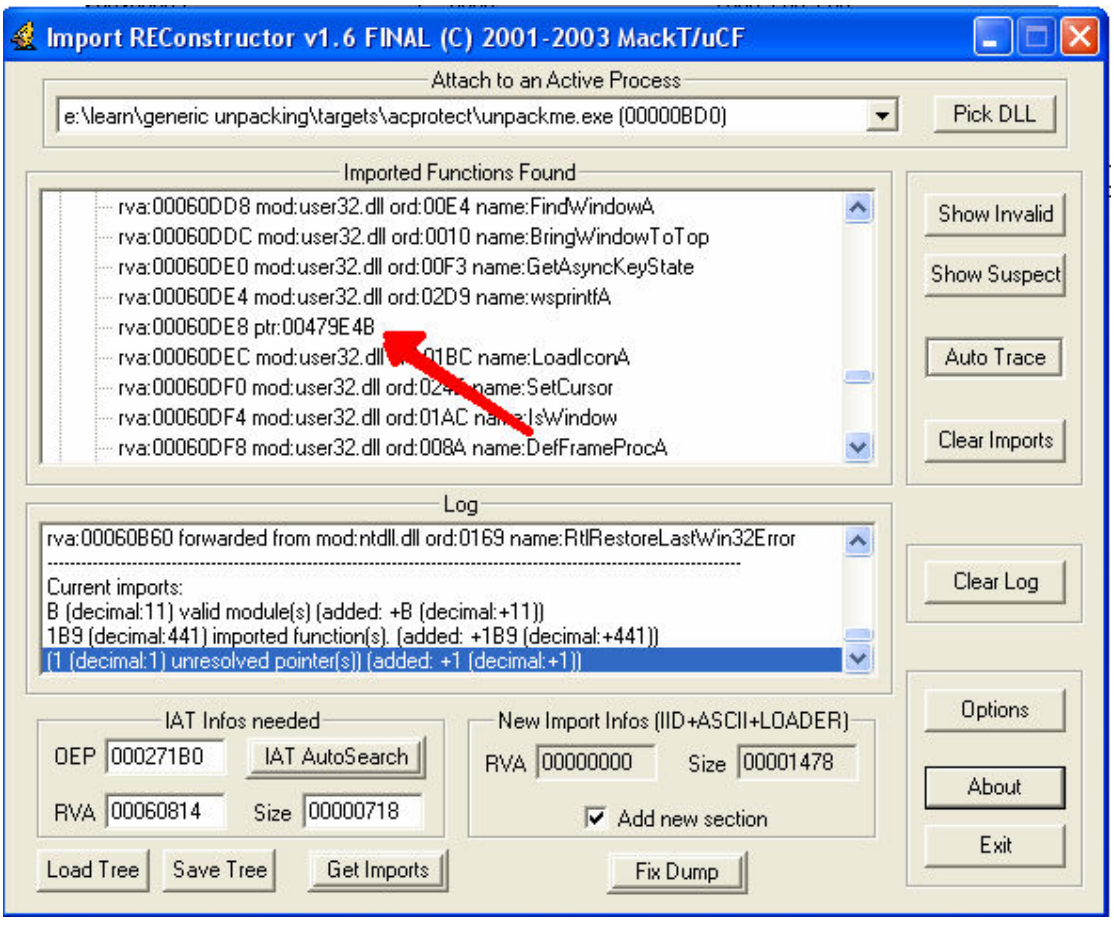
یک F9 دیگر:



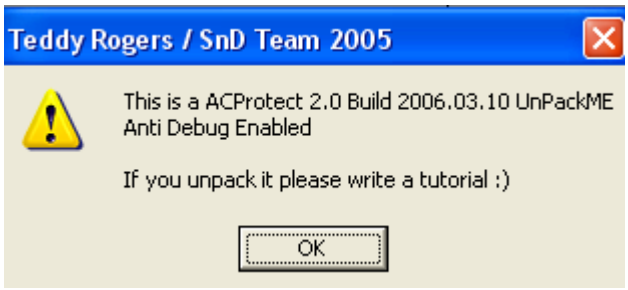
آها،دیگر تقریبا به OEP رسیدیم.یکبار F8 را بزنید:



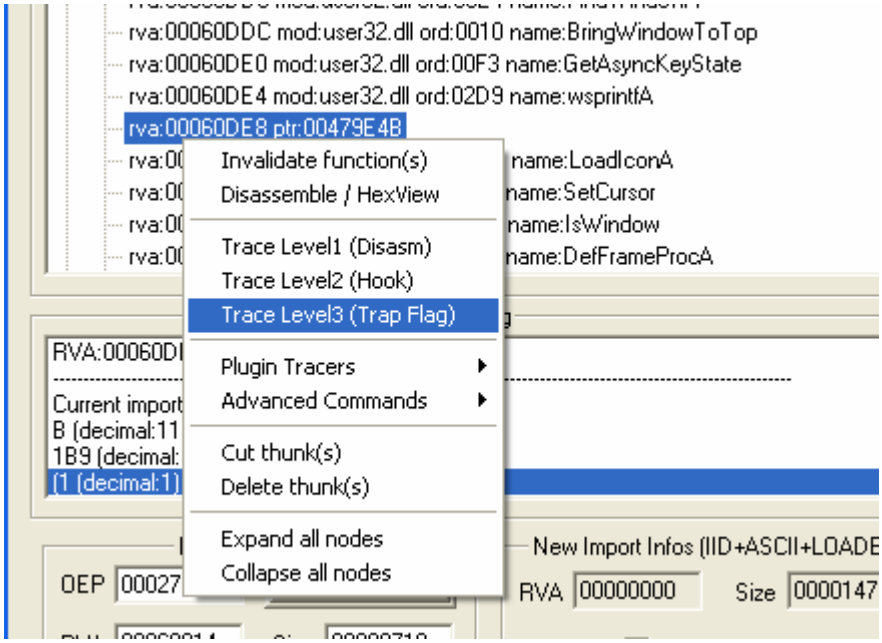
این پرش مستقیم به OEP می رود.یک بار F8 را بزنید تا به OEP برسید و بعد از فایل دامپ بگیرید.
حالا ImportREC را باز کنید.OEP را به برنامه بدهید و سایر مراحل لازم را انجام دهید:



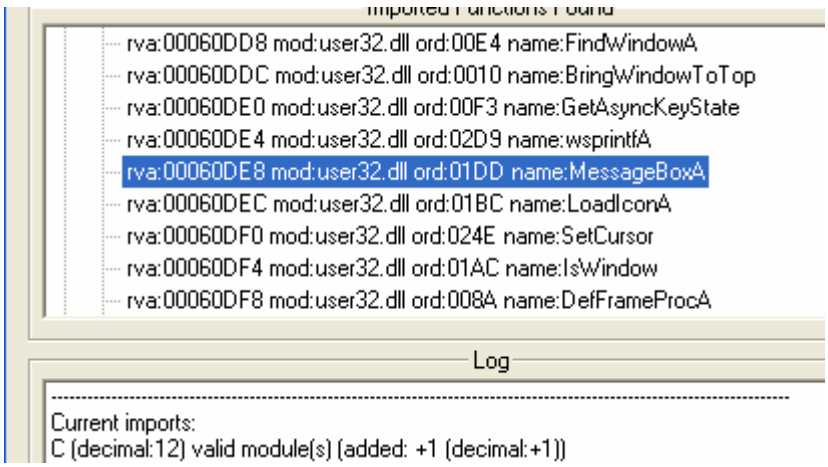
همانطور که می بینید یکی از توابع ما Redirect شده است.برای یافتن تابع اصلی راه حل های زیادی وجود دارد.(حتی من اولین بار توانستم حدس بزنم که تابع اصلی چی ممکن است باشد)من در اینجا می خواهم از قابلیت Trap Flag در ImportREC استفاده کنیم.ImportREC با تزریق یک Thread در پروسه ما متوجه می شود که این آدرس در 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 Trace Level3 (Trap Flag) 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 توانستید 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 مطمئنا باید MessageBoxA باشد.)

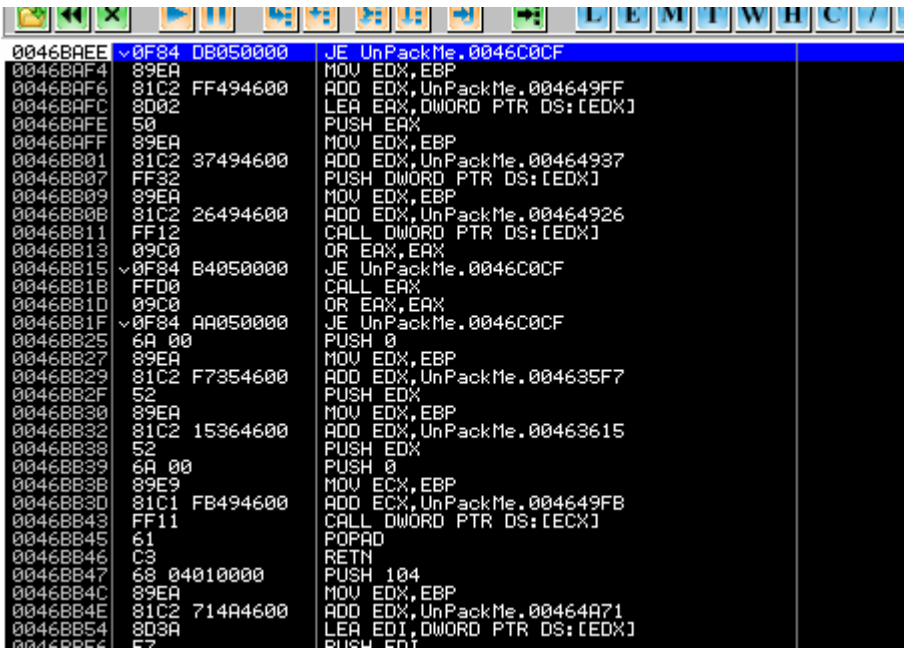
حالا فایل دامپ شده خود را فیکس کنید،می بینید که فایل آنپک شده بی هیچ مشکلی اجرا می شود.

NoName Packer

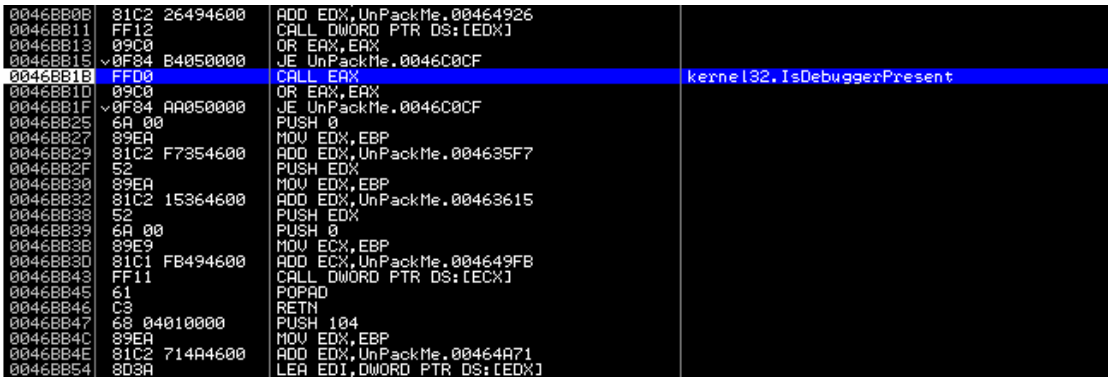
فایل را در OllyDBG بدون پلاگین باز کنید،کلید F9 را بزنید تا برنامه اجرا شود:

می بینید؟...این پیغام 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:

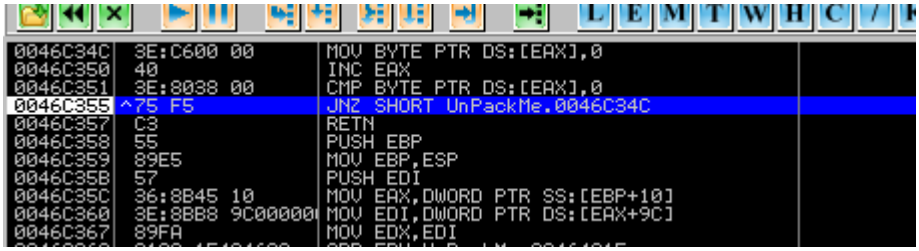
پس این پیغام در خط 0046BB43 اجرا می شود.چند خط بالاتر از این خط را Hardware Breakpoint بگذارید تا کدهای این قسمت را با دقت بیشتری بررسی کنیم.ممکن است سوال کنید چرا software breakpoint(کلید F2) نگذاشتم؟...چون پکرها معمولا کدهای خودشون رو تغییر می دهند(Self-modifying) به همین دلیل اگر ما یک Software breakpoint بگذاریم.چون OllyDBG دستور INT3 را اضافه می کند.پکر در عملیات Self-modifying دچار مشکل شده و برنامه Don't send 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:



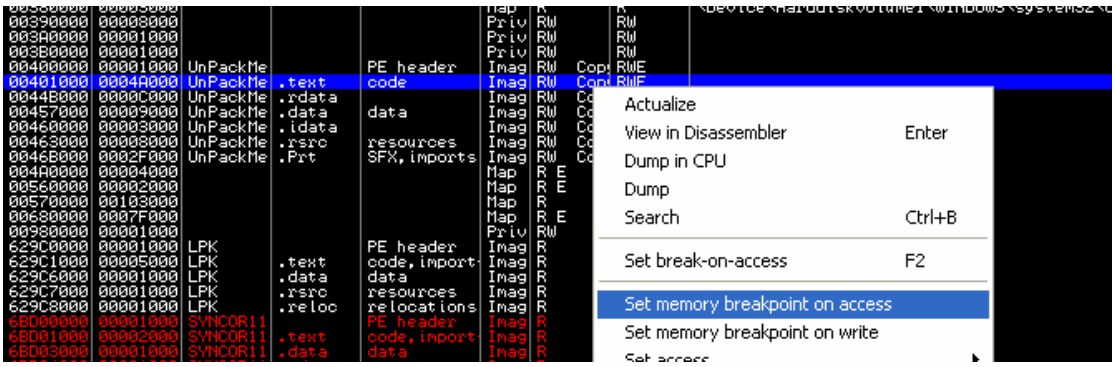
چند بار کلید F8 را بزنید تا به این خط برسید:



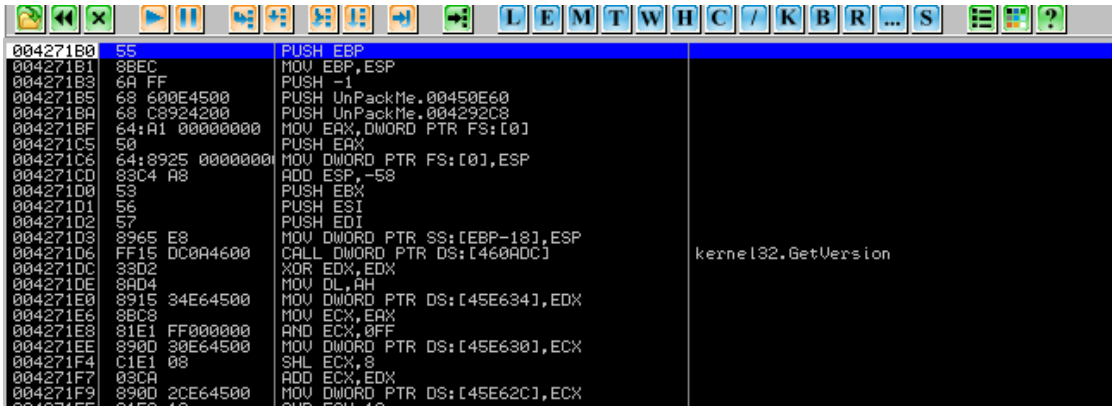
در این خط تابع IsDebuggerPresent اجرا می شود.پس این تابع همان آنتی دیباگ پکر ما بود.خوب...این بار از پلاگین برای رد کردن این آنتی دیباگ از پلاگین استفاده می کنیم...پلاگین HideDBG.dll را در پوشه Plugin های OllyDBG کپی نمایید و OllyDBG را یکبار ببندید و دوباره باز کنید،حالا در منوی Plugins گزینه HideDBG و سپس گزینه HideAll را بزنید.و بعد کلید F9 را بزنید و برنامه را اجرا کنید.اینبار 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.
برنامه را ری استارت کرده و با کلید F8 انقدر به جلو بروید تا به این حلقه برسید:



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



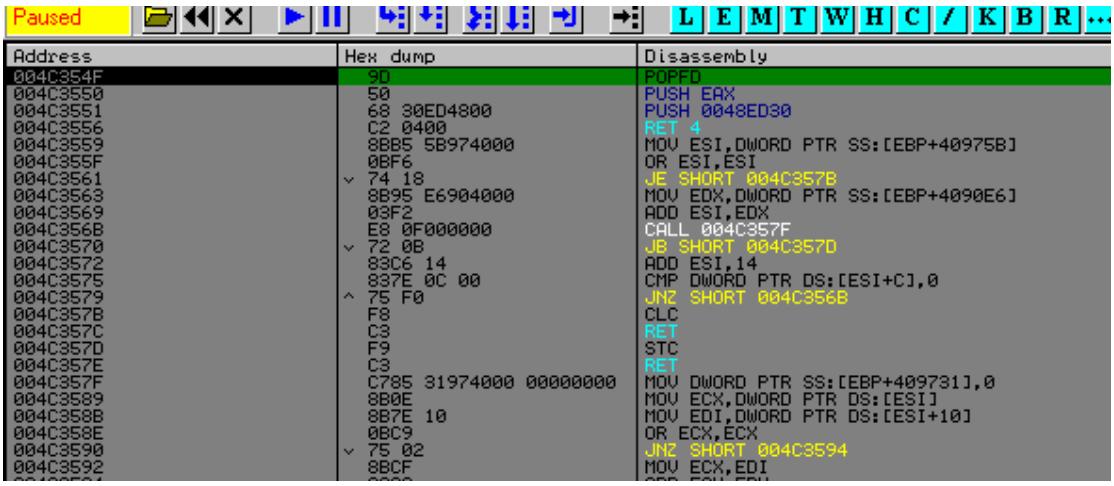
حالا کلید F9 را بزنید تا به OEP برسید:



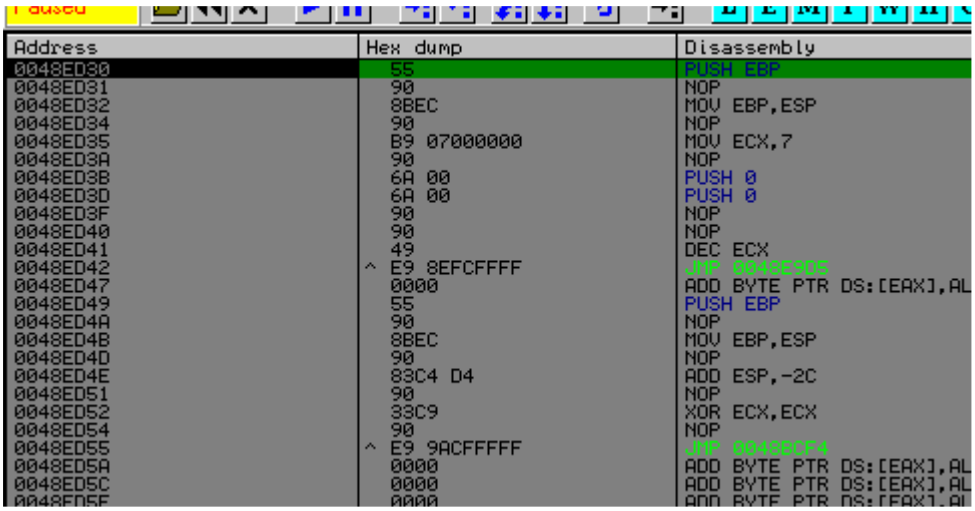
از این به بعد دیگر مشکلی نیست،دامپ می گیرید و فیکس می کنید.

NoName Packer 2

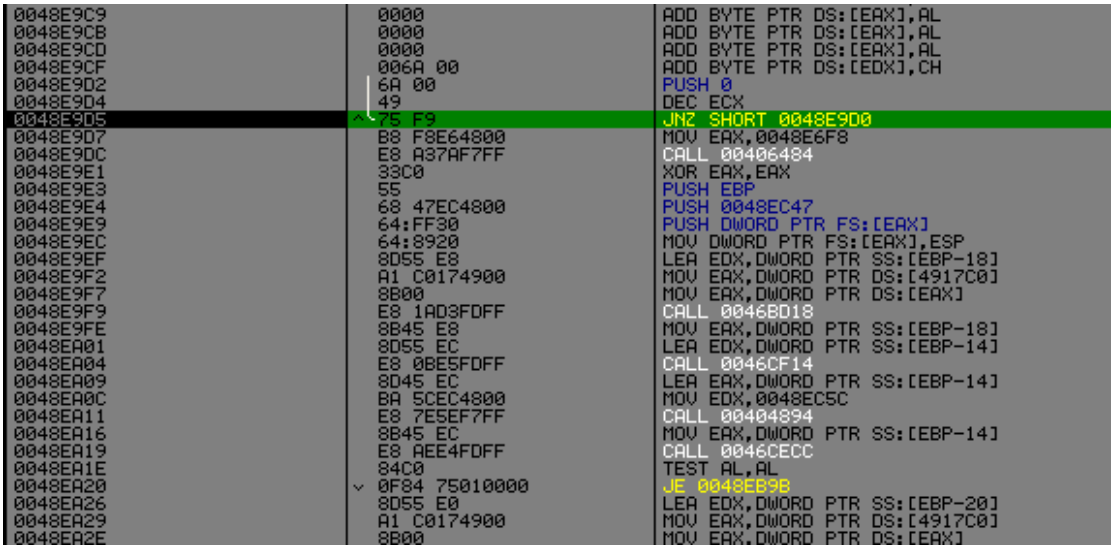
فایل مورد نظر را در OllyDBG باز کنید،سـه بـار کلیـد F8 را بزنیـد تـا دسـتور PushAD اجـرا شـود و بعـد روی مقـدار رجیـستر ESP یـک Hardware Breakpoint on access بگذارید.و کلید F9 را بزنید.برنامه در اینجا متوقف می شود:



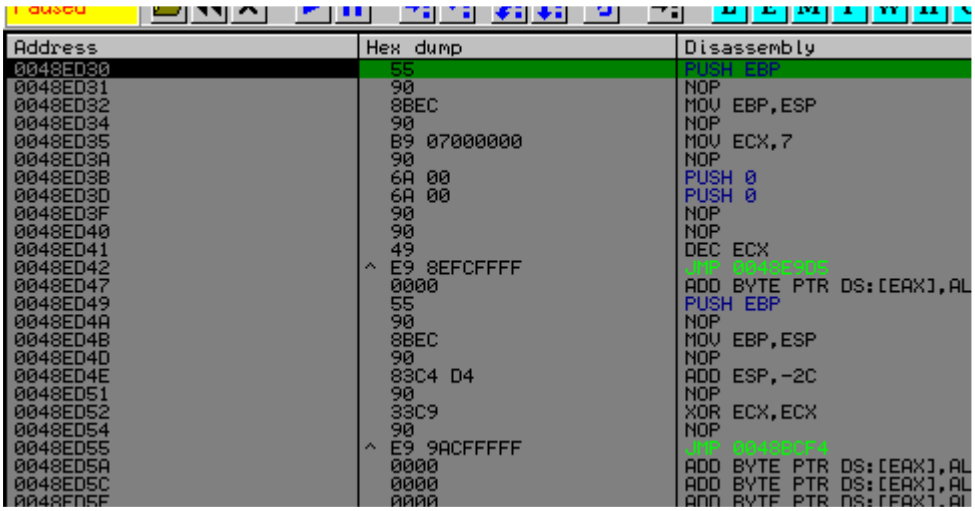
چهار بار کلید F8 را بزنید تا به اینجا برسید:



اینجا باید OEP ما باشد...اما کدهای زیادی در اینجا موجود نیست.پس بهتر است بـا F8 برنامـه را ادامـه بـدهیم تـا ببیـنم بقیـه کـدها کجاست؟...چند بار کلید F8 را بزنید تا به خط 48ED42 برسید،این پرش شما را به سایر کدهای برنامه می برد:



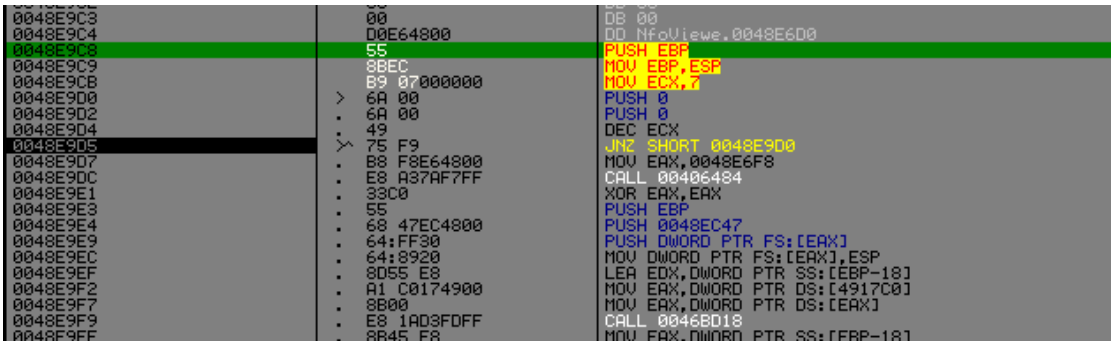
می بینید؟..سایر کدهای ما در اینجا قرار دارد.این یعنی این پکر از قابلیت Stolen Bytes استفاده می کند.یعنی پکر می آید و چند دستور ابتدایی فایل ما را از ابتدا برمی دارد و در جایی دیگر اجرا می کند...حالا باید چه کار کنیم؟...ما باید بایت های دزدیده شده را از آنجا برداریم و به جای اصلیش منتقل کنیم،یک بار دیگر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



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

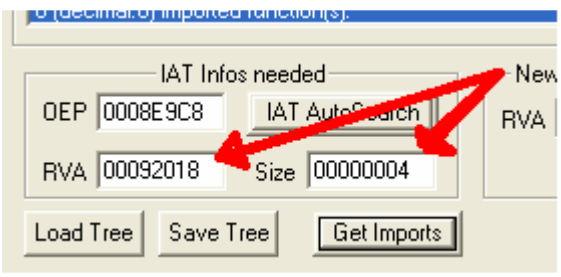
حالا کلیدهای CTRL + A را بزنید تا کدهای این سکشن آنالیز شود و بعد این سه دستور را از خط 0048E9C8 به سایر کدهای برنامه اضافه می کنیم تا این مشکل هم حل شود :



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

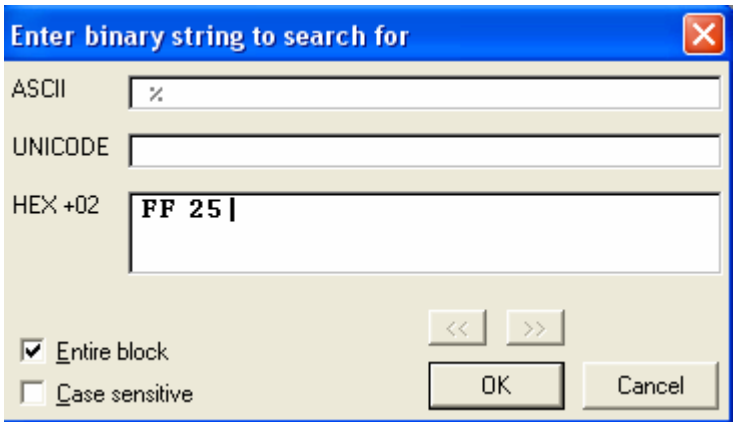
توضیح: گزینه New Origin Here مقدار رجیستر EIP را تغییر می دهد.و چون رجیستر EIP نشان دهنده خطی است که قرار است اجرا شود.ما با تغییر این رجیستر می توانیم روند اجرا شدن کدها را تغییر دهیم. ☺

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:

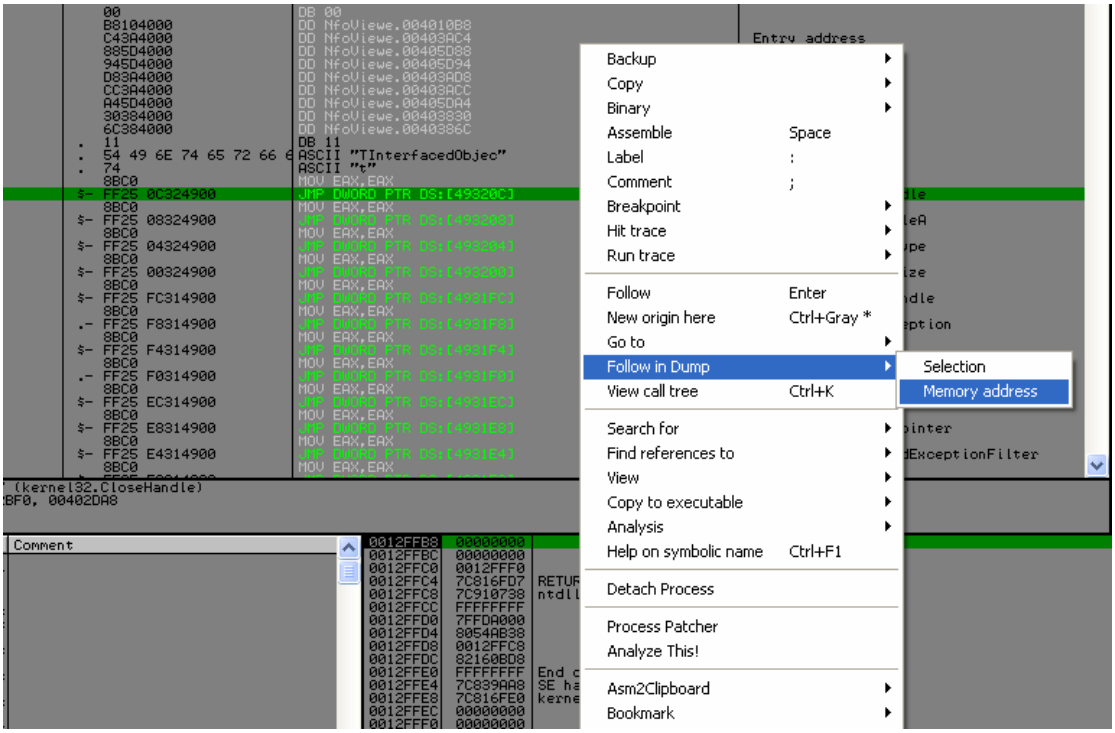


همانطور که می بینید ImportREC نتوانسته IAT را درست پیدا کند.پس باید خودمان دست به کار بشویم و IAT را پیدا کنیم.

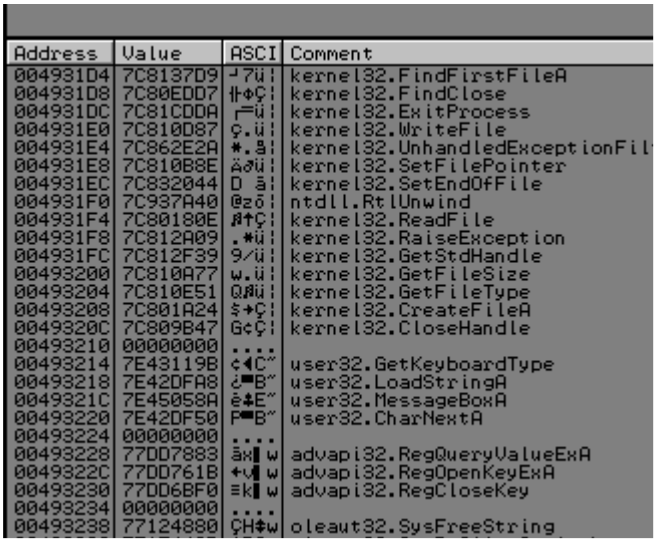
در پنجره OllyDBG کلیدهای CTRL + B را بزنید و تایپ کنید FF25 تا بتوانیم Jump Table را پیدا کنیم:



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.

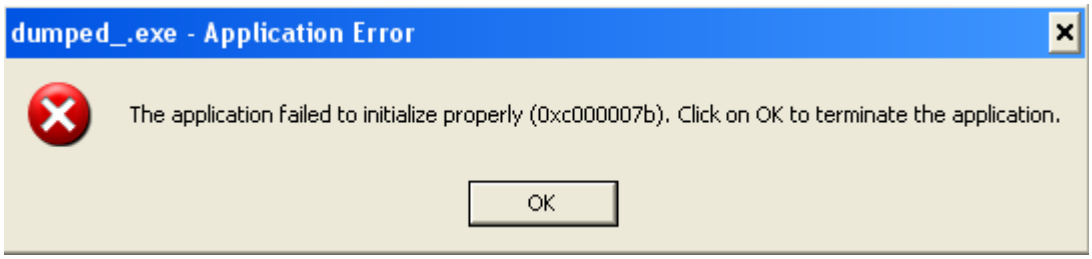


Address	Value	ASCII	Comment
004931D4	7C8137D9	┘7ü¦	kernel32.FindFirstFileA
004931D8	7C80EDD7	╫ΦÇ¦	kernel32.FindClose
004931DC	7C81CDDA	┌═ü¦	kernel32.ExitProcess
004931E0	7C810D87	ç.ü¦	kernel32.WriteFile
004931E4	7C862E2A	♦.å¦	kernel32.UnhandledExceptionFil
004931E8	7C810B8E	Ä∂ü¦	kernel32.SetFilePointer
004931EC	7C832044	D â¦	kernel32.SetEndOfFile
004931F0	7C937A40	@zô¦	ntdll.RtlUnwind
004931F4	7C80180E	♫↑Ç¦	kernel32.ReadFile
004931F8	7C812A09	.♦ü¦	kernel32.RaiseException
004931FC	7C812F39	9/ü¦	kernel32.GetStdHandle
00493200	7C810A77	w.ü¦	kernel32.GetFileSize
00493204	7C810E51	Q♫ü¦	kernel32.GetFileType
00493208	7C801A24	$→Ç¦	kernel32.CreateFileA
0049320C	7C809B47	G¢Ç¦	kernel32.CloseHandle
00493210	00000000	....	
00493214	7E43119B	¢◄C~	user32.GetKeyboardType
00493218	7E42DFA8	¿▀B~	user32.LoadStringA
0049321C	7E45058A	è♣E~	user32.MessageBoxA
00493220	7E42DF50	P▀B~	user32.CharNextA
00493224	00000000	....	
00493228	77DD7883	âx▌w	advapi32.RegQueryValueExA
0049322C	77DD761B	←v▌w	advapi32.RegOpenKeyExA
00493230	77DD6BF0	≡k▌w	advapi32.RegCloseKey
00493234	00000000	....	
00493238	77124880	ÇH♦w	oleaut32.SysFreeString

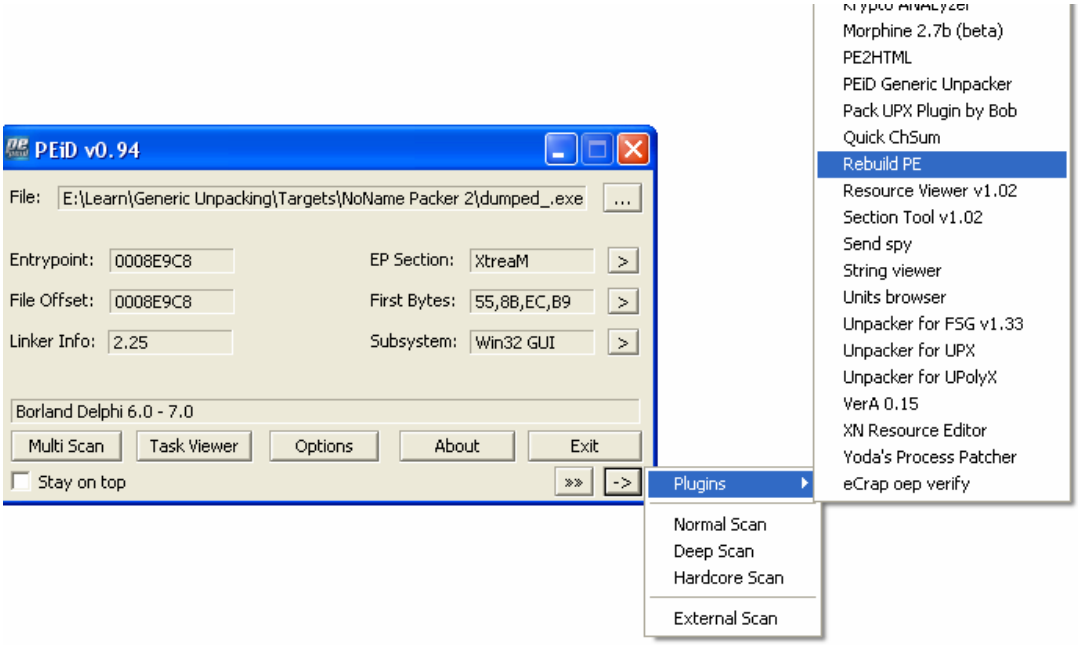
من در مثال های قبل طریقه به دست آوردن IAT را توضیح دادم.پس اینجا هیچ توضیحی نمی دهم و فقط اطلاعات را می دهم:

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

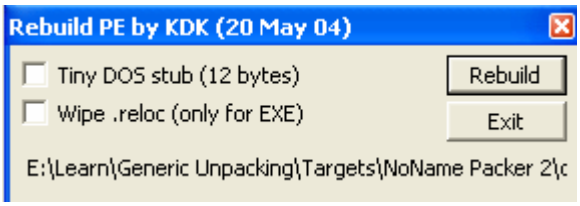
حالا این اطلاعات را در ImportREC وارد کنید و کلید Get Imports را بزنید،و بعد فایل دامپ شده خود را فیکس کنید.حالا فایل آنپک شده خود را اجرا کنید:



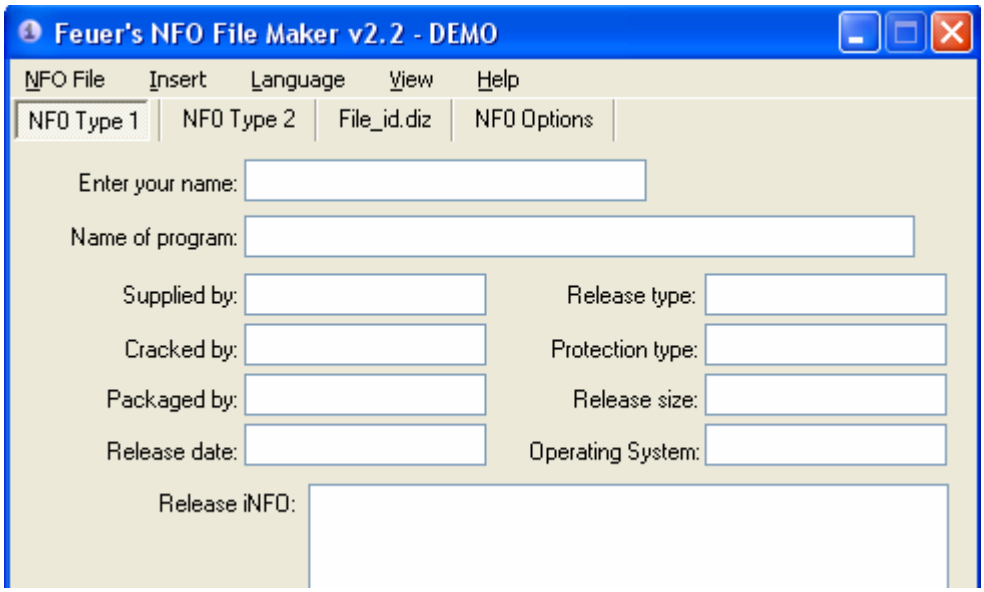
فایل اجرا نمی شود،اما مشکلی نیست.ما با ساختن دوباره فایل آنپک شده می توانیم مشکل را حل کنیم(Rebuild PE)
برای ساختن دوباره فایل می توانیم از هر نرم افزاری که دلمان خواست استفاده کنیم.اما در اینجا برای تنوع از PEiD استفاده می کنیم.برای این کار فایل مورد نظر را در PEiD باز کنید،و همانند تصویر زیر عمل کنید:



حالا هم گزینه Rebuild را بزنید:



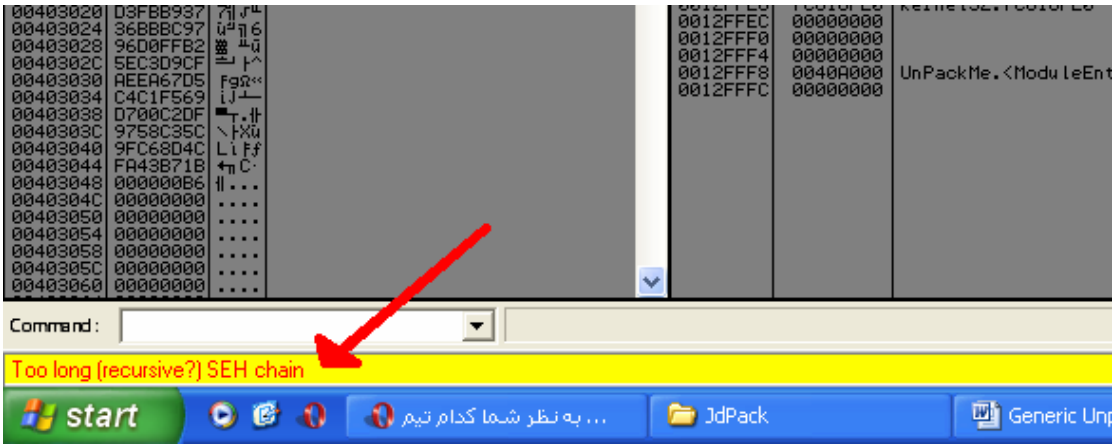
دوباره فایل را اجرا کنید:



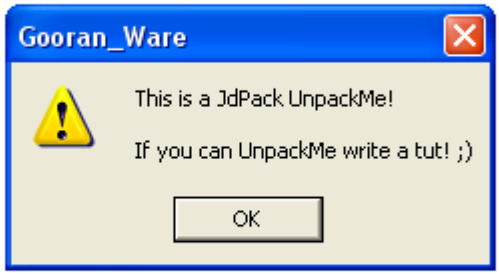
این بار فایل بی هیچ مشکلی اجرا شد.

JDPack

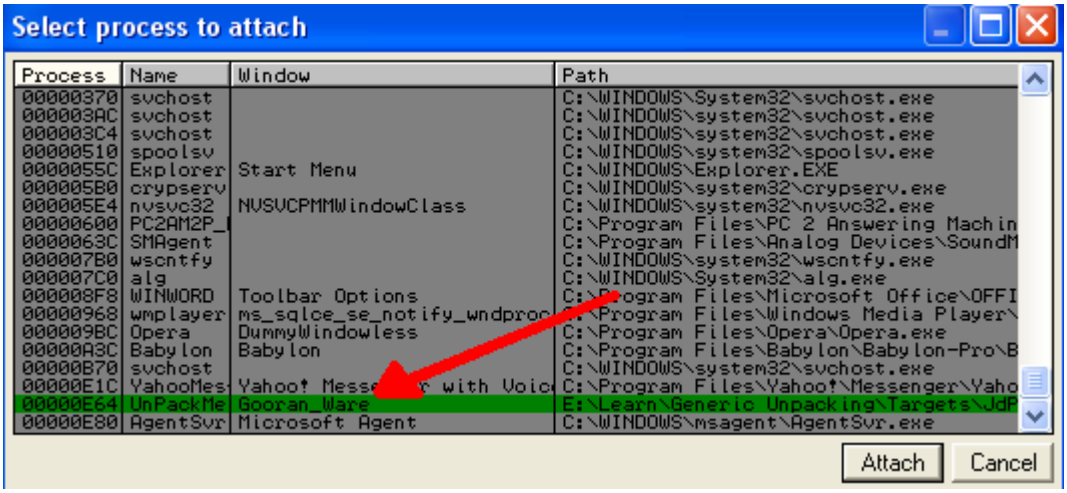
فایل مورد نظر را در OllyDBG باز کنید و بعد کلید F9 را بزنید تا برنامه اجرا شود.



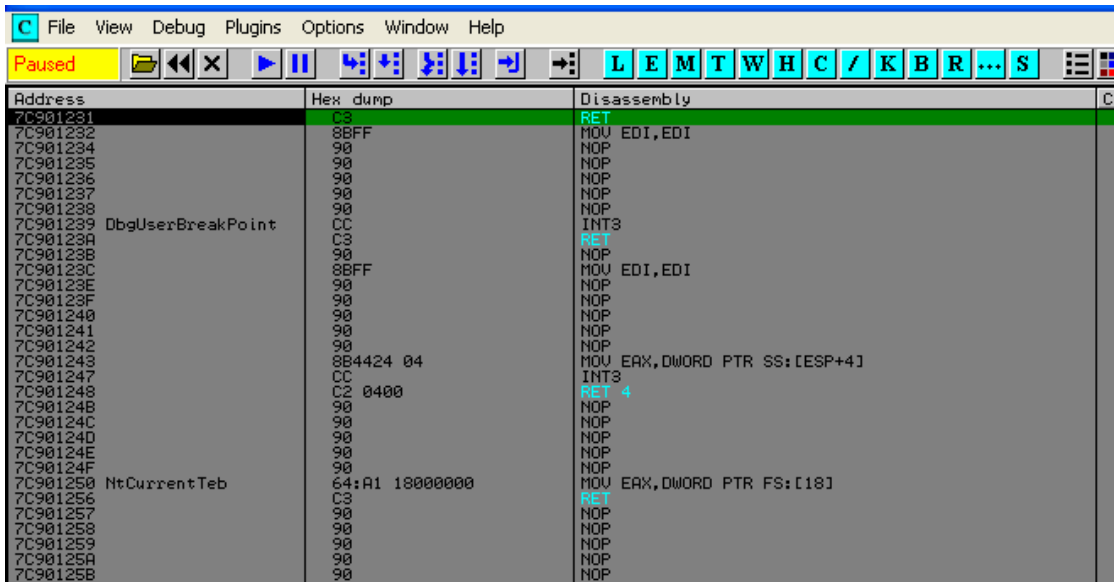
همانطور که می بینید فایل مورد نظر به سختی در OllyDBG دیباگ می شود و خیلی دیر اجرا می شود ☹
شاید این یک آنتی دیباگ باشد.(هر چند من معتقدم که این آنتی دیباگ نیست،چون فایل در داخل ویندوز هم دیر اجرا می شود)
خوب...حالا باید چه کار کنیم؟...بی خیال بشیم؟...نه،همیشه یک راه حلی هست.کلید ALT + F2 را بزنید تا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



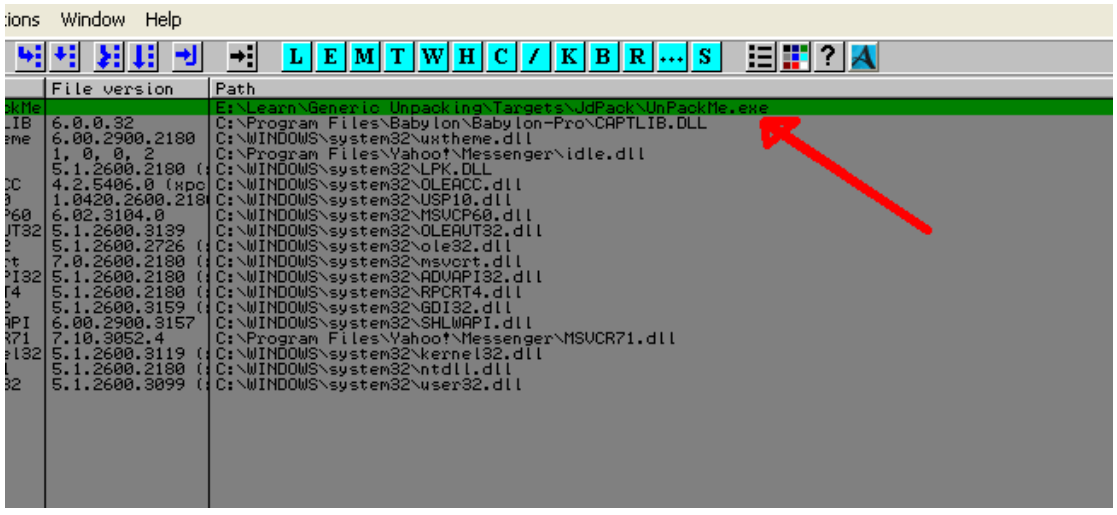
بعد از اجرا کردن برنامه توسط ویندوز،حالا به پروسه این فایل با OllyDBG ، Attach می کنیم.
در OllyDBG در منوی File گزینه Attach را می زنیم و بعد پروسه مورد نظرمان را انتخاب می کنیم:

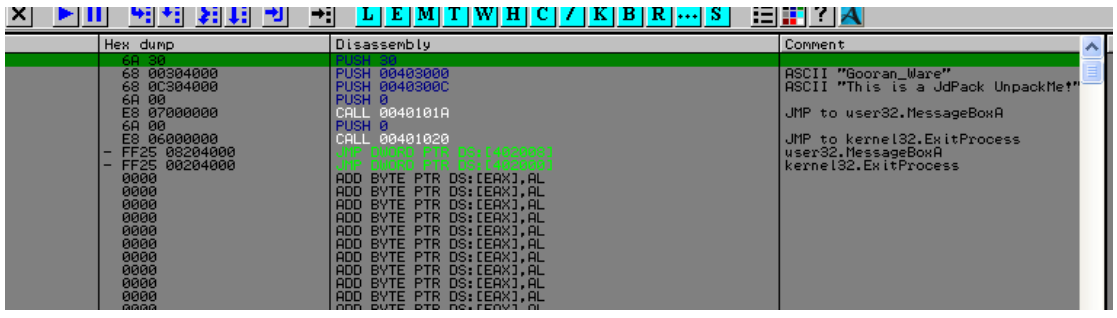


حالا گزینه Attach را بزنید تا به پروسه بچسبید.

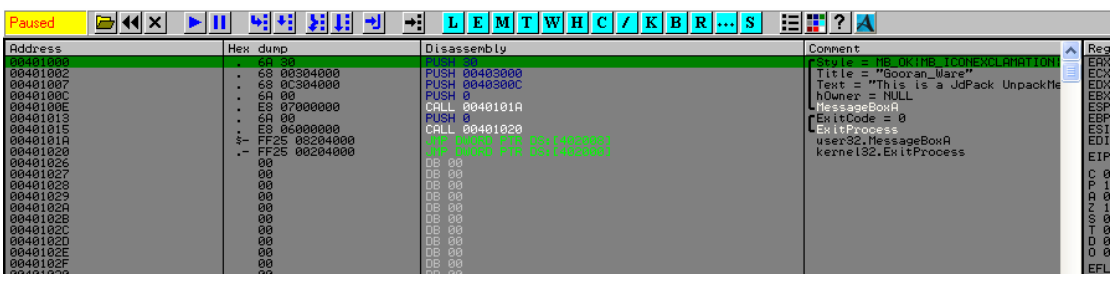


حالا در منوی View گزینه Executable modules را بزنید و بر روی پروسه مورد نظرتان دو بار کلیک کنید:





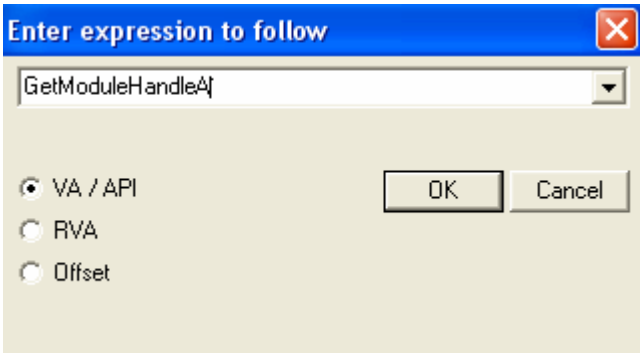
کلیدهای CTRL + A را بزنید تا کدها آنالیز شود.



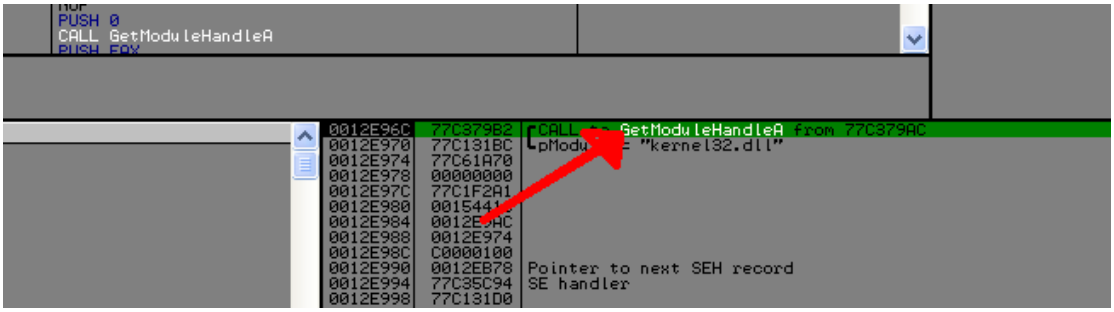
همانطور که می بینید برنامه ما در MASM نوشته شده و همین چند خط کد را دارد.با تابع MessageBoxA یک تابع را نشان می دهد و بعد با تابع ExitProcess برنامه بسته می شود.پس OEP ما اولین خط در سکشن اصلی فایل می باشد.یک خط 00401000 بر روی خط 00401000 کلیک راست کرده و گزینه New origin here را بزنید.حالا از فایل 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 معمولا در نزدیکی های OEP در C و دلفی تابع GetModuleHanldeA اجرا می شود.
کلیدهای CTRL + G را بزنید.و تایپ کنید : GetModuleHandleA

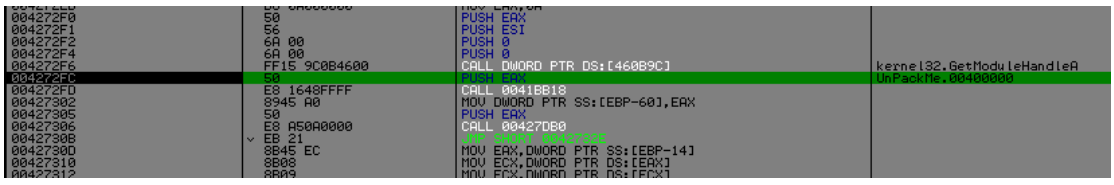


و بعد کلیک راست کرده،گزینه Breakpoint و سپس گزینه Hardware,on execution را بزنید.حالا کلید F9 را بزنید:

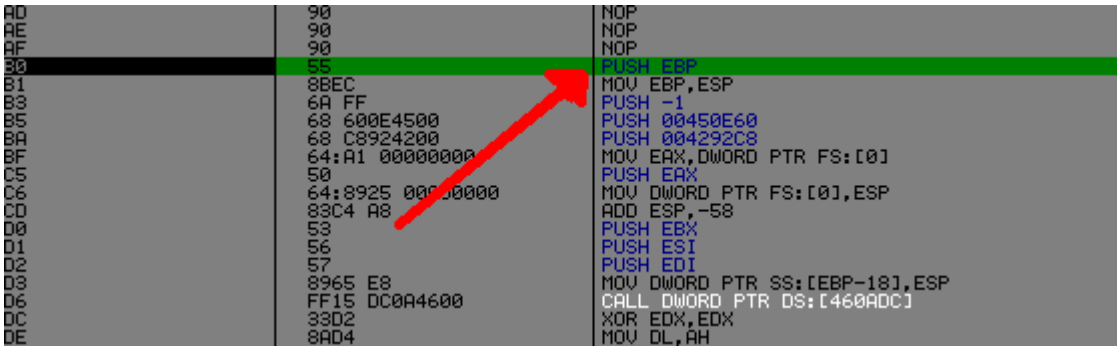


پارامتر Pmodule در اینجا Kernel32.dll است.این پارامتر باید Null باشد.پس این چیزی نیست که ما می خواهیم.آنقدر کلید F9 را بزنید تا به اینجا برسید:

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



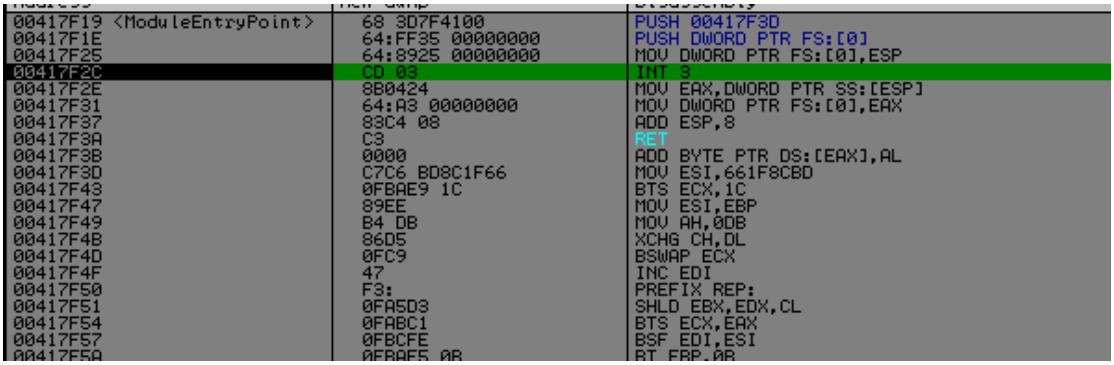
اینجا نزدیکی های OEP است.مقداری به بالا بروید تا سایر توابعی که در نزدیکی های OEP می آیند را ببینید و بعد به خود OEP می رسید:



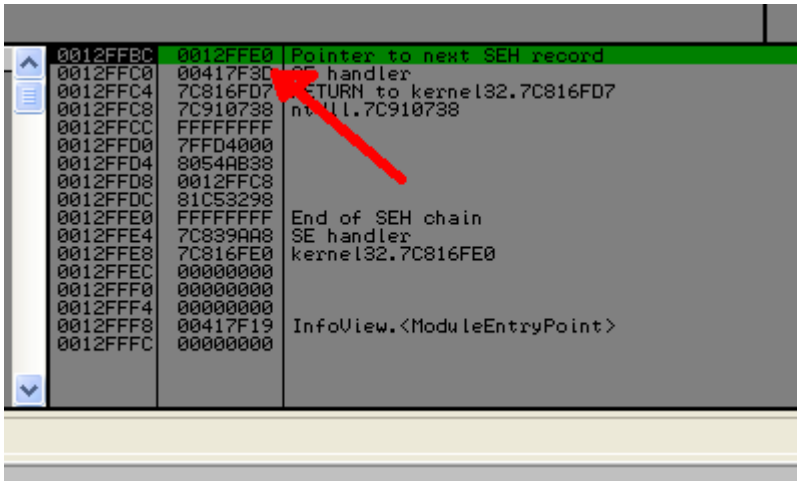
حالا کلیک راست کرده و گزینه new origin here را بزنید و بعد از فایل دامپ بگیرید(ترجیحا با LordPE این کار را بکنید.) و فیکس کنید

MJo0T MIV InfoViewer

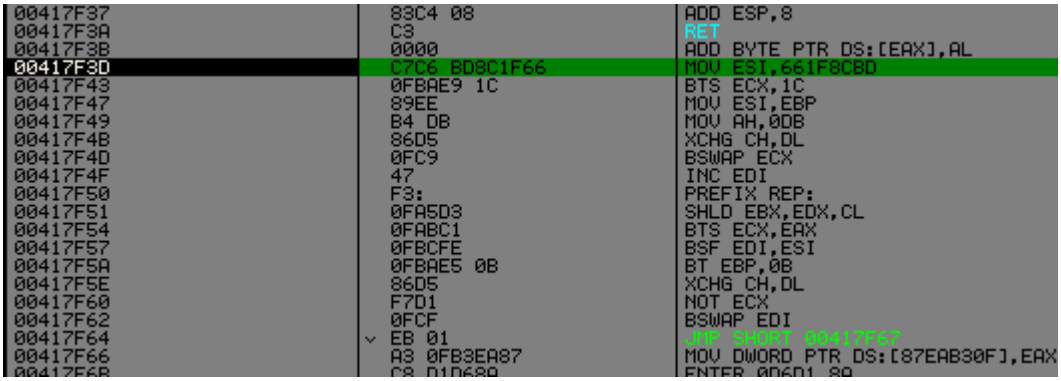
برنامه را در OllyDBG باز کنید.۳ بار کلید F8 را بزنید تا به دستور INT3 برسید.



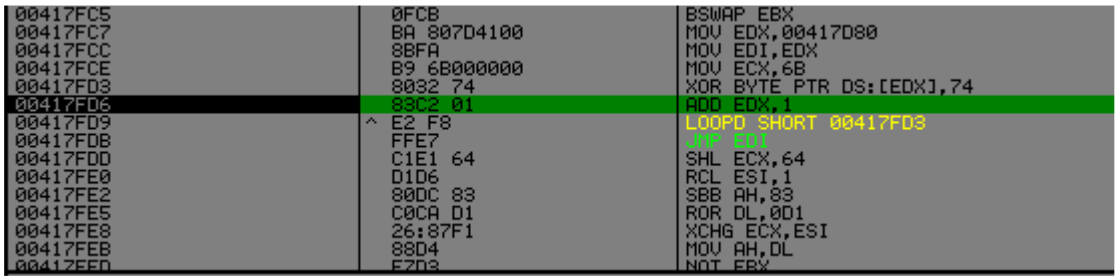
برنامه در این خط دچار خطای INT 3 Break می شود.برنامه بعد از رسیدن به خطا به تابع SEH Handler می رود.به پنجره دامپ نگاه کنید:



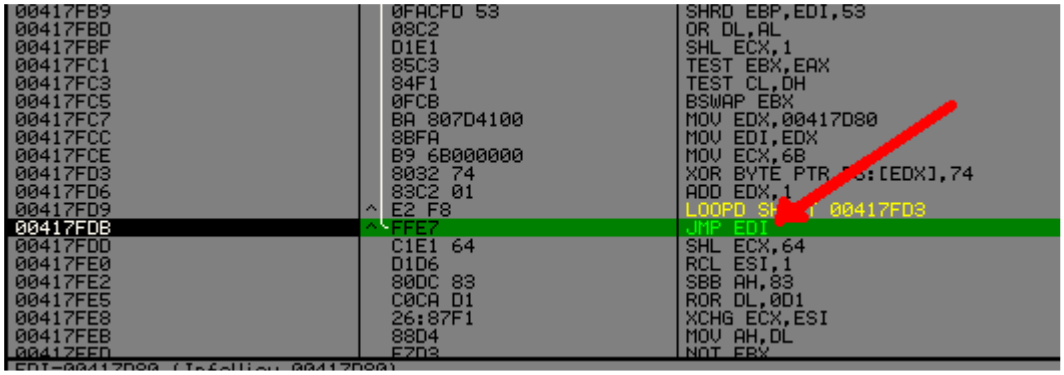
پس برنامه ما با این خطا به خط 417F3D می رود.بر روی این خط Brekapoint بگذارید و برنامه را با F9 اجرا کنید تا در جایی که BP گذاشتیم،متوقف شویم:



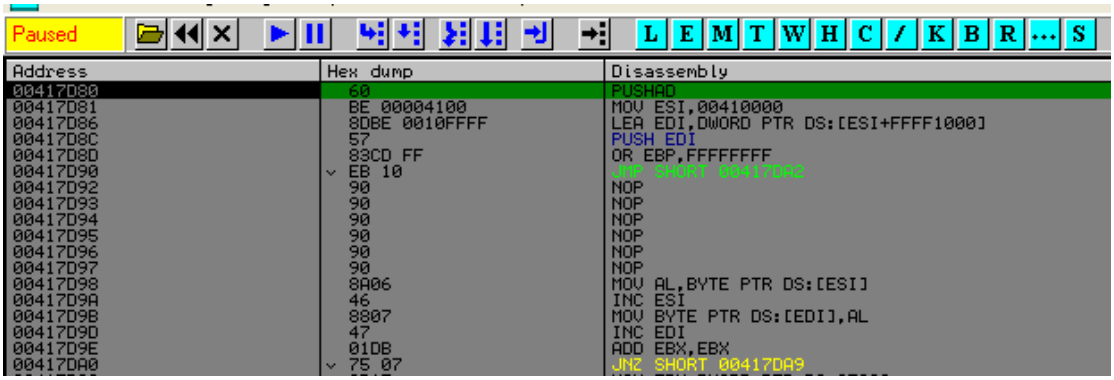
حال برنامه را با F8 ادامه دهید تا به این خط برسید:



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



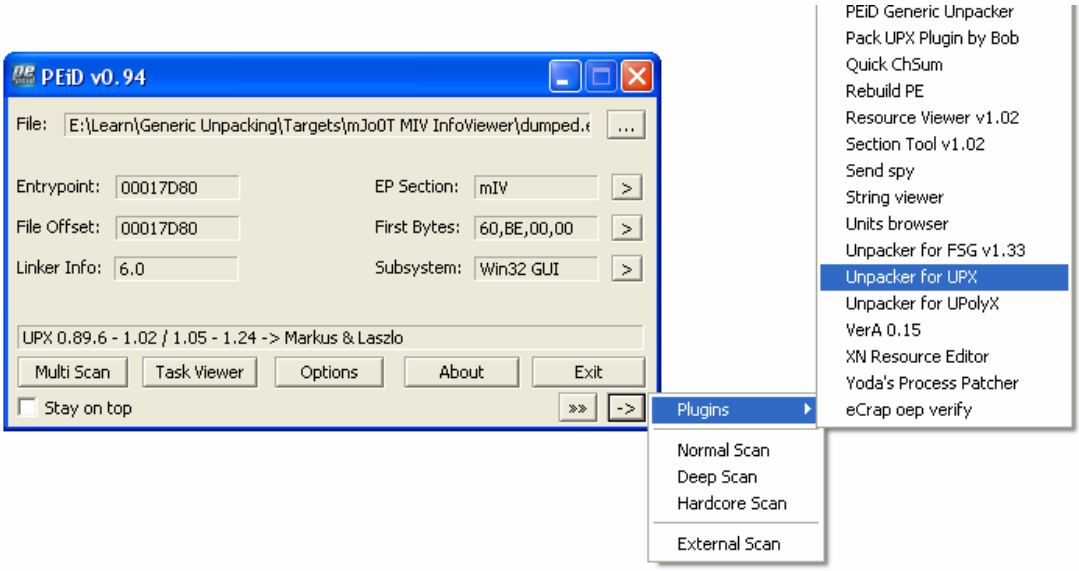
یکبار F8 بزنید:



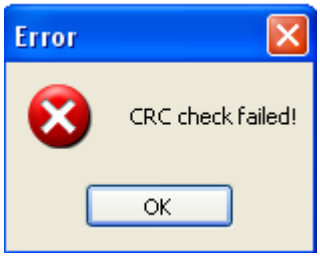
این دستورات شما را یاد چیزی نمی اندازد؟بله...این دستوراتی است که در ابتدای UPX قرار دارد.پس این پکر در واقع UPX بوده و Entry point آن دستکاری شده بود...حالا چه کار کنیم؟...همینجا از فایل دامپ می گیریم و با یک UPX Unpacker فایل را آنپک می کنیم ☺

با پلاگین OllyDump از فایل دامپ بگیرید.(تیک گزینه ReBuild Import را بزنید.تا IAT ساخته شود.)

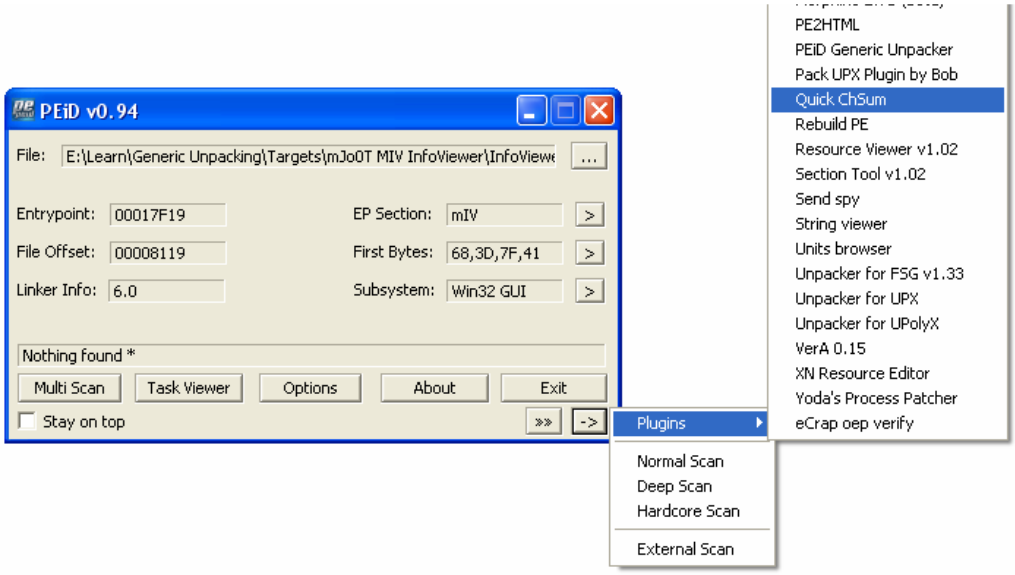
حالا فایل ما با UPX پک شده،پس با پلاگین PeiD فایل را آنپک می کنیم.فایلی که دامپ گرفتیم را در Peid باز کنید و همانند تصویر عمل کنید:

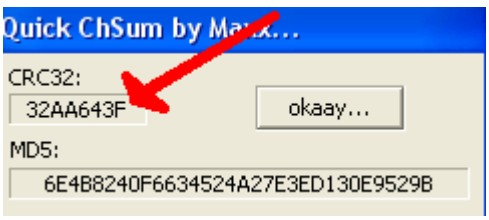


فایل آنپک شده را باز کنید:

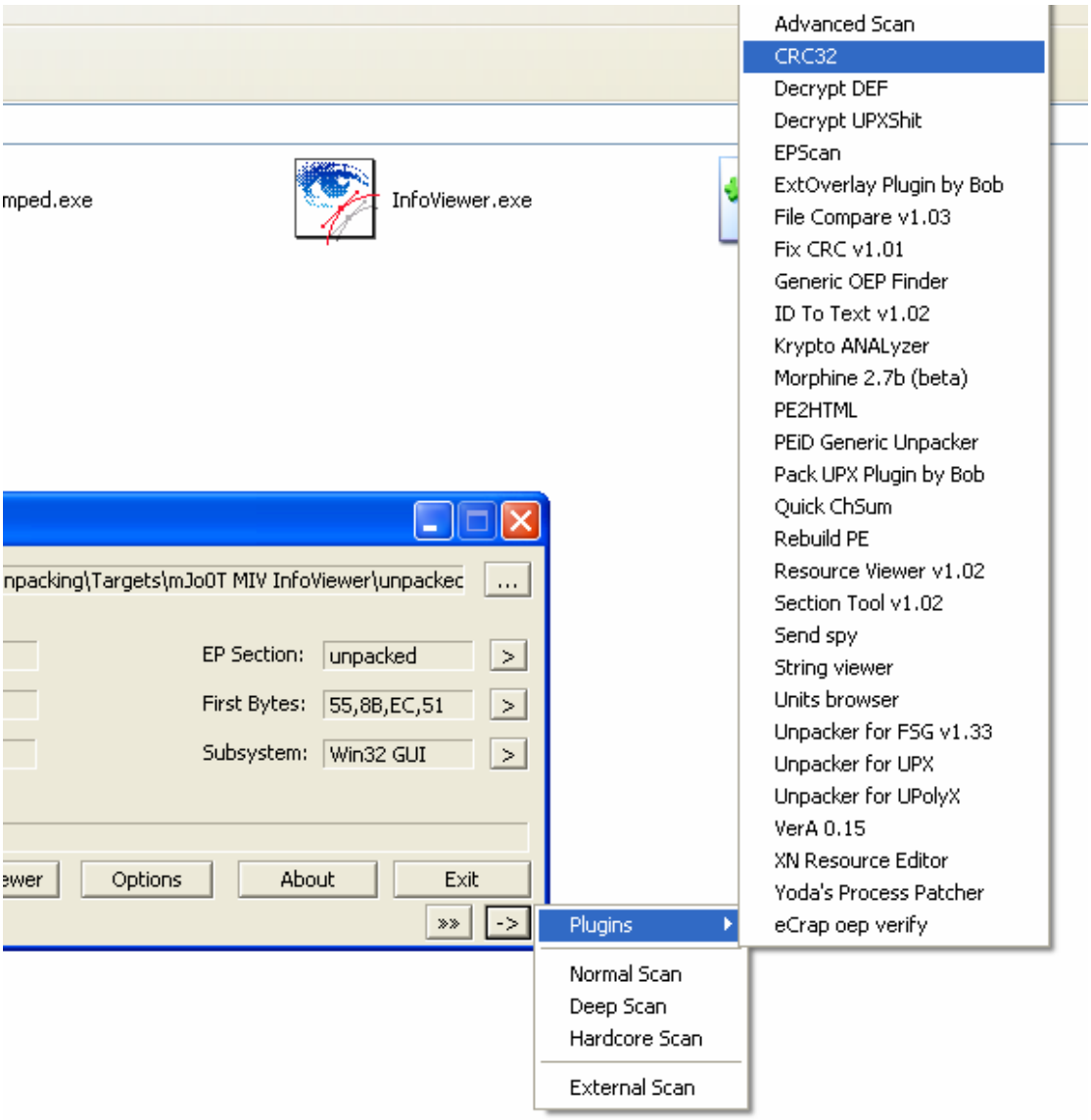


همانطور که می بینید برنامه CRC خودش را چک می کند و اگر بفهمد که فایل دستکاری شده است،جلوی اجرا شدن برنامه را می گیرد و این پیغام را می دهد.حالا چه کار باید بکنیم؟
حالا می توانیم با پلاگین CRC، PeiD فایل را تغییر دهیم.اول باید ببینیم که CRC اصلی فایل چه بوده؟...فایل اصلی(فایل پک شده) را در PeiD باز کنید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

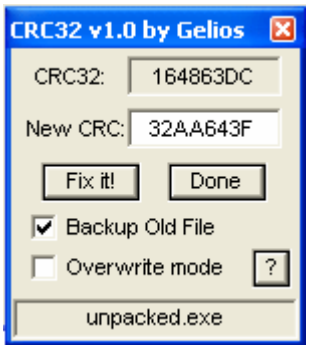




پس CRC فایل اصلی ما 32AA643F است.ما CRC فایل آنپک شده خود را تغییر می دهیم،تا به این ترتیب برنامه متوجه دستکاری شدن فایل نشود.حالا فایل آنپک شده را PEiD باز کنید و همانند تصویر عمل کنید:



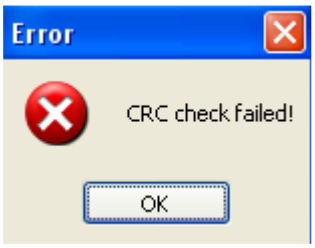
حالا هم CRC را به 32AA643F تغییر می دهید،و گزینه Fix It را بزنید،همانند تصویر:



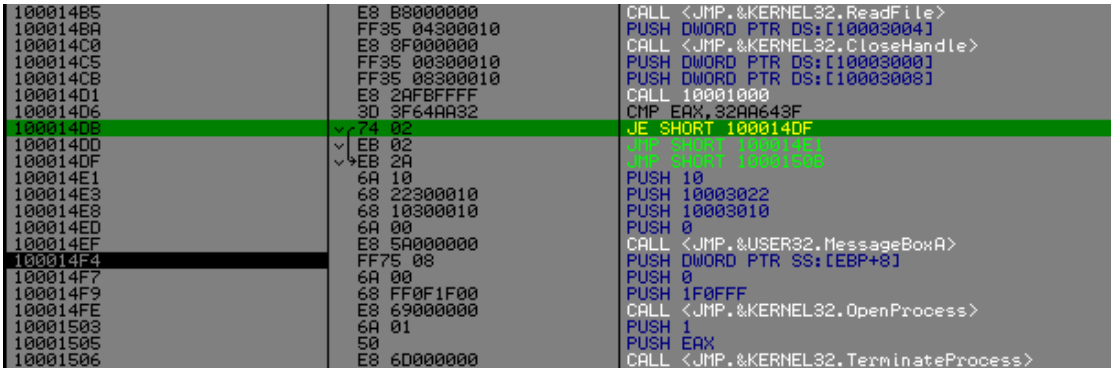
اینبار فایل آنپک شده را اجرا کنید،می بینید که فایل آنپک شده بی هیچ مشکلی اجرا می شود.

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:



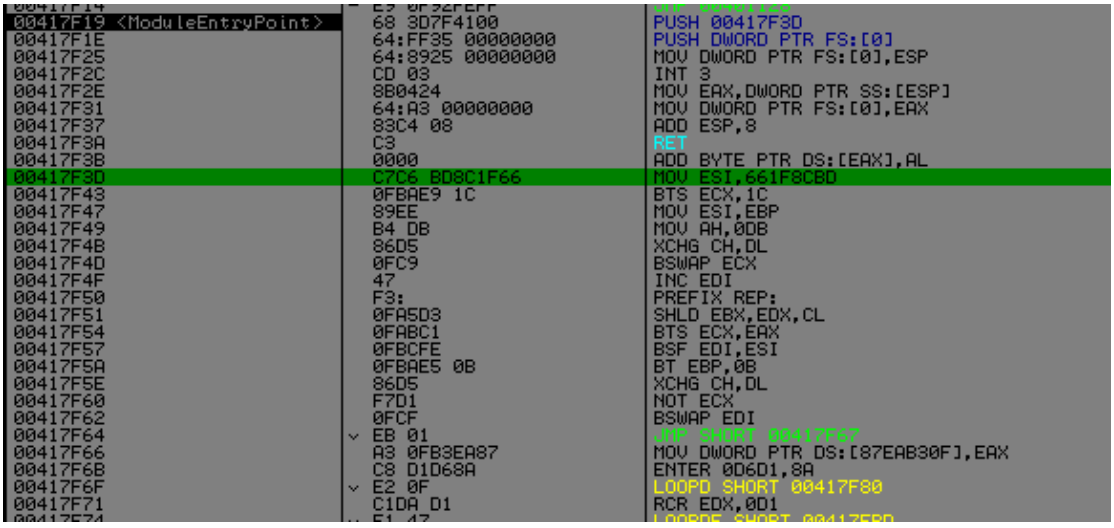
کلید F12 را بزنید تا برنامه متوقف شود و ALT + F9 را بزنید.تا وارد حالت Back to user شوید.و بعد کلید OK را بزنید تا این پیغام از بین برود.



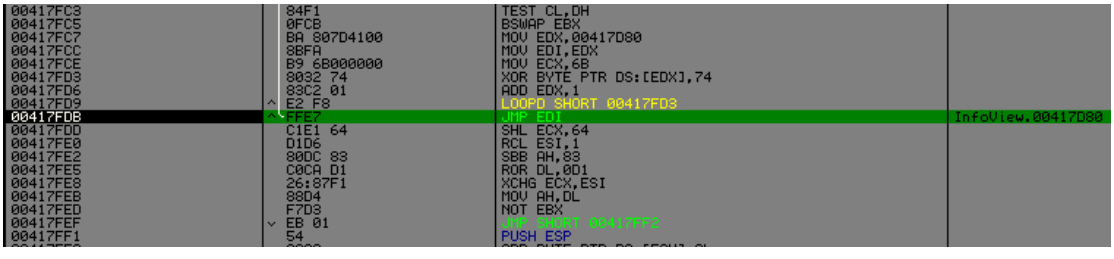
همانطور که می بینید از قیافه کدها مشخص است که پرشی که در خط 100014D8 قرار دارد،تصمیم گیرنده است.اگر این پرش انجام شود،CRC مشکلی ندارد و در غیر این صورت با پیغام CRC Check Error روبرو می شویم....حالا چه کار باید بکنیم؟...این پرش را به JMP تغییر دهید و فایل خود را ذخیره کنید.(همان طور که متوجه شدید عملیات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

راه حل دیگر:

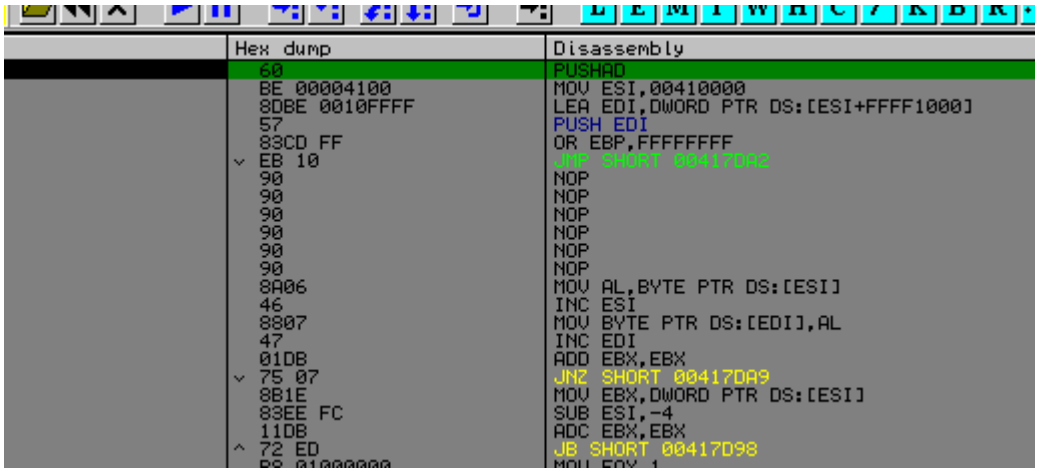
چون پکر ما در واقع UPX بوده،ما می توانیم همان کاری که در مورد UPX کردیم،اینجا هم بکنیم.فایل مورد نظر(فایل پک شده) را در OllyDBG باز کنید:



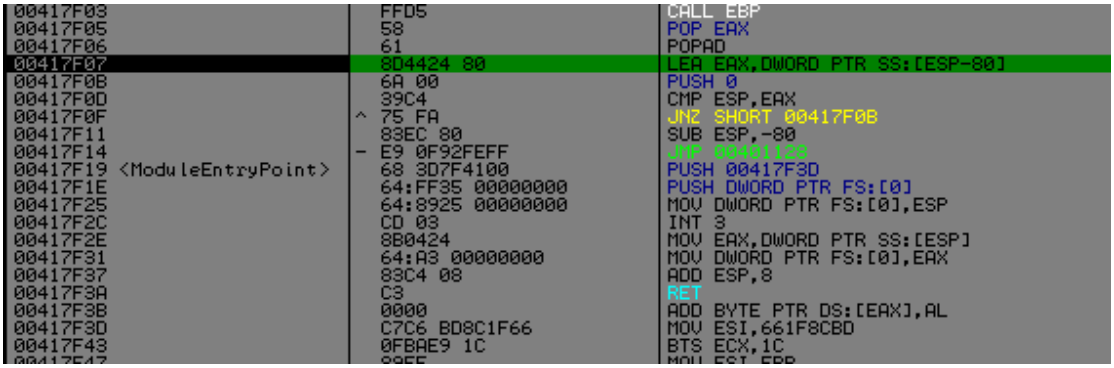
چند خط پایین تر بر روی دستور JMP EDI یک Breakpoint بگذارید و برنامه را با F9 اجرا کنید:



یکبار کلید F8 را بزنید:



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 Brekapoint on access می گذاریم.و بعد کلید F9 را بزنید:

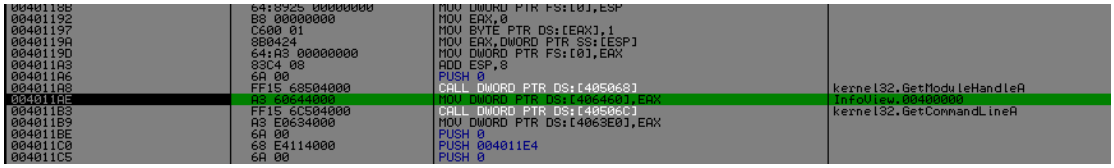


پرش JMP 00401128 ما را به OEP می برد.
همانطور که می بینید درست زیر این پرش،Entry point ما قرار دارد.این یعنی اینکه درست زیر پرشی که به OEP می رود،کدهایی تزریق شده و Entry point به آنجا منتقل شده است.
بعد از پیدا کردن OEP از فایل دامپ می گیرید و فیکس می کنید.

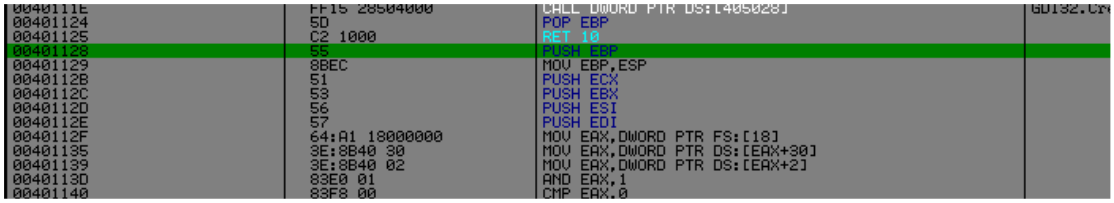
راه حل دیگر:

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.

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



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



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:

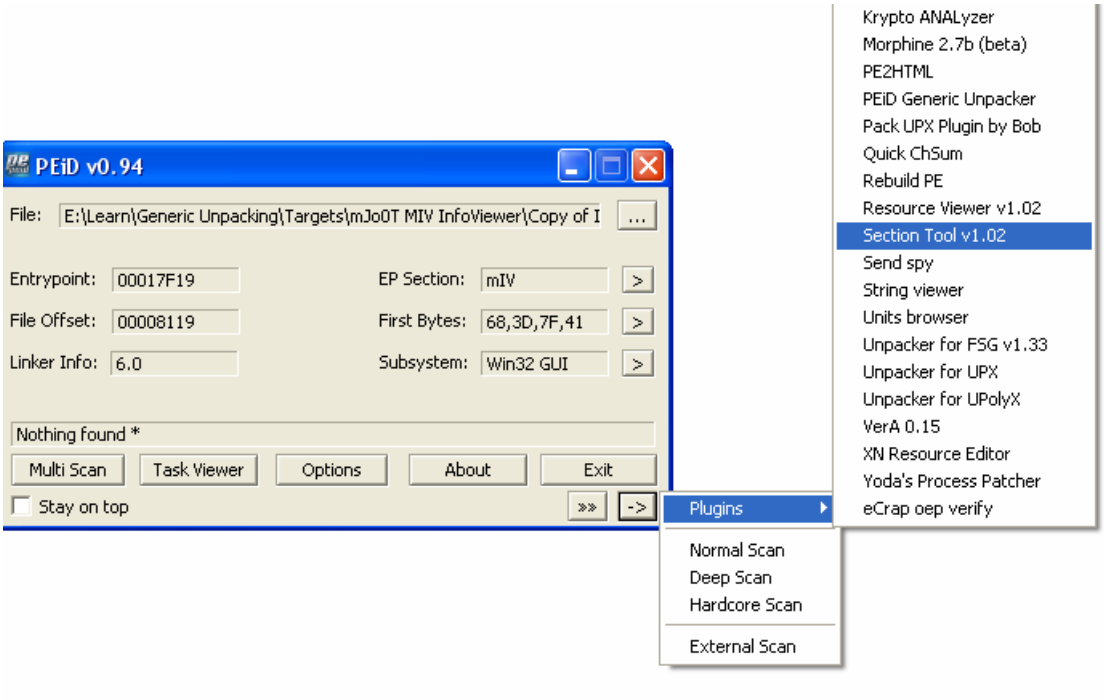
فرقي نمي كند كه Packer ما چي باشد،هر چي باشد،به هر حال،در يک جا و در يک زماني،اختيار از دست Decompression Stub خارج شده و به دست فايل اصلي ما مي افتد.

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.

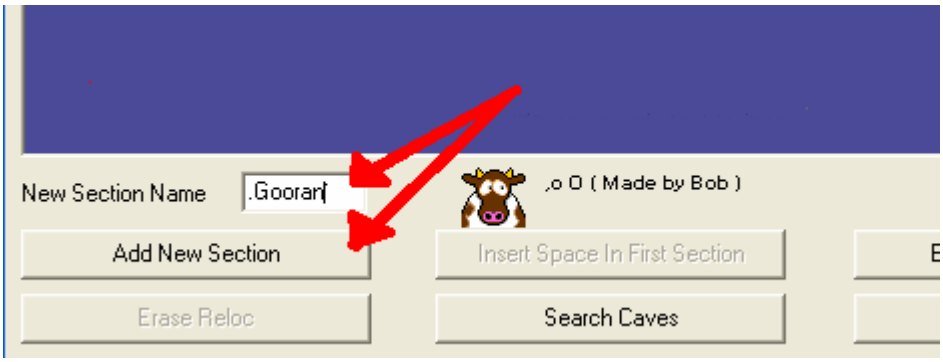
Inline Patch کردن نرم افزار خصوصا در مواردي که حجم فایل بالاست،یکی از بهترین راه حل هاست.

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.

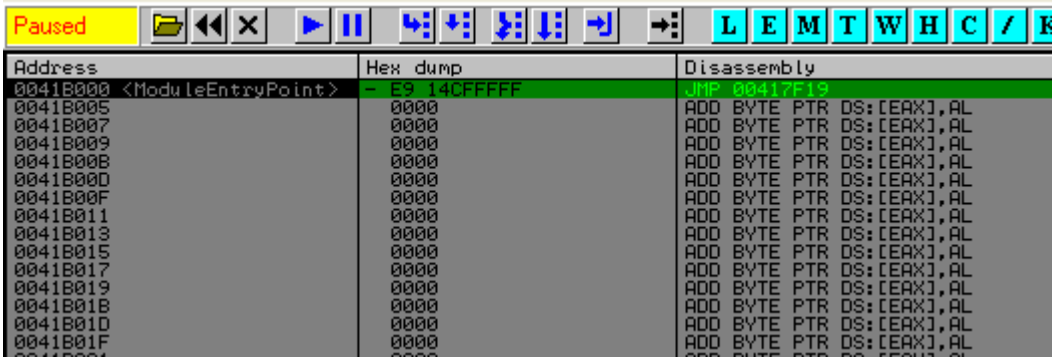
برنامه را در PeiD باز کنید و همانند تصویر عمل کنید:



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

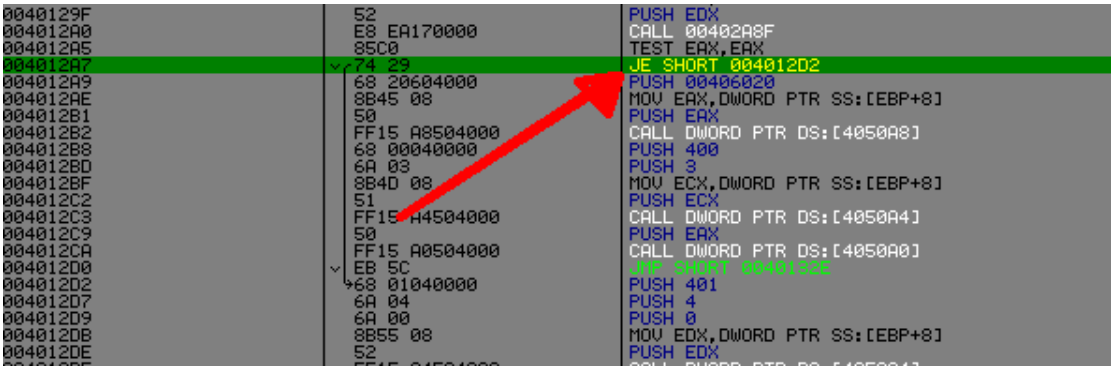


حالا فایلی که به آن سکشن اضافه کرده اید را در OllyDBG باز کنید:



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

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



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 ۹۰ است.یعنی برای NOP کردن پرشی که در 4012A7 قرار دارد،باید دو بایت 4012A7 و 4012A8 را برابر 90 کنم.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 کنم؟...معادل JMP, Opcode برابر EB است.بنابراین باید این بایت را برابر EB کنم:

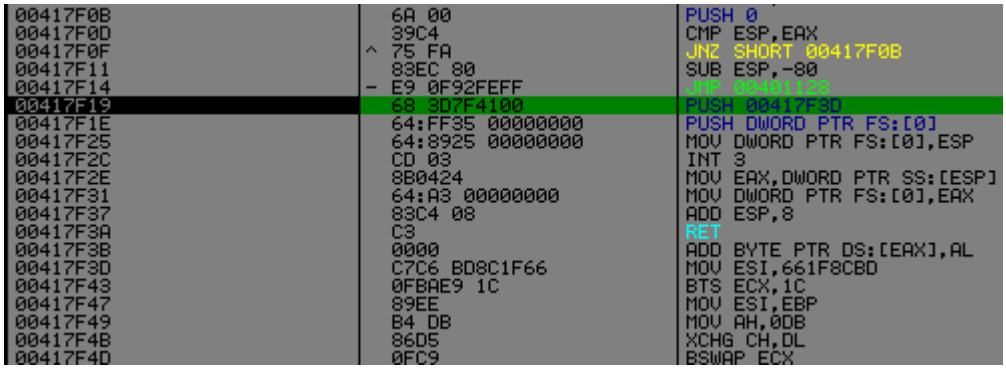
```
MOV Byte PTR DS: [401265], 0EB
```

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

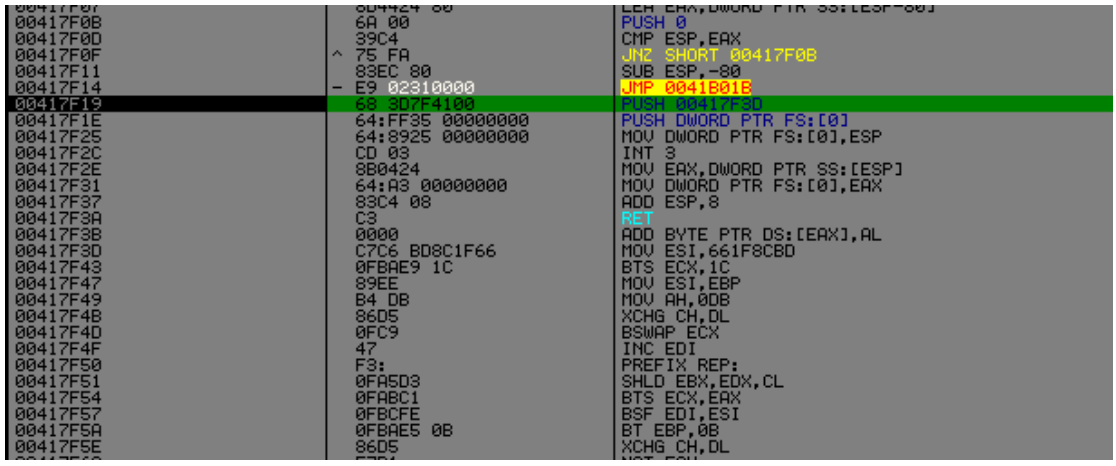
```
MOV WORD PTR DS: [4012A7], 9090
MOV BYTE PTR DS:[401265],0EB
```

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

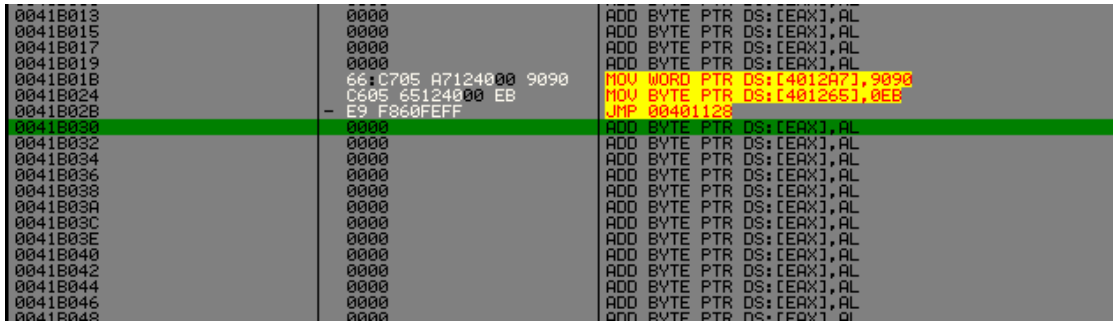
من در خط 41B01B این کدها را اضافه کنیم.شما می توانید در هر جایی از این سکشن که دلتان خواست،این دو خط کد را اضافه کنید.
حالا چه طور این کدها را اجرا کنیم؟...این که این کدها را اضافه کردیم که برنامه کرک نمی شود.حتما باید این کدها اجرا شود...اما چه طوری؟...ما می توانیم ببینیم برنامه در چه خطی به OEP می رود و بیایم 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:



ما الان در Entry point قبلی فایل هستم.(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 آنپک 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:



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



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

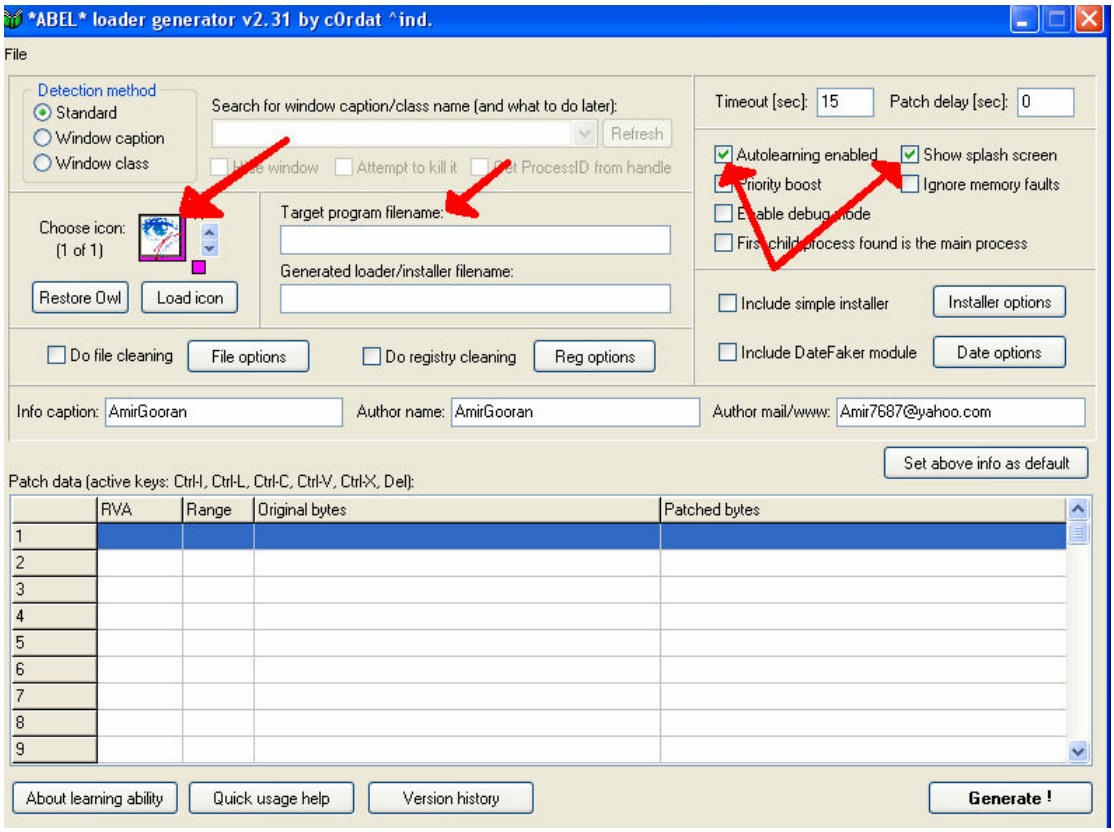
Loader

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:

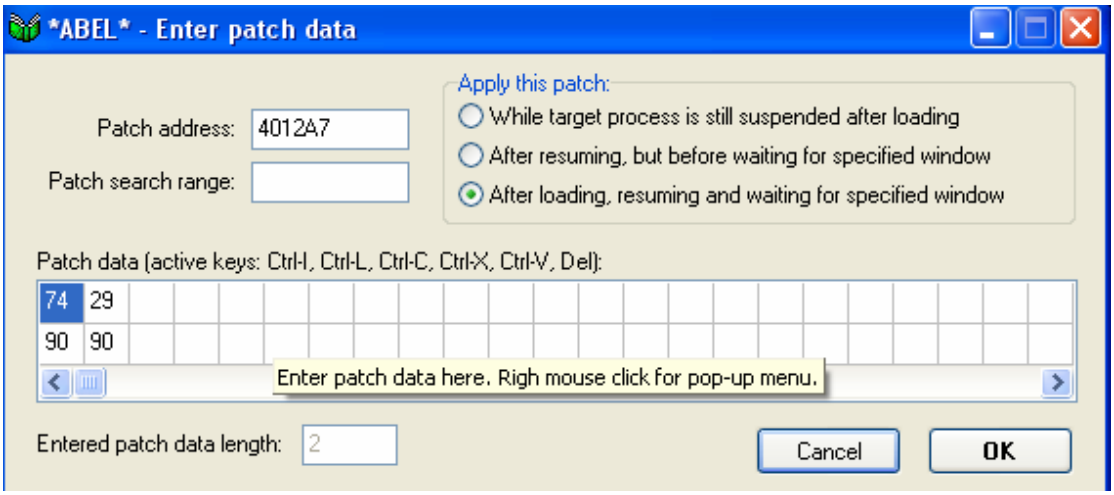
4012A7 → 90
4012A8 → 90
401265 → EB

ABEL Loader

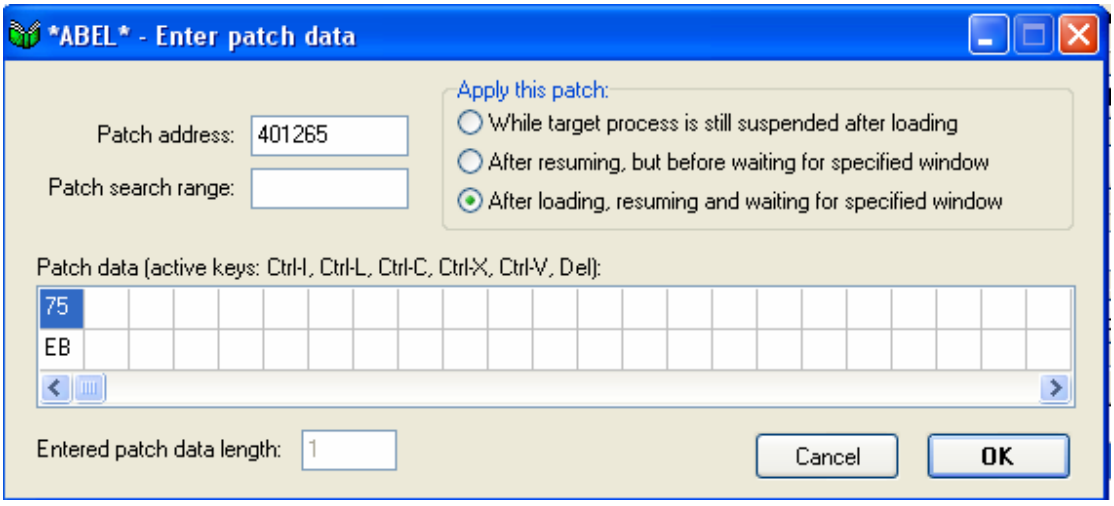
برای ساخت Loader نرم افزارهای زیادی وجود دارد که ما دو تا از معروفترین را آنها استفاده می کنیم.ابتدا برای ساخت Loader از ABEL Loader استفاده می کنیم،برنامه را باز کنید:



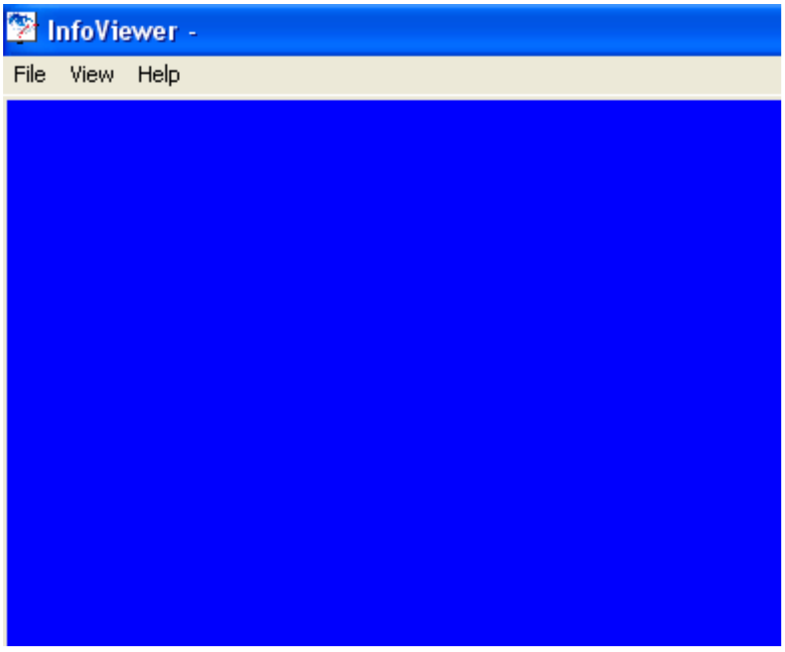
اول از همه تیک دو گزینه Autolearning Enabled و Show splash screen را بردارید.اگر تیک این دو گزینه را برندارید در هنگام اجرای Loader یک Splash screen نمایش داده می شود.
گزینه Load Icon را بزنید و آیکنی برای Loader خود انتخاب کنید.
در قسمت Target program filename هم نام فایل هدف را بنویسید(InfoViewer.exe)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

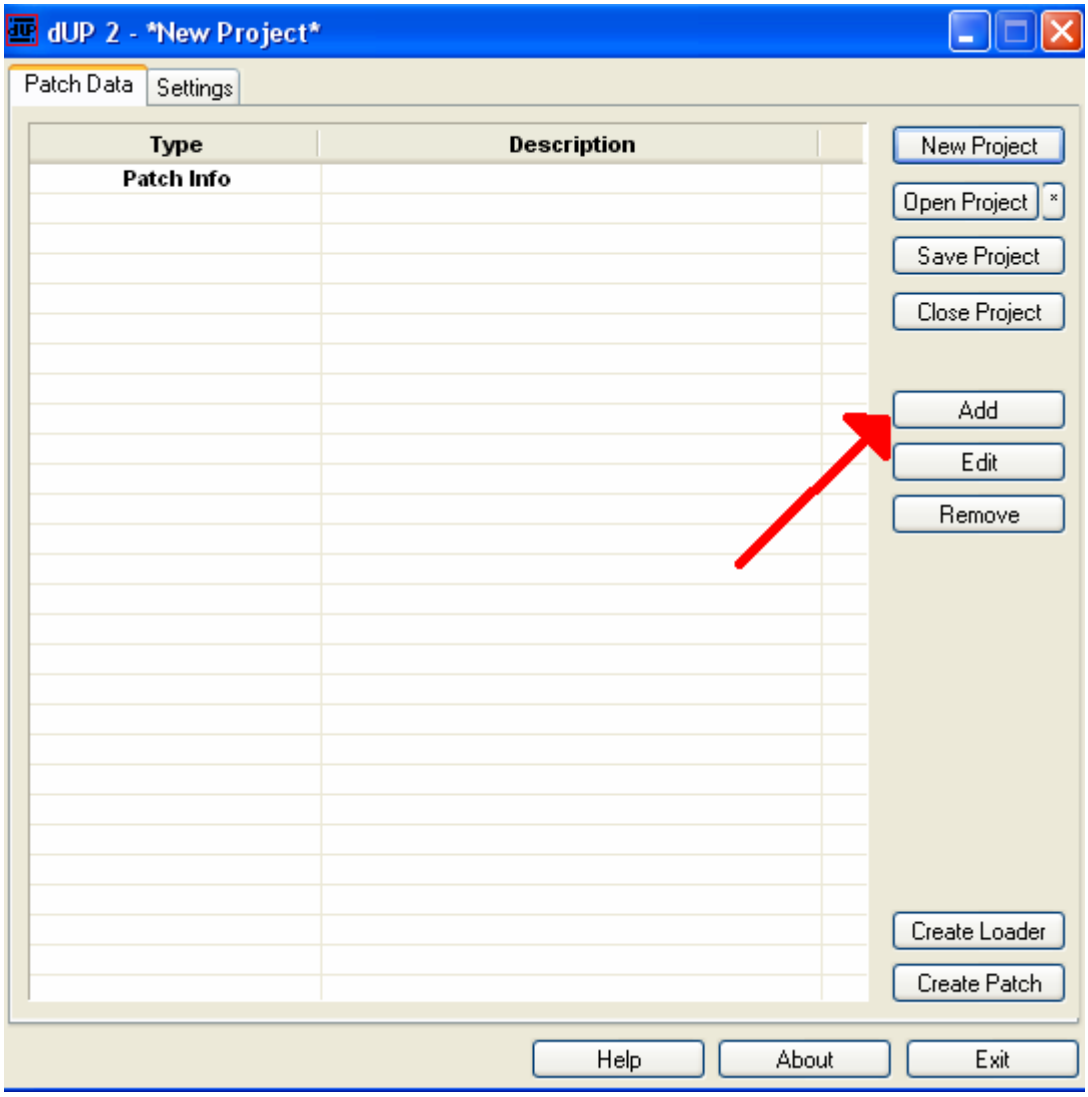


دوباره کلید OK را بزنید و گزینه Generate را بزنید و Loader را در همان پوشه ای که نرم افزار هدف قرار دارد،ذخیره کنید.حالا Loader را اجرا کنید.می بینید که Loader به خوبی توانسته بایت ها را دستکاری کند و برنامه رجیستر شده است.

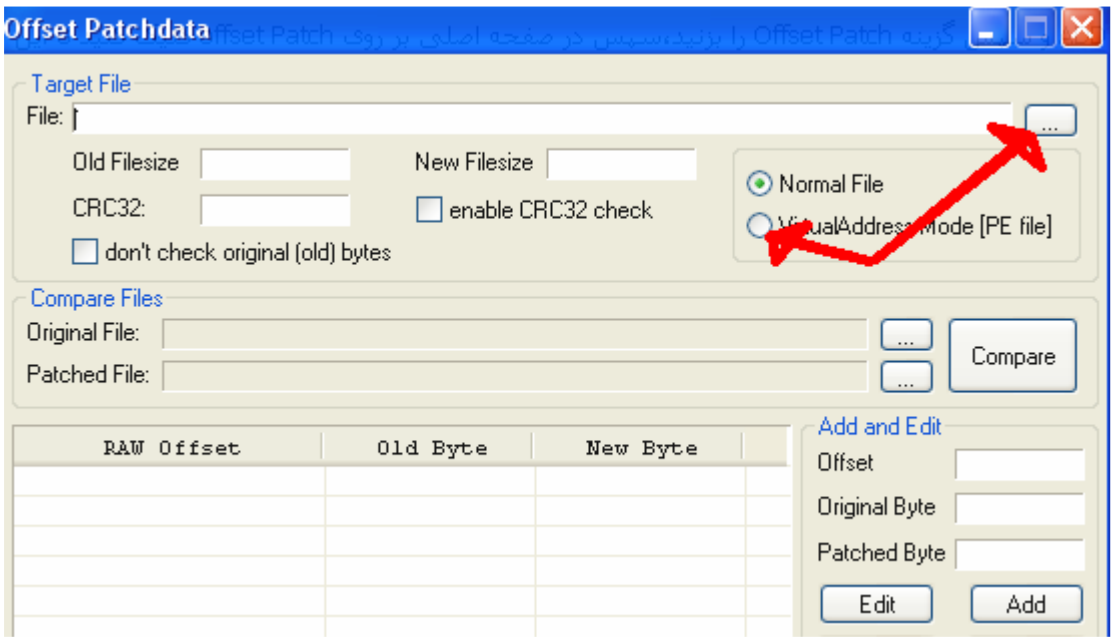


Diablo2oo2

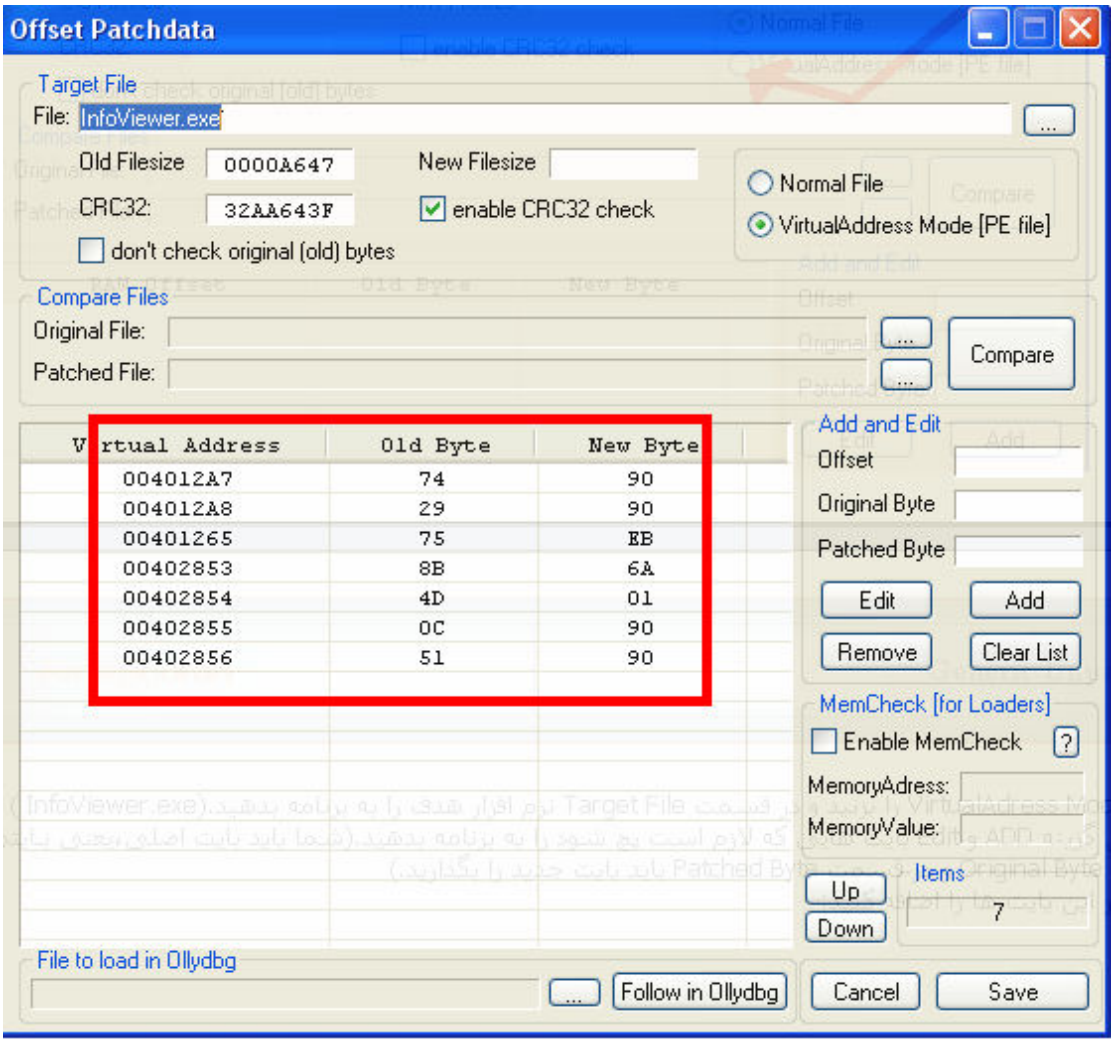
بیایید یک بار دیگر،اما این بار با نرم افزار Diablo2oo2 یک Loader بسازیم.برنامه را اجرا کنید و گزینه New Project و سپس گزینه Save را بزنید.



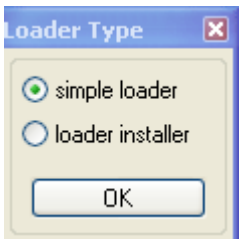
گزینه ADD و سپس گزینه Offset Patch را بزنید،سپس در صفحه اصلی بر روی offset Patch کلیک کنید تا این صفحه ظاهر شود:



تیک گزینه VirtualAdress 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:



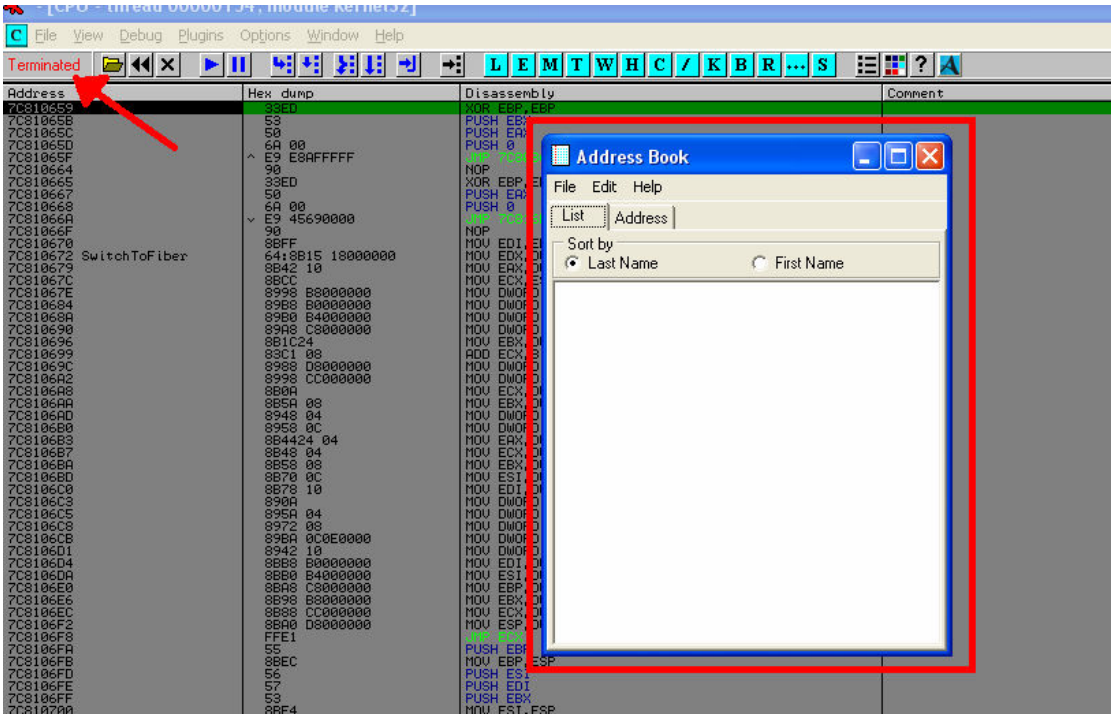
ممکن است سوال کنید چرا هفت بایت را دستکاری کردم؟...به خاطر اینکه یکبار Loader را با تغییر همان سه بایت ساختم.اما دیدم برنامه خطای Can't open the file را می دهد.به همین دلیل مجبور شدم چند بایت دیگر را هم پچ کنم.
خوب،گزینه Save را بزنید.و سپس گزینه Create Loader را بزنید:



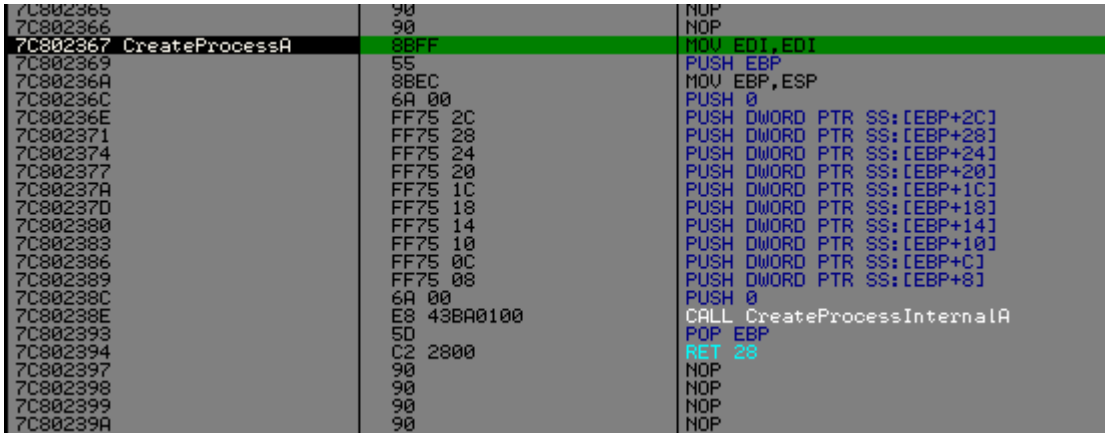
اگر می خواهید فقط یک Loader ساده بسازید،گزینه Simple loader را بزنید.ولی اگر می خواهید یک Loader بسازید که نام فایل اصلی را تغییر دهد و خودش را بگذارد جای فایلی اصلی،گزینه Loader installer را بزنید.
حالا کلید OK را بزنید و Loader خود را ذخیره کنید.و آن را اجرا کنید.می بینید که برنامه Crack شده است و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

! EP Exepack

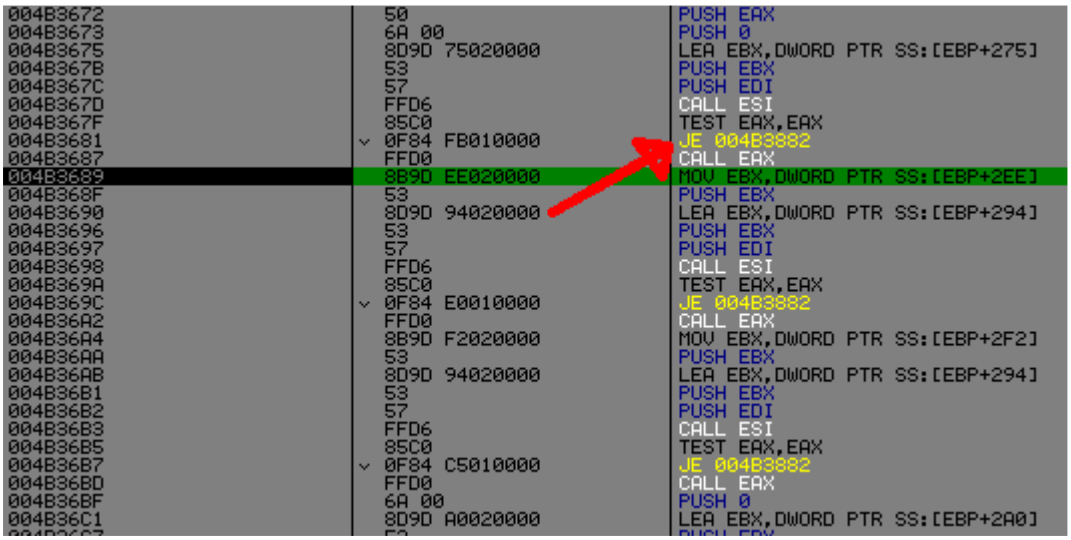
فایل مورد نظر را در OllyDBG باز کنید و کلید F9 را بزنید:



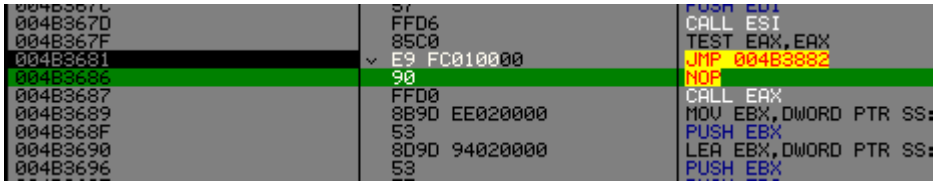
همانطور که می بینید برنامه به طور کامل اجرا شده...اما OllyDBG می گوید که برنامه Terminate شده است...خوب این یعنی چی؟...یعنی اینکه برنامه بعد از آنپک کردن فایل،یک پروسه جدید می سازد و در آن پروسه جدید،فایل را اجرا می کند...خوب،حالا چه کار کنیم؟...ما باید ببینیم که برنامه در کجا پروسه جدید را می سازد؟...بر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



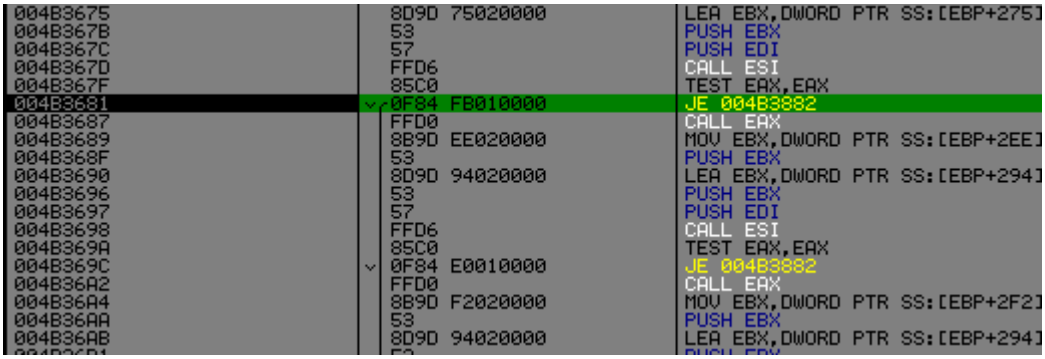
کلید ALT + F9 را بزنید.تا وارد کدهای فایل اصلی خودمان بشویم.



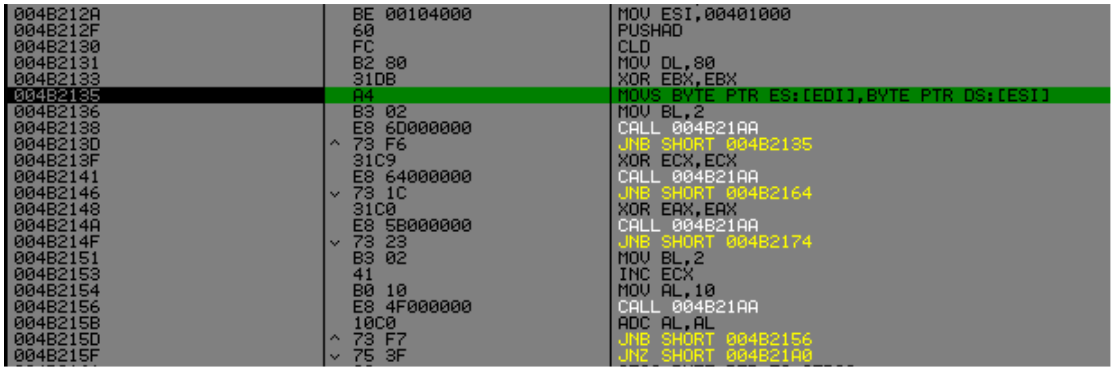
همانطور که می بینید پروسه ما اجرا شده،خط بالاتر از دستور Call EAX یک پرش شرطی وجود دارد که تصمیم گیرنده است.اگر این پرش انجام شود،برنامه در پروسه فعلی اجرا می شود و اگر پرش انجام نشود،یک پروسه جدید ساخته می شود...لذا بر روی این پرش Hardware breakpoint on execution بگذارید و برنامه را Restart کنید و کلید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



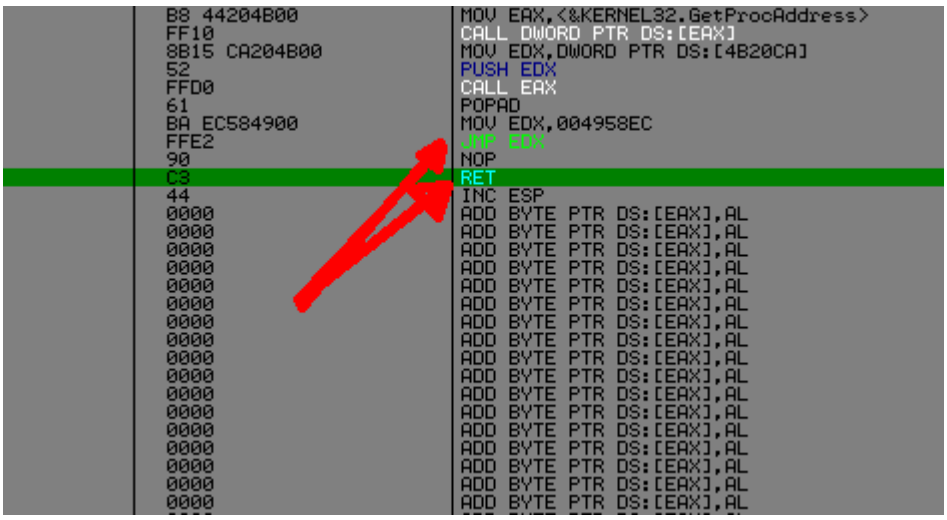
حالا کلید F9 را بزنید می بینید که برنامه این بار در درون پروسه فعلی اجرا شده است.
خوب حالا چه طور OEP را پیدا کنیم؟
برنامه را ری استارت کنید.و کلید F9 را بزنید:



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:



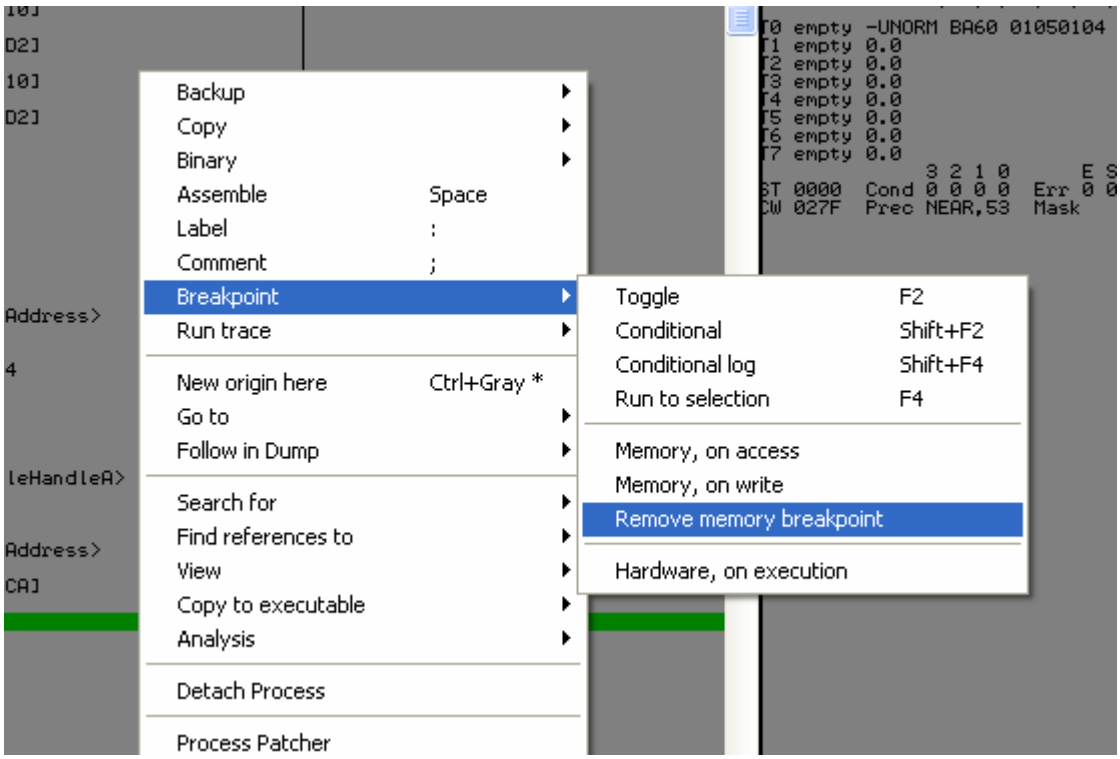
اینجا جایی هست که کدهای سکشن Code در حال Decrypt شدن هستند.وقتی کدها به طور کامل Decrypt شد به OEP می رویم.خوب کافیست بر روی دستورهایی که در آخر این کدها قرار دارند،Breakpoint بگذارید تا به OEP برسید...با موس به سمت پایین بروید تا به آخر این کدها برسید:



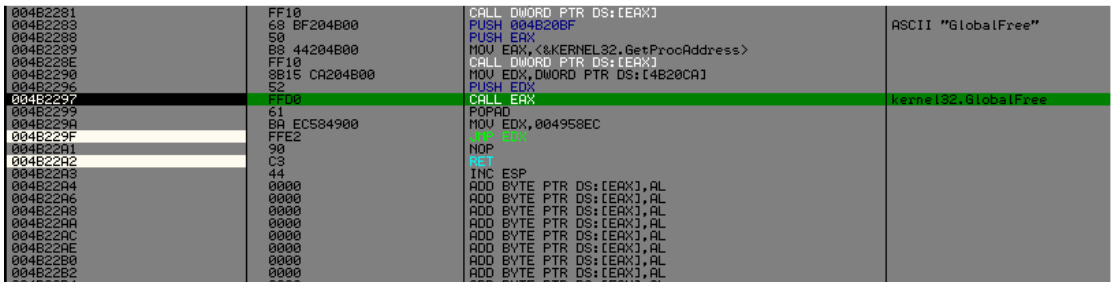
بر روی سه دستور JMP EDX و RET و CALL EAX ، Breakpoint بگذارید،چون این سه دستور،ممکن است ما را به OEP ببرند...از کجا فهمیدم؟...خوب،همانطور که قبلا گفتم چنین دستورهایی ممکن است ما را به OEP ببرند:

JMP Register (JMP Eax, JMP EBX, JMP EBP....)
Call Register (Call EAX, Call EBX, Call ECX...)
Return

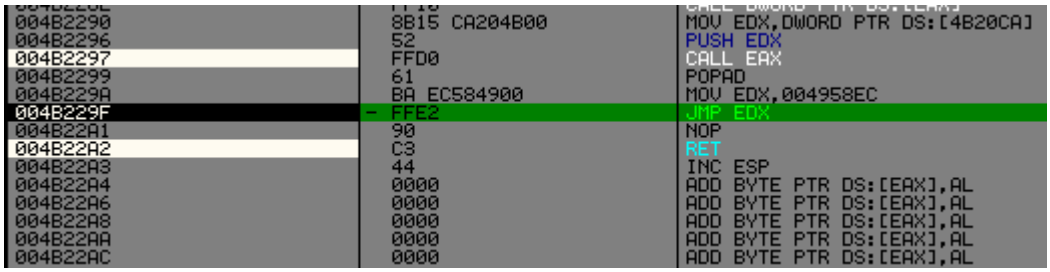
خوب...حالا Memory Breakpoint که قبلا گذاشته بودیم را پاک کنید،همانند تصویر:



حالا کلید F9 را بزنید:



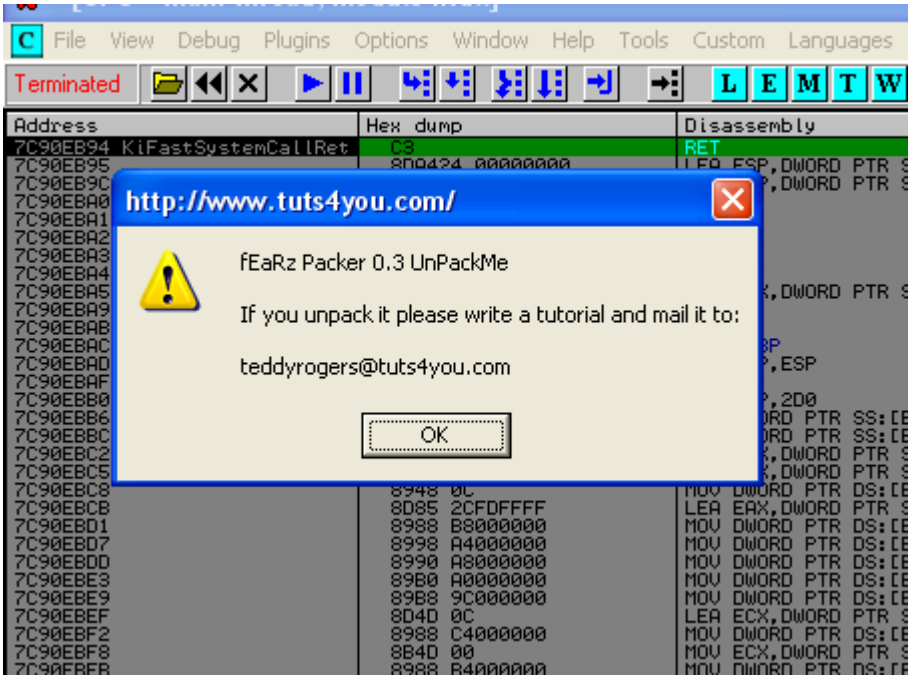
این CALL EAX که به تابع GlobalFree می رود یک بار دیگر F9 بزنید:



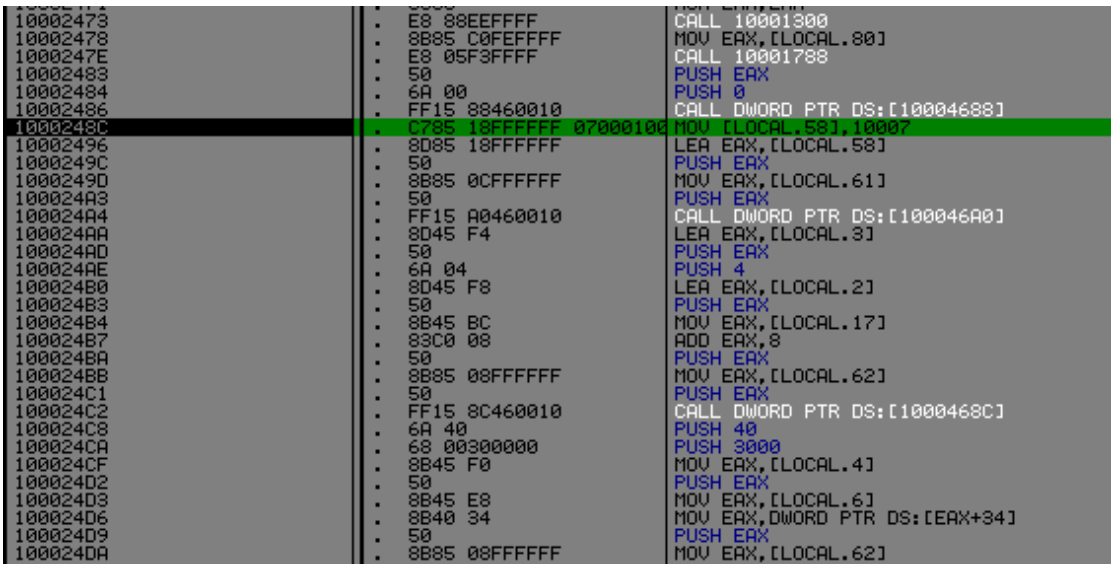
آها ☺ دستور JMP EDX ما را به OEP می برد.
بعد از یافتن OEP مشکلی به وجود نمی آید،دامپ می گیرید و فیکس می کنید.

Fearz Packer

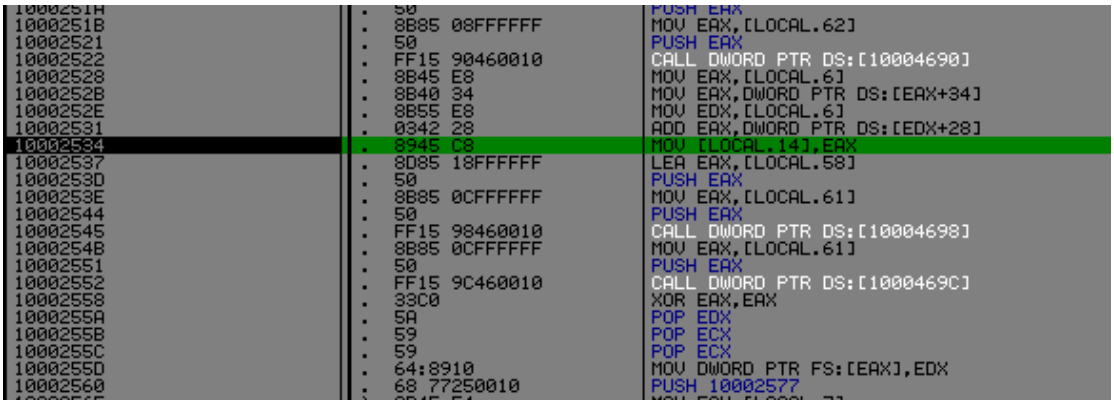
فایل مورد نظر را در OllyDBG باز کنید،و کلید F9 را بزنید تا برنامه اجرا شود:



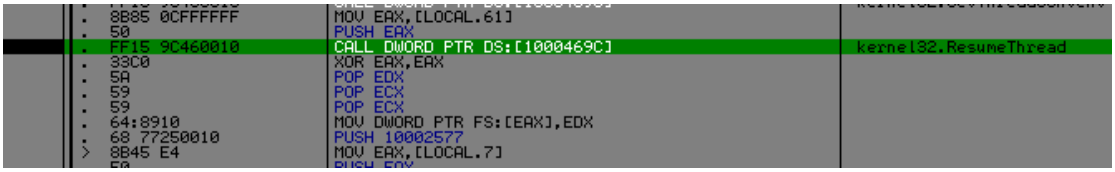
همانطور که می بینید برنامه در داخل OllyDBG بسته شده،اما پروسه همچنان اجراست.این نشان می دهد که یک پروسه جدید ساخته شده و فایل ما در آن پروسه اجرا شده است.خوب برنامه را Restart کرده،بر روی تابع CreateProcessA بریک پوینت بگذارید.(برای این کار در پلاگین Command Bar بنویسید : BP CreateProcessA)
و بعد برنامه 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:



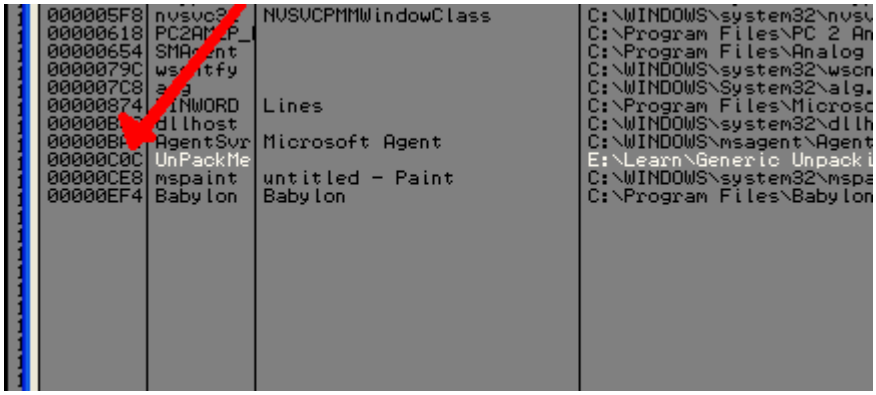
برنامه را با کلید F8 ادامه دهید تا به این خط برسیم:



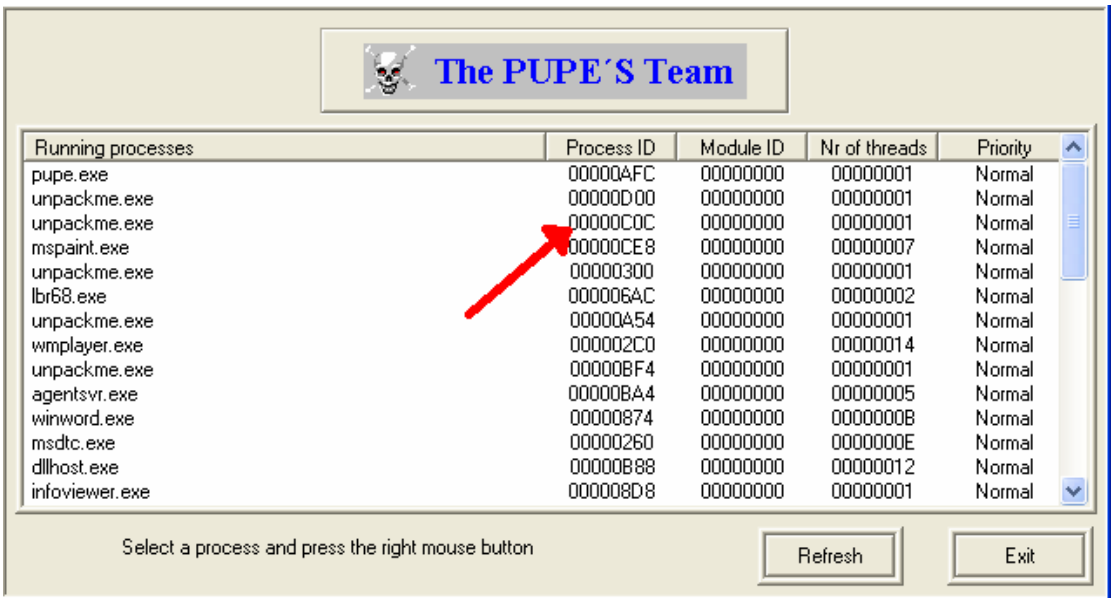
الان مقدار رجیستر EAX همان OEP ماست.(در واقع Entry Point پروسه جدید)
برنامه را ادامه دهید تا به این خط برسید:



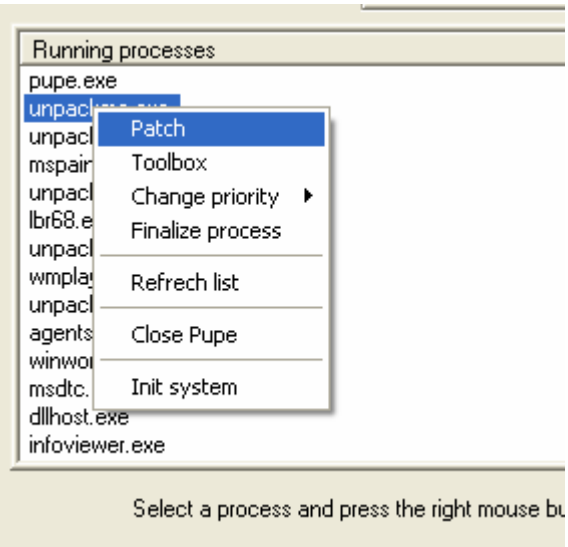
با اجرا شدن تابع Resume Thread برنامه اجرا می شود.خوب،پس اگر یک بار دیگر F8 بزنید،برنامه اجرا می شود...اما ما باید روی OEP پروسه جدیدمان متوقف شویم تا بتوانیم از فایلمان دامپ بگیریم و بعد هم دامپ را فیکس کنیم...اما اگر این دستور(Resume Thread) اجرا شود،برنامه به طور کامل اجرا می شود...چه کار باید بکنیم؟...ما 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 :



همانطور که می بینید پروسه فایل من برابر C0C است.(البته در کامپیوتر من)
برای متوقف کردن پروسه جدید از نرم افزار PUPE استفاده می کنیم.این برنامه را باز کنید:

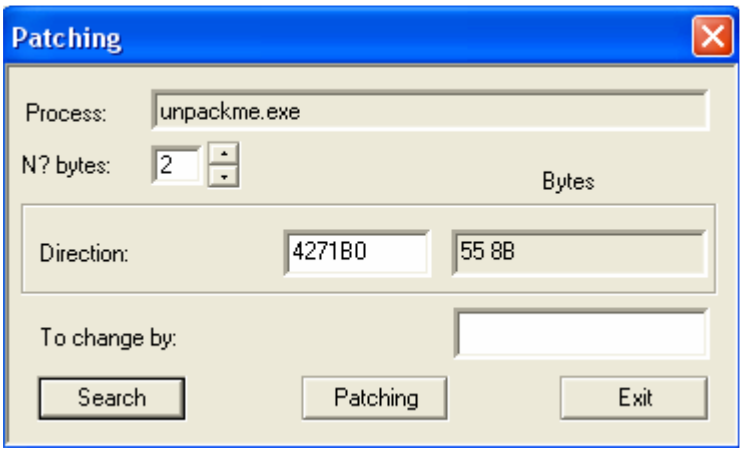


همانطور که می بینید دو پروسه Unpackme داریم.یکی که پروسه ای است که در OllyDBG قرار دارد.(که Process ID آن برابر C0C است) و دیگری که PID آن برابر D00 است.من باید پروسه جدید را متوقف کنم.لذا بر روی پروسه ای که با تابع CreateProcessA ایجاد شده(همانی که PID آن در کامپیوتر من D00 است) کلیک راست کرده و گزینه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

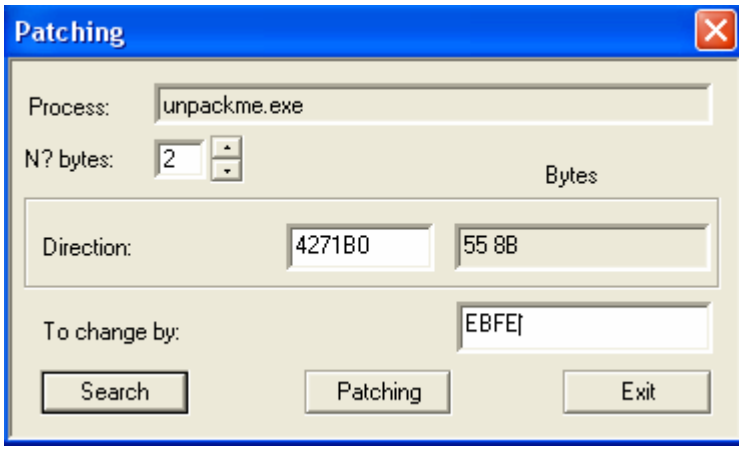


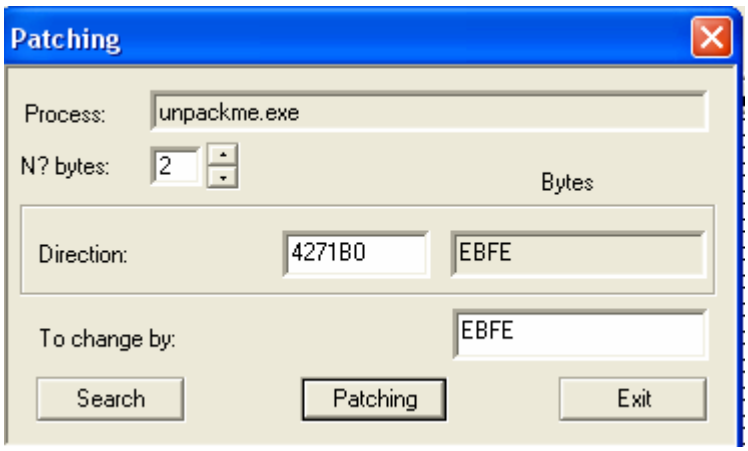
خوب...حالا ما باید کاری کنیم که پروسه جدید ما از Entry point خودش تکان نخورد و همانجا بماند؟...چه طوری؟...من می توانم از دستور infinite loop استفاده کنم.این دستور یک خط را به تعداد نامحدود اجرا می کند و در واقع برنامه در خطی که این دستور را دارد متوقف می شود.معادل opcode این دستور EBFE است.لذا من باید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

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

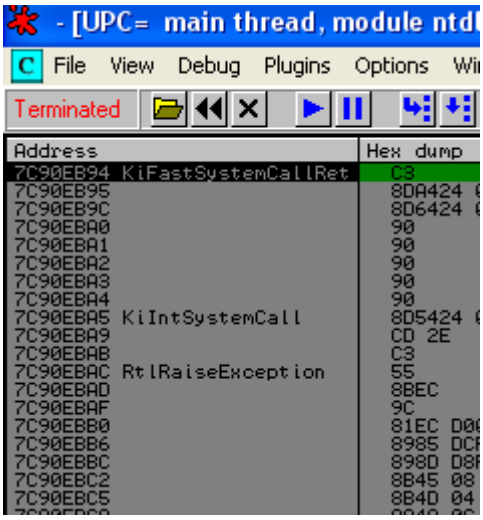


همانطور که می بینید در قسمت Bytes نوشته شده 558B یعنی هم اکنون در بایت 4271B0 مقدار 558B قرار دارد.در TextBox زیر Bytes بنویسید: EBFE و سپس گزینه Patching را بزنید:

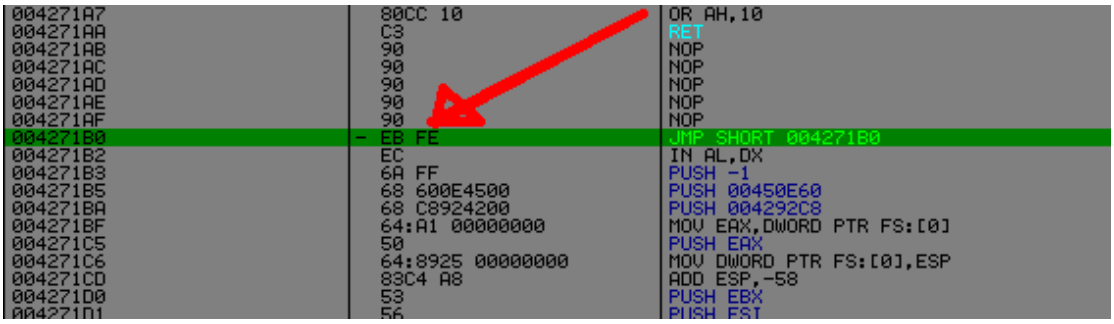




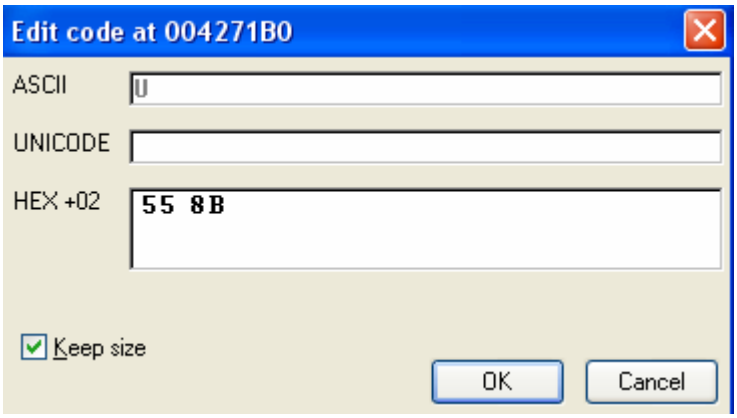
حالا برنامه PUPE را ببندید و به OllyDBG برگردید و کلید F9 را بزنید تا برنامه به طور کامل اجرا شود:



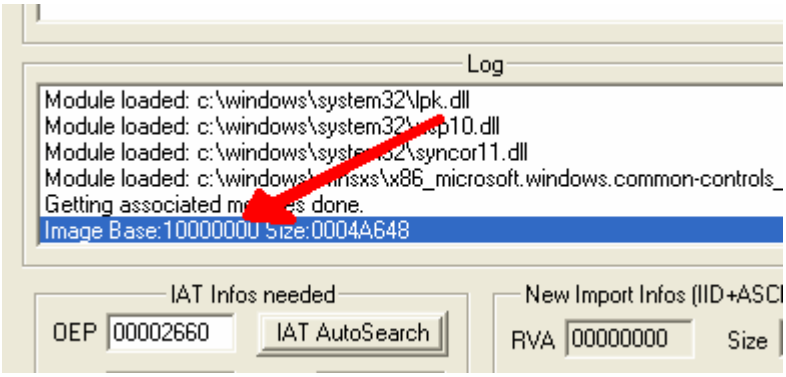
همانطور که می بینید برنامه Terminate شده اما پیغامی داده نشده.حالا به پروسه جدید خودمان Attach می کنیم.در منوی File گزینه Attach را بزنید.و پروسه ی جدید(که در کامپیوتر من PID آن D00 بود)را انتخاب کنید و گزینه Attach را بزنید و بعد به خط 4271B0 بروید،همانطور که می بینید در این خط بایت های EBFE قرار دارد:



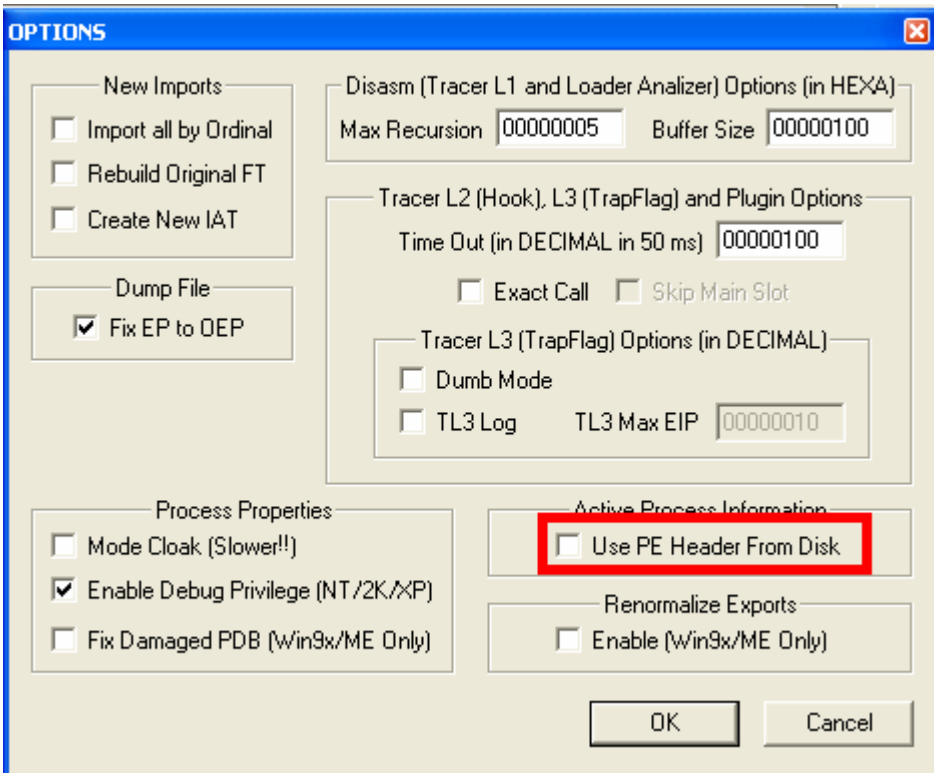
خوب،حالا باید این بایت ها را همانند قبلش کنیم.قبلا این بایت ها بوده : 558B
پس کلیدهای CTRL + E را بزنید و همانند تصویر بایت ها را به حالت قبل برگردانید:



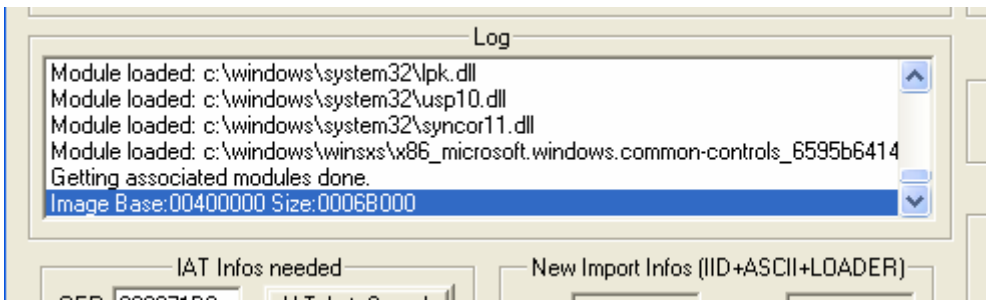
کلید OK را بزنید و بعد هم بر روی OEP برنامه کلیک راست کرده و گزینه new origin here را بزنید.حالا هم از فایل دامپ بگیرید.(ترجیحا با PeTools این کار را بکنید.)
حالا ImportREC را باز کنید.و پروسه مورد نظر را در ImportREC باز کنید:



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:



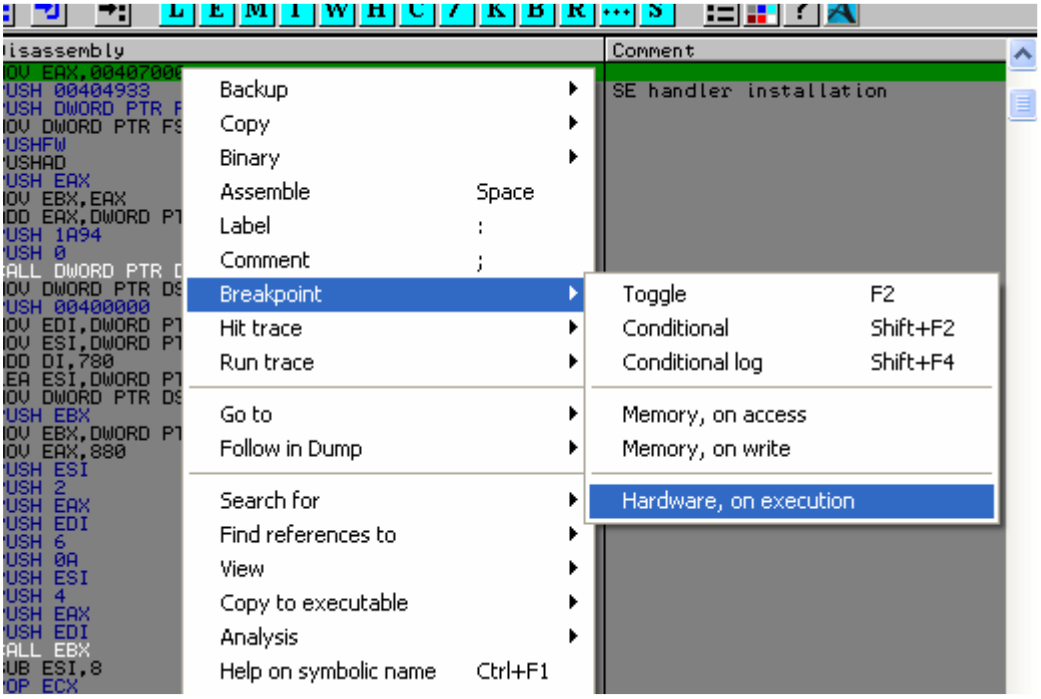
کلید OK را بزنید و دوباره پروسه را در ImportREC باز کنید:



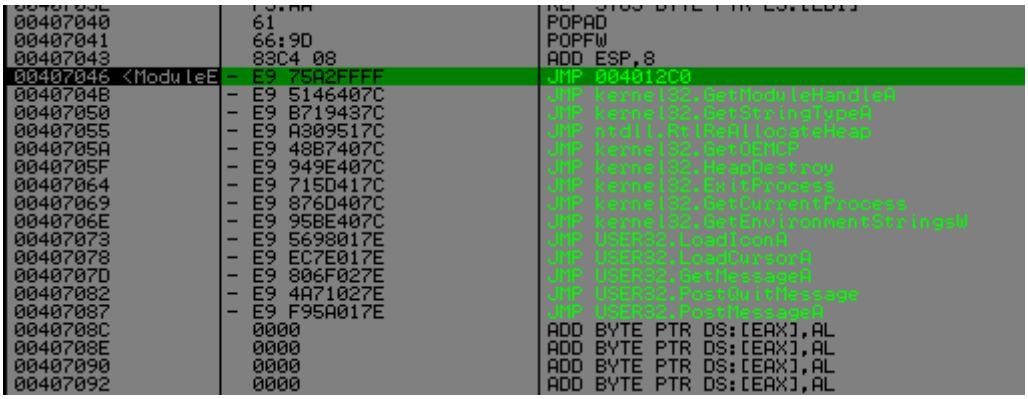
همانطور که می بینید اینبار ImageBase درست نوشته شده ☺
خوب...از این به بعد دیگر مشکلی نیست،کافیست OEP را بدهید و فایل خود را فیکس کنید.

PeTite

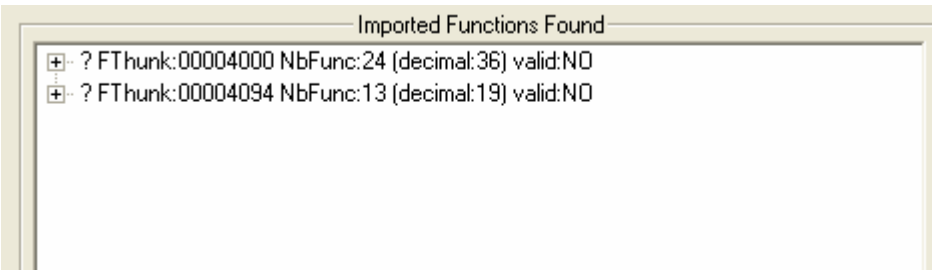
فایل مورد نظر را در OllyDBG باز کنید.برای یافتن OEP یک روش خاص برای PeTite وجود دارد.آن هم اینکه چون PeTite از Self-modifying (یعنی کدهای خودش را تغییر می دهد تا کار آنالیز پکر را سخت کند.) استفاده می کند و دستوری که ما را به OEP می برد.درست در Entry point فایل قرار می گیرد.ما می توانیم با hardware breakpoint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



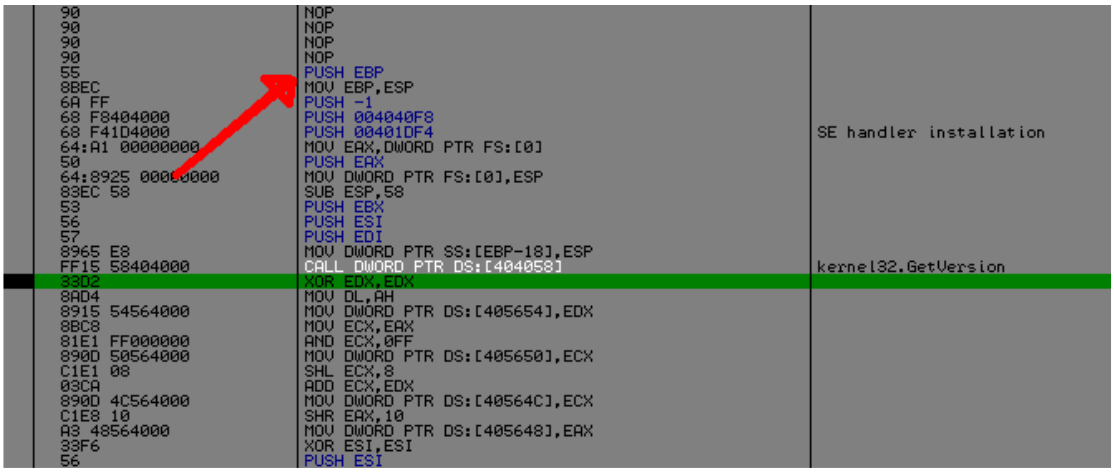
می بینید؟...اینجا همان Entry point ماست.اما کدهایش تغییر کرده.این پرش هم درست به OEP می رود.یکبار کلید F8 را بزنید تا به OEP برسید.
حالا ImportREC را بزنید.OEP را به برنامه بدهید.و سایر مراحل لازم را انجام دهید.می بینید که چند تابع ما Invalid می باشد(Redirect شده است.):



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:

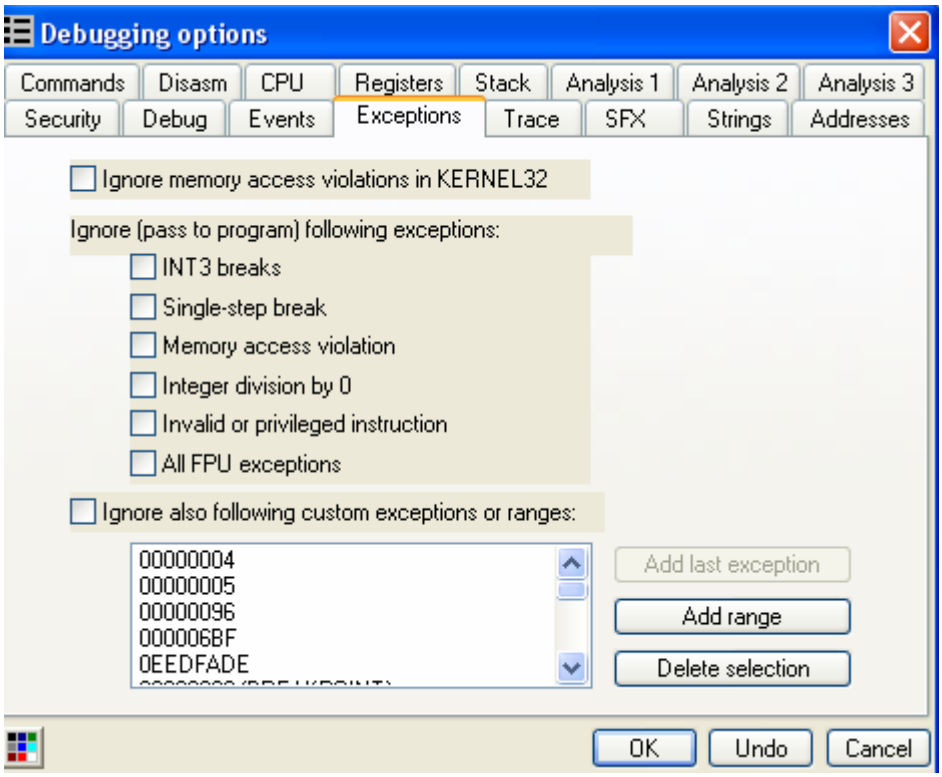
برای یافتن OEP می توانیم از ساختار زبان برنامه نویسی استفاده کنیم.بر روی تابع GetVersion یک Breakpoint بگذارید.(در CommandBar تایپ کنید: BP GetVersion و کلید Enter را بزنید)
برنامه را با کلید F9 اجرا کنید تا در تابع GetVersion متوقف شوید.کلید ALT + F9 را بزنید تا وارد کدهای فایل بشوید:



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

PCGuard

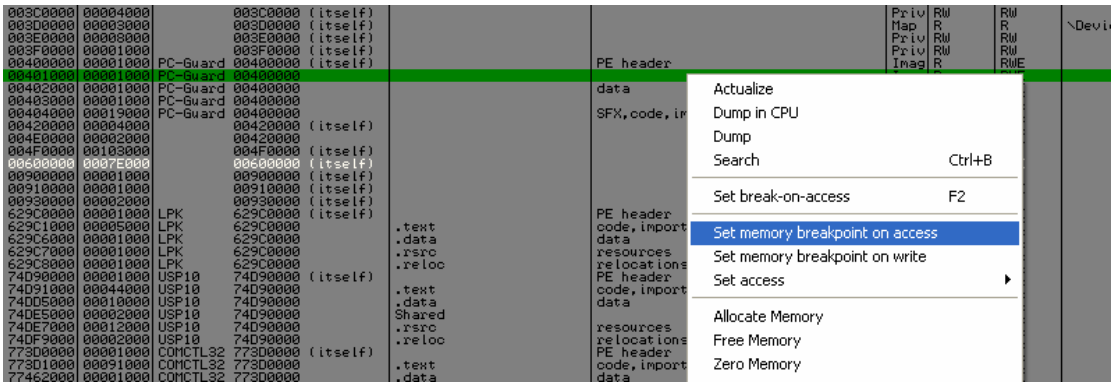
فایل مورد نظر را در OllyDBG باز کنید.(توجه کنید که این فایل محدودیت اجرا دارد.یعنی بعد از چند بار اجرا شدن دیگر در سیستم شما اجرا نمی شود.برای حل این مشکل می توانید از نرم افزارهایی مثل Trial-Reset استفاده کنید.)
برای یافتن OEP می توانیم از Memory Breakpoint استفاده کنیم.ابتدا باید آخرین خطایی که در 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.



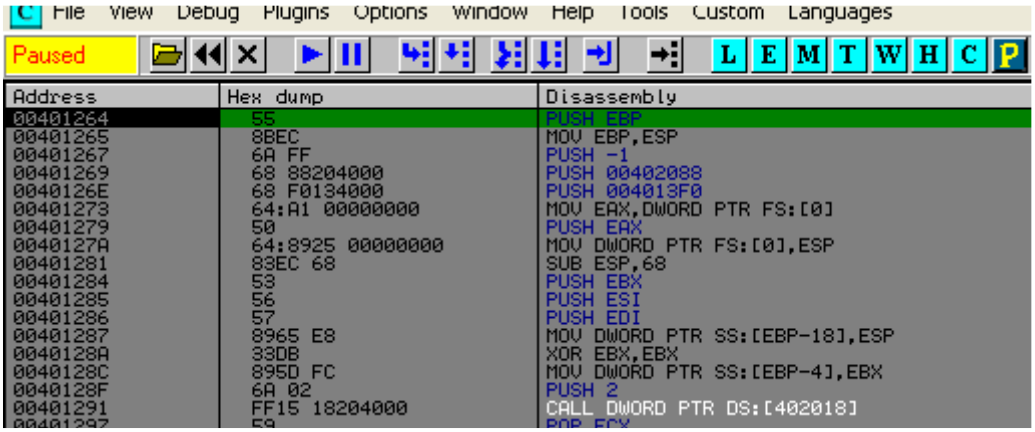
حالا کلید OK را بزنید و با Shift + F9 برنامه را اجرا کنید.چون باید آخرین خطایی که در برنامه اتفاق می افتد را پیدا کنیم.یکبار تعداد خطاهایی که در برنامه اتفاق می افتد را بشمارید.من ۹ بار کلید Shift + F9 را زدم،تا برنامه اجرا شد.
برنامه را ری استارت کنید و این بار ۸ بار کلید Shift + F9 را بزنید تا به آخرین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

```
0041B04D    EB 01       JMP SHORT 0041B050
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    D5 9D       AAD 9D
```

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



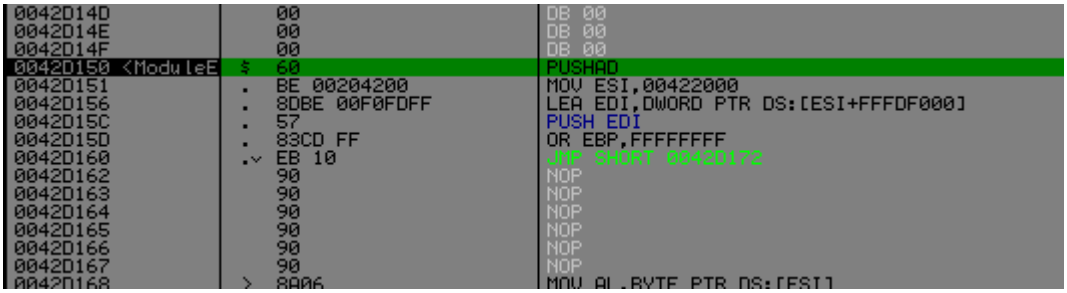
حالا هم کلید F9 را بزنید:



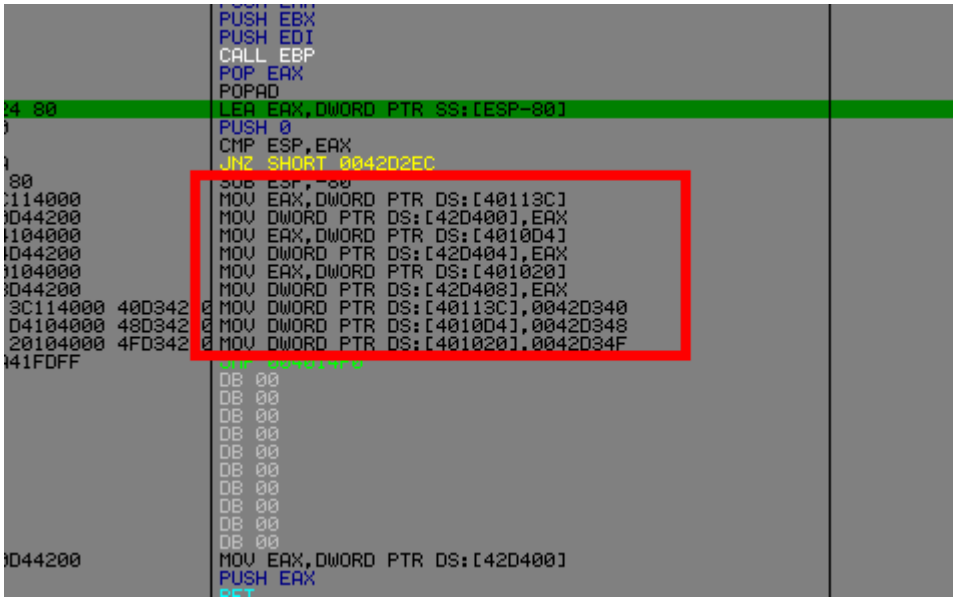
همانطور که می بینید وارد سکشن اصلی برنامه شدیم و به OEP رسیدیم ☺
خوب...بعد از یافتن OEP دیگر کار سختی در پیش ندارید.دامپ می گیرد و فیکس می کنید.

Gooran_Ware UnpackMe#1

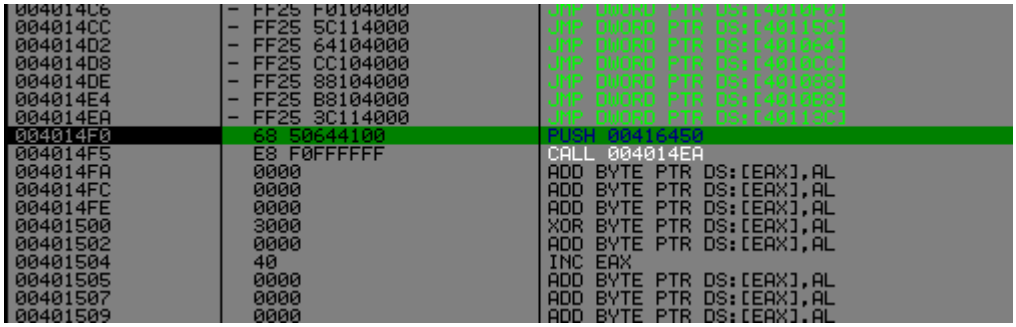
فایل مورد نظر را در OllyDBG باز کنید:



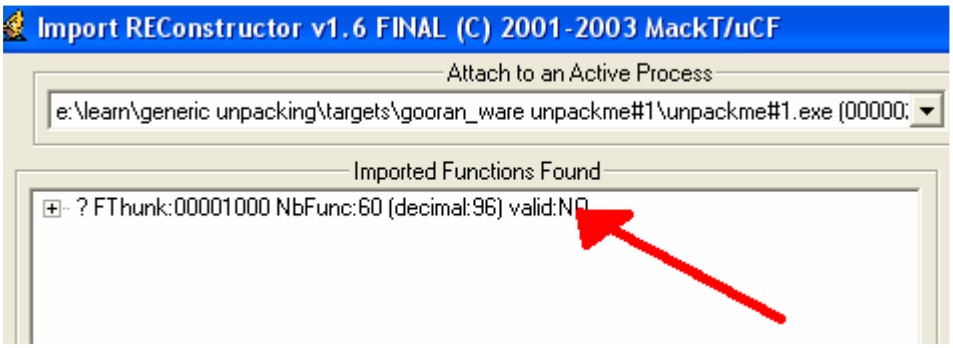
چون فایل با UPX پک شده است،به راحتی می توان از طریق رجیستر ESP ،به OEP رسید.
کلید F8 را بزنید تا مقدار رجیستر ESP تغییر کند.بعد بر روی مقدار رجیستر ESP یک Hardware breakpoint on acess بگذارید و کلید F9 را بزنید تا به این آدرس برسید:



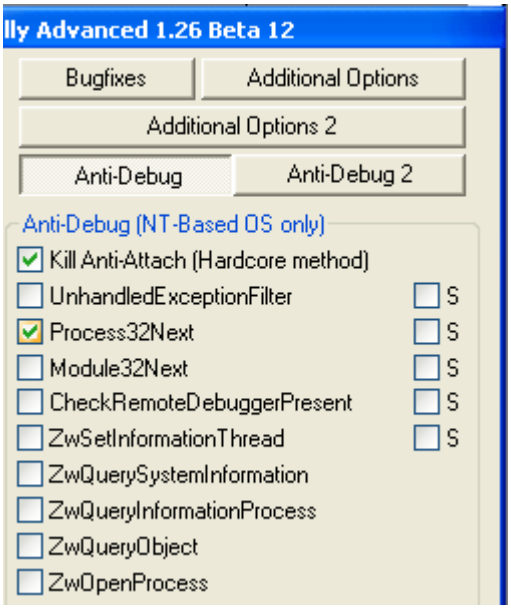
اگر برنامه را با کلید F8 به دقت اجرا کنید،متوجه خواهید شد که این کدها قصد دارند قسمتی از توابع ما را Redirect کنند.ولی مشکلی نیست.شما با F8 ادامه دهید تا به دستور JMP 4014F0 برسید.پرشی که ما را به OEP می برد:



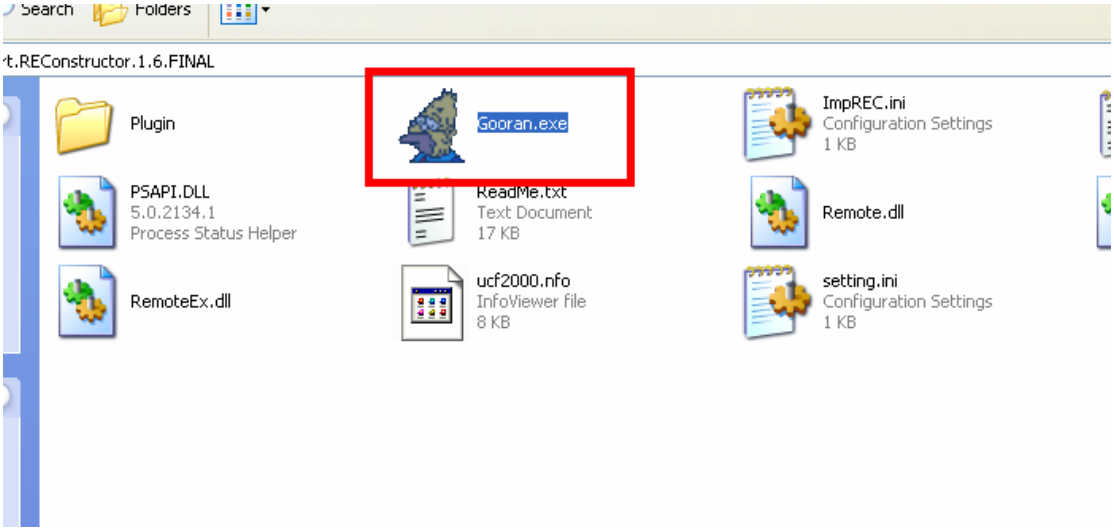
خوب،حالا از فایل دامپ بگیرید و ImportREC را باز کنید:



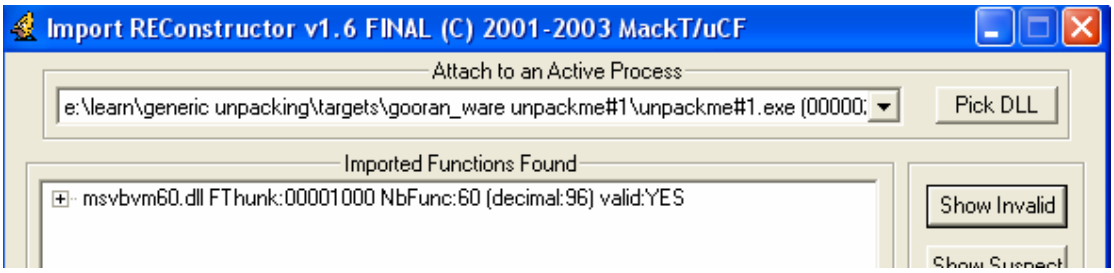
همانطور که می بینید چند تا از توابع ما Invalid می باشد.برای یافتن این توابع می توانیم از قابلیت Trap Flag استفاده کنیم.ابتدا برنامه را در داخل OllyDBG به طور کامل باز کنید.با اجرا شدن برنامه ممکن است OllyDBG شما بسته شود.برای حل این مشکل می توانید از پلاگین OllyAdvanced استفاده کنید.(این پلاگین را در 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:



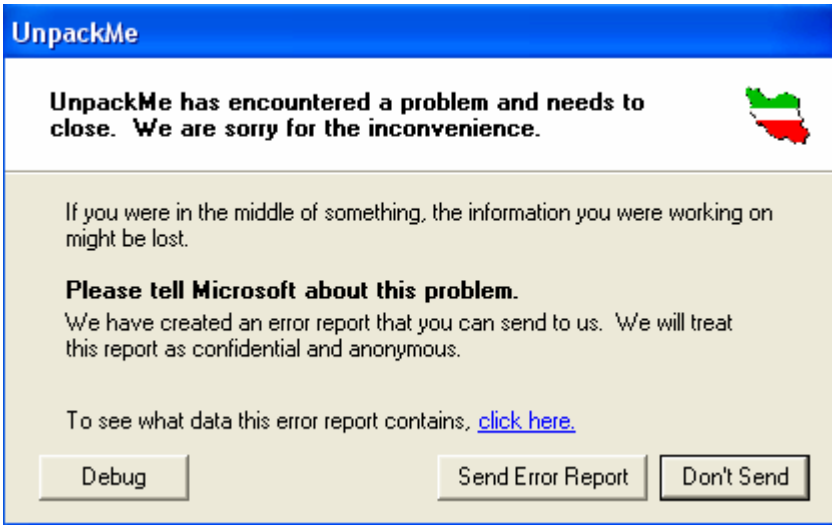
البته از پلاگین های دیگر هم می توان برای از بین بردن این تابع استفاده کرد.حالا دیگر برنامه بدون هیچ مشکلی در OllyDBG اجرا می شود.اگر UnpackMe برنامه ImportREC را هم می بندد.خیلی راحت می توانید نام فایل ImportREC را عوض کنید.مثلا بگذارید Gooran.exe و بعد ImportREC را اجرا کنید:



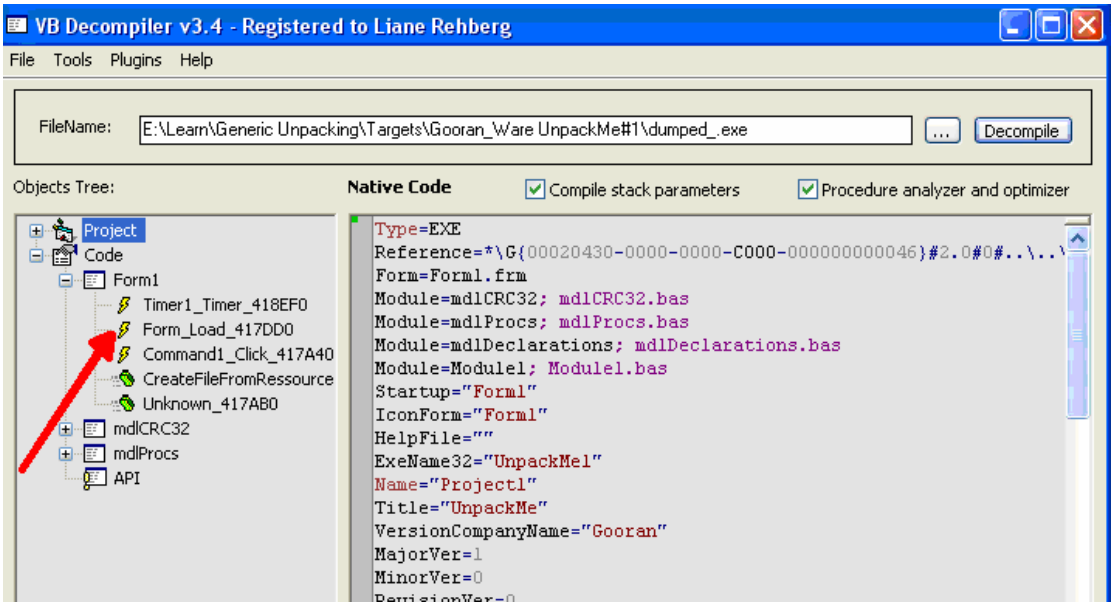
حالا دیگر برنامه قادر به شناسایی ImportREC نیست و از ImportREC هم می توانید به راحتی استفاده کنید.حالا دوباره ImportREC را باز کنید.OEP را بدهید و سایر مراحل را انجام دهید.حالا گزینه Show invalid را بزنید و بر روی توابع Redirect شده کلیک راست کرده و گزینه Trace Level3 (Trap flag) را بزنید تا تمامی توابع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



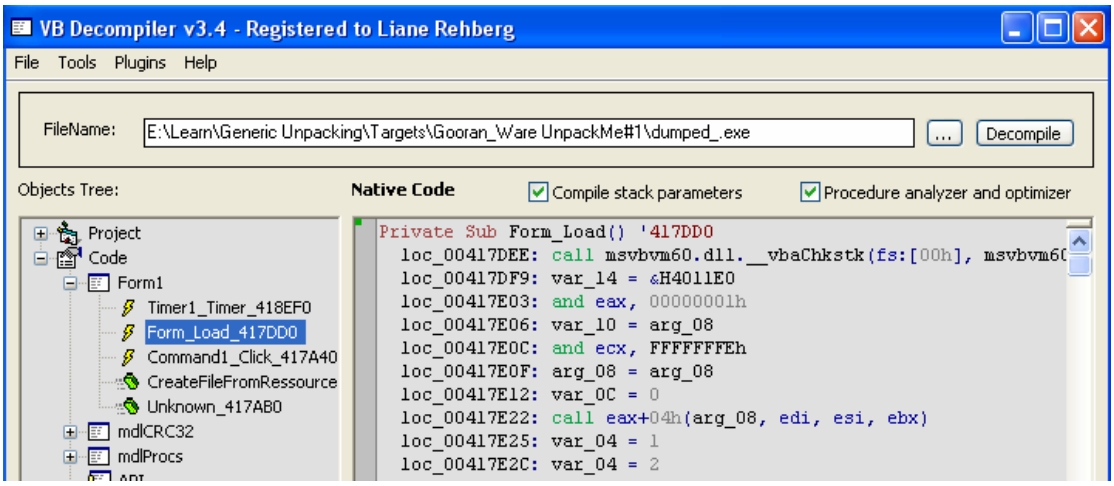
حالا فایل دامپ خود را فیکس کنید و فایل آنپک شده را اجرا کنید:



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.):

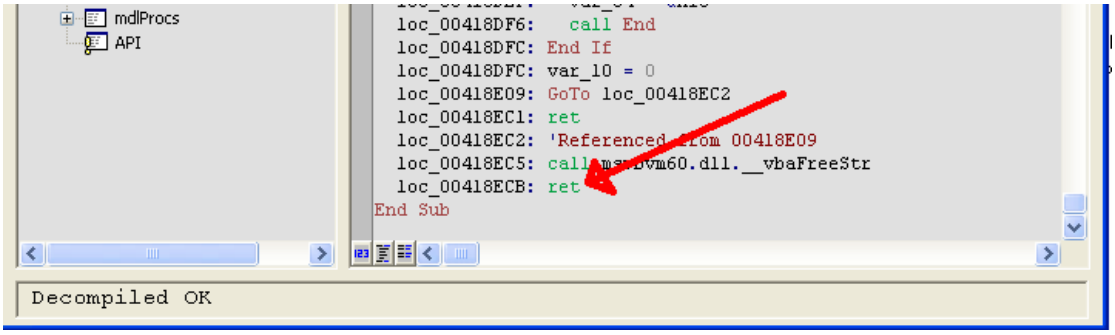


خوب،حالا در تصویر بالا بر روی گزینه Form_Load_417DD0 (در قسمت Object Tree) دو بار کلیک کنید:



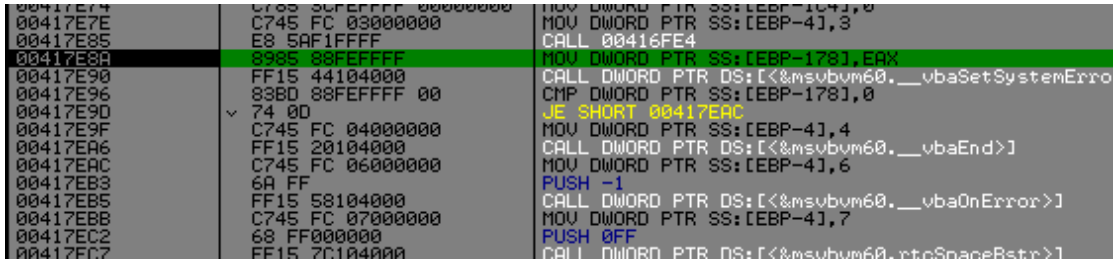
Form_Load دستوراتی است که در هنگام اجرا شدن یک فرم در برنامه های VB اجرا می شود.چون فایل آنپک شده ما در هنگام اجرا شدن خطا می داد.لذا ممکن است دلیل اجرا نشدن برنامه در همین تابع باشد.

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:

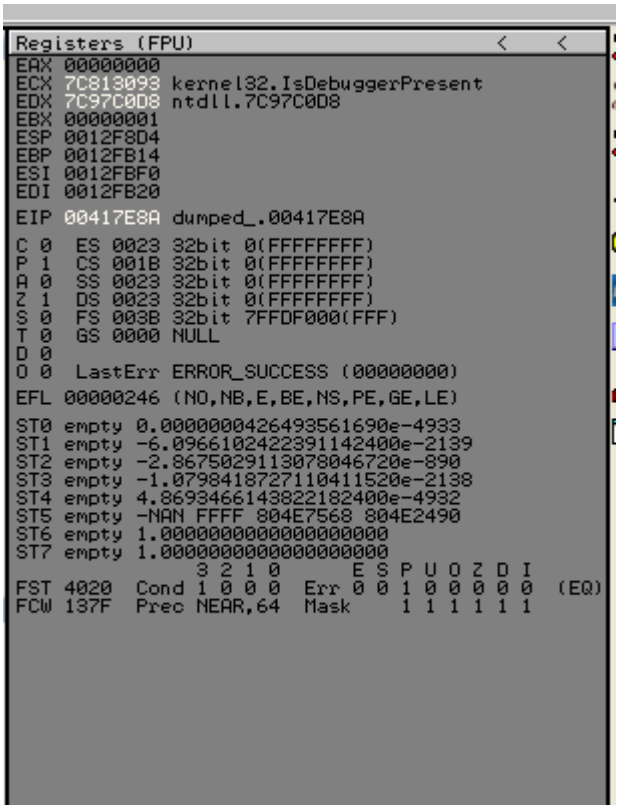


خط 418ECB می باشد.حالا فایل آنپک شده را با OllyDBG باز کنید و بر روی خطوط ابتدایی و انتهایی Form_load، Breakpoint بگذارید.و بعد برنامه را با F9 اجرا کنید:

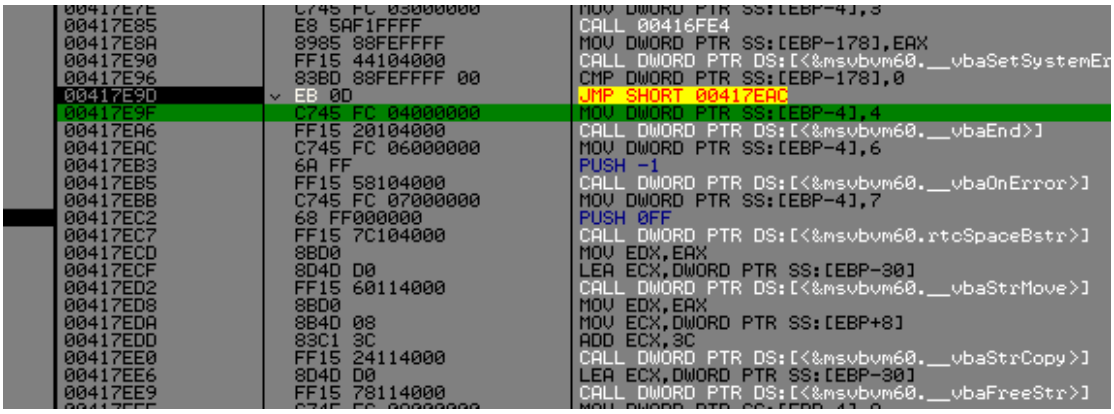
اینجا ابتدای تابع Form_load ماست.برنامه را با F8 ادامه دهید تا به این خط برسید:



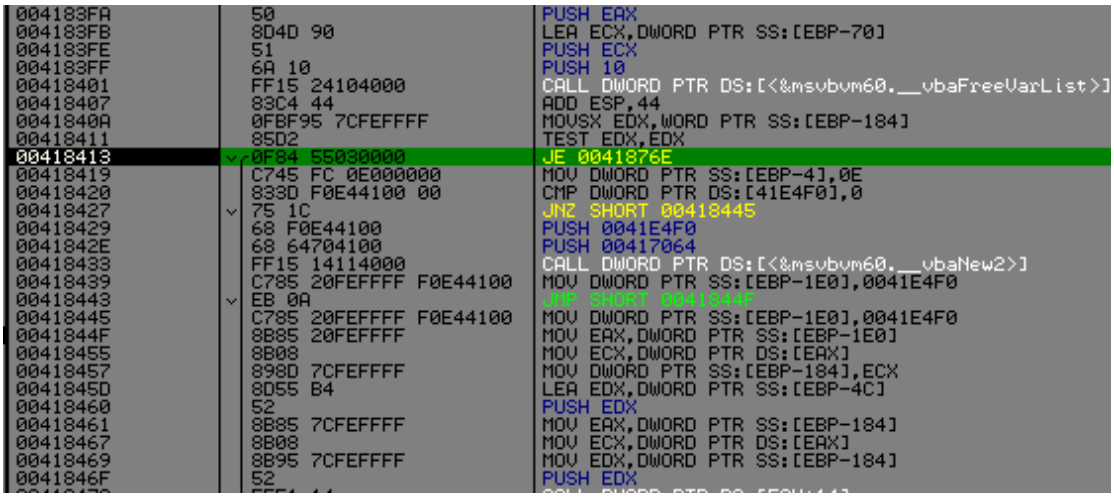
الان یک نگاهی به رجیسترها بیندازید:



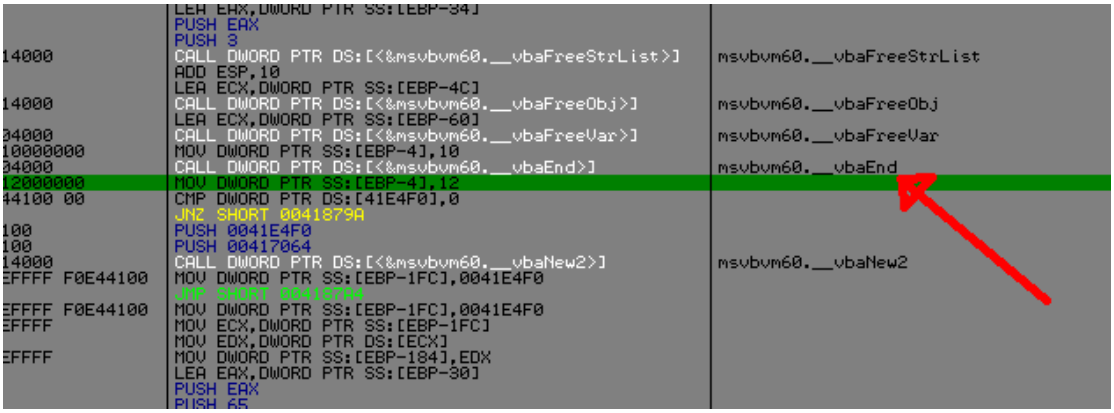
مقدار رجیستر ECX برابر IsDebuggerPresent است.این یعنی برنامه در این خطوط می خواهد این تابع را اجرا کند.پرشی که در خط 417E9D قرار دارد،تصمیم گیرنده است.یعنی اگر این پرش انجام شود.نشان می دهد که دیباگر وجود ندارد و اگر این پرش انجام نشود یعنی دیباگر وجود دارد.آنوقت تابع vbaEnd__ اجرا می شود و برنامه بسته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



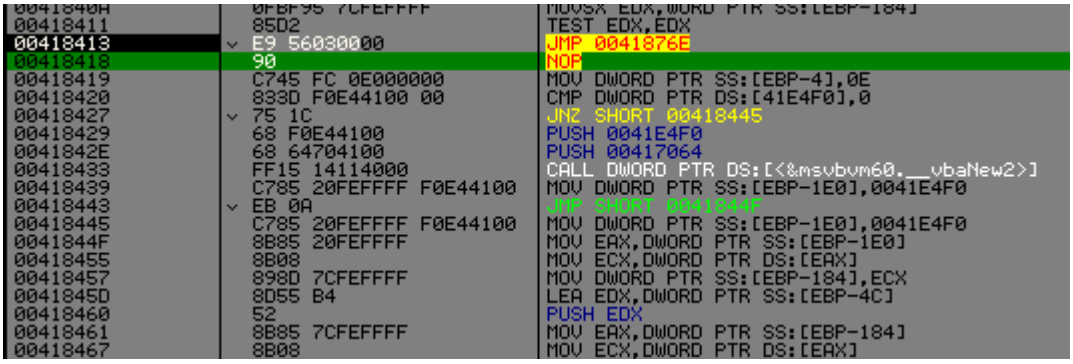
خوب،باز هم برنامه را با F8 ادامه دهید،تا به اینجا برسید:



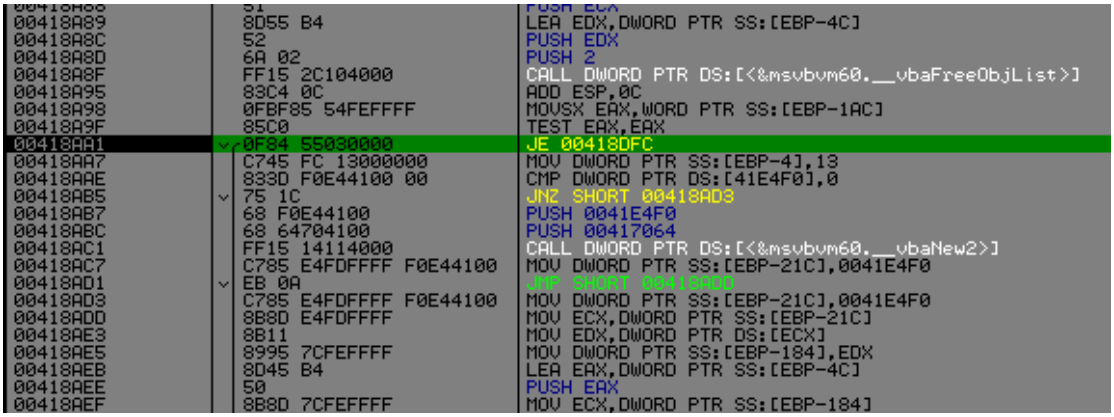
این یک پرش بسیار بلند است.بر روی آن کلید Enter را بزنید تا ببینیم به کجا می رود:



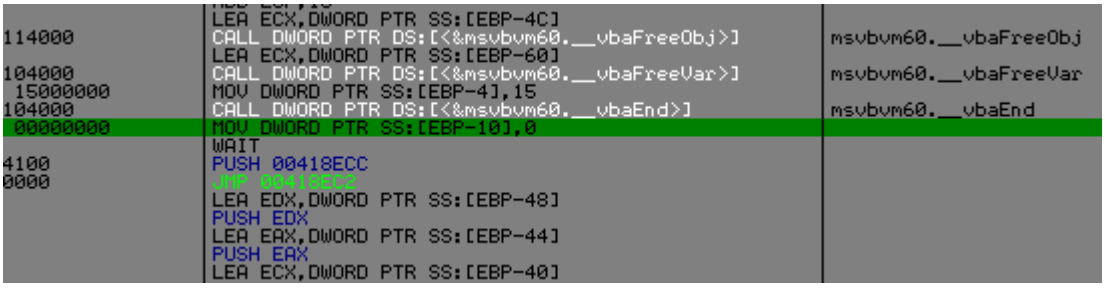
می بینید؟...این پرش تابع vbaEnd را رد می کند.یعنی این پرش باید حتما انجام شود،چرا که اگر انجام نشود به تابع vbaEnd برمی خوریم و پروسه بسته می شود.پس این پرش را هم JMP کنید و برنامه را با F8 ادامه دهید:



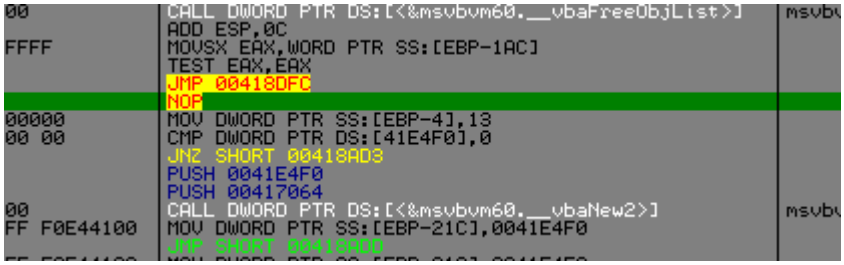
آنقدر با F8 جلو بروید تا به این خط برسید:



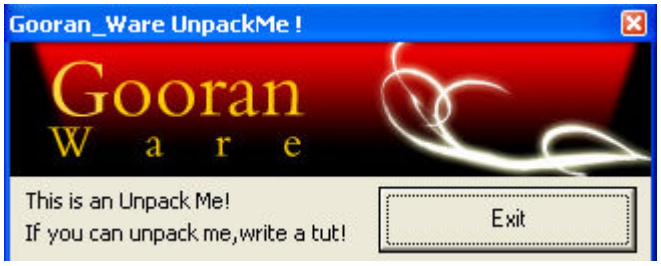
این هم یک پرش دیگر،باز هم بر روی آن کلید Enter را بزنید:



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



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.

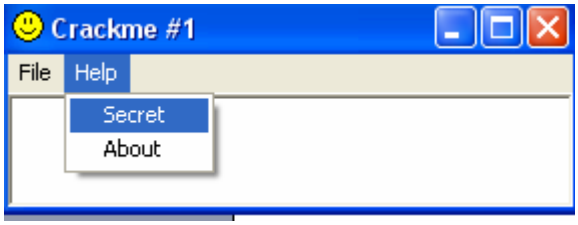
حالا تغییرات را ذخیره کنید.تا کار آنپک این فایل هم تمام شود.

Inline Patching (Level 2)

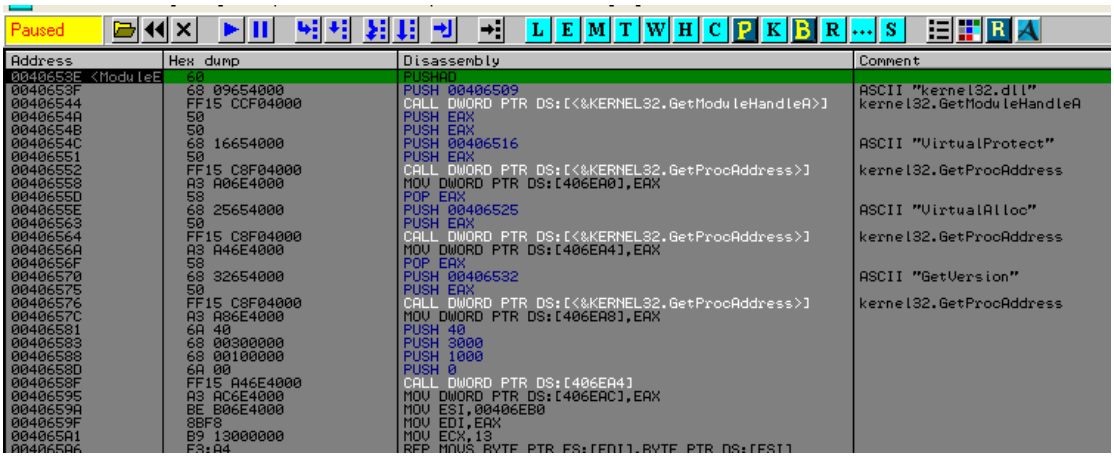
من در بخش ساختار زبان برنامه نویسی گفتم معمولا بعد از نزدیکی های Entry point در زبان C++ تابع GetVersion (یا GetVersionExa در نسخه های جدیدتر) می آید.در نزدیکی های Entry point در زبان دلفی معمولا GetModlueHandleA و در زیر Entry point در VB تابع ThunRTMain می آید...خوب،حالا بیایید فرض کنیم که یک فایل پک شده داریم که در C++ نوشته شده است...من می توانم بیام و تابع GetVersion در فایل Kernel32.dll را دستکاری کنم(Hook کنم)...یعنی بیام 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 ازبین 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:

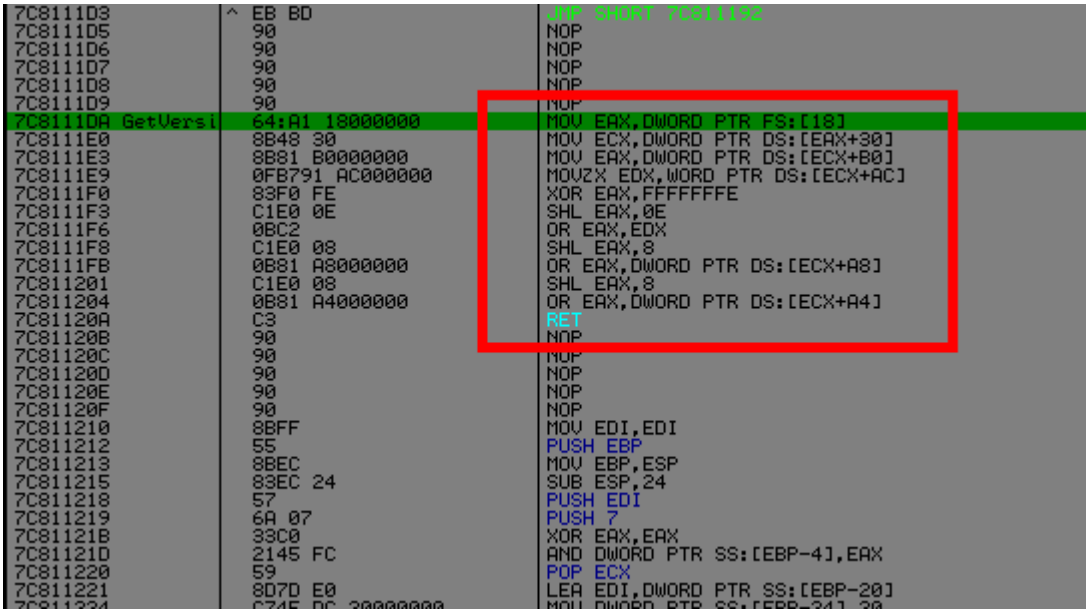


می بینید؟خبری از Nag Screen نیست و گزینه Secret هم فعال شده ☺
خوب،فایل پچ شده را در OllyDBG باز کنید،تا به بررسی تغییراتی که انجام دادم بپردازیم:

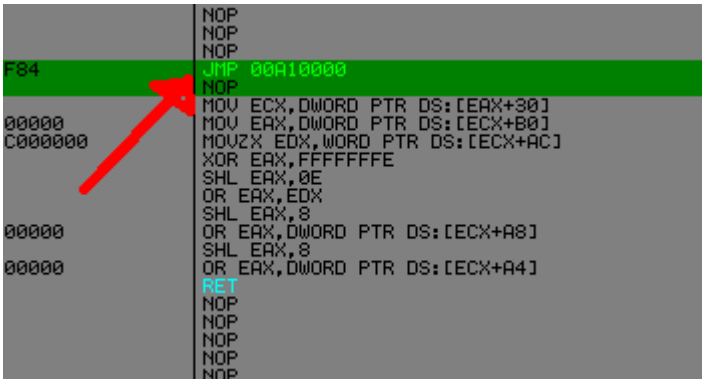


اینجا قبلا کدی قرار نداشت.من با تزریق این کدها تابع GetVersion را Hook کردم.همچنین Entry point فایل را از خط 4070BE به اینجا منتقل کردم.

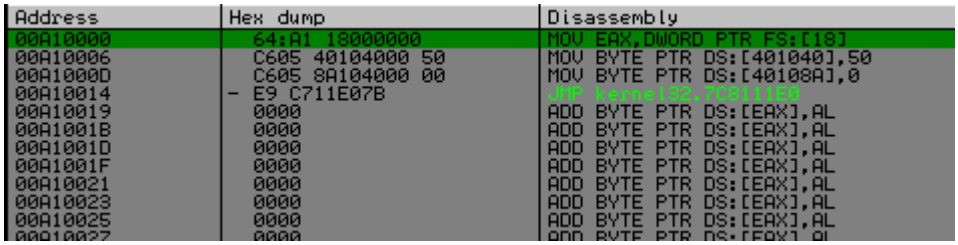
خوب،بهتر است قبل از اجرای کدها نگاهی به تابع GetVersion بیندازیم(CTRL + G را بزنید و تایپ کنید GetVersion):



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:



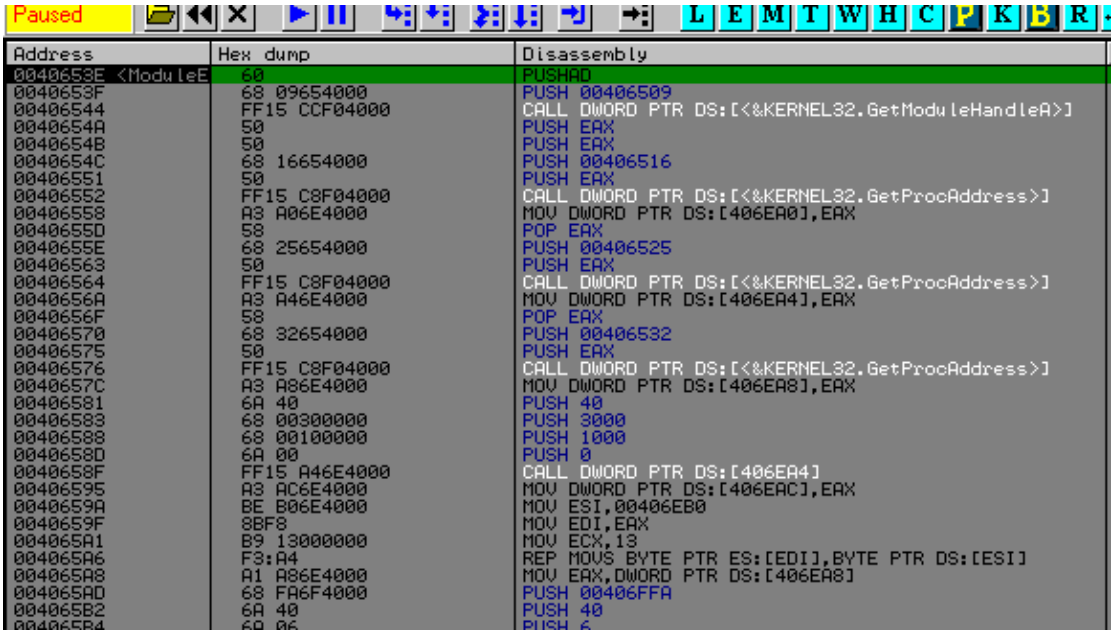
می بینید؟دستوری که قبلا Mov EAX,Dword ptr FS:[18] بوده،الان شده JMP 00A10000...پس کدها توانسته اند تابعGetVersion را Hook کنند.حالا ببینم این پرش به کجا می رود؟...بر روی این پرش کلید Enter را بزنید تا ببینیم این پرش به کجا می رود:



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:



اولین دستوری که اجرا کردم،تابع GetModuleHandleA بود که برای به دست آوردن Handle فایل Kernel32.dll بود و بعد از آن با تابع GetProcAddress آدرس توابعی که لازم دارم را به دست آوردم.(چون IAT 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 MOVS 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 ندارد.من با این تابع کاری کردم که بتوان تابع GetVersion را دستکاری کرد.
برای اطلاعات بیشتر راجع به این توابع(VirtualPrteoct و VirtualAlloc و GetProcAddress) می توانید به سایت MSDN.microsoft.com یا فایل راهنمای توابع API در OllyDBG مراجعه کنید.
بعد از اجرای تابع Virtual Protect 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 ☹

چند نکته:

۱.روش Hook کردن بیشتر در پروتکتورهایی 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 اکثرا 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 معمولا 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.حتما 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 ☺

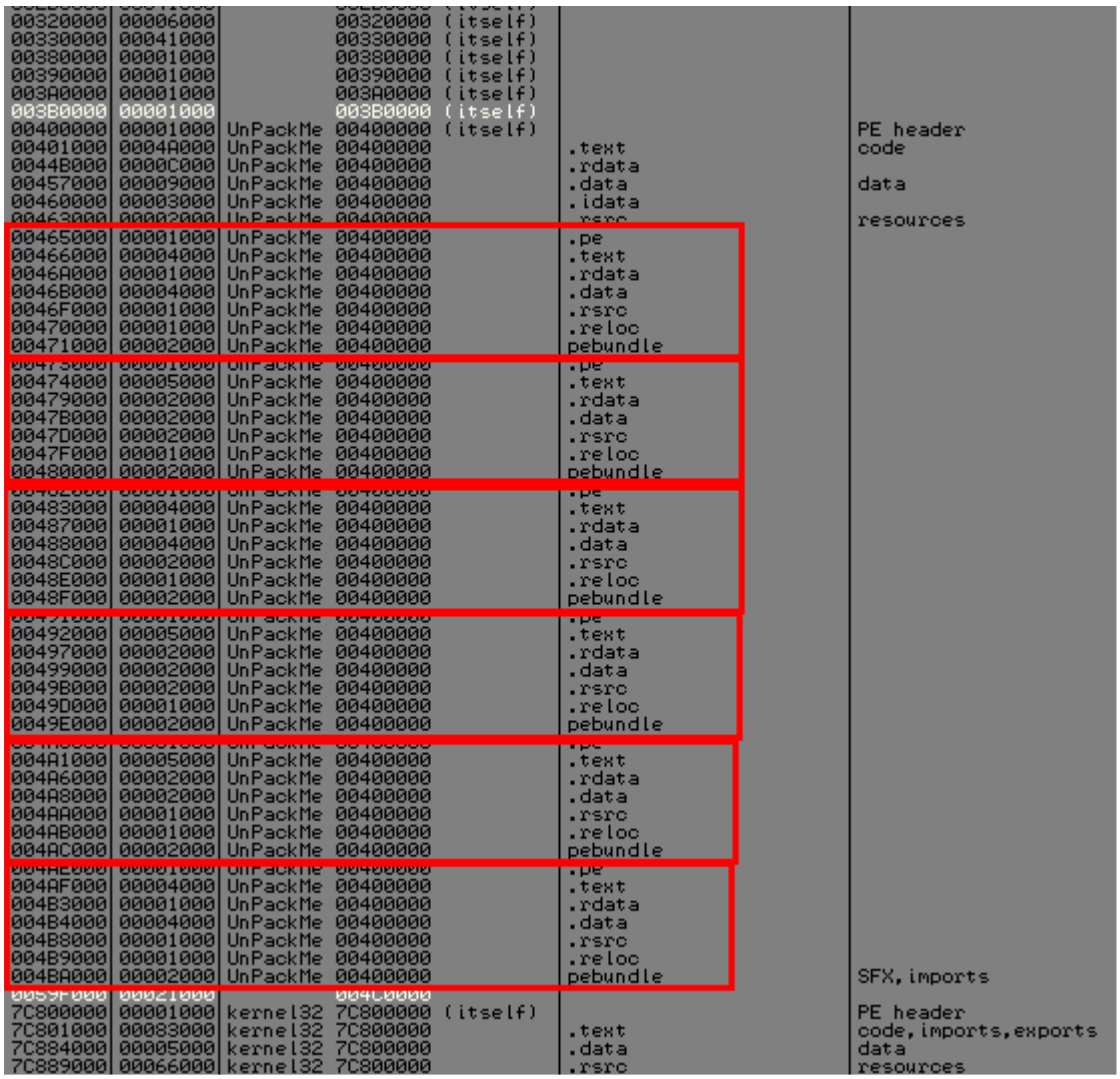
PeBundle

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 نوشتید 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.

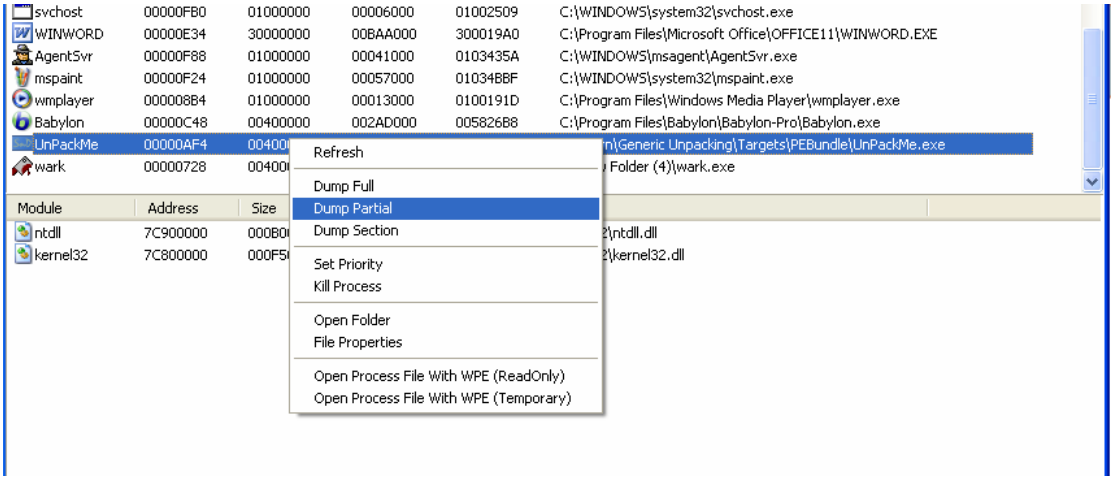
System Dirctory (C:\Windows\system32)
Runtime Folder
Executable's Folder
Windows folder
Temp
Custom path (مسیر انتخابی)

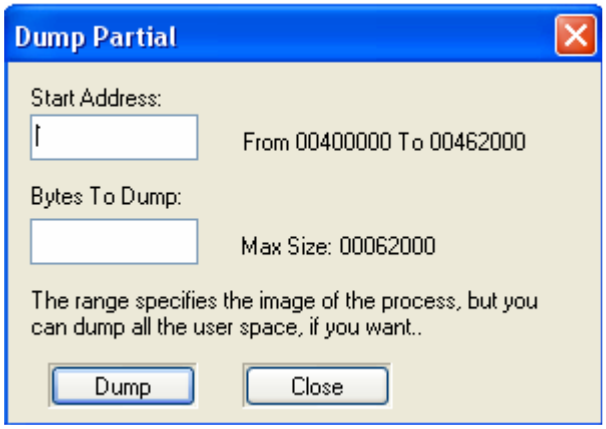
پس همیشه این جاها را بگردید شاید فایل DLL را پیدا کردید.می توانید از نرم افزارهایی مثل FileMon برای اینکه بفهمید فایل DLL کجا ریخته شده استفاده کنید.

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:

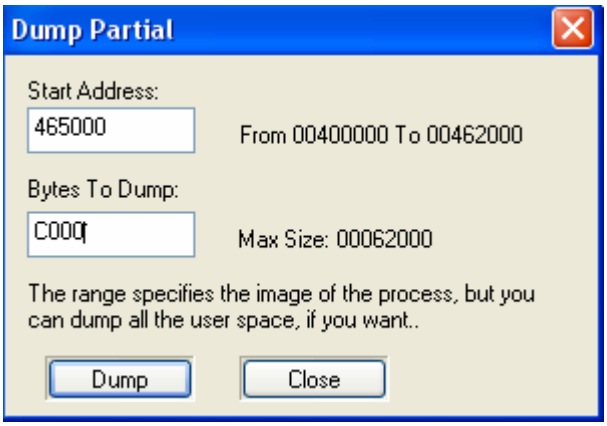


فایل های DLL در سکشن هایی جدید به فایل exe اضافه می شوند.در تصویر بالا،همانطور که می بینید ۶ فایل DLL در درون فایل قرار دارند.(سکشن های هر فایل DLL در درون مستطیل قرار گرفتند).من اولین فایل DLL را از داخل فایل exe بیرون می آورم.۵ فایل دیگر با شما ☺
در تصویر بالا اولین مستطیل قرمز(اولین فایل DLL) از خط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

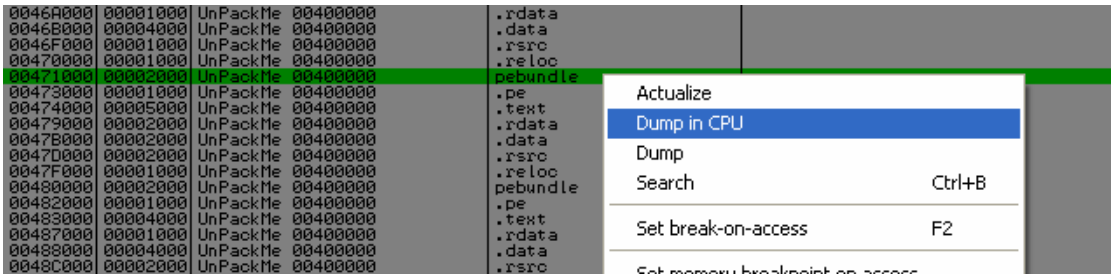




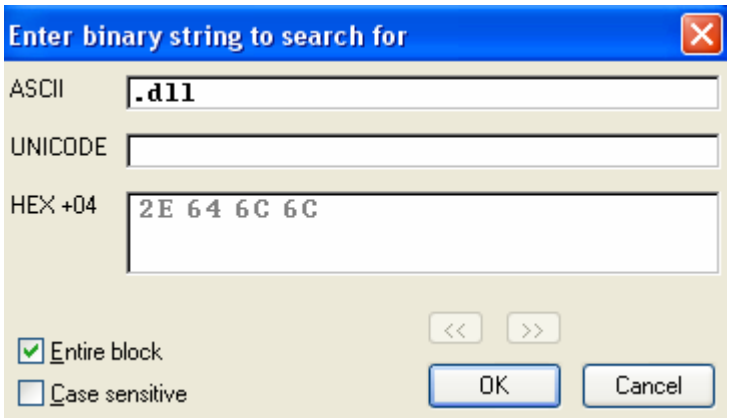
حالا محل شروع فایل DLL را در Start Address می نویسیم:465000 و در قسمت Bytes to dump سایز فایل DLL را می نویسیم(با ماشین حساب ویندوز،انتهای فایل را منهای ابتدای آن کنید.)



حالا گزینه دامپ را بزنید 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 DLL کلیک راست کرده و گزینه Dump in CPU را بزنید:



حالا در پنجره دامپ کلیدهای CTRL + B را بزنید:

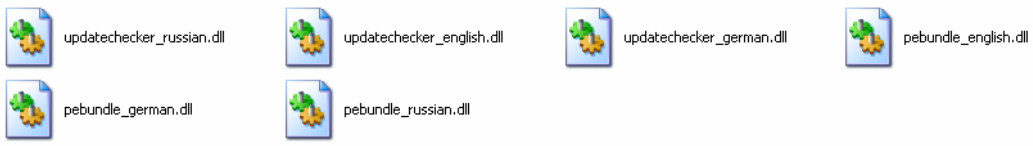


همانند تصویر تایپ کنید dll. (چون نام اصلی فایل در داخل این سکشن نوشته می شود.ما به دنبال مقدار dll. می گردیم تا نام فایل را پیدا کنیم.).کلید OK را بزنید:

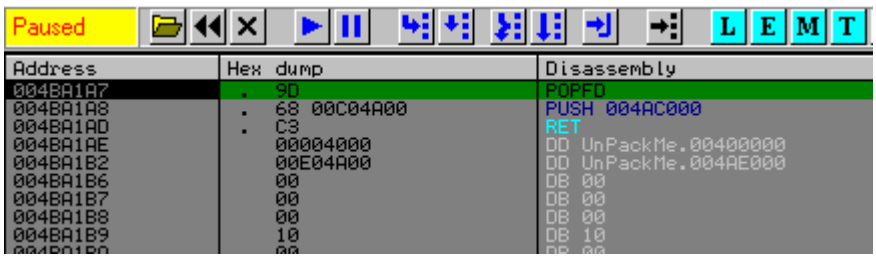
اینجا نوشته شده USER32.dll پس اینجا جایی نیست که ما می خوایم.کلیدهای CTRL + L را بزنید تا دوباره جستجو انجام شود:

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

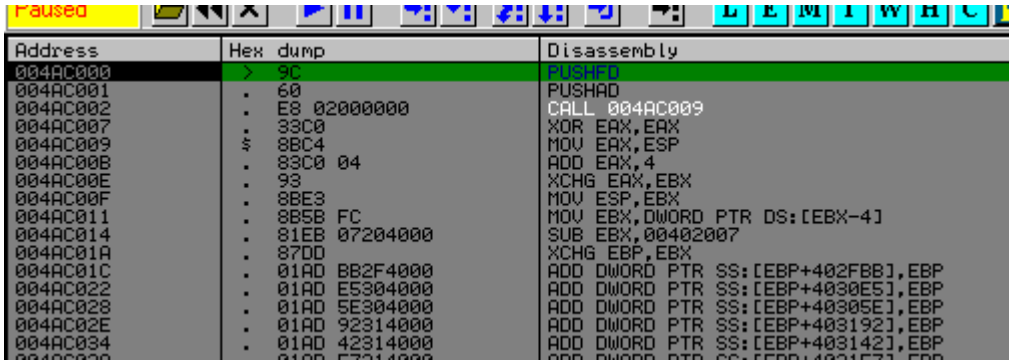
خوب،حالا ۵ فایل DLL دیگر را هم به این ترتیب از داخل فایل بیرون بیاورید:



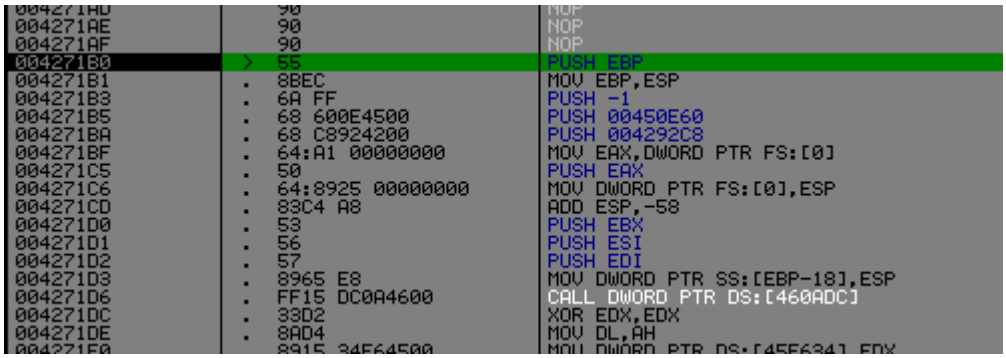
حالا نوبت فایل exe است.برای آنپک کردن فایل exe باید OEP را پیدا کنیم.برای یافتن OEP هم می توانیم از رجیستر ESP استفاده کنیم ☺
دو بار کلید F8 را بزنید تا مقدار رجیستر ESP تغییر کند،بعد بر روی مقدار رجیستر ESP یک Hardware breakpoint on access می گذاریم و بعد برنامه با را با F9 اجرا می کنیم:



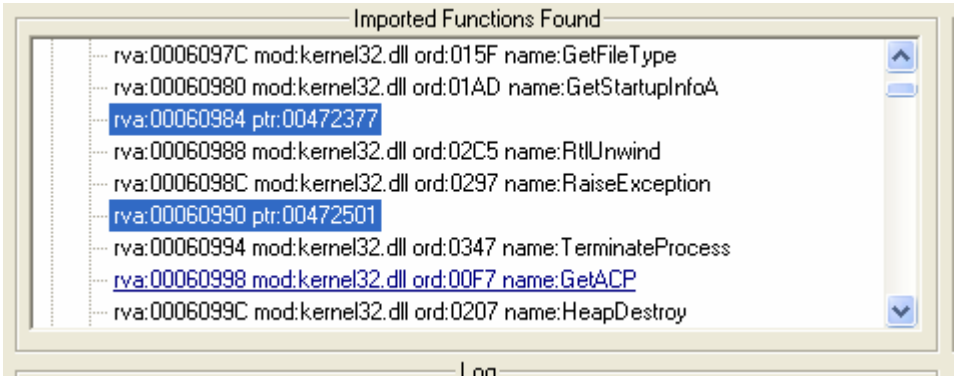
سه بار کلید F8 را بزنید:



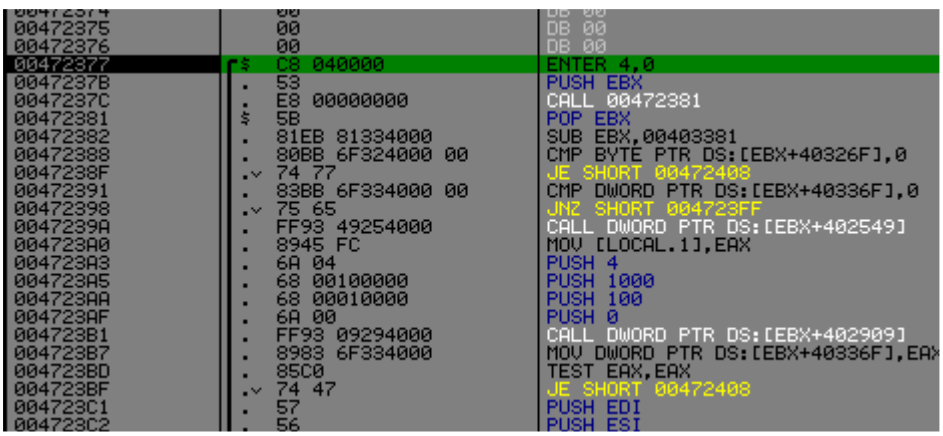
دوباره همین کار را بکنید.یعنی بعد از اجرا شدن دستور PushAD بر روی مقدار رجیستر ESP، Hardware Breakpoint بگذارید.
آنقدر این عمل را انجام دهید تا به OEP برسیم:



حالا از فایل دامپ بگیرید و ImportREC را باز کنید:

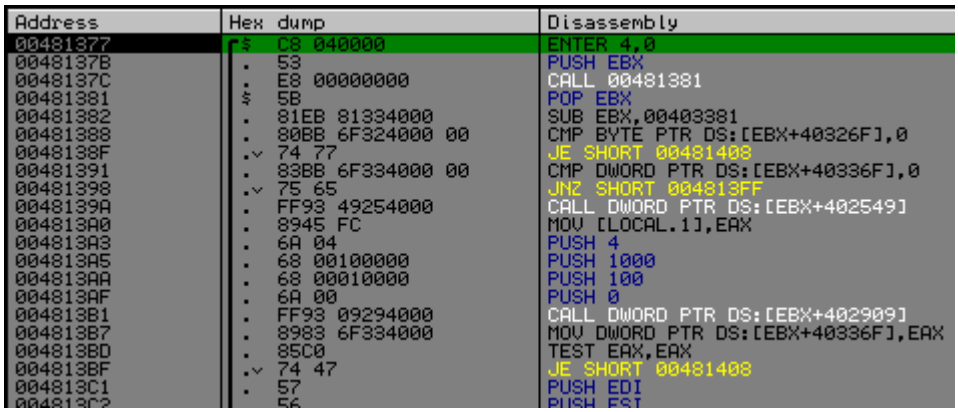


همانطور که می بینید V تا از توابع ما Invalid هستند.(Redirect شده اند) من یکی از آنها را پیدا می کنم،بقیه با شما ☺
همانطور که در تصویر بالا می بینید در خط 60984 (بر حسب RVA) مقدار 472377 قرار دارد که یک تابع API نیست.
در OllyDBG کلیدهای CTRL + G را بزنید و تایپ کنید 472377 و کلید OK را بزنید تا به این 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.

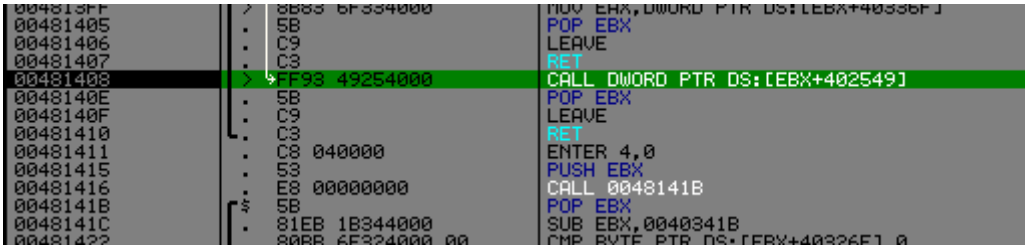


برنامه را با F8 ادامه دهید،پرشی که در 47238F قرار دارد در کامپیوتر من انجام شده است،بنابراین به این خط رسیدم:

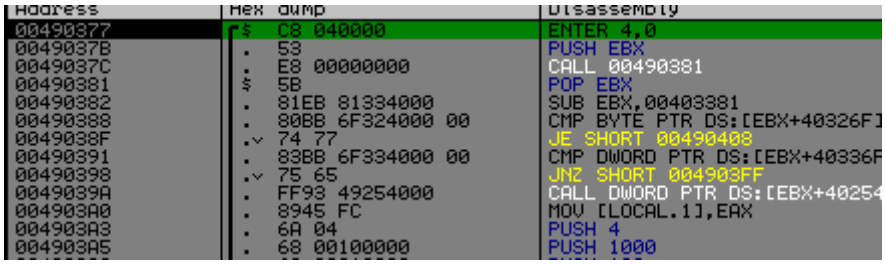
حالا کلید F7 را بزنید تا به داخل تابع بروید.باز هم به جایی مثل خط 472377 رسیدیم:



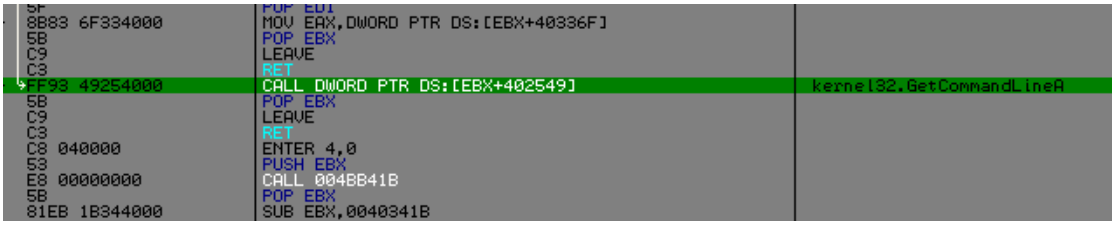
حالا دوباره همین عمل را انجام می دهیم.یعنی چند بار F8 می زنیم تا به دستوری شبیه Call Dword ptr DS:[EBX + xxxxx] برسیم:



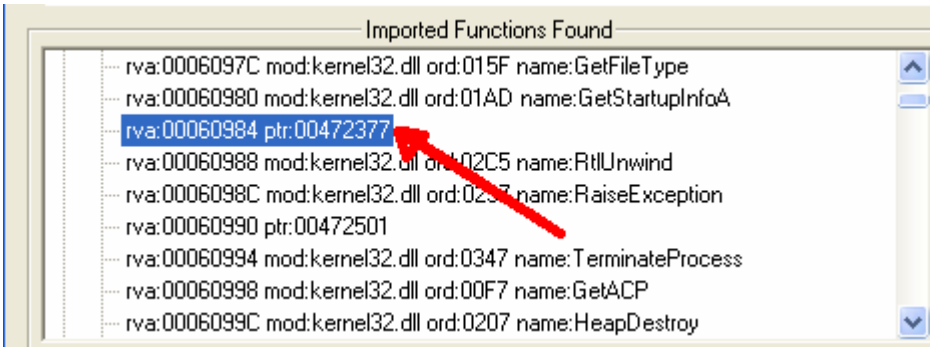
این بار هم F7 می زنیم و به درون این تابع می رویم.باز هم به جایی شبیه خط 472377 رسیدیم:



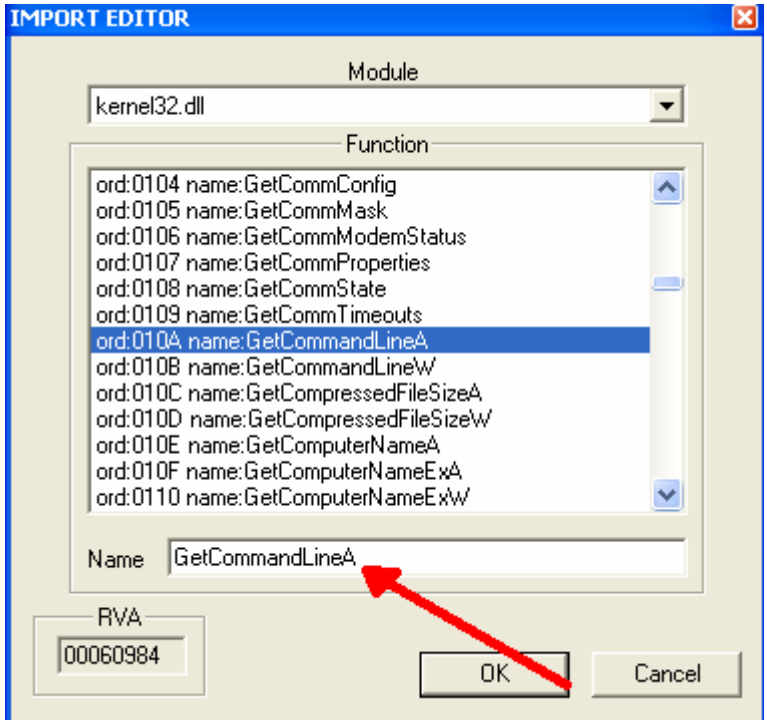
آنقدر این عمل را انجام دهید تا بالاخره به اینجا برسید:



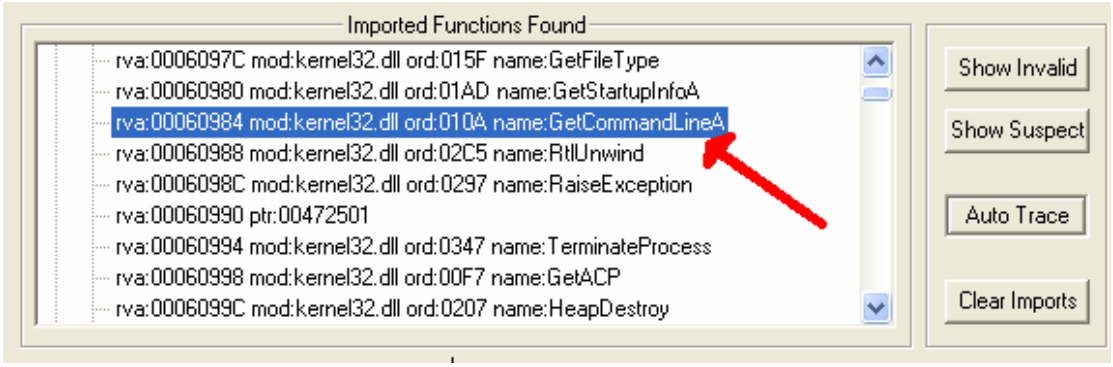
همانطور که می بینید در نهایت خط 472377 به تابع GetCommandLineA می رود.پس دوباره به ImportREC برگردید:



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 اطلاعتان لیست توابع صحیح را نوشتم،شما با اندکی تلاش می توانید اینها را به دست بیاورید.)

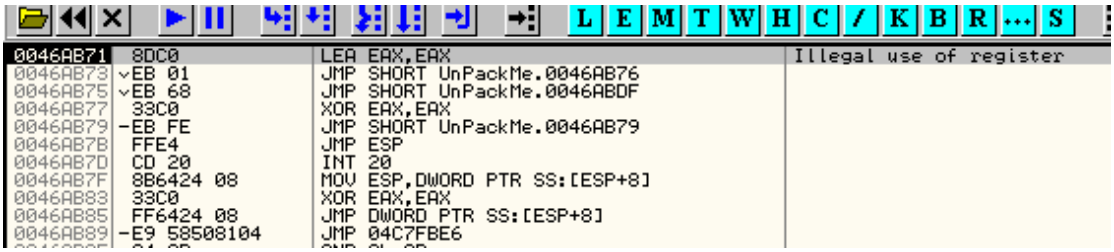
Rva: 00060984=GetCommandLineA
Rva: 00060990=ExitProcess
Rva: 00060AD8=GetModuleFileNameA
Rva: 00060B3C=LoadLibraryA
Rva: 00060B40=GetProcAddress
Rva: 00060B44=FreeLibrary
Rva: 00060B9C=GetModuleHandleA

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

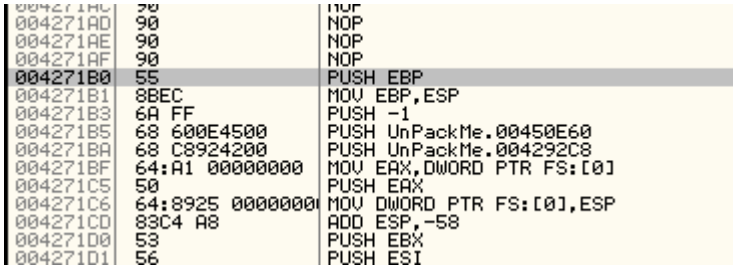
اگر برنامه پیغام زیر را داد،یعنی یکی از فایل های DLL را درست Extract نکرده اید:

Telock

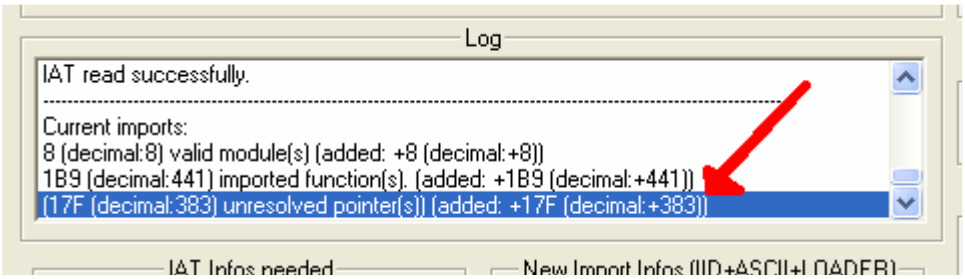
فایل مورد نظر را در OllyDBG باز کنید.(این فایل از چند آنتی دیباگ استفاده می کند.برای همین آن را در OllyDBG modified by SLV باز کنید.در این OllyDBG برنامه بی هیچ مشکلی اجرا می شود.)برای یافتن OEP هم می توانیم از Memory breakpoint استفاده کنیم.ابتدا تیک تمامی خطاها را بردارید.(کلیدهای ALT + O را بزنید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



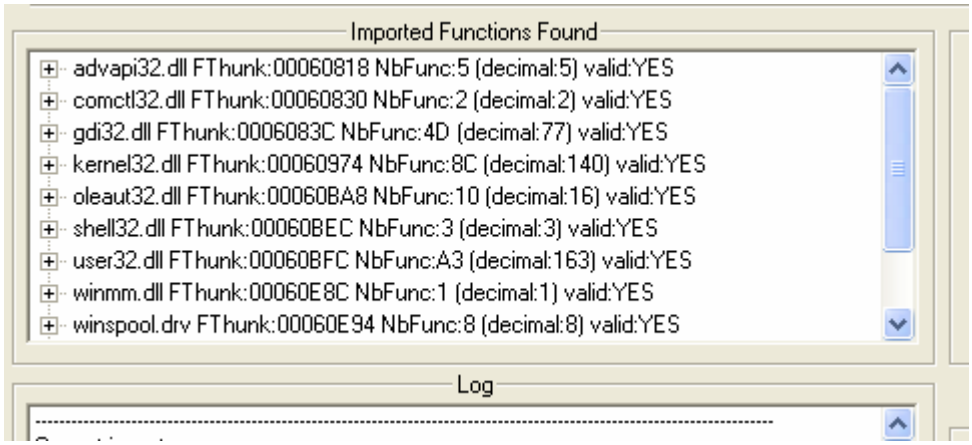
اینجا دیگر کدهای ما Decrypt شده است و می توانیم Memory breakpoint بگذاریم.بر روی سکشن text. یک Memory breakpoint on access بگذارید.و بعد هم کلید F9 را بزنید:



حالا از فایل دامپ بگیرید و ImportREC را باز کنید:



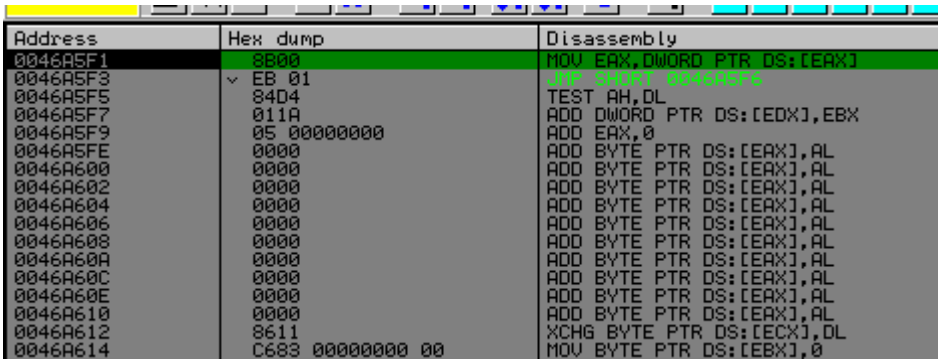
همانطور که می بینید ۳۸۳ تابع Invalid داریم.برای فیکس کردن این توابع می توانیم از قابلیت Trace level3 (trap flag) استفاده کنیم.ابتدا برنامه را به طور کامل در OllyDBG باز کنید(کلید F9 را بزنید.) و بعد بر روی توابع Invalid کلیک راست کرده و گزینه trap flag را بزنید تا تابع valid شود.(اگر از سیستم ضعیفی استفاده 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.)



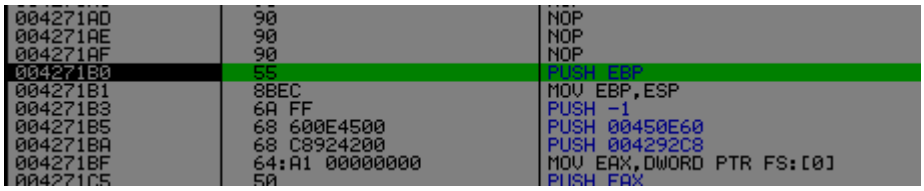
خوب...حالا که تمامی توابع Valid شد،فایل دامپ شده را فیکس کنید.

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):



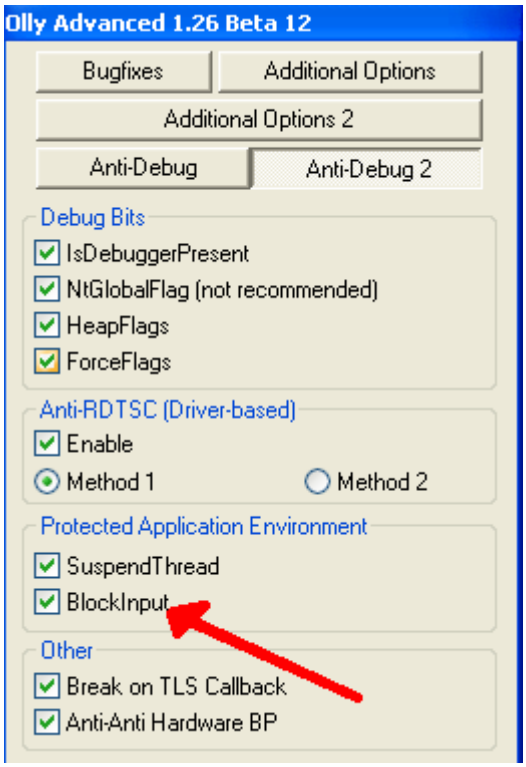
حالا هم همانند مثال های قبل یک Memory Breakpoint بر روی سکشن text. بگذارید و بعد برنامه را با F9 اجرا کنید:



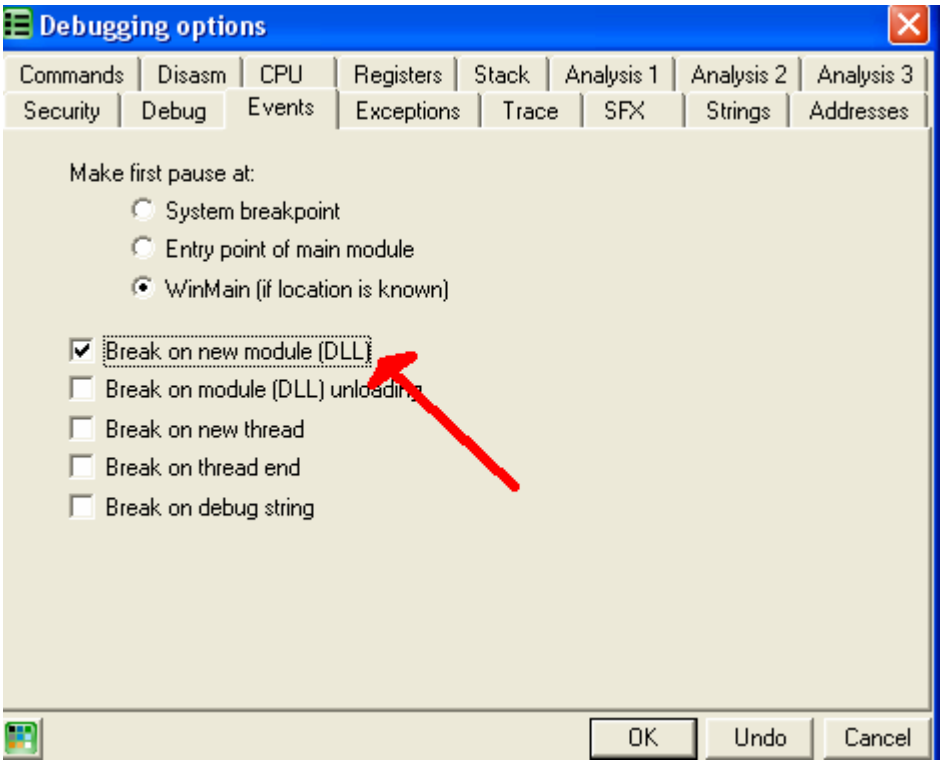
به همین راحتی به OEP رسیدیم.حالا دامپ بگیرید و فیکس کنید ☺

Yoda's Protector

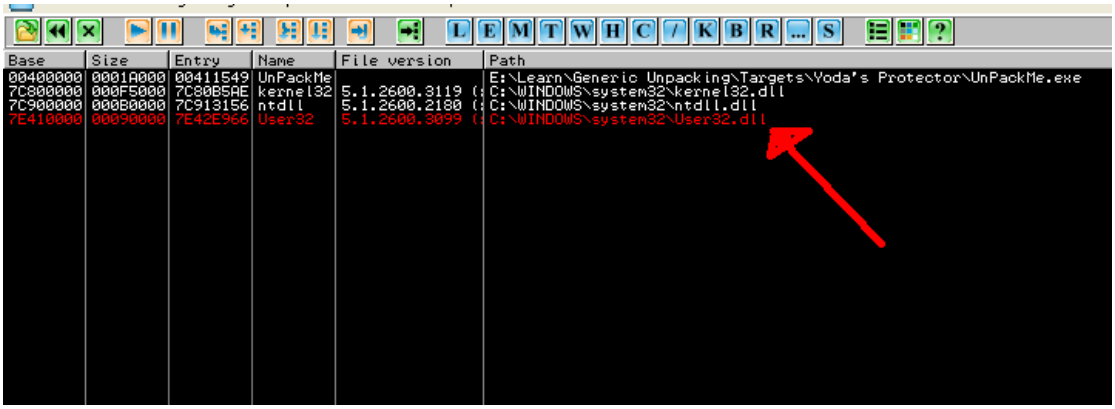
فایل مورد نظر را در OllyDBG باز کنید. چون Yoda's Protector در صورت شناسایی Debugger تابع BlockInput را به جون ما می اندازد.(این تابع موس و کیبرد را از کار می اندازد.هیچکدام از دکمه های کیبرد عمل نمی کنند،مگر (ALT + CTRL + Del
به همین دلیل ما باید این تابع را از بین ببریم.در منوی Plugins گزینه OllyAdvanced (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:

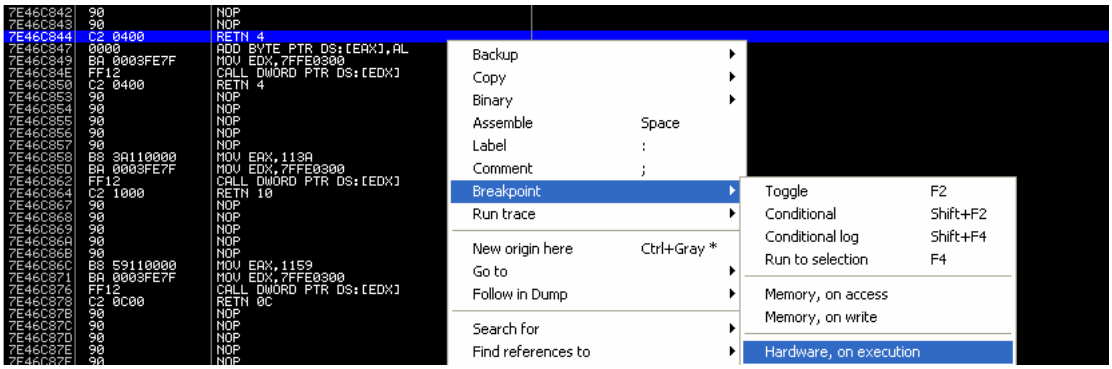


کلید OK را بزنید و دو بار کلید F9 را بزنید تا فایل User32.dll اجرا شود:

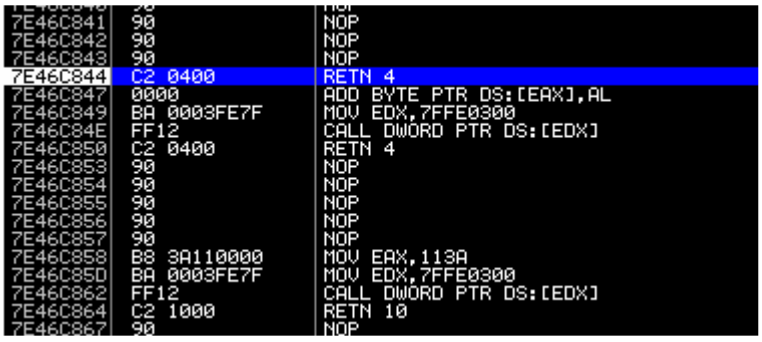


خوب،حالا کلیدهای ALT + C را بزنید تا وارد پنجره اصلی(CPU) شوید و بعد کلیدهای CTRL + G را بزنید و تایپ کنید:
BlockInput

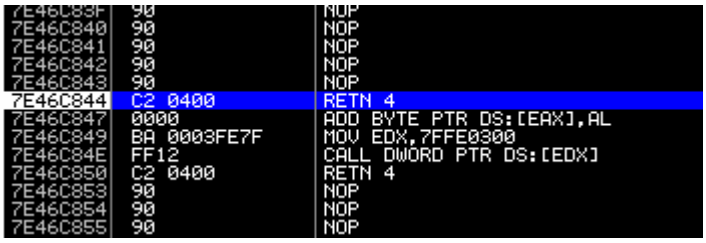
کلید OK را بزنید و بر روی این تابع یک Hardware BP on execution بگذارید:



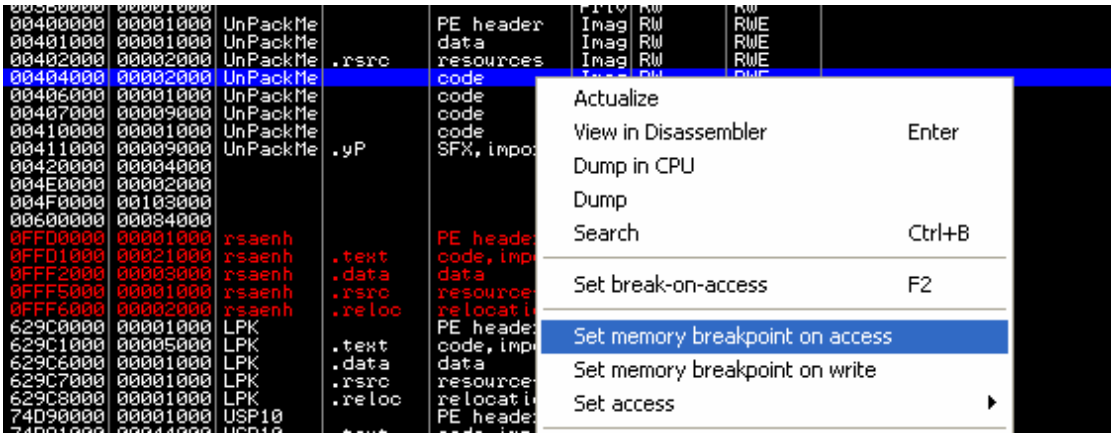
حالا دیگر نیازی به گزینه Break on new module نداریم.(کلیدهای ALT + O را بزنید و در قسمت Events تیک گزینه Break on new modules را بردارید.)



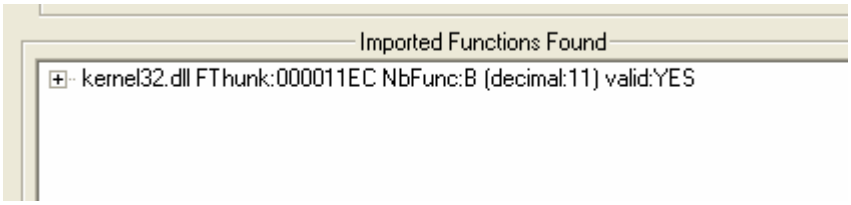
در تابع BlockInput متوقف شدیم.یکبار دیگر F9 بزنید:



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:



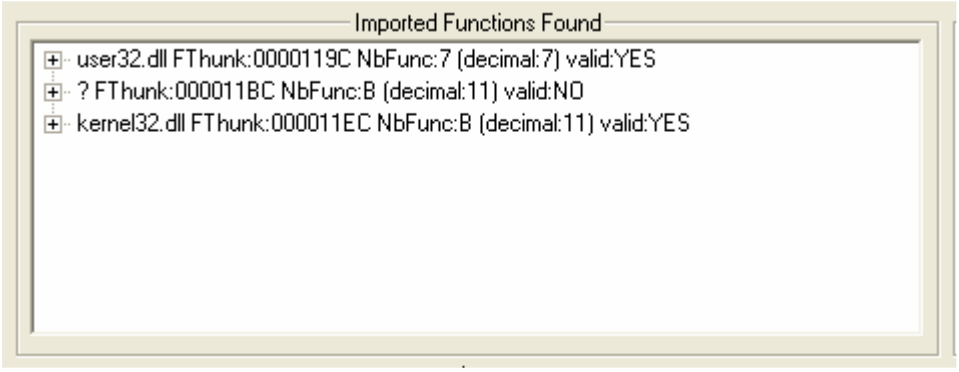
حال کلید F9 را بزنید تا به OEP برسید.(کلیدهای CTRL + A را بزنید تا این سکشن آنالیز شود)
بعد از فایل دامپ بگیرید.حالا ImportREC را باز کنید.OEP را به برنامه بدهید و سایر مراحل لازم را انجام دهید:



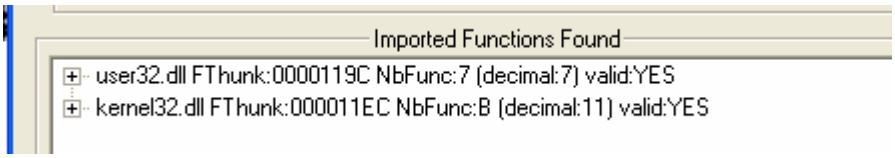
همانطور که می بینید ImportREC نتوانسته IAT را به درستی به دست بیاورد.پس باید خودمان شروع IAT را پیدا کنیم:

Start of IAT= 401198 (RVA: 1198)
End of IAT = 401218 (RVA: 1218)
Size = 80

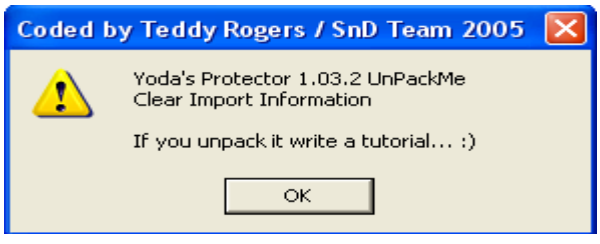
این اطلاعات را در ImportREC وارد کنید و Get Imports را بزنید:



حالا گزینه show Invalid را بزنید،بر روی توابع Invalid کلیک راست کرده و گزینه Cut Thunk را بزنید:

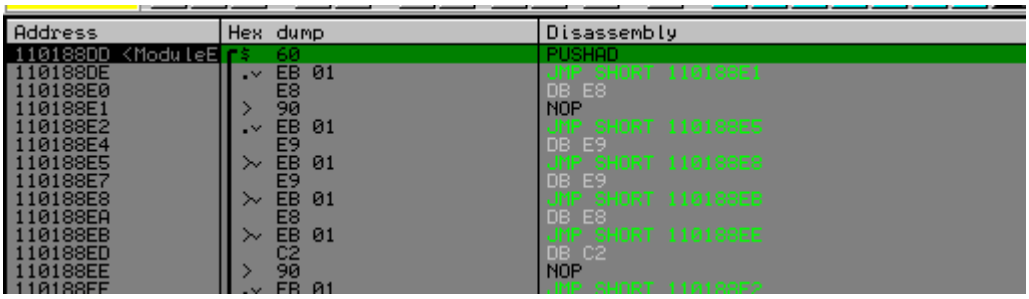


حالا هم فایل دامپ خود را فیکس کنید:

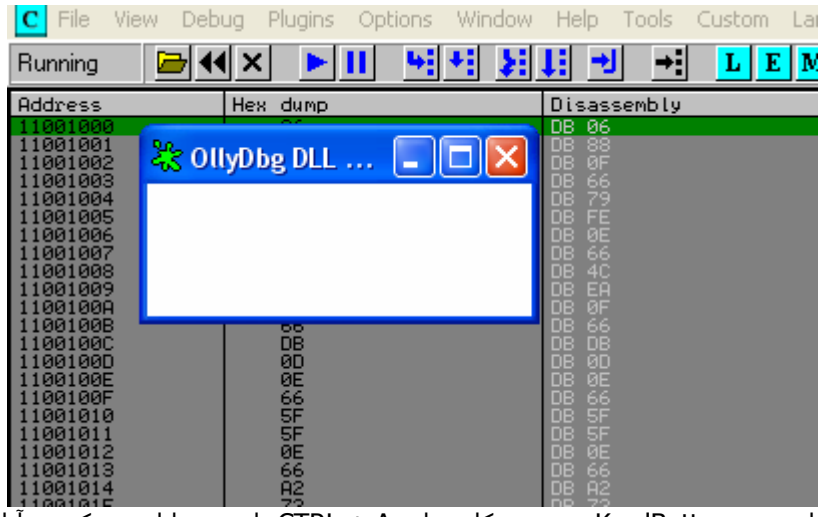


Yoda's Protector OCX

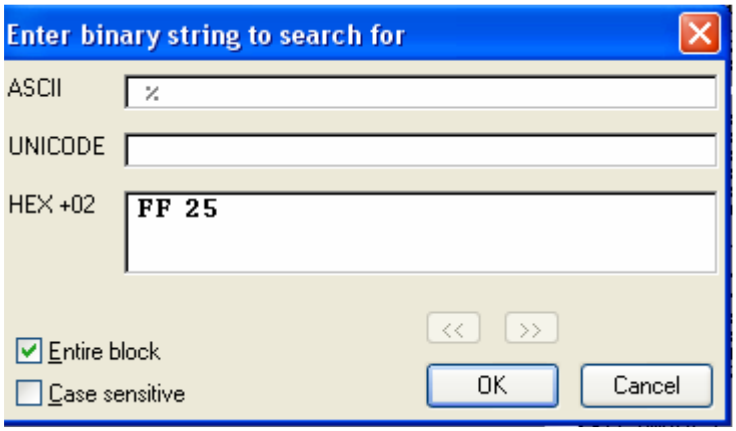
فایل مورد نظر(KewlButtonz.ocx) را در OllyDBG باز کنید:



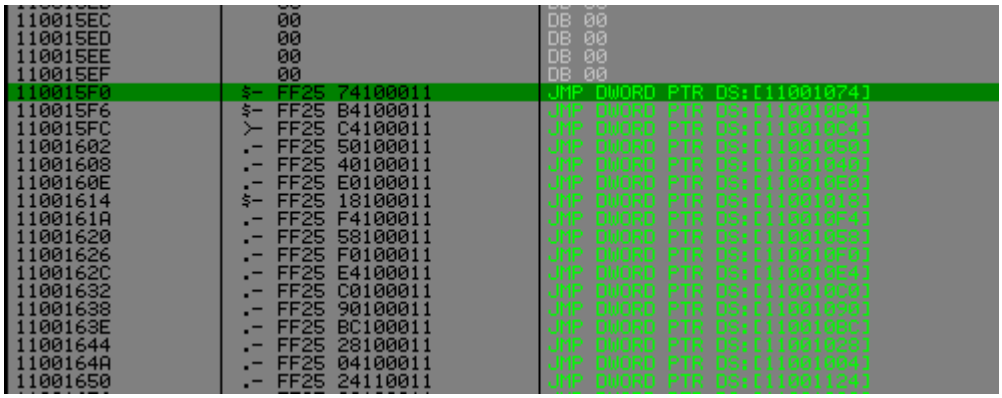
برای اجرا شدن فایل OCX چند بار کلید F9 را بزنید تا فایل Loaddll.exe اجرا شود:



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 :

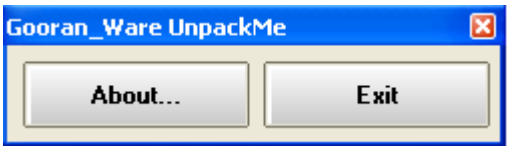


کلید OK را بزنید:



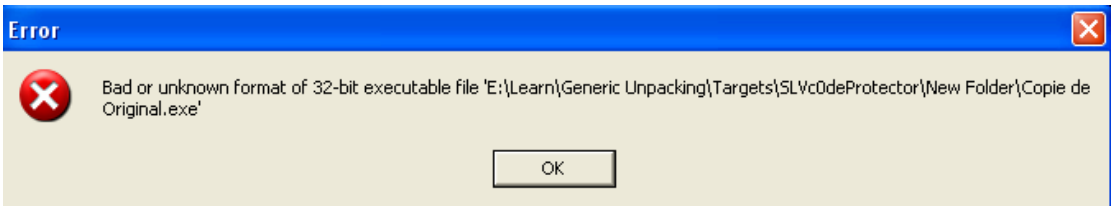
همانطور که می بینید به JUMP Table رسیدیم.همانطور که می دانید درست زیر Jump Table ،OEP قرار دارد.پس:

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:

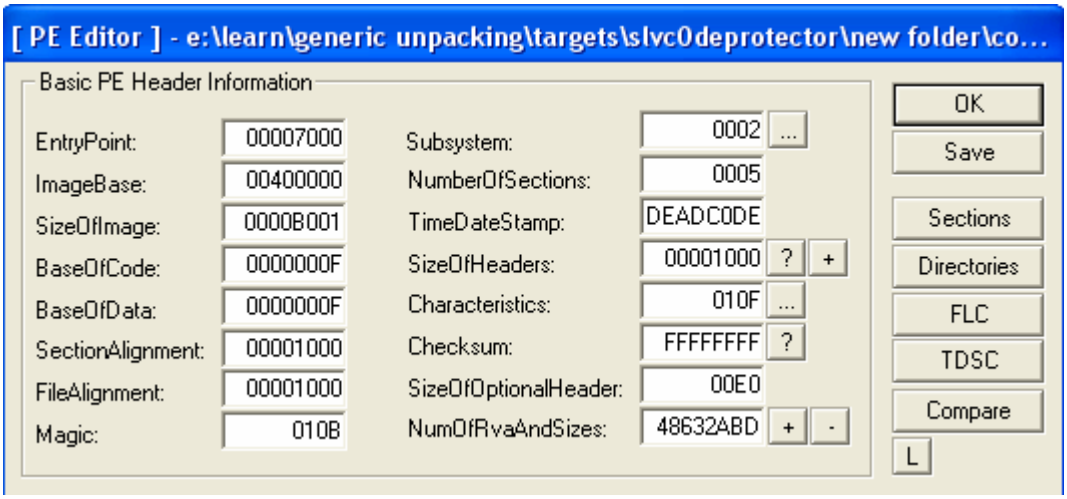


SLVc0deProtector

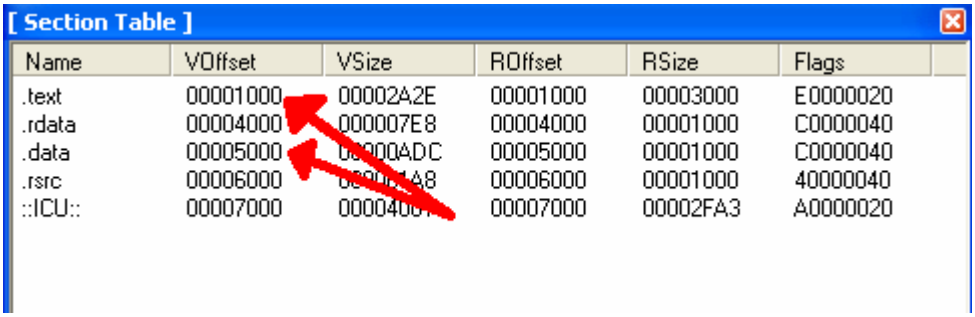
خوب،فایل مورد نظر را در OllyDBG باز کنید(یک OllyICE که فقط پلاگین های HideDBG،OllyDump و OllyDump رو داشته باشد):



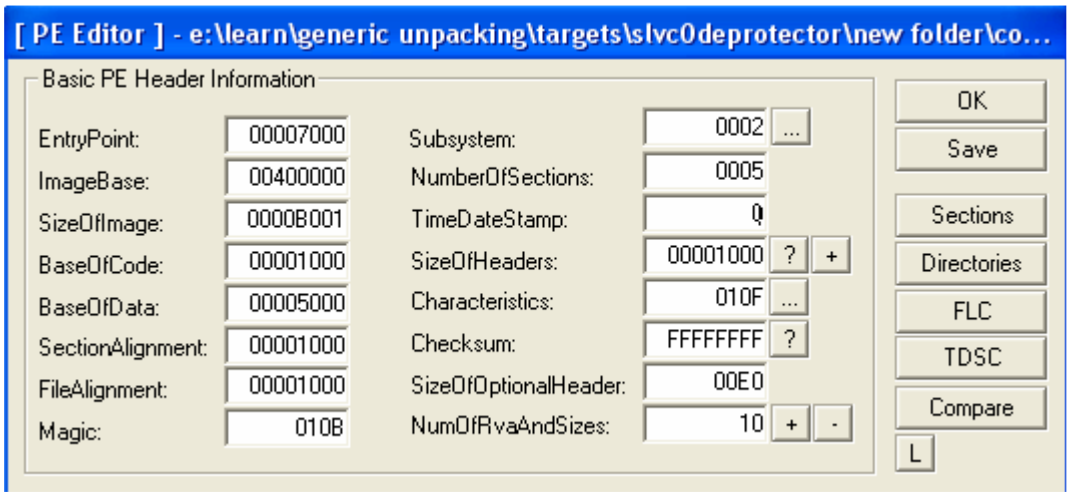
همانطور که می بینید،PE Header فایل دستکاری شده است.به همین دلیل به درستی در OllyDBG اجرا نمی شود ☹
فایل را در LordPe باز کنید،تا مشکلات فایل را بررسی کنیم:



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:

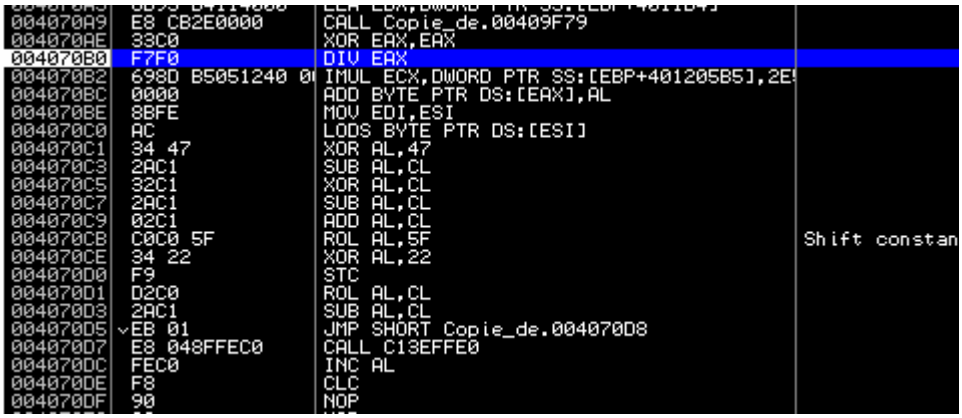


همانطور که می بینید محل شروع سکشن text. برابر ۱۰۰۰ است.پس BaseOfCode در واقع باید ۱۰۰۰ باشد.
در ضمن محل شروع سکشن data. برابر ۵۰۰۰ است.پس BaseofData هم در واقع باید ۵۰۰۰ باشد.
۴.بهتر است مقدار Checksum رو هم صفر کنید.ولی اگر نکردید هم مشکلی پیش نمی آید.

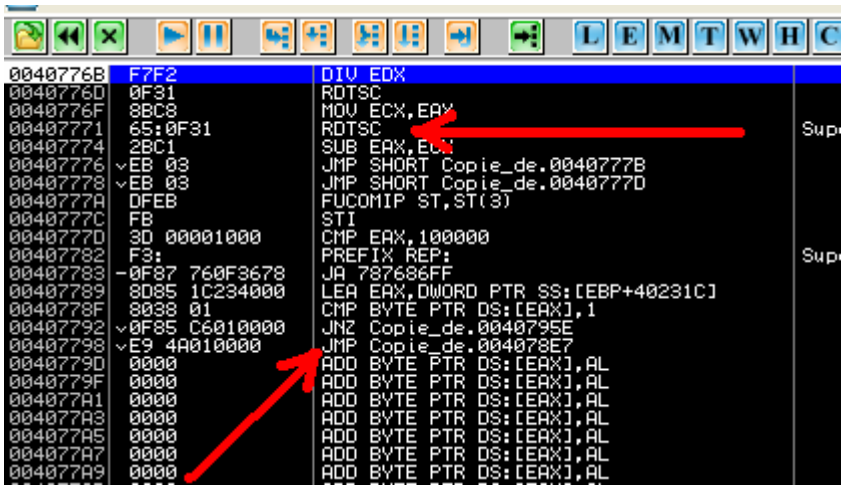


حالا تغییرات را ذخیره کنید و فایل را در OllyDBG باز کنید.می بینید که فایل بی هیچ مشکلی در OllyDBG باز می شود.اما اگر فایل را با خود ویندوز اجرا کنید:

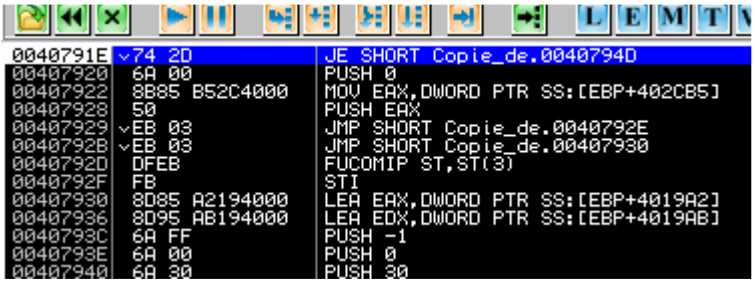
همانطور که می بینید برنامه متوجه شده است که ما PE Header فایل را دستکاری کرده ایم ☹
خوب در داخل OllyDBG ابتدا پلاگین HideDBG را فعال کنید.(در منوی Plugins گزینه HideDBG و سپس گزینه Hide All را بزنید.)
و بر روی تابع FindWindowA یک Hardware Breakpoint بگذارید.(چون فایل ما از این تابع برای شناسایی OllyDBG استفاده می کند.)
در ضمن تیک تمامی خطاها را بردارید و برنامه را اجرا کنید،به خاطر خطا در اینجا متوقف شدیم:



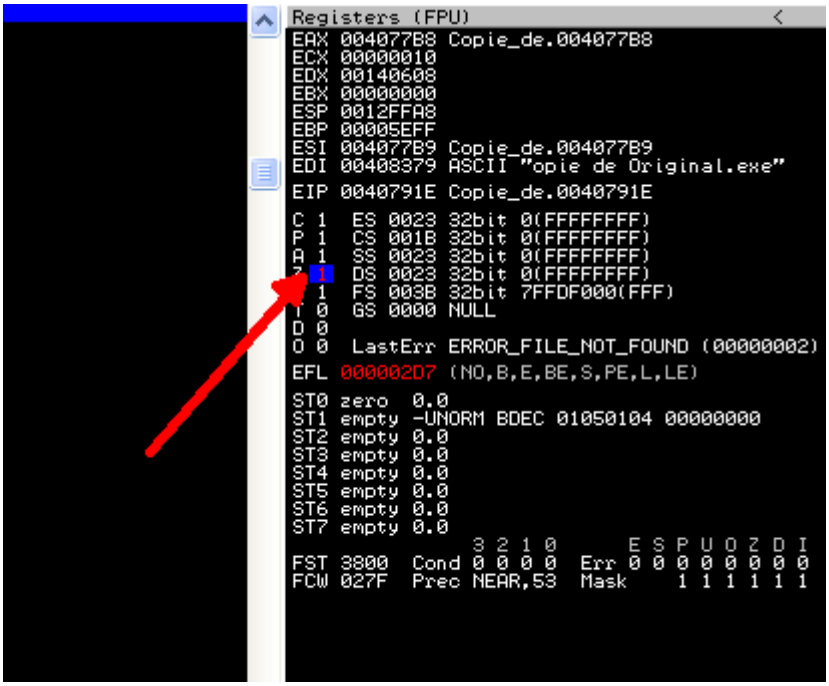
یکبار دیگر Shift + F9 را بزنید،تا چند بار در تابع FindWindowA متوقف شویم.بعد از آن به خاطر خطا در اینجا متوقف می شویم:



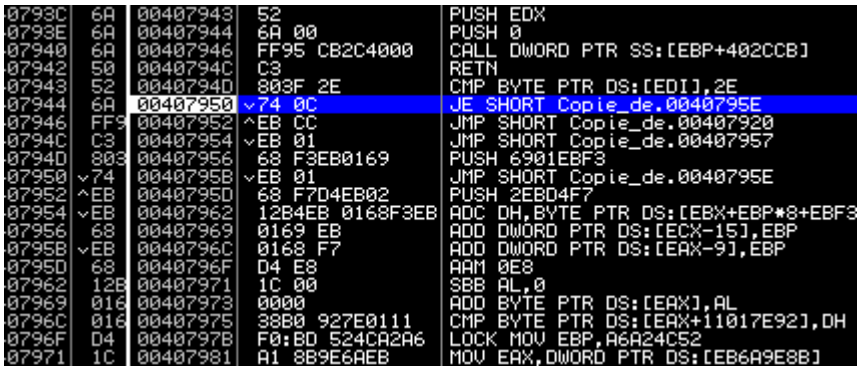
همانطور که می بینید در اینجا چند دستور آنتی دیباگ قرار دارد.تابع RDTSC که زمان اجراي یک دستور را محاسبه می کند یکی از آنهاست.بر روی خط 407798 یک BP بگذارید و برنامه را با Shift + F9 اجرا کنید،بعد از آن چند بار کلید F8 را بزنید تا به اینجا برسید:



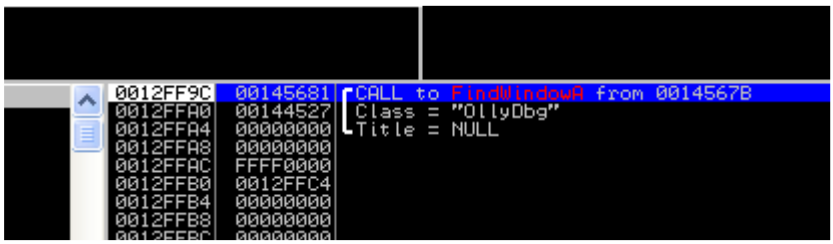
این پرش شرطی چک می کند،آیا نام فایل تغییر کرده است یا خیر؟...این پرش باید همواره انجام شود.پس وقتی به این پرش رسیدید.مقدار Z-Flag را تغییر دهید تا این پرش انجام شود:



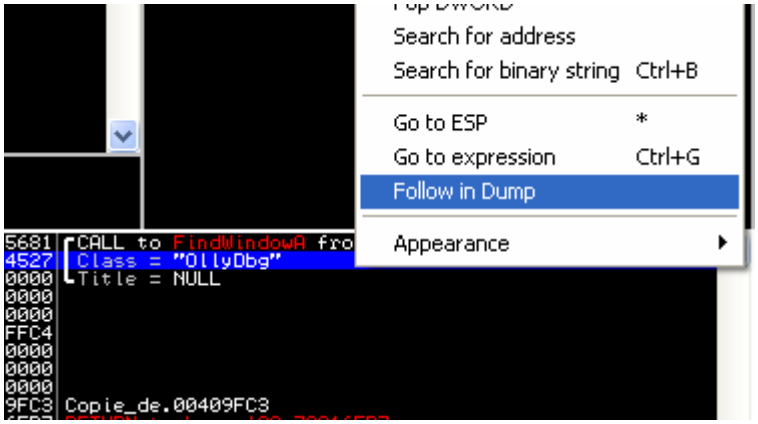
حالا برنامه را با F8 ادامه دهید تا به یک پرش شرطی دیگر برسید:

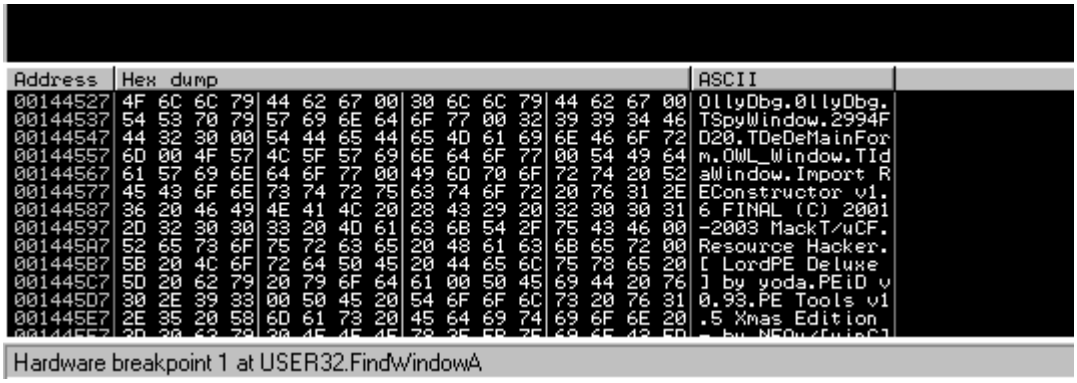


این پرش هم باید انجام شود.(مقدار Z-flag را در اینجا برابر یک کنید.)...اگر این پرش انجام نشود.پیغام File name changed داده می شود ☹
حالا برنامه را با F9 اجرا کنید،نگاهی به پنجره Stack بیندازید:

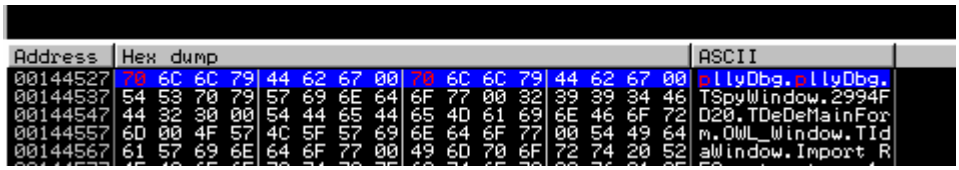


می بینید؟برنامه با تابع FindWindowA به دنبال OllyDbg می گردد ☹ خوب...بر رو نام OllyDBG کلیک راست کرده و گزینه Follow in dump را بزنید:



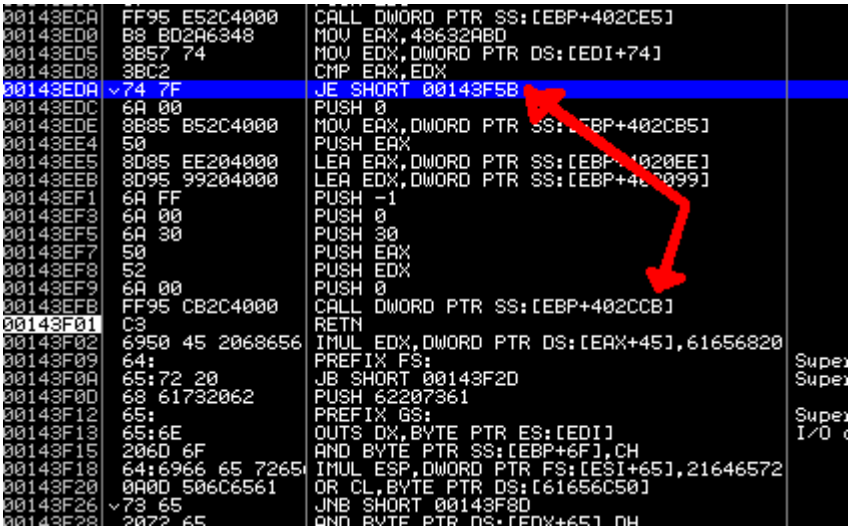


همانطور که می بینید در اینجا دو مقدار با نام OllyDBG نوشته شده است.ما برای اینکه برنامه نتواند پنجره OllyDBG را پیدا کند،می توانیم نام این دو مقدار را عوض کنیم(برای این کار در قسمت ASCII نام ها را انتخاب کنید و کلیدهای CTRL + E را بزنید و آن قسمت از دامپ را ویرایش کنید)

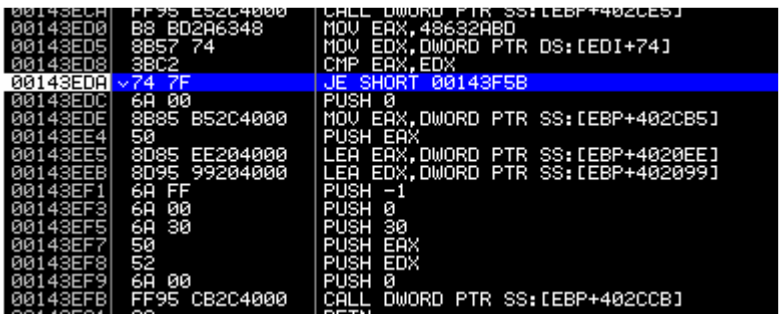


همانطور که می بینید من نام OllyDBG را به pllyDBG تغییر دادم.تا برنامه نتواند چیزی پیدا کنید.خوب،برنامه را با F9 ادامه دهید،در چند جا متوقف می شویم،برنامه به غیر از ollyDBG به دنبال برنامه دیگری هم می گردد.(مثل ImportREC) ☹
بعد از اینکه تمامی اینها را رد کردیم.به این پیغام می رسیم:

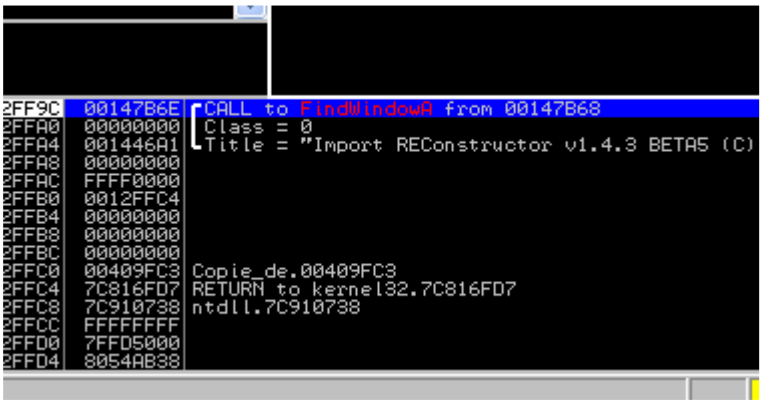
حالا چه جوری این پیغام را از بین ببرم؟برنامه را در OllyDBg متوقف کنید.(کلید F12 را بزنید.)و کلید ALT + F9 را بزنید تا وارد حالت Back to user شوید.و بعد هم کلید OK را بزنید تا پیغام داده شده برود،به این ترتیب در اینجا متوقف می شویم:



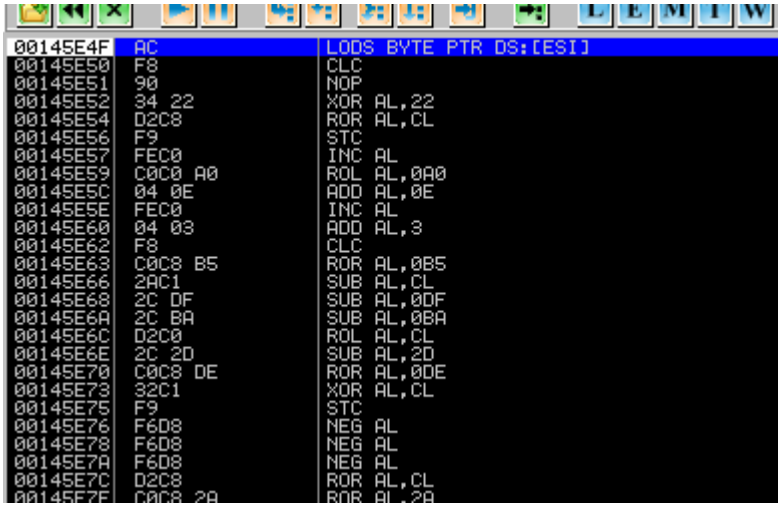
پرشی که در خط 143EDA (ممکن است آدرس این خطوط در سیستم شما فرق داشته باشد) تصمیم گیرنده است.یعنی اگر این پرش انجام شود،برنامه چنین پیغامی را نمی دهد.پس بر روی این پرش Hardware breakpoint on execution بگذارید و برنامه را ری استارت کنید.حالا تمامی کارهایی که قبلا انجام دادیم را دوباره انجام دهید.تا دوباره به این پرش برسیم:



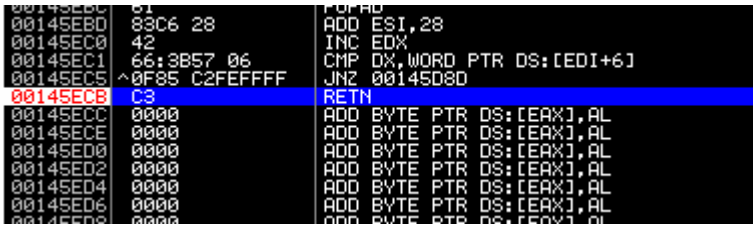
حالا مقدار Z-flag را تغییر دهید تا این پرش انجام شود.حالا کلید F9 را بزنید،می بینید برنامه دوباره می خواهد با تابع FindWindowA ،پنجره OllyDBG را پیدا کند،اما ما قبلا نام OllyDBG را دستکاری کردیم،پس مشکلی به وجود نمی آید. ☺ چند بار کلید F9 را بزنید تا به آخرین تابع FindWindowA برسیم:



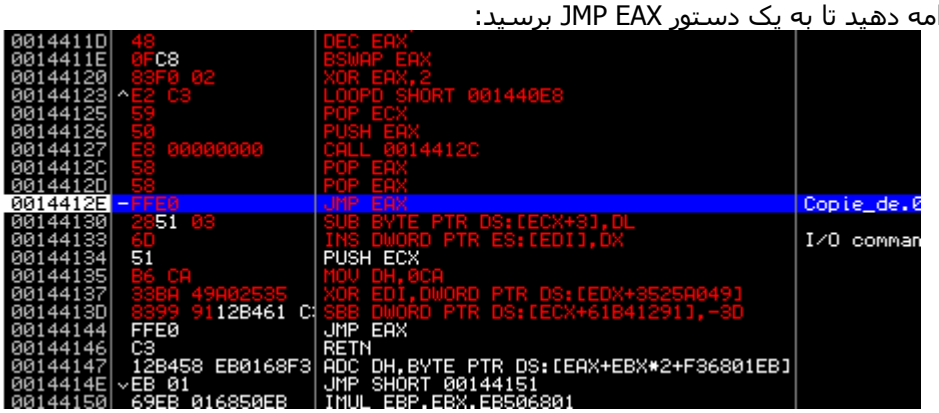
اینجا دیگر آخرین جایی است که برنامه ما از تابع FindWindowA استفاده می کند.حالا بر روی سکشن text. برنامه یک Memory Breakpoint بگذارید.و بعد برنامه را با F9 اجرا کنید:



اینجا جایی است که سکشن text. ، Decrypt می شود.پس با موس به پایین بروید و بر روی RET که در آخر این 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:



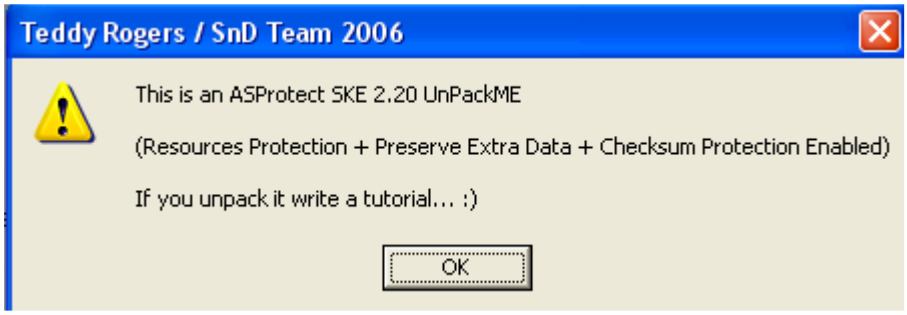
حالا با F8 برنامه را ادامه دهید تا به یک دستور JMP EAX برسید:



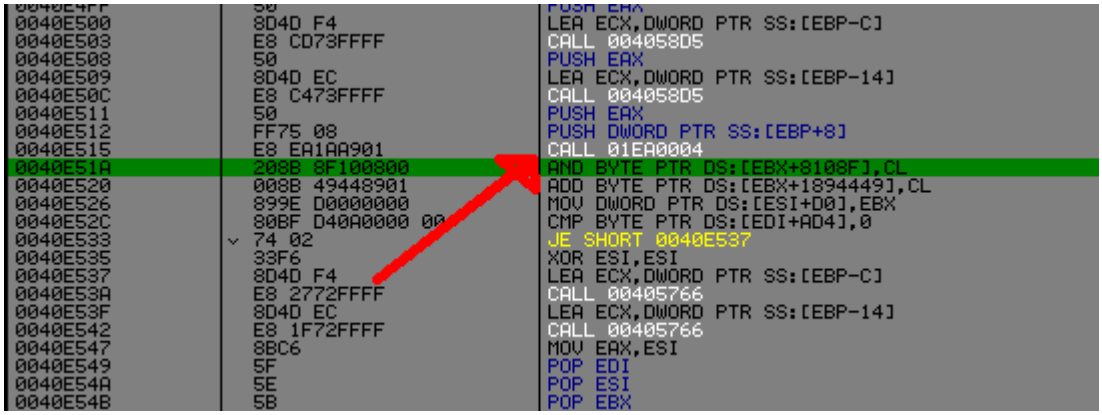
این پرش ما را به OEP می برد ☺
حالا از فایل دامپ بگیرید.(ترجیحا با خود ImportREC این کار را بکنید.)
و بعد با ImportREC فایل را فیکس کنید.(تیک گزینه Use PE Header from disk را هم باید بزنید.)
فایل آنپک شده را اجرا کنید،می بینید که بی هیچ مشکلی اجرا می شود ☺

Asprotect

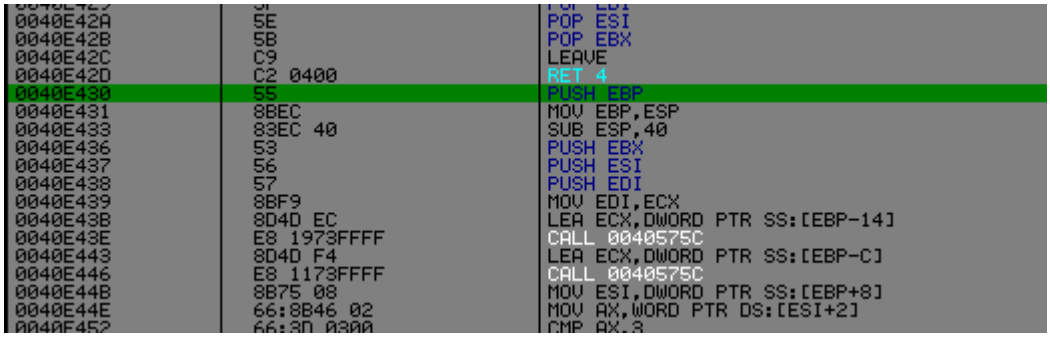
فایل مورد نظر را در OllyDBG باز کنید.برای یافتن OEP می توانیم از روش Back trace استفاده کنیم.یعنی جایی از برنامه را پیدا کنیم که مطمئن باشیم،آنجا بعد از OEP است و بعد هم آنقدر به عقب برویم تا به OEP برسیم ☺
ابتدا برنامه را به طور کامل اجرا کنید:



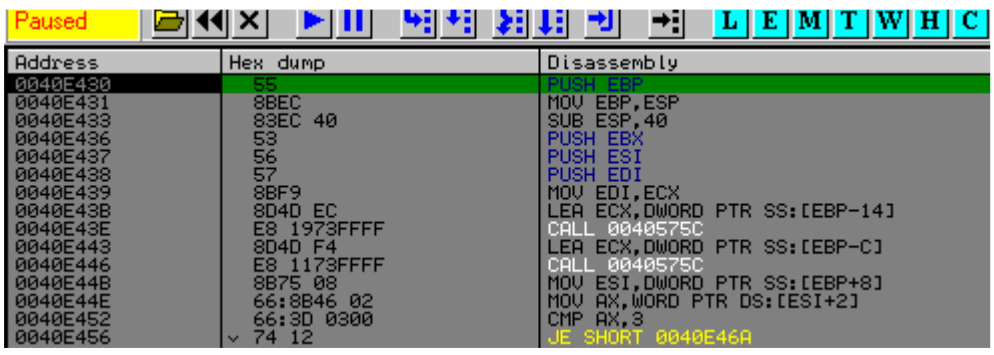
حالا کلید F12 را بزنید تا برنامه متوقف شود و بعد از آن کلید ALT + F9 را بزنید تا وارد حالت Back to user شوید.بعد هم کلید OK را بزنید تا این پیغام از بین برود و برنامه در اینجا متوقف می شود:



تابع MessageBoxA از داخل تابع بالایی صدا زده شده.خوب،با موس به بالا بروید تا ابتدای این Routine را پیدا کنید:

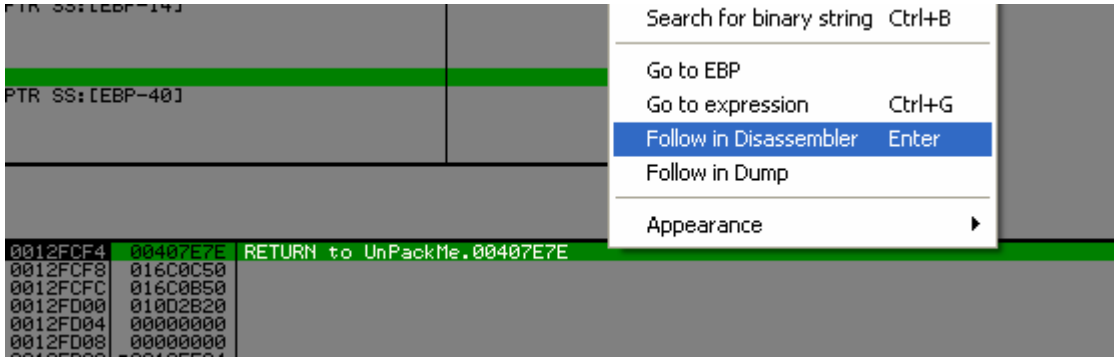


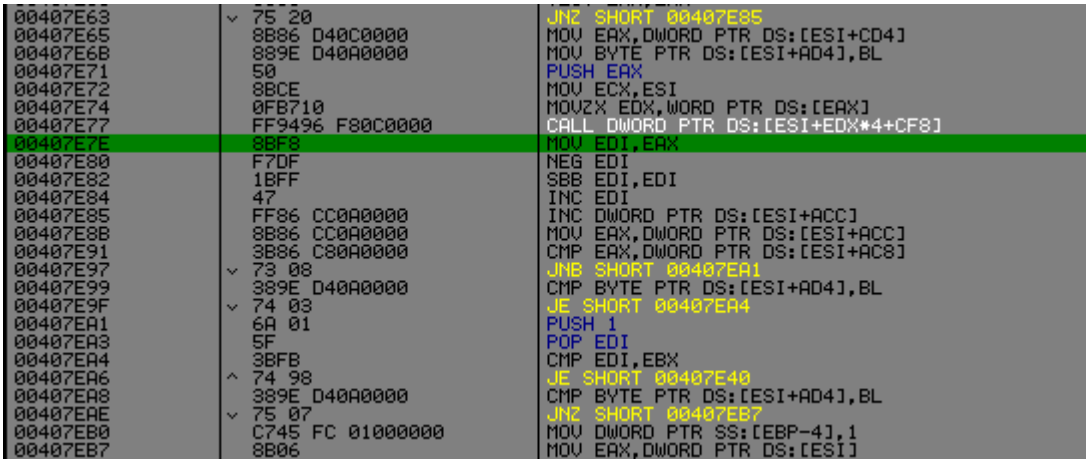
اینجا ابتدای Routine است.بر روی آن Hardware breakpoint on execution بگذارید و برنامه را ری استارت کنید و دوباره اجرا کنید:



ما در جایی که Breakpoint گذاشتیم متوقف شدیم.حالا باید ببینیم این Routine از کجا صدا زده شده؟

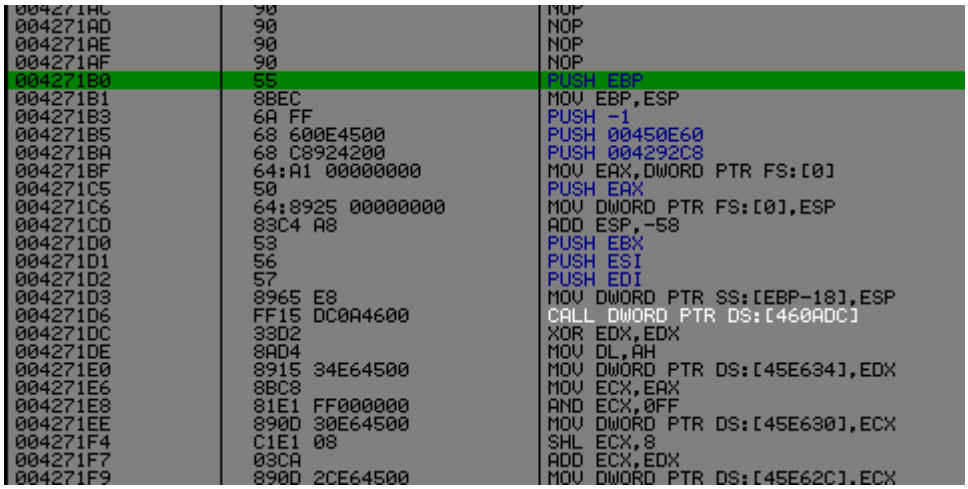
در پنجره دامپ همانند تصویر کلیک راست کرده و گزینه Follow in disassembler را بزنید:



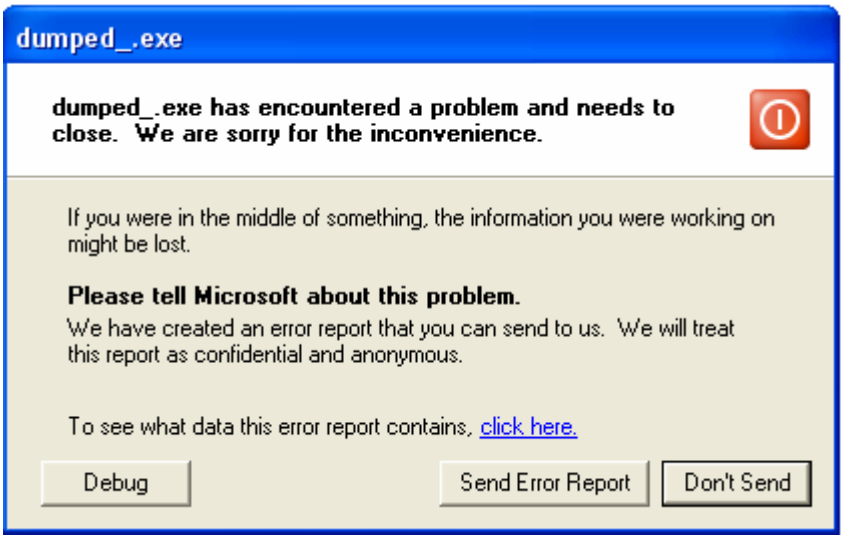


حالا هم دوباره همان عمل را انجام دهید.یعنی با موس بـه ابتـدای Rotine برویـد و بـر روی آن Hardware breakpoint on execution بگذارید.

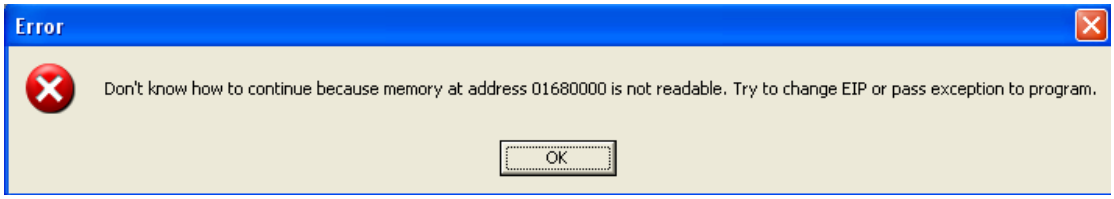
چند بار این عمل را انجام دهید.(یعنی ابتدای Routine را پیدا می کنید.Hardware breakpoint می گذارید،برنامه را ری استارت مـی کنید و با استفاده از پنجره دامپ جایی که این Routine صدا زده شده را پیدا کنید.)
تا بالاخره به اینجا می رسید:



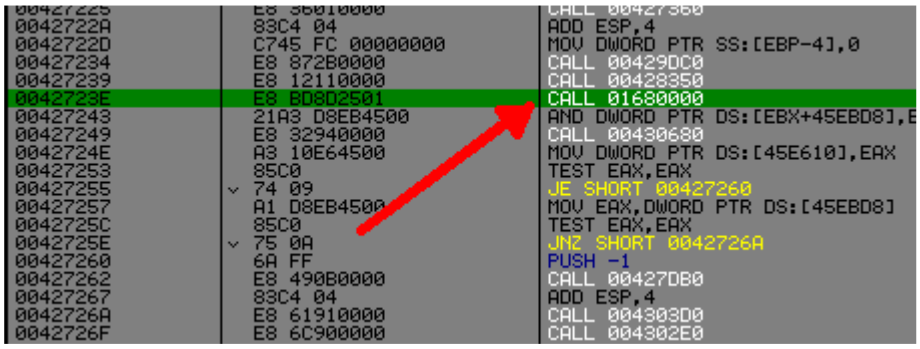
اینجا OEP ماست...از کجا فهمیدم؟...خوب چون چند خط بعد از این تابع GetVersion وجود دارد و در ضمن اگر بر روی این خط هم یک Hardware breakpoint بگذارید.و بخواهید با پنجره Dump جایی که این تابع صدا زده شده را پیدا کنید.می بینید که در پنجره Dump چیزی نیست ☺
حالا بر روی OEP، Hardware bp بگذارید و دوباره برنامه 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:



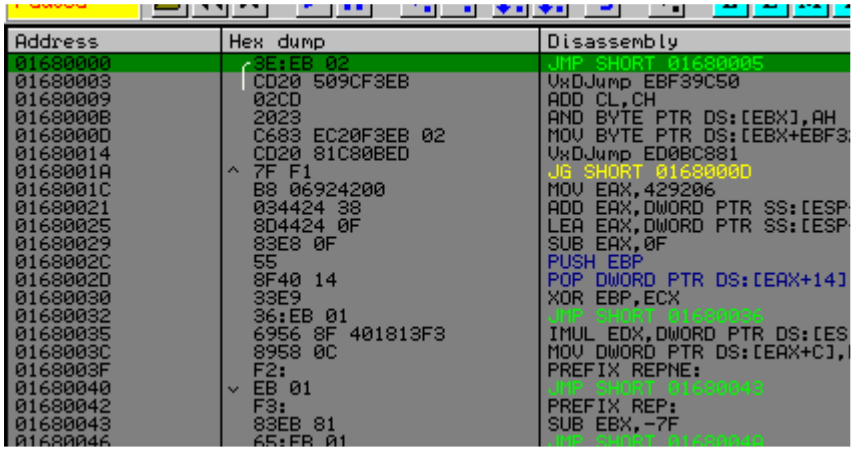
فایل دامپ شده اجرا نمی شود ☹ اصولا در مورد هر پکری به این مشکل برخوردید و دیدید که فایل شما اجرا نمی شود.همیشه فایل آنپک شده را با OllyDBG باز می کنید تا ببینید مشکل چیست؟...چرا فایل درست جواب نمی دهد؟...خطا در چه خطی اتفاق می افتد؟...چه طور می شود مشکل را حل کرد؟
یک OllyDBG دیگر باز کنید و فایل آنپک 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:



می بینید چند خط زیر OEP در فایل اصلی ما دستور Call 01680000 وجود دارد.بر روی آن کلید Enter را بزنید:

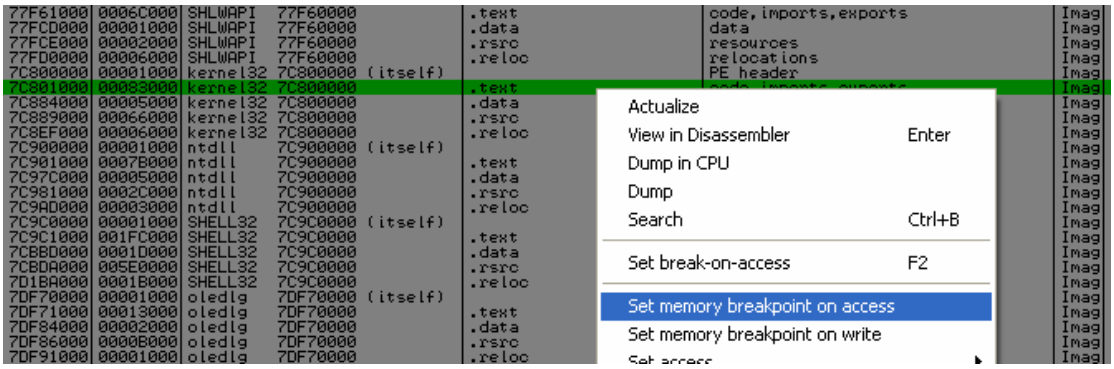


می بینید؟این آدرس در فایل پک شده وجود دارد.ولی در فایل آنپک شده وجود ندارد(در واقع Asprotect قبل از رسیدن به OEP این آدرس ها را تهیه کرده.و این سکشن ها را ساخته است.)
با اندکی تلاش متوجه موضوعی خواهید شد.ببینید ما در تعمیر IAT به مشکلی برنخوردیم...چرا؟...چون Asprotect با IAT کاری نداشته...و آنها را همونجوری 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:

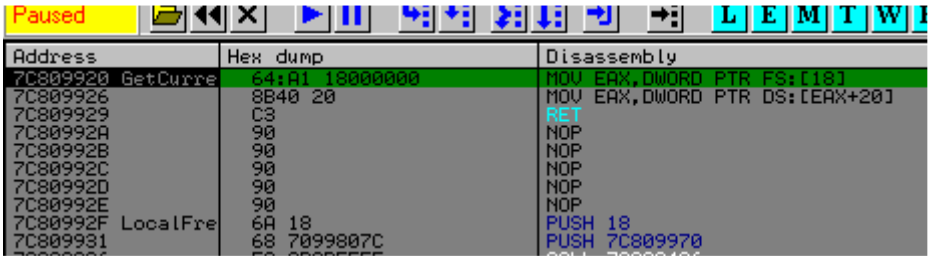
Call DWORD PTR DS: [xxxxxx]

خوب،حالا برای حل این مشکل چه کار کنیم؟...یک راه حل این است که ما بیاییم و سکشنی را که در آن خط 1680000 قرار دارد، به فایل آنپک 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:

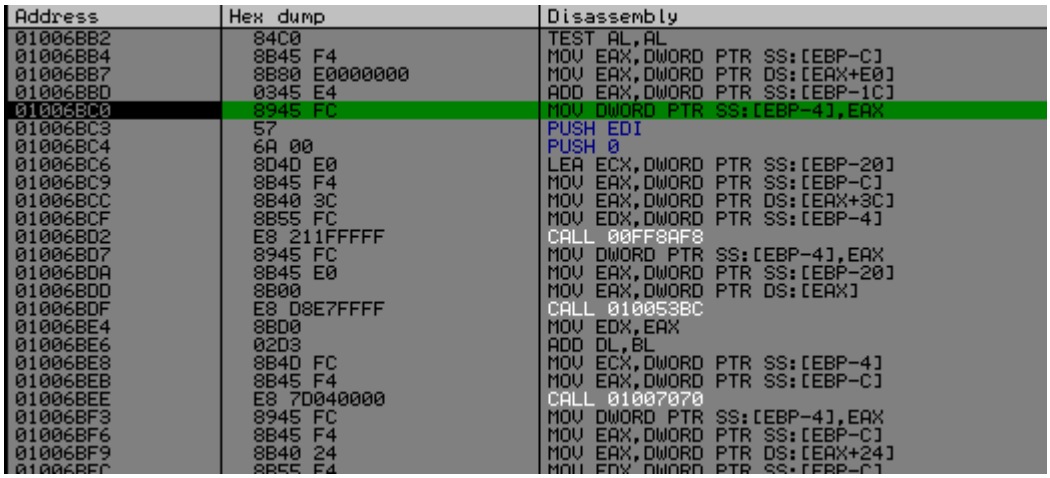


حالا برنامه را با F9 اجرا کنید:

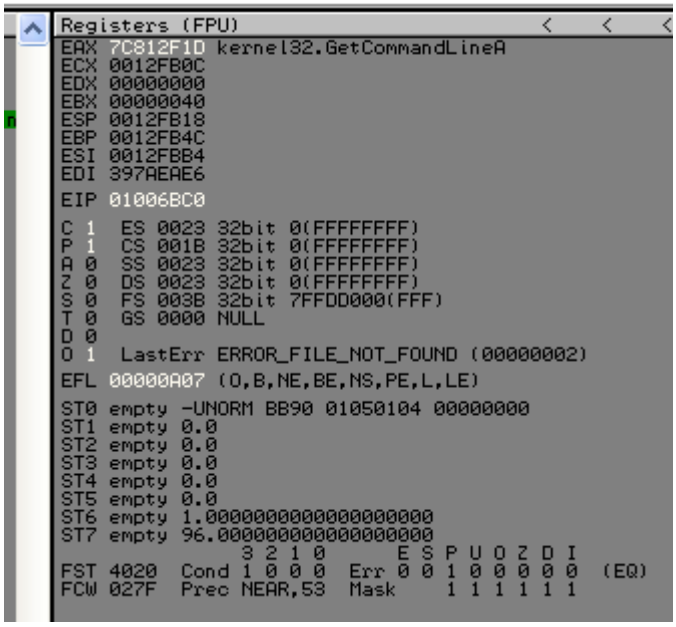


حالا سه بار کلید F9 را بزنید تا به اینجا برسیم:

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:

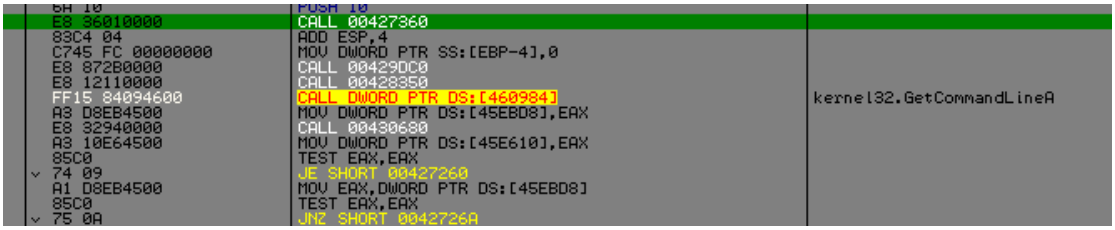


توجه کنید که این آدرس ها در سیستم شما فرق دارد.بر روی جایی که الان هستید،یک Hardware breakpoint بگذارید، حالا نگاهی به رجیسترها بیندازید:

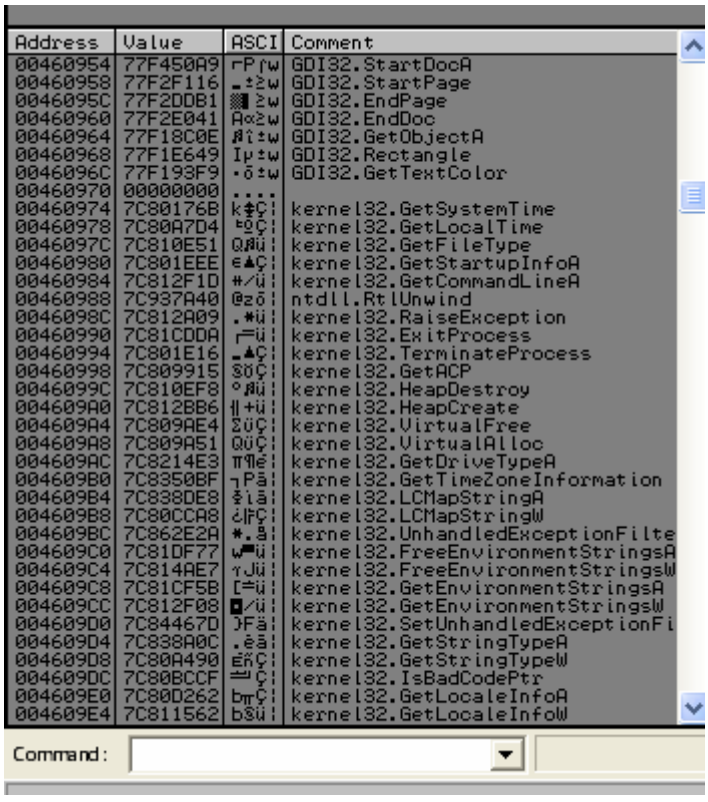


متغیر EAX الان نشاندهنده خطی است که تابعی که در خط (42723E) قرار دارد در واقع به آنجا می رود ☺ پس این خط در واقع به GetCommandLineA می رود ☺
دوباره به آدرس 42723E بروید و جای دستور CALL 1680000 بنویسید:

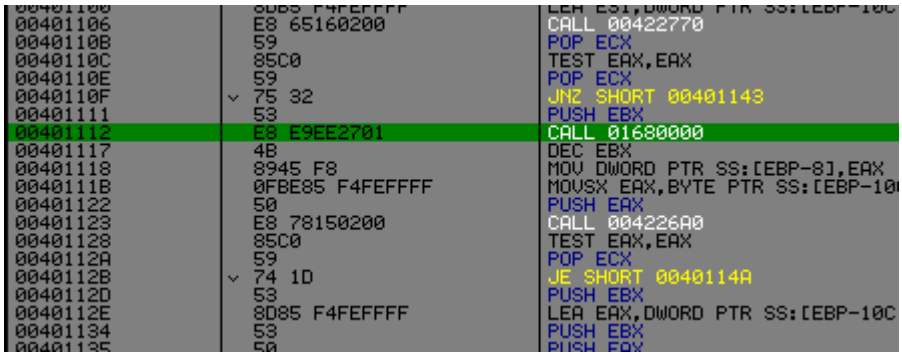
Call Dword ptr DS: [460984]



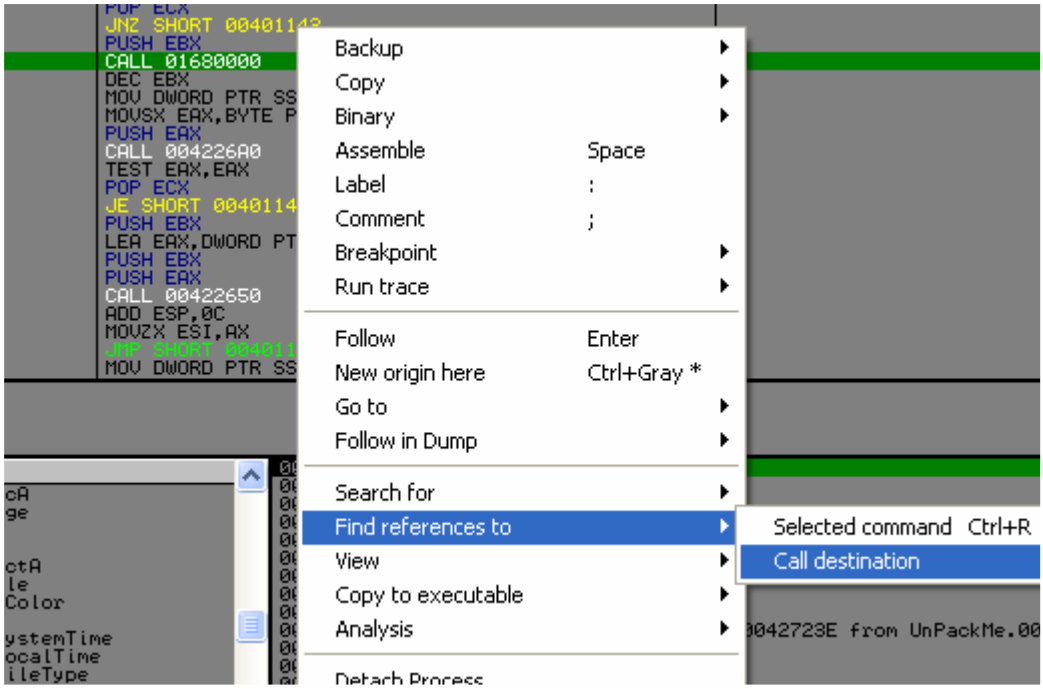
حواستان باشد که توابع را باید به این ترتیب درست کنید.یعنی از IAT استفاده کنید.الان آدرس تابع GetCommandLineA در خط 460984 قرار دارد(در IAT برنامه که از خط 460814 شروع می شود)،نگاهی به IAT بیندازید:

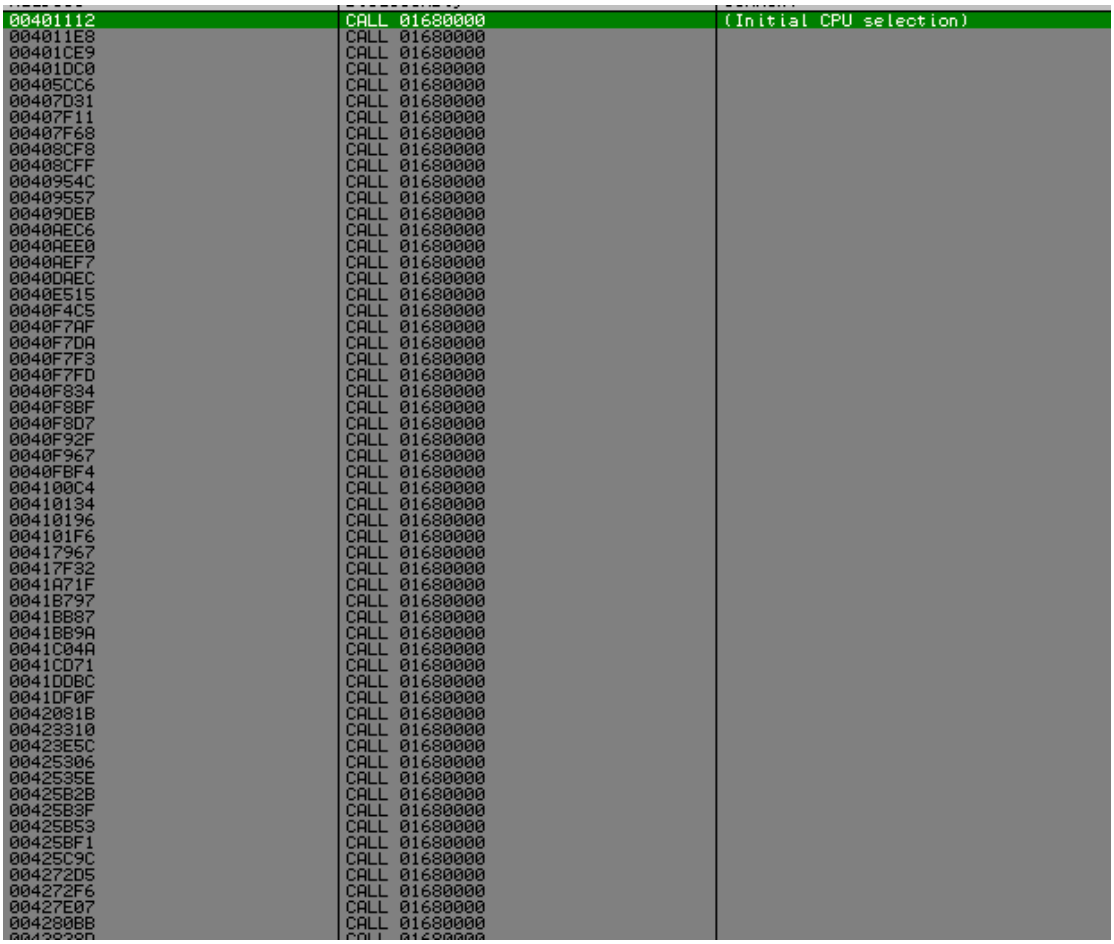


خوب...حالا به نظرتان چند تا از توابع فایل توسط Asprotect دستکاری شده است؟
کلیدهای CTRL + F را بزنید و تایپ کنید: CALL 1680000

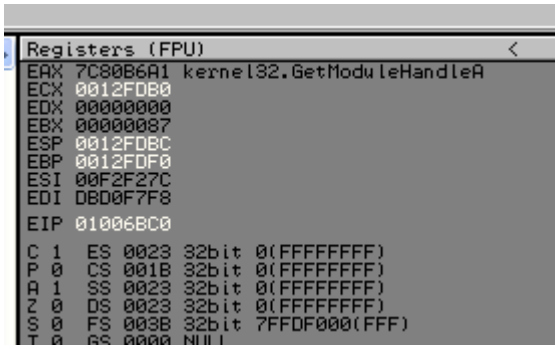


بفرما ! یک تابع دیگر پیدا کردیم.بذارید ببینیم چند تا از این دستورها هست؟...همانند تصویر کلیک راست کرده و گزینه Find refrences to و سپس گزینه Call Destination را بزنید:

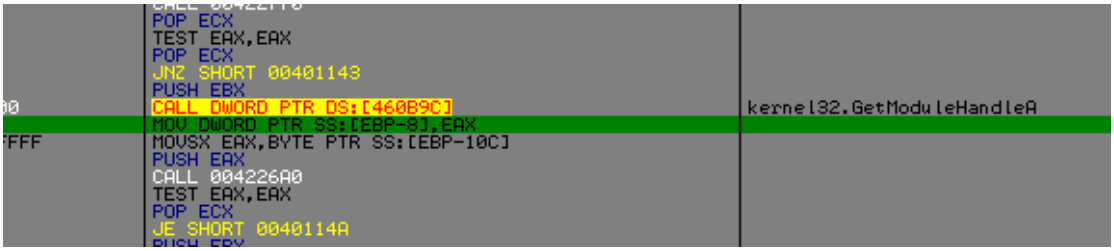




می بینید؟...این همه از این دستورها داریم که باید همه آنها را فیکس کنیم...همانطور که می بینید آنپک کردن این فایل اصلا کار سختی نیست...فقط وقت زیادی را از ما می گیرد.بذارید یک تابع دیگر را هم فیکس کنیم،اولین تابع که در خط 401112 قرار دارد را انتخاب می کنم.بر روی آن کلیک راست کرده و گزینه new origin here را بزنید و بعد کلید F9 را بزنید(حتما باید توی اون خطی که قبلا بهتون گفته بودم،Hardware breakpoint بگذارید،اون Memory BP را هم باید پاک کنید) :



باز هم مثل قبل مقدار رجیستر EAX تابع اصلی ماست.پس این خط را هم اینگونه فیکس می کنیم:



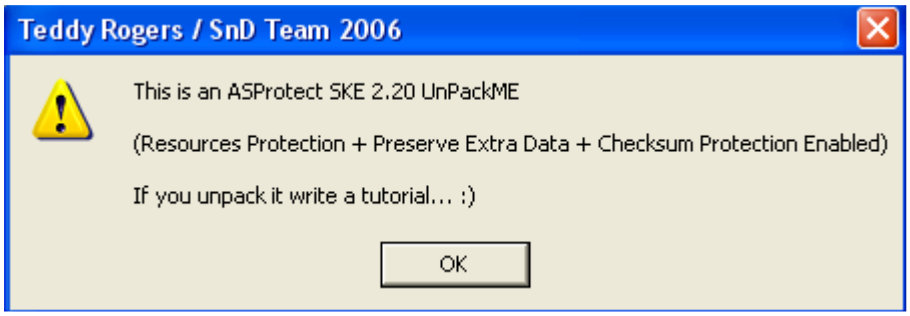
اگر بخواهیم تمام این Call ها را فیکس کنیم...دو سه ساعتی از ما وقت می گیرد ☹ پس بهتر است یک اسکریپت بنویسم(این اسکریپت به هیچ وجه یک اسکریپت برای Asproctect نیست،فقط برای اینکه کارم سریعتر برای این فایل پیش برود این اسکریپت را نوشتم) :

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 Call 1680000 باشم

Loop:
findop a,#E8????2? 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 Call 1680000 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 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:

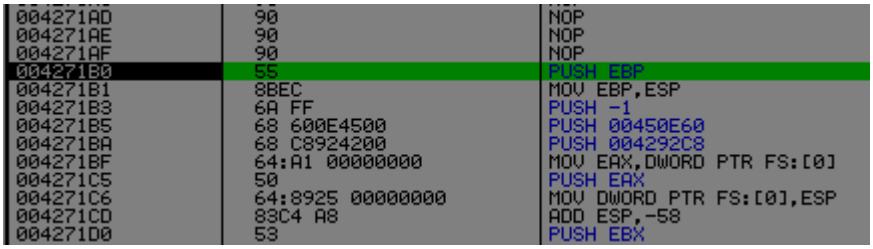
حالا بر روی OEP یک New origin here بگذارید و بعد از فایل دوباره دامپ بگیرید و بعد فایل دامپ را با ImportREC فیکس کنید و فایل دامپ شده را اجرا کنید:



این بار فایل بی هیچ مشکلی اجرا می شود ☺

راه حل 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 ☺

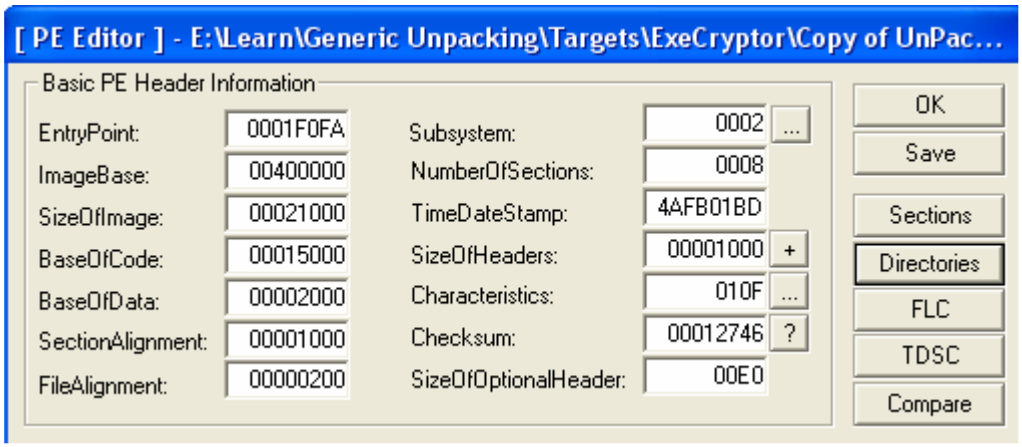


همانطور که می بینید اسکریپت در OEP متوقف شده است ☺
خوب،اسکریپت از فایل دامپ هم گرفته و با نام de_UnpackMe.exe ذخیره کرده است.حالا تنها کاری که لازم است انجام دهید این است که با ImportREC این فایل دامپ را فیکس کنید ☺

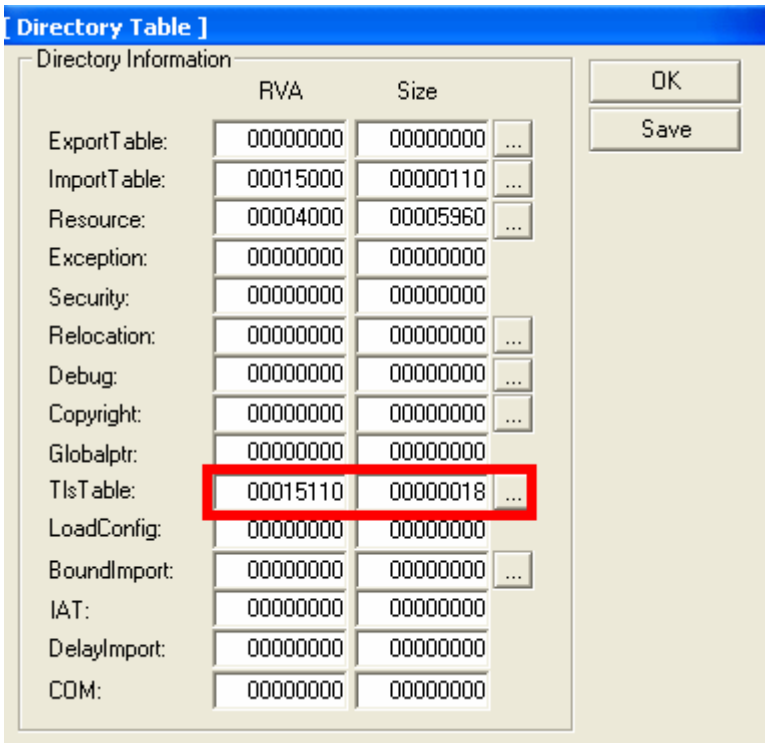
ExeCryptor

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 ...با TLS 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 ☹
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:



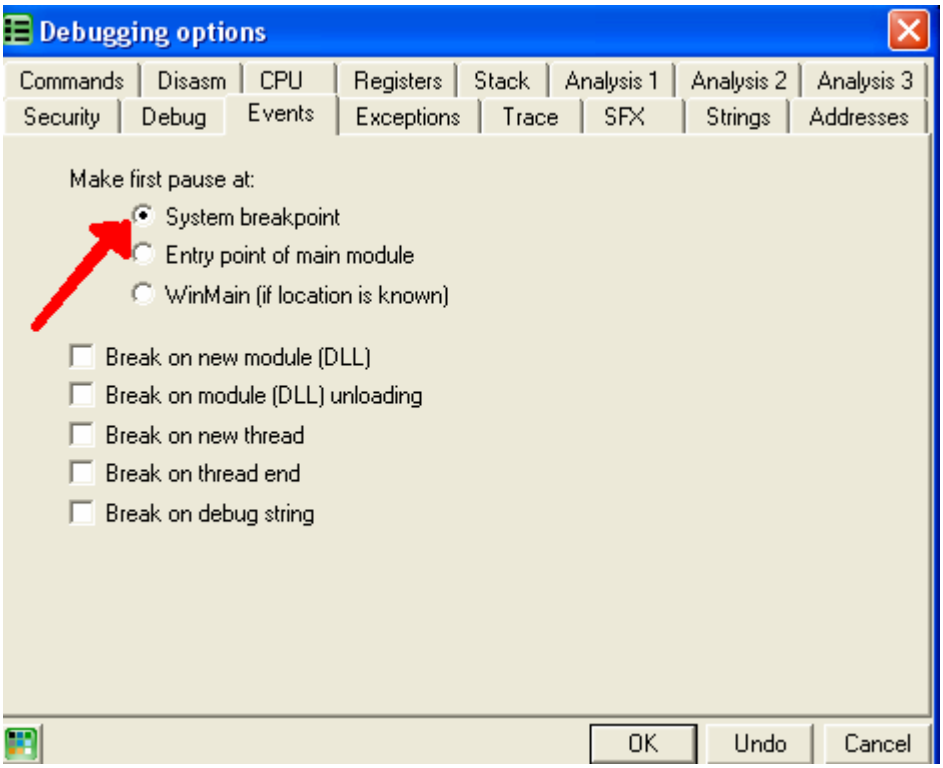
گزینه Directories را بزنید:



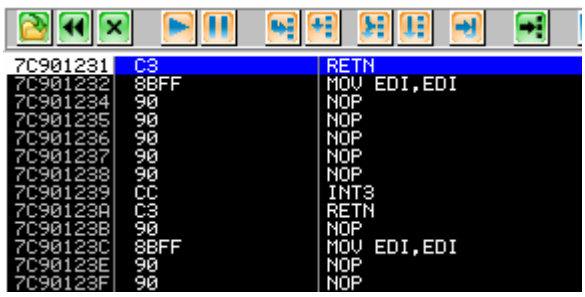
همانطور که می بینید مقدار TlsTable صفز نیست...یعنی این برنامه از TLS CallBack Function استفاده می کند.ما اگر این دو را صفر کنیم.به طور کامل TLS CallBack Function از بین می رود و برنامه در OllyDBG اجرا می شود.این دو مقدار را صفر کنید و گزینه Save را بزنید.

راه حل دیگر:

روش قبل برای از بین بردن TLS CallBack Function روش خوبیست...اما همیشه جواب نمی دهد.چرا که ExeCryptor ،قابلیت Integerity 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:

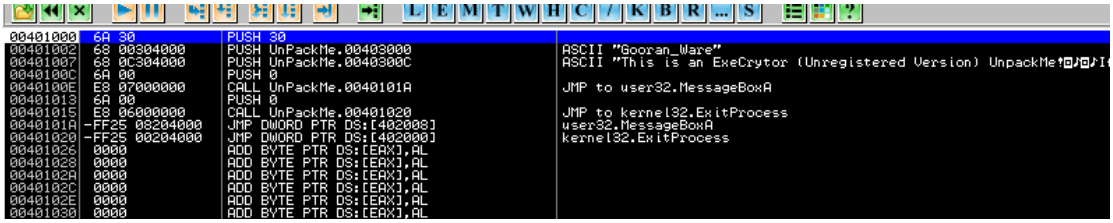


این بار در System BP متوقف شدیم نه در Entry point فایل،خوب برای رد کردن آنتی دیباگی که در TLS CallBack Function قرار دارد،در منوی View گزینه Breakpoints را بزنید:

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 پیغام که مربوط به رجیستر نبودن برنامه است مواجه شوید:

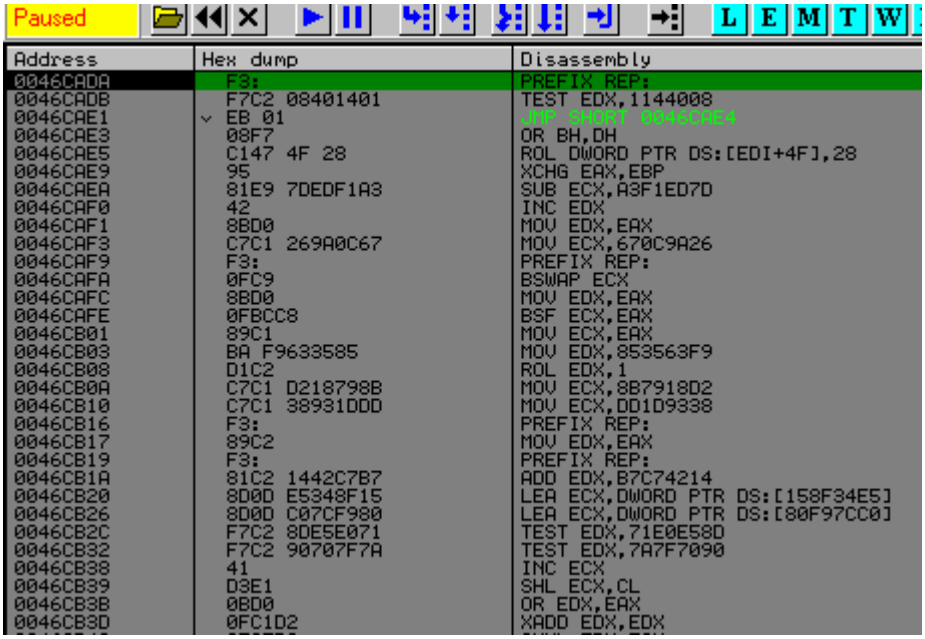
این پیغام موقعی نمایش داده شده است که کدهای سکشن اصلی برنامه کامل Decrypt شده است.پس می توانیم بر روی سکشن text. یک Memory Breakpoint بگذاریم.حالا کلید OK را بزنید تا این پیغام برود و در اینجا که OEP ماست،متوقف شویم:



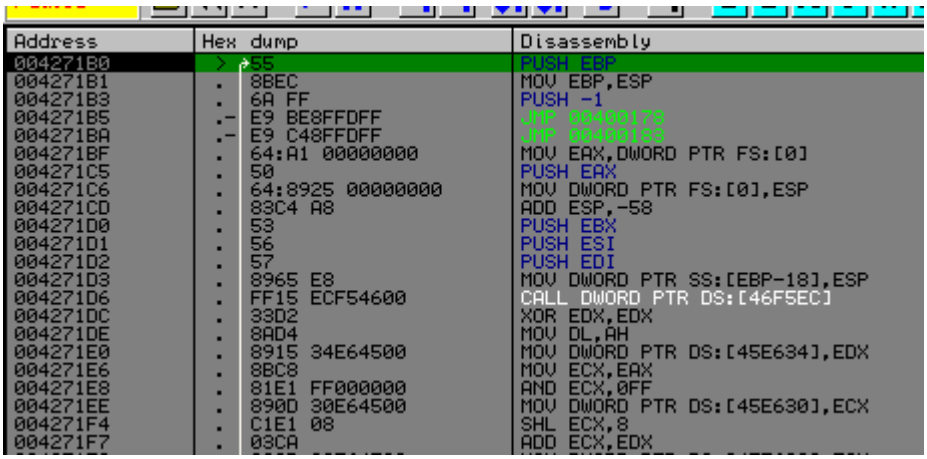
همانطور که می بینید برنامه ما در MASM نوشته شده و کدهای خیلی کمی دارد.بعد از این دیگر مشکلی نیست،دامپ می گیرید و فیکس می کنید ☺

PeSpin

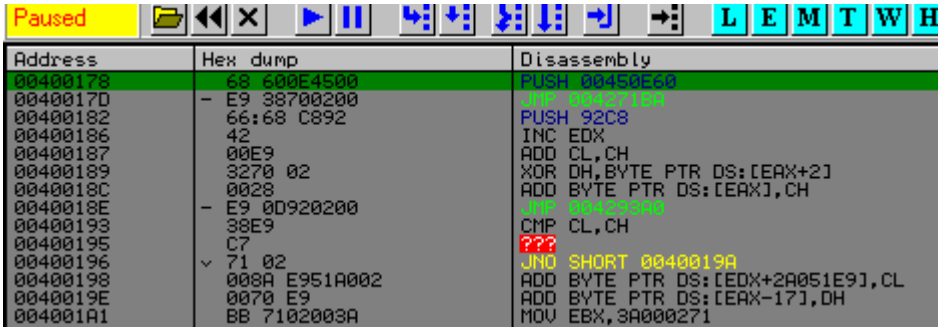
برای یافتن OEP در PeSpin می توانیم از رجیستر ESP استفاده کنیم.دو بار کلید F8 را بزنید تا دستور PushAD اجرا شود و بعد بر روی مقدار رجیستر ESP یک Hardware Breakpoint on access بگذارید،و برنامه را با F9 اجرا کنید:



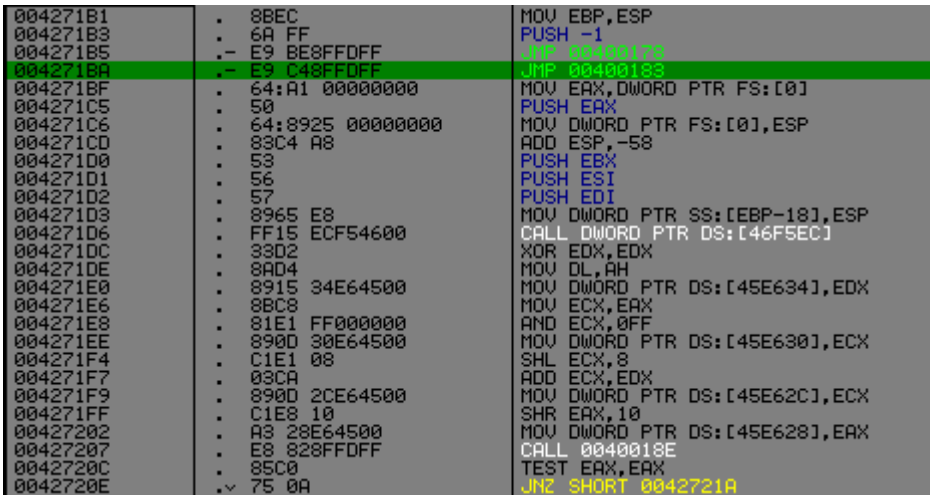
یکبار دیگر F9 را بزنید:



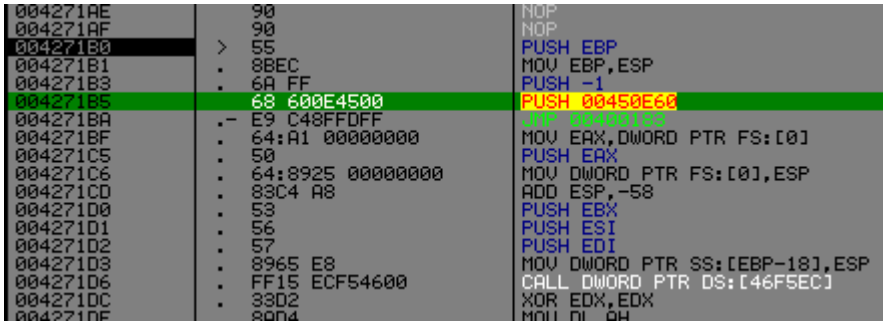
اینبار به OEP رسیدیم.به پرشی که چند خط زیر OEP قرار دارد(JMP 00400178) ،با دقت نگاه کنید.ببینید در این پرش چه اتفاقی می افتد؟...بر روی آن کلید Enter را بزنید تا ببینیم این پرش به کجا می رود؟



خوب،در اینجا دستور Push 00450E60 اجرا می شود،حالا بر روی پرشی که زیر Push 00450E60 قرار دارد،کلید Enter را بزنید.

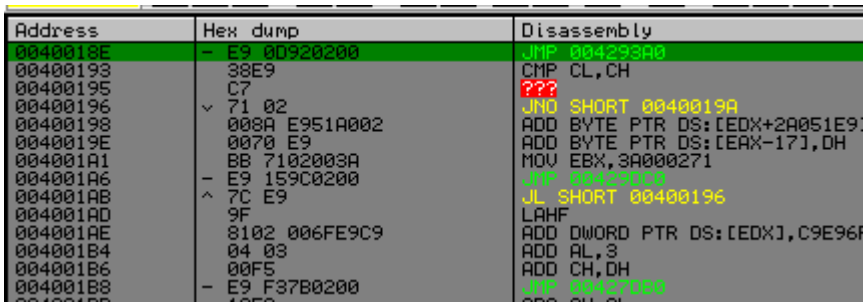


می بینید؟...دوباره به نزدیکی های OEP برگشتیم...این یعنی چی؟...یعنی پکر ما تعدادی از کدهای برنامه را منتقل کرده به یک آدرس دیگر(کدها را به PE Header منتقل کرده است)...به این قابلیت پکرها می گویند: Code Redirection ...خوب،حالا ما چه کار کنیم؟...ما باید تمام این Code Redirection ها را درست کنیم ☹ مثلا پرشی 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:

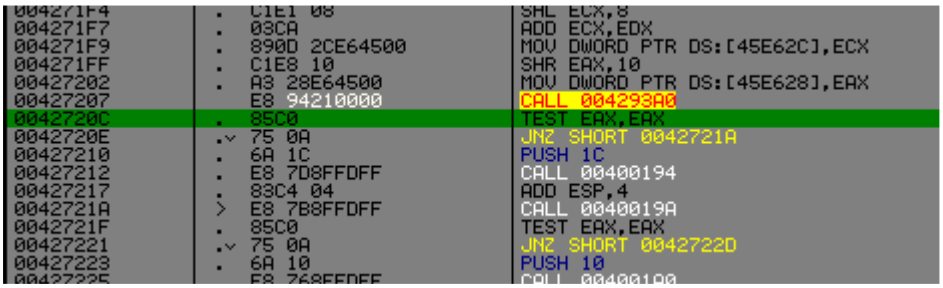


PeSpin فقط با پرش (JMP) کدها را Redirect نمی کند،بلکه با CALL هم این کار را می کند.به خط 427207 نگاه کنید:

خوب،بر روی آن کلید Enter را بزنید:



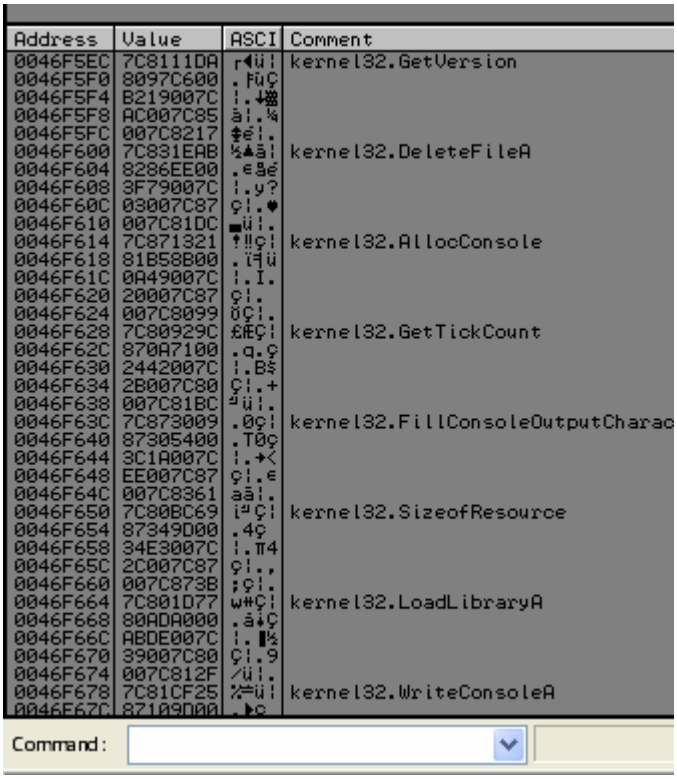
می بینید؟...این CALL به جای اینکه مستقیم به خط مورد نظرش برود.از یک پرش به عنوان واسطه استفاده می کند و این پرش در واقع جایی است که این CALL باید برود.خوب پس این CALL در واقع به خط 004293A0 می رود.پس این مورد را هم تصحیح کنید:



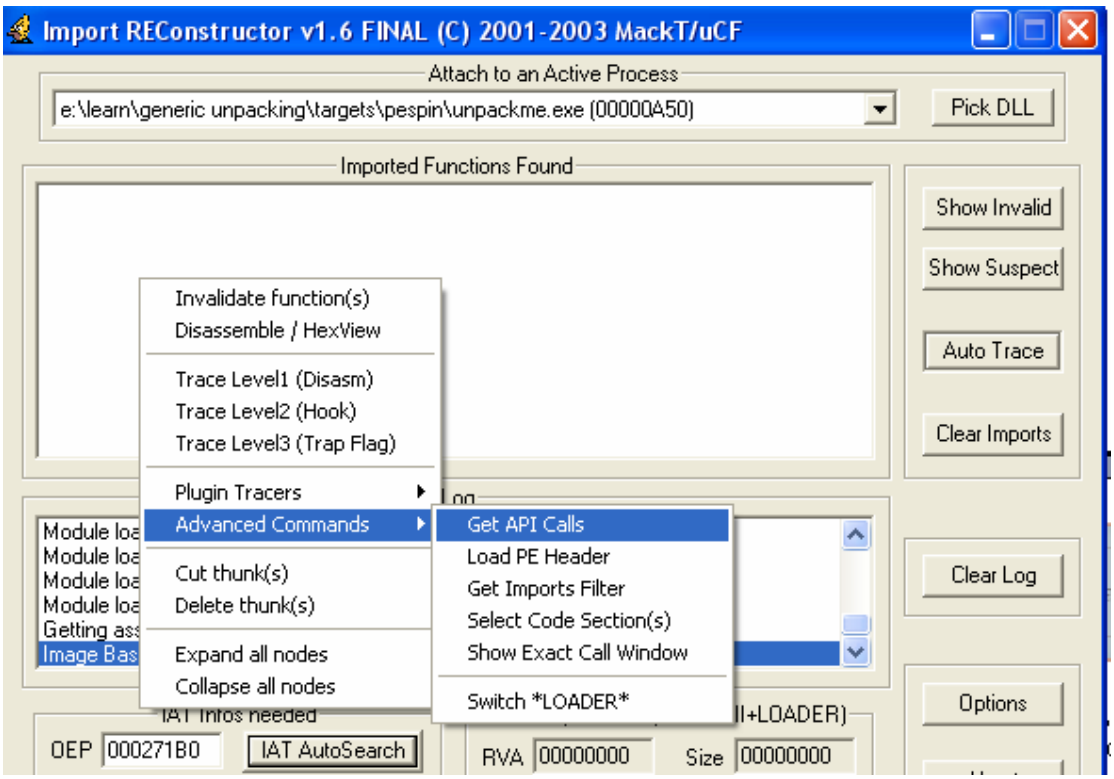
خوب،حالا باید از OEP (4271B0) شروع کنیم و تمام JUMP ها و CALL هایی که Redirect شده اند را درست کنیم ☹ می دانم که کار وقت گیری است.اما خوب،چاره ی دیگری نیست.جز اینکه بخوایم یک اسکرپیت بنویسیم.خوب حالا از OEP شروع کنید،تا به اینجا برسید:

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:

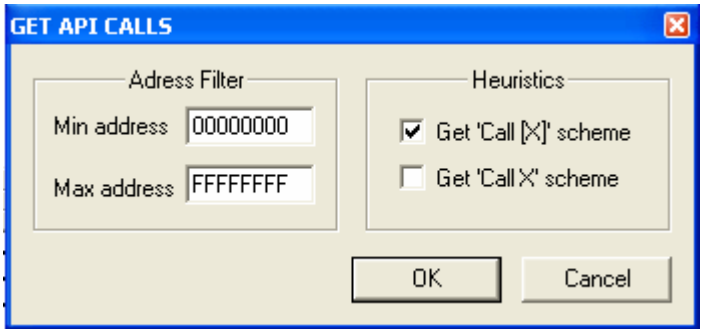
ImportREC نمی تواند IAT را پیدا کند ☹ پس بهتر است خودمان ببینیم چه بلایی سر IAT آمده؟
چند خط زیر OEP تابع GetVersion وجود دارد(خط 4271D6)...بر روی این خط کلیک راست کرده،گزینه Follow in dump و سپس گزینه Memory Address را بزنید:



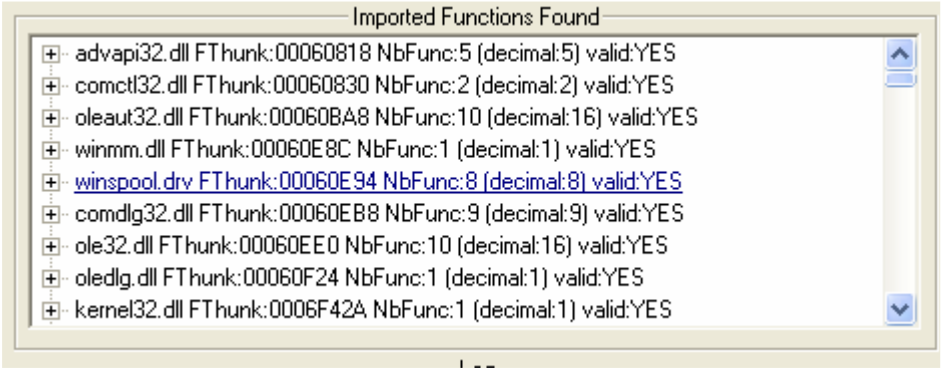
همانطور که می بینید PeSpin ،IAT را کامل به هم ریخته و آدرس توابع را با فاصله نوشته است.حالا چه کار کنیم؟...ما می توانیم به ImportREC بگوییم که تمام Call هایی که به توابع API می روند را در برنامه پیدا کند و به این ترتیب آدرس تمامی توابع API را پیدا کند.بر روی ImportREC کلیک راست کنید،گزینه Advanced commands و سپس گزینه Get API Calls را بزنید:



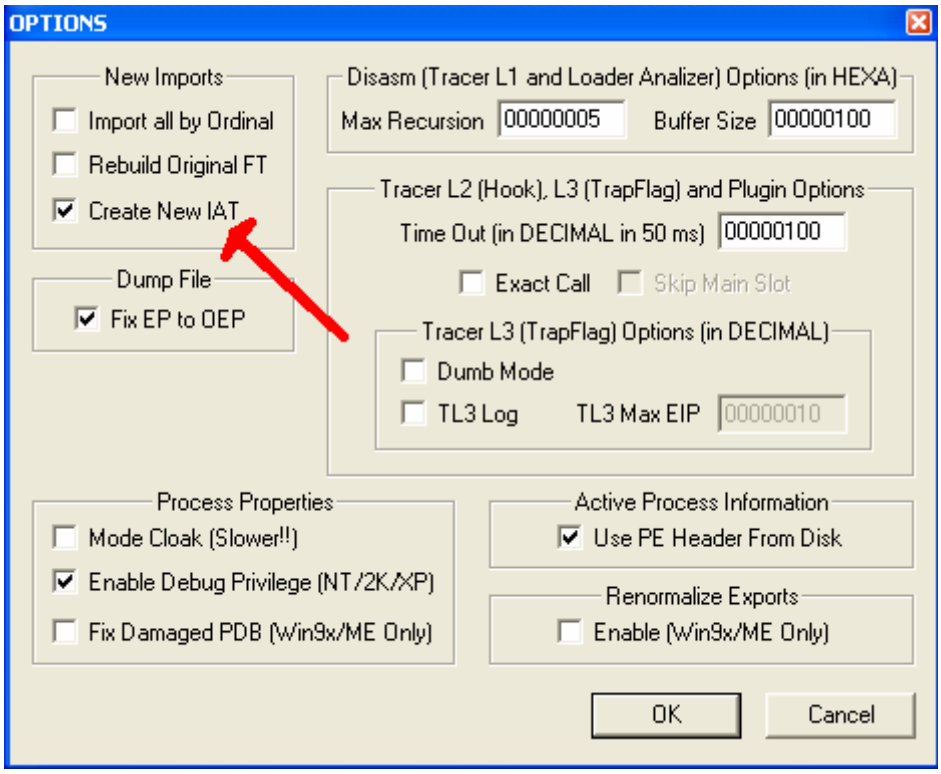
و بعد کلید OK را بزنید:



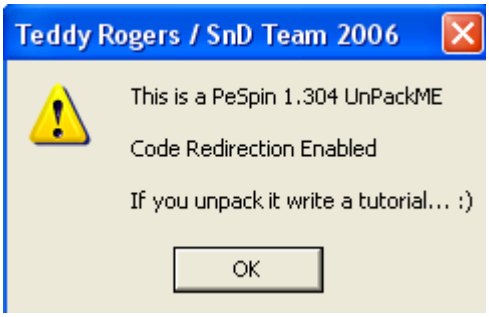
حالا گزینه Show Invalid را بزنید و بر روی توابع Invalid کلیک راست کرده و گزینه Cut Thunk را بزنید:



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):

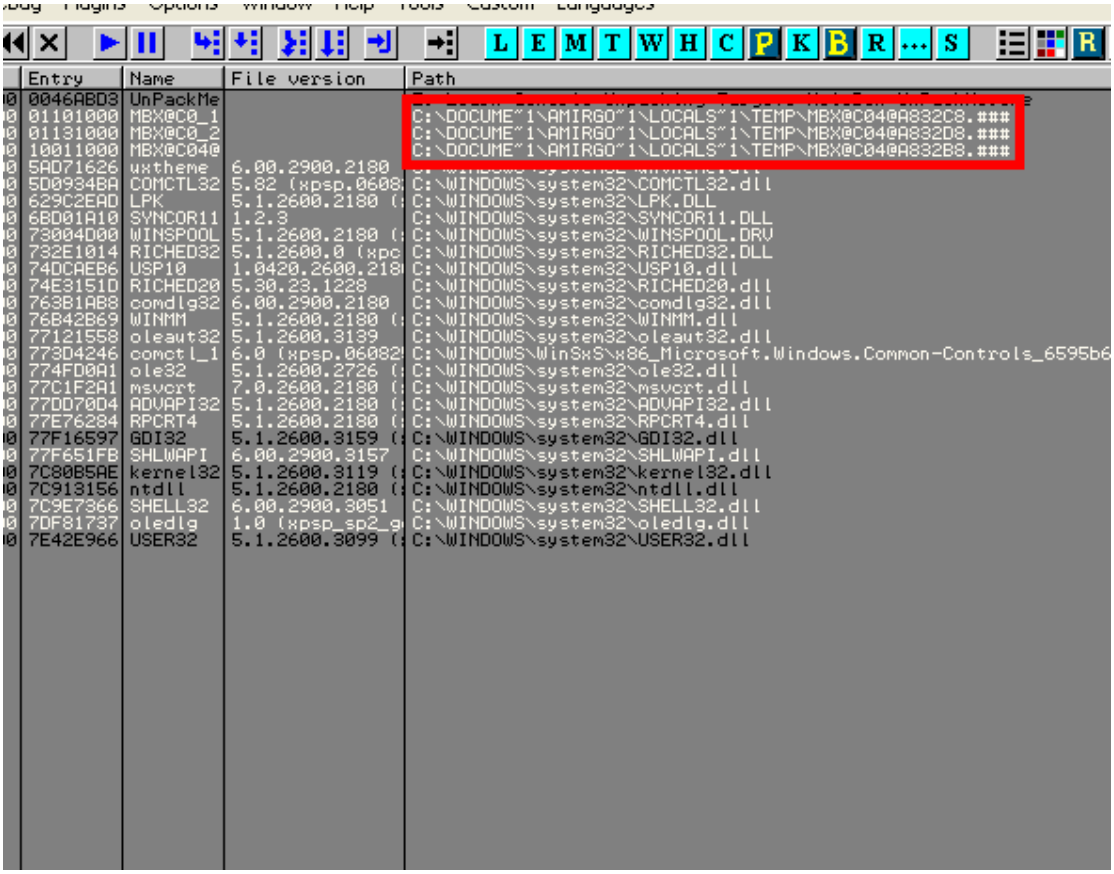


خوب،حالا فایل دامپ خود را فیکس کنید :

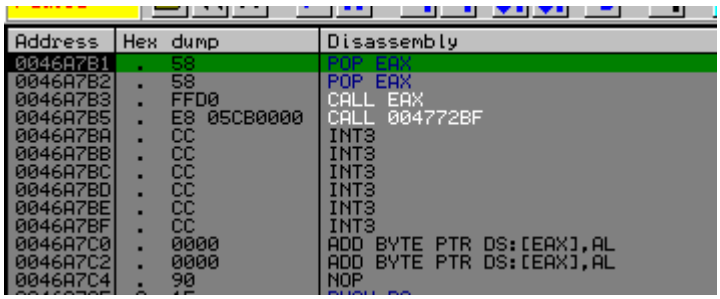


MoleBox

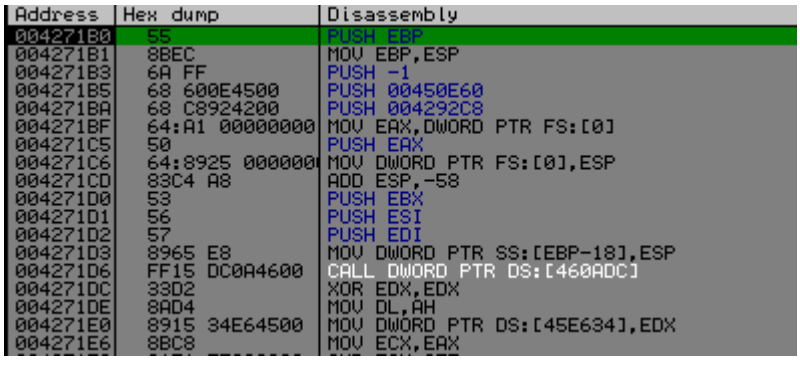
MoleBox یکی دیگر از پکرهایی است که قابلیت Bundle کردن فایل ها را دارد.در این مثال ما باید علاوه بر آنپک کردن فایل Exe، فایل های DLL را هم از آن استخراج کنیم...حالا از کجا بفهمیم چند فایل DLL در درون این فایل قرار دارد؟
فایل مورد نظر را در OllyDBG باز کنید،کلید F9 را بزنید تا برنامه اجرا شود،حالا در منوی View گزینه Executable modules را بزنید:



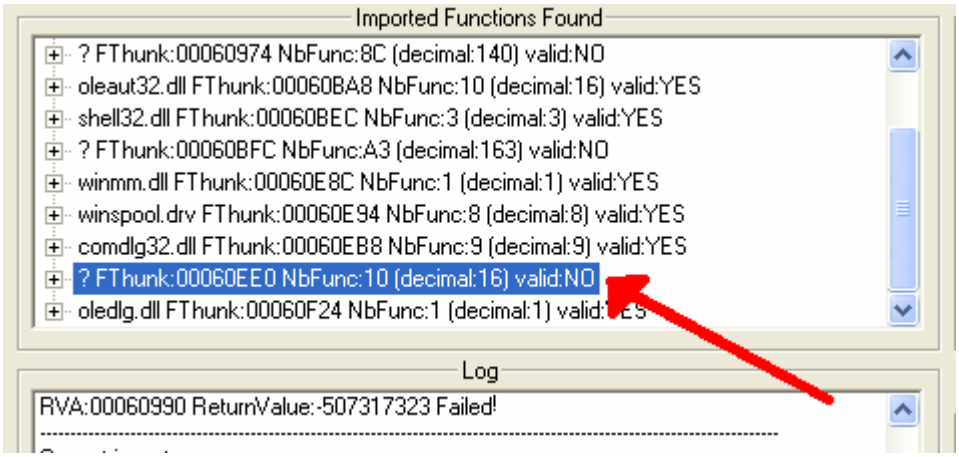
همانطور که می بینید،سه فایل DLL در TEMP قرار دارد که این فایل ها از داخل خود فایل استخراج شده و Load شده اند.
یافتن OEP بسیار راحت است،به راحتی می توانیم از رجیستر ESP برای یافتن OEP استفاده کنیم ☺ دو بار کلید F8 را بزنید تا مقدار رجیستر ESP تغییر کند،بعد بر روی آن یک Hardware bp on access بگذارید و برنامه را با F9 اجرا کنید:



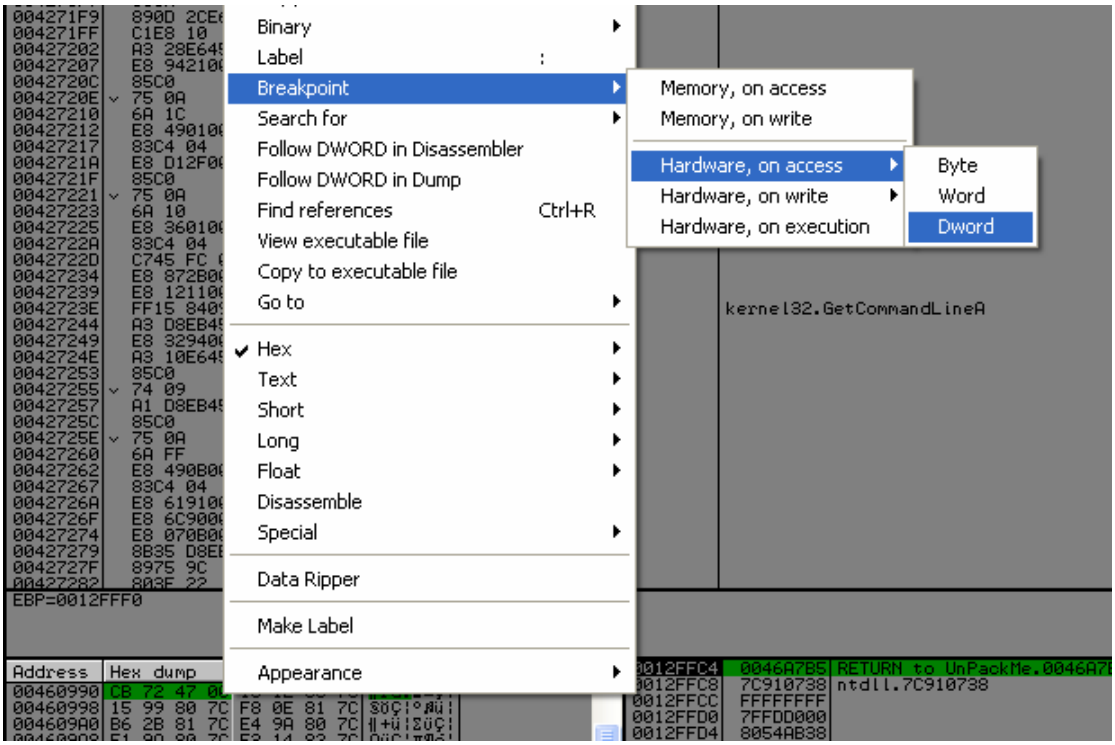
دستور CALL EAX که چند خط پایینتر قرار دارد،ما را به OEP می برد.پس سه بار F7 بزنید تا به OEP برسید:



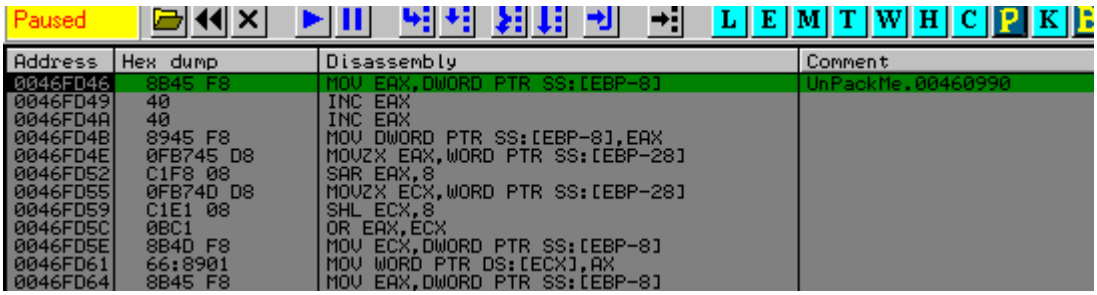
از فایل دامپ بگیرید و ImportREC را باز کنید:



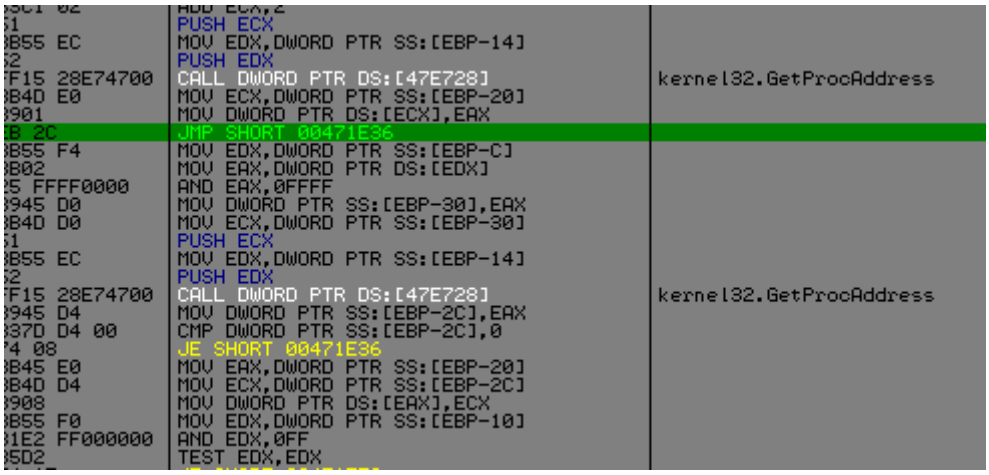
همانطور که می بینید،چند تابع Invalid داریم ☹ برای یافتن این توابع باید در داخل کدهای پکر بگردیم و ببینم کجا این توابع دستکاری می شود و ما آنجا را اصلاح کنیم(یافتن Magic Jump)
بر روی آدرس یکی از توابع Invalid (مثلا خط ۴۶۰۹۹۰) یک Hardware bp on write بگذارید و برنامه را ری استارت کنید:



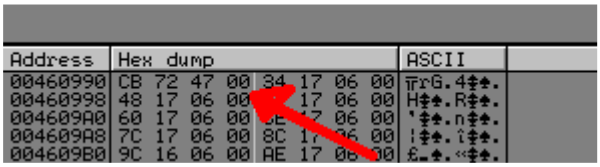
و اجرا کنید:



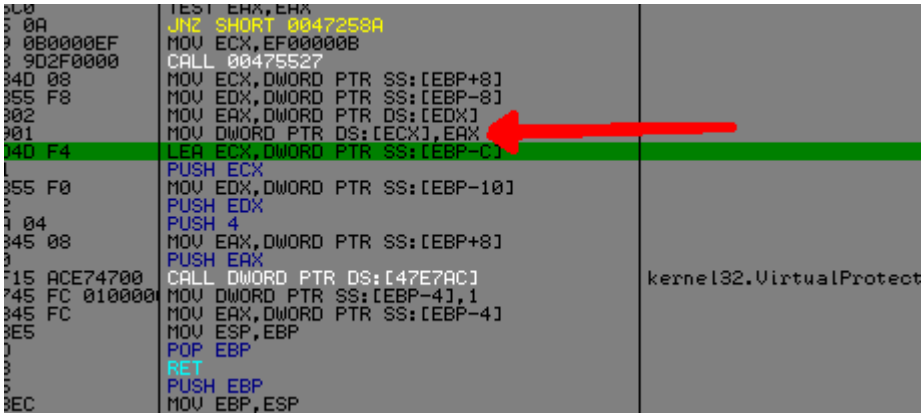
هنوز به جایی که توابع Redirect می شوند،نرسیدیم.(می توانید به خط ۴۶۰۹۹۰ در دامپ نگاه کنید،می بینید که در این آدرس هنوز تابع Redirect شده نوشته نشده است.)
سه بار دیگر F9 بزنید:



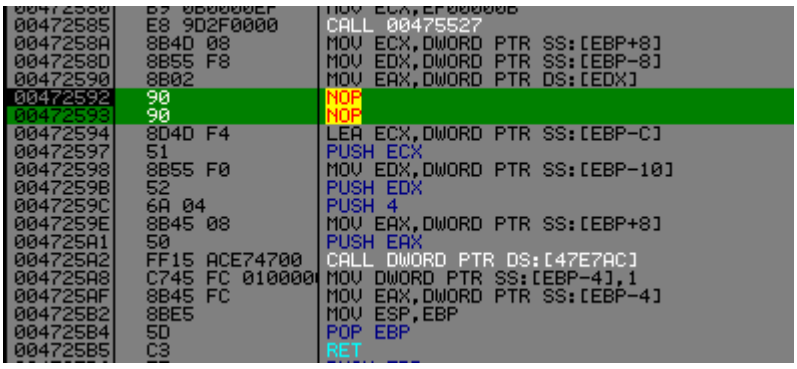
الان اگر نگاهی به مقدار رجیستر EAX بیندازید،متوجه می شوید که این آدرس از IAT در واقع تابع ExitProcess بوده،پس یکبار دیگر F9 بزنید:



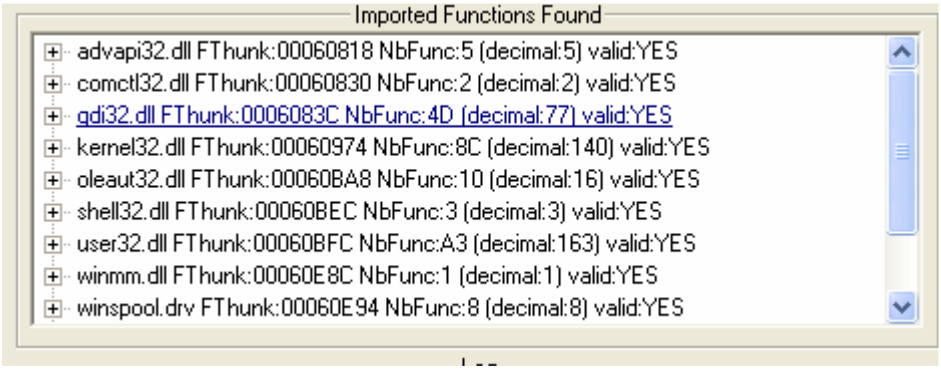
می بینید؟...تابع ExitProcess به خط 4772CB منتقل شده ☹ نگاهی به پنجره Disassembler هم بندازید:



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:

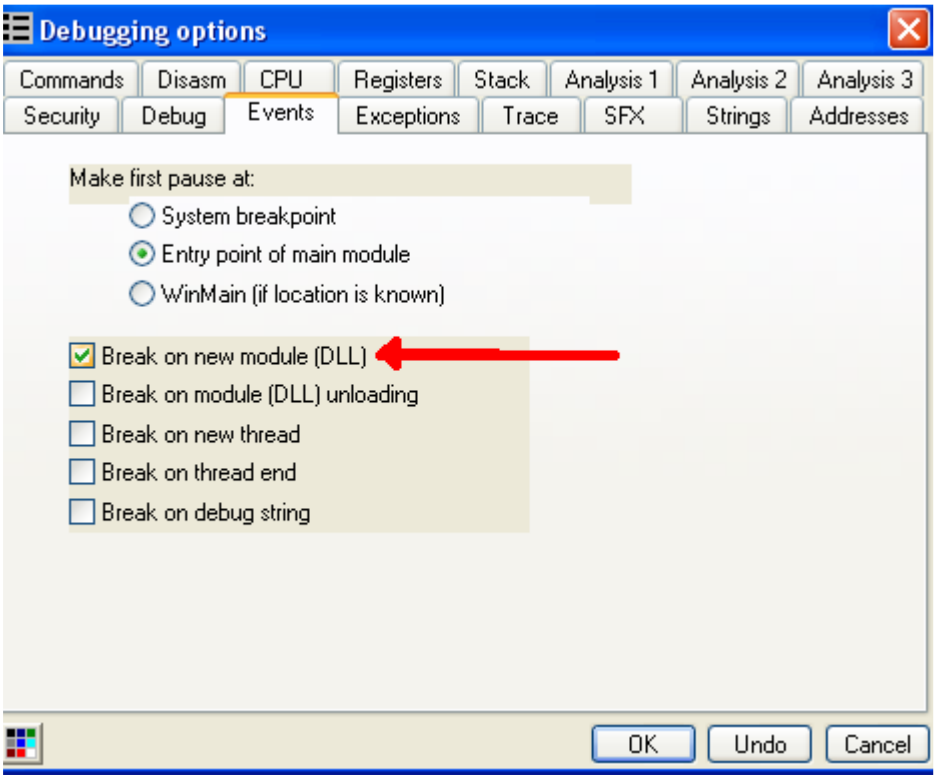


حالا Hardware BP که در اینجا گذاشته بودید،پاک کنید.برنامه را اجرا کنید و دوباره ImportREC را باز کنید:

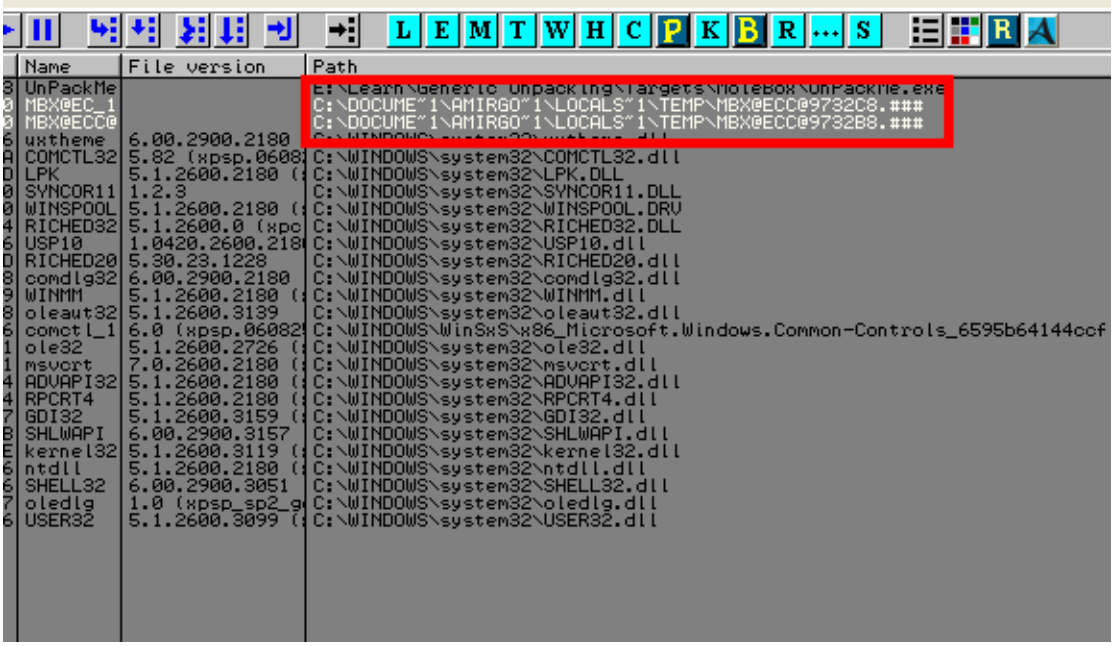


این بار تمامی توابع Valid هستند ☺ حالا فایل دامپ خود را فیکس کنید:

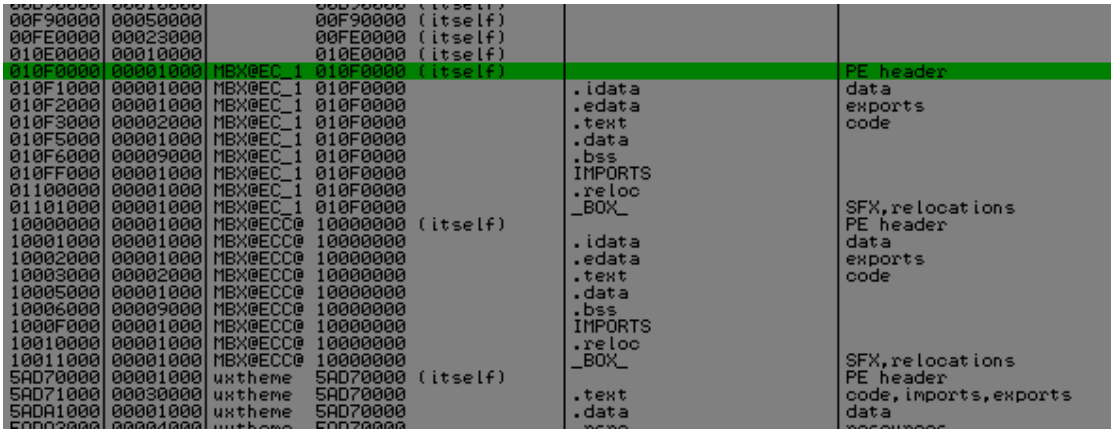
فایل آنپک شده است.اما به خاطر اینکه فایل های DLL را استخراج نکردیم و برنامه این فایل ها را ندارد،دچار خطا می شود.پس باید این فایل ها را استخراج کنم.
برای استخراج DLL اول باید دستور CALL ی را پیدا کنیم که به OEP فایل های DLL می رود.
برنامه را ری استارت کنید.کلیدهای ALT + O را بزنید و وارد قسمت Events شوید.حالا تیک گزینه Break on new module را بزنید:



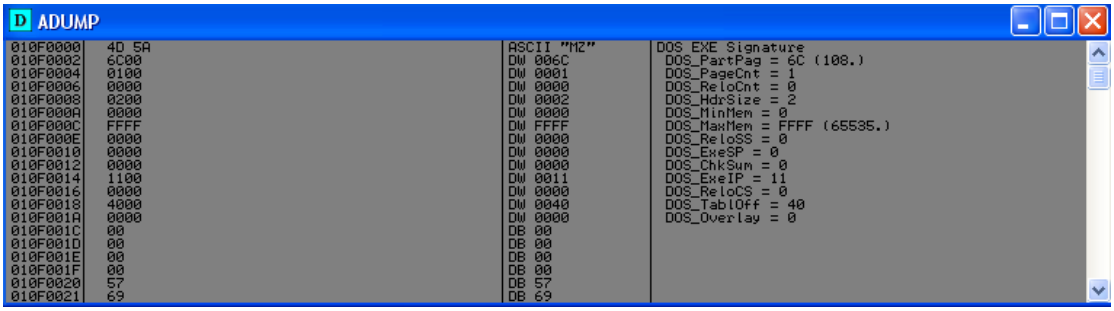
به این ترتیب اگر یک فایل DLL اجرا شود،OllyDBG متوقف می شود.چند بار کلید F9 را بزنید تا فایل های DLL که می خواهیم استخراج کنیم،اجرا شوند:



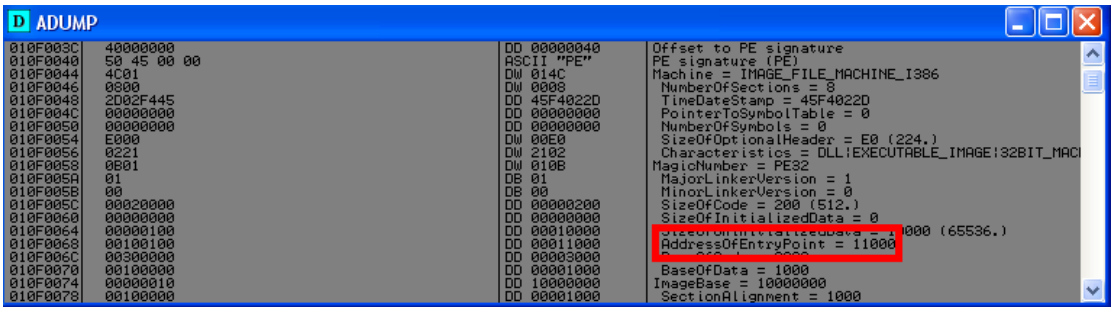
خوب،حالا باید Entry point فایل DLL را پیدا کنیم. کلیدهای ALT + M را بزنید تا وارد Memory Map شوید:



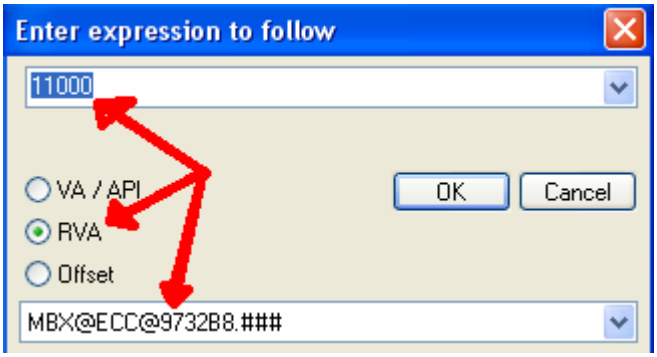
برای یافتن Entry Point می توانیم مشخصات PE Header فایل را در OllyDBG نگاه کنیم.برای دیدن PE Header یک فایل در Memory Map کافی است بر روی PE Header دو بار کلیک کنید.(در تصویر بالا بر روی قسمت سبز رنگ دو بار کلیک کنید.)

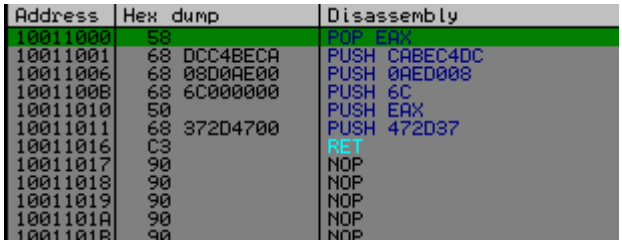


همانطور که ما در PeTools ، LordPE یا PEID مشخصات PE Header فایل را می دیدیم،در OllyDBG هم می توانیم این مشخصات را ببینیم.
در این پنجره به پایین بروید تا به اینجا برسید:

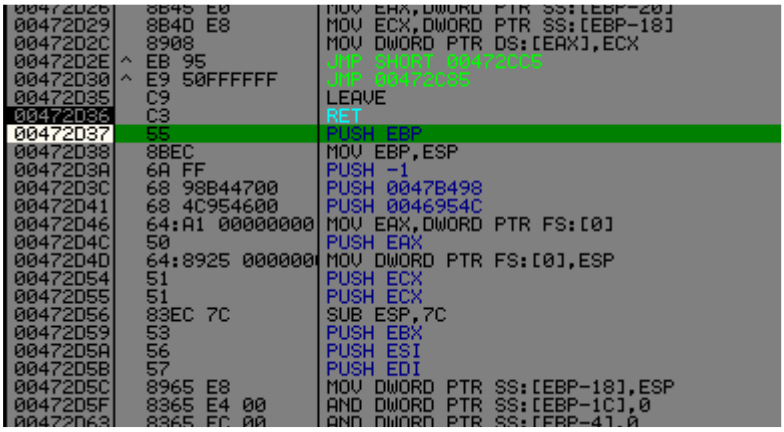


پس Entry point فایل ما بر حسب RVA برابر 11000 است ☺
حالا کلیدهای ALT + C را بزنید تا وارد پنجره CPU شوید و کلیدهای CTRL + G بزنید و به Entry Point فایل بروید:

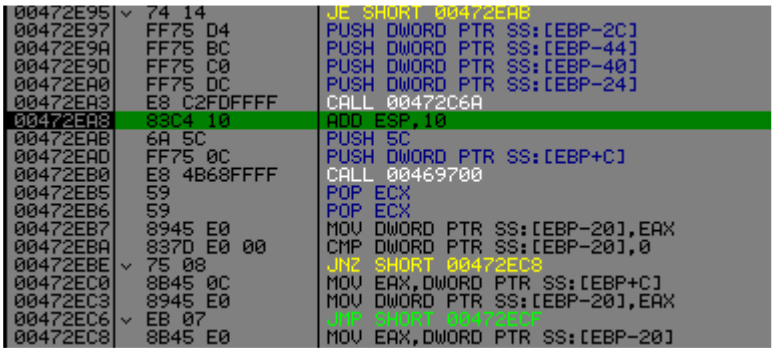




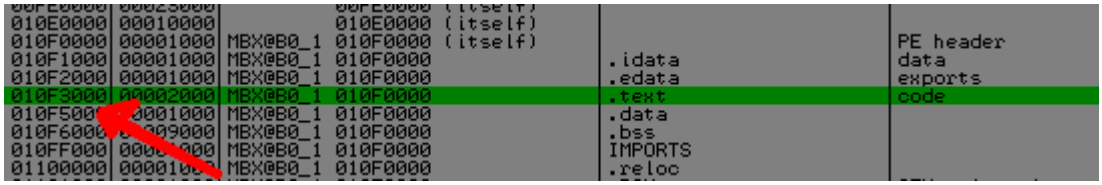
اینجا Entry Point فایل ماست،به خط 10011011 نگاه کنید.دستور Push 472D37 ،مقدار 472D37 را وارد حافظه Stack می کند و با دستور RET به خط 472D37 می رود.(همانطور که قبلا گفتم **Push xxx + RET = JMP xxx**)
پس بر روی خط 472D37 یک BP بگذارید و برنامه را اجرا کنید.



حالا بر روی RET بالای سر آن یک BP بگذارید و برنامه را اجرا کنید،و یک بار کلید F8 را بزنید:



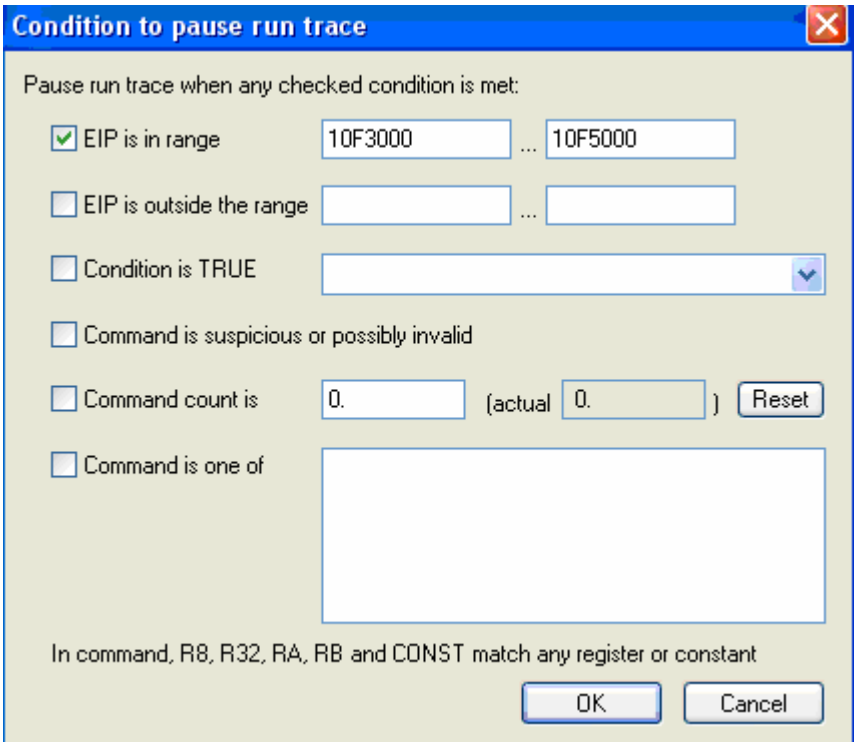
ما الان در کدهای پکر هستیم،حالا برای یافتن OEP می توانیم از Run Trace استفاده کنیم.
Run Trace یکی از قابلیت های بسیار خوب OllyDBG است.OllyDBG با Run Trace با توجه به شرطی که شما داده اید،کدها را اجرا می کند و لیست تمامی خط هایی که اجرا شده را در منوی View ،قسمت Run Trace نگهداری می کند☺
برای اجرای یک Run Trace ابتدا سایز و شروع سکشن اصلی این فایل DLL را از Memory Map بگیرید:



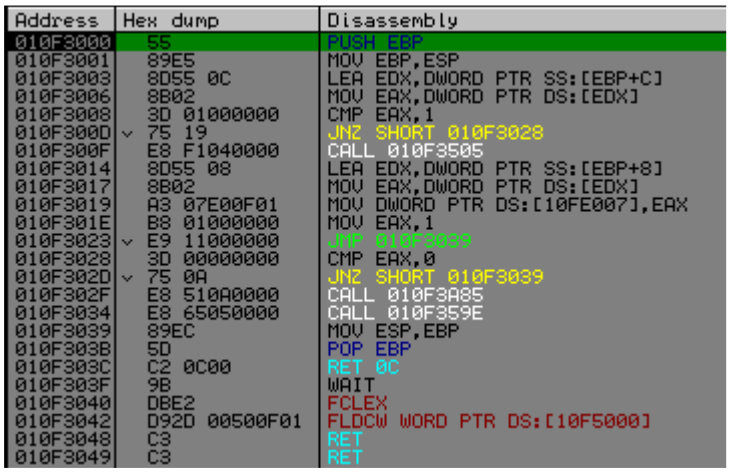
همانطور که می بینید محل شروع سکشن text. در این فایل DLL برابر 10F3000 است و سایز آن هم ۲۰۰۰ است.پس:

End of code section=Start + Size= 10F3000 + 2000 = 10F5000

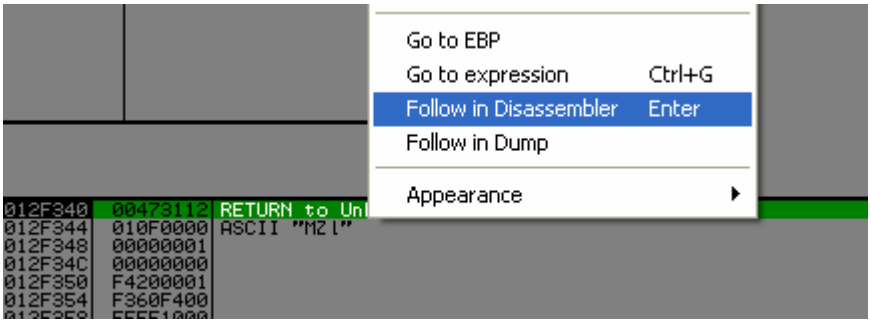
البته ممکن است این مقادیر در سیستم شما فرق داشته باشد.در اینجا مقدار ImageBase(محل شروع PE Header) برابر 10F0000 است.این مقدار را به خاطر داشته باشید.چون بعدا به دردمان خواهد خورد.
حالا برای اجرا Run Trace کلیدهای CTRL + T را بزنید و در قسمت EIP in range همانند تصویر بنویسید:

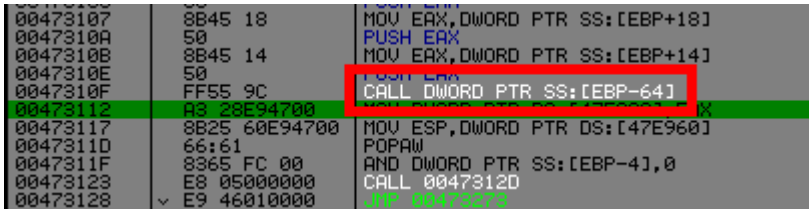


ما می خواهیم ببینیم که در چه موقعی وارد سکشن text. در این فایل DLL می شود.پس نوشتم هر وقت مقدار EIP بین 10F3000 و 10F5000 باشد.عملیات Trace تمام شود.
کلید OK را بزنید و برای اجرای Trace کلیدهای CTRL + F11 را بزنید.(کلید CTRL + F11 همانند کلید F7 در Trace عمل می کند و کلید CTRL + F12 همانند کلید F8 ☺)
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) :

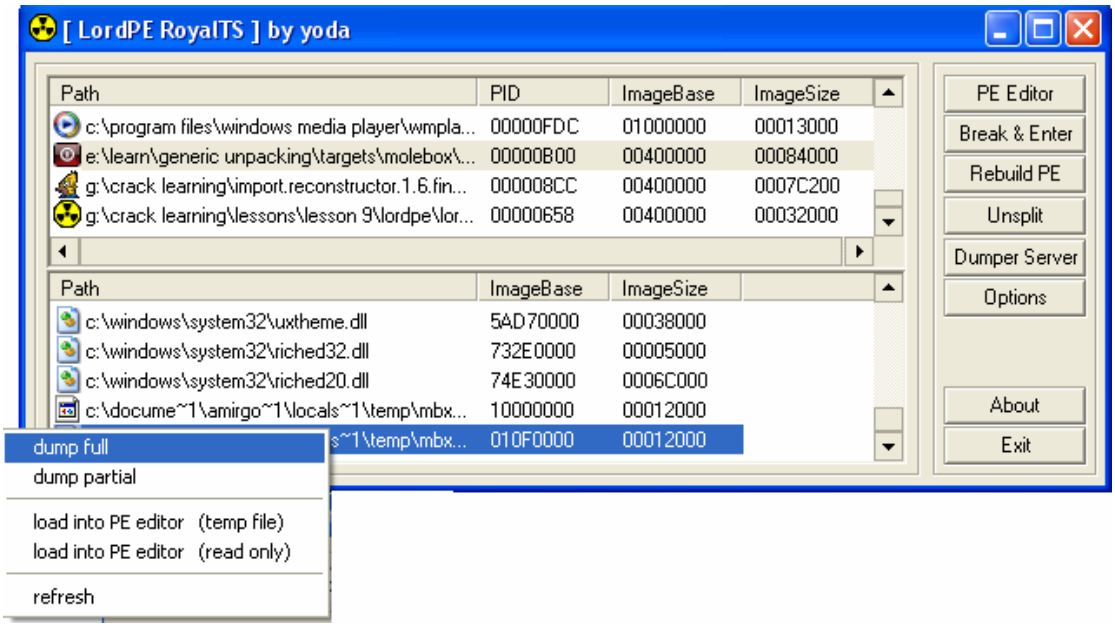


اینجا OEP این فایل DLL ماست.چون در MoleBox ،OEP تمامی DLL ها توسط یک CALL فراخوانده می شود،ما اگر آن CALL را پیدا کنیم،می توانیم OEP سایر DLL ها را هم پیدا کنیم ☺
پس در پنجره Stack کلیک راست کرده و گزینه Follow in disassembler را بزنید تا بفهمیم از کجا به اینجا آمدیم:

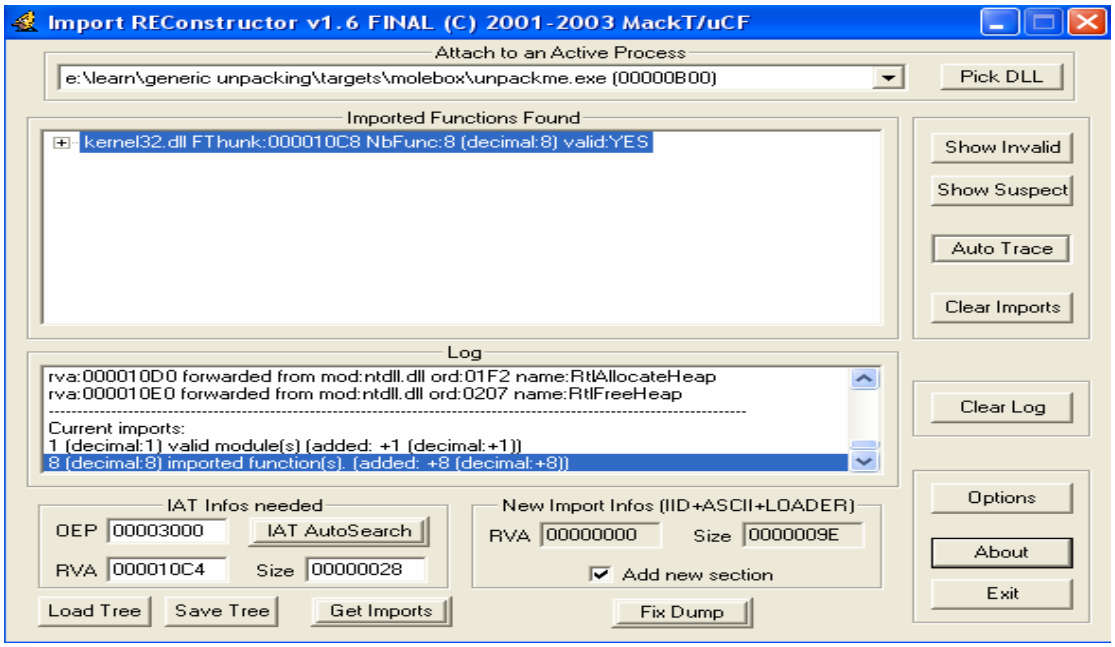




پس این CALL که در خط 473112 قرار دارد،همان CALL ی است که به OEP تمامی DLL ها می رود.تمامی Breakpoint ها را پاک کنید(همچنین تیک گزینه Break on new module در قسمت تنظیمات را بردارید.)و فقط بر روی این دستور hardware bp on execution بگذارید،و برنامه را دوباره اجرا کنید.
کلا سه بار در این خط متوقف می شویم.که هر سه بار این دستور به OEP یکی از این فایل های DLL می رود.(هر سه فایل DLL،OEP آنها برابر 3000 است.البته بر حسب RVA)
با LordPE(یا نرم افزارهای مشابه) دامپ بگیرید:



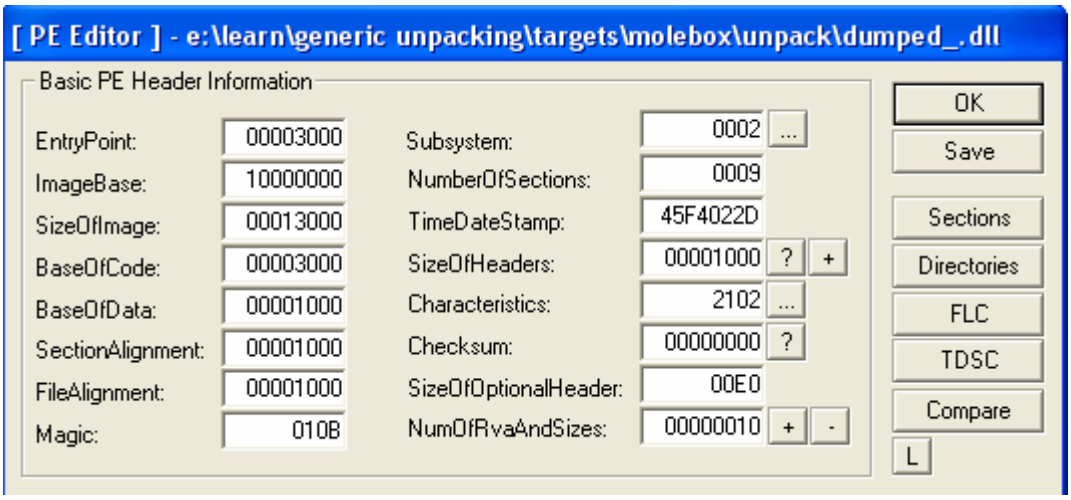
و با ImportREC فیکس کنید:



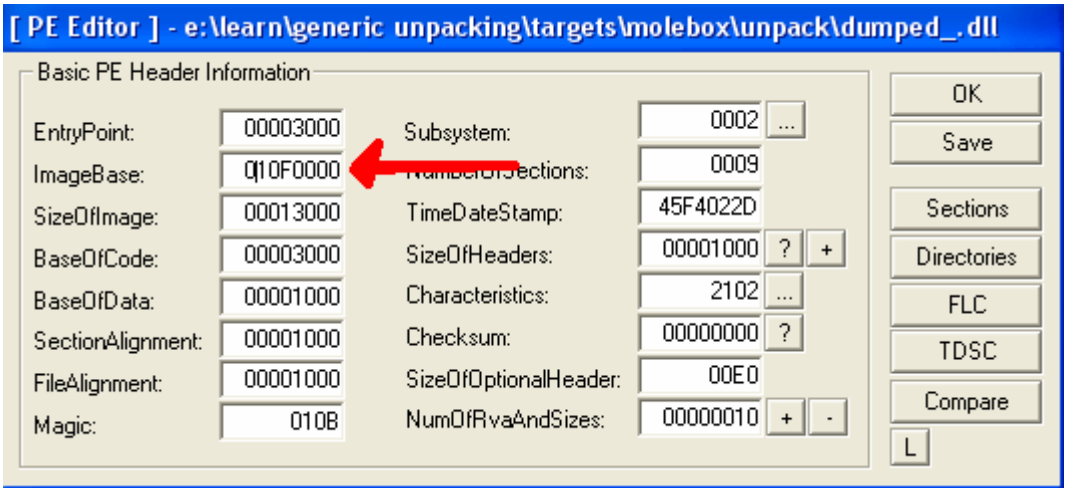

dumped_.exe


dumped_.dll

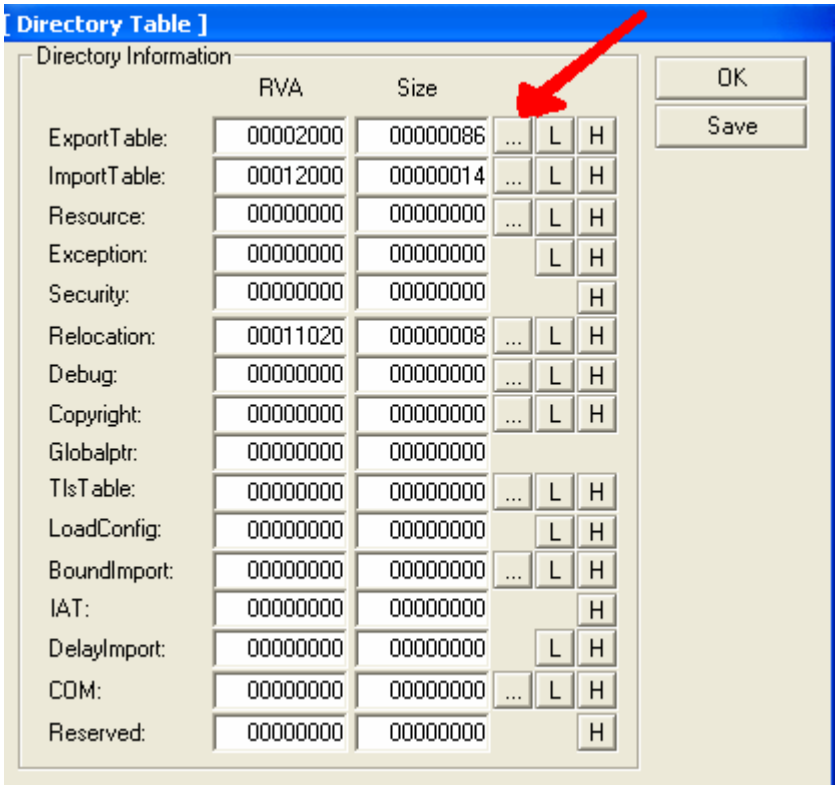
حالا فایل DLL آنپک شده را در LordPE باز کنید:



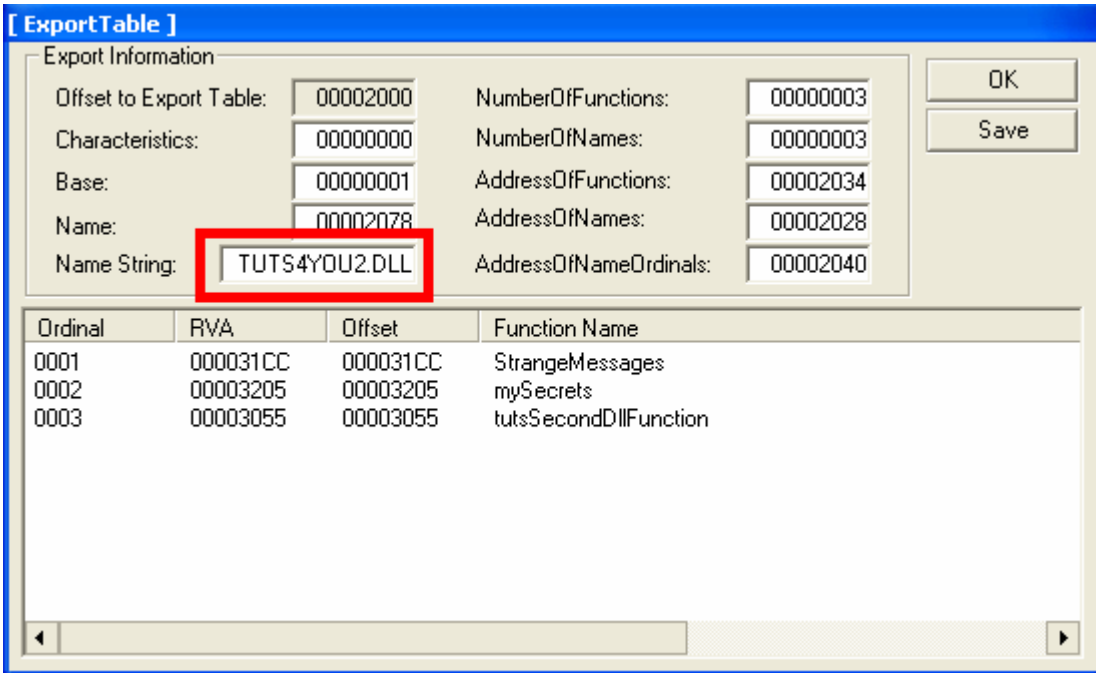
همانطور که می بینید،ImageBase فایل آنپک شده ما 1000000 است،در حالی که وقتی این فایل در Memory بوده،ImageBase آن برابر 10F0000 بوده است.پس این مورد را درست کنید:



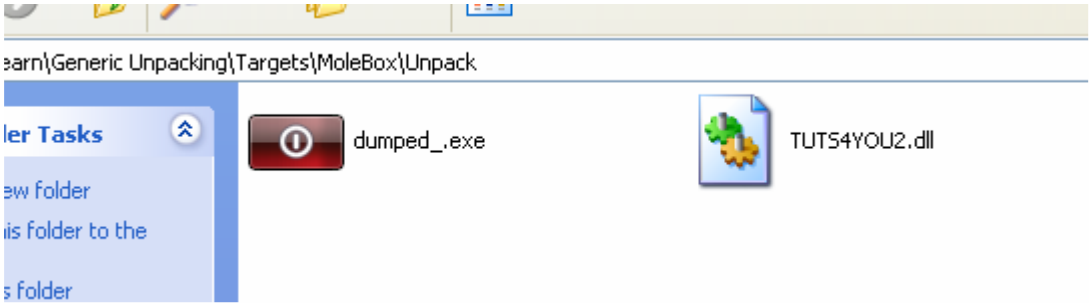
خوب،حالا باید بفهمیم نام واقعی این فایل چیست؟...برای این کار گزینه Directories را بزنید:



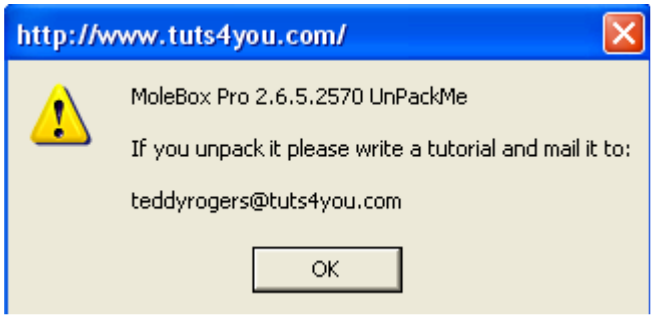
دکمه کنار Export Table را بزنید:



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:

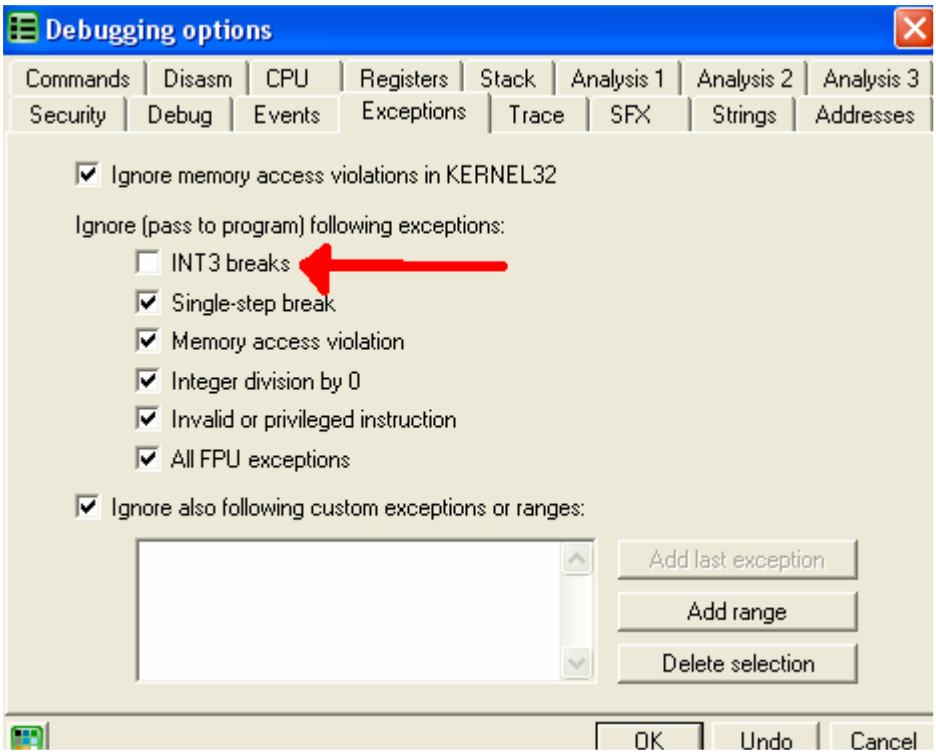


همانطور که می بینید یکی از DLL ها استخراج شده است.دو تای دیگر هم مثل همین است.البته دیگر لازم نیست که Run Trace اجرا کنیم.چون CALL ی که به DLL، OEP ها می رود را پیدا کردیم.حالا کافیست دو فایل DLL دیگر را هم به همین ترتیب استخراج کنیم تا فایل ما بی هیچ مشکلی اجرا شود ☺

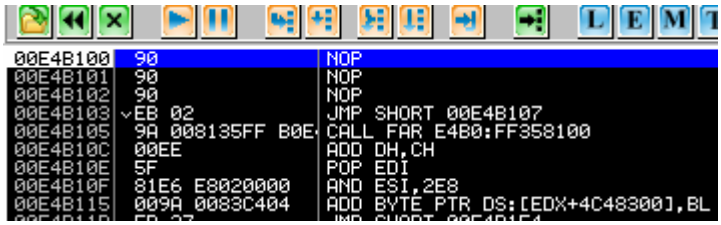


Enigma

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:

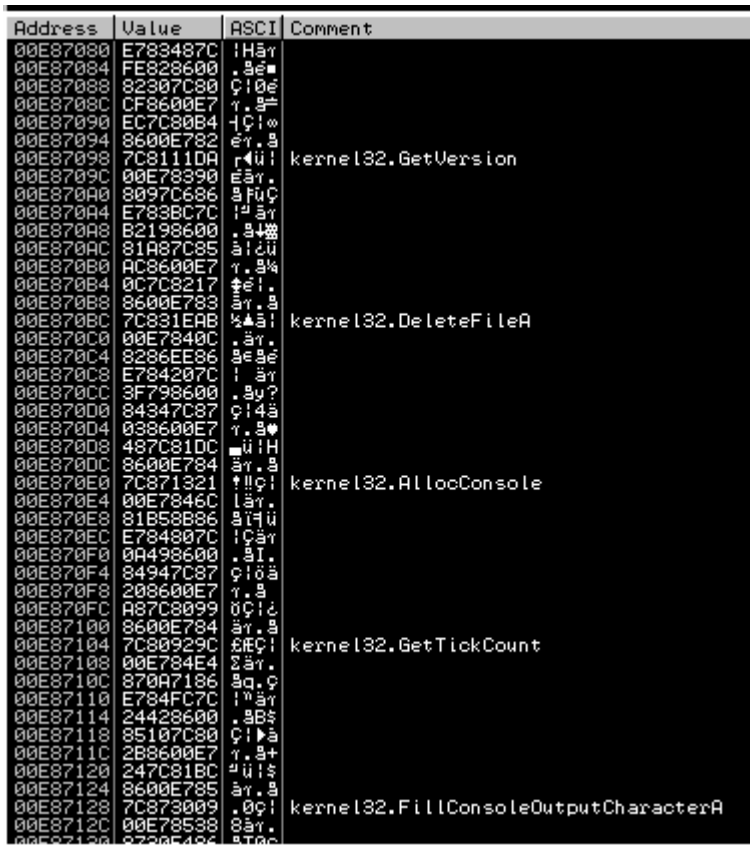


و بعد برنامه را با F9 اجرا کنید.(ممکن است برنامه مقداری کند اجرا شود.چون پکر برای کند کردن دیباگ برنامه،خطای Access violation on write را به وجود می آورد.در ضمن پلاگین HideDBG فراموش نشود،چون این پکر از چند تابع برای شناسایی دیباگر استفاده می کند.)
چند بار کلید F9 را بزنید تا به آخرین خطایی که در برنامه قبل از اجرا شدن اتفاق می افتد،برسید:

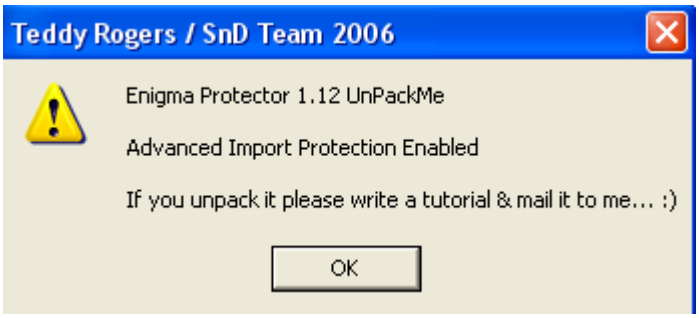


حالا بر روی سکشنی که محتوی Code است(اولین سکشن زیر PE Header) یک Memory breakpoint on access بگذارید و بعد برنامه را با F9 اجرا کنید:

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:

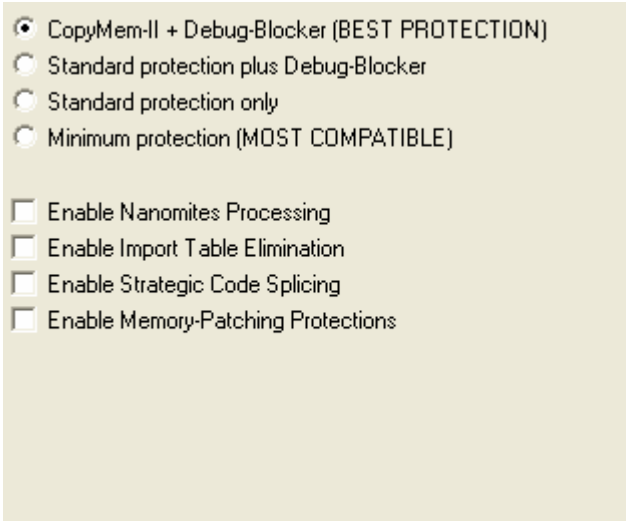


می بینید؟...این پکر هم همان کاری را کرده که PeSpin کرده...پس می توانیم همان کاری را بکنیم که در مورد PeSpin کردیم.یعنی در ImportREC کلیک راست کرده،گزینه Advanced Commands و سپس گزینه Get API Calls را بزنید.کلید OK را بزنید و بعد هم توابع Invalid را از بین ببرید(Cut Thunk) را بزنید و بعد هم فایل دامپ را فیکس کنید(یادتان باشد تیک گزینه Create New IAT را بزنید.)
حالا فایل آنپک شده را اجرا کنید:

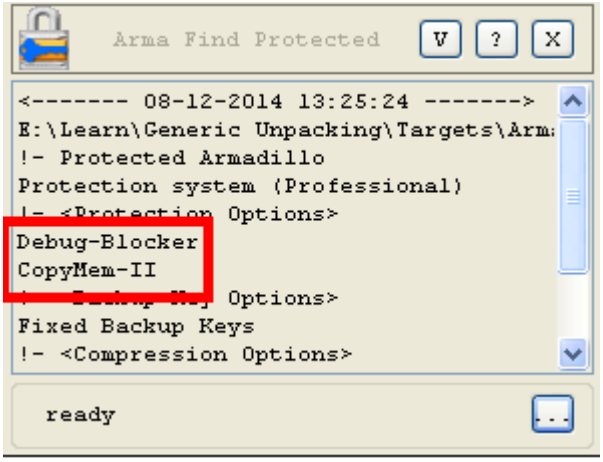


Armadillo

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.



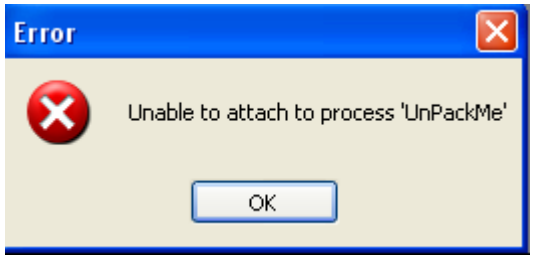
خوب،حالا از کجا بفهمیم که برنامه نویس کدام یکی از این گزینه ها را تیک زده است؟برای این کار می توانیم از نرم افزار Armdillo Find Protected استفاده کنیم.این برنامه را اجرا می کنید و فایل را به برنامه می دهید:



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 کاملا دستی این کار را بکنیم،راه تقریبا 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 ☹)

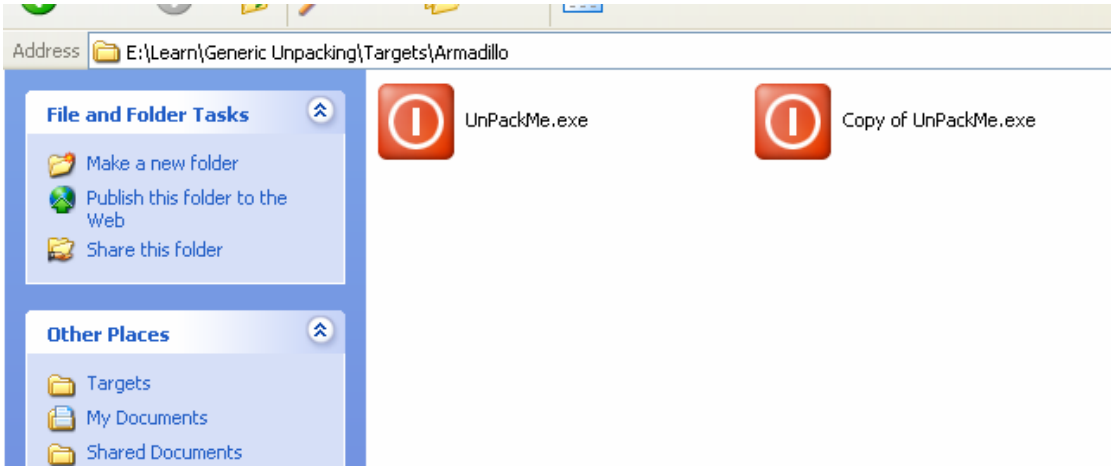


قابلیت CopyMem II در Armadillo چیست؟

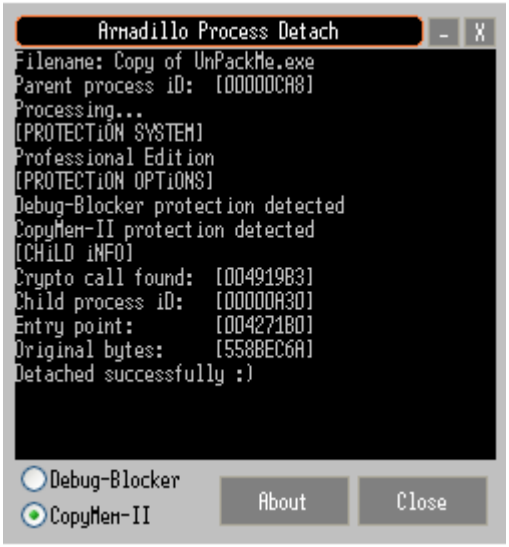
در قابلیت CopyMem II هم دو پروسه ایجاد می شود.اما اینبار پروسه پدر وظیفه Decrypt کردن پروسه فرزند را برعهده دارد.بگذارید مراحل انجام CopyMem II را بنویسم:

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.

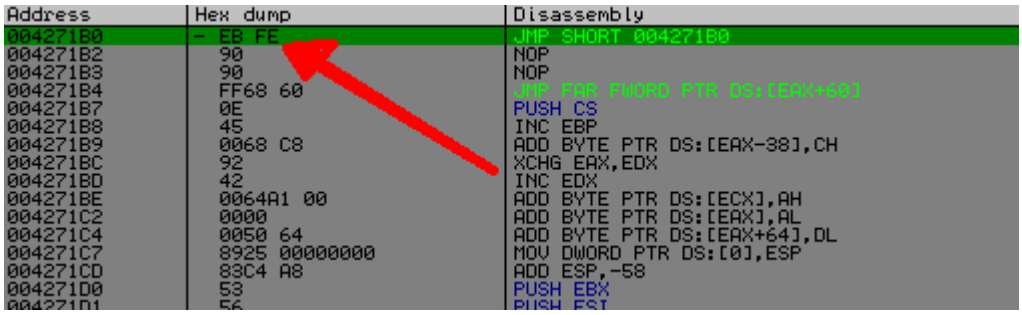
برای از بین این دو قابلیت می توانیم از نرم افزار Arma-Detach که توسط یکی از اعضای تیم RESURRECTi0N نوشته شده استفاده کنیم.ابتدا یکی کپی از فایل مورد نظر بگیرید:



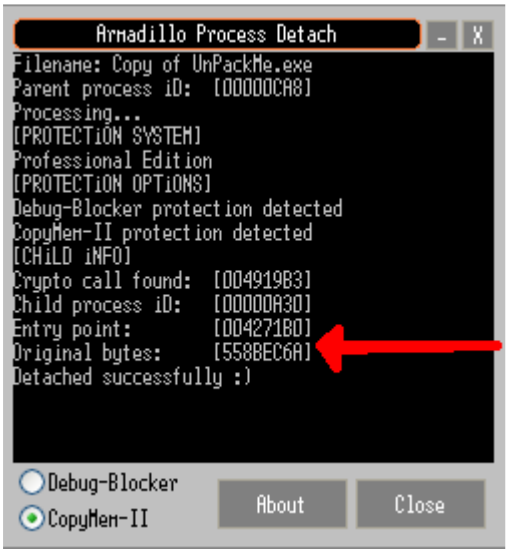
خوب،حالا برنامه Arma-Detach را باز کنید،تیک گزینه CopyMem II را بزنید و فایل کپی شده (Copy of Unpackme.exe) را به داخل برنامه Drag کنید :



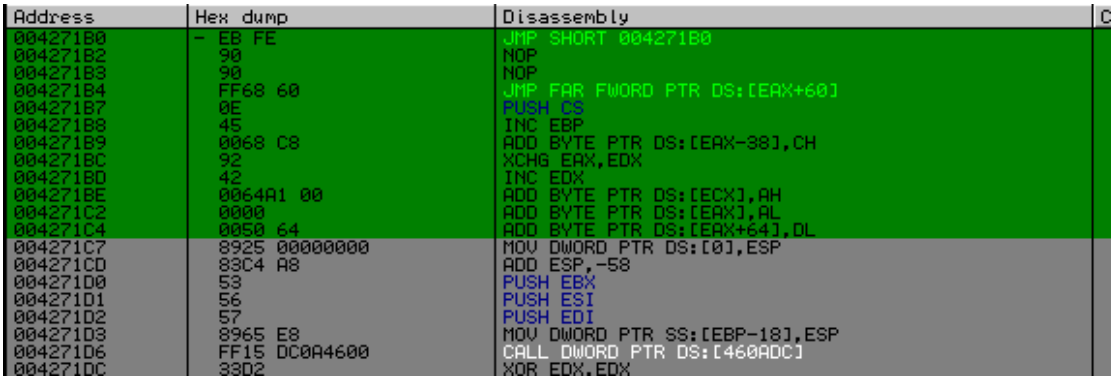
خوب،برنامه توانسته با موفقیت قابلیت CopyMem II را از بین ببرد.همچنین توانسته Entry point پروسه فرزند (که همان OEP ماست) را به دست بیاورد.(004271B0)
همانطور که در تصویر بالا می بینید،Process ID پروسه فرزند من در کامپیوتر من برابر A30 است.خوب،حالا می توانم به پروسه فرزند Attach کنم.(در OllyDBG در منوی 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:

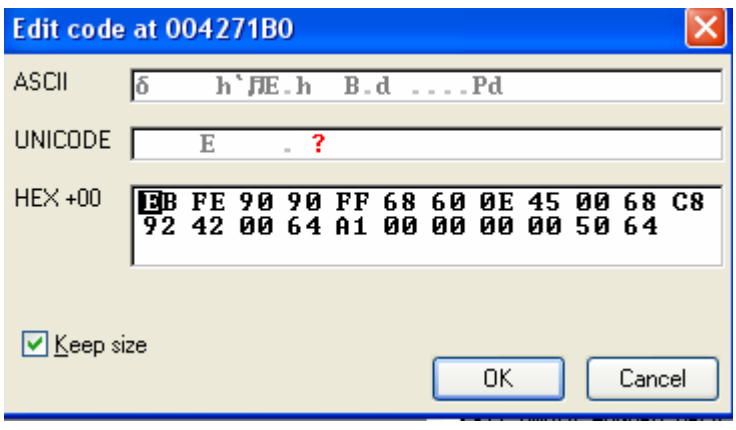


همانطور که می بینید،Arma-Detach برای اینکه از اجرا شدن کامل پروسه جلوگیری کند در Entry point بایت های EBFE را گذاشته.ما باید این بایت ها را به سر جایش باز گردانیم،یک بار دیگر نگاهی به Arma-Detach بیندازید:

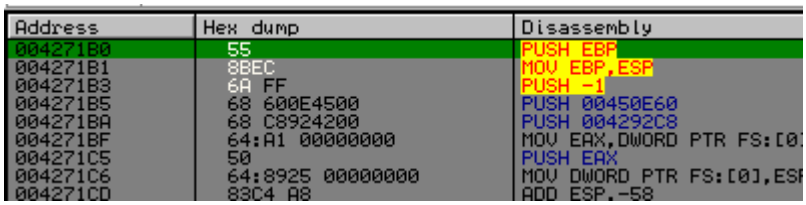


همانطور که می بینید Arma-Detach می گوید،در ابتدا بایت های 558BEC6A در Entry point قرار داشته.پس ما باید این بایت ها را به سر جایش برگردانیم،چند خط ابتدای پروسه فرزند را انتخاب کرده و کلیدهای CTRL + E را بزنید تا این بایت ها را Edit کنیم:

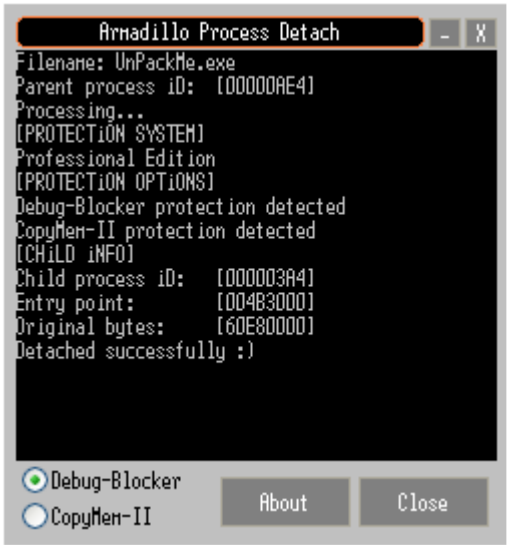




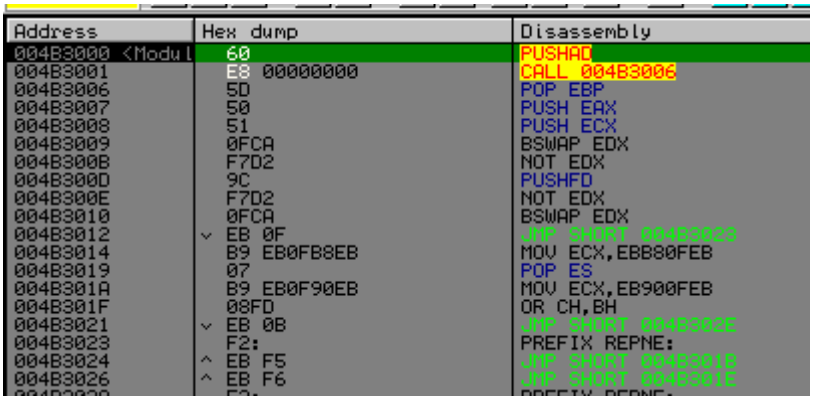
حالا بایت های اصلی(558BEC6A) را بنویسید و کلید OK را بزنید.



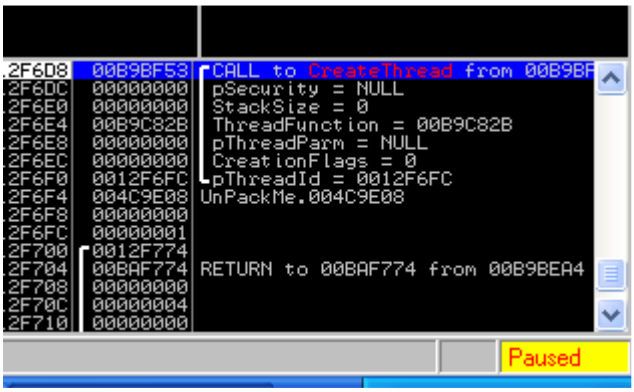
خوب،حالا باید قابلیت Debug-Blocker را از بین ببریم.برای این کار یک Arma-Detach دیگر باز کنید و این بار تیک گزینه Debug-Blocker را بزنید و فایل اصلی (Unpackme.exe) را به آن بدهید:



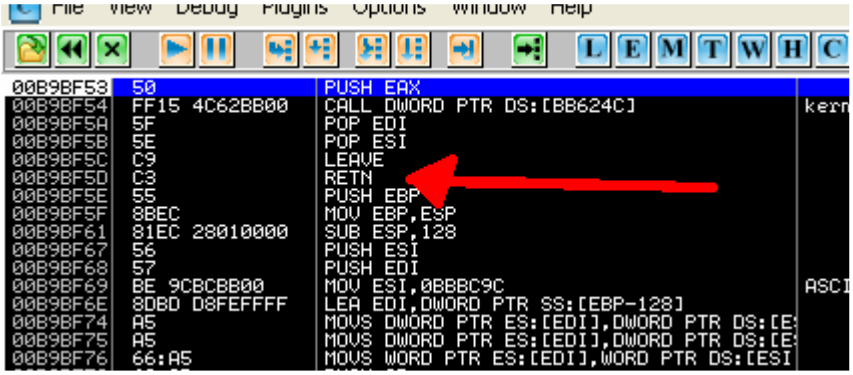
خوب،حالا یک پروسه دیگر برای از بین بردن قابلیت Debug-Blocker ساخته شده،یک OllyDBG دیگر باز کنید و به این پروسه جدید(همانطور که می بینید در کامپیوتر من این پروسه 3A4 است) Attach کنید.حالا هم همانند قبل به Entry point این پروسه بروید: (004B3000) و بعد بایت های اصلی را جایگزین EBFE کنید(60E8) :



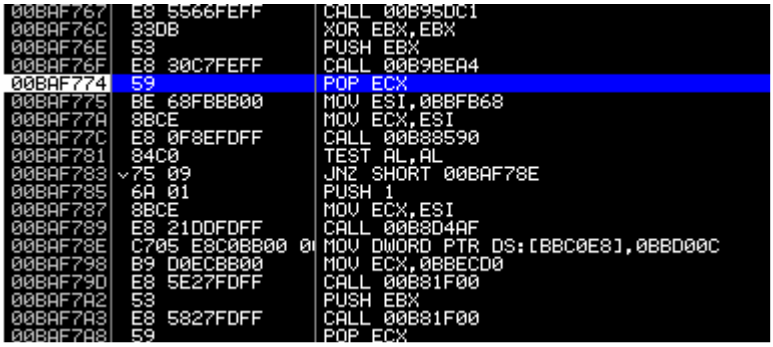
خوب،حالا باید در داخل این پروسه جدید OEP را به دست بیاوریم...برای این کار می توانیم از روشی مخصوص Armadillo استفاده کنیم.یعنی استفاده از تابع CreateThread ☺
در پلاگین Command Bar تایپ کنید : HE CreateThread (تا بر روی تابع CreteThread یک Hardware BP on excution بگذاریم)
و بعد برنامه را با F9 اجرا کنید.(مراقب 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.)



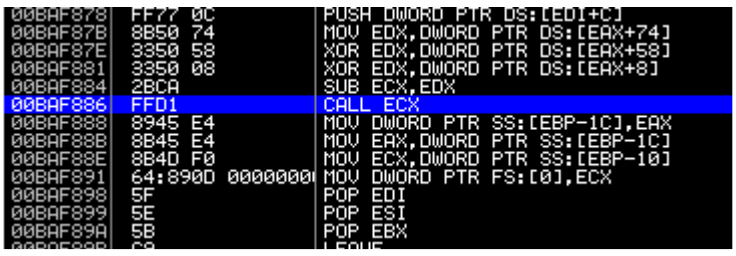
خوب،در تابع CreateThread متوقف شدیم،کلیدهای ALT + F9 را بزنید:



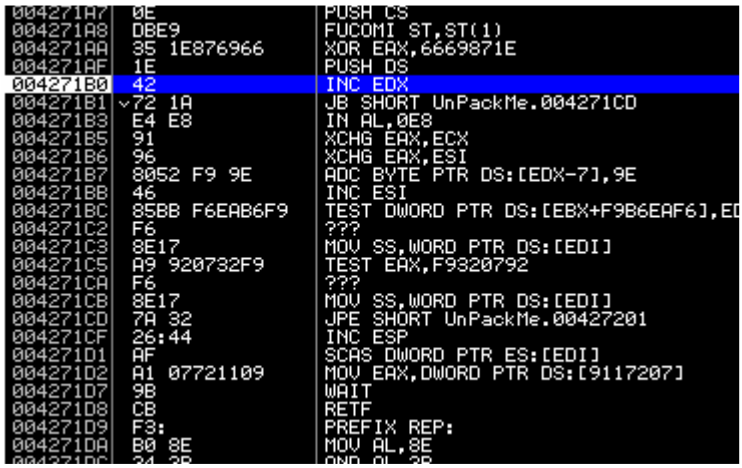
چند بار کلید F8 را بزنید تا از دستور RET رد بشوید.



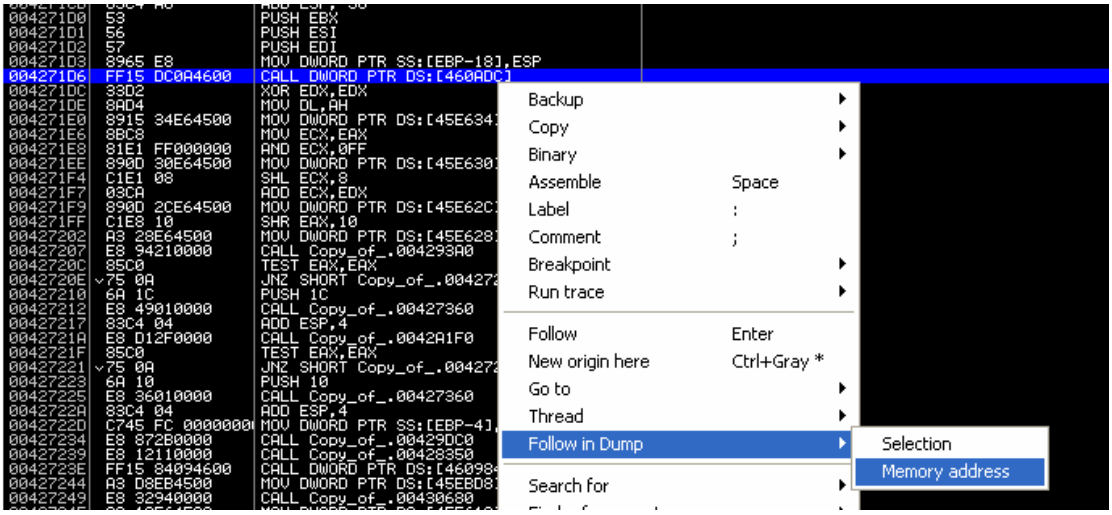
معمولا دومین دستور CALL ECX زیر این خط به OEP می رود ☺



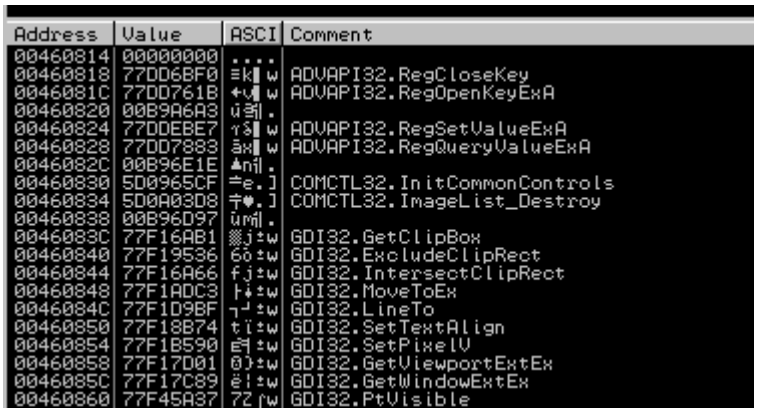
پس بر روی این خط یک BP بگذارید و برنامه را با F9 اجرا کنید و یک بار F8 را بزنید تا به OEP برسید:



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 Adress را می زنیم.)



حالا به پنجره دامپ نگاه کنید:

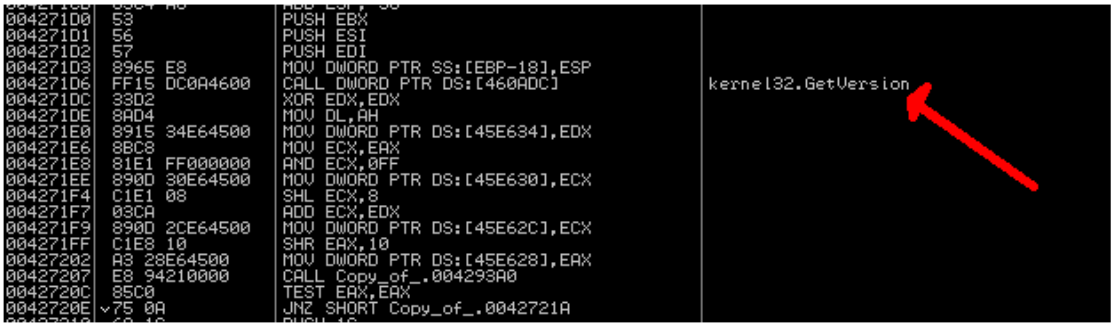


حالا همانند مثال های قبل اطلاعات IAT را به دست می آوریم:

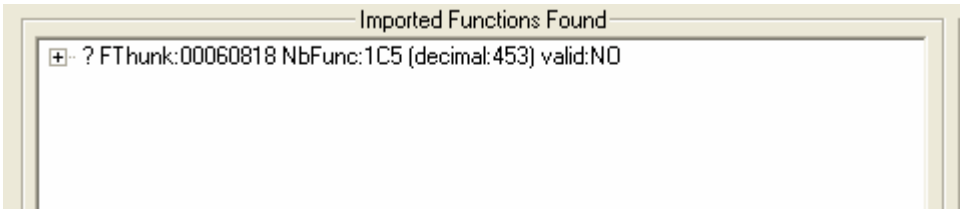
Start of IAT=460814 (RVA: 60814)
End of IAT=460F2C (RVA: 60F2C)
Size of IAT=718

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.

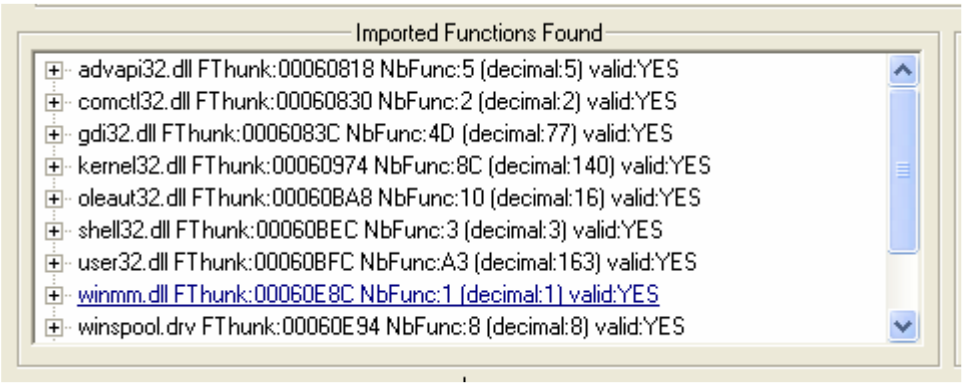
پس ما از خط 460814 تا خط 460F2C را از پروسه دوم کپی کرده و در پروسه اول(Paste (Copy of unpackme.exe می کنم.(ابتدا در پنجره دامپ این خطوط را انتخاب می کنید،کلیک راست کرده و گزینه Binary و سپس گزینهBinary Copy را می زنید،حالا در پروسه اول(Copy of unpackme.exe)،در پنجره دامپ این خطوط(460814 تا 460F2C) را انتخاب می کنید و کلیک راست کرده گزینه Binary و سپس گزینه Binary Paste را می زنید.)



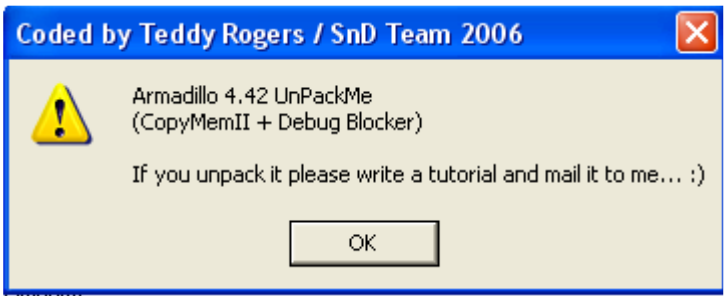
همانطور که می بینید حالا IAT درسته شده است.پس از پروسه اول(Copy of unpackme.exe) دامپ بگیرید و این پروسه را در ImportREC باز کنید،OEP را بدهید و سایر مراحل را انجام دهید:



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 ☺

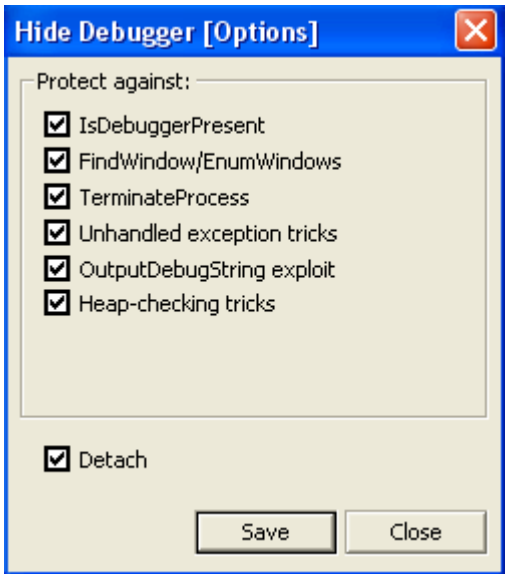


توضیح:اگر برنامه نویس علاوه بر قابلیت های CopyMemII و Debug Blocker تیک گزینه Import Table elimination را می زد.برای درست کردن IAT باید از برنامه ArmInline استفاده می کردیم ☹

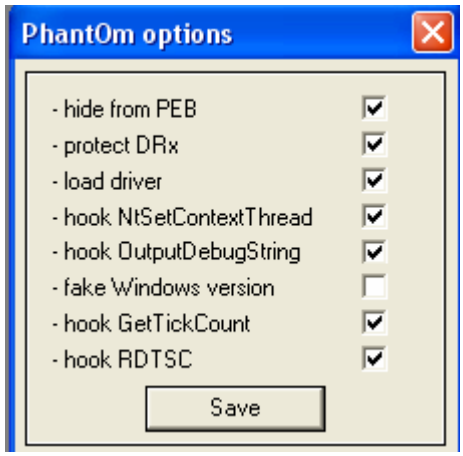
TheMida

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:

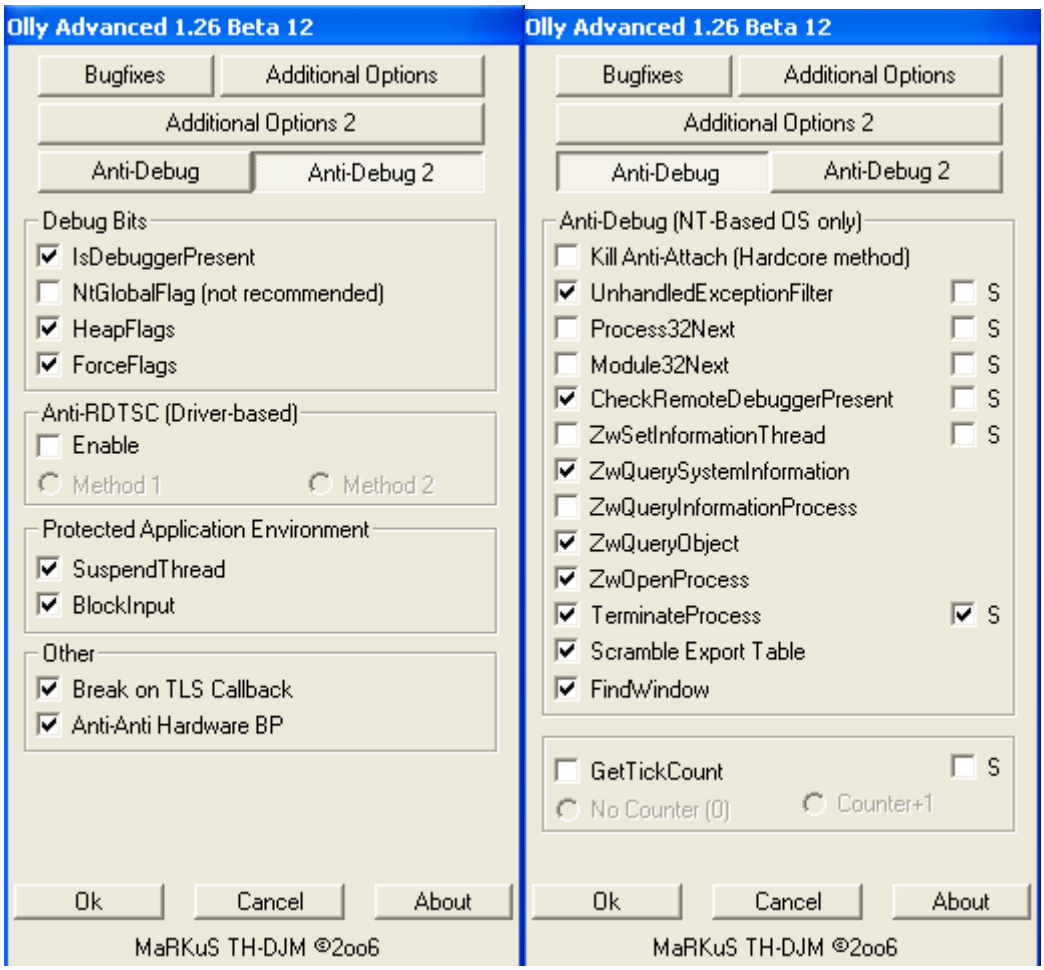
خوب،این پروتکتور از آنتی دیباگ های بسیار زیادی استفاده می کند.پس باید در مقابل آنها مقاوم باشیم.آخرین ورژن پلاگین HideDebugger را دانلود کرده و تمامی گزینه هایش را تیک بزنید:



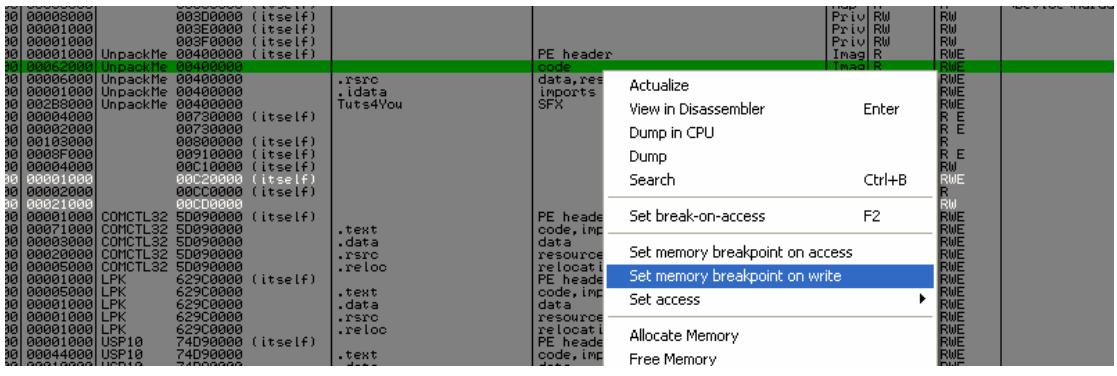
پلاگین PhantOm را هم دانلود کنید:



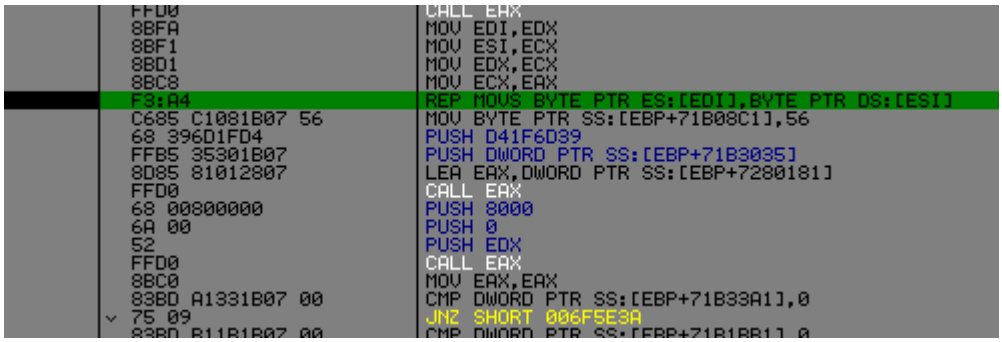
در پلاگین PhantOm به هیچ وجه گزینه Fake Windows Version را تیک نزنید.پلاگین OllyAdvanced را هم دانلود کنید و همانند تصویر زیر تنظیم کنید:



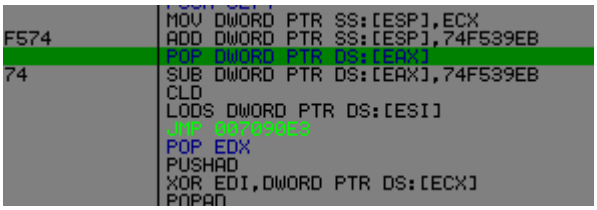
خوب،حالا دیگر فایل باید در OllyDBG اجرا شود.(برای تست کلید F9 را بزنید.)
این فایلی که ما داریم IAT را Redirect که چه عرض کنم...کلا نابود می کند...در ضمن Call هایی که به توابع API می روند هم به طور کامل Redirect شده اند...لذا ما اگر بخواهیم بعد از یافتن OEP، IAT و CALL ها را درست کنیم.کار بسیار سخت و 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:



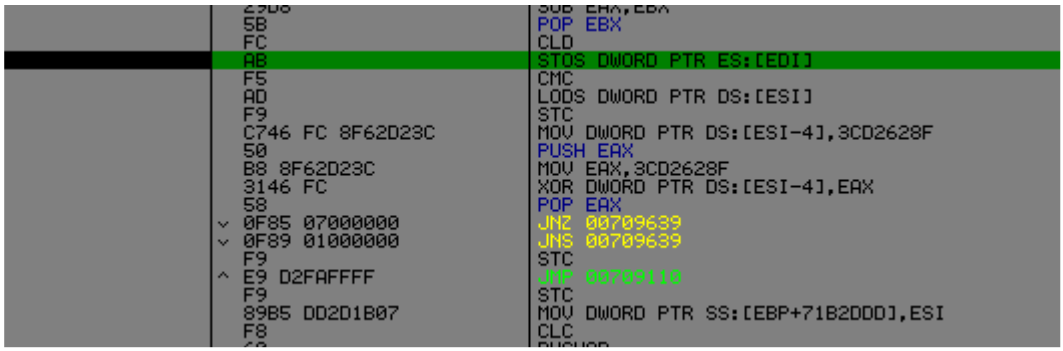
برنامه را با F9 اجرا کنید:



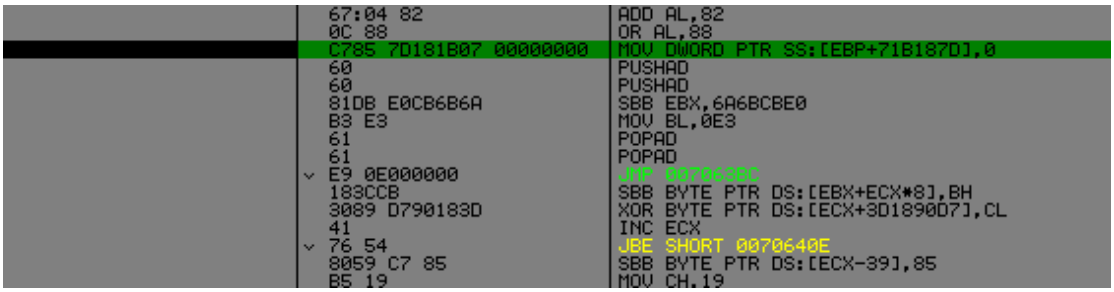
اینجا چیز خاصی نیست،کلید F8 و سپس F9 را بزنید تا از اینجا رد شوید:



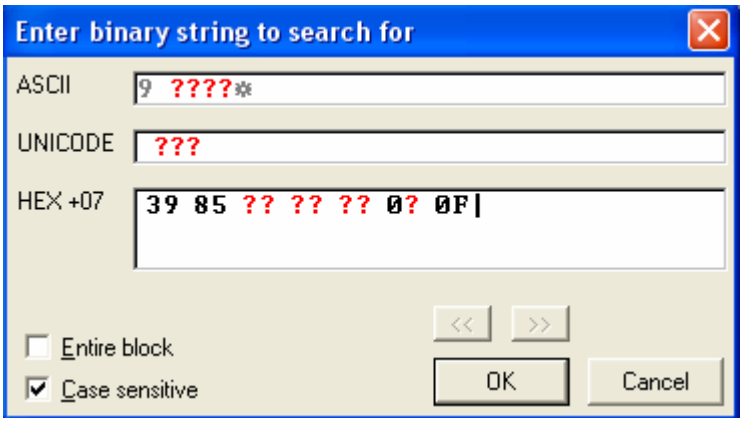
حالا چهار بار F9 را بزنید:



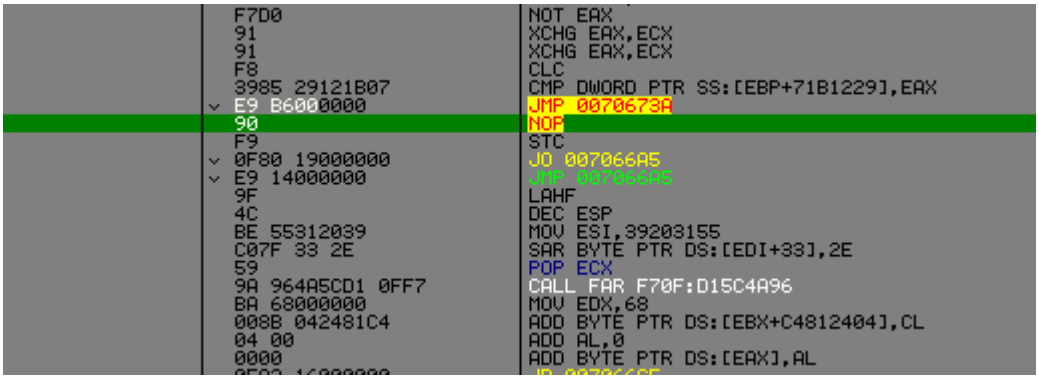
حالا با F8 به جلو بروید تا به اینجا برسید:



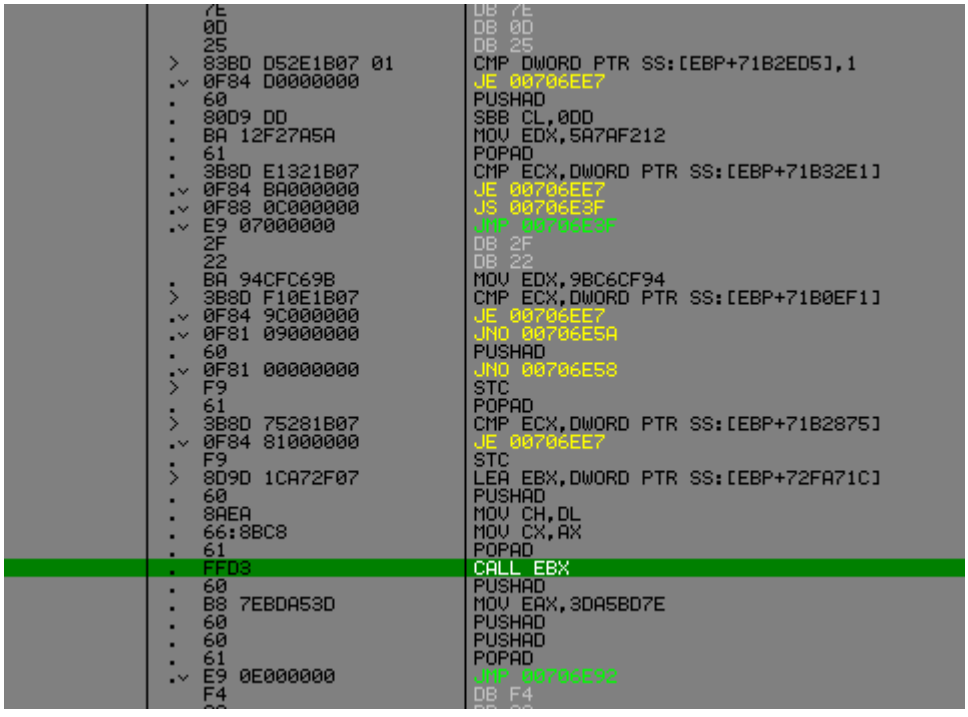
حالا کلیدهای CTRL + B را بزنید و تایپ کنید: 0F ?0 ?? ?? 85 39 و کلید OK را بزنید:



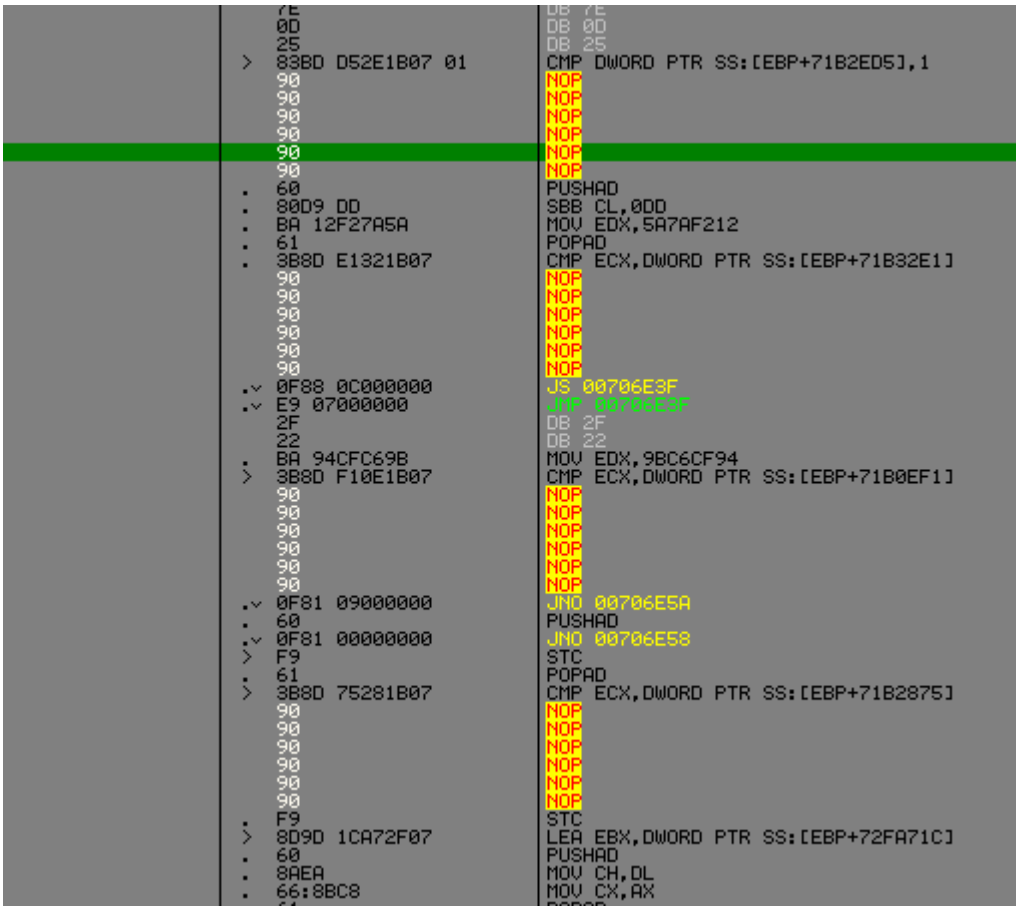
پرشی که در زیر این خط قرار دارد،(JE 0070673A)،باید JMP شود:



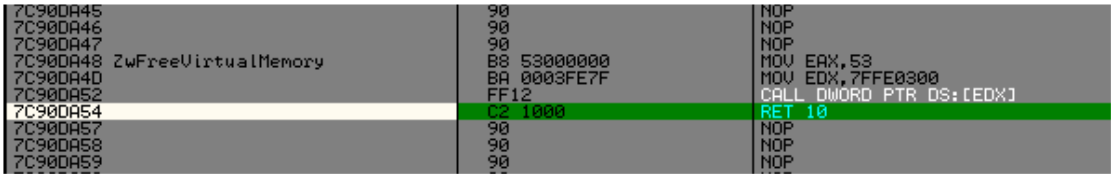
حالا دوباره کلیدهای CTRL + B را بزنید و تایپ کنید FF D3



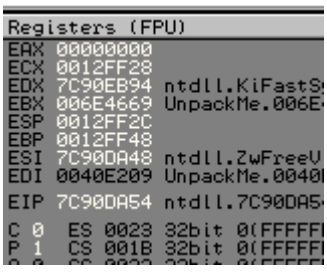
بالای این خط چهار پرش وجود دارد که هر چهار تای آن به یک خط می روند.شما باید هر چهار تا را Nop کنید.(بهتر است اول کدها را آنالیز کنید تا بهتر ببینید:



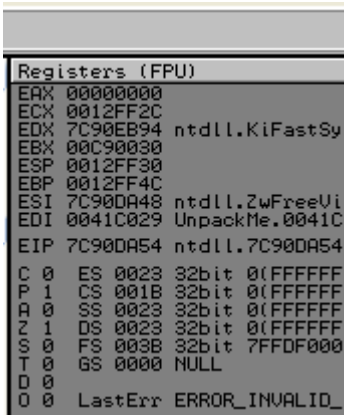
همانطور که دیدید ما پنج پرش را دستکاری کردیم،به این ترتیب IAT ما Redirect نمی شود.اما همچنان مشکلاتی با CALL هایی که به توابع API می روند،خواهیم داشت.
برای یافتن OEP می توانیم از Memory Breakpoint استفاده کنیم.ابتدا باید بفهمیم که در کجا کدها به طور کامل Decrypt شده اند؟برای این کار روی آخر تابع ZwFreeVirtualMamory یک BP بگذارید:



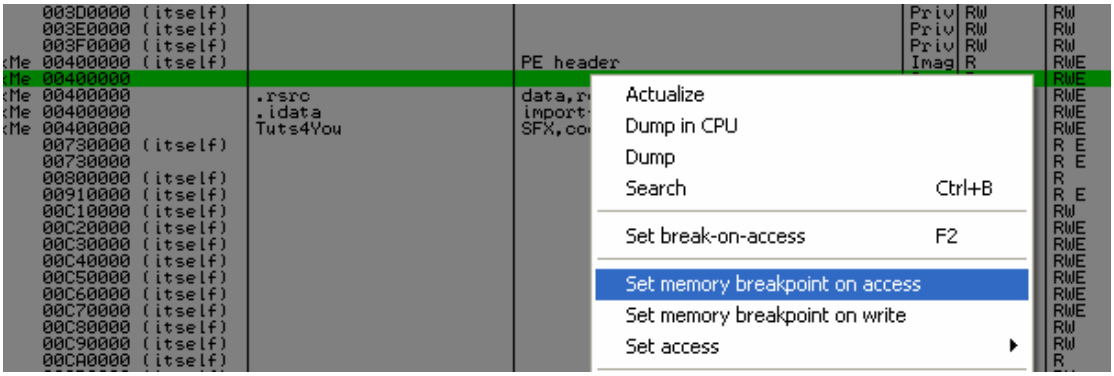
ممکن است سوال کنید چرا آخر این تابع BP گذاشتم؟...برای اینکه برخی پروتکتورها در ابتدای توابع API به دنبال بایت CC می گردند.چون OllyDBG از این بایت برای گذاشتن BP استفاده می کند.پروتکتورها متوجه گذاشتن Breakpoint شده و در نتیجه دیباگر را شناسایی می کنند ☹...لذا اگر با یک پروتکتور خوب طرف هستید،سعی کنید یا Hardware BP بگذارید،یا اگر بر روی تابع BP،API 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،معمولا این کار را نمی کنند.
برنامه را با F9 اجرا کنید:



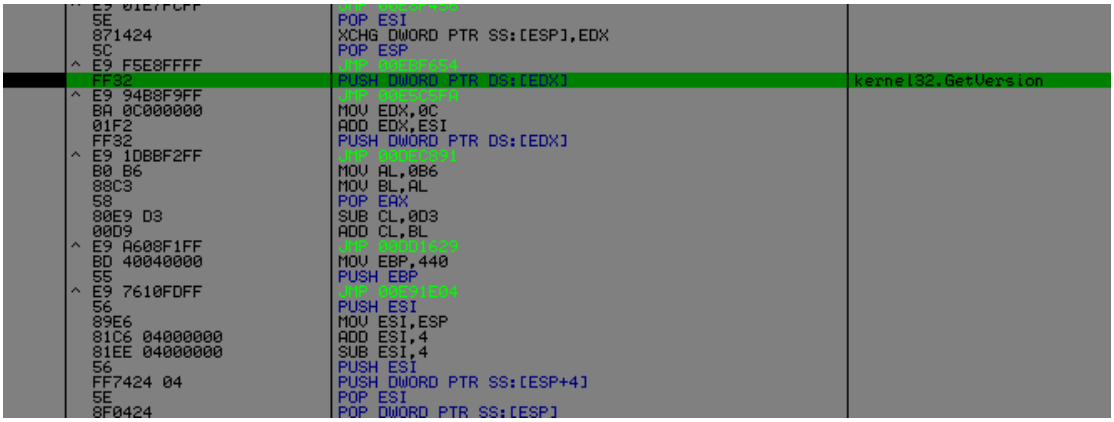
می بینید؟...مقدار رجیسترها تغییر کرده اند(رنگشان سفید شده)...تا زمانی کلید F9 را بزنید که مقدار رجیسترها تغییر نکند(یعنی همه شان سیاه باشند،در ضمن آن Memory BP را هم پاک کنید):



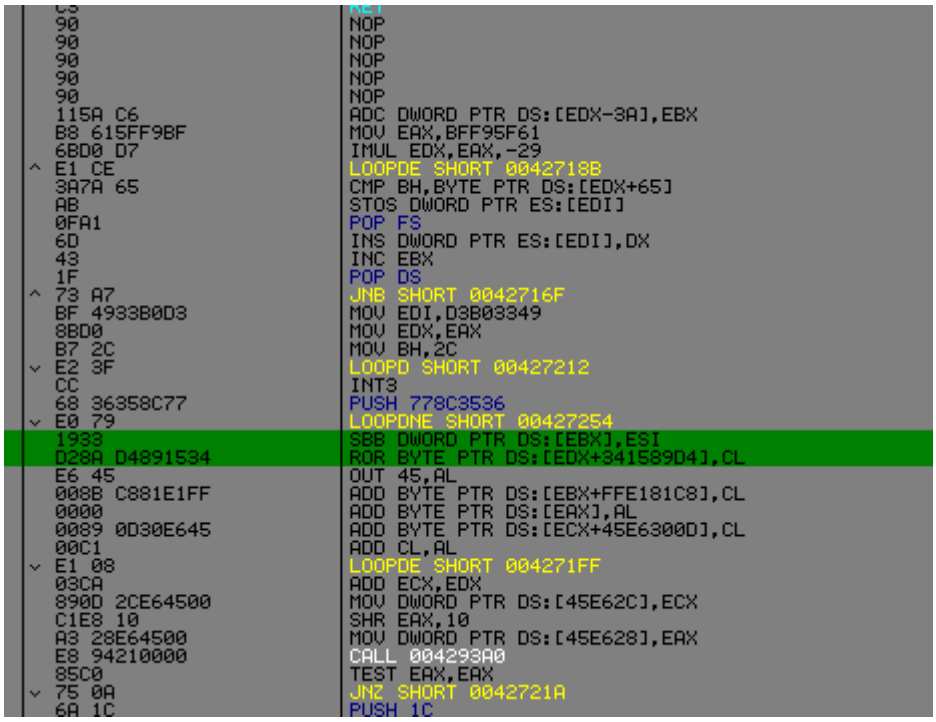
اینجا دیگر جاییست که کدهای ما Decrypt شده و می توانیم بر روی سکشنی که محتوی code است،Memory BP بگذاریم:



کلید F9 را بزنید(بریکپوینت که بر روی تابع ZwFreeVirtualMemory گذاشته بودید را پاک کنید):



این پکر چند بایت از بایت های ابتدای فایل اصلی را دزدیده است.(Stolen Bytes)...اینجا که ما هستیم،جایی است که آن بایت های دزدیده شده به آنجا منتقل شده اند. (البته پیدا کردن آنها از بین این همه دستور،زیاد کار راحتی نیست)
یکبار دیگر F9 بزنید:

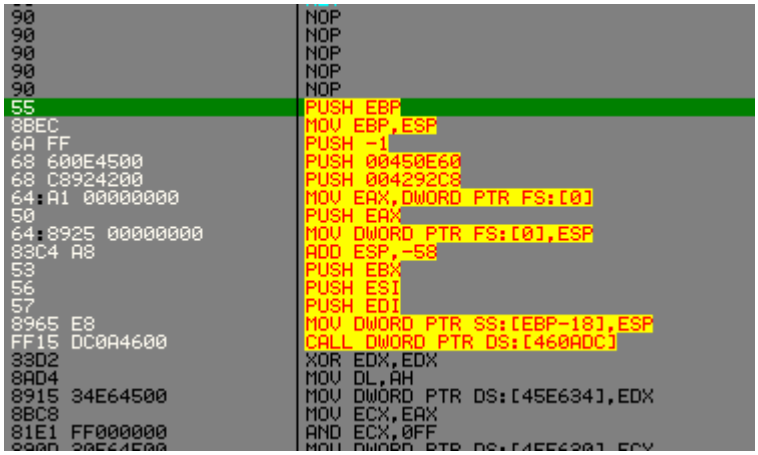


ما الان در نزدیکی های OEP و در سکشن اصلی هستیم.اما چند بایت در نزدیکی OEP دزدیده شده است.برای یافتن بایت های دزدیده شده معمولا دو راه وجود دارد:

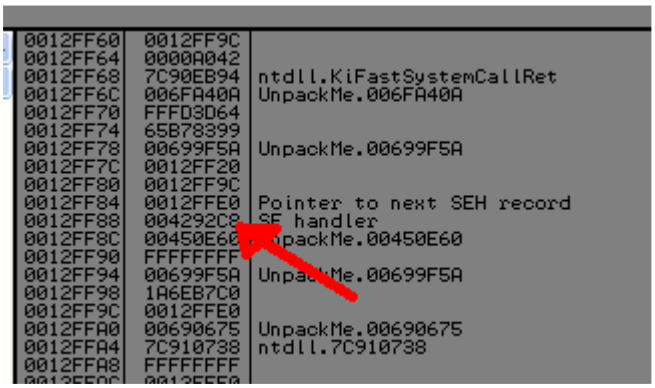
۱.در داخل پکر بگردیم و بایت ها را پیدا کنیم(کاری که در No Name Packer 2 کردیم)
۲.از ساختار زبان نویسی استفاده کنیم.

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.)

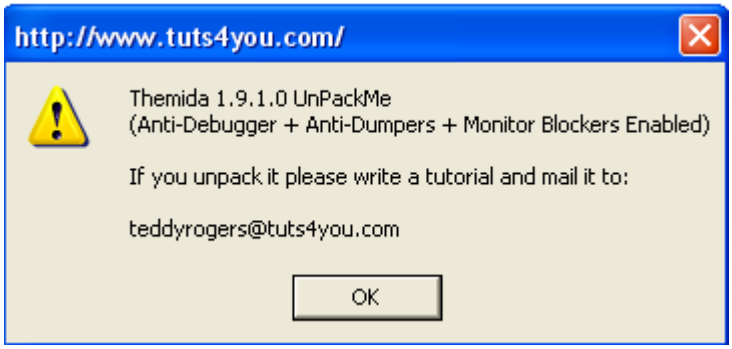
خوب،برنامه در زبان Visual C++ نوشته شده است.پس یک برنامه پیدا می کنم که در این زبان نوشته شده باشد و کدهای ابتدایی آن را به کدهای این فایل اضافه می کنم.(یعنی آن فایل(فرقی نمی کند فایل چی باشد،فقط در C++ ورژن ۵ نوشته شده باشد.) را در OllyDBG باز کنیم و کدهای ابتدای آن را انتخاب کرده،کلیک راست می کنم،گزینه Binary و سپس Binary Copy را می زنم و بعد کدهایی که کپی کردم را در این فایل Binary Paste می کنم☺)



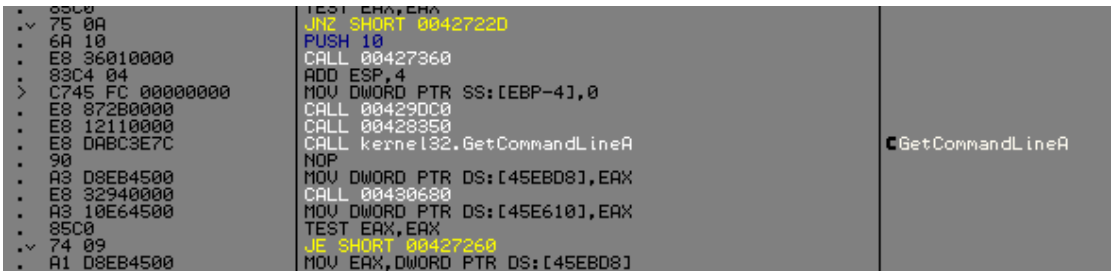
البته بعد از Binary Copy/paste کردن باید مقداری کدها را با توجه به فایل خود دستکاری کنید.مثلا خط پنجم برای نصب SEH هست،ما باید در پنجره Stack ببینم در کجا SEH نصب شده و آدرس آنجا را در این خط بدهم:



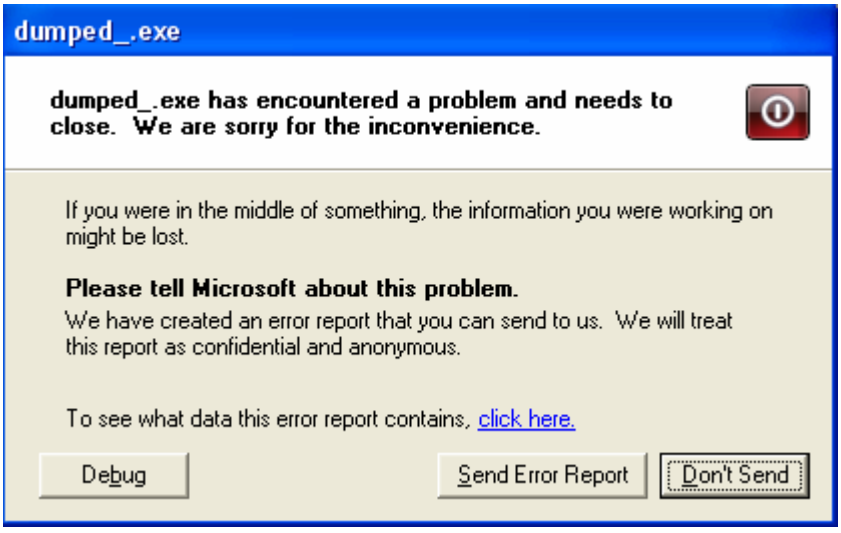
خوب،حالا بر روی OEP دزدیده شده برنامه(4271B0) کلیک راست کرده و گزینه New origin here را بزنید و بعد از فایل دامپ بگیرید.
حالا فایل را با ImportREC فیکس کنید و اجرا کنید:



بله،فایل آنپک شده در سیستم شما اجرا می شود ☺ اما این فایل را در OllyDBG باز کنید و نگاهی دقیق به آن بندازید:



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 آنپک شده را در سیستم دیگر اجرا کنید:



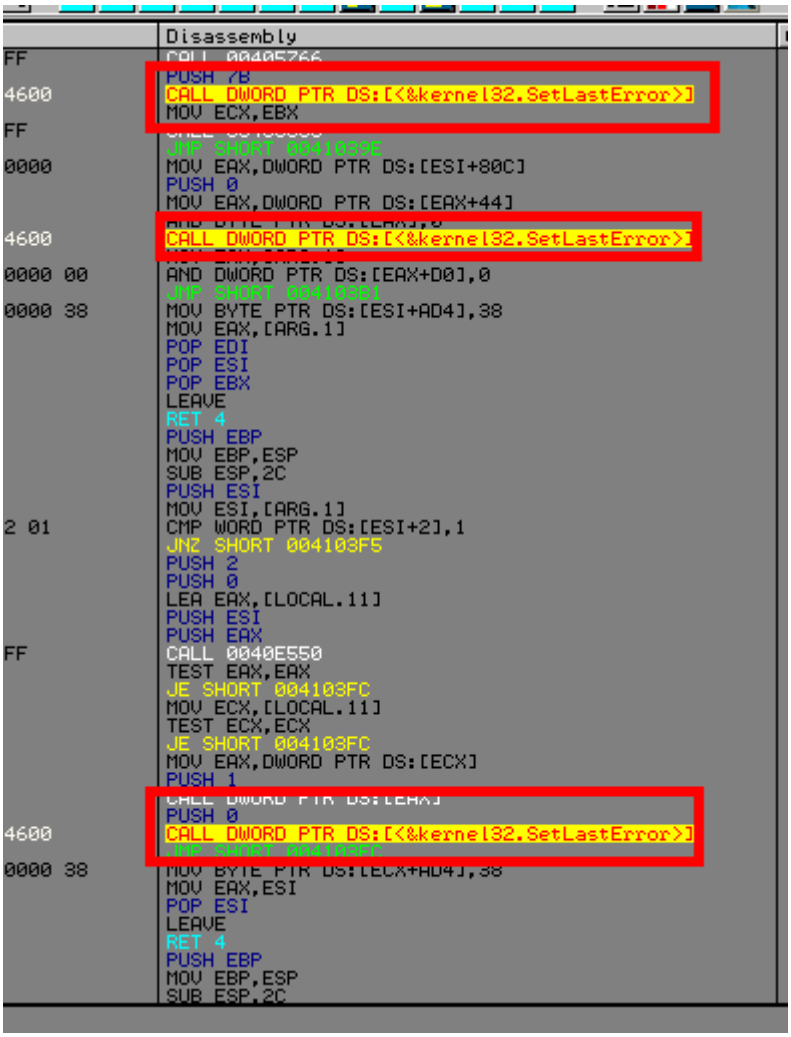
حالا چه کار کنیم که این برنامه در هر سیستمی اجرا شود؟...من یکسری کد نوشتم که شما این کدها را در آدرس 460F2D که فضای خالی در آنجا موجود است،کپی می کنید...این کدها به طور خودکار تمامی CALL ها را پیدا می کنند و آنها را درست می کنند ☺(این کدها در داخل فایل Themida_codes.txt موجود است.):

```
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:

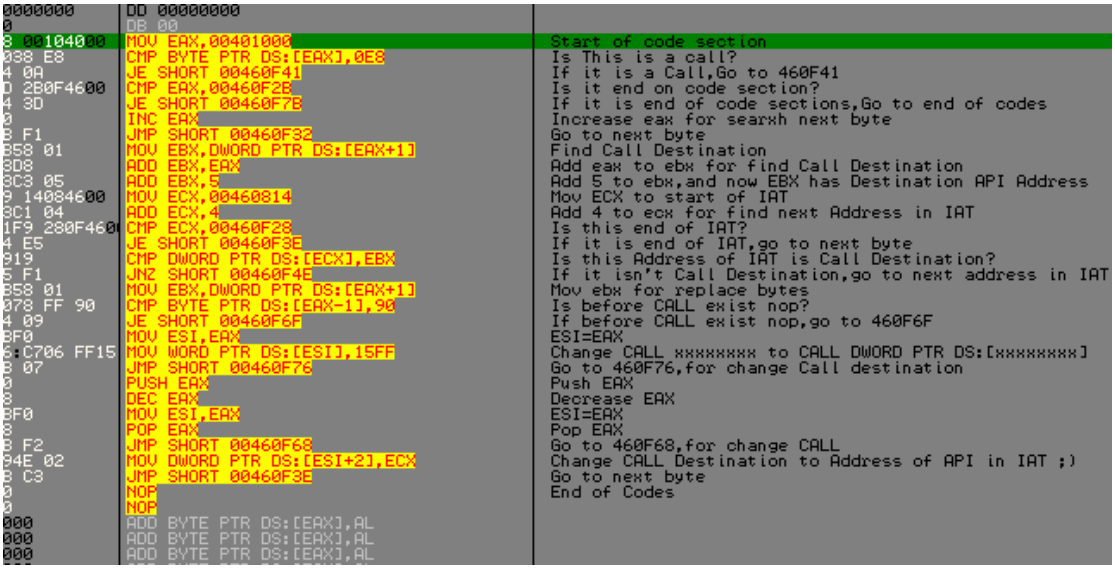
```
460F64  74 09     JE SHORT 00460F6F
460F66  8BF0      MOV ESI,EAX
460F68  66:C706 FF15  MOV WORD PTR DS:[ESI],15FF
460F6D  EB 07     JMP SHORT 00460F76
460F6F  50        PUSH EAX
460F70  48        DEC EAX
460F71  8BF0      MOV ESI,EAX
460F73  58        POP EAX
460F74  EB F2     JMP SHORT 00460F68
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
```

خوب...این کدها هم اجرا شد،حالا دوباره نگاهی به CALL های برنامه بیندازید:



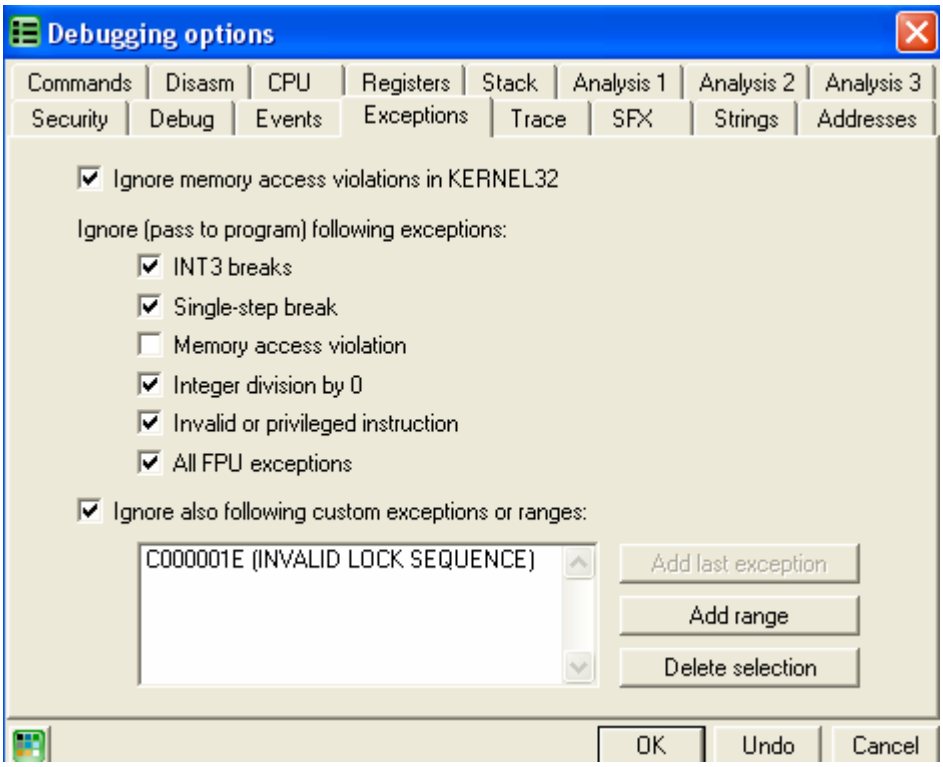
می بینید؟...تمامی CALL های برنامه درست شده و این بار از IAT استفاده می کنند ☺،حالا می توانید فایل آنپک شده را در هر سیستمی تست کنید ☺

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 فایلی 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:

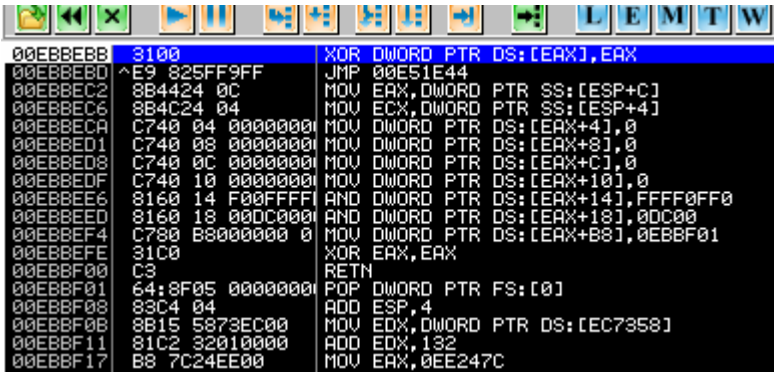


Virtual Machine

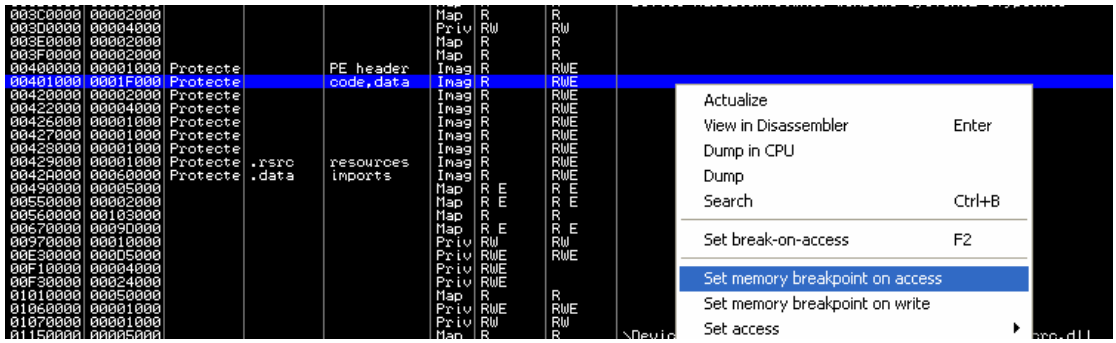
Virtual Machine یکی از پروتکشن هایی است که امروزه اکثر پروتکتورها سعی می کنند این قابلیت را به پروتکتور خود اضافه کنند.زیر این روش کار آنپک شدن فایل را سخت می کند.و تنها کسانی قادر به آنپک کردن آن می باشند که تجربه کافی در زمینه آنپکینگ داشته باشند ☹
قاعده کلی Virtual Machine در این است که تعدادی از Function ها و کدهای برنامه توسط پروتکتور شبیه سازی (Emulate) می شود،در Virtual Machine قسمتی از کدهای برنامه(معمولا یک Function) از داخل سورس کد فایل استخراج می شود و به زبانی رمزگذاری می شود که فقط خود پروتکتور از آن سر در می آورد.
این قسمت از کدهای برنامه رمزگذاری شده می مانند،تا زمانی که برنامه نیاز به این کدها پیدا کند،در این موقع پروتکتور رمز کدها را باز می کند تا برنامه بتواند از کدهای آن قسمت استفاده کند.
برای بررسی Virtual Machine من یک فایل را با Enigma Protector پک کردم(تمامی پروتکشن های 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 می کنند.ولی ما اینجا از Olly استفاده کردیم)
برای یافتن OEP می توانیم از Memory Breakpoint استفاده کنیم.ابتدا باید آخرین خطایی که در برنامه اتفاق می افتد 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.) :



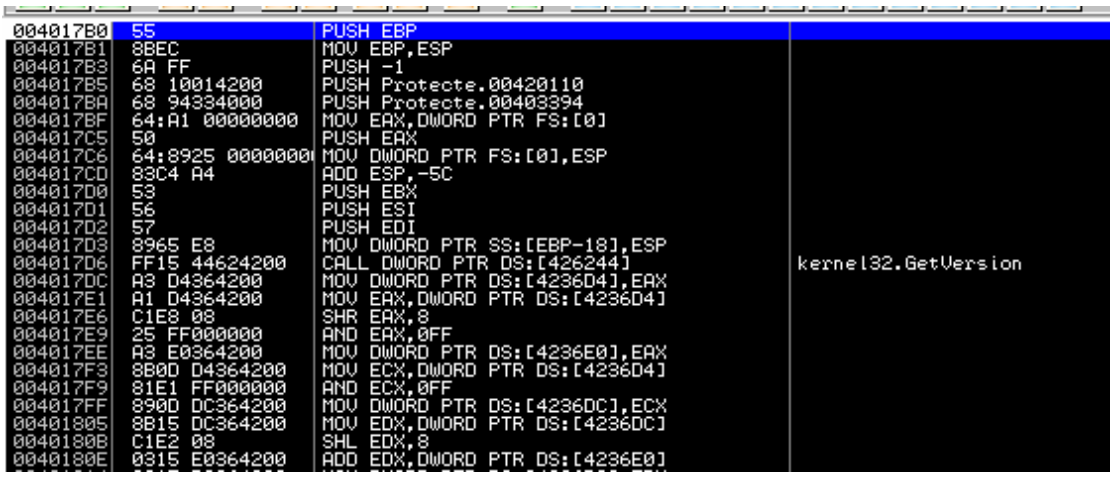
حالا برنامه را با F9 اجرا کنید و آنقدر با Shift + F9 برنامه را ادامه دهید،تا به آخرین خطای برنامه برسید:



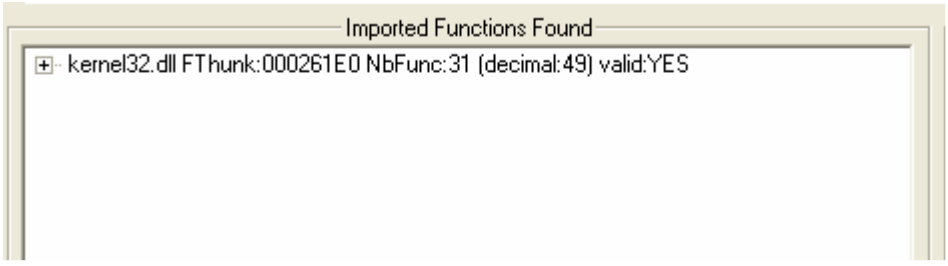
خوب،حالا کلیدهای ALT + M را بزنید.و بر روی سکشنی محتوی code است،Memory BP بگذارید:



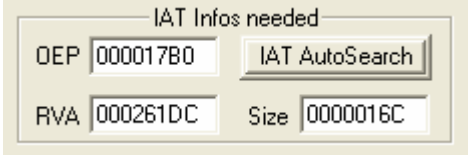
حالا کلید Shift + F9 را بزنید تا به OEP برسید:



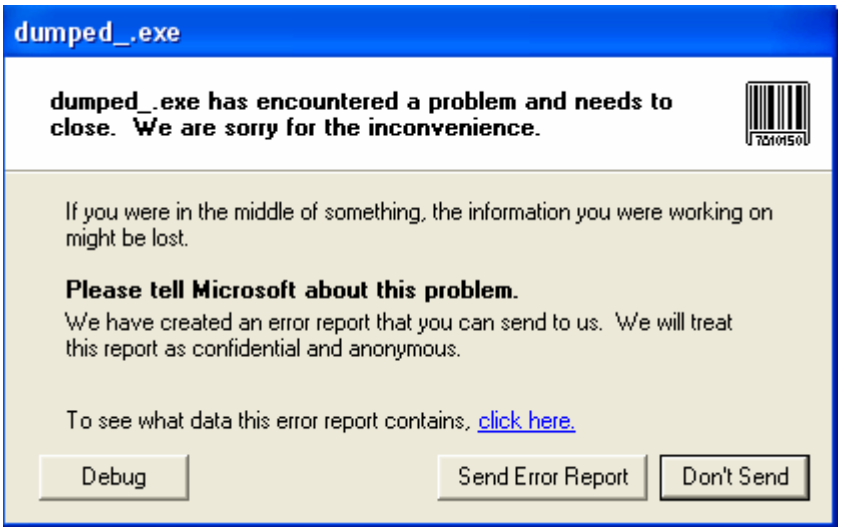
حالا از فایل دامپ بگیرید و ImportREC را باز کنید:



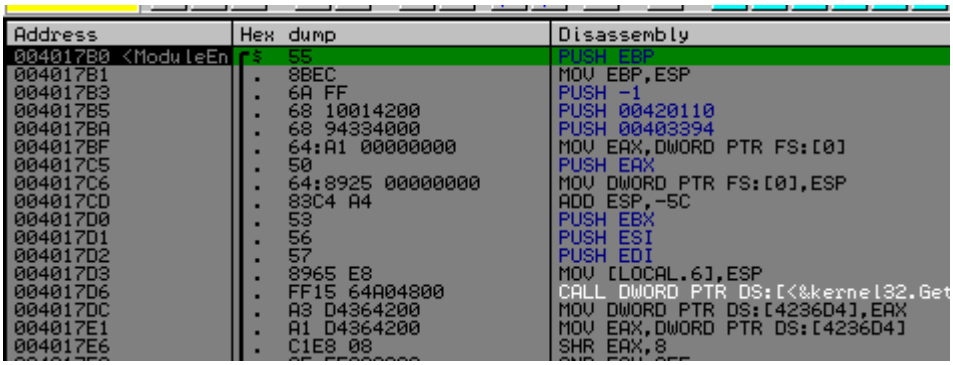
ImportREC محل شروع IAT را درست به دست آورده،اما Size آن اشتباه است.باید برابر 16C باشد:



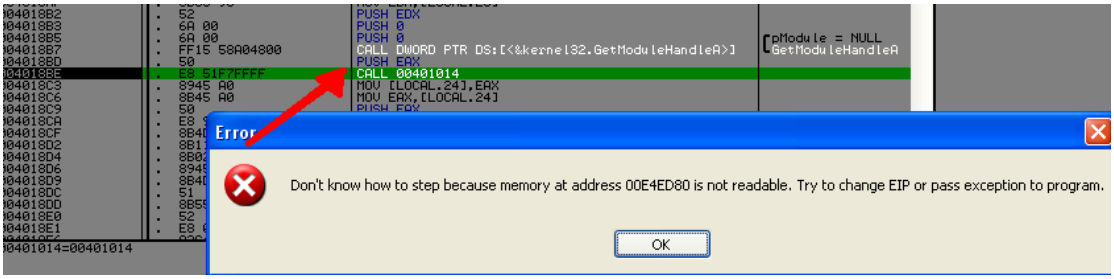
حالا فایل دامپ خود را فیکس کنید و آن را اجرا کنید:



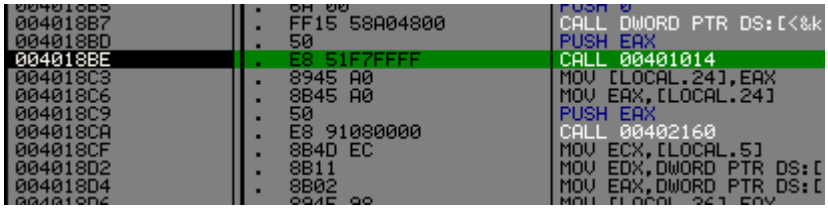
☹ خوب،معمولا در این موارد باید بررسی کنیم ببینیم چرا این فایل اجرا نمی شود؟...در چه خطی خطا می دهد؟...پس فایل آنپک شده را در OllyDBG جدید(آن Olly که فایل پک شده را در آن دیباگ کرده بودید،نبندید) باز کنید:



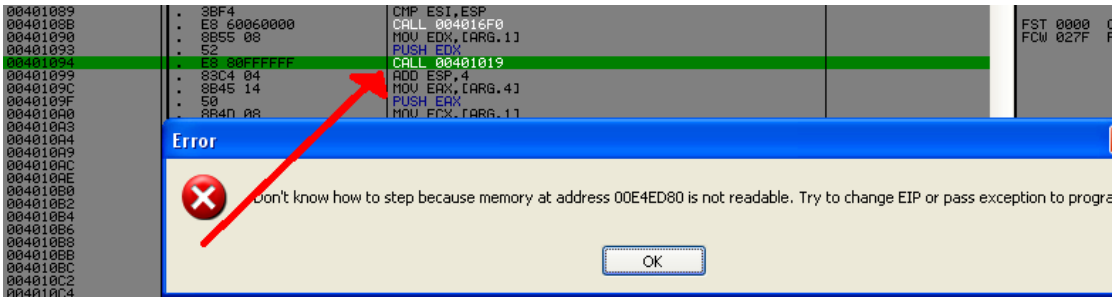
برنامه را با F8 ادامه دهید تا ببینیم در کجا برنامه دچار خطا می شود؟



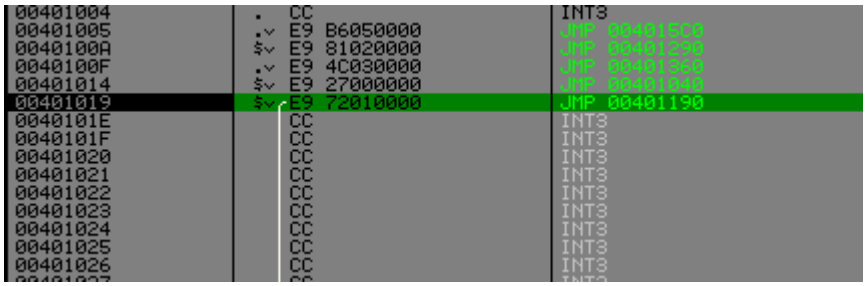
می بینید؟...وقتی برنامه به خط 4018BE رسید،و آن خط اجرا شد.برنامه با خطا مواجه شد.پس خطا باید در داخل این CALL باشد.پس بر روی این خط BP گذاشته و برنامه را ری استارت می کنیم و دوباره با F9 اجرا می کنیم تا به این خط برسیم:



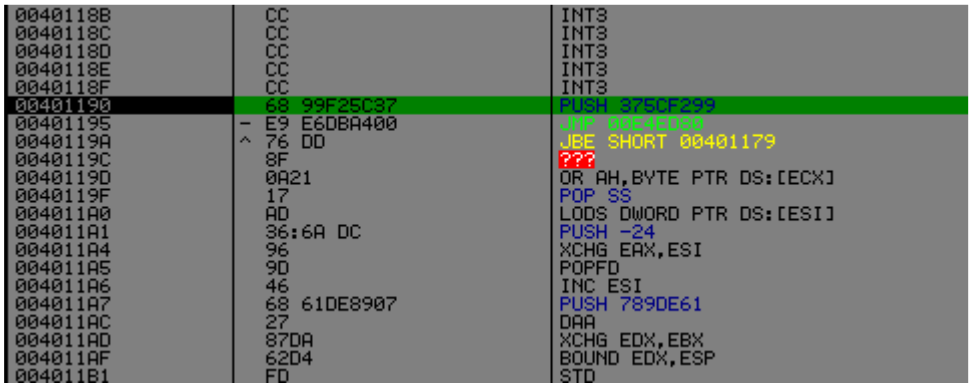
خوب،ما می دانیم خطا در داخل این CALL است.پس F7 را می زنیم و به داخل این CALL برویم و بعد با F8 ادامه می دهیم تا محل خطا را پیدا کنیم:



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:

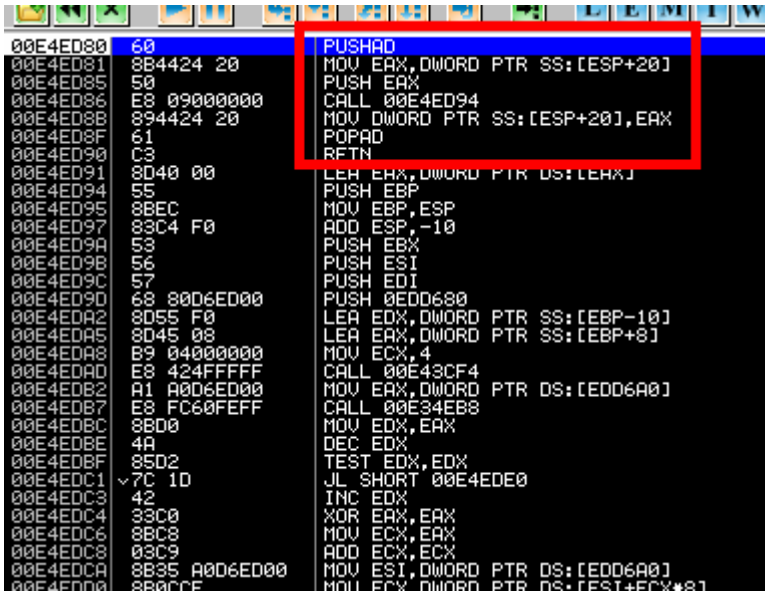


یکبار دیگر F8 را بزنید(و آنالیز OllyDBG را فایل بردارید) :

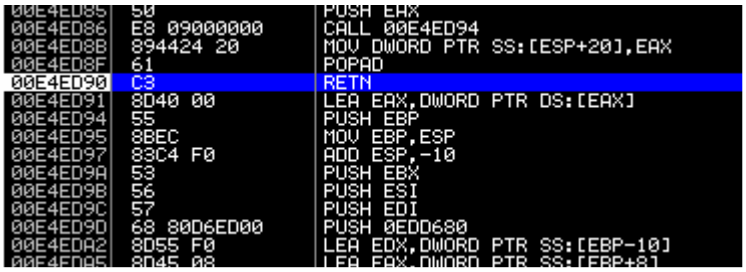


خطا در خط بعدی یعنی در 401195 اتفاق می افتد ☹ از کجا فهمیدم؟...خب معلومه...چون نوشته شده است JMP 00E4ED80 ...و اصلا آدرس 00E4ED80 در داخل فایل وجود ندارد.لذا فایل در این خط دچار خطا می شود...خوب،حالا بذارید ببینیم چرا وقتی فایل پک شده بود،خطا اتفاق نیفتاد و فایل اجرا شد؟...پس در فایل پک شده (آن Olly را که نبستید؟) به این خط بروید:

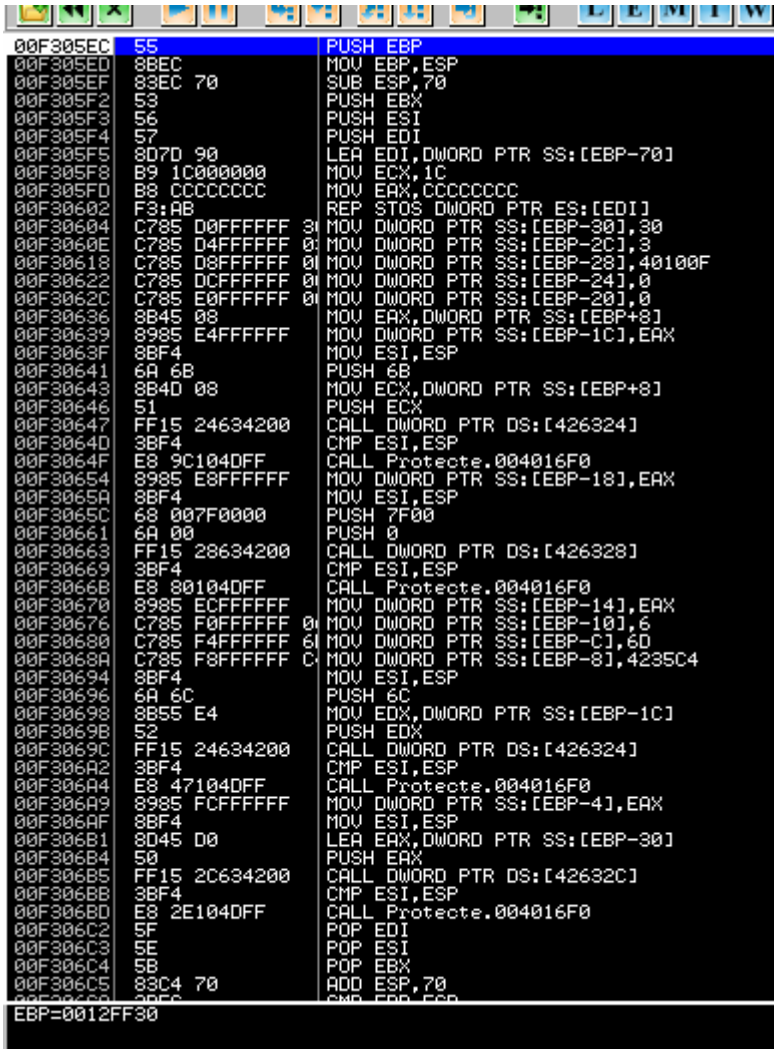
در این خط BP بگذارید و برنامه(فایل پک شده) را با F9 اجرا کنید تا در اینجا متوقف شویم،حالا کلید F8 را بزنید:



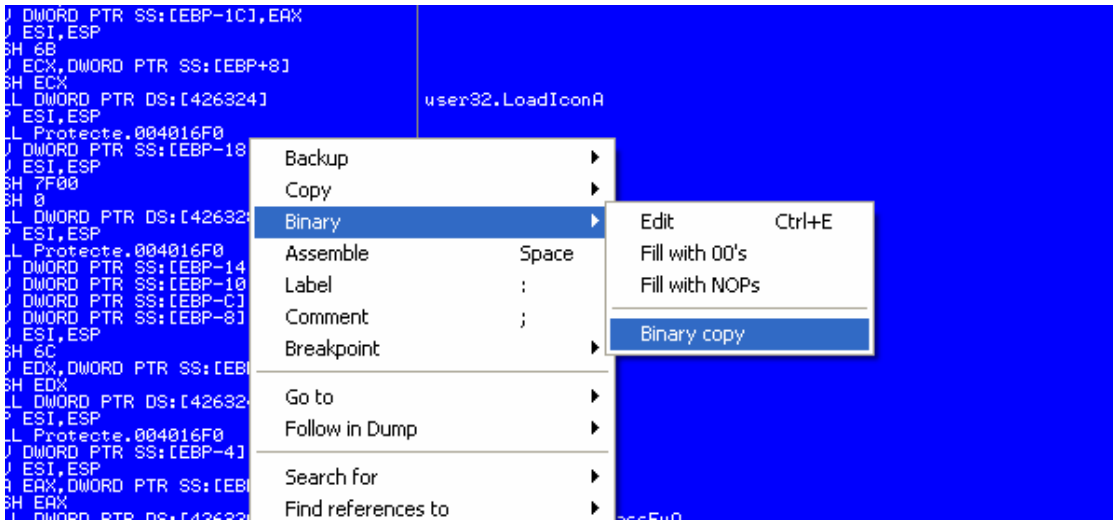
می بینید؟...در فایل پک شده یک همچین آدرسی وجود دارد.کدهایی که در مربع قرار دارند،کدهایی هستند که رمز Function را باز می کنند...بعد از اینکه کدها به طور کامل Decrypt شد،برنامه به خطی می رود که کدهای Decrypt شده در آنجا قرار دارد.پس چند بار کلید F8 را بزنید تا به RETN برسید:



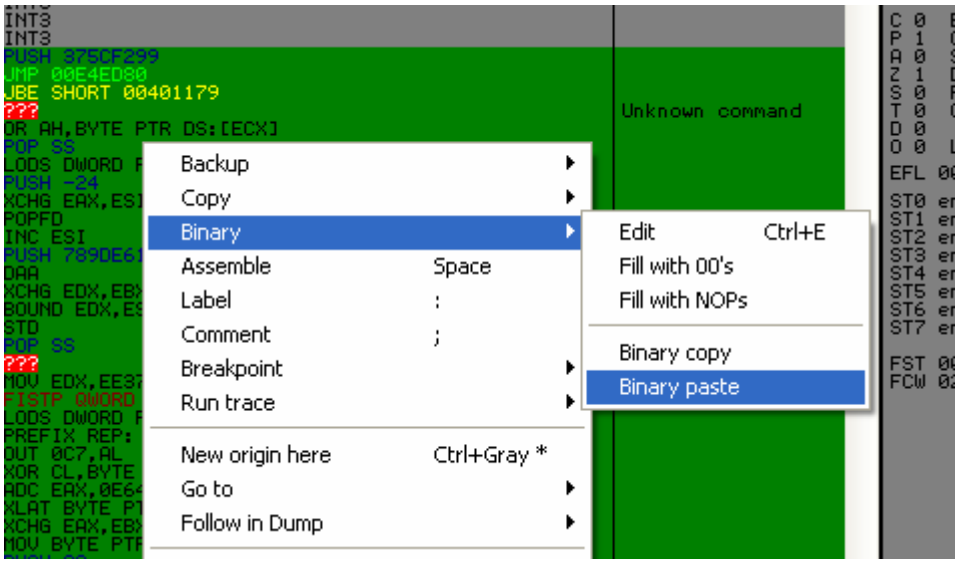
حالا یکبار F8 بزنید:

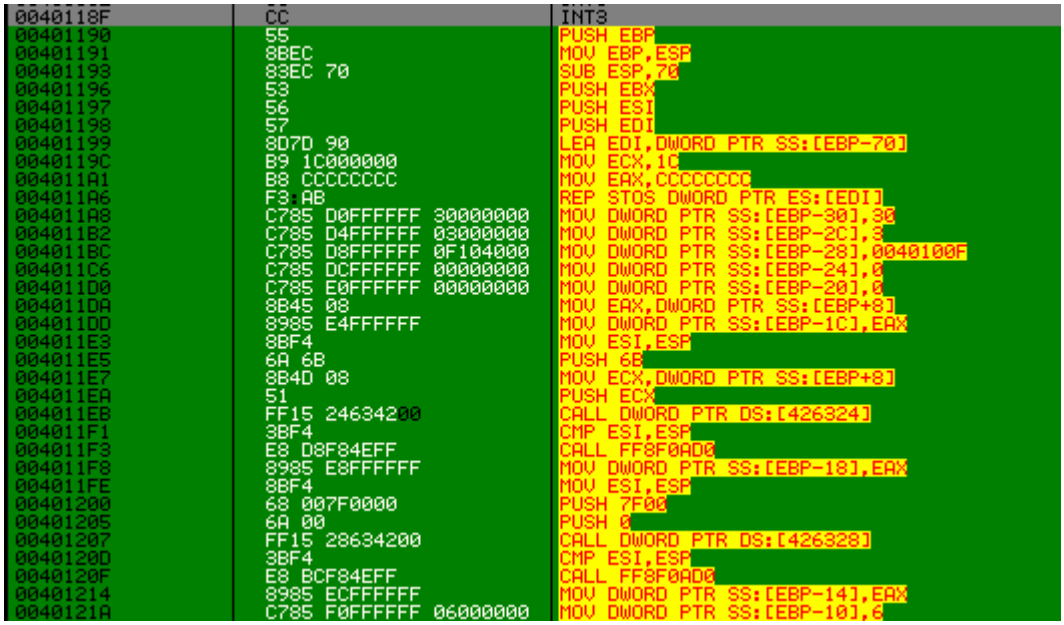


کدهایی که در اینجا قرار دارد،کدهای برنامه اصلی ما بوده...کدهایی که رمزگذاری شده و با آن چند خط رمزش باز شده...حالا برنامه می تواند از این کدها استفاده کند.
خوب،برای حل مشکل اجرا نشدن برنامه ما می توانیم این کدها را به فایل آنپک شده خود اضافه کنیم.این کدها را انتخاب می کنید و همانند تصویر کلیک راست کرده 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):





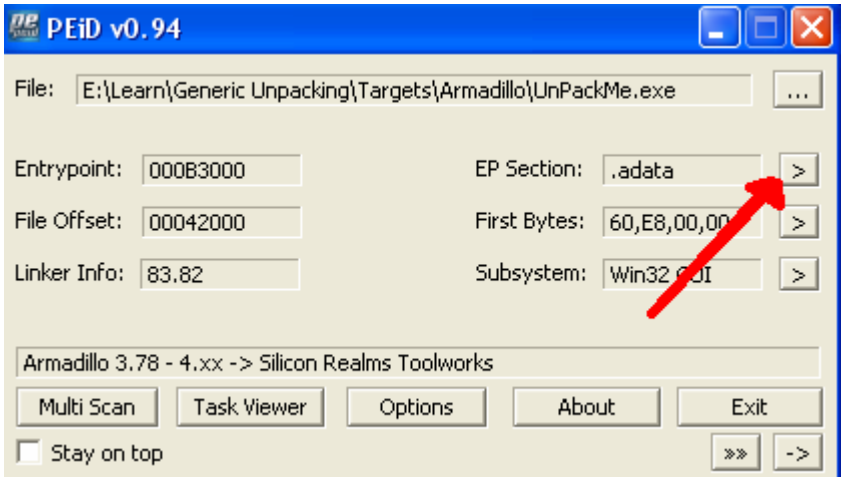
خوب...حالا کدهایی رو که کپی کرده اید را با دقت نگاه کنید.چون اطلاعات Binary Copy شده،آدرس Call ها تغییر کرده.الان مثلا در تصویر بالا در خط 4011F3 نوشته شده : CALL FF8F0AD0،اما اگر به فایل پک شده نگاه کنید،می بینید نوشته شده : CALL 004016F0 ...پس این موارد کوچک را درست کنید و بعد کدها را یک دور با دقت 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.

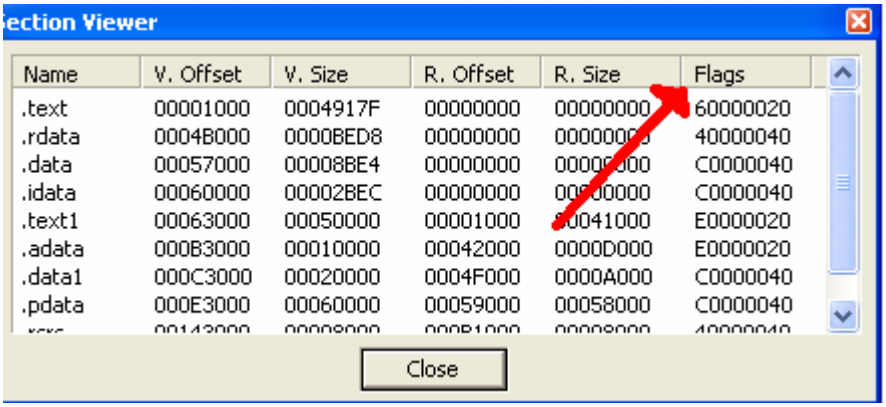
نکات ریز و درشتی که یادگیری آن لازم است

۱.FLAG ها

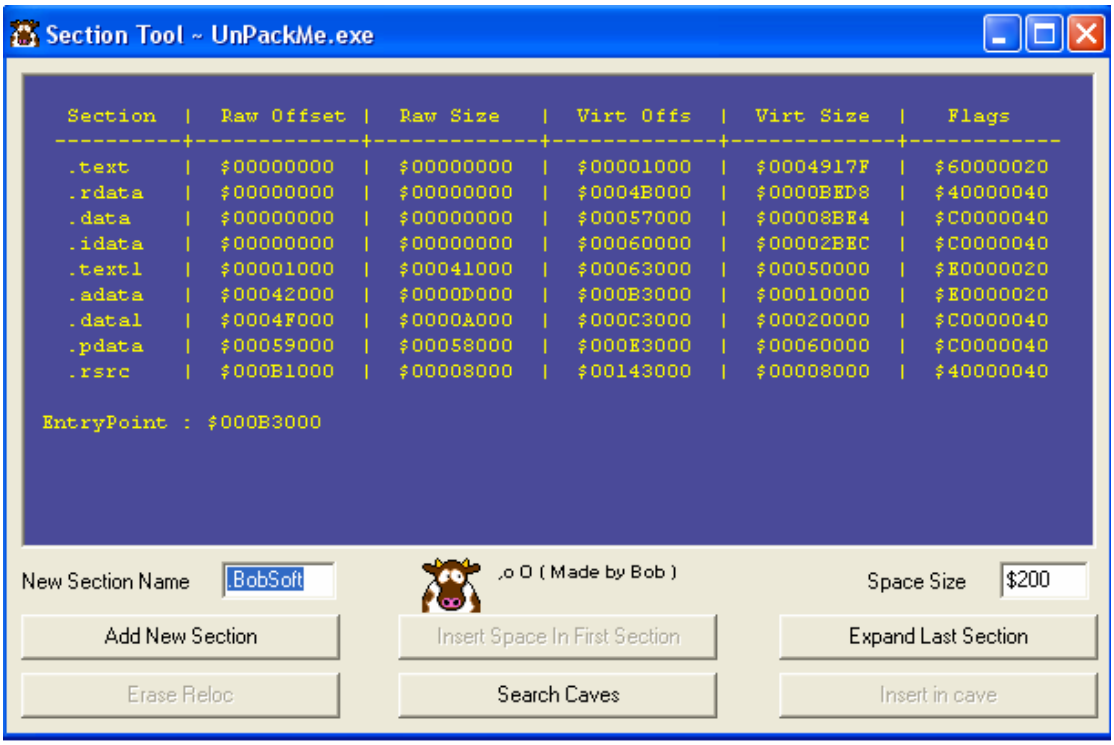
یک فایل در PEID باز کنید:



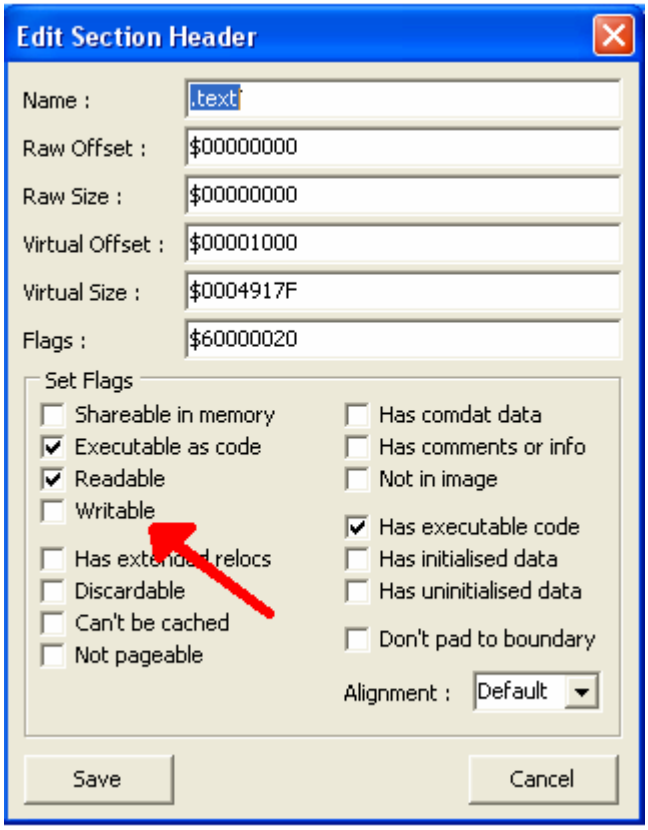
دکمه نشان داده شده را بزنید:



قبلا در مورد چهار قسمت V.Offset و ... توضیح دادم،اما در مورد Flags توضیحی ندادم.**هر سکشن 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:



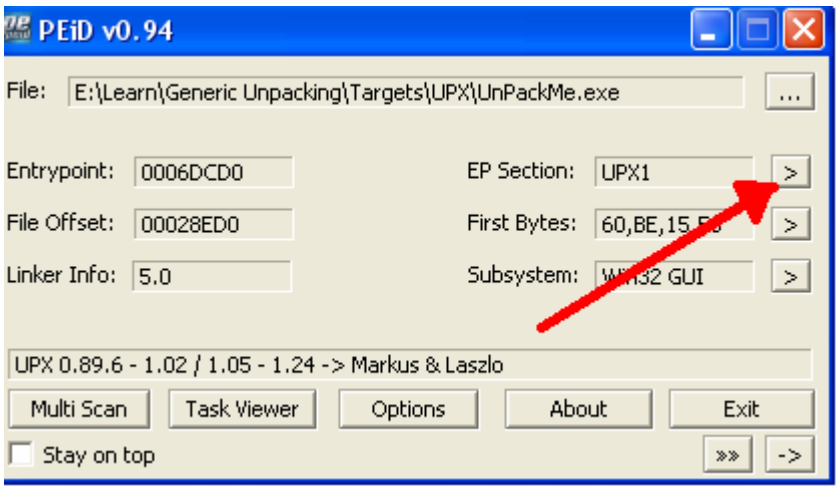
حالا بر روی هر سکشنی که کلیک راست کنید.می توانید مشخصات آن سکشن را ببینید.بر مثال من بـر روی سکـشن text. کلیـک راست می کنم:



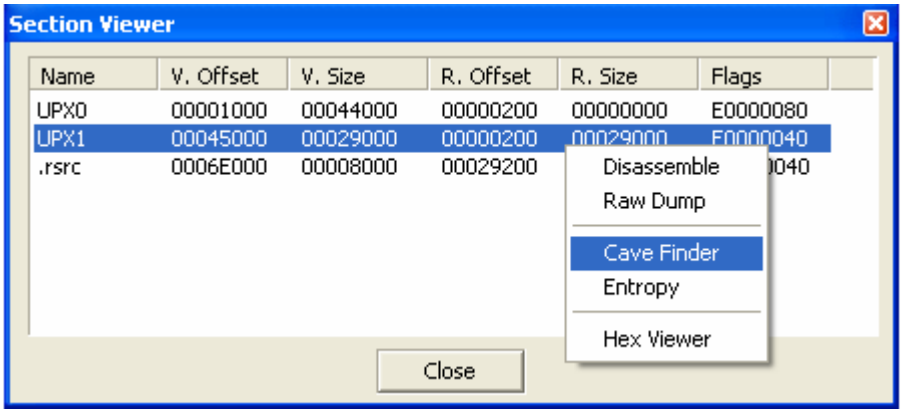
همانطور که می بینید،این سکشن Writable نیست.یعنـی اگـر شـما در Inline Patch کـدهایی بنویـسید کـه ایـن سکـشن را تغییـر دهد،برنامه دچار خطا می شود چرا که این سکشن Writable نیست.در موقع Inline Patch کـردن بایـد حواسـتان بـه Flag سکـشن مورد نظر باشد.و اگر لازم شد آن سکشن را Writable کنید.(یعنی در اینجا تیک 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.



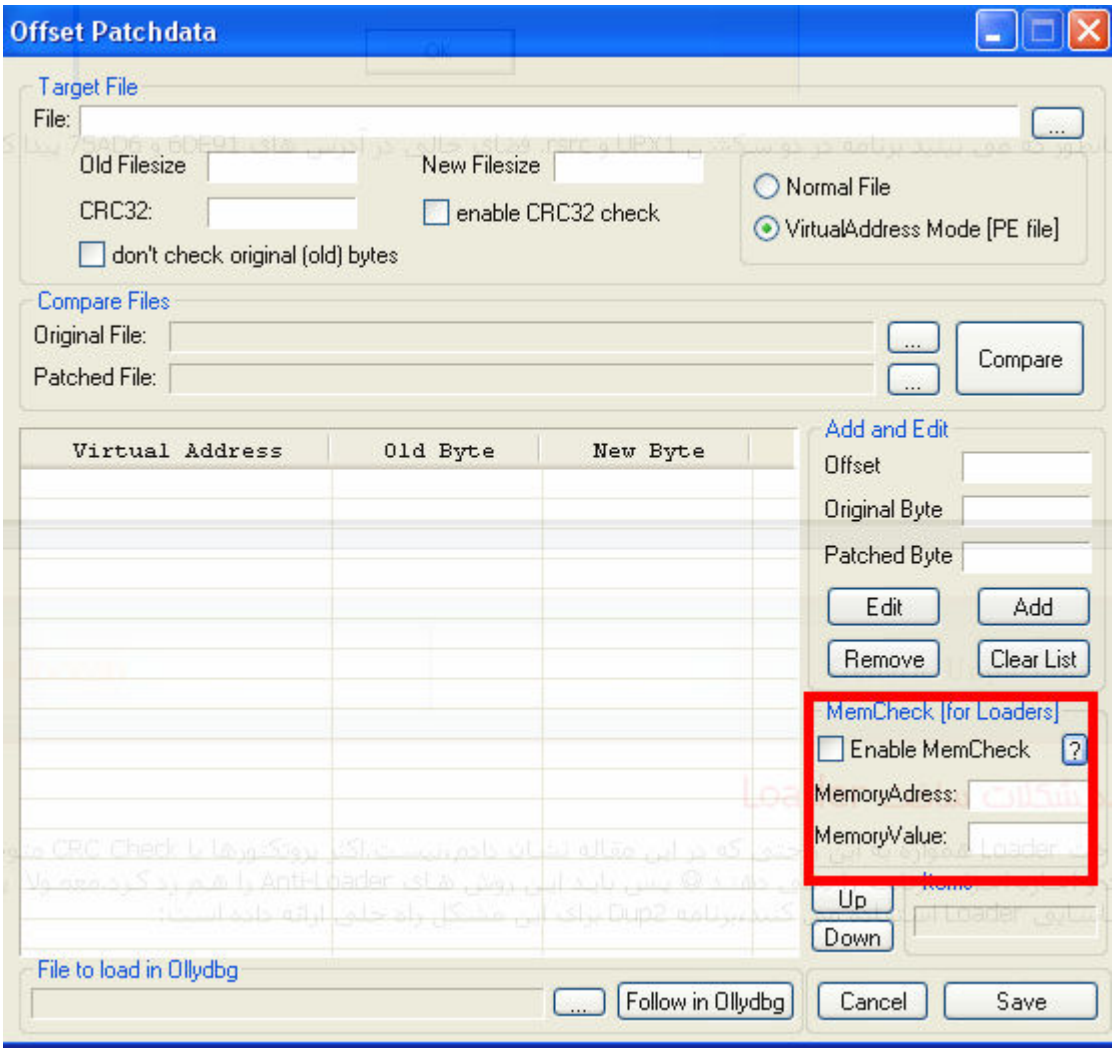
دکمه نشان داده شده را بزنید.حالا برای یافتن فضای خالی بر روی یکی از سکشن ها کلیک راست کرده و گزینه Cave Finder را بزنید:



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

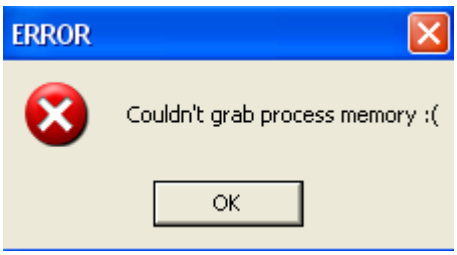
ساخت Loader همواره به این راحتی که در این مقاله نشان دادم،نیست.اکثر پروتکتورها با CRC Check متوجه وجود Loader خواهند شد و اجازه اجرای برنامه را نمی دهند ☺ پس باید این روش های Anti-Loader را هم رد کرد.معمولا پکرها از CRC Check برای شناسایی Loader استفاده می کنند،برنامه Dup2 برای این مشکل راه حلی ارائه داده است:



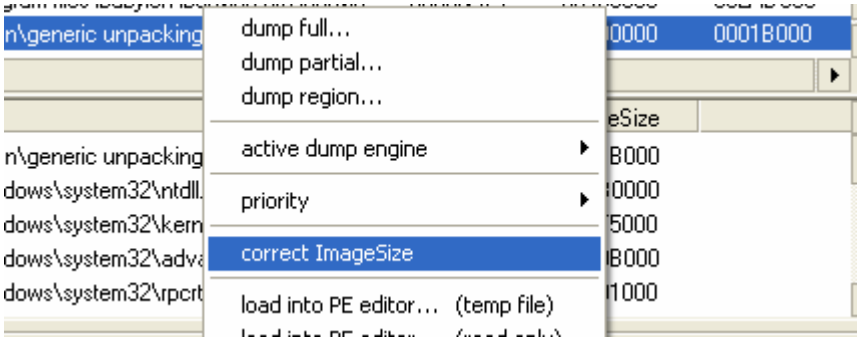
جایی که در تصویر بالا نشان دادم،قابلیتی است که با آن می شود، Anti-Loader ها را رد کرد.شما باید مقدار Dword که بعد CRC Check نوشته می شود را به برنامه بدهید تا برنامه قابلیت Anti-Loader در پکر را رد کند ☺
البته ABEL Loader هم قابلیتی برای رد کردن Anti-Loader دارد.آن هم اینکه Caption برنامه هدف را به ABEL 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

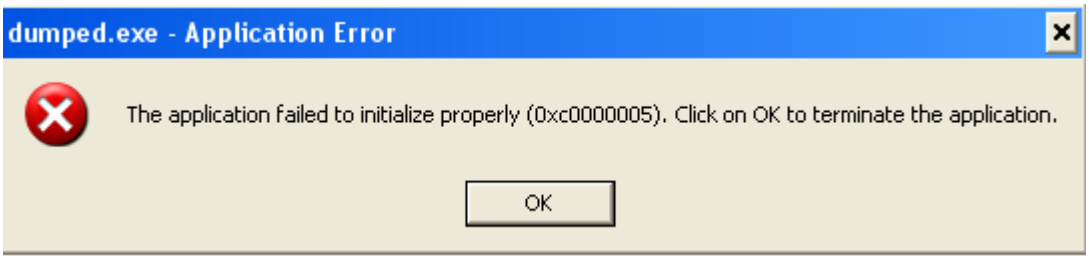
برخی از پکرها،قابلیت هایی برای سخت کردن دامپ گرفتن از برنامه دارند.مثلا یک پکر می آید و سایز یکی از سکشن هایش(معمولا سکشن آخری) افزایش می دهد تا نتوان یک دامپ درست از فایل گرفت:



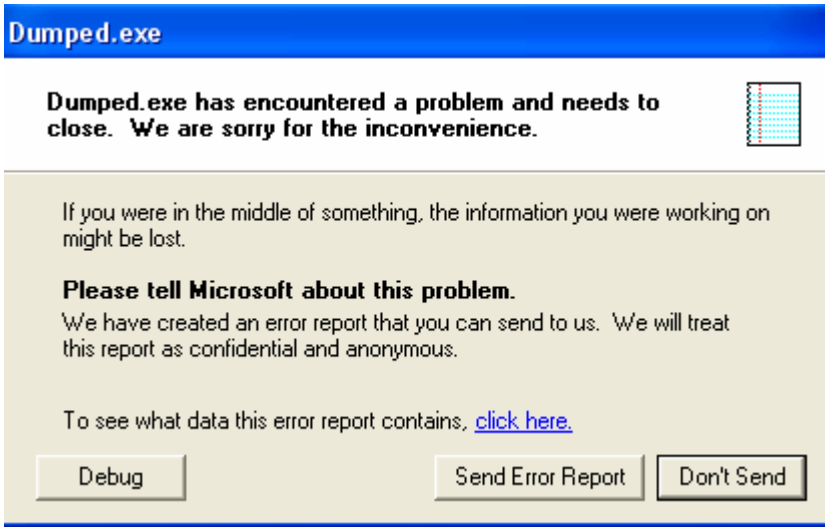
در این موارد در برنامه LordPE بر روی فایل مورد نظر کلیک راست کرده و گزینه Correct ImageSize را بزنید و دوباره برای دامپ گرفتن از فایل تلاش کنید:



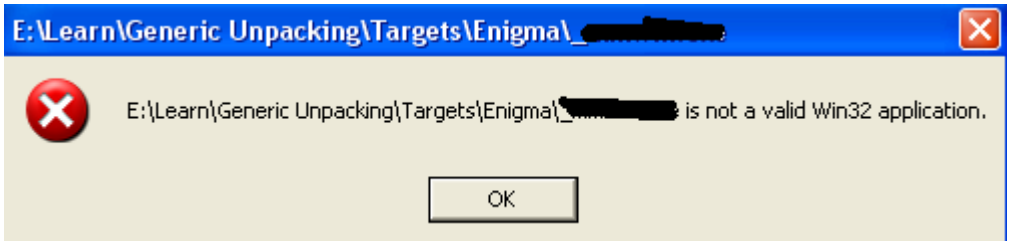
سعی کنید برای دامپ کردن فایل های DLL از OllyDump استفاده نکنید.بهتر است از LordPe یا PeTools استفاده کنید.
یک چیز دیگر اینکه،سعی کنید برای دامپ کردن،برنامه های مختلفی در اختیار داشته باشید،اگر یکی جواب نداد.اون یکی جواب می ده ☺
در ضمن،یک Dump سالم باید یک همچین پیغامی بدهد(به خاطر نبود Import Table همچین خطایی می دهد) :



یا Don't send بدهد:

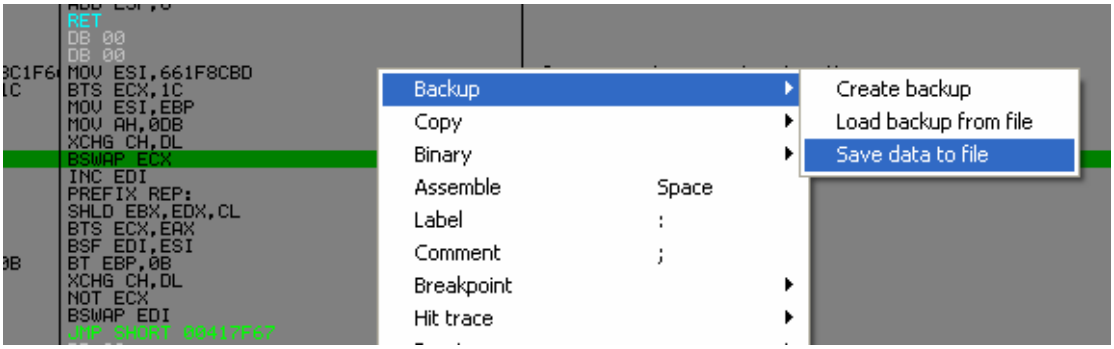


اگر فایل دامپ، همچین پیغامی بدهد یعنی فایل درست دامپ نشده:

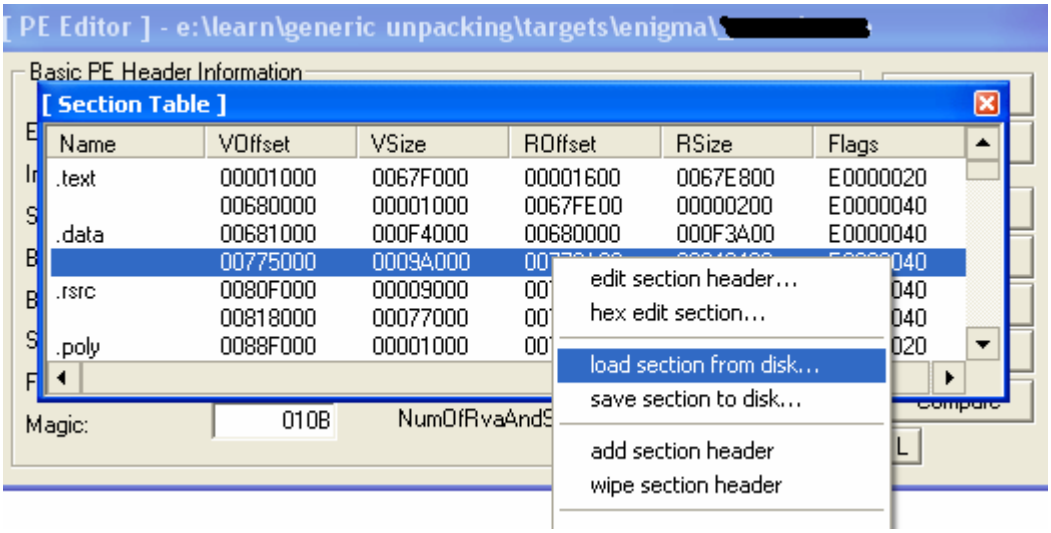


۵.اضافه کردن سکشن

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 OllyDBG 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 در LordPe 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 ☺