بسم الله الرحمن الرحیم

سری مقالات آموزش کرک

ترجمه و تالیف:

سید وحید هاشمی

(netspc)

مقدمه:

به نام خداوند جان و خرد.

در ابتدا جا دارد از دوستان خوبم در سایت آنریل (www.unreal-rce.net) بابت توجه و کمک به بنده در زمینه مهندسی معکوس تشکر عرض نمایم بخصوص از دوستان عزیزم :
jnop790, soda, black.byte, android , Newbie و دیگر دوستانی که اسامی آنها از خاطرم رفته است.

این سری مقالات حاصل تلاش چندین ماهه بنده در امر مهندسی معکوس میباشد و بر اساس سری آموزشی ۱۰ قسمتی اساتید سایت ARteam میباشد. تمامی این ۱۰ قسمت در طی ۲ ماه تهیه گشته است ولی بنا به دلایلی قسمت دهم این مجموعه در اختیار دوستان قرار نخواهد گرفت در ضمن در آخر دو ضمیمه وجود دارد که یکی نحوه آنپک MEW میباشد که بصورت تجربی کسب کرده بودم و دیگری ترجمه و جمع آوری چند مقاله درباره نحوه نفوذ به RING 0 در سیستم عامل ویندوز میباشد.

نحوه نوشتار این سری مقالات بصورت ترجمه و تالیف میباشد در واقع در بدو شروع کار ترجمه این سری سعی کردم تمامی مطالب نویسندگان اصلی را خود درک کنم و از آنها بهره کافی را ببرم و سپس بصورت نوشتار ساده و با استفاده از تجربیات خود آنها را در اختیار دوستان قرار دهم. چهار قسمت از ده قسمت این سری بصورت جداگانه در سایت آنریل قرار داده شده بود و قسمتهای بعدی نیز آماده ارائه بود ولی به دلیل اتخاذ برخی از تصمیمات و عملی نشدن آنها و درضمن به دلیل سفر علمی بنده به هندوستان نشر این سری کمی به تعویق افتاد. مقالات موجود بصورت ویرایش نشده از نظر نثر پارسی میباشد که امیدوارم دوستان به بزرگواری خودشان بنده را عفو نمایند.

برای استفاده از این مجموعه فقط یک نکته ای که میتوانم به دوستان خاطر نشان کنم اینست :تا هنگامی که دقیقاً چرایی و چگونگی دلیل به کار رفته در آموزشها برای شما کشف نشده و دقیقاً مطلب مورد نظر را نفهمیده اید بر روی آن مسئله فکر کنید ، چرا که در زمینه مهندسی معکوس به نظر من مهمترین چیز استفاده از عقل خود میباشد و اینکه همیشه سعی کنید خود را به جای برنامه نویسی که برنامه مورد نظر را نوشته است قرار دهید و فکر کنید در آن حالت از چه تکنیکی برای جلوگیری از نفوذ استفاده میکردید.

در آخر پوزش بنده را به دلیل تاخیر و نثر ویرایش نشده بپذیرید، به دلیل علاقه به مبحث برنامه نویسی سیستم عامل (http://www.persianos.org) و کمی وقت دیگر نمیتوانم در این زمینه فعالیتی انجام بدهم امیدوارم در آینده نه چندان دور دوباره بتوانم به جمع شما عزیزان بپیوندم و از تجربیات شما دوستان استفاده کافی را ببرم.

و من الله توفیق

سید وحید هاشمی

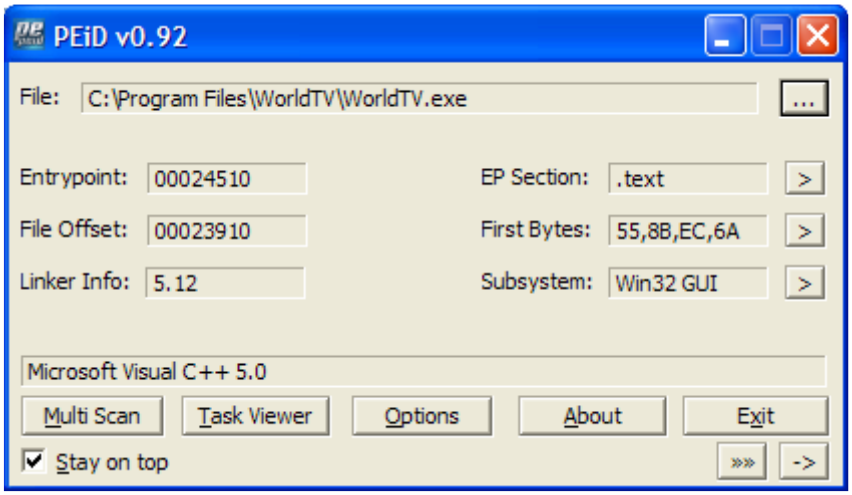
اردیبهشت ۱۳۸۵ – تهران

فروردین ۱۳۸۶ – بنگلور

آموزش کرکینگ با ollydbg قسمت اول
تاریخ : فروردین ۱۳۸۵ April 2006
نوع حمله serial fishing
نرم افزار هدف : WorldTV 7.1 ضمیمه شده
نرم افزارهای مورد استفاده :ollydbg 1.10 , wdasm32 ,PEid 0.92
سطح : مبتدی

سلام این مقاله اولین سری از ترجمه مقالات گابریل میباشد البته خودم یکسری از مطالب رو که گابریل توضیح نداده اسـت رو واضح تر توضیح دادم پس میشه گفت تالیف!!!!

خب اول نرم افزار رو نصب میکنیم.
بعد از نصب همیشه اولین کاری که باید بکنیم اینه که اول بفهمیم برنامه رو با چی نوشتن و با چی پک شده برای اینکار میتونید از نرم افزارهایی مانند PEid و یا RDG Packer Detector استفاده کنید من اینجا از PEid استفاده کردم.
برنامه PEid رو باز کرده و فایل WorldTV.exe رو بندازید توش و یا از دکمه ...(open) برای تستِ فایل استفاده کنید

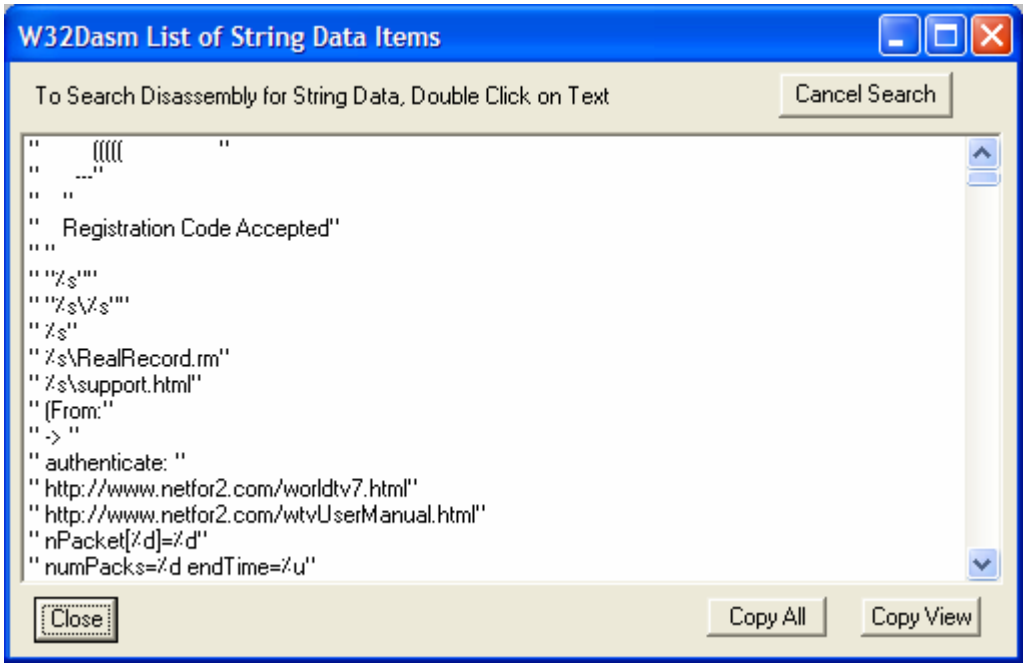


شکل ۱

بعد از نصب یکبار برنامه رو اجرا کرده یک صفحه میاد که میگه سریال نامبر بده یه شماره همینجوری میزنیم یـه پیغـامی میـاد اونو یادداشت میکنیم برای راحتی کار من اون پیغام رو براتون مینویسم:

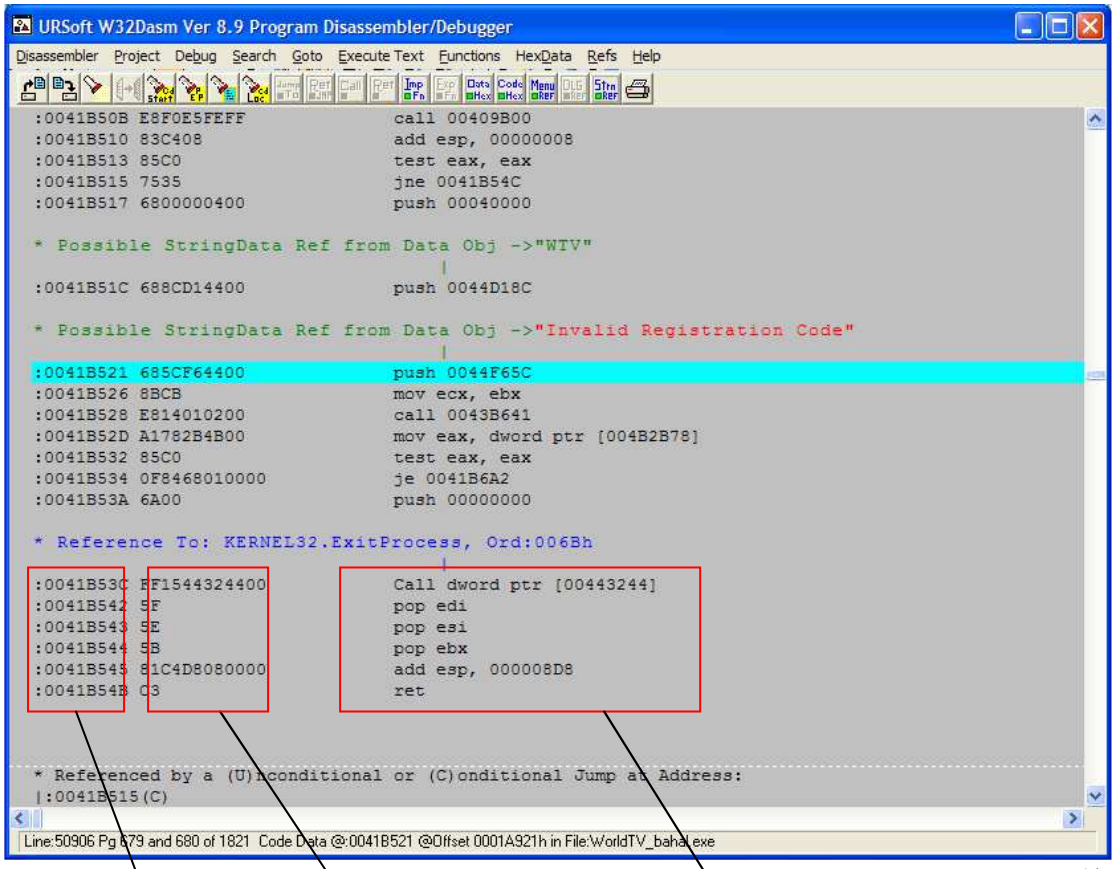
Invalid Registration Code

خب حالا w32dasm رو باز کرده اول از همه از منوی فایل disassembler گزینه font رو انتخـاب کـرده و بعـد از اون گزینه select font بعد فونتی که باهاش راحتر هستید رو انتخاب کنید و بعد دوباره تو همون منو گزینـه save default font رو بزنید برای اینکه همیشه محیط w32dasm با همین فونت باز بشه!
خب بعد از اینکار دوباره منوی disassembler رو زده و گزینه open file.... رو انتخاب کرده و آدرس جایی کـه برنامـه رو نصب کردید بدید و فایل WorldTV.exe رو بازکنید و از منوی Refs گزینه String Data Refrence رو انتخاب کرده یک دایلوگ جدید باز میشه مثل زیر:



شکل ۲

از توی این دایلوگ بگردید تا اون رشته ای که قبلاً یادداشت کردید رو پیدا کنید وقتی پیداش کردید حالا روی اون دوبار کلیک کنید وقتی اینکار رو میکنید صفحه پشتی نوشته هاش عوض میشه حالا این دایلوگ رو بسته شکلی همانند شکل زیر خواهید دید.



شکل ۳

آدرسها

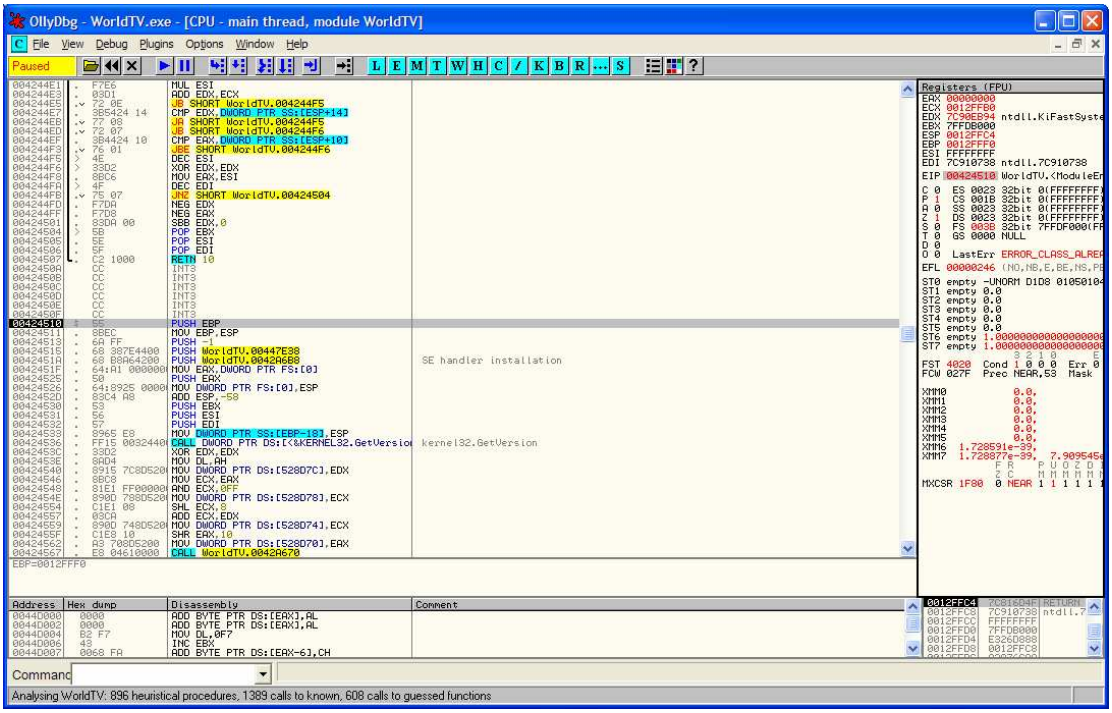
دستورات ماشین یا OP code

کدهای disassembly شده

خب اینجا جایی هست که وقتی ما کد اشتباه وارد میکنیم به این قسمت از برنامه می‌آییم.
حال میخواهم کمی در مورد ساختار این برنامه توضیح بدم توجه کنید که کدهایی کـه در زیـر میبنیـد تمامـاً pseudo code هست و هیچ ربطی به کد اصلی برنامه نداره
فرض کنیم من کسی که این برنامه رو نوشته میشناسم و این کدها رو هم اون به من داده
وقتی برنامه شروع میشه این برنامه میره یه جایی رو برای کد رجیستر میگرده وقتی پیدا نکرد میاد داخل یه شرط مثل زیر:

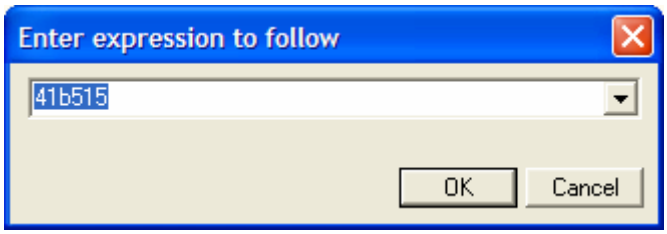
```
If  found the register key then
        Jump to start program address
Else If not found the register key then
        Jump to exit program address
        Or goto evaluation procedure
End if
```

خب با توجه به شبه کد بالا دوباره به شکل۳ توجه کنید اون رشته مورد نظر ما بین دو دستور jump هـست یکـی در بـالا در آدرس 41B515 (این آدرس رو یادداشت کنید) و دیگری در پایین در آدرس 41B534 با توجه به توضیح بالا میفهمیم کـه دستور پرش اول برای اجرای کد برنامه و پرش دوم برای خروج از آن هست خب حال باید چه کنیم وقتی که ما شماره سـریالی در دست نداریم؟
بسیار ساده هست کافیست که به دستور شرطی پرش اول بگوییم که باید همیشه این شرط اجرا شده و پرش انجام شـود.خـب برای اینکار آدرس پرش اول را یادداشت کرده و برنامه w32dasm رو بسته و برنامه ollydbg را باز کرده و فایل مورد نظر را درون آن باز کرده مانند :



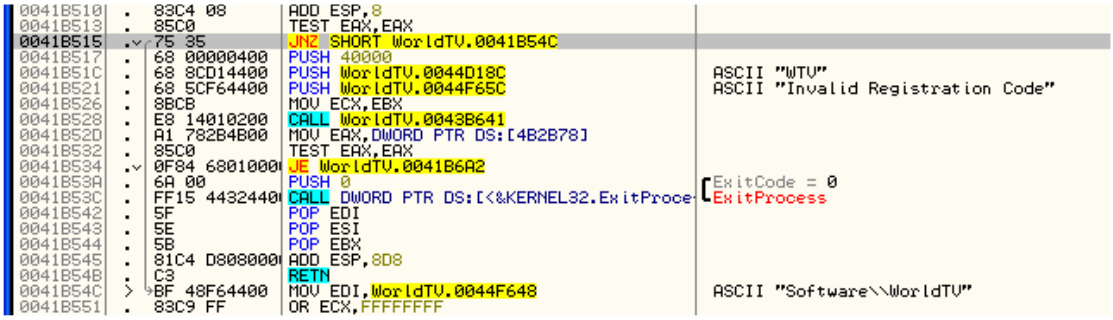
شکل ۴

خب از جعبه ابزار بالا دکمه  را انتخاب کرده و آدرسی را که یادداشت کرده بودیم در آن وارد میکنیم مانند



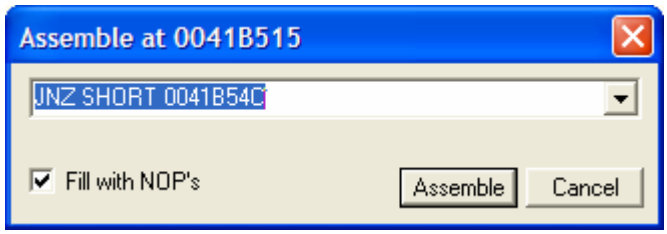
شکل ۵

و روی دکمه ok کلیک کرده تا olly به آدرس مشخص شده برود مانند



خب این نقطه جایی هست که ما باید اولین تلاش خود را برای کرک کردن برنامه بکنیم.

برای کرک کردن برنامه کدهای جاری را با کدهایی که خود میخواهیم اجرا شود باید جابجا کنیم برای اینکار از سمت راست بر روی ستون سوم یعنی همانجایی که نوشته شده است JNZ دوبارکلیک کرده تا مانند شکل زیر دایلوگی را ببینید:



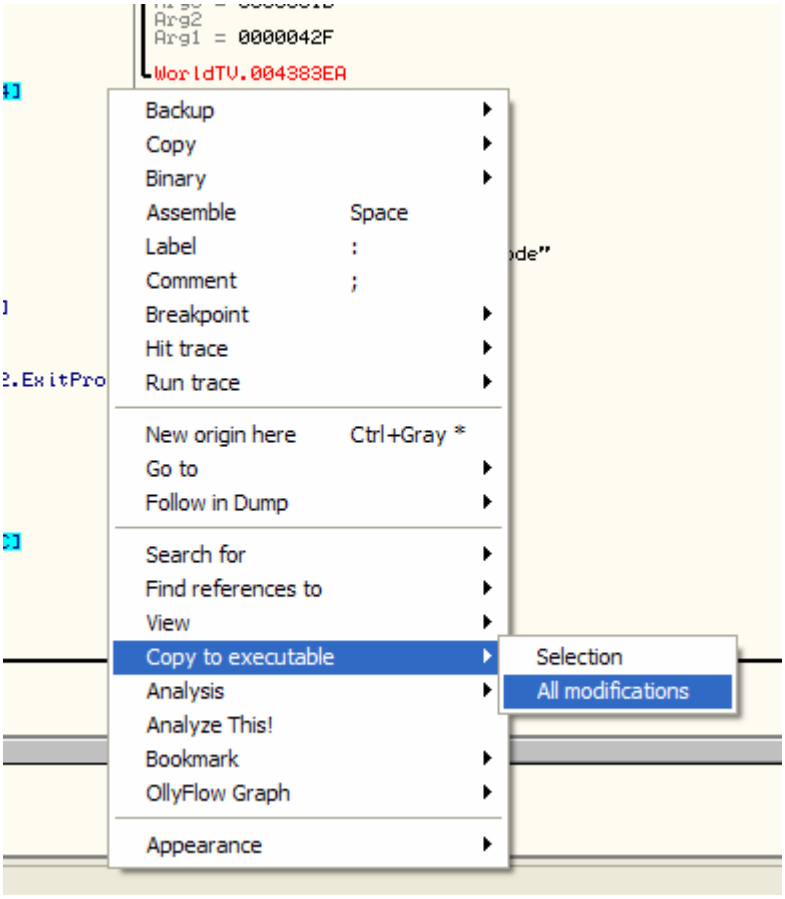
شکل ۶

خب حال دستورالعملی که در این دایلوگ قرار دارد را با این دستورالعمل جایگزین کنید:

JMP SHORT 0041B54C

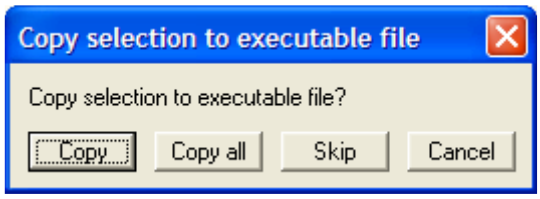
خب این کد یعنی چه؟ در کدی که در خود برنامه وجود داشت پرش هنگامی انجام میشد که مقدار EAX (به شکل ۶ دستور بالای JNZ توجه کنید) صفر نشود ولی چون ما شماره سریال معتبر نداریم کافیست این دستور را (JNZ)با دستور (JMP) که به معنی پرش بدون شرط میباشد جایگزین میکنیم.

حال برای ذخیره فایل کرک شده بر روی جایی که سفید میباشد کلید سمت راست موس را فشار داده و مانند شکل زیر عمل میکنیم:



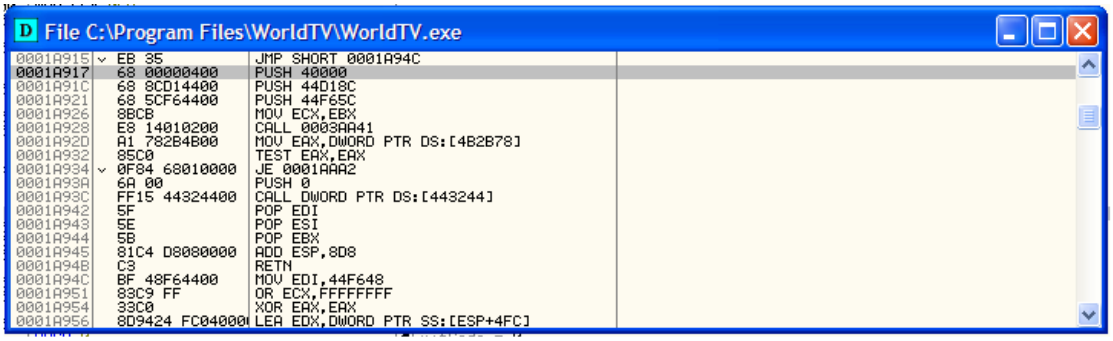
شکل ۷

بعد گزینه all modifications را انتخاب کرده دایلوگی مانند زیر خواهید دید:



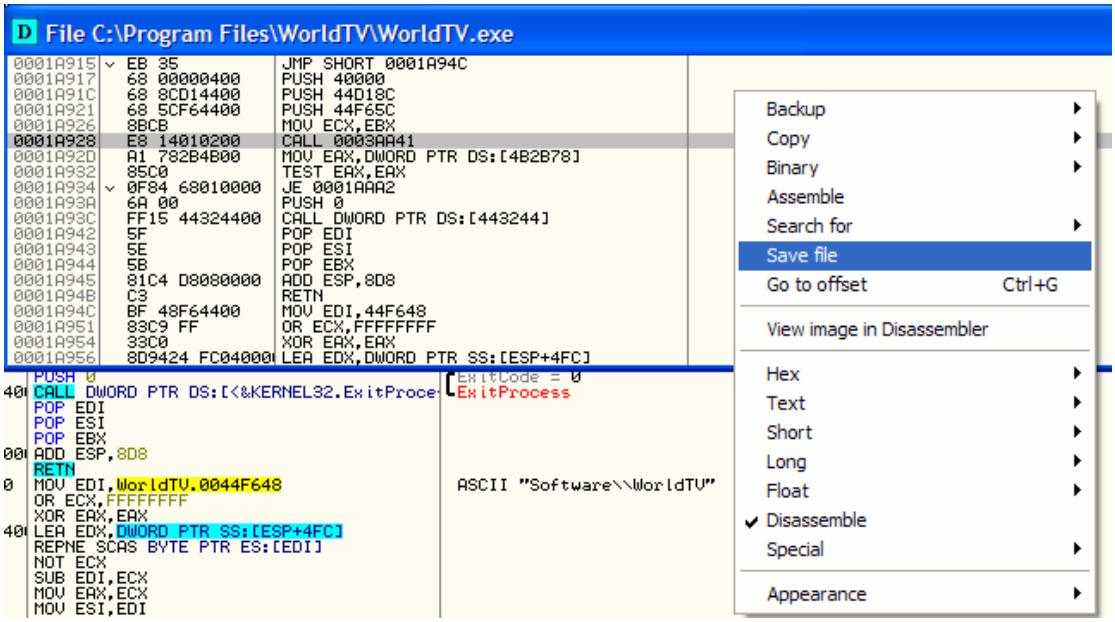
شکل ۸

گزینه copy all را انتخاب کرده دایلوگی مانند زیر خواهید دید:



شکل ۹

دوباره بر روی یک جای سفید دکمه سمت راست موس را بزنید و مانند شکل زیر گزینه save file را انتخاب کنید:



شکل ۱۰

توجه کنید حتماً فایل تغییر یافته را با نام دیگری غیر از نام اصلی فایل ذخیره کنید!

حال برنامه را اجرا کرده و عدی مثل ۱۲۳۴۵۶ را در داخل textbox زده و برروی دکمه validate کلیک میکنیم به قول خارجیهای Bingo ما موفق شدیم پیغامی با این مظمون میاد که رجیستر کد ما مورد تایید هست.

خب حالا برنامه رو ببندید و دوباره باز کنید میبینیم که هنوز از شما کد فعالسازی رو میخواد در واقع ما فقط تونستیم دستورالعملهای مربوط به چک کردن شماره سریال رو در فرم گرفتن شماره سریال خنثی کنیم.حال باید دوباره به ollydbg مراجعه کرده و ببینیم به چه راه حلی میتوانیم برسیم.

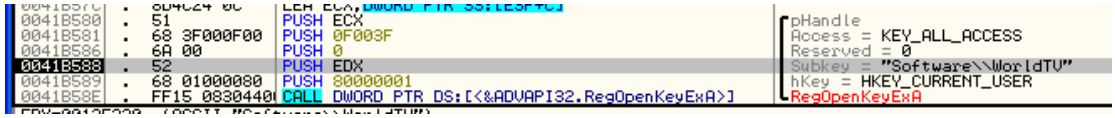
برای کرک برنامه ها معمولاً دو راه وجود دارد:

- پیدا کردن شماره سریال
- خنثی کردن کدهایی که شماره سریال را کنترل میکنند

بعد از باز کردن برنامه کرک شده اولین کاری که میکنیم به آدرس 41B515 رفته به همان طریقی که قبلاً به آن اشاره شد وبعد از آن در همان جا دکمه F2 را زده میبینیم که آدرس مورد نظر به رنگ قرمز در میآید این یعنی اینکه هر وقت ollydbg به این خط برسد اجرای دستورات را متوقف کرده و ما خواهیم فهمید که برنامه به این خط رسیده است خب برای اجرای برنامه کلید F9 را فشار دهید برنامه اجرا میشود دوباره یک عدد در textbox وارد میکنیم و دکمه validate را فشار میدهیم بعد از چند لحظه ollydbg در آدرسی که ما در آنجا F2 زدهایم متوقف خواهد شد. حالا برای اجرای قدم به قدم برنامه کلید F8 را فشار میدهیم. میبینیم که برنامه خط به خط اجرا شده و ما میتوانیم بر اجرای اون نظارت کنیم.

بعد از چند خط اجرای قدم به قدم با دکمه F8 متوجه میشیم که برنامه بعد از گرفتن سریال نامبر وارد رجیستری ویندوز میشود خب حال ما متوجه شدیم که برنامه برای ذخیره شماره سریالها به رجیستری ویندوز مراجعه میکند.

در آدرس 41B588 متوجه مطلب زیر میشویم:

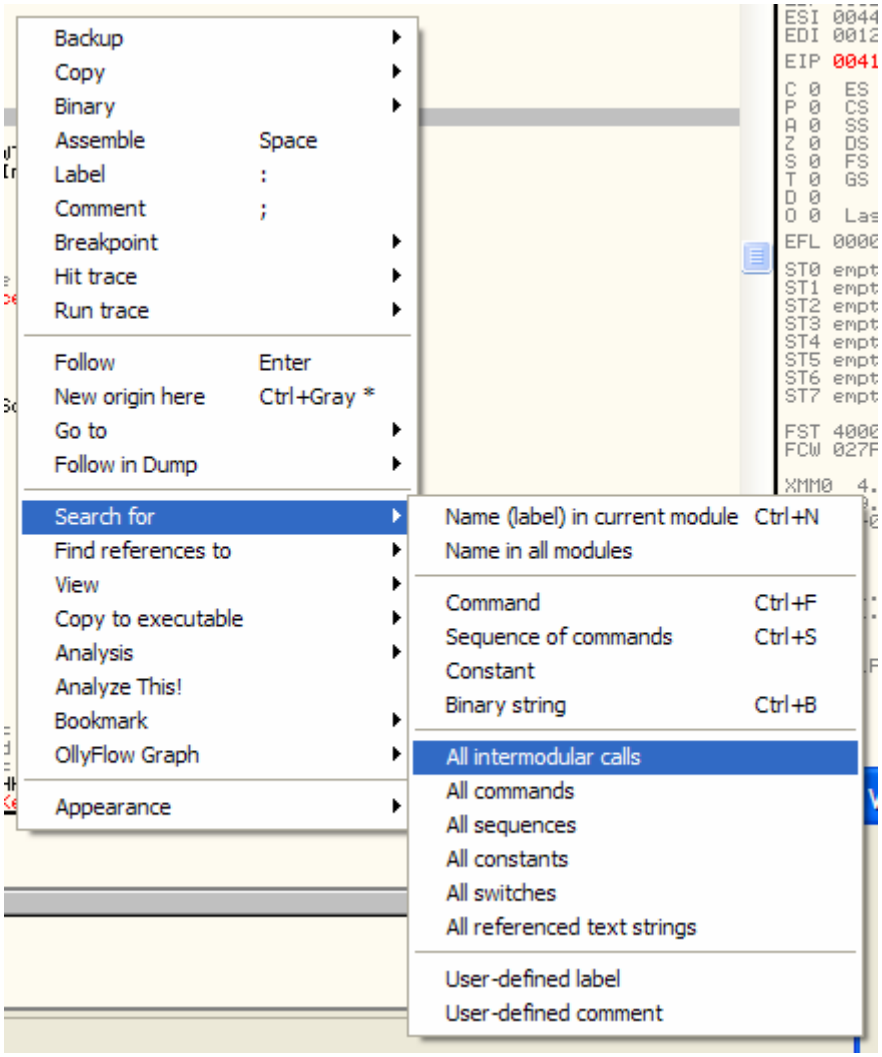


شکل ۱۱

یعنی در HKEY_CURRENT_USER\Software\\WORLDTV تمامی تنظیمات این برنامه ذخیره میشود ازجمله شماره سریال آن خب با توجه به این موضوع حال باید کاری کنیم که هرگاه برنامه درخواست خواندن یک مقدار از ریجستری را کرد ollydbg به ما اطلاع دهد برای اینکار ما باید از توابع API مطلع باشیم(برای اینکه در زمینه کرکینگ بهتر بتوانید پیشرفت کنید لازم است توابع API را بشناسید).

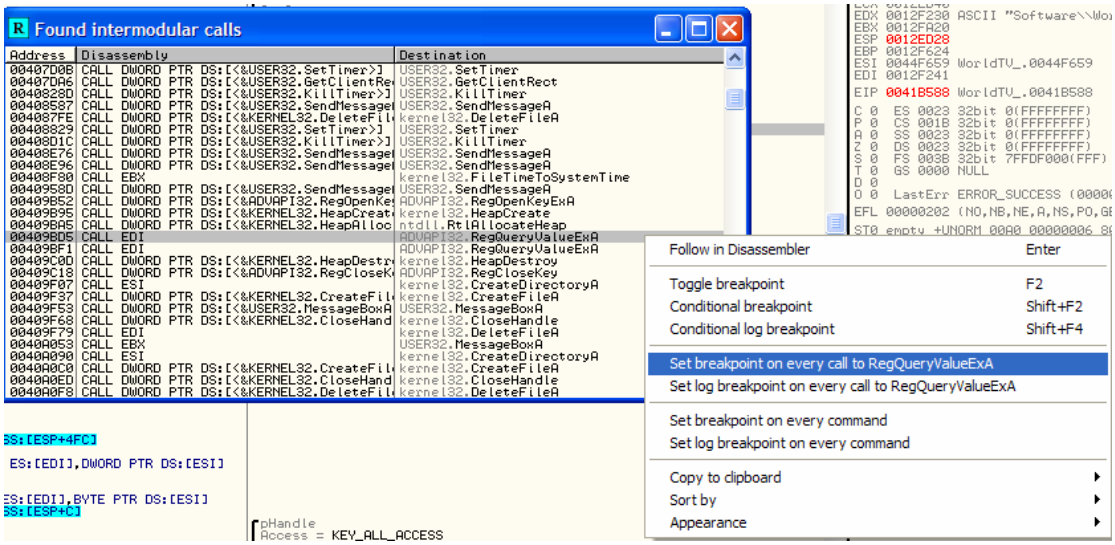
برنامه را بوسیله  ریستارت کنید و کارهای زیر را انجام دهید.

تابعی که مقادیر را از رجیستری ویندوز میخواند ReqQueryValueEx میباشد خب برای اینکه به ollydbg بگوییم هرگاه این تابع از طرف برنامه صدا زده شد به ما خبر بده دکمه سمت راست موس را زده و گزینه searchو بعد از آن گزینه all intermodular calls را انتخاب کنید مانند شکل زیر:



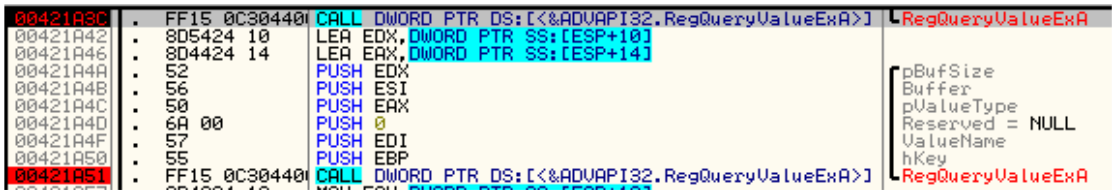
شکل ۱۲

صفحه مانند شکل زیر ظاهر میشود در این صفحه ReqQueryValueEx را تایپ کرده هنگامی که تابع مورد نظر یافت شد بر روی آن کلیک سمت راست کرده و از منوی مربوطه گزینه set breakpoint on every call to Req.. را انتخاب کنید با انتخاب این گزینه باعث میشود هرکجا که این تابع دربرنامه صدا شود ollydbg برنامه را متوقف کرده و به ما اطلاع دهد.



شکل ۱۳

بعد از انجام کارهای بالا پنجره Intermodular را ببنید حال برنامه را با F9 اجرا کنید تا به اولین نقطه توقف برسید اولین نقطه در آدرس 421A3C قرار دارد مانند شکل زیر:

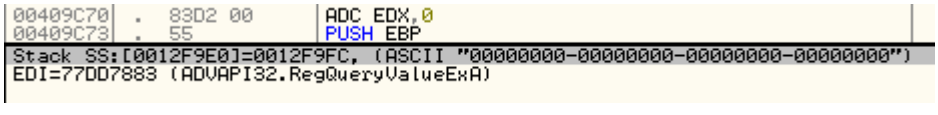


شکل ۱۴

خب حال در قسمت registers به ثبات EDI توجه کنید:

شکل ۱۵

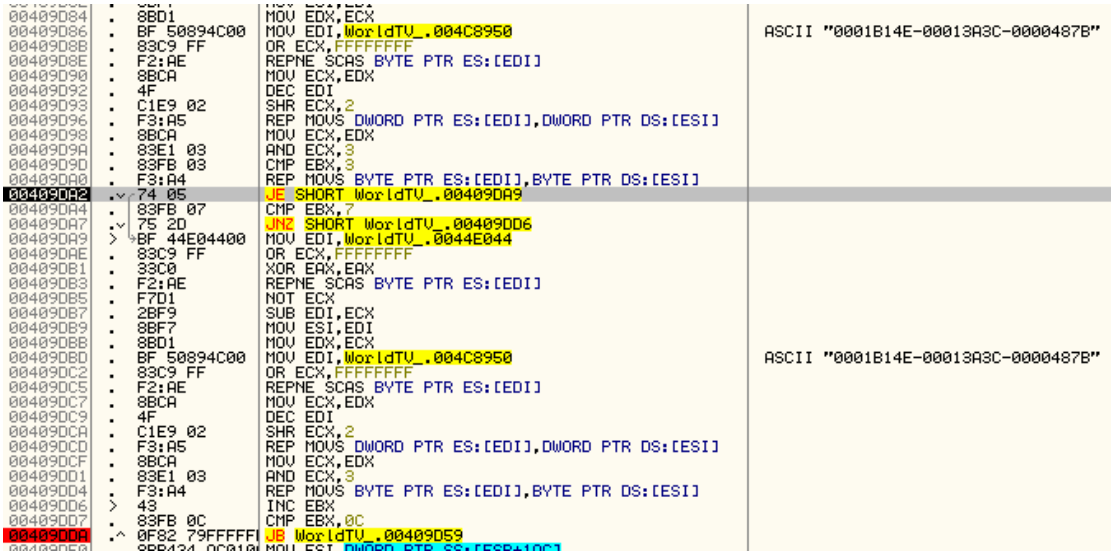
یک مقدار ASCII در آن هست بنام Recordings اینها تمام entryهای ریجستری مربوط به برنامه میباشد چندین بار کلید F9 را فشار دهید تا برنامه اجرا شود و آیکون آن را بتوانید مشاهده کنید و ثبات EAX مقدار Regcode را نشان دهد. بعد از آنکه متوجه این مقدار شدید برنامه را با کلید F8 آنقدر اجرا کنید. بعد از چند بار اجرای خط به خط بـه آدرسِ409BF3 میرسید در پنجره Registers میتوانید مقدار ESI را ببینید در واقع مقداری که شما در برنامه وارد کرده اید حال بار دیگر کلید F8 را زده اینبار میبینیم که مقدار 00000000-00000000-00000000-00000000 در ثبات EDIقرار میگیرد



شکل ۱۶

پس میتوان حدس زد که شماره سریال برنامه باید بدین صورت باشد خب دوباره کلید F8 را زده تا به آدرسِ 409D59 برسید باز کلید F8 را فشار دهید تا دوباره به همین آدرس برسیم خوب پس معلوم میشود که ما در یک حلقه هستم پـس میتـوانیم در آدرس 409DDA (در واقع جایی که شرط حلقه چک میشود) یکبار کلید F2 را فشار داده و بعد از آن برنامه را بـا کلیـد F9 دنبال کنیم چون ما در یک حلقه هستیم هرگاه که برنامه به این شرط میرسد متوقف شده و کنترل برنامه در دسـت مـا خواهـد بود. هنگامی که در حلقه هستیم و در حال زدن کلید F9 حتماً متوجه خواهید شد که در آدرس 409D86 یک شماره در حال پر شدن هست خب حال اینکار آنقدر ادامه دهید تا ۲۲ کاراکتر از ۲۴ کاراکتری کـه در حـال پـر شـدن در ثبـات EDI هـست

کامل شود هنگامی که ۲۲ کاراکتر کامل شد از این به بعد برنامه رو با کلید F8 دنبال کنید تـا بـه آدرس 409DDA برسـید حال شما یک شماره سریال دارید و میتوانید آنرا وارد کنید در واقع این حلقه بوسیله یک تایمر کنترل میشود شما میتوانید شماره سریالهای دیگر هم بدست بیاورید برای بدست آوردن شماره های دیگر کافیست F9 را نگه دارید تا خود متوجه قضیه بشوید:



شکل ۱۷

خب این کرکینگ دیگر تمام شد.

حال بیاییم کاری کنیم که هر شماره سریالی که دلمان میخواهد بدهیم و برنامه بدون هیچ مشکلی کار کند:
فرض کنیم شما شماره سریال بدست آورده را وارد کرده اید دوبـاره برنامـه را بـا olldbg بـاز کـرده و در آدرس 409DDA دوباره یک F2 میزنیم و دوباره برنامه را مانند توضیحات بالا با F9 آنقدر اجرا کرده تا به اواخر برنامه برسیم حـال برنامـه را بـا F8 دنبال کرده تا به آدرس409DF4 برسیم در اینجا یک شرط وجود دارد که هنگامی اجرا میشود که مقـدار شـماره سـریال موجود در رجیستری برابر با صفر باشد خب حال دوبار برروی آن کلیک کرده و مقدار آن را با NOP (دستورالعمل NOP بـه معنی No Operation یعنی هیچکاری انجام نشود میباشد) عوض کنید حال در آدرس 409DF8 یک پرش دیگـر وجـود دارد که هنگامی انجام میشود که تمام شماره سریال چک شده باشد و درست باشد خب برای اینکه این پرش را از حالت شرطی خارج کنیم کافیست پرش JE را با پرش JMP جایگزین کنید خب حال هر چه شما وارد کنید برنامه قبول میکند.
برای روشنتر شدن موضوع به دو عکس زیر توجه کنید:

```
00409DE7  .  B8 50894C00  MOV EAX,WorldTV_.004C8950
00409DEC  >  8A10         MOV DL,BYTE PTR DS:[EAX]
00409DEE  .  8A1E         MOV BL,BYTE PTR DS:[ESI]
00409DF0  .  8ACA         MOV CL,DL
00409DF2  .  3AD3         CMP DL,BL
00409DF4  .v 75 1E        JNZ SHORT WorldTV_.00409E14
00409DF6  .  84C9         TEST CL,CL
00409DF8  .v 74 16        JE SHORT WorldTV_.00409E10
00409DFA  .  8A50 01      MOV DL,BYTE PTR DS:[EAX+1]
00409DFD  .  8A5E 01      MOV BL,BYTE PTR DS:[ESI+1]
00409E00  .  8ACA         MOV CL,DL
00409E02  .  3AD3         CMP DL,BL
00409E04  .v 75 0E        JNZ SHORT WorldTV_.00409E14
00409E06  .  83C0 02      ADD EAX,2
00409E09  .  83C6 02      ADD ESI,2
00409E0C  .  84C9         TEST CL,CL
00409E0E  .^ 75 DC        JNZ SHORT WorldTV_.00409DEC
```

شکل ۱۸ قبل از کرک شدن

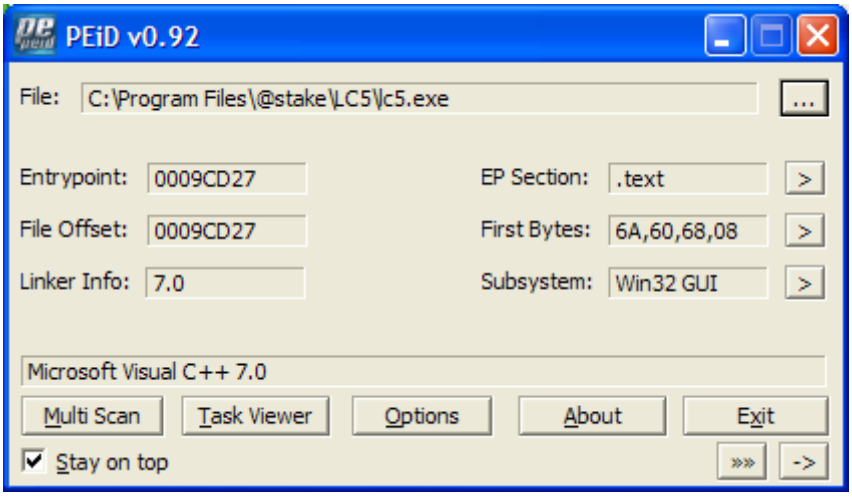
```
00409DF2  .  3AD3         CMP DL,BL
00409DF4  .  90           NOP
00409DF5  .  90           NOP
00409DF6  .  84C9         TEST CL,CL
00409DF8  .v EB 16        JMP SHORT WorldTV_.00409E10
00409DFA  .  8A50 01      MOV DL,BYTE PTR DS:[EAX+1]
00409DFD  .  8A5E 01      MOV BL,BYTE PTR DS:[ESI+1]
00409E00  .  8ACA         MOV CL,DL
00409E02  .  3AD3         CMP DL,BL
00409E04  .v 75 0E        JNZ SHORT WorldTV_.00409E14
00409E06  .  83C0 02      ADD EAX,2
00409E09  .  83C6 02      ADD ESI,2
00409E0C  .  84C9         TEST CL,CL
00409E0E  .^ 75 DC        JNZ SHORT WorldTV_.00409DEC
00409E10  >  33C0         XOR EAX,EAX
```

شکل ۱۹بعد از کرک شدن

پایان

آموزش کرکینگ با ollydbg قسمت دوم
تاریخ : فروردین ۱۳۸۵ April 2006
نوع حمله Internal Keygen and Patching
نرم افزار هدف : LC 5 ضمیمه شده
نرم افزارهای مورد استفاده :ollydbg 1.10 , DUP 2.0,PEid 0.92
سطح : مبتدی

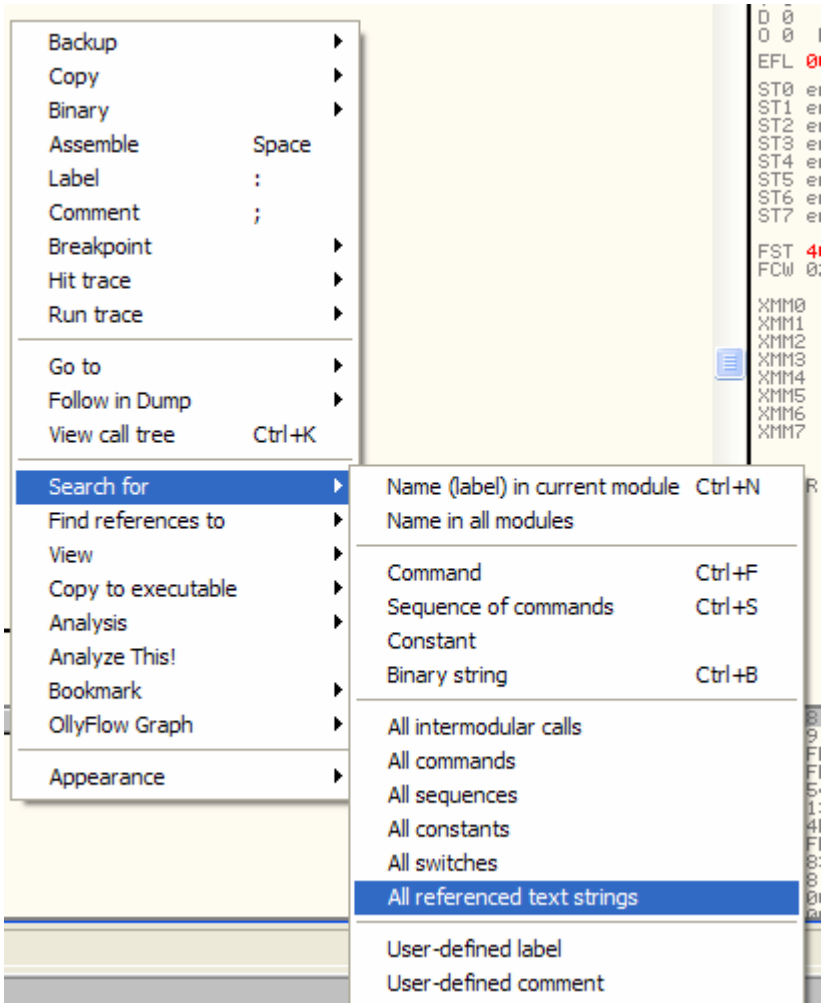
خب اول نرم افزار رو نصب میکنیم.
بعد از نصب همیشه اولین کاری که باید بکنیم این هست که اول بفهمیم برنامه رو با چی نوشتن و با چی پک شده برای اینکار میتونید از نرم افزارهایی مانند PEid و یا RDG Packer Detector استفاده کنید من اینجا از PEid استفاده کردم.
برنامه PEid رو باز کرده و فایل LC5.exe رو بندازید توش و یا از دکمه ...(open) برای تستِ فایل استفاده کنید



شکل ۱

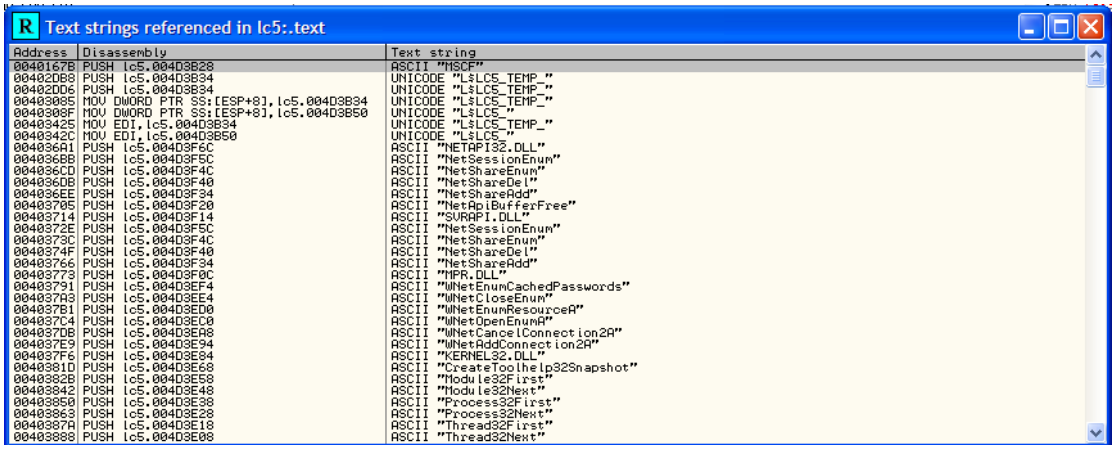
خب حال برنامه رو اجرا مکنیم یک صفحه nag باز شده و سه گزینه به شما نشان داده میشود خب طبق معمول گزینه register را انتخاب میکنیم یک دایلوگ دیگر باز شده و از ما شماره سریال میخواهد خب مثل همیشه ما سریال نامبر اصلی برنامه را نداریم! پس یک عدد مثلاً ۱۲۳۳۴۴ میزنیم و روی گزینه ok کلیک میکنیم تا ببینیم چه پیغامی نمایش داده میشود.
یک message box باز شده و حاوی متن زیر میباشد:
You have entered an invalid code. Please try again.

این متن رو یادداشت میکنیم و برنامه رو میبندیم.
حال ollydbg را باز کرده و فایل LC5.exe را در آن باز میکنیم.دکمه سمت راست موس را زده و گزینـه search for و بعد از آن گزینه all referenced text strings را انتخاب میکنیم:



شکل ۲

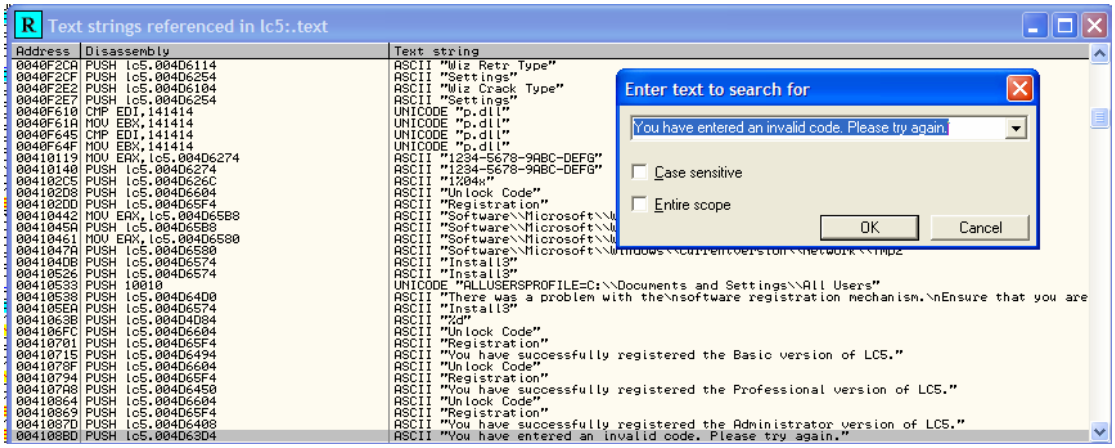
دایلوگ Text Strings Referenced باز شده نوار پیماش را به بالا برده و بر روی اولین گزینه کلیک کنید مانند شکل زیر:



شکل ۳

توجه کنید که حتماً باید اینکار را انجام بدهید بدین دلیل که ollydbg بصورت بالا به پایین عمل جستجو را انجام میدهد و از جایی این عمل انجام میگیرد که شما آنجا را انتخاب کرده باشید.

خب حال در این دایلوگ دکمه سمت راست را زده و گزینه Search for Text را انتخاب کرده و متن را که یادداشت نمودید در آن وارد کنید مانند شکل زیر:



شکل ۴

خب هنگامی که بر روی دکمه ok کلیک کنید ollydbg شما را بر روی رشته مورد نظر برده یعنی به آدرس 4108BD .
خب حال برروی آن آدرس دابل کلیک کنید شما بر روی آن آدرس قرار میگیرید کمی نوار پمایش را بالاتر ببرید متوجه خواهید این نرم افزار سه نسخه administrator ,professional ,basic دارد خب ما میخواهیم این برنامه را به نسخه administrator تبدیل کنیم خب با کمی دقت به قسمتی که که برنامه بصورت administrator ثبت میشود در آدرس 41082A متوجه دستور TEST EAX,EAX میشویم این دستور به این معنا میباشد که یک مقداری در حال مقایسه میباشد برای تفهیم بیشتر به pseudo code زیر توجه کنید :

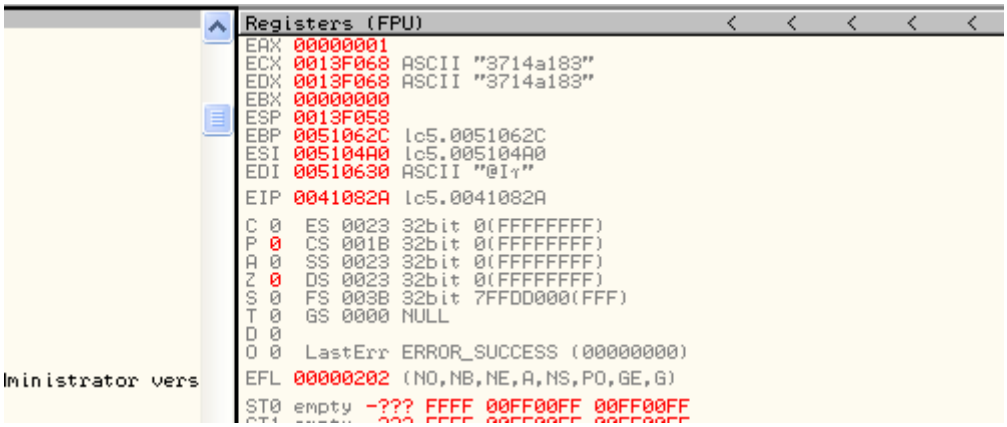
```
If user s/n = original s/n then
        Save user s/n
        Goto line n if the user S/N is not matched
End if
```

خب کد disassembly این کد مشابه زیر خواهد بود

```
TEST EAX,EAX
Push EBX
JNZ line n
```

در خط اول کد disassembly مقدار داده شده با مقدار مرجع چک میشود
در خط دوم مقدار پارامتر اول ذخیره میشود
در خط سوم اگر مقدار پارامتر اول برابر با شماره سریال اصلی نشد به خط مورد نظر پرش میکند.

خب حال با این توضیح بر روی آدرس41082A رفته و دکمه F2 را میزنیم و بعد برنامه را با دکمه F9 اجرا میکنیم وقتی برنامه اجرا شد برروی دکمه register کلیک کرده و دوباره یک شماره سریال زده و برروی دکمه ok کلیک کرده اینبار ollydbg در آدرسی که ما breakpoint گذاشته ایم توقف کرده حال یکبار دکمه F8 را زده و اینبار به پنجره registers توجه کنید مانند شکل زیر:



شکل ۵

در ثباتهای ECX و EDX متوجه مقداری خواهید شد(توجه کنید این مقدار در کامپیوتر شما متفاوت خواهد بود) این مقدار را یادداشت کرده و در textbox مربوط به رجیستر کردن برنامه وارد نمایید حال برروی دکمه ok کلیک کرده یک دایلوگ جدید بازشده و رجیستر شدن برنامه را به شما تبریک میگویید.

بسیار آسان بود؟

خب تا اینجا دیدیم شماره سریال چگونه ساخته میشود حال میخواهیم کاری کنیم که بتوانیم این شماره سریال را به یک کاربر معمولی نشان داده و به او بگوییم که این شماره سریال برنامه شما میباشد.

برای اینکار ابتدا از start گزینه run را انتخاب کرده و regedit را تایپ کرده تا وارد محیط registry windows شویم در آنجا به آدرس :

HKEY_CURRENT_USER\Software\@stake\LC5\Registration

رفته برروی گزینه Unlock Code دابل کلیک کرده و محتوی آنرا پاک کرده و گزینه OK را زده و از Regedit خارج شوید اینکار را به این دلیل انجام میدهیم که میخواهیم بر روی هدفمان دوباره کار کرده و در واقع یک internal keygen بر اساس آن بسازیم.البته اینگونه keygen ساختن رایج نبوده و کاری حرفه ای نمیباشد ولی برای یادگیری مقدماتی تمرین خوبی خواهد بود. به همین دلیل برنامه را دوباره به حالت unregistered درآورده‌ایم

خب قبل از شروع کار ما باید اهداف خود را مشخص کنیم:

۱. کد فعال سازی کجا ساخته میشود و در کجا ذخیره میشود
۲. راهی برای نشان دادن این کد باید پیدا کرد
۳. باید به کاربر دقیقاً اطلاع دهیم که این کد ، کد فعالسازی برنامه میباشد

ما همچنین میدانیم که کد فعال سازی در آدرس 41082A ساخته شده و در ثباتهای ECX,EDX ذخیره میگردد (قدم اول)

قدم بعدی نشان دادن این کد به کاربر میباشد.

خب دوباره برنامه را در ollydbg ریستارت کرده و در آدرس 41082A کلید F2 را زده و دوباره برنامه را با کلید F9 اجرا کنید.

در برنامه دوباره کلید register را انتخاب و یک شماره اشتباه وارد کنید تا ollydbg بر روی آدرس مورد نظر توقف کند.

خب دقیقاً به پنجره registers دقت کنید مخصوصاً ثباتهای ECX و EDX حال برنامه را با کلید F8 دنبال کنید.

میبینید که هنگامی که به آدرس 4108B2 میرسیم یعنی دستور

XOR ECX,ECX

مقدار ECX صفر خواهد شد(برای اطلاع بیشتر در مورد عملگر XOR به یک کتاب برنامه نویسی مراجعه کنید)

در آدرس 4108B4 داریم:

MOV DWORD PTR SS:[ESP+14],ECX

که این آدرس هم باعث از بین رفتن مقدار EDX خواهد شد

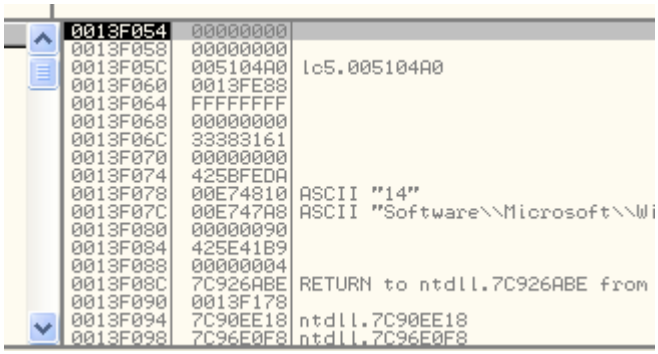
در آدرس 4108B8 داریم

PUSH EBX

که ثبات EBX را در پشته قرار میدهد ثبات EBX حاوی یک مقدار میباشد.

پشته جایی از حافظه میباشد که مقادیری که میخواهیم ذخیره کنیم در آن قرار میگیرد (برای اطلاع بیشتر از کارکرد پشته به مستندات اسمبلی مراجعه کنید)

برای دیدن پشته در ollydbg به پنجره سمت چپ پایین توجه کنید مانند شکل زیر:



شکل ۶

در آدرس 4108B9 داریم:

MOV DWORD PTR SS:[ESP+1C],ECX

که بازهم این مقدار باعث از بین رفتن مقدار ثبات EDX میشود.

و در آخر در آدرس 4108C2 داریم:

MOV BYTE PTR SS:[ESP+24],CL

که این دستور هم باعث از بین رفتن ثباتها EDX ما میشود.

به این نکته توجه داشته باشید که باید کاری کنیم که بعد از اینکه شماره سریال اصلی برنامه در ثباتهای ECX و یا EDX ذخیره شد باید یکی از این ثباتها این مقدار را نگه داشته تا ما بتوانیم آنرا به کاربر نهایی نشان دهیم.

خب حال در آدرسهایی که باعث از بین رفتن مقدار ثباتهای ما میشود رفته و دستورالعملهای آنها را با دستورالعمل NOP عوض میکنیم.

یعنی در آدرسهای:

- 4108B4
- 4108B9
- 4108C2

برای یادآوری برای عوض کردن دستورالعملها بر روی آدرسهای مورد نظر رفته از سمت راست برروی ستون سوم دابل کلیک کرده در دایلوگی که باز میشود دستورالعمل جاری را پاک کرده و دستورالعمل NOP را جایگزین میکنیم.

توجه داشته باشید که گزینه Fill with NOP's در این دایلوگ حتماً انتخاب شده باشد والا برنامه دچار مشکل میشود.

کمی توضیح در رابطه با جایگزین کردن دستورالعملهای دلخواه:

شاید از آموزش قبل متوجه شده باشید که هنگامی که یک دستورالعمل را برای مثال دستورالعمل :

MOV DWORD PTR SS:[ESP+1C],ECX

این دستور هنگامی که به دستورات ماشین یعنی opcode برگردانده میشود به صورت زیر درمی‌آید:

894C24 1C

خب طول این opcode ۸ کاراکتر میباشد.

خب حال فرض کنید که ما میخواهیم این دستورالعمل را با دستورالعمل NOP جایگزین کنیم، opcode مربوط به NOP

90

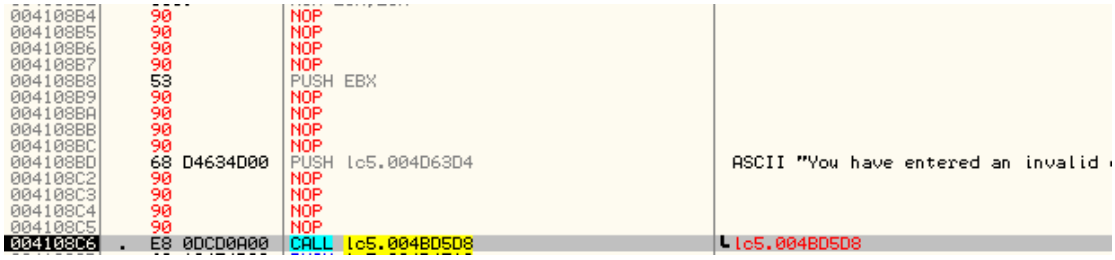
میباشد و ۲ کاراکتر طول دارد

و این را هم باید بدانیم هنگامی که در حال ویرایش opcodeهای یک برنامه هستیم باید به همان اندازه از opcodeها را تغییر دهیم (از نظر طول) و نه بیشتر و نه کمتر والا برنامه خراب شده و دیگر کار نمیکند.

خب پس با این اوصاف برای تغییر دستورالعمل بالا با دستورالعمل NOP ما باید ۴ دستورالعمل NOP را جایگزین کنیم یعنی

90909090

خب پس از تغییر دادن آدرسهایی که در بالا به آنها اشاره شد با NOP مانند شکل زیر خواهیم داشت:



شکل ۷

خب حال ما تا اینجا موفق شدیم که مقدار ثبات EDX را دست نخورده نگه داریم

خب برنامه را دقیقاً تا آدرس 4108C5 را با F8 اجرا کنید

حال باید مقدار EDX را به کاربر نشان دهیم چگونه؟

به آدرس 4108BD توجه کنید دستور:

PUSH 4D63D4

و مقدار این دستور برابر است با :

You have entered an invalid code. Please try again

حال باید دستورالعمل آدرس 4108BD را عوض کرده و بجای اینکه این دستورالعمل مقدار برنامه را نشان دهد مقدار ثبات EDX را نشان دهد برای اینکار برروی آدرس 4108BD ستون سوم دابل کلیک کرده و دستورالعمل زیر را جایگزین کنید:

PUSH EDX

حال بار دیگر F8 را زده و متوجه message box زیر خواهیم شد:

شکل ۸

خب ما موفق شدیم که قدم دوم را هم برداریم حال تنها کاری که مانده اینست که به کاربر معمولی اعلام کنیم این شماره سریال شما میباشد آنرا وارد کنید.
بهترین جایی که میتوان اینکار را انجام داد در messagebox قسمت title میباشد.

حال برنامه را ذخیره میکنیم بر روی ollydbg دکمه سمت راست موس را زده و گزینه copy to Executable را انتخاب کرده و بعد از آن all modifications و بعد از آن دکمه Copy all و بعد از آن دوباره سمت راست موس را در دایلـوگ جدید زده و گزینه Save File را انتخاب کنید.
توجه کنید اسم فایل را متفاوت از اسم فایل اصلی انتخاب کنید.

کمی درباره تابع MessageBox
– این تابع یک تابع داخلی ویندوز بوده و از آن برای نشان دادن پیغامهای ویندوز در هر برنامه‌ای میتوان استفاده کرد
–این تابع مانند بقیه توابع استاندار پارامترهایی دارد که بعضی از آنها اختیاری و بعضی از آنها اجباری میباشند
–پارامترهای این تابع عبارتند از:

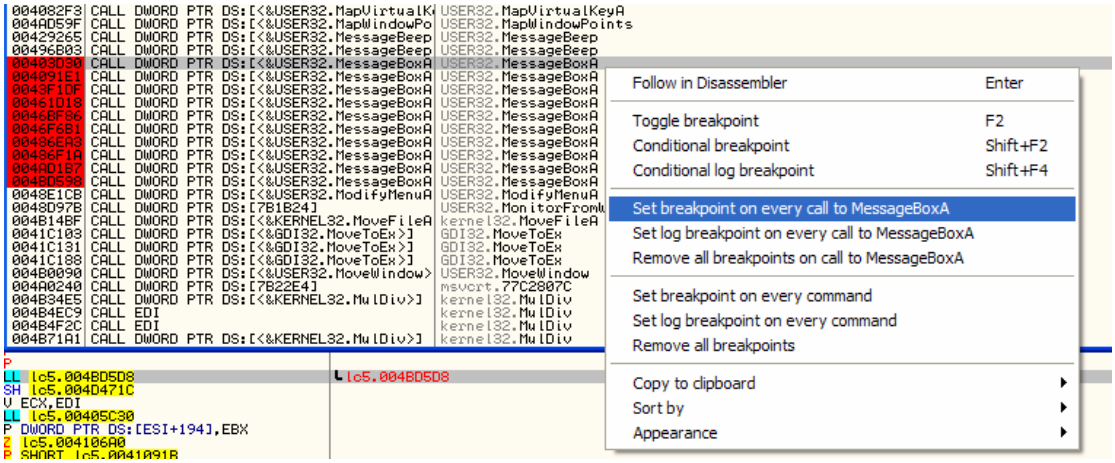
HWND hwnd
–شماره ای که مشخص کننده دایلوگی میباشد که تابع را صدا زده.

LPCTSTR lpText
–آدرس متنی که در وسط نشان داده خواهد شد

LPCTSTR lpCaption
–آدرس متنی که در بالای messagebox نشان داده خواهد شد

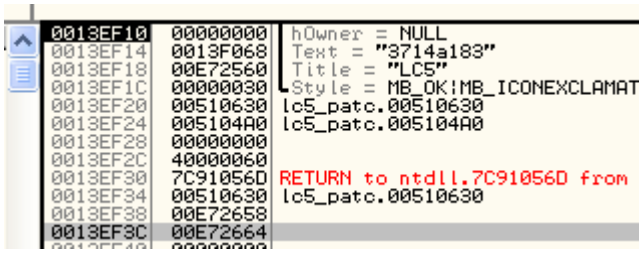
UNIT uType
–مدل messagebox

خب حال فایلی را که تغییرات آنرا ذخیره کرده ایم در ollydbg باز کرده دکمه سمت راسـت مـوس را زده گزینـه search for را انتخاب و بعد از آن گزینه all intermodular calls را انتخاب کرده و در دایلوگ جدید که بـاز شـده تایـپ کنیـد messagebox تا بر روی این توابع قرار گیرید بعد برروی یکی از این توابع دکمه سمت را زده و گزینه set breakpoint on every call to messageboxa را انتخاب کنید مانند شکل زیر:



شکل ۹

خب حال این پنجره را بسته و برنامه را با کلید F9 اجرا کنید و دوباره برروی گزینه register کلیک کرده و دوباره یک شماره سریال اشتباه وارد نمایید ollydbg برنامه رو متوقف کرده و در آدرس 48D598 برنامه متوقف شده حـال بـه پنجـره پـشته توجه کنید:



شکل ۱۰

میبینید که مقدار title برابر است با LC5 خب حال ما باید این رشته را با رشته ای دلخواه تغییر دهیم.

خب چون ما نمیتوانیم هرآنچه که دلمان میخواهد در برنامه بنویسیم باید جایی را جستجو کرده که کدهای کلیدی را در آن قسمت نبوده و آن کدهای قدیمی را با کدهای خود جایگزین کرده تا به مقصودمان برسیم.برای اینکار بهترین راه اینست که برنامه را چند خط بالاتر از آدرسی که الان در آن قرار داریم دنبال کنیم تا ببینیم آیا به نکته‌ای برخورد میکنیم یا نه!

به نکته زیر توجه کنید:

اگر از قسمت قبل به یاد داشته باشید ما آدرس 4108BD را با دستور PUSH EDX خود override کردیم نتیجه میگیریم که یک رشته به مضمون :

You have entered an invalid code. Please try again

بلا استفاده باقی مانده است که این رشته دقیقاً در آدرس lc5.004D63D4 قرار دارد

برنامه را ریستارت کنید. به آدرس 48D598 رفته و کمی نوار پیماش را به بالا میکشیم.

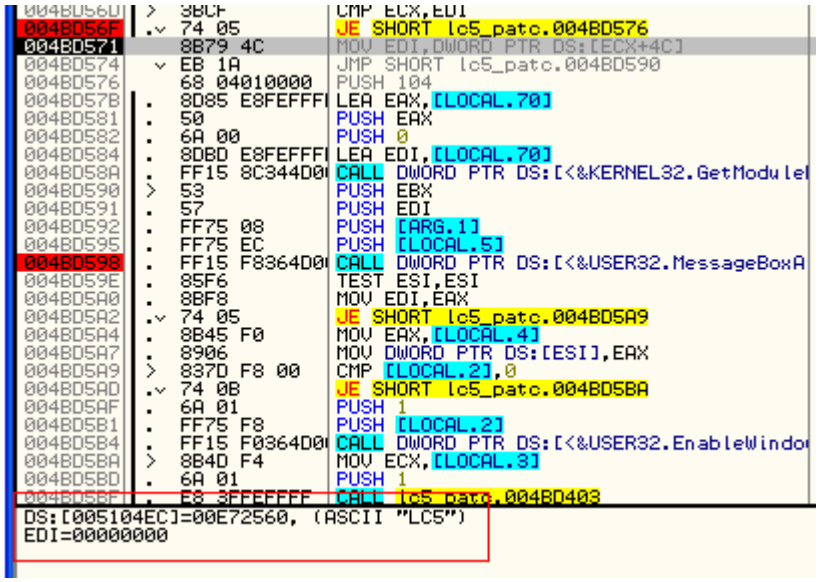
با کمی دقت توجه‌مان به آدرس 4BD56F جلب میشود در واقع در این آدرس یک پرش وجود دارد که هیچگاه اجرا نمیشود (یا حداقل تا این زمان اجرا نشده است!)

در این آدرس بوسیله F2 یک breakpoint میگذاریم برنامه را دوباره با F9 اجرا کرده تا به آدرس 48D56F برسیم از این به بعد برنامه را با F8 دنبال کرده

در آدرس 4BD571 دستورالعمل زیر قرار دارد:

MOV EDI,DWORD PTR DS:[ECX+4C]

با کمی دقت درمیابیم که دراین آدرس مقدار EDI برابر با LC5 میشود مانند شکل زیر:



شکل ۱۱

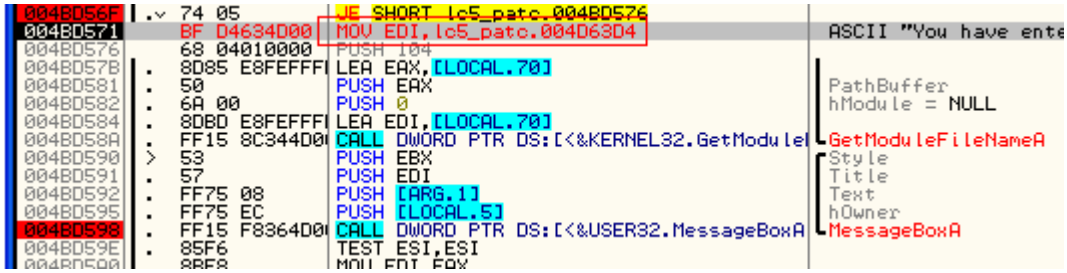
در واقع title برای messagebox در اینجا شکل میگیرد!

خب ما اینجا میتوانیم title را تغییر دهیم اما چگونه؟

خب اگر به نکته ای که اخیراً به شما گفتم توجه کرده باشید خواهید فهمید که میتوان مقدار پارامتر دوم عملگر MOV را به آدرس 4D63D4 تغییر بدهید یعنی مانند زیر:

MOV EDI, 4D63D4

مانند شکل زیر:



شکل ۱۲

خب با تغییر این خط (به شکل ۱۱ توجه کنید) میبینید که دستور

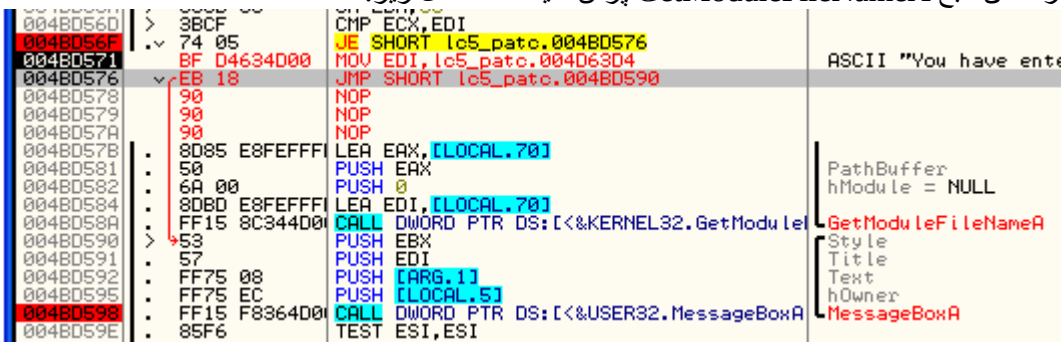
JMP SHORT lc5modif.004BD590

پاک شده است به دلیلی که قبلاً در بالا به آن اشاره شده است

خب حالا اگر توجه کرده باشید با این کار ما باعث شده است که آدرسهای 4BD576 تا 4BD58A اجرا شود و درضمن در این بین در آدرس 4BD584 یک مقداری در ثبات EDI جایگزین میشود درواقعی ثباتی که ما در آنجا میخواهیم title مورد نظر خود را در آن ذخیره کرده و به کاربرنشان دهیم.برای اینکه این مشکل را حل کنیم بر روی آدرس 4BD576 رفته و دابل کلیک کرده و کد زیر را جایگزین کد اصلی میکنیم:

JMP SHORT 4BD590

امکان دارد آدرس پرش با آدرس پرش شما متفاوت باشد بدین خاطر بدین صورت بیان میکنم به آدرس بعد اجرا شدن تابع GetModuleFileNameA پرش کنید مانند شکل زیر:



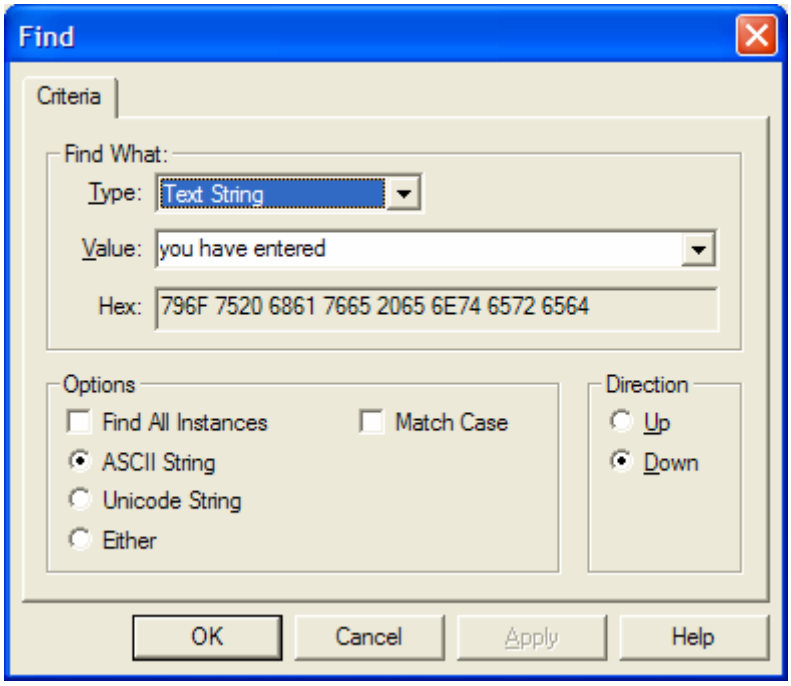
شکل ۱۳

حال برنامه را F9 دنبال کرده تا پیغام زیر را مشاهده کنید:

شکل ۱۴

خب میبینید که ما موفق شدیم title را تغییر دهیم ولی اینکار یک اشکال کوچک دارد و آنهم اینکه title مورد نظر چیزدیگریست برای رفع این مشکل برنامه را که تا اینجا تغییر داده ایم به همان روشی که قبلاً گفتم ذخیره کرده و ollydbg را میبندیم.

حال hexworkshop را باز کرده و برنامه‌ای را که تغییرات آخر را بر روی انجام داده ایم در آن باز کرده وقتی برنامه ما در hexworkshop باز شد دکمه ctrl+f را زده تا دایلوگ جستجو باز شده و مانند شکل زیر عمل کنید:



شکل ۱۵

خب حال شما دقیقاً درجایی قرار دارید که متن را باید تغییر دهیم فقط به این نکته توجه کنید کـه بـه تعـداد کاراکترهـایی کـه موجود میباشد میتوانید کلمات را تغییر دهید

```
6172 740A 204C 4335 2066 6F72 206D 6F72 6520 696E 666F 2E00 596F  art. LC5 for more info..Yo
7520 6861 7665 2065 6E74 6572 6564 2061 6E20 696E 7661 6C69 6420  u have entered an invalid
636F 6465 2E20 506C 6561 7365 2074 7279 2061 6761 696E 2E00 596F  code. Please try again..Yo
7520 6861 7665 2073 7563 6365 7373 6675 6C6C 7920 7265 6769 7374  u have successfully regist
```

شکل ۱۶

خب حال که تمام تغییرات اعمال شد. از منوی file گزینه saveرا انتخاب کرده وفایل را save کنید.

ساخت patch:

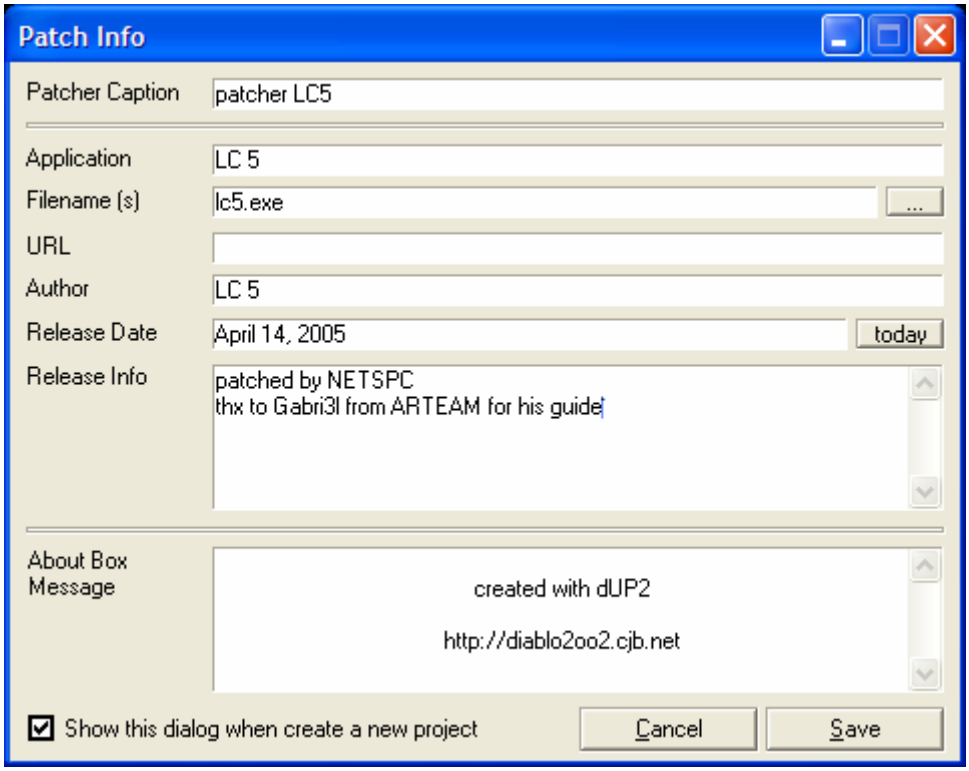
کار ساخت internal keygen تمام شد حال فقط یک موضوع باقیست ساخت patch .

چرا باید patch بسازیم؟

یکی از دلایل ساخت patch به این دلیل میباشد که، چون حجم فایلی که ما تغییرات را در آن اعمال میکنیم امکان دارد زیاد باشد برای مثال LC5.exe در حدود ۳ مگابایت میباشد و انتقال این حجم فایل با مشکل روبرو هست به همین دلیل میتـوان از طریق ساخت یک patch که شاید حجمی در حدود ۳۰ کیلوبایت دارد همان کار را انجام داد.

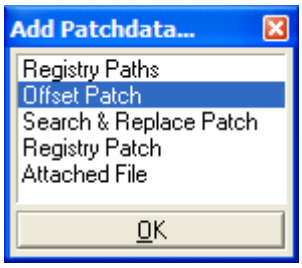
یکی دیگر از دلایل ساخت patch ناآشنا بودن کاربران معمولی با مباحث مهندسی معکوس و کرکینگ میباشد.

برای ساخت patch از دو طریق میتوان عمل کرد برنامه نویسی یک patch جدید و یا اسـتفاده از برنامـه هـای آمـاده بـرای ساخت patch من در اینجا از برنامه Diablo 2oo2 نسخه 2.12 استفاده میکنم.خب قدم به قدم کارهـای زیـر را انجـام میدهیم برنامه Diablo را اجرا کرده گزینه new project را انتخاب کرده و محتویات آنرا مانند شکل زیر پر کنید:



شکل ۱۷

و بعد از آن گزینه save را انتخاب کرده در صفحه اصلی برنامه دکمه add را زده گزینه offset patch را انتخاب کنید



شکل ۱۸

در صفحه اصلی گزینه ای بنام offset patch اضافه میشود برروی آن دابل کلیک کنید :

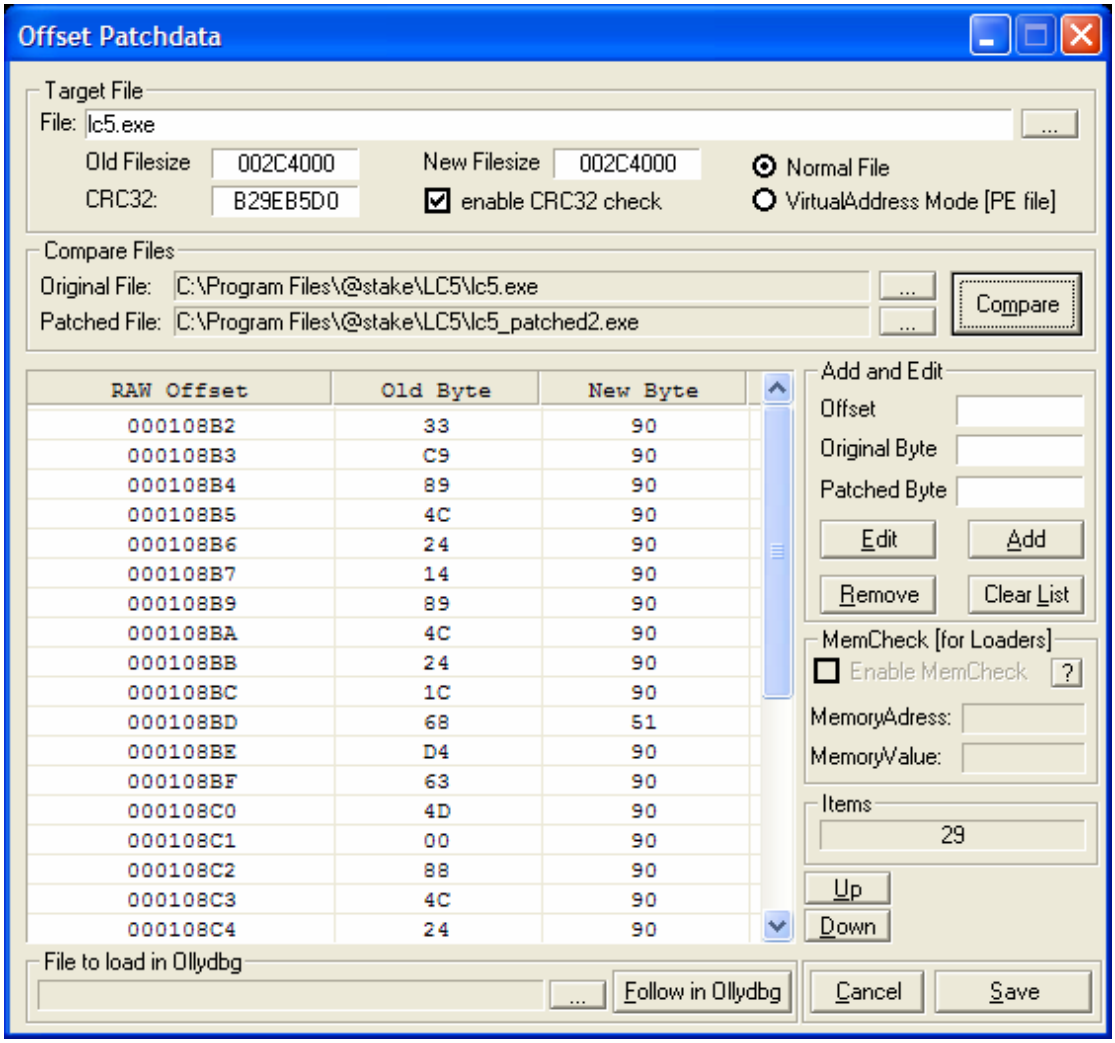
دایلوگی جدید باز شده در قسمت target file آدرس فایل اصلی دست نخورده را بدهید

در قسمت compare files:

در قسمت original file دوباره آدرس فایل اصلی را بدهید

در قسمت patched file آدرس فایلی را که تمامی تغییرات را در آن اعمال کرده اید.

بعد بر روی دکمه compare کلیک کرده مانند شکل زیر:



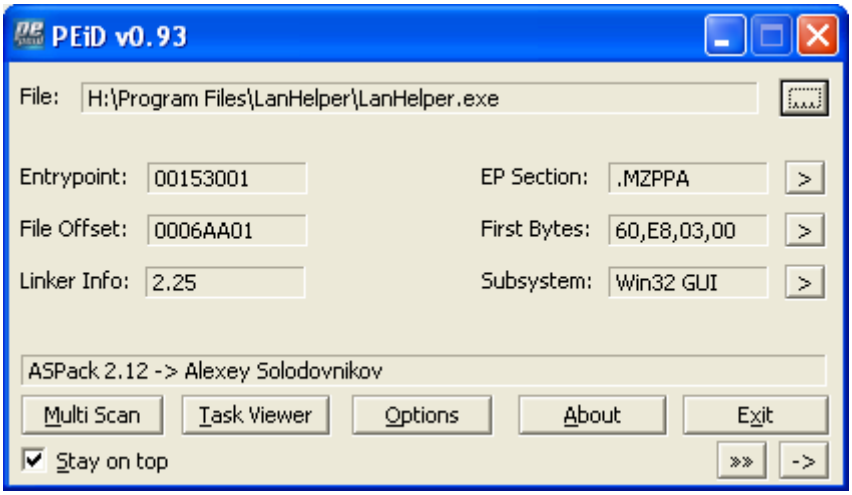
شکل ۱۹

حال دکمه save را زده و در صفحه اصلی هم دکمه create patchرا زده از شما یک مسیر برای ذخیره patch ساخته شده میخواهد مسیر را بدهید.

پایان

آموزش کرکینگ با ollydbg قسمت سوم
تاریخ : فروردین ۱۳۸۵ April 2006
نوع حمله unpacking Aspack 2.12 ,serial protection
نرم افزار هدف : Lan helper 1.40 ضمیمه شده
نرم افزارهای مورد استفاده :ollydbg 1.10 (hidedebugger ,ollydump) plugin , DUP 2.0,PEid 0.93 ,ImpRec 1.6
سطح : متوسط (در این مثال از ۲ کامپیوتر استفاده شده است ممکن است بعضی از آدرسها فرق کند)

خب اول نرم افزار رو نصب میکنیم.
بعد از نصب همیشه اولین کاری که باید بکنیم این هست که اول بفهمیم برنامه رو با چی نوشتن و با چی پک شده برای اینکار میتوانید از نرم افزارهایی مانند PEid و یا RDG Packer Detector استفاده کنید من اینجا از PEid استفاده کردم.
برنامه PEid رو باز کرده و فایل Lanhelper.exe رو بندازید توش و یا از دکمه ...(open) برای تستِ فایل استفاده کنید



شکل ۱

خب این اولین برنامه ای هست که بصورت pack شده میباشد و ما میخواهیم برروی آن کار کنیم.

کمی در رابطه با packerها:

منظور از اینکه فایل اجرایی pack شده است یعنی چه؟
اگر با نرم افزارهایی مانند winrar و یا winzip آشنایی داشته باشید میدانید که این نرم افزارها فایلها را بوسیله الگوریتم خاص خودشان بصورت فشرده درمی آورند که به این روش pack کردن میگویند.در واقع در فایلهای اجرایی هم این روش مرسوم هست البته به دو دلیل:

۱. کم کردن حجم فایل اجرایی(فشرده کردن)
۲. سخت کردن و یا غیرممکن کردن دستکاری فایل اجرایی توسط روشهای مهندسی معکوس(کرک کردن).

خب حال این سوال پیش می آید که حال که فایل اجرایی بصورت pack شده درآمد چگونه اجرا میشود (در واقع چگونه unpack شده و اجرا میشود).
باز به ذکر مثال درمورد نرم افزارهایی مانند winrar میپردازم همانگونه که این نرم افزارها خود فایلها را فشرده میکنند خود همان نرم افزارها الگوریتم هایی را بابت باز کردن فایلها دراختیار دارند و آن فایلها را unpack میکنند.

این موضوع در رابطه با فایلهای اجرایی هم صدق میکند با این تفاوت که نرم افزار مورد نظر (در این مثال Aspack) که وظیفه pack کردن و unpack کردن برعهده اوست در خود فایل اجرایی قرار میگیرد. در واقع اینگونه نرم افزارها خود را به فایل اجرایی اصلی تزریق کرده و هنگامی که شما فایل موردنظرتان را اجرا میکنید اول فایل packer (در این مثال Aspack) اجرا میشود و بعد به نقطه ای دیگر در فایل اجرایی پرش کرده که آن نقطه ، نقطه آغاز برنامه اصلی ما میباشد که اصطلاحاً به آن نقطه OEP(Original Entry Point) میگویند. و برنامه اصلی را به حافظه بارگذاری میکند و آنرا اجرا میکند.

به نرم افزاری که وظیفه pack کردن و unpack کردن برعهده اوست و در فایل اجرایی تزریق شده است STUB میگویند. کاری که ما باید در مورد تمامی packer ها انجام بدهیم (البته بصورت دستی) اینست که منتظر بمانیم تا فایل در حافظه لود شود و بعد از آنکه فایل از حالت pack شده درآمد و در حافظه قرار گرفت ما OEP آنرا را پیدا کرده و فایل اجرایی لود شده در حافظه را برروی هارددیسک خود منتقل کنیم.که اصطلاحاً به این عمل dumping میگویند.

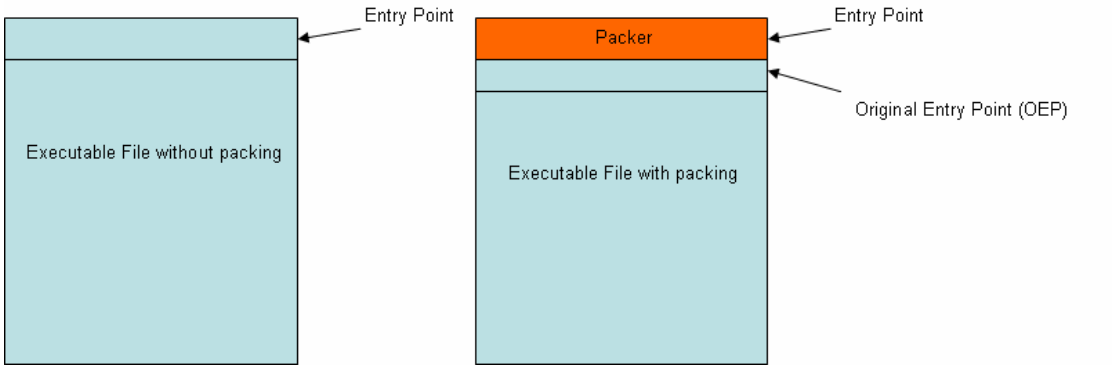
خب بنظر میرسد که کار تمام است ما توانستیم فایل اجرایی که pack شده بود را پیدا کرده و روی هارددیسک کپی کنیم.

ولی درواقع OEP که ما پیدا کرده ایم OEP فایل اجرایی قابل اجرا نمیباشد . فایل اجرایی دارای چندین section میباشد یکی از مهمترین sectionهای فایل اجرایی IAT(Import Address Table) نام دارد.بعد از عمل dumping ما باید این جدول را ترمیم کنیم.

کار IAT چیست؟

این جدول به برنامه اجرایی اجازه دسترسی به توابع API(Application Programming Language) را میدهد برای نمونه تابع Messagebox یک تابع API میباشد که خارج از برنامه اجرایی بوده. ولی برنامه اجرایی میتواند از آن برای نشان دادن پیغامهای خود استفاده کند.

هنگامی که برنامه اجرایی میخواهد از یک تابع API استفاده کند سیستم عامل ویندوز فایل کتابخانه ای مورد نظر که حاوی آن تابع میباشد را در حافظه لود میکند و آدرس آن تابع را به IAT برنامه مربوطه میدهد.



شکل ۲

برای شروع کار برنامه را اجرا میکنیم بعد از چند لحظه دایلوگی ظاهر میشود و به ما هشدار میدهد که فقط ۳۰ روز دیگر به اتمام برنامه مانده است در دایلوگ مورد نظر گزینه register را انتخاب کرده:

شکل ۳

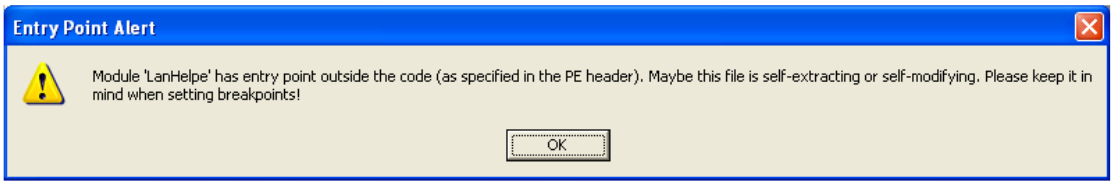
در دایلوگ ظاهر شده یک نام و یک شماره وارد میکنیم با پیغام خطایی مانند زیر مواجه میشویم:

شکل ۴

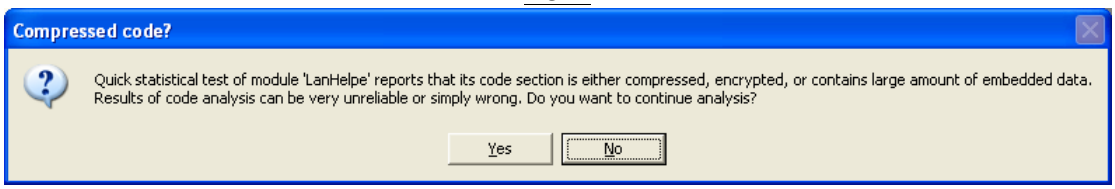
خب دوباره یک پیغام و یک messagebox (مانند دو مثال قبل)!
پیغام خطا را یادداشت کرده و برنامه را ببندید.

Unpacking:

Unpacking یکی از وقتگیر ترین کارها در مهندسی معکوس میباشد.اغلب شرکتهای نرم افزاری به این دلیل برنامه خود را pack میکنند که کاربران معمولی و متوسط کامپیوتر نتوانند برنامه های آنها را تغییر دهند.
بسیاری از برنامه های packing از متدهایی برای یافتن نرم افزارهای debugger استفاده میکنند اینکار بدین دلیل انجام میشود که کاربر نتواند برنامه مورد نظر را تغییر دهد برای اینکار در ollydbg چندین pulg-in نوشته شده است من در این مثال از hidedebugger استفاده میکنم.این plug-in را در دایرکتوری که ollydbg را نصب کرده اید کپی کنید که هنگامی که بر روی برنامه کار میکنیم Aspack متوجه وجود ollydbg نشود.
خب فایل lanhelper.exe را در ollydbg باز کنید:
خب در بدو ورود به ollydbg با دو پیغام مواجه میشویم:



شکل ۵

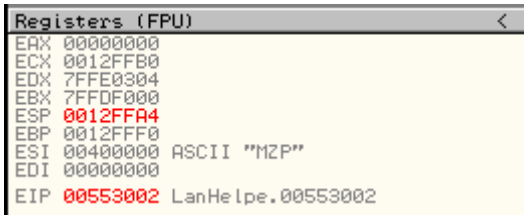


شکل ۶

این دوپیغام بدلیل وجود packer میباشد.
گزینه yes را انتخاب کرده تا ollydbg فایل را بازکند

شکل ۷

همانند شکل بالا شما در آدرس 553001 قرار خواهید گرفت.
خب برنامه را با کلید F7 اجرا کنید (البته میتوانید از step into استفاده کنید. اجرای برنامه بدین صورت باعث میشود که برنامه به هر زیر برنامه ای که میرسد به داخل آن را رفته و آنرا اجرا کند و کاربر را در جریان اجرا قرار دهد برای فهمیدن بیشتر به ادامه آموزش توجه کنید).
حال بعد از زدن کلید F7 به پنجره ثباتها توجه کنید:

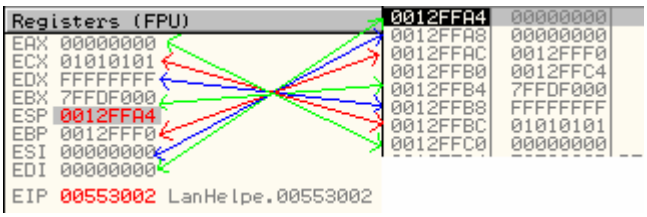


شکل ۸

ثباتها قسمتی از CPU بوده و بسته به نوع پردازنده میتوانند ۸ بیت ، ۱۶ بیت ، ۳۲ بیت و یا ۶۴ بیت باشند.
(فهمیدن طرز کار ثباتها و کاربرد آنها جزوی از کار یک کرکر میباشد)
ثباتها جایی هستند که برنامه میتواند مقادیر خود را در آن نگه داشته و بدون نیاز به حافظه اضافه محاسبات خود را در آن انجام دهد هرچه اندازه این ثبات بیشتر باشد مقادیر بیشتری را در خود جای داده و محاسبات پیچیده تری را انجام میدهد به همین دلیل امروزه پردازنده های ۶۴ بیتی رو به افزایش است.

در یک پردازنده ۳۲ بیت ما ۴ ثبات عمومی به نامهای EAX,EBX,ECX,EDX را داریم. این ثباتها بیشتر برای محاسبات در برنامه ها بکار برده میشود.ثباتهای دیگری مانند :
Index Register , Pointer Register در پردازنده موجود میباشد : ESI,EDI,EBP,EIP,ESP نام این ثباتها میباشد.
این ثباتها کارهای مختلفی انجام میدهند اما اکثر کار آنها در مورد اجرای برنامه ای هست که در حال اجراست (در واقع ثبات EIP شماره خط برنامه ای که باید اجرا شود را نگهداری میکند).
ESP نگه دارنده آدرس بالایی پشته میباشد (ثبات ESP مخفف Stack Pointer میباشد E مربوط به ۳۲ بیت بودن این ثبات هست)
برای اطلاعات بیشتر در مورد ثباتها میتوانید به آدرسهای زیر مراجعه کنید:
http://www.mujweb.cz/www/komsbomb/article/x86reg.htm
http://www.swansontec.com/sregisters.html

خب حال نگاهی به پینجره ثبات بیاندازید و بعد از آن به پنجره پشته نگاه کنید: (توجه کنید آدرس پشته شما با آدرس پشته ای که هم اکنون میبینید فرق میکند)



شکل ۹

میبینید که تمامی مقادیر ثباتها در پشته قرار گرفته است ولی بصورت کاملاً برعکس یعنی مقدار ثبات EDI در اول قرار گرفته است و مقدار ثبات EAX در آخر قرار گرفته است.این کار بوسیله تابع PUSHAD که چند لحظه پیش اجرا شد انجام میگیرد.
کاری که تابع انجام میدهد اینست که تمامی مقادیر ثباتها را در پشته ذخیره میکند.این تابع هنگامی بکار برده میشود که برنامه بخواهد مقادیر ثباتها را دستکاری کند ولی جایی برای نگه داری مقادیر قبلی ندارد.مقادیر اولیه ذخیره شده و به هنگام نیاز بازیابی میشود.تابع PSUHAD یک تابع قدرتمند و مفید برای تمامی Packerها میباشد.بوسیله این تابع تمامی مقادیر اولیه ثباتها در پشته قرار گرفته بعد STUB اجرا میشود و برنامه اصلی در حافظه لود میشود و دوباره مقادیری که در پشته ذخیره شده بودند به ثباتها برگردانده میشوند بدین معنا که اصلاً packer ی وجود نداشته است!
کاری که ما میخواهیم انجام دهیم اینست که ببینیم کجا مقادیر ذخیره شده در پشته ، به ثباتهای خود برگردانده میشود چرا که هنگامی که ما بتوانیم به این مهم دست بیابیم همانطور که در بالا توضیح داده شد ما OEP را پیدا کرده ایم.
تابعی که تمامی مقادیر ذخیره شده در پشته را به همان ثباتهای خود بازمیگرداند POPAD نام دارد.

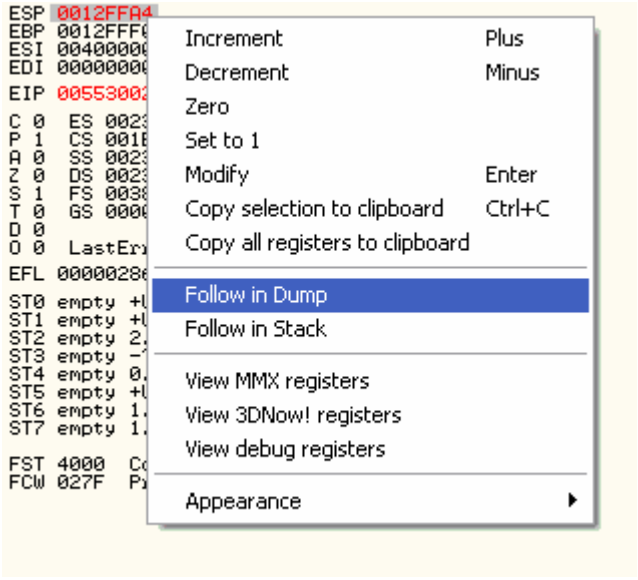
این را هم باید بدانیم مقادیر ثباتهایی که در پشته قرار گرفته اند بصورت معکوس از پشته خارج میشود مثال اگر ثبات EAX اولین ثباتی هست که مقادرش در پشته قرار میگیرد و ثبات EDI آخرین ثبات. در برگرداندن این مقادیر، ثبات EDI اولین برگشتی است و ثبات EAX آخرین(برای روشنتر شدن مطلب به شکل ۹ توجه کنید).

خب همانطور که در بالا گفته شد ما در این برنامه با ثبات ESP سروکار خواهیم داشت این هم به این دلیل میباشد که تمامی مقادیر ثباتها در پشته ذخیره میشود پس ما باید با ثباتی کار کنیم که رابطه مستقیم با پشته و کنترل آن دارد.

حال در ollydbg از پنجره ثبات بر روی مقداری که جلوی ESP نوشته شده است دکمه راست موس را زده و از منوی باز شده مانند شکل زیر گزینه :

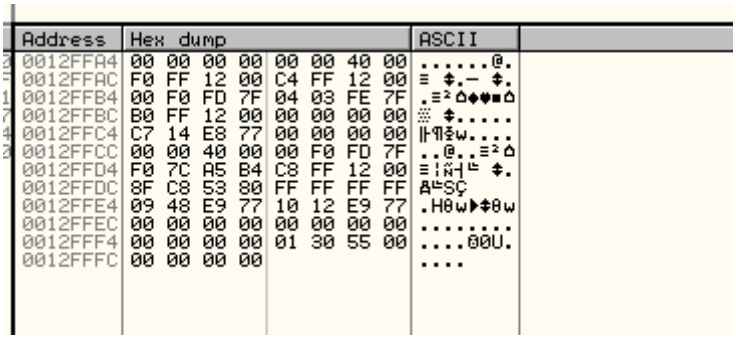
Follow in Dump

را انتخاب میکنیم:



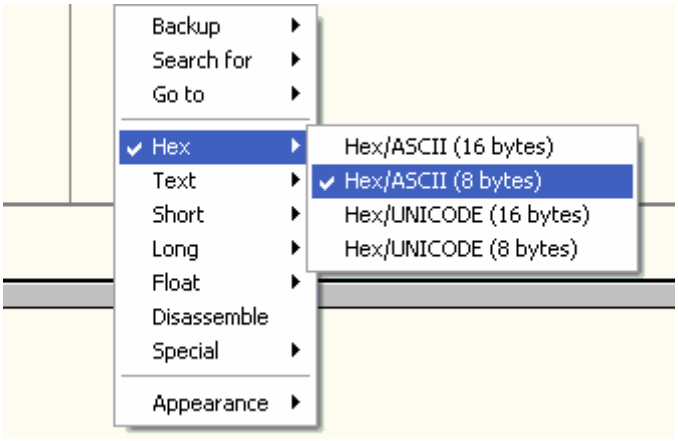
شکل ۱۰

حال به پنجره dump که در پایین صفحه قرار گرفته است توجه کنید:



شکل ۱۱

در این پنجره برای راحتی کار بر روی پنجره دکمه سمت راست موس را زده و از منوی ظاهر شده گزینه Hex - > Hex/ASCII(8 bytes) را انتخاب کنید:



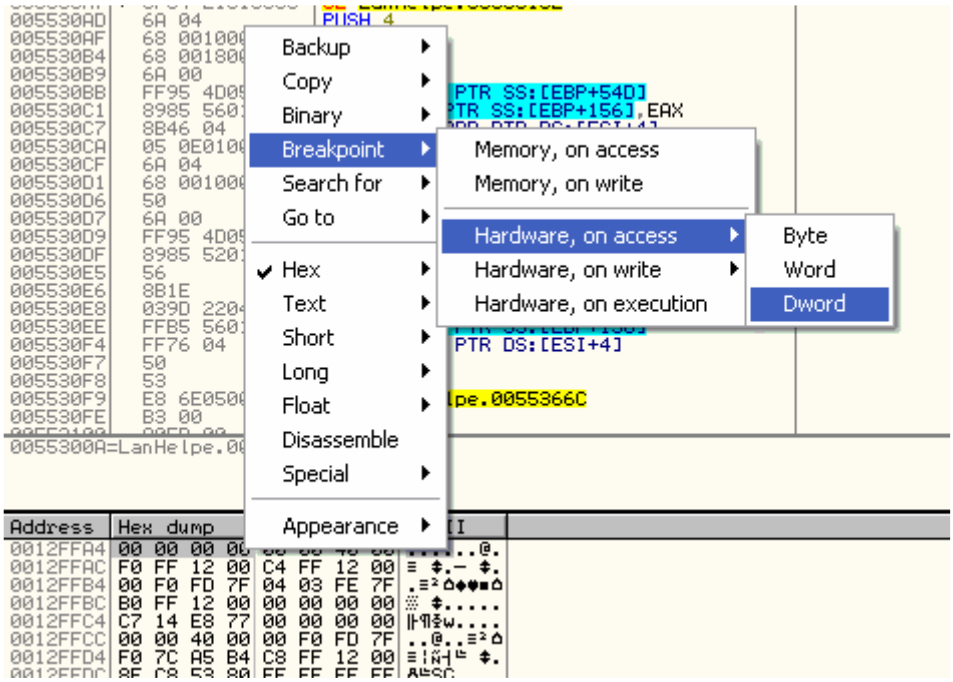
شکل ۱۲

حال در پنجره dump مانند شکل زیر آدرس 12FFA4 را انتخاب کرده و ۴ بایت اول را highlight کنید (توجه کنید آدرس مزبور همان مقداری میباشد که در ESP قرار دارد)

حال که highlight کردید دکمه سمت راست را زده و گزینه:

Breakpoint -> Hardware on access -> Dword

را انتخاب کنید:



شکل ۱۳

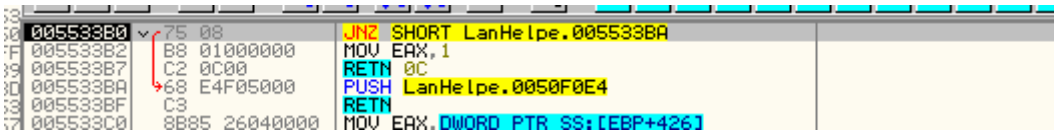
آدرس 12FFA4 (آدرس بالای پشته میباشد)

ما با انتخاب چهار بایت اول پشته و قرار دادن Breakpoint در آنجا میخواهیم هنگامی که مقدار EDI (یعنی اولین ثباتی که مقدارش برمیگردد) برگشت داده میشود ما مطلع شویم.

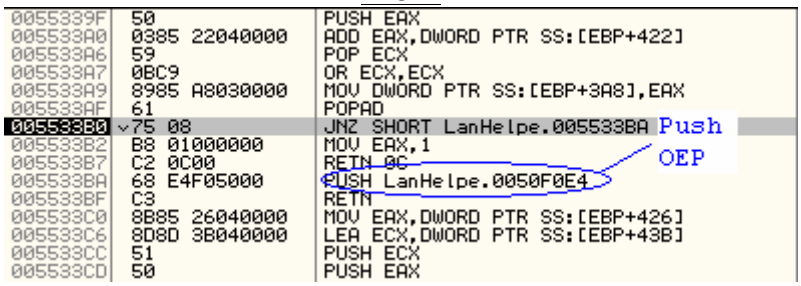
دلیل اینکه Dword را انتخاب کرده ایم این بود که ما ۴ بایت را انتخاب کردیم اگر ۲ بایت را انتخاب میکردیم باید Word را انتخاب میکردیم.

خب حالا برنامه را با F9 اجرا کنید.

در آدرس 5533B0 برنامه متوقف میشود کمی نوار پیمایش را بالا ببرید میبنید کـه دقیقـاً قبـل از آدرس 5533B0 دسـتور POPAD وجود دارد خب ما در بهترین جایی که ممکن بود متوقف شده ایم. اما هنوز OEP را پیدا نکـردیم Packer بایـد عملیات decrypt خود را به اتمام برساند تا به OEP برگردد.



شکل ۱۴



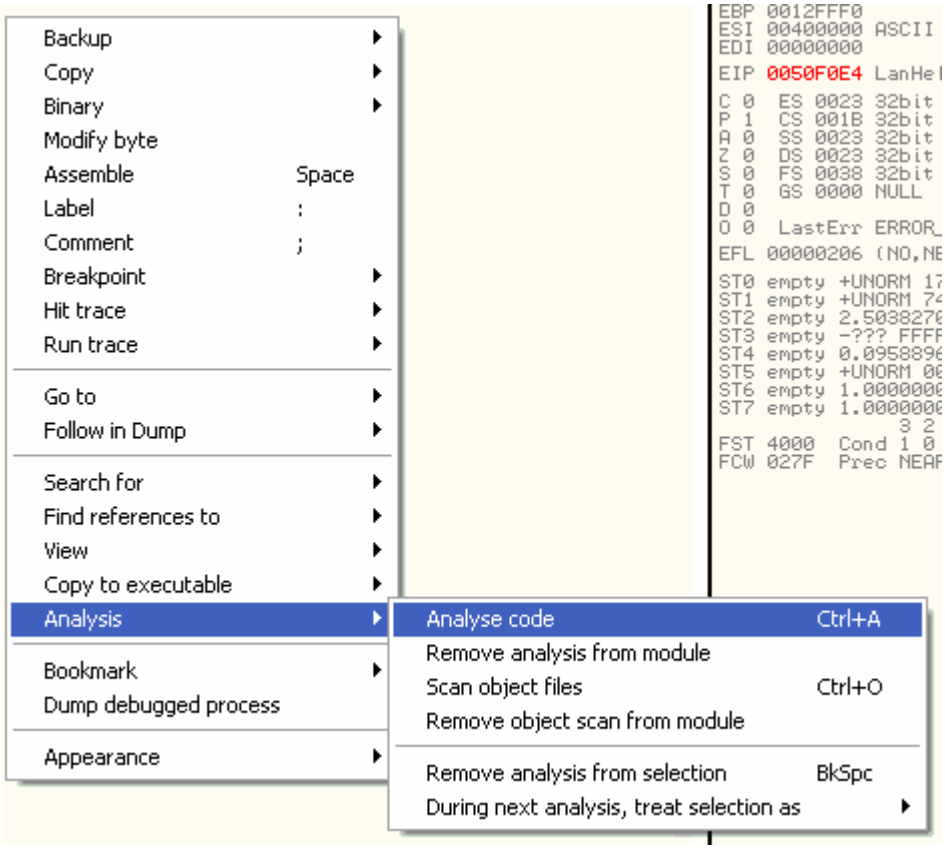
شکل ۱۵

برای پیگیری برنامه دوبار کلید F7 را فشار دهیدمیبیند که در آدرس 5533BA مقداری در پـشته قـرار میگیـرد ایـن مقـدار ، مقدار OEP هست که در پشته قرار میگیرد.یکبار دیگر F7 را زده تا آدرس 5533BF یعنی دستور RETN اجرا شود بعد از اجرای این دستور ما در OEP قرار داریم.

مانند شکل زیر:

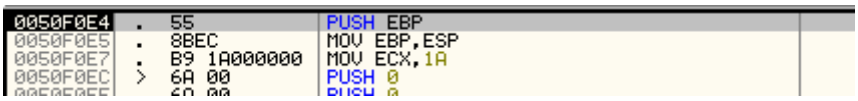
شکل ۱۶

خب حال در این نقطه کلید ترکیبی ctrl + a را فشار دهید و یا مانند شـکل زیـر دکمـه سـمت راسـت مـوس را زده و گزینـه analyse code را انتخاب کنید.



شکل ۱۷

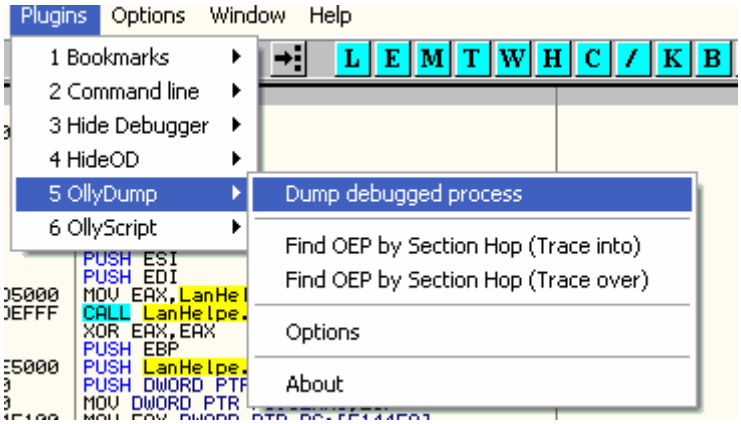
بعد از آنالیز کد ollydbg کد برنامه را بصورت زیر نشان خواهد داد:



شکل ۱۸

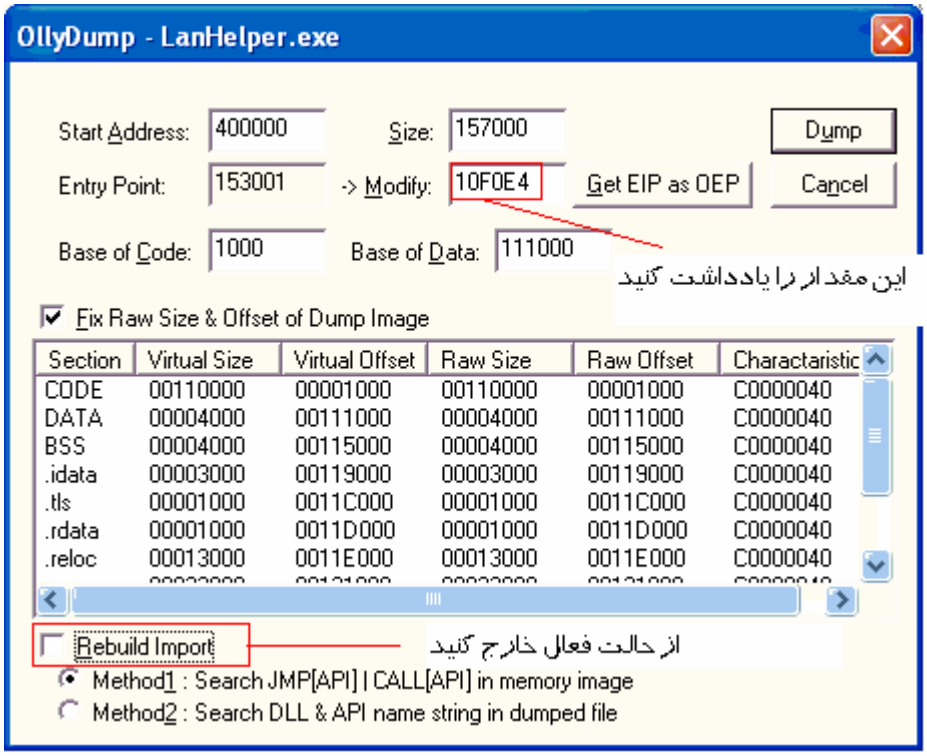
خب حال که ما OEP را پیدا کردیم وقت Dumping میباشد من در اینجا از plug-in بنام ollydumper استفاده کرده ام:

از منوی plugin بصورت شکل زیر منوی dumped debug process را انتخاب کنید:



شکل ۱۹

دایلوگی مانند شکل زیر خواهید دید:

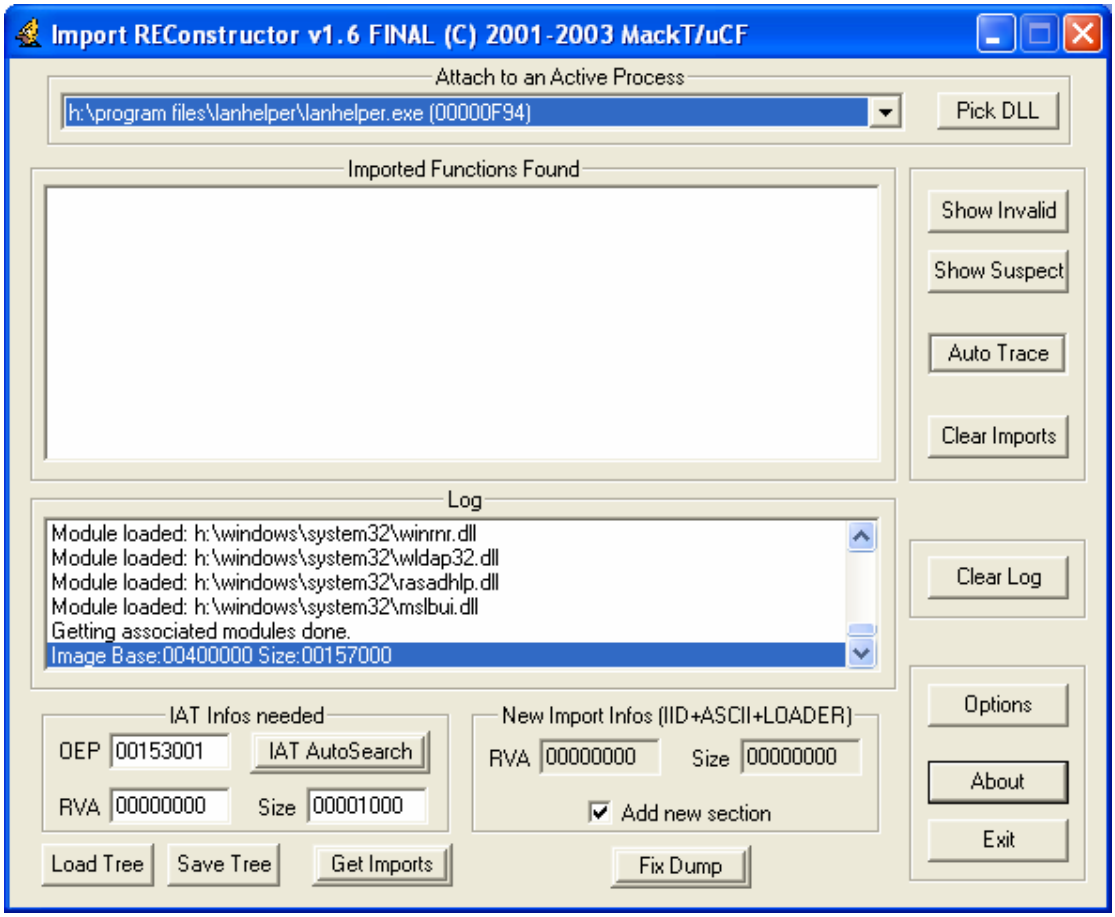


شکل ۲۰

خب مقدار 10F0E4 مقدار OEP بوده توجه کنید مانند شکل ۲۰ حتماً عمل کنید.
خب حال برروی دکمه dump کلیک کرده از شما میخواهد که نامی برای برنامه ای که تا اینجا dump کرده ایم انتخاب کنید و آنرا ذخیره کنید نام آنرا lanhelper_dump.exe گذاشته و آنرا در فولدری که lanhelper نصب شده است ذخیره کنید.
خب حال ما برنامه را از حالت pack شده درآورده و توانسته ایم آنرا ذخیره کنیم ولی برنامه dump شده ما یک مشکل دارد و آن ایست که اجرا نمیشود.
اگر بخاطر داشته باشید درمورد یکی از sectionهای برنامه های اجرایی بنام IAT برای شما توضیح دادم این جدول هنگامی که برنامه را dump میکنیم dump نمیشود و در واقع خود ما باید این جدول را بسازیم و آنرا ضمیمه فایل اجرایی خود کنیم.

ترمیم و ساخت مجد IAT:

برای ساخت مجدد IAT ما به یک نرم افزاری احتیاج داریم که این جدول را از فایل اصلی خوانده و در فایل dump شده ما قرار دهد. ما در اینجا از ImpREC استفاده خواهیم کرد.
خب برای شروع کار فایل lanhelper اصلی را اجرا کنید و بعد از آن نر افزار ImpREC را اجرا کنید و مانند شکل زیر Lanhelper را انتخاب کنید:



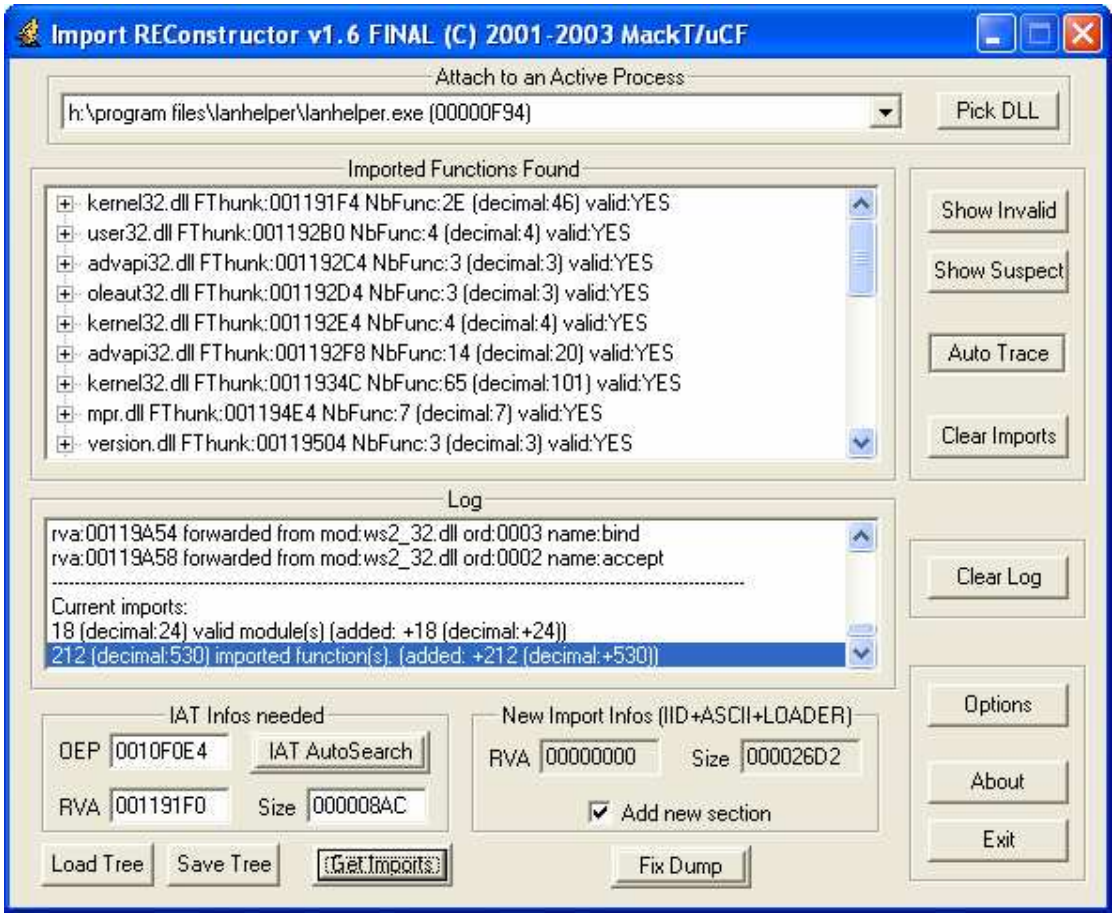
شکل ۲۱

ImpREC برای اینکه بتواند آدرس IAT را بدست آورد احتیاج به دانستن آدرس OEP دارد اگر از قسمت قبل به یاد داشته باشید در ollydump یک آدرس را یادداشت کرده بودیم : 10F0E4، حال این آدرس را در textbox مربوط به OEP بدین صورت وارد کنید:0010F0E4

توجه کنید حتماً به هیمن صورت باشد بعد از وارد کردن آدرس درست OEP دکمه IAT AutoSearch را فشار دهید یک پیغام بصورا زیر ظاهر خواهد شد:

شکل ۲۲

حال دکمه Get Imports را فشار دهید ImpREC بصورت زیر خواهد شد:



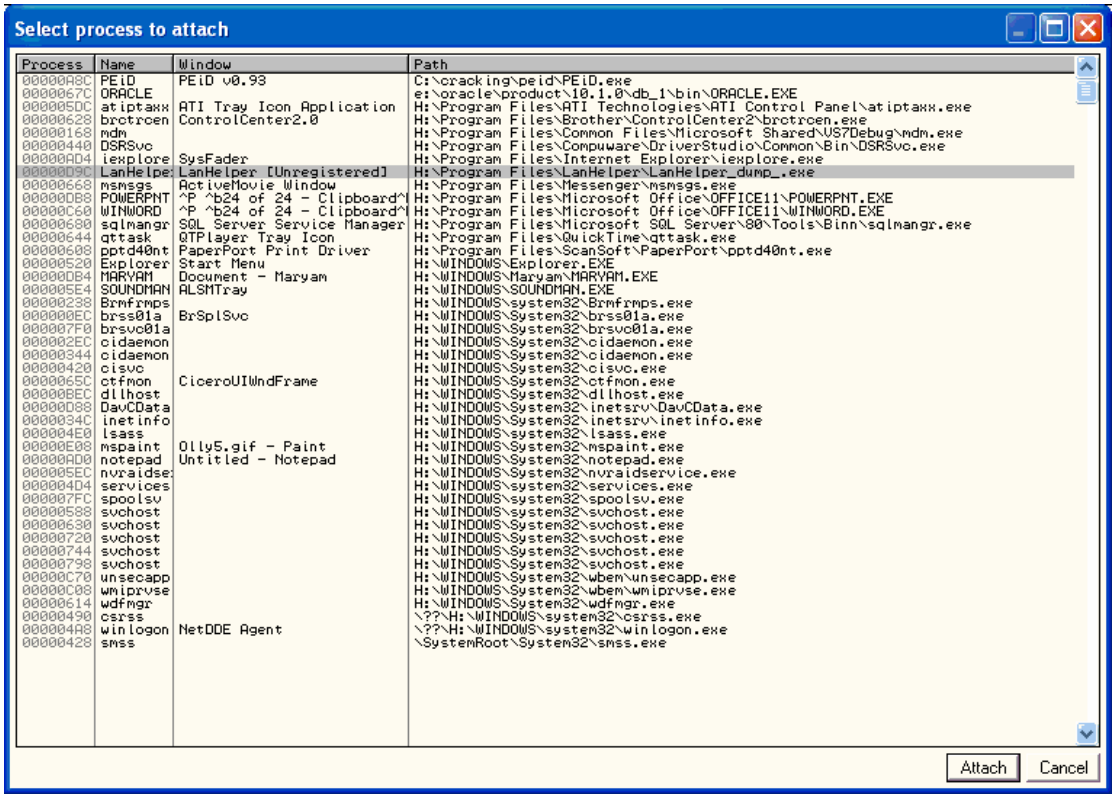
شکل ۲۳

اگر packer بعضی از توابع را بصورت درهم ریخته ذخیره کند در پنجره Imported Functions Found در آخر خط بجای اینکه بنویسد Found : Yes مینویسد Found : No برای اینکه مطئمن شویم تمام توابع شناسایی شده اند برروی دکمه Show Invalid کلیک کنید میبینید که هیچ اتفاقی نیفتاد یعنی اینکه تمامی توابع پیدا شده اند.

حال برروی دکمه Fix Dump کلیک کنید حال در دایلوگ باز شده فایل lanhelper_dump.exe را دابل کلیک کنید تا IAT آن ترمیم شود.
ImpREC یک فایل جدید با نام lanhelper_dump_.exe میسازد.
خب ImpREC و lanhelper اصلی را بسته و فایلی که ImpREC ساخته است دابل کلیک کنید که ببینیم آیا توانسته ایم برنامه را unpack کنیم یا نه؟
خب برنامه کار میکند حال فقط یک مشکل باقیست و آنهم اینکه برنامه رجیستر نشده است.

Patching برنامه dump شده:

برای اینکه بتوانیم شماره سریال برنامه dump شده را patch کنیم ابتدا برنامه lanhelper_dump_.exe را اجرا میکنیم.
حال ollydbg را اجرا میکنیم. از منوی File گزینه Attach را انتخاب میکنیم.
در اینجا میخواهم به شما طریقه استفاده از ویژگی attach در ollydbg را نشان دهم به این دلیل که، فایلی که بصورت dump شده میباشد نمیتواند بصورت مستقیم در ollydbg باز شود چراکه باعث میشود ollydbg در یک حلقه بی انتها گیر کند و شاید شما مجبور شوید کامپیوتر خود را ریستارت کنید.
حال مانند شکل زیر فایل lanhelper_dump_.exe را از لیست انتخاب کنید:



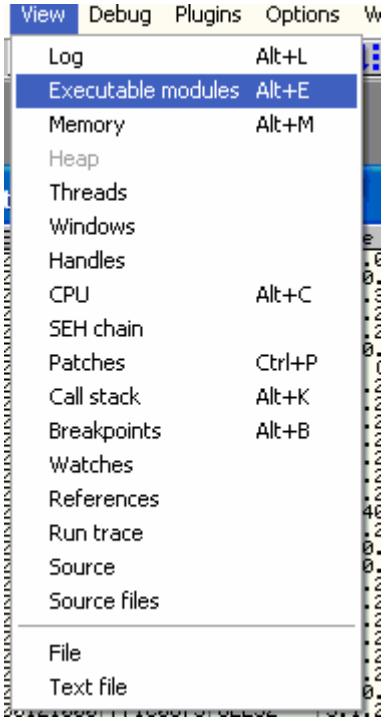
شکل ۲۴

و دکمه Attach را فشار دهید.

برنامه درون ollydbg لود میشود ولی به قسمت caption نرم افزار ollydbg توجه کنید:

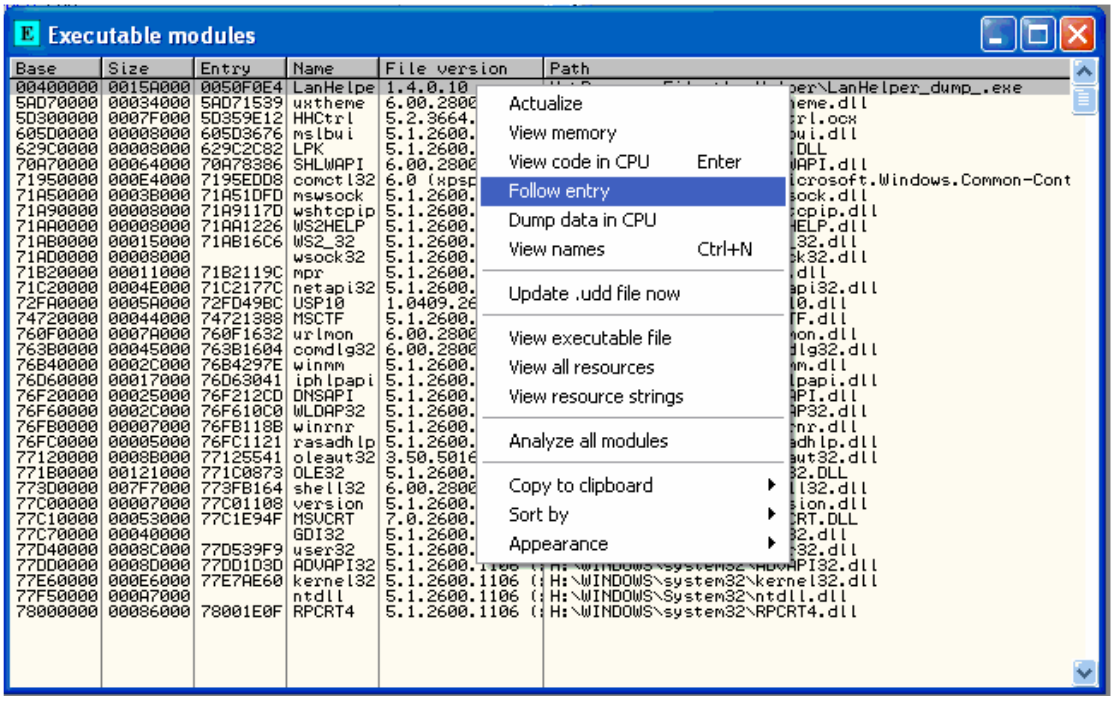
شکل ۲۵

در واقع شما الان در فایل ntdll.dll هستید برای اینکه به فایل موردنظر خودمان برگردیم از منوی view گزینه executable modules را انتخاب کرده



شکل ۲۶

در پنجره پدیدار شده برروی گزینه _lanhelper_dump دکمه سمت راست موس را زده و مانند شکل زیر گزینه follow entry point را انتخاب کنید.

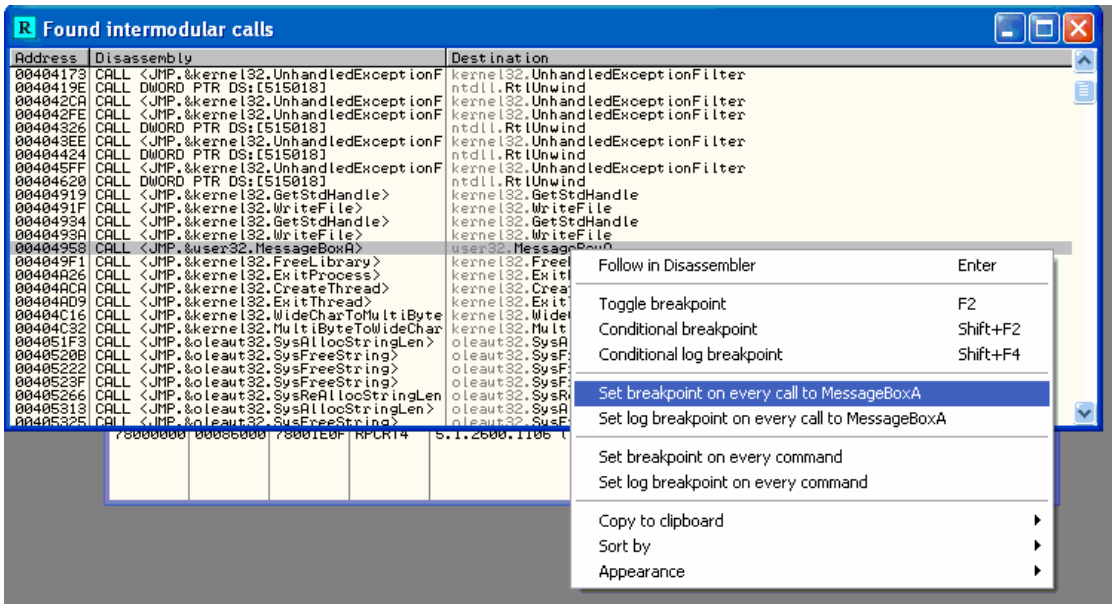


شکل ۲۷

خب حال ollydbg پنجره کد مربوط به برنامه موردنظر ما را نشان میدهد.

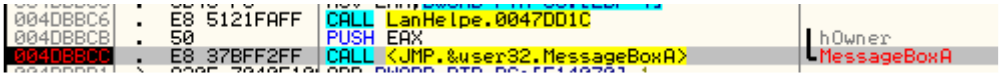
همانطور که از قبل به خاطر دارید ما یک پیغام خطا داشتیم با این عنوان The registration code you have entered is not correct این پیغام دردرون یک messagebox نشان داده میشود اگر از مطالب قبلی به خاطر داشته باشید بر روی پنجره کد در ollydbg دکمه سمت راست موس را زده از منوی پدیدار شده گزینه search for و بعد از آن

گزینه all intermodullar calls را انتخاب میکنیم در پنجره پدیدار شده messagebox را تایپ کرده تا بر روی یکی از توابع messagebox بایستیم بر روی آن دکمه سمت راست را زده و گزینه set BreakPoint in every call to Messagebox را انتخاب میکنیم.



شکل ۲۸

حال برنامه را با F9 ادامه داده و میرویم که برنامه را رجیستر کنیم برای اینکار در برنامه اصلی به منوی Help و گزینه About رفته در پنجره پدیدار شده enter registration code را انتخاب کنید یک نام و یک شماره بنویسید و دکمه ok را کلیک کنید هنگامی که بر روی دکمه ok کلیک میکنید برنامه در آدرس4DBBCC توقف میکند اینجا همانجایی هست که هنگامی که کد اشتباه وارد میکنیم وارد آن میشویم به شکل زیر توجه کنید:



شکل ۲۹

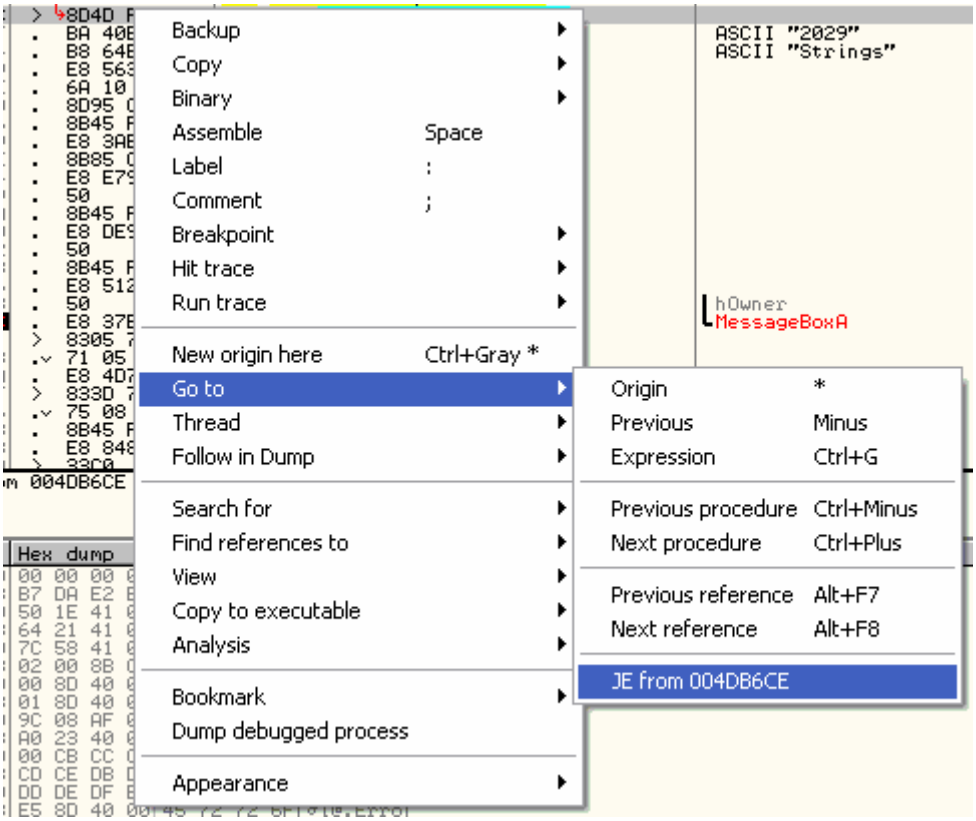
خب حال کمی نوارپیمایش را بالاتر ببیرید متوجه چیز مشکوکی نخواهید شد حال دوباره کلیدهای ctrl+a را فشار دهید تا کد آنالیز شود میبینید که در آدرس 4DBB8C یک entry point وجود دارد مانند شکل زیر:

شکل ۳۰

این entry point جایی هست که کد چک میشود و درصورت غلط بودن ما به این نقطه خواهیم آمد از کجا متوجه شدیم که این یک entry point هست؟

با کمی توجه میفهمیم که در کنار هر entry point یک علامت > وجود دارد به همین سادگی.

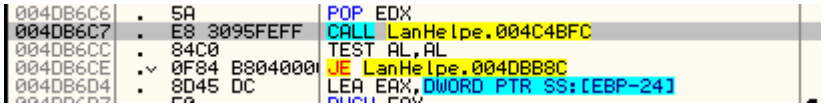
خب ما باید به دنبال کدی در آدرس بالاتر بگردیم حال بر روی آدرس 4DBB8C دکمه سمت راست مـوس را زده از گزینـه goto آخرین گزینه یعنی JE from 004DB6CE را انتخاب کنید.



شکل ۳۱

خب حال ما در آدرس 4DB6CE هستیم.

کمی نوارپیماش را بالاتر ببرید دقیقاً در بالای کد JE دستور TEST AL,AL قرار دارد این یعنی اینکه در اینجا مقداری که ما داده ایم چک میشود و درصورت درست بودن تصمیم مورد نظر گرفته میشود در آدرس4DB6C7 یک دستور call وجـود دارد این دستور یعنی همان تابعی که مقدار ورودی را چک کرده و جواب را بـه آدرس 4DB6CC یعنـی TEST AL,AL منتقل میکند.

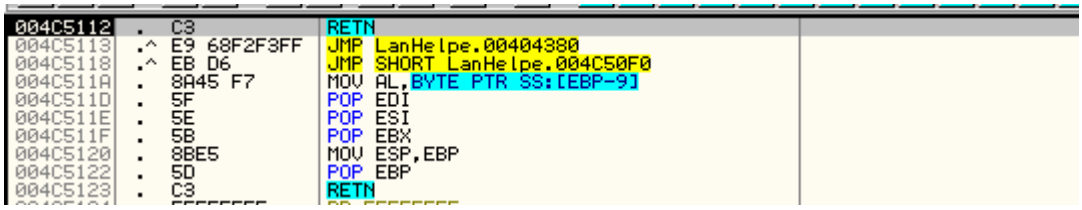


شکل ۳۲

خب حال در آدرس 4DB6C7 کلید F2 را بزنید و بعد کلید F9 را . حال در برنامه lanhelper جعبه پیغام خطا را بسته و دوباره یک اسم کاربر و پسورد غلط وارد نمایید تا ollydbg ما را در خط 4DB6C7 متوقف کند.

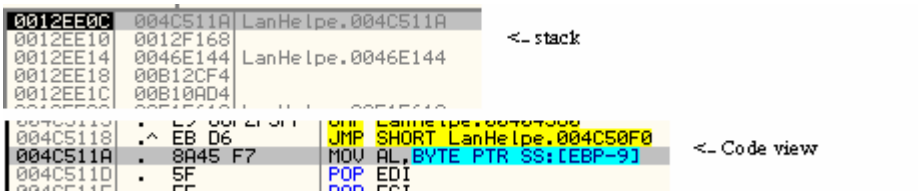
حال در نقطه توقف برنامه را با کلید F7 پیگیری کنید با زدن یکبار F7 به آدرس 4C4BFC خواهیم رفت یعنی تابع چک کردن اسم کاربر و شماره سریال او.

خب حال که ما وارد کد تابع کنترل کننده شده ایم بجای آنکه تمام کد تابع را بگردیم به دنبال مقدار AL به انتها کد خواهیم رفت (اینکار به این دلیل انجام میشود که ما از قبل میدانیم که مقادار AL یا صفر است و یا یک) بوسیله ctrl+F9 و یا زدن دکمه  به انتهای تابع خواهیم رفت.



شکل ۳۳

بوسیله این دکمه (ctrl+F9) به ollydbg فرمان میدهیم که کد تابع را تا جایی که به دستور RETN میرسد اجرا کند. هنگامی که به آدرس 4C5112 رسیدیم یعنی همان آدرسی که ollydbg ما را در آن متوقف میکند نگاهی به پنجره پشته بیاندازید.



شکل ۳۴

این یعنی اینکه دستور RETN ما را به آدرس 4C511A هدایت خواهد کرد حال به این آدرس مزیور نگاه بیاندازید:

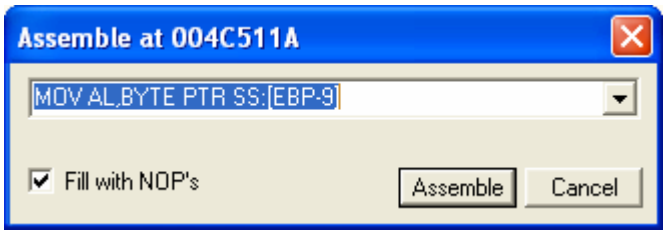
شکل ۳۵

در این آدرس همانطور که میبینید مقدار Byte PTR SS:[EBP-9] درون AL قرار میگیرد برای اینکه ببینید این مقدار چیست در پنجره ثباتها بر روی مقدار EBP دکمه سمت راست را زده و گزینه Follow in Dump را انتخاب کنید در پنجره Dump ۹ بایت به عقب برگشته مقداری که در آنجا ذخیره شده است 00 است. این به این معنا میباشد که AL حاوی مقدار صفر خواهد شد.

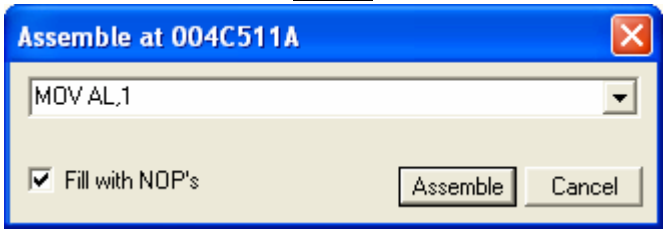
حال از قبل میدانیم که AL باید حاوی مقدار 1 باشد تا برنامه بصورت رجیستر شده دربیاید برای اینکار بر روی آدرس 4C511A ستون سوم دابل کلیک کنید و مقدار دستور مورد نظر به این دستور تغییر دهید:

MOV AL,1

مانند شکل زیر:



شکل ۳۶



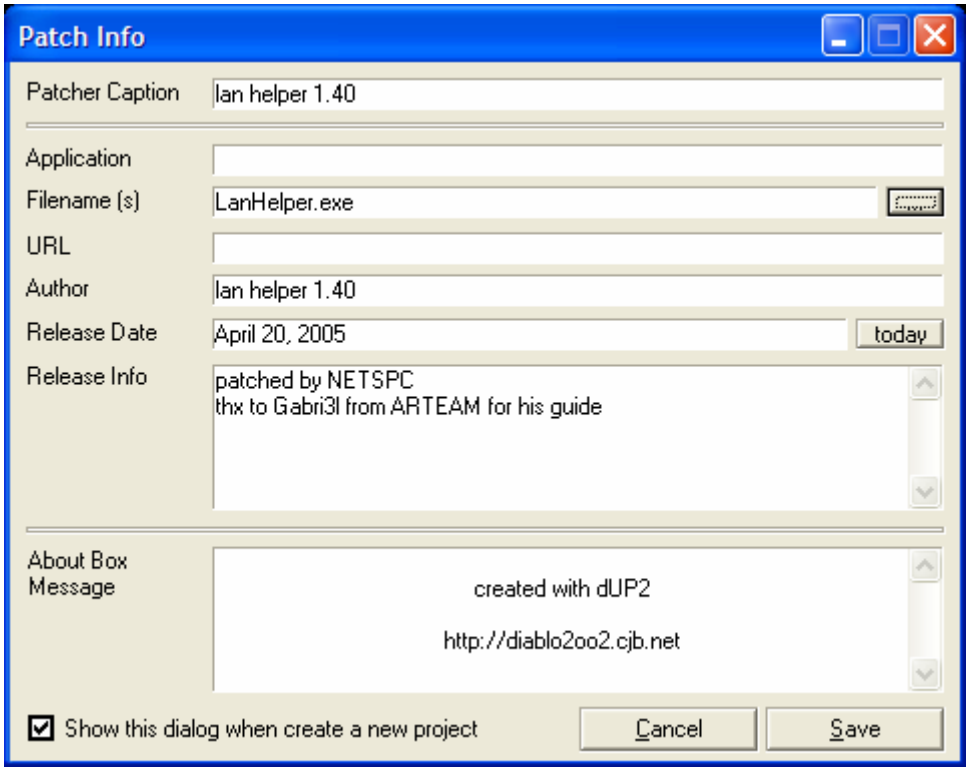
شکل ۳۷

خب حال بر روی دکمه assemble کلیک کنید دکمه F9 را فشار دهید تا برنامه اجرا شود.
برای ذخیره برنامه patch شده بر روی پنجره کد دکمه سمت راست را بزنید از گزینه copy executable گزینه All modification و بعد از آن دکمه copy all و در پنجره جدیدی که باز میشود دوباره دکمه سمت راست را زده و گزینه save file را انتخاب کنید فایل را با نام Lanhelper_dump_p.exe ذخیره کنید.
حال برای اینکه مطمئن شویم برنامه کار میکند ollydbg را بسته و فایل patch شده را اجرا کنید و یک نام و شماره بصورت غلط وارد کنید برنامه از شما میخواهد برنامه را بسته و دوباره باز کنید هنگامی که اینکار را میکنید دیگر پیغامی مبنی بر اینکه شما چند روز دیگر فرصت ندارید نخواهید دید این برنامه بصورت کامل کرک شد.

شکل ۳۸

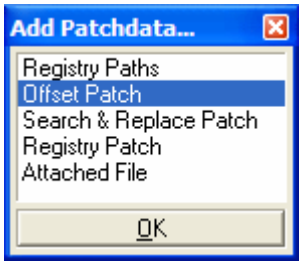
ساخت patch:

همانطور از مقالات قبلی به خاطر دارید من در این مقالات از نرم افزار dup 2 استفاده میکنم.
در اینجا میخواهم نحوه ساخت یک patch را از روی فایل اصلی که بصورت pack درآمده هست را به شما یاد بدهم.
برای ساخت patch برنامه dup 2 را اجرا کرده و بر روی دکمه new project کلیک کرده در پنجره پدیدار شده اطلاعات را مانند شکل زیر کامل کنید:



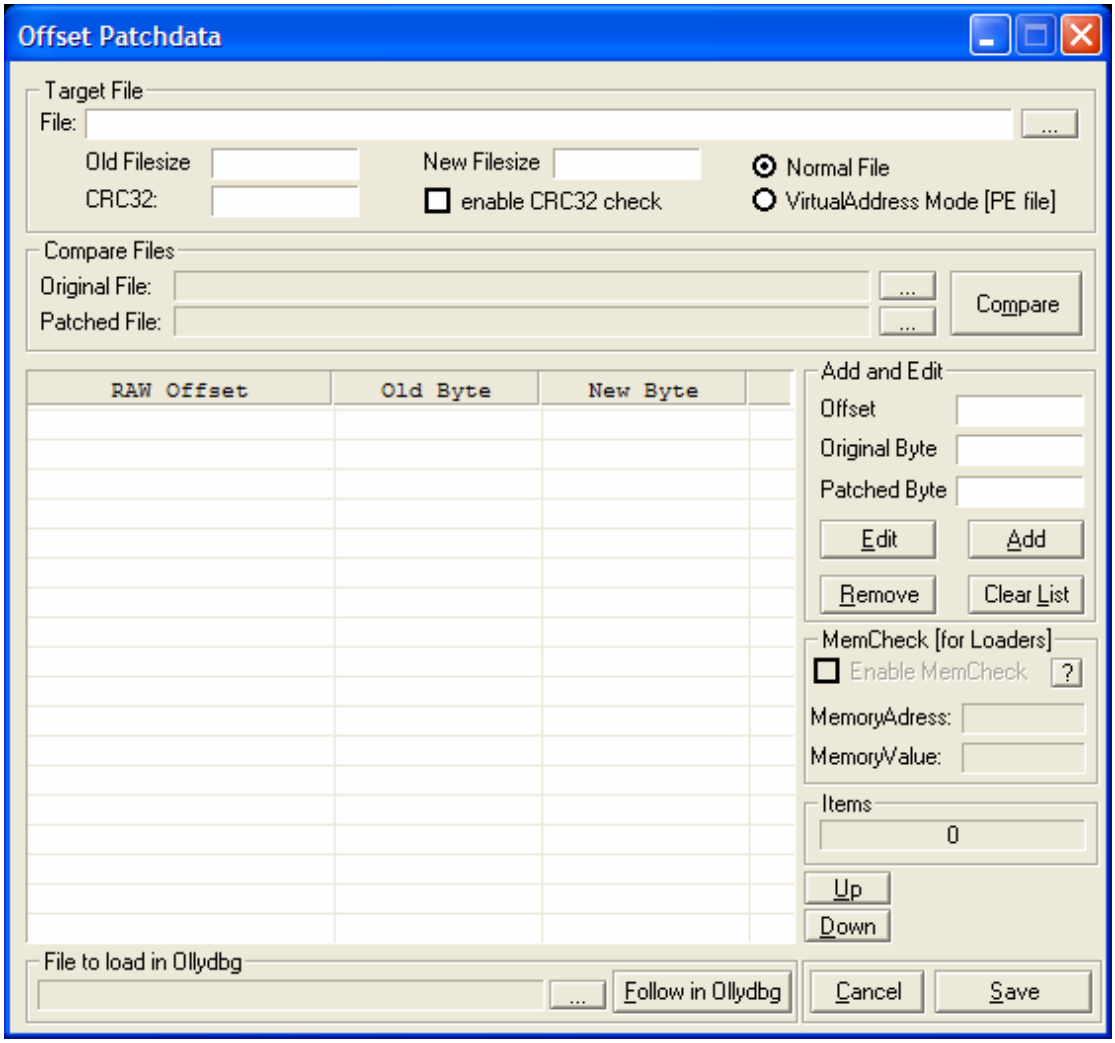
شکل ۳۹

و بعد از آن بر روی دکمه save کلیک کنید.
حال دوباره در پنجره اصلی dup بر روی دکمه Add کلیک کرده از منوی پدیدار شده گزینه offset patch را انتخاب کنید



شکل ۴۰

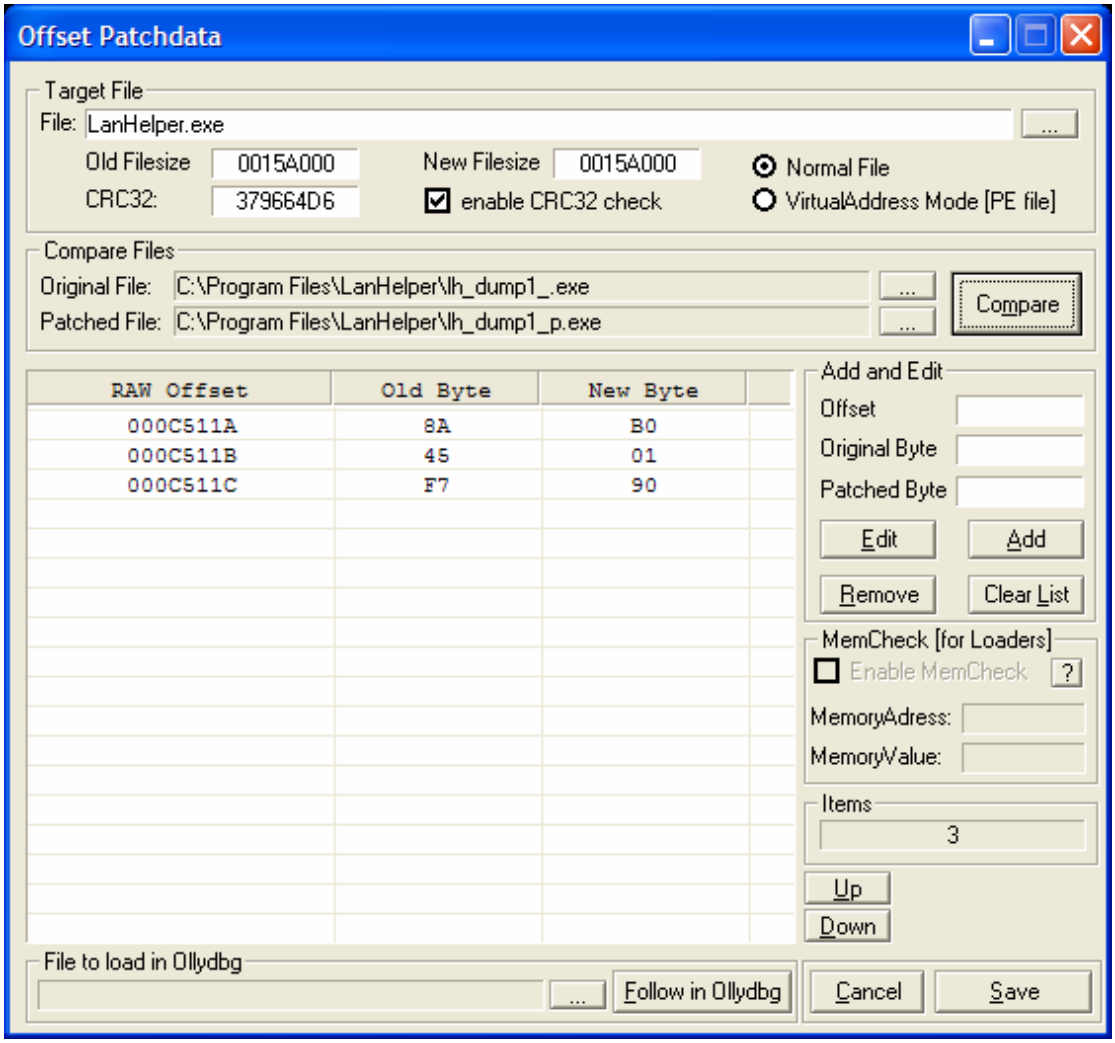
بعد از زدن دکمه ok یک گزینه در پنجره اصلی اضافه میشود بنام Offset Patch بر روی آن دابل کلیک کنید تا یک صفحه مانند زیر پدیدار شود.



شکل ۴۱

اگر از آموزش قبل به خاطر داشته باشید در پانل compare files باید آدرس فایل patch نشده و فایل patch شده را بدهیم تا dup بتواند patch مخصوص را بسازد ولی در این حالت چند کار اضافی دیگر باید انجام دهیم.

۱- از پانل target file گزینه Open را انتخاب کرده و فایل اصلی lanhelper.exe که نه unpack شده و نه patch شده است را انتخاب کنید.
۲- گزینه CRC checked علامت خورده باشد
۳- گزینه virtual address انتخاب شده باشد(چون فایل اصلی pack شده است)
۴- در پانل compare file در فیلد original file فایلی که قبلاً dump کرده بودیم ولی patch نشده بود انتخاب شود
۵- در پانل compare file در فیلد patched file فایلی که dump شده بود و patch شده بود انتخاب گردد
۶- دکمه compare را فشار دهید.



شکل ۴۲

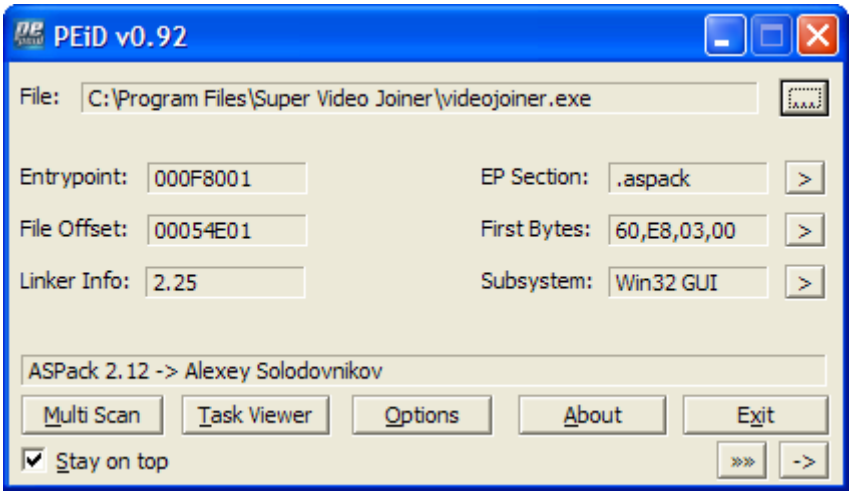
حال برروی دکمه save کلیک کنید و patch ساخته شده را ذخیره کنید

در صفحه اصلی برروی دکمه create patch کلیک کنید از شما آدرس جایی را که میخواهید patch شما ذخیره گردد را میخواهد آنرا مشخص کنید.

پایان

آموزش کرکینگ با ollydbg قسمت چهارم
تاریخ : اردیبهشت ۱۳۸۵ 2006 April
نوع حمله serial protection ,unpacking Aspack 2.12 with crippled IAT
نرم افزار هدف : Super Video Joiner 1.8.0
نرم افزارهای مورد استفاده :ollydbg 1.10 (hidedebugger ,ollydump) plugin , DUP 2.0,PEid 0.92 ,ImpRec 1.6
سطح : متوسط

خب اول نرم افزار رو نصب میکنیم.
بعد از نصب همیشه اولین کاری که باید بکنیم این هست که اول بفهمیم برنامه رو با چی نوشتن و با چی پک شده برای اینکار میتوانید از نرم افزارهایی مانند PEid و یا RDG Packer Detector استفاده کنید من اینجا از PEid استفاده کردم.
برنامه PEid رو باز کرده و فایل videojoiner.exeرو بندازید توش و یا از دکمه ...(open) برای تستِ فایل استفاده کنید



شکل ۱

نکته:
شاید بصورت تجربی متوجه شده باشید که نرم افزار PEiD گاهی نمیتواند نوع زبان برنامه‌نویسی و نوع packer و یا protector برنامه هدف را بصورت صحیح تشخیص دهد.
این خطا به این دلیل میباشد که نرم افزار PEiD تمامی امضا(packer (signatureها و protectorها را نمیشناسد. در واقع این امضاها در فایلی بنام USERDB.TXT ذخیره میشوند. این فایل بصورت دستی باید هر چند وقت یکبار توسط شما باید جایگزین فایل قدیمی شود. محلی هم که میتوانید این فایل را بصورت بروز شده و تازه بیابید انجمن های در رابطه با مهندسی معکوس و کرک میباشد.
هنگامی که نرم‌افزار PEiD را تازه نصب میکنید فایل USERDB.TXT بصورت نمونه در آن موجود میباشد.

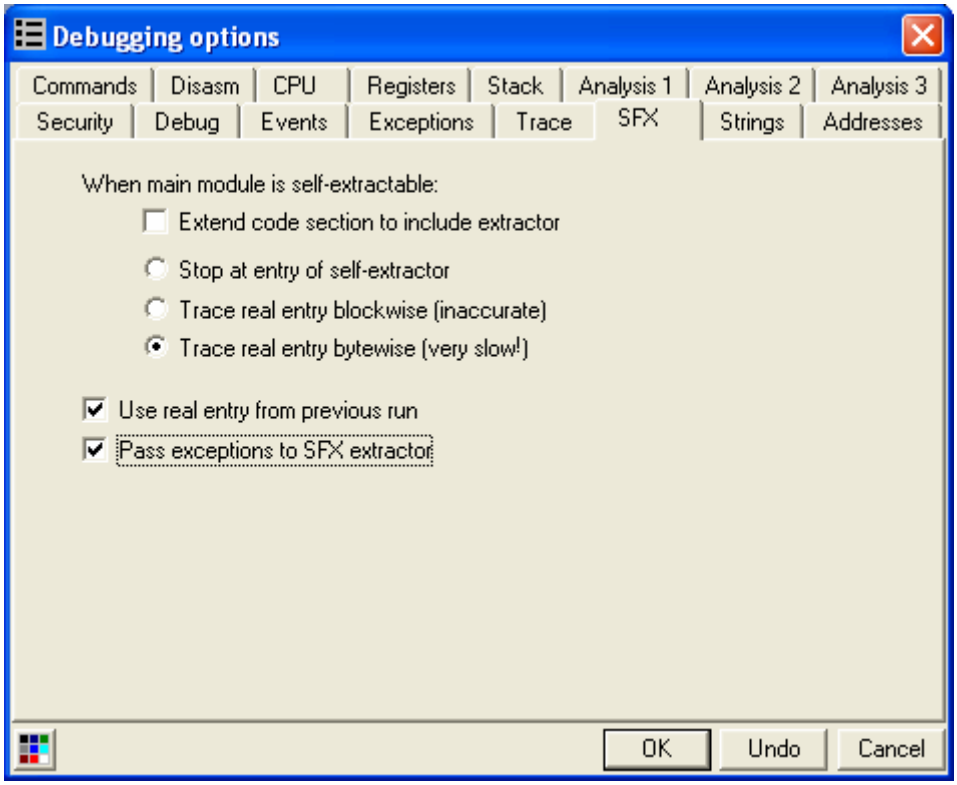
این برنامه هم مانند برنامه قبل بصورت ساده ای پک شده است.
در واقع Aspack فقط یک packer میباشد . هدف یک packer فقط کاهش حجم فایل اصلی میباشد و تقریباً هیچگونه متدی برای جلوگیری از دیباگ شدن و یا کرک شدن برنامه ندارد.
خاصیت packer ها "re-entrant" بودن آنها میباشد یعنی بعد از اینکه برنامه بصورت unpack در حافظه قرار گرفت هیچ ردی از خود برجای نمیگذارند یعنی اینطور به نظر میرسد که اصلاً هیچ اتفاق خاصی نیافتاده است.(به مقاله قبلی توجه کنید).
در طرف دیگر protectorها هستند مانند Asprotect بجای اینکه مانند packerها فقط حجم فایل را کاهش دهند، وظیفه عمده آنها محافظت از تغییر کد اصلی میباشد.همچنین وقتی برنامه اصلی از حالت pack شده در آمده و در حافظه قرار میگیرد تغییرات بسیاری در بعضی از sectionهای آن بوسیله protector بوجود می آید در واقع protectorها مانند packerها "re-entrant" نمیباشند و تغییراتی را برای سخت کردن و یا غیر ممکن کردن عمل دیباگینگ و کرک بکار میگیرند.

ولی در این آموزش قصد ندارم فقط بر روی آنپک کردن برنامه و پچ کردن آن بحث کنم بلکه در این آموزش میخواهم شما را کمی با تئوری کارکرد ImpREC و اینکه پچ چرا و چگونه کار میکند آشنا کنم.

بیایم با unpack کردن اتوماتیک بوسیله ollydbg آشنا شویم:

مانند مثال قبل همانطور که بخاطر دارید باید برنامه را در ollydbg باز کنیم.
(برای تمرین بیشتر خودتان میتوانید بصورت دستی برنامه مورد نظر را unpack کنید.)
در اینجا میخواهم روش دیگر را برای unpack کردن به شما نشان دهم. درواقع میخواهم به شما یکی دیگر از ویژگیهای ollydbg را نشان دهم.

برای اینکه با این روش جدید آشنا شوید در ollydbg در منوی option گزینه Debugging Option را انتخاب کنید برای اینکار میتوانید از دکمه  استفاده کنید.
در پنجره باز شده گزینه SFX را انتخاب کنید و تنظیمات حال ollydbg را بخاطر داشته باشید(آنها را یادداشت کنید) حال تنظیمات را مانند شکل زیر تغییر دهید.

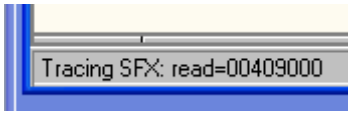


شکل ۲

با اینکار ollydbg متوجه میشود که خود باید بصورت اتوماتیک برنامه را از حالت pack شده خارج کند.

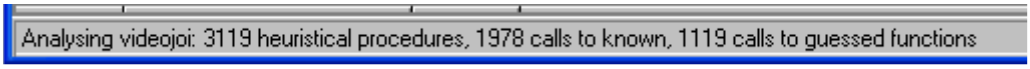
حال فایل videojoiner.exe را در ollydbg باز کنید.

هنگامی که فایل را در ollydbg باز میکنید به قسمت status bar توجه کنید:



شکل ۳

میبینید که یک شمارنده ای در حال شمارش میباشد بعد از مدتی (بستگی به نوع CPU شما دارد) status bar مانند شکل خواهد شد:



شکل ۴

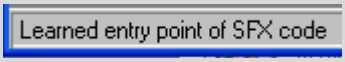
و پنجره کد ollydbg مانند شکل زیر:

شکل ۵

این روش برای تمامی packerهایی که ویژگی "re-entrant" را دارا میباشند کاربرد دارد.

نکته:

نرم افزار ollydbg اطلاعات فایلهایی را که دیباگ میکند در فایلی همنام فایل مربوطه ولی با پسوند udd در دایرکتوری خود نگه داری میکند هنگامی که ollydbg را میبندید و بار دیگر همان برنامه را در ollydbg باز میکنید در قسمت status bar حتماً متوجه نوشته مربوط خواهید شد:



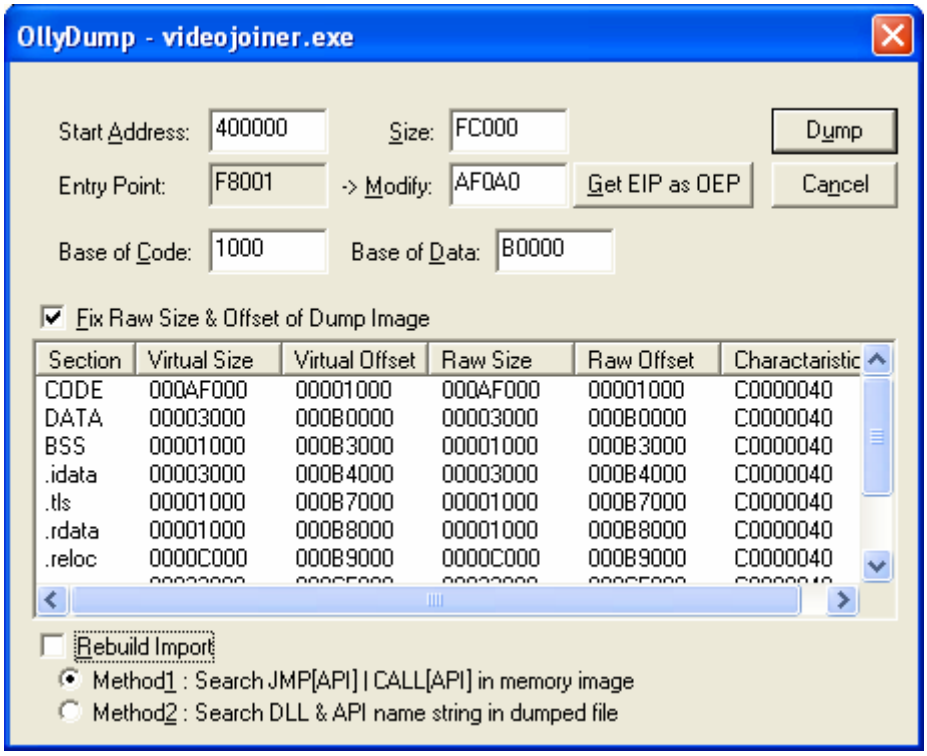
اینکار به این دلیل انجام میگیرد که در هر بار باز کردن برنامه در ollydbg وقت شما گرفته نشود

Dumping برنامه و درست کردن IAT:

الان ما میدانیم OEP کجاست و الان وقت dumping میباشد در این قسمت برای تمرین سعی کنید مانند مقاله قبل عملیات dumping را انجام دهید میبینید که از اتمام عملیات و درست کردن IAT بوسیله ImpREC باز برنامه اجرا نخواهد شد.

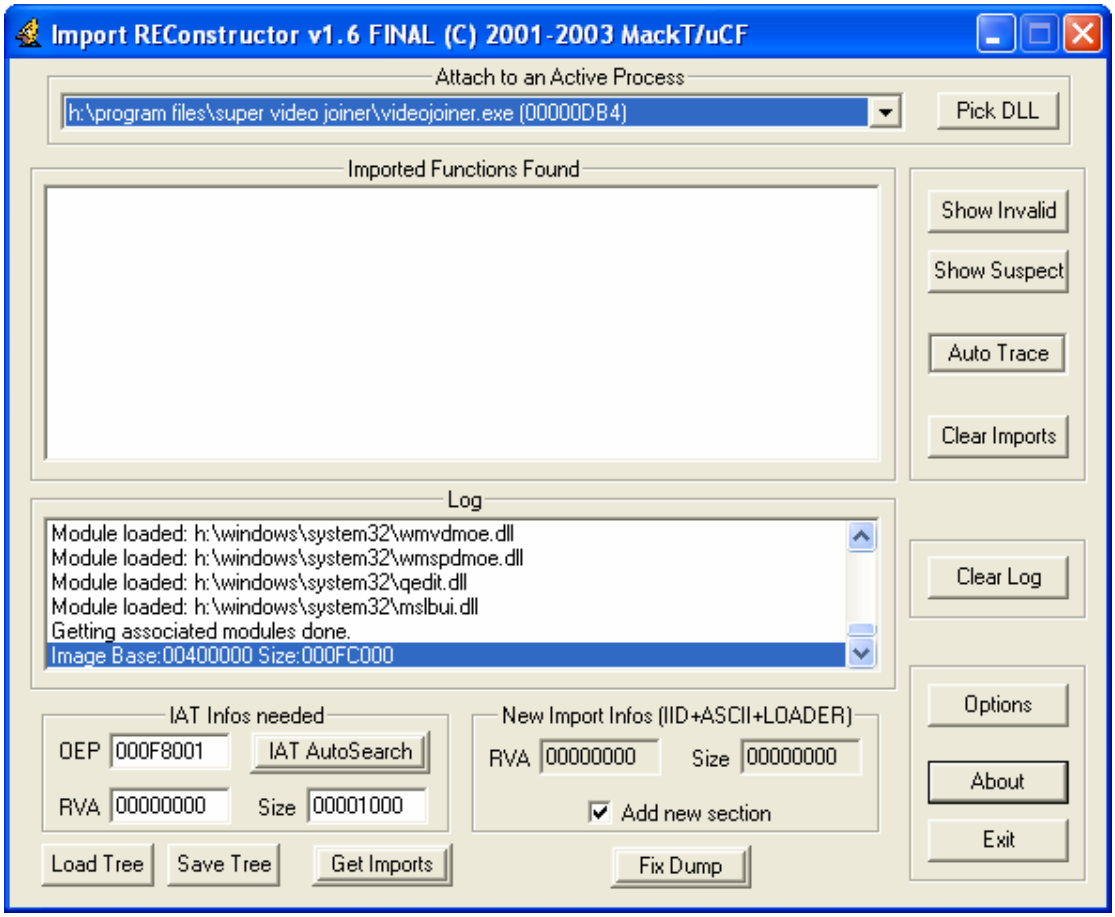
توجه:
برنامه ضمیمه شده بدین صورت نمیباشد و در ویندوز xp sp1&2 شما متوجه این مشکل نخواهید شد این به این دلیل میباشد که بعضی از ارجاعات درون برنامه به APIها ویندوز نبوده.
ولی در اینجا تمامی مراحل برای یادگیری شما توضیح داده شده است.

Dumping برنامه:
بوسیله plugin مربوط به dumping به اسم ollydump گزینه dump debugged process را انتخاب کنید مانند شکل زیر را خواهید داشت:



شکل ۶

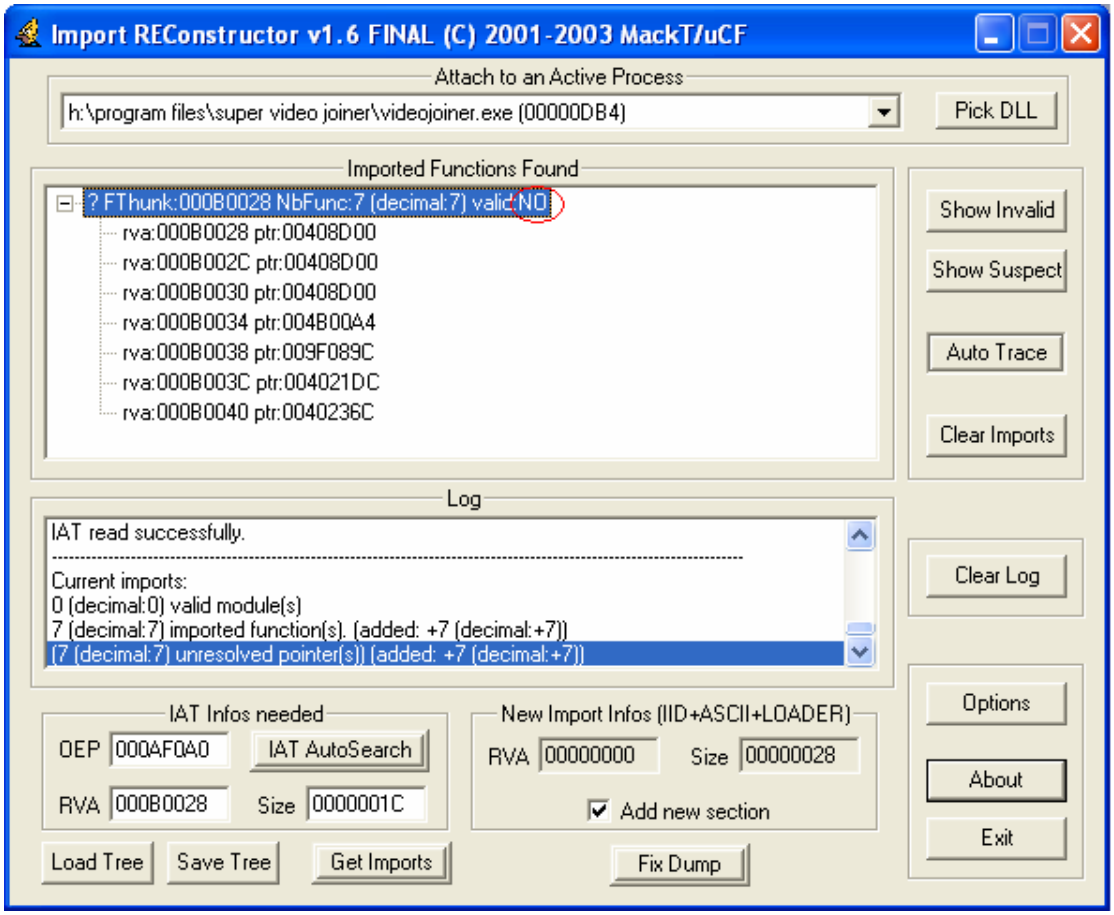
توجه داشته باشید که گزینه Rebuild Import غیر فعال باشد و به این نکته توجه داشته باشید که OEP در آدرس AF0A0 میباشد.
خب حال بر روی دکمه dump کلیک کرده و فایل dump شده را ذخیره کنید من برای راحتی کار نام این فایل را dump.exe میگذارم.
حال برنامه اصلی بنام videojoiner.exe را اجرا کنید و بعد از آن ImpREC را اجرا کنید و از لیست processها گزینه videojoiner را انتخاب کنید مانند شکل زیر خواهید داشت:



شکل ۷

حال OEP را در textbox مربوط به OEP وارد کنید.

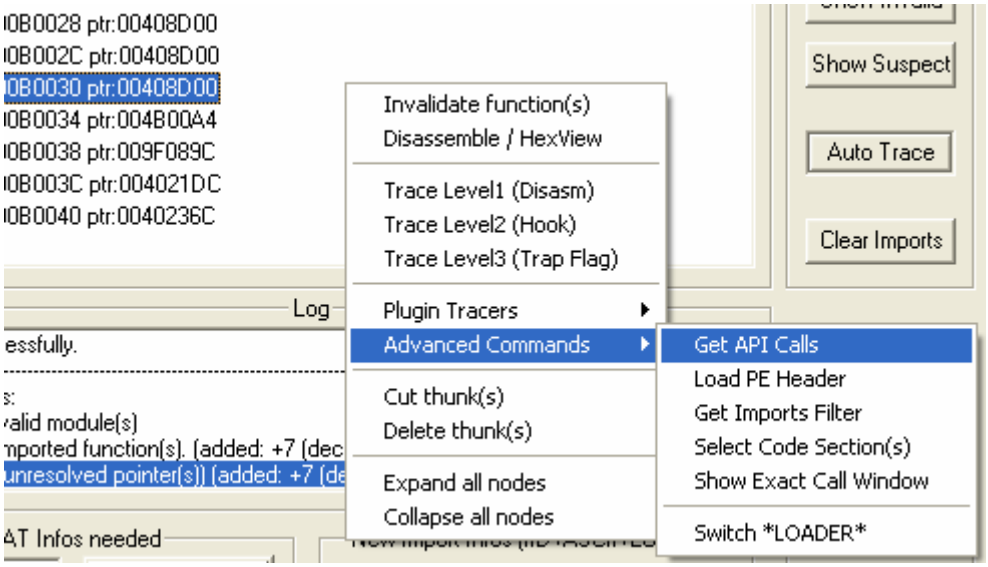
وبعد از آن دکمه IAT AutoSearch را بزنید یک پیغام با این مضمون که OEP درست میباشد خواهید دید. بعد از آن برروی دکمه Get Imports کلیک کنید تا جدول IAT آنرا بدست بیاورید مانند شکل زیر:



شکل ۸

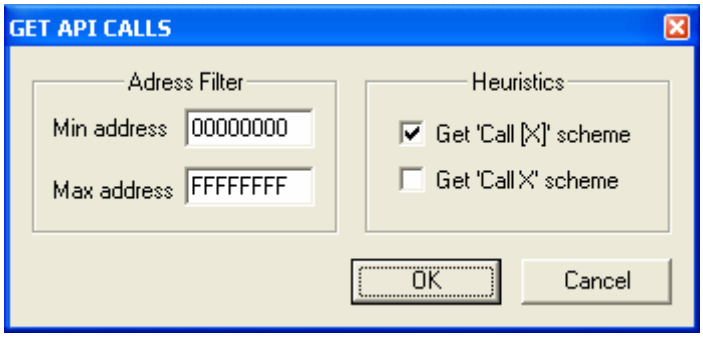
حتماً متوجه خواهید شد که در پانل Imported Functions Found بجای اینکه نوشته شده باشد Valid:Yes نوشته شده است Valid:No این یعنی اینکه IAT بصورت crippled درآمده(cripple در لغت به معنی فلج و معلـول میباشـد) یعنی اینکه این section از فایل اجرایی توسط packer به هم ریخته شده است تا کرکر نتواند راحتتـر برنامـه مـورد نظـر را تغییر دهد.

حال برروی دکمه Show Invalid کلیک کنید تا مانند شکل بالا تمامی ارجاعات نامعتبر را ببینید.حال در یک جای سفید در پانل ImpREC کلید سمت راست موس را زده و از منوی ظـاهر شـده گزینـه Advanced Commands و بعـد از آن گزینه Get Api Calls را انتخاب کنید مانند شکل زیر:



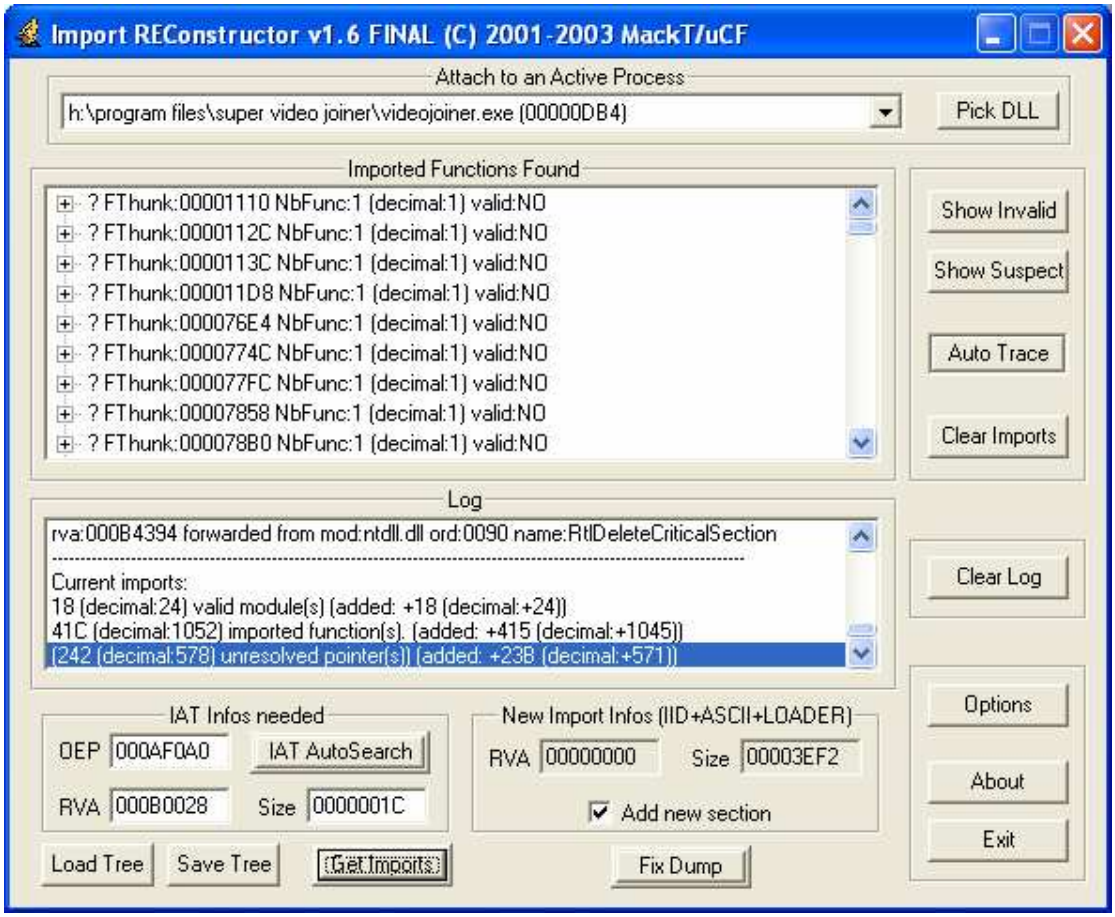
شکل ۹

حال صفحه پدیدار شده به تنظیمات پیش فرض دست نزنید و دکمه Ok را بزنید:



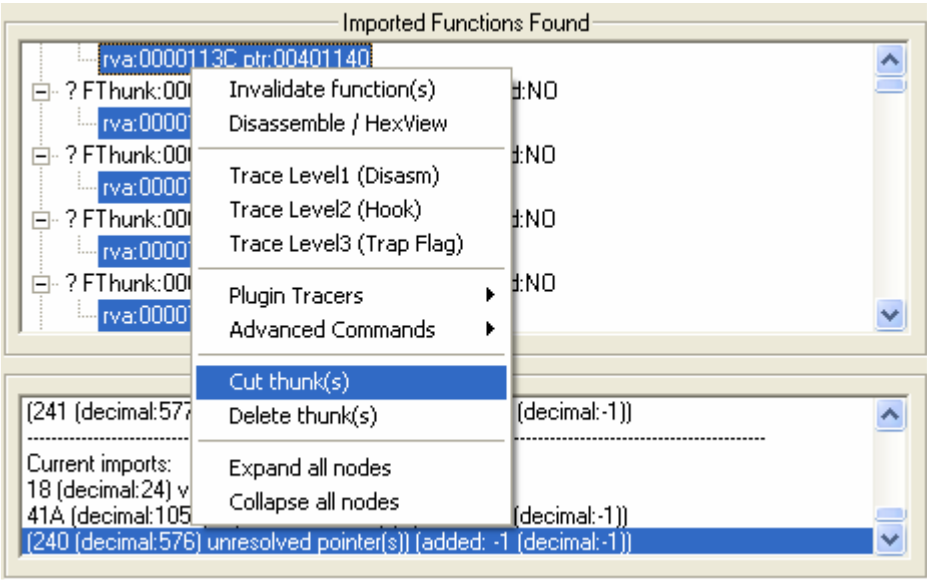
شکل ۱۰

هنگامی که کار تمام شود پانل اصلی ImpREC مانند شکل زیر خواهد شد



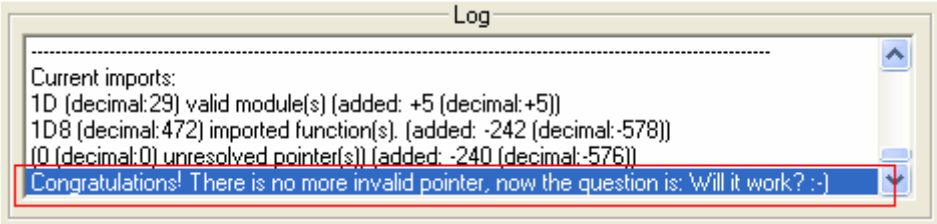
شکل ۱۱

حال یکی بر روی دکمه show invalid کلیک کنید تا تمامی ارجاعات نامعتبر بصورت highlight شده مشخص شده باشد بدون اینکه آنها را از highlight دربیاورید بر روی یکی از آنها دکمه سمت راست را زده و از منوی ظاهر شده گزینه Cut Thunk(s) را انتخاب کنید مانند شکل:



شکل ۱۲

خب با اینکار تمامی ارجاعات نا معتبری که در IAT موجود بود را حذف کردیم در واقع این ارجاعات باعث میشوند که برنامه dump شده نتواند اجرا شود.
حال دوباره برروی دکمه Show Invalid کلیک کنید میبینید که دیگر ارجاع نامعتبری وجود ندارد و در پانل Log برنامه ImpREC نوشته ای مانند زیر خواهید دید:



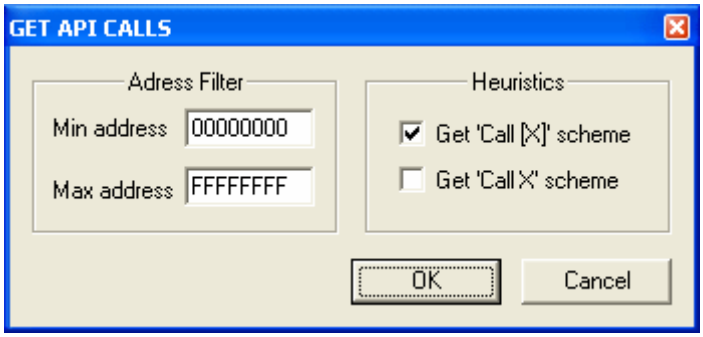
شکل ۱۳

حال برروی دکمه dump.exe کلیک کنید و در پنجره باز شده برنامه dump.exe را انتخاب کنید ImpREC فایلی جدیدی بنام dump_.exe خواهد ساخت.آنرا اجرا کنید ،بله شما موفق شدید برنامه را بصورت کامل unpack کنید.

توجه:
همانطور که قبلاً گفته شده چون نسخه برنامه ضمیمه شده با نسخص اصلی کمی فرق میکند در این نسخه در ویندوز XP این اتفاق نخواهد افتاد و در واقع اگر شما Invalids Callها را اگر حذف کنید برنامه به هیچ عنوان اجرا نخواهد شد پس در ویندوز XP از قسمت Cut Thunk(s) صرف نظر کنید.
این اشکال به این دلیل میباشد که بعضی از ارجاعات نامعتبر برای توابع API نبوده و در واقع داخلی بوده و هیچگونه ربطی به IAT ندارد و اگر این ارجاعات حذف شود در واقع بخشی از برنامه اصلی از بین رفته و برنامه اجرا نخواهد شد(نویسنده اصلی).

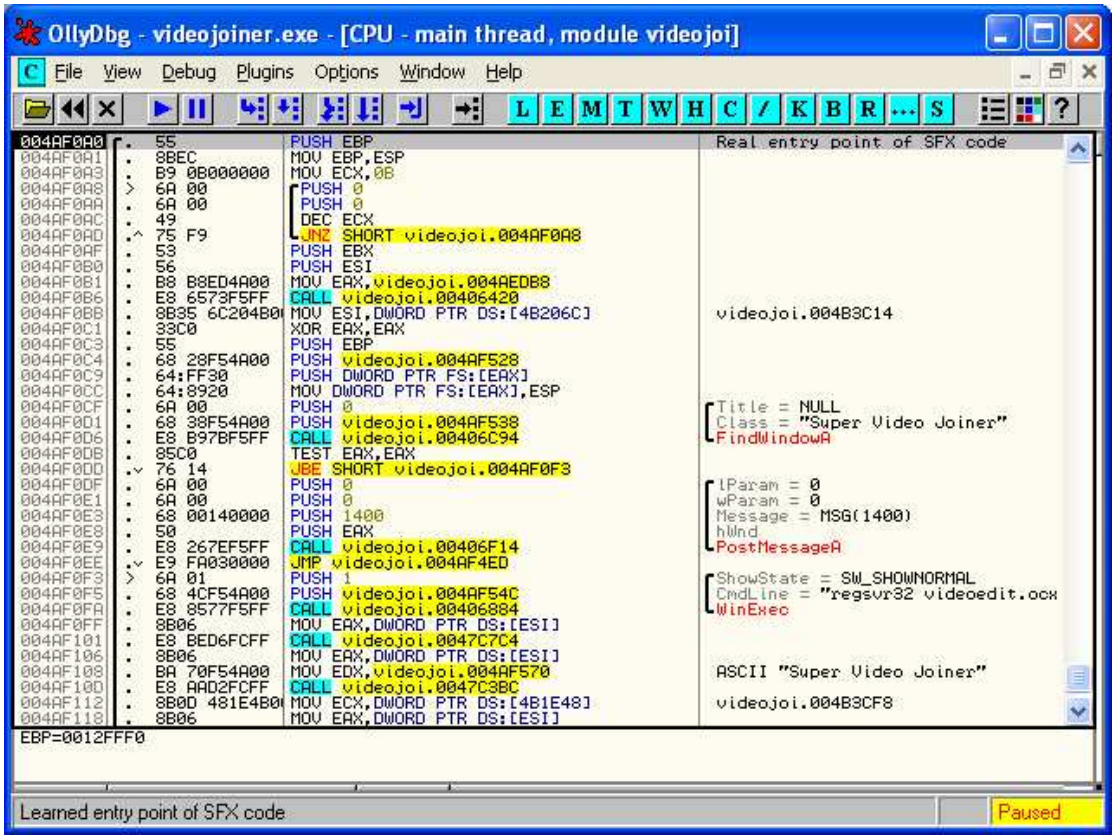
ImpREC چگونه کار میکند؟

خب در بالا چگونگی Unpack کردن یک برنامه را یادگرفتید ولی در واقع خودتان اصلاً متوجه نشدید که چرا در ImpREC چه گذشت و چرا اصلاً ما میبایستی کارهای اضافی در ImpREC انجام میدادیم خب دوباره به شکل زیر نگاهی بیاندازید:



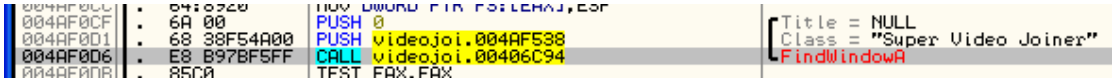
شکل ۱۴

در این دایلوگ به ImpREC گفتیم که از آدرس 00000000 تا آدرس FFFFFFFF را بدنبال تمامی ارجاعات API بگردد البته بصورت Call [X](به سمت راست تصویر توجه کنید).
این یعنی چه؟
برای فهم بیشتر به ollydbg برگردید شما باید قطعاً در آدرس AF0A0 باشید مانند شکل زیر:



شکل ۱۵

به آدرس AF0D6 توجه کنید(برای راحتی شما شکل را بزرگتر کرده ام):



شکل ۱۶

خب میبینید که نوشته شده است :

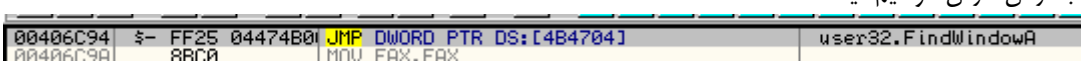
```
CALL videojoi.00406C94
```

و جلوی آن در قسمت توضیحات ollydbg نوشته شده است FindWindowA که مشخص میکند یک تابع API میباشد.

حال به آدرس 00406C94 یعنی همان آدرسی که در جلوی دستور Call وجود دارد بروید

(برای رفتن به آدرسی در فایل میتوانیم از دکمه  استفاده کنیم و در پنجره باز شده آدرس خود را وارد کنید تا دقیقاً ollydbg ما را به آن آدرس ببرد)

خب در آن آدرس خواهیم دید:



شکل ۱۷

دستور JMP در آدرس 406C94 پرش مستقیم فایل user32.dll برای تابع FindWindowA خواهد بود اگر کمی دقت کنید خواهید دید که در این قسمت اینگونه پرشها بسیار زیاد هست.

جمع بندی:
ما یک دستور Call در آدرس 4AF0D6 داریم این Call خود به آدرس 406C94 اشاره دارد(در واقع پرشی است بـه آن آدرس)
و در آدرس 406C94 یک دستور پرش به آدرسی در حافظه بصورت زیر داریم:

```
JMP DWORD PTR DS:[4B4704]
```

در واقع مجموع دستورات بالا مانند دستور زیر میباشد:

```
CALL [address_containing_the_real_addr_to_call]
```

که در قسمت آدرس میتوان همان 4B4704 را نوشت.

با این توضیحات حالا شما میدانید که یک دستور CALL که بخواهد به یک تابع API ارجاع کند دستوری مانند بالا خواهـد بود.
دلیل اینکه در ImpREC در سمت راست گزینه scheme Call[x] انتخاب شده است به این خاطر میباشد.
و منظور از اینکه در قسمت address filter میخواهیم از 00000000 تا FFFFFFFF را جستجو کنـد بـه ایـن خـاطر میباشد که میخواهیم تمامی کد را بدنبال دستورات مورد نظر باشد اگر شما واقعاً میدانید کـه در چـه قـسمتی از کـد میخواهیـد ارجاعات را اصلاح نمایید میتوانید در قسمت address filter مشخص کنید.
خب حال بگذارید کمی عمیق تر به مبحث ارجاعات نگاه بیاندازیم.
برای شروع بیایید نگاهی به نحوه ارجاع به یک تابع API بیاندازیم:
دو روش برای صداکردن(ارجاع) توابع API موجود میباشد.

۱. روش موثر و کارآمد(efficient)
۲. روش ناکارآمد(inefficient)

در روش کارآمد یک ارجاع مانند زیر میباشد:

```
CALL DWORD PTR [0x00405030]
```

اگر با اسمبلی x86 آشنایی ندارید دستور بالا به این معنا میباشد که ارجاع به یک اشاره گر(pointer) که مقدار آن Dword میباشد، در حافظه اشاره میکند. این آدرس هر چه میخواهد باشد یک آدرس IAT و مربوط به توابع API میباشد.

در روش ناکارآمد یک ارجاع بصورت زیر میباشد:

```
CALL 0x0040100C
........
0x0040100C:
JMP       DWORD PTR [0x00405030]
```

در این روش دستور CALL به یک آدرس در درون برنامه اشاره میکند هنگامی که برنامه به آدرس موردنظر پرش میکنـد(در واقع وارد یک stub میشود جایی که تمامی ارجاعات IAT در آنجا قرار دارد) از آنجا به تابع API مورد نظر پرش میکند همانند برنامه ای که هم اکنون در مورد آن بحث میکنیم.
در واقع در روش ناکارآمد برنامه برای صدا کردن یک تابع API، ۵ بایت بیشتر فضا اشـغال میکنـد و زمـان بیـشتری را بـرای اجرای آن تابع از CPU میگیرد.

ولی دلیل بسیار خوبی برای اینکه برخی از کامپایلرها از آن بهره میگیرند وجود دارد و آنهم اینکه کامپایلر از این راه میتواند بین ارجاعات داخلی و ارجاعات به توابع API فرق گذاشته و آنها را شناسایی کند.

```
CALL XXXXXXXX
```

در این مثال XXXXXXXX یک آدرس واقعی است که بوسیله linker پر خواهد شد.توجه داشته باشید که این یک ارجاع به توابع API نمیباشد بلکه یک آدرس واقعی در درون برنامه بوده. برای اینکه کامپایلر بتواند بین ارجاعات بین برنامه و ارجاعات IAT تفاوت قائل شود مجبور است یک مکانی را بنام stub در درون فایل اجرایی در نظر بگیرد و آدرسهای واقعی را در آن مکان تبدیل به اشاره گرهای حافظه کرده و از توابع API استفاده کند.
خب حال این پرش به اشاره گر از کجا آمده است؟
اگر به خاطر داشته باشید IAT فقط یک جدول از آدرسهای حافظه بوده که در آنها توابع API مربوطه آدرس دهی شده اند، که با صدا کردن هریک از آنها ما به یکی از توابع API دسترسی خواهیم داشت.با فرض این موضوع به راحتی خواهیم فهمید که پرشهایی که در درون stub قرار دارد به کدام بخش پرش خواهند کرد.

سوال بعدی که پیش می آید اینست که جطور میتوانیم حالت ناکارآمد را بصورت حالت کارآمد بهینه کنیم؟

جواب به روش راهنمایی شما به کامپایلر بر میگردد.
تابع __declspec(dllimport) تابعی میباشد که به کامپایلر میگوید ارجاع مورد نظر در درون یک فایل کتابخانه‌ای دیگر میباشد و کامپایلر باید دستوری مانند زیر تولید کند:

```
CALL DWORD PTR [XXXXXXXX]
```

به جای اینکه بدین صورت عمل کند:

```
CALL XXXXXXXX
```

به این صورت کامپایلر بصورتی فرض را قرار میدهد که تمامی توابعی که به صورت :

```
__imp_functionname
```

تعریف میشوند به این معنا هست که در فایل کتابخانه ای دیگر بوده و جزو IAT محسوب خواهند شد و دیگر کامپایلر برای استفاده از آنها به stub نیازی ندارد.

نتیجه اینکه اگر شما میخواهید از کدهای بهینه استفاده کنید به خاطر داشته باشید در فایل header خود از تابع __declspec(dllimport) استفاده کنید مانند مثال زیر:

```
__declspec(dllimport) void Foo(void);
```

Patch کردن برنامه:
در این قسمت به علت ساده بودن عملیات patch فقط به ذکر آدرسهای آن بسنده میکنم. به این دلیل که این آموزش بیش از حد از حالت ساده درآمده و در مورد مباحث پیشرفته صحبت کرده.
در آخر فایل patch شده را با نام dump.patch.exe ذخیره کنید.

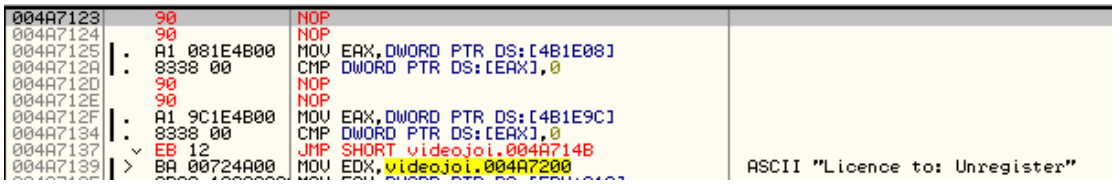
Patch1

Original Code:

```
004A7123 |. /75 14 JNZ SHORT videojoi.004A7139
004A7125 |. |A1 081E4B00 MOV EAX,DWORD PTR DS:[4B1E08]
004A712A |. |8338 00 CMP DWORD PTR DS:[EAX],0
004A712D |. |74 0A JE SHORT videojoi.004A7139
004A712F |. |A1 9C1E4B00 MOV EAX,DWORD PTR DS:[4B1E9C]
004A7134 |. |8338 00 CMP DWORD PTR DS:[EAX],0
004A7137 |. |75 12 JNZ SHORT videojoi.004A714B
004A7139 |> \BA 00724A00 MOV EDX,videojoi.004A7200 ;
ASCII "Licence to: Unregister"
```

Patched Code:

```
004A7123 . 90 NOP
004A7124 . 90 NOP
004A7125 . A1 081E4B00 MOV EAX,DWORD PTR DS:[4B1E08]
004A712A . 8338 00 CMP DWORD PTR DS:[EAX],0
004A712D . 90 NOP
004A712E . 90 NOP
004A712F . A1 9C1E4B00 MOV EAX,DWORD PTR DS:[4B1E9C]
004A7134 . 8338 00 CMP DWORD PTR DS:[EAX],0
004A7137 . EB 12 JMP SHORT videojoi.004A714B
004A7139 . BA 00724A00 MOV EDX,videojoi.004A7200 ; ASCII
"Licence to: Unregister"
```



شکل ۱۸

Patch2

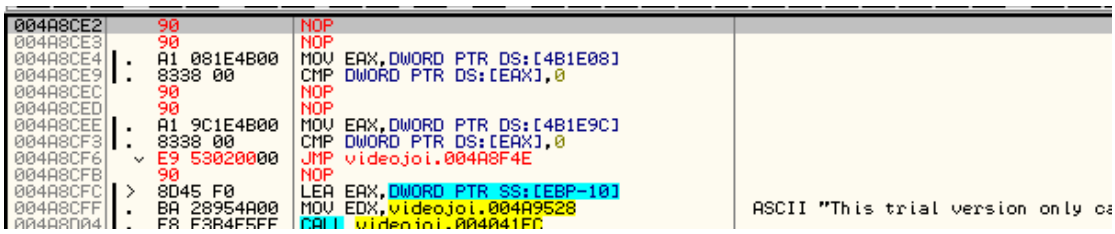
Original Code:

```
004A8CE2 |. /75 18 JNZ SHORT videojoi.004A8CFC
004A8CE4 |. |A1 081E4B00 MOV EAX,DWORD PTR DS:[4B1E08]
004A8CE9 |. |8338 00 CMP DWORD PTR DS:[EAX],0
004A8CEC |. |74 0E JE SHORT videojoi.004A8CFC
004A8CEE |. |A1 9C1E4B00 MOV EAX,DWORD PTR DS:[4B1E9C]
004A8CF3 |. |8338 00 CMP DWORD PTR DS:[EAX],0
004A8CF6 |. |0F85 52020000 JNZ videojoi.004A8F4E
```

```
004A8CFC |> \8D45 F0 LEA EAX,[LOCAL.4]
004A8CFF |. BA 28954A00 MOV EDX,videojoi.004A9528 ; ASCII
"This trial version only can join less than 2 video files
at a time,this limit only can be eliminated when it is
registered!"
004A8D04 |. E8 E3B4F5FF CALL videojoi.004041EC
```

Patched Code:

```
004A8CE2 . 90 NOP
004A8CE3 . 90 NOP
004A8CE4 . A1 081E4B00 MOV EAX,DWORD PTR DS:[4B1E08]
004A8CE9 . 8338 00 CMP DWORD PTR DS:[EAX],0
004A8CEC . 90 NOP
004A8CED . 90 NOP
004A8CEE . A1 9C1E4B00 MOV EAX,DWORD PTR DS:[4B1E9C]
004A8CF3 . 8338 00 CMP DWORD PTR DS:[EAX],0
004A8CF6 . E9 53020000 JMP videojoi.004A8F4E
004A8CFB . 90 NOP
004A8CFC . 8D45 F0 LEA EAX,DWORD PTR SS:[EBP-10]
004A8CFF . BA 28954A00 MOV EDX,videojoi.004A9528 ; ASCII
"This trial version only can join less than 2 video files
at a time,this limit only can be eliminated when it is
registered!"
004A8D04 . E8 E3B4F5FF CALL videojoi.004041EC
```



شکل ۱۹

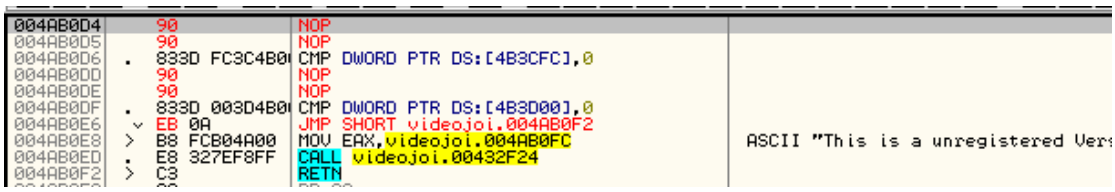
Patch3

Original Code:

```
004AB0D4 . /75 12 JNZ SHORT videojoi.004AB0E8
004AB0D6 . |833D FC3C4B00 00 CMP DWORD PTR DS:[4B3CFC],0
```

```
004AB0DD . |74 09 JE SHORT videojoi.004AB0E8
004AB0DF . |833D 003D4B00 00 CMP DWORD PTR DS:[4B3D00],0
004AB0E6 . |75 0A JNZ SHORT videojoi.004AB0F2
004AB0E8 > \B8 FCB04A00 MOV EAX,videojoi.004AB0FC ; ASCII
"This is a unregistered Version, Don't forget to register
it."
004AB0ED . E8 327EF8FF CALL videojoi.00432F24

Patched Code:
004AB0D4 . 90 NOP
004AB0D5 . 90 NOP
004AB0D6 . 833D FC3C4B00 00 CMP DWORD PTR DS:[4B3CFC],0
004AB0DD . 90 NOP
004AB0DE . 90 NOP
004AB0DF . 833D 003D4B00 00 CMP DWORD PTR DS:[4B3D00],0
004AB0E6 . EB 0A JMP SHORT videojoi.004AB0F2
004AB0E8 . B8 FCB04A00 MOV EAX,videojoi.004AB0FC ; ASCII
"This is a unregistered Version, Don't forget to register
it."
004AB0ED . E8 327EF8FF CALL videojoi.00432F24
```



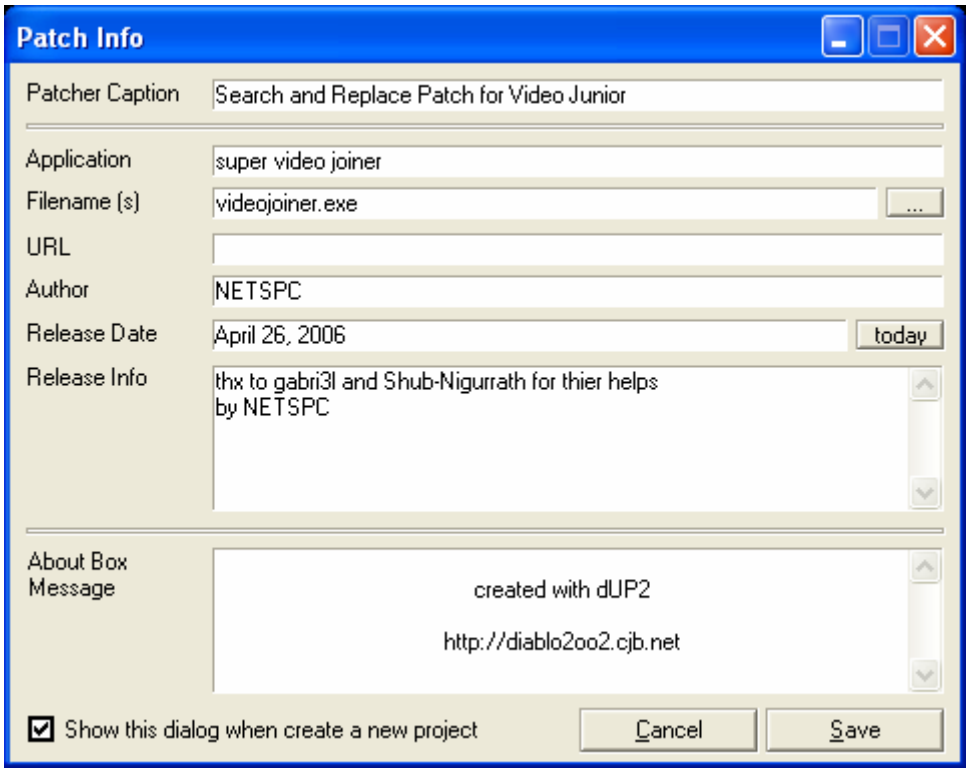
شکل ۲۰

ساخت نرم افزار Patch:

دوباره از نرم افزار diablo2oo2 برای ساخت patch خود استفاده خواهیم کرد ولی در این بخش قصد دارم نحوه ساخت متفاوتی را به شما نشان دهم.
(برای آزمایش خود میتوانید از دستور العملی که در مقاله قبل در مورد ساخت patch آموخته اید استفاده کنید).

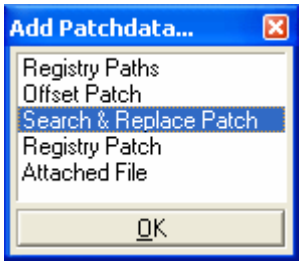
در این بخش میخواهم شما را با روش search and replace آشنا کنم.
یعنی جستجو کردن مقادیر مورد نظر و جایگزین کردن آنها با مقدار دلخواه.اما این روش میتواند بسیار خطرناک و مخرب باشد چرا که اگر pattern(الگو) بصورت غلط تنظیم شود باعث خراب شدن فایل مورد نظر میشود ولی اگر pattern بصورت درست و دقیق تنظیم شود شما با یک بار درست کردن patch میتوانید نسخه های مختلفی از یک نرم افزار را کرک کنید.برای مثال نرم افزار X در حال حاضر نسخه 2.0.1 آن در بازار موجود است ولی به دلایل فنی نسخه 2.05 جایگزین میشود ولی در این نسخه هم از همان روش برای پیگیری شماره سریال و از همان packer استفاده شده است.خب با این اوصاف شما اگر یکبار patch بسازید برای هر دو نسخه کار خواهد کرد.

خب برنامه DUP2 را باز کنید و بر روی دکمه new project کلیک کرده در صفحه ظاهر شده اطلاعات را مانند شکل زیر وارد کنید:



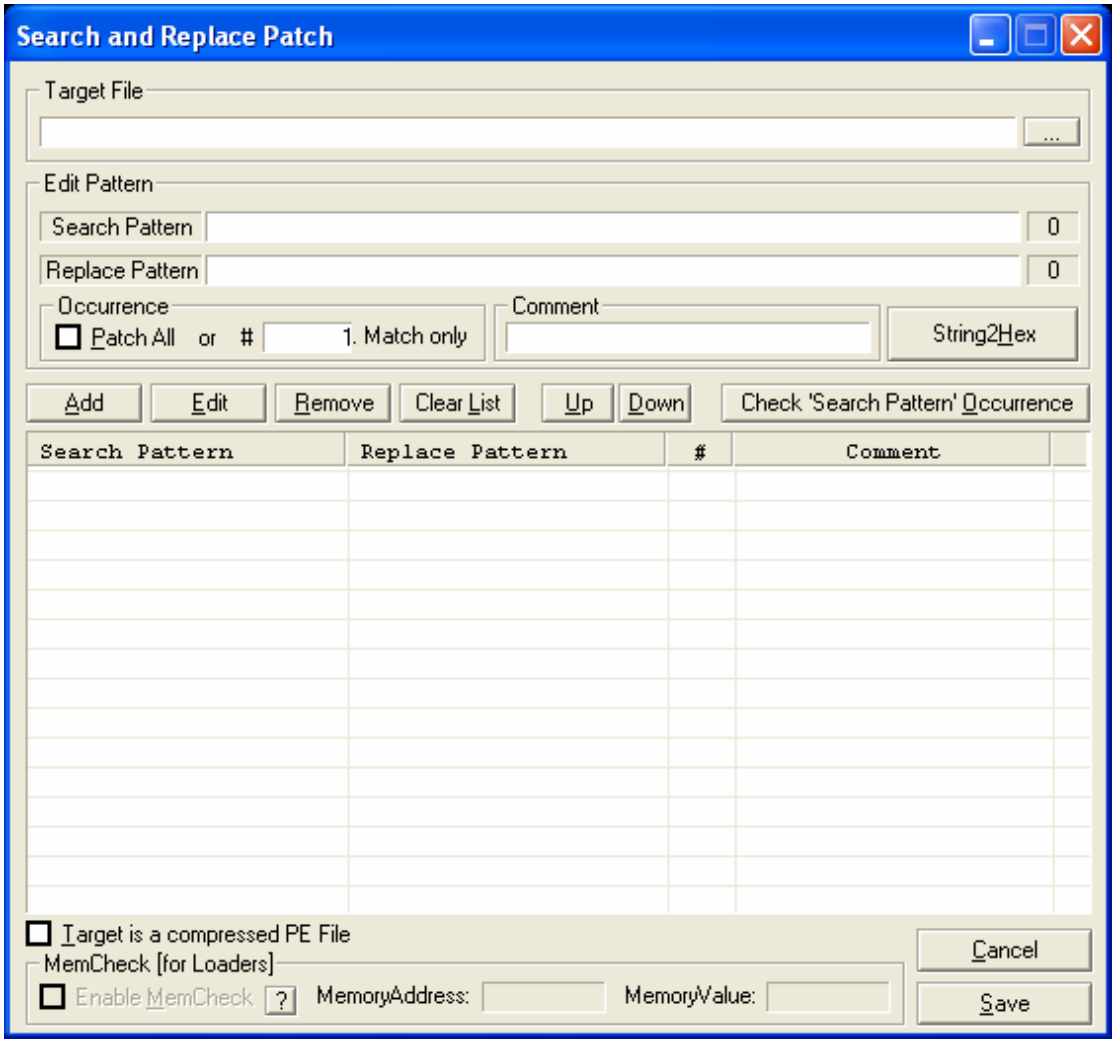
شکل ۲۱

خب بر روی دکمه save کلیک کرده ، حال در صفحه اصلی دکمه add را زده و از صفحه ظاهر شده گزینه search & replace patch را انتخاب کنید مانند شکل زیر



شکل ۲۲

حال در صفحه اصلی گزینه ای اضافه شده است برروی آن دابل کلیک کنید تا صفحه ای مانند شکل زیر باز شود:



شکل ۲۳

شما باید الگو مورد نظر را در textbox مربوط به search pattern وارد نمایید.
در textbox مربوط به Replace pattern الگویی که میخواهید جایگزین شود را وارد نمایید.
بعد بر روی دکمه add کلیک کنید تا الگوهای مورد نظر شما وارد لیست شود.
و بعد از آن برای چک کردن اینکه الگو موردنظر شما در برنامه درست میباشد یا نه بر رئی دکمه check 'Search Pattern' Occurrence کلیک کنید.
و در آخر لیست تغییراتی که باید انجام شود را در پانل اصلی خواهید دید.

قبل از شروع کار میخواهم یک نکته را برای شما واضحتر توضیح بدهم و آن نکته در مورد OpCodeها میباشد.
هر دستور اسمبلی یک معادل دستوری بصورت عدد دارد که به آن OpCode گفته میشود که بصورت hexadecimal نوشته میشود و با تبدیل آن به سیستم دودویی به همان 0و1 های معروف خواهیم رسید!
در واقع زبان ماشین.

```
A1 081E4B00 MOV EAX,DWORD PTR DS:[4B1E08]
```

در اینجا دستور MOV EAX معادل A1 میباشد و باقی عددها پارامترهای این دستور میباشند که بصورت برعکس درآمده اند و اینهم به دلیل نوع پردازنده میباشد(نوع پردازنده را با شرکت سازنده آن اشتباه نگیرید).
خب برای شروع کار من در اینجا patch قسمت اول را میاورم و در باقی کار فقط جوابها را خواهم نوشت برای تمرین خودتان دو patch باقی مانده را حل کنید:

```
004A7123 |. /75 14 JNZ SHORT videojoi.004A7139
004A7125 |. |A1 081E4B00 MOV EAX,DWORD PTR DS:[4B1E08]
004A712A |. |8338 00 CMP DWORD PTR DS:[EAX],0
004A712D |. |74 0A JE SHORT videojoi.004A7139
004A712F |. |A1 9C1E4B00 MOV EAX,DWORD PTR DS:[4B1E9C]
004A7134 |. |8338 00 CMP DWORD PTR DS:[EAX],0
004A7137 |. |75 12 JNZ SHORT videojoi.004A714B
004A7139 |> \BA 00724A00 MOV EDX,videojoi.004A7200 ;
ASCII "Licence to: Unregister"
```

که OpCodeها آنها در بایت بدین صورت میشود:

75,14,A1,08,1E,4B,00,83,38,00,74,0A,A1,9C,1E,4B,00,83,38,00,75,12,BA,00,72,4A,00

توجه داشته باشید که اعدادی که با قرمز مشخص شده اند همان پارامترهای دستورالعملها میباشند.
خب حال بیاید از خود سوال کنیم هنگامی که یک شرکت نسخه جدیدی از یک نرم افزار ارائه میدهد چه چیزهایی از آن تغییر میکند؟
رشته ها ، آدرس افستها و دستورات MOV

خب با این نتیجه تمامی اعداد قرمز را با علامت * جایگزین میکنیم:

75,**,A1,**,**,**,**,83,38,**,74,0A,A1,**,**,**,**,83,38,**,75,**,BA,**,**,**,**

با کمی فکر کردن به پارامترها میبینیم که میتوانیم بعضی از آنها را دستنخورده باقی بگذاریم برای مثال در آدرس 4A7123 در این آدرس ما یک پرش داریم به 14 بایت جلوتر داریم که با کمی تامل درمیابیم که این آدرس ، آدرسی نیست که تغیر بکند چرا که یک آدرس رابطه ای بوده و آدرس مطلقی به شمار نمی آید.
و برای اطمینان از اینکه الگو patch ما در آینده هم خوب کار کند میتوانیم مقدار آدرس 4A7134 را دستنخورده و ثابت نگه داریم تا دستورالعمل مقایسه (CMP) کار خود را انجام دهد.(شاید کمی سخت به نظر بیاید ولی با کمی تکرار و تمرین خواهید توانست مقادیری که باید ثابت بمانند را به درستی حدس بزنید).
اینهم آخرین تغییرات ما در الگو مورد نظر:

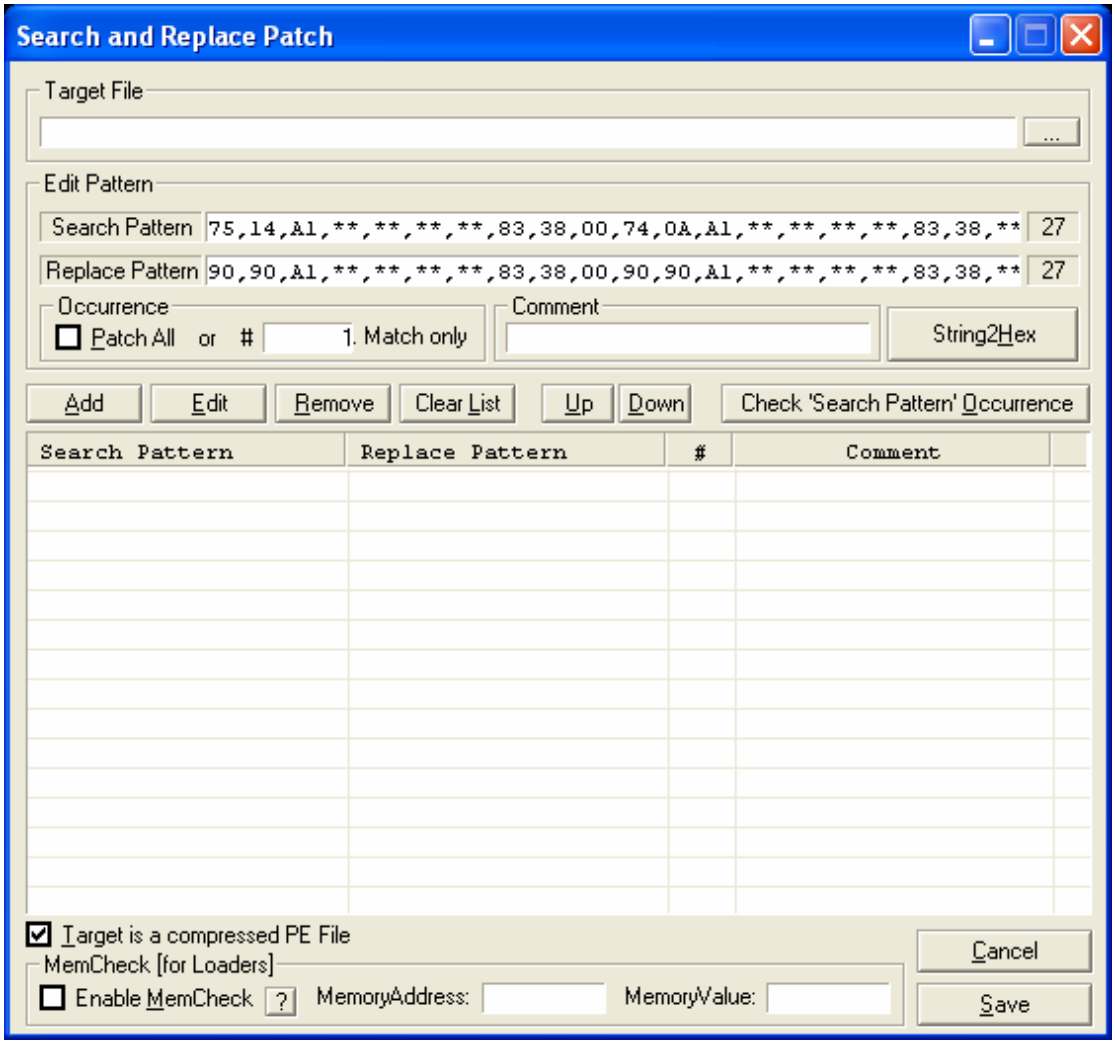
75,14,A1,**,**,**,**,83,38,00,74,0A,A1,**,**,**,**,83,38,**,75,12,BA,**,**,**,**

الگوی جایگزین ماهم از روی patch بدین صورت خواهد بود:

```
004A7123 . 90 NOP
004A7124 . 90 NOP
004A7125 . A1 081E4B00 MOV EAX,DWORD PTR DS:[4B1E08]
004A712A . 8338 00 CMP DWORD PTR DS:[EAX],0
004A712D . 90 NOP
004A712E . 90 NOP
004A712F . A1 9C1E4B00 MOV EAX,DWORD PTR DS:[4B1E9C]
004A7134 . 8338 00 CMP DWORD PTR DS:[EAX],0
004A7137 . EB 12 JMP SHORT videojoi.004A714B
004A7139 . BA 00724A00 MOV EDX,videojoi.004A7200 ; ASCII
"Licence to: Unregister"
```

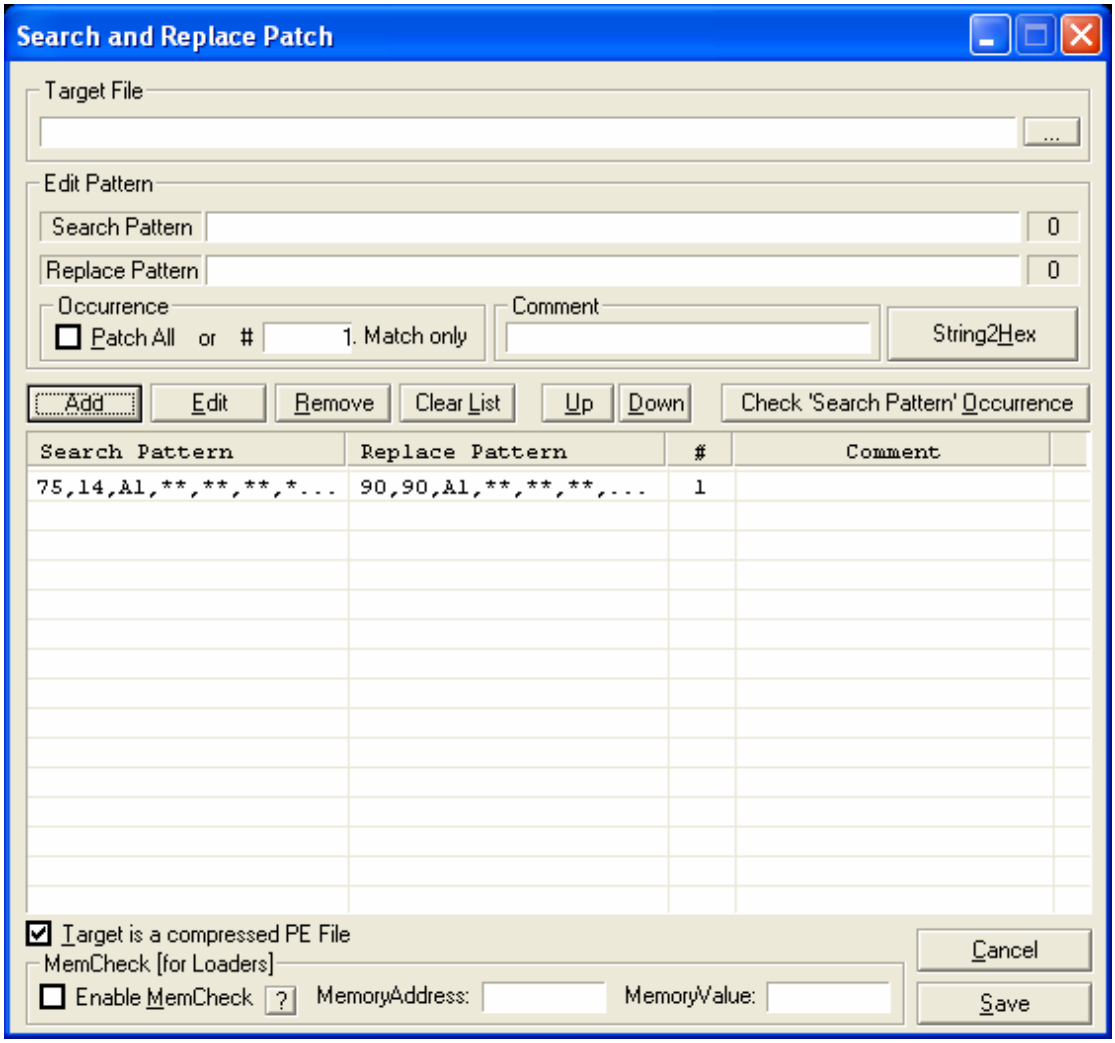
90,90,A1,**,**,**,**,83,38,00,90,90,A1,**,**,**,**,83,38,**,EB,12,BA,**,**,**,**

خب این الگو را در قسمت replace pattern وارد کنید. همانند شکل زیر خواهید داشت:



شکل ۲۴

بعد از اینکه الگوها را در جای مناسب قرار دادید بر روی دکمه Add کلیک کنید تا مانند شکل زیر داشته باشید:



شکل ۲۵

حال دیگر الگوها را بصورت خلاصه در زیر میبینید:

Patch 2
Original byte sequence:
75,18,A1,**,**,**,**,83,38,00,74,0E,A1,**,**,**,**,83,38,00,0F,85,**,**,**,**,8D,45,**,BA,**,**,**,**

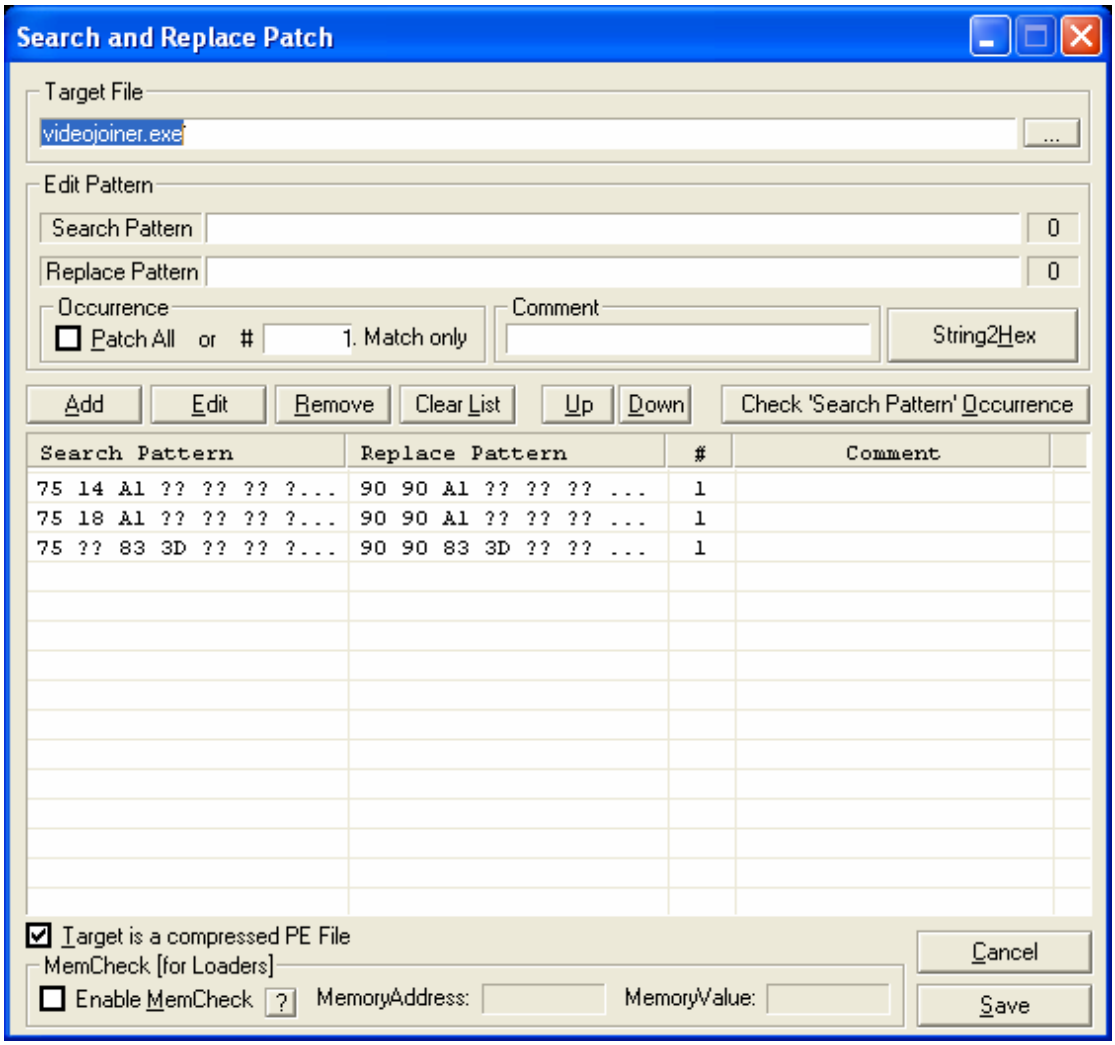
Patched byte sequence:
90,90,A1,**,**,**,**,83,38,00,90,90,A1,**,**,**,**,83,38,00,E9,53,02,00,00,90,8D,45,**,BA,**,**,**,**

Patch 3
Original byte sequence:
75,**,83,3D,**,**,**,**,00,74,09,83,3D,**,**,**,**,00,75,0A,B8,**,**,**,**

Patched byte sequence:
90,90,83,3D,**,**,**,**,00,90,90,83,3D,**,**,**,**,00,EB,0A,B8,**,**,**,**

توجه کنید گزینه target is a compressed PE file فعال باشد.
و در قسمت target file حتماً فایل اصلی برنامه را معرفی کنید و نه فایلهای dump شده را، در غیر اینصورت patch شما کار نخواهد کرد.برای چک کردن pattern خود میتوانید از دکمه ... check استفاده کرده در آنجا pattern مورد نظر را وارد کرده و فایل dump شده را معرفی کنید.

در آخر مانند شکل زیر خواهید داشت:



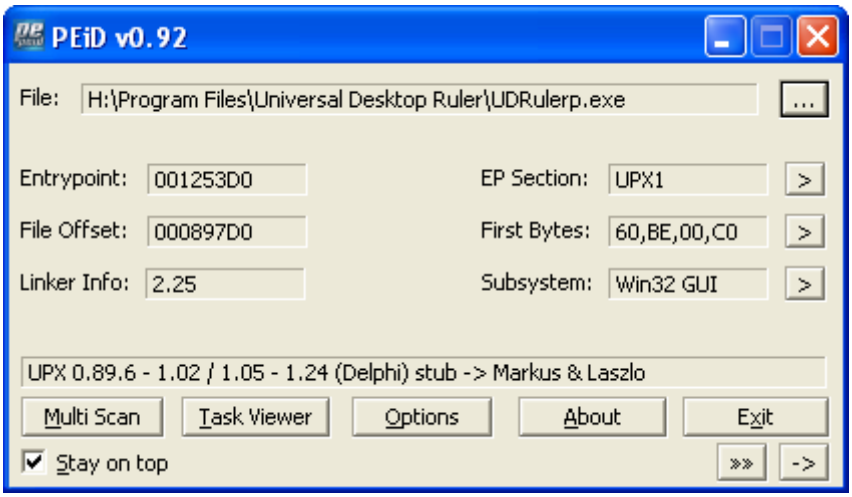
شکل ۲۶

حال دکمه save را زده و در صفحه اصلی هم create patch را بزنید.

پایان

آموزش کرکینگ با ollydbg قسمت پنجم
تاریخ : اردیبهشت ۱۳۸۵ April 2006
نوع حمله Serial,inline patching
نرم افزار هدف : Universal Desktop Ruler 2.5.872
نرم افزارهای مورد استفاده : ollydbg 1.10,PEiD 0.93
سطح : متوسط

خب اول نرم افزار رو نصب میکنیم.
بعد از نصب همیشه اولین کاری که باید بکنیم این هست که اول بفهمیم برنامه رو با چی نوشتن و با چی پک شده برای اینکار میتوانید از نرم افزارهایی مانند PEid و یا RDG Packer Detector استفاده کنید من اینجا از PEid استفاده کردم.
برنامه PEid رو باز کرده و فایل UDRulerp.exeرو بندازید توش و یا از دکمه ...(open) برای تستِ فایل استفاده کنید



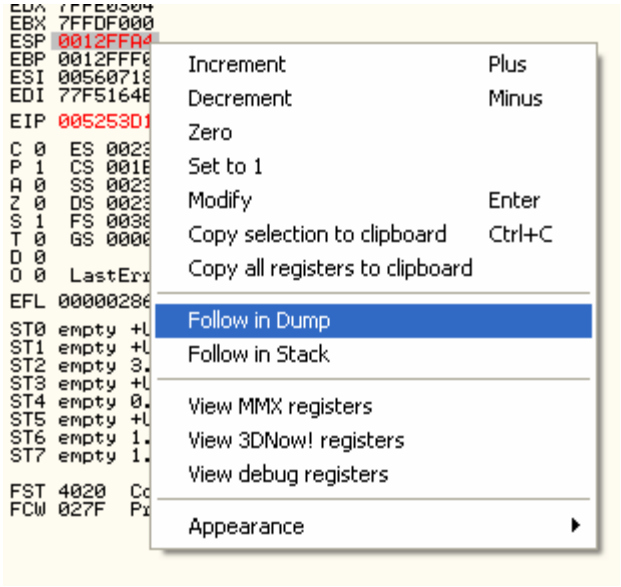
شکل ۱

این برنامه با UPX پک شده است ولی با توجه به مقالات ۳ و ۴ حتماً به خاطر دارید که تمام packerهایی که خاصیت re-entrant را دارا میباشند را میتوان به دو طریق آنپک کرد.
ولی در این آموزش نمیخواهم این برنامه را آنپک کنم ولی اگر مایل هستید برای تمرین بیشتر میتوانید اینکار را انجام بدهید.

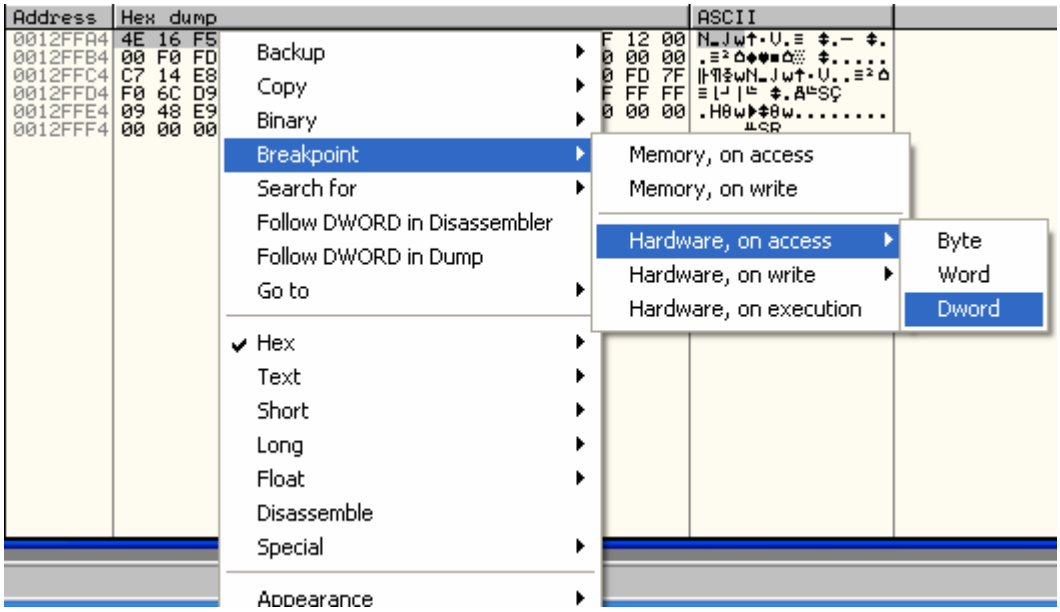
حال برنامه را بوسیله ollydbg باز کنید دو پیغام به شما نشان داده میشود دکمه yes را بزنید تا ollydbg فایل را بـاز کنـد شما مانند شکل زیر در آدرس 005253D0 قرار دارید:

```
005253D0  $  60            PUSHAD
005253D1  .  BE 00C04900   MOV ESI,UDRulerp.0049C000
005253D6  .  8DBE 0050F6F LEA EDI,DWORD PTR DS:[ESI+FFF65000]
```

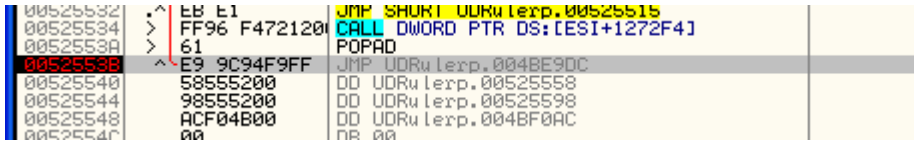
اگر از آموزش شماره ۳ به خاطر داشته باشید گفته شد که اکثر پکرها از تابع pushad استفاده میکنند و در مقابل آن یک تابع popad حتماً باید وجود داشته باشد.برای اینکه متوجه بشویم که پکر چه وقت برنامه را آنپک کرده به OEP پرش میکند برنامه را با F8 اجرا کرده(یکبار F8 را فشار دهید)به پنجره Stack توجه کنید میبینید که مقدار ثبات ESP تغییر پیدا کرده است(به رنگ قرمز درآمده است) خب بر روی مقدار ESP دکمه سمت راست موس را زده و از منوی ظاهر شده گزینه Follow in Dump را انتخاب کنید:



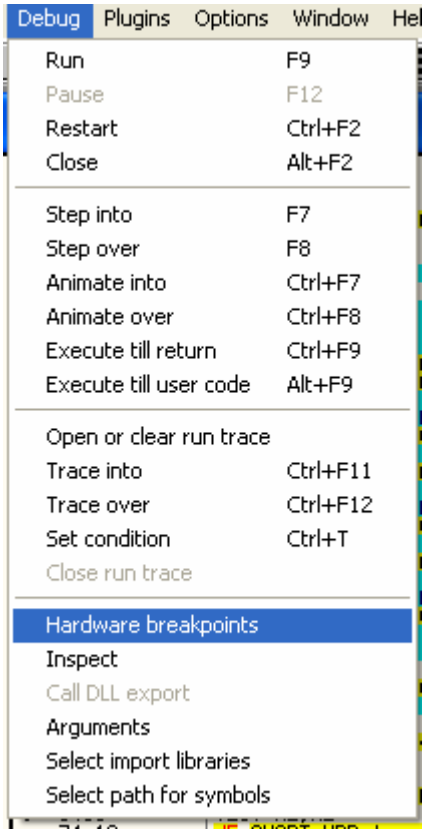
در پنجره dump شما بر روی مقدار آدرس 0012FFA4 قزار خواهید گرفت حال ۴ بایت اول را highlight کنید و دکمه سمت راست موس را زده و گزینه breakpoint -> hardware on access -> DWord را انتخاب کنید



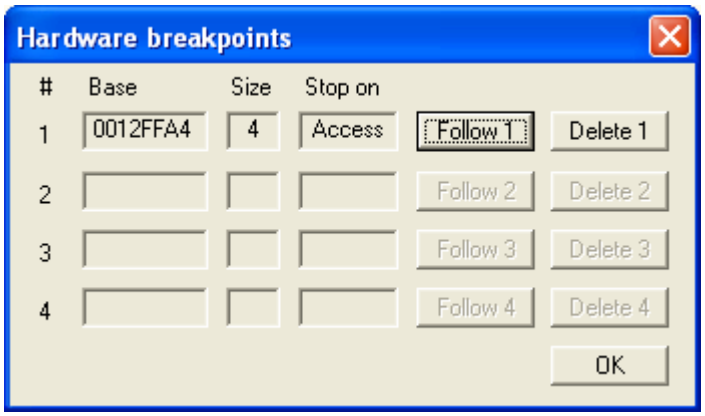
حال برنامه را با زدن کلید F9 اجرا کنید بعد از مدت زمان کوتاهی برنامه در آدرس 52553B متوقف میشود این نقطه همان جایی هست که ما به OEPخواهیم رفت مانند شکل زیر:



در اینجا باید breakpoint مربوط به پشته را حذف کنیم برای اینکار به منوی debug رفته و گزینه hardware breakpoint را انتخاب کنید



یک پنجره باز خواهد شد:

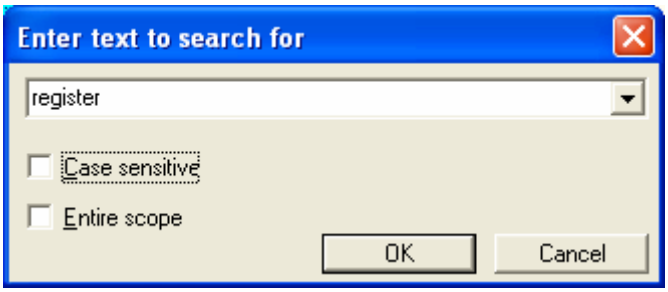


در این پنجره دکمه delete را بزنید تا breakpoint حذف شود.

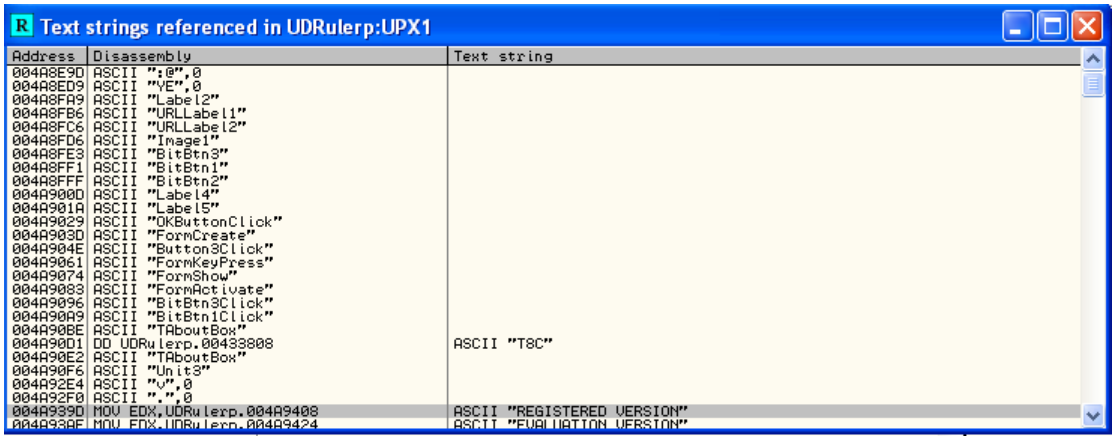
حال یکبار دکمه F8 را فشار دهید شما دقیقاً در OEP (آدرس OEP را یادداشت کنید) قرار دارید.
در این نقطه دکمه ctrl+a را فشار دهید تا کد دوباره از طرف ollydbg آنالیز شود.

بعد از آنالیز شدن کد دکمه سمت راست موس را زده از منوی ظاهر شده گزینه search for ->all referenced text را انتخاب کنید در پنجره ظاهر شده نوار پیمایش را به بالا برده و اولین ردیف را انتخاب کنید و دکمه سمت راست موس را زده

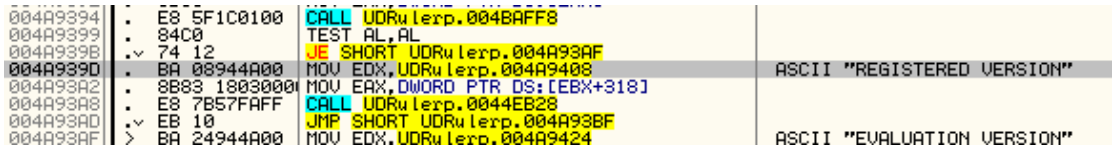
گزینه search را انتخاب کنید در دایلوگ باز شده کلمه register را تایپ کنید و دقت کنید کـه گزینـه case sensitive غیرفعال باشد مانند شکل زیر:



دکمه ok را فشار دهید تا برنامه شما را بر روی خطی که این نوشته وجود دارد راهنمایی کند:



بر روی آن خط دابل کلیک کرده:

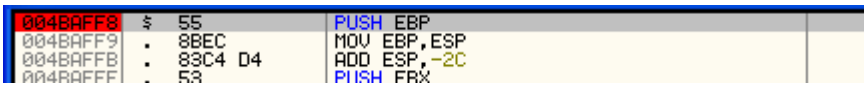


شما در آدرس 4A939D قرار خواهید گرفت.

یک خط بالاتر یک دستور پرش قرار دارد (JE) و قبل از آن دستور یک دستور TEST میبینید که در این خط دستور TEST دو مقدار nibble مربوط به AL را مقایسه میکند و اگر مقدار AL برابر با ۱ شود یعنی اینکه برنامه ثبت شده است و اگر ۰ شود یعنی برنامه ثبت نشده است.خب پس ما باید جایی را پیدا کنیم که این مقدار را همیشه ۱ نگه داریم.

قبل دستور TEST یک ارجاع(CALL) را میبینیم این به این معنا میباشد که برای چک کردن شماره سریال در اینجا ما با یک روتین مواجه هستیم که مقدار AL را مشخص میکند. بر روی آن ارجاع رفته یعنی در آدرس 4A9394 و کلید enter را بزنید تا به روتین مورد نظر برویم.

حال در آدرس 4BAFF8 یعنی خط اول روتین کلید F2 را بزنید:



حال برنامه را با زدن کلید F9 اجرا کنید.

برنامه اجرا میشود و یک ICON در قسمت taskbar قرار میگیرد بر روی آن ICON دکمه سمت راست موس را زده ollydbg شما را در آدرس 4BAFF8 متوقف میکند. خب چون ما میدانیم که مقدار AL باید برابر با ۱ شود و اینکار در اکثر مواقع چون در آخر برنامه انجام میشود به زدن کلید به آخر روتین خواهیم رفت:

```
004BB325  .  C3            RETN
004BB326  .- E9 9D8FF4FF   JMP UDRuler.004042C8
004BB32B  .^ E9 5BFFFFFF   JMP UDRuler.004BB28B
004BB330  .  33C0          XOR EAX,EAX
004BB332  .  5A            POP EDX
004BB333  .  59            POP ECX
```

میبینید که بر روی دستور RETN توقف خواهیم کرد خب این دستور به کجا باز خواهد گشت؟
برای پیدا کردن جواب این سوال به پنجره stack توجه کنید:

```
0012FC38  004BB330  UDRuler.004BB330
```

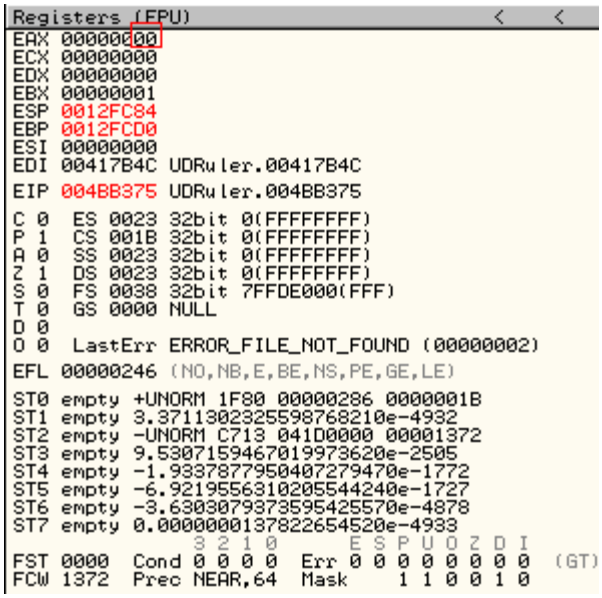
به آدرس 4BB330 یعنی ۳ خط پایین تر خب دوباره با زدن کلید F8 برنامه را دنبال کنید تا دستور RETN بعدی برسید و دوباره به پنجره stack توجه کنید، متوجه خواهید شد که آدرس بعدی هم چندین خط پایینتر قرار دارد یکبار دیگر دکمه F8 را زده تا در آخرین روال قرار گیرید:

```
004BB36D  .  8BC3          MOV EAX,EBX
004BB36F  .  5F            POP EDI
004BB370  .  5E            POP ESI
004BB371  .  5B            POP EBX
004BB372  .  8BE5          MOV ESP,EBP
004BB374  .  5D            POP EBP
004BB375  .  C3            RETN
```

توجه کنید این آخرین روالی است که برای چک کردن شماره سریال استفاده میشود برای اینکه بفهمید که این آخرین روال هست یا نه دو راه دارید:

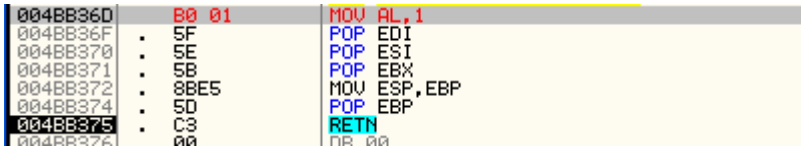
- یک راه آزمایش و خطاست در واقع میتوانید با زدن کلید F8 این روال را هم اجرا کنید و بعد از رسیدن به RETN خواهید دید که بعد از این روال برنامه از جایی که شماره سریال باید تست میشد اجرا میشود.
-
- راه دیگر توجه به پنجره stack میباشد،چرا که بعد از خاتمه روالها همیشه پشته مانند قبل خواهد شد در واقع همانند وقتی که هنوز به روالهای دیگر وارد نشده بود در اینجا چند دستور آزادسازی ثبات داریم (POP EDI,ESI,EBX) و بعد از آن مقدار ثبات EBP به داخل مقدار ثبات ESP ریخته میشود و ثبات ESP همیشه آدرس بالایی پشته را در خود نگهداری میکند.

در این روال شما میبینید که ثبات EAX با مقدار ثبات EBX پر میشود و اگر دقت کنید میبینید که چون ما شماره سریال نداریم و اگر به خاطر داشته باشید یادتان هست که مقدار نیم ثبت AL تست میشد که آیا 0 است و یا 1 ، با توجه به مقدار ثبات EAX در پنجره register میبینید که مقدار آخر ثبات EAX صفر است یعنی اینکه شماره سریال نادرست است:



خب کاری که ما باید انجام دهیم اینست که همیشه این مقدار را 1 نگه داریم برای اینکار در آدرس 4BB36D دستور
Mov al,1

را جایگزین دستور جاری میکنیم مانند شکل زیر:



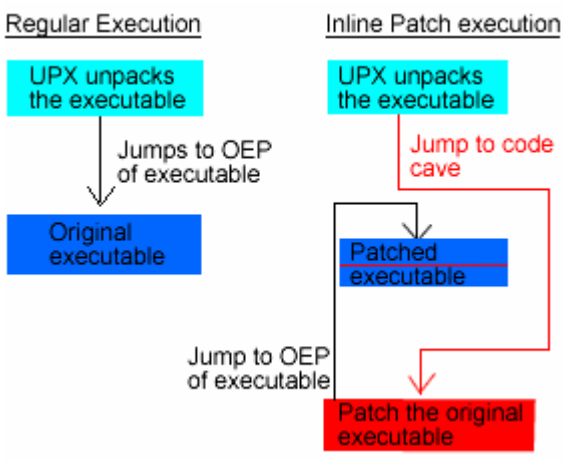
چون در این آموزش قصد دارم نوع دیگر کرک کردن را به شما آموزش دهم کار ما در اینجا خاتمه نمیابد در اینجا آدرسی که دستور تغییر کرده است و مقدار opcode آنرا یادداشت کنید مانند زیر توجه کنید هر opcode یک آدرس از حافظه را اشغال میکند.پس با توجه به تغییرات بالا ما باید دو آدرس را یادداشت کنیم یعنی:

4BB36D B0
4BB36E 01

حال برنامه را با زدن کلید ctrl+F2 ریستارت کنید.

درباره Inline Patching:

Patch کردن درون خط (inline) منظور تغییر کد برنامه بدون عملیات dumping میباشد در واقع ما برنامه را همانطور بصورت pack شده نگه میداریم و فقط تغییرات خود را در آن اعمال میکنیم.
برای اینکار ما باید برنامه را جوری تغییر دهیم که اول به کدهایی که ما تغییر داده ایم پرش کند و بعد از آن به سمت OEP



در واقع ما برای اینکه کدهای خود را در درون برنامه pack شده قرار دهیم باید در درون برنامه جایی که دستورات خاصی وجود ندارد را پیدا کرده و کدهای patch خود را در آنجا قرار دهیم که به اینکار اصطلاحاً code cave میگویند.
اما پیدا کردن جایی مناسب برای code cave ها ممکن است مشکل بنظر برسد که با تجربه و تلاش بیشتر میتوان بر این مشکل فارغ آمد.

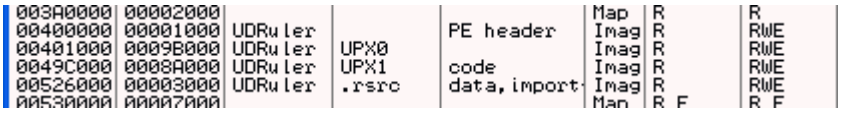
عملیات inline patching:
در ollydbg ازمنوی view گزینه memory را انتخاب کنیدو یا ctrl+m را فشار دهید :

Memory map

Address	Size	Owner	Section	Contains	Type	Access	Initial	Mapped as
00010000	00001000				Priv	RW	RW	
00020000	00001000				Priv	RW	RW	
0012B000	00001000				Priv	RW Gua	RW	
0012C000	00004000			stack of ma	Priv	RW Gua	RW	
00130000	00001000				Map	R	R	
00140000	00002000				Map	R	R	
00150000	00005000				Priv	RW	RW	
00250000	00006000				Priv	RW	RW	
00260000	00001000				Map	RW	RW	
00270000	00016000				Map	R	R	\Device\HarddiskVolume3\WINDOWS\system3
00290000	00034000				Map	R	R	\Device\HarddiskVolume3\WINDOWS\system3
002D0000	00041000				Map	R	R	\Device\HarddiskVolume3\WINDOWS\system3
00320000	00006000				Map	R	R	\Device\HarddiskVolume3\WINDOWS\system3
00330000	00001000				Priv	RWE	RWE	
00340000	00004000				Priv	RW	RW	
00350000	00003000				Map	R	R	\Device\HarddiskVolume3\WINDOWS\system3
00360000	00008000				Priv	RW	RW	
00370000	00001000				Priv	RW	RW	
00380000	00001000				Priv	RW	RW	
00390000	00002000				Map	R	R	
003A0000	00002000				Map	R	R	
00400000	00001000	UDRuler		PE header	Imag	R	RWE	
00401000	0009B000	UDRuler	UPX0		Imag	R	RWE	
0049C000	0008A000	UDRuler	UPX1	code	Imag	R	RWE	
00526000	00003000	UDRuler	.rsrc	data,import	Imag	R	RWE	
00530000	00007000				Map	R E	R E	
005F0000	00002000				Map	R E	R E	
00600000	00103000				Map	R	R	
00710000	000D8000				Map	R E	R E	
00A10000	00001000				Priv	RW	RW	
629C0000	00001000	LPK		PE header	Imag	R	RWE	
629C1000	00004000	LPK	.text	code,import	Imag	R	RWE	
629C5000	00001000	LPK	.data	data	Imag	R	RWE	
629C6000	00001000	LPK	.rsrc	resources	Imag	R	RWE	
629C7000	00001000	LPK	.reloc	relocations	Imag	R	RWE	
70A70000	00001000	SHLWAPI		PE header	Imag	R	RWE	
70A71000	0005B000	SHLWAPI	.text	code,import	Imag	R	RWE	
70ACC000	00001000	SHLWAPI	.data	data	Imag	R	RWE	
70ACD000	00002000	SHLWAPI	.rsrc	resources	Imag	R	RWE	
70ACF000	00005000	SHLWAPI	.reloc	relocations	Imag	R	RWE	
71950000	00001000	comctl32		PE header	Imag	R	RWE	
71951000	00088000	comctl32	.text	code,import	Imag	R	RWE	
719D9000	00001000	comctl32	.data	data	Imag	R	RWE	
719DA000	00054000	comctl32	.rsrc	resources	Imag	R	RWE	
71A2E000	00006000	comctl32	.reloc	relocations	Imag	R	RWE	
72FA0000	00001000	USP10		PE header	Imag	R	RWE	
72FA1000	0003A000	USP10	.text	code,import	Imag	R	RWE	
72FDB000	00009000	USP10	.data	data	Imag	R	RWE	
72FE4000	00002000	USP10	Shared		Imag	R	RWE	
72FE6000	00012000	USP10	.rsrc	resources	Imag	R	RWE	

ما جایی را میخواهیم که خالی بوده و بتوانیم کدهای خود را در آنجا قرار دهیم.
در ضمن اینرا هم بدانید پایان هر بخش مساوی است با شروع بخش دیگر. و در پایان هر بخش جاهای خالی بسیار زیادی موجود میباشد که برای اهداف ما مناسب به نظر میرسند.

به شکل زیر توجه کنید:



003A0000	00002000				Map	R	R
00400000	00001000	UDRuler		PE header	Imag	R	RWE
00401000	0009B000	UDRuler	UPX0		Imag	R	RWE
0049C000	0008A000	UDRuler	UPX1	code	Imag	R	RWE
00526000	00003000	UDRuler	.rsrc	data,import	Imag	R	RWE
00530000	00007000				Map	R E	R E

بعد از header فایل اجرایی در آدرس 40000 و بعد از header پکر UPX بخش منابع (rsrc) شروع میشود یعنی آدرس 526000. این قسمت همیشه یکی از بهترین قسمتها برای code cave ها میباشد.

پنجره memory map را ببندید و در پنجره اصلی ctrl+g و یا  را فشار دهید. در پنجره ظاهر شده آدرس 526000 را بزنید تا به بخش منابع بروید مانند شکل زیر:

```
00526064   E7 89            OUT 89,EAX                          I/O command
00526066   42               INC EDX
00526067   3200             XOR AL,BYTE PTR DS:[EAX]
00526069   0000             ADD BYTE PTR DS:[EAX],AL
0052606B   0000             ADD BYTE PTR DS:[EAX],AL
0052606D   0009             ADD BYTE PTR DS:[ECX],CL
0052606F   0001             ADD BYTE PTR DS:[ECX],AL
00526071   0000             ADD BYTE PTR DS:[EAX],AL
00526073   00B8 00008002    ADD BYTE PTR DS:[EAX+28000000],BH
00526079   0000             ADD BYTE PTR DS:[EAX],AL
0052607B   00E0             ADD AL,AH
0052607D   0000             ADD BYTE PTR DS:[EAX],AL
0052607F   8003 00          ADD BYTE PTR DS:[EBX],0
00526082   0000             ADD BYTE PTR DS:[EAX],AL
00526084   0801             OR BYTE PTR DS:[ECX],AL
00526086   0080 04000000    ADD BYTE PTR DS:[EAX+4],AL
0052608C   3001             XOR BYTE PTR DS:[ECX],AL
0052608E   0080 05000000    ADD BYTE PTR DS:[EAX+5],AL
00526094   58               POP EAX
00526095   0100             ADD DWORD PTR DS:[EAX],EAX
00526097   8006 00          ADD BYTE PTR DS:[ESI],0
0052609A   0000             ADD BYTE PTR DS:[EAX],AL
0052609C   8001 00          ADD BYTE PTR DS:[ECX],0
0052609F   8007 00          ADD BYTE PTR DS:[EDI],0
005260A2   0000             ADD BYTE PTR DS:[EAX],AL
005260A4   A8 01            TEST AL,1
005260A6   0080 08000000    ADD BYTE PTR DS:[EAX+8],AL
005260AC   D001             ROL BYTE PTR DS:[ECX],1
005260AE   0080 09000000    ADD BYTE PTR DS:[EAX+9],AL
005260B4   F8               CLC
005260B5   0100             ADD DWORD PTR DS:[EAX],EAX
005260B7   8000 00          ADD BYTE PTR DS:[EAX],0
005260BA   0000             ADD BYTE PTR DS:[EAX],AL
005260BC   E7 89            OUT 89,EAX                          I/O command
005260BE   42               INC EDX
005260BF   3200             XOR AL,BYTE PTR DS:[EAX]
005260C1   0000             ADD BYTE PTR DS:[EAX],AL
005260C3   0000             ADD BYTE PTR DS:[EAX],AL
005260C5   0001             ADD BYTE PTR DS:[ECX],AL
005260C7   0000             ADD BYTE PTR DS:[EAX],AL
005260C9   0000             ADD BYTE PTR DS:[EAX],AL
005260CB   00D0             ADD AL,DL
005260CD   0000             ADD BYTE PTR DS:[EAX],AL
005260CF   0070 74          ADD BYTE PTR DS:[EAX+74],DH
005260D2   0D 00340100      OR EAX,13400
```

خب نوارپیمایش را آنقدر پایین بیاورید تا به سلسله دستورات:

ADD BYTE PTR DS:[EAX],AL

برسید در کامپیوتر من این دستورات از آدرس 00528487 شروع میشود مانند شکل زیر:

```
00528465   0000             ADD BYTE PTR DS:[EAX],AL
00528467   0056 65          ADD BYTE PTR DS:[ESI+65],DL
0052846A v 72 51            JB SHORT UDRuler.005284BD
0052846C v 75 65            JNZ SHORT UDRuler.005284D3
0052846E v 72 79            JB SHORT UDRuler.005284E9
00528470   56               PUSH ESI
00528471   61               POPAD
00528472   6C               INS BYTE PTR ES:[EDI],DX            I/O command
00528473 v 75 65            JNZ SHORT UDRuler.005284DA
00528475   41               INC ECX
00528476   0000             ADD BYTE PTR DS:[EAX],AL
00528478 v 73 6E            JNB SHORT UDRuler.005284E8
0052847A   64:50            PUSH EAX                            Superfluous prefix
0052847C   6C               INS BYTE PTR ES:[EDI],DX            I/O command
0052847D   61               POPAD
0052847E v 79 53            JNS SHORT UDRuler.005284D3
00528480   6F               OUTS DX,DWORD PTR ES:[EDI]          I/O command
00528481 v 75 6E            JNZ SHORT UDRuler.005284F1
00528483   64:41            INC ECX                             Superfluous prefix
00528485   0000             ADD BYTE PTR DS:[EAX],AL
00528487   0000             ADD BYTE PTR DS:[EAX],AL
00528489   0000             ADD BYTE PTR DS:[EAX],AL
0052848B   0000             ADD BYTE PTR DS:[EAX],AL
0052848D   0000             ADD BYTE PTR DS:[EAX],AL
```

به opcode این دستوران توجه کنید : 0000.

خب ما در اینجا باید مقادیری که قبلاً patch کرده بودیم و یادداشت کرده بودیم را بصورت زیر وارد نماییم

MOV BYTE PTR DS:[ADDRESS], DATA

این دستور به این معنا میباشد:DATA که همان مقادیر opcode که یادداشت کرده بودیم را باید در آدرس byte ptr [address]:ds جایگزین کنیم.

یعنی

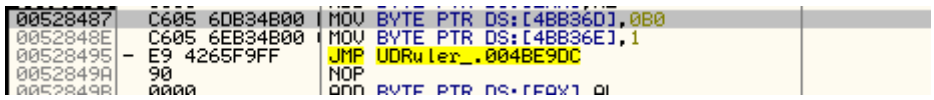
MOV BYTE PTR DS:[004BB36D],0B0
MOV BYTE PTR DS:[004BB36E],01

صفر اول دستور B0 به این دلیل میباشد که اسمبلر بفهمد این یک مقدار میباشد و نه یک دستورالعمل.

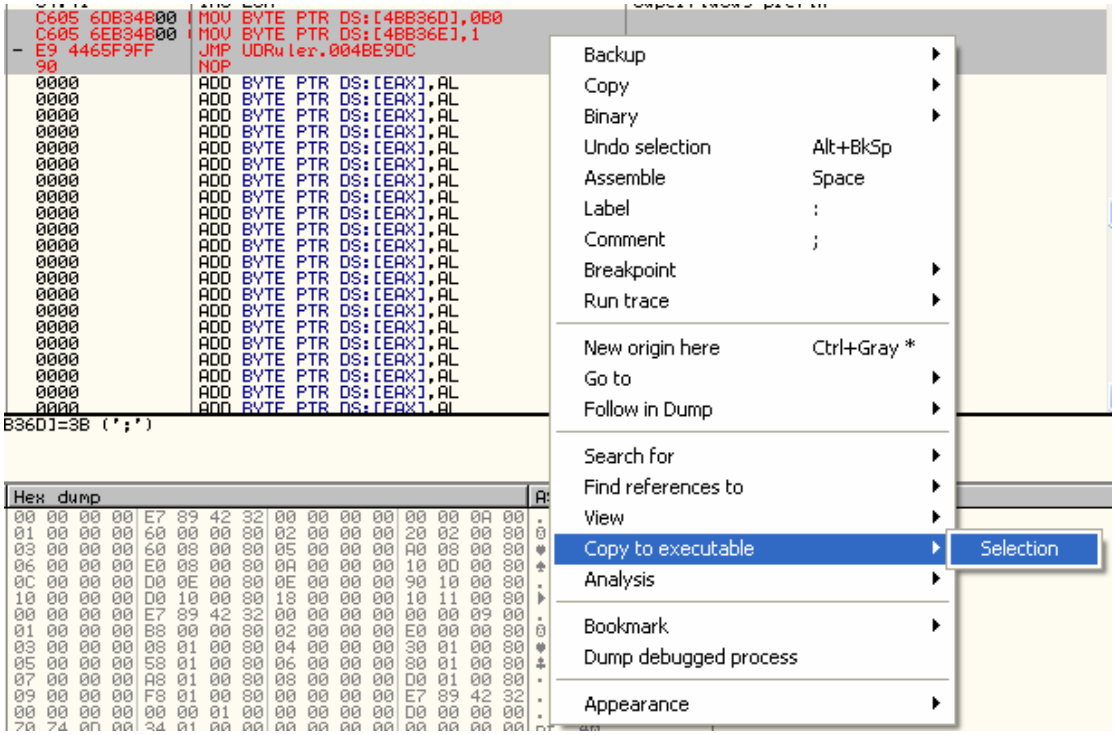
و در آخر هم دوباره باید به OEP پرش کنیم. (آدرس OEP در کامپیوتر من 4BE9DC میباشد) مانند دستور زیر:

JMP 4BE9DC

Code cave ساخته شده مانند شکل زیر خواهد بود:



خب حالا این ۴ دستور(به انضمام NOP) را highlight کرده و مانند مثالهای قبل برنامه را ذخیره کنید.



من فایل را بنام UDRuler_p.exe ذخیره کردم.

حال آخرین کاری که باید انجام دهیم اینست که به آخرین دستور روتین pack مربوط به upx که در اول برنامه قرار دارد رفته و در آنجا دستورات را طوری تغییر دهیم که بجای آنکه بعد از اتمام روال به OEP پرش کند به code cave که ساخته ایم پرش کند.خب اولین آدرس code cave ما 528487 میباشد.

حال برای تغییر دستور مورد نظر فایل UDRuler_p.exe را در ollydbg باز کنید و یا همانند اول آموزش یک breakpoint برروی ESPبگذارید تا به دستور پرش مورد نظر بروید و یا مستقیماً به آدرس پرش یعنی در آدرس 52553B بروید دستور این آدرس را به این دستور تغییر دهید:

JMP 528487

با تغییر این پرش ، باعث میشود برنامه اول به code cave رفته و دستورات آنجا را اجرا کند و بعد به OEP پرش کند خب حال برنامه را مانند قبل ذخیره کنید.

برنامه بصورت ثبت شده اجرا خواهد شد.

پایان.

آموزش کرکینگ با ollydbg قسمت ششم
تاریخ : اردیبهشت ۱۳۸۵ 2006 May
تئوری packerها
نرم افزار هدف : ندارد
نرم افزارهای مورد استفاده : ندارد
سطح : متوسط

مختصری درباره packerها:

Packer برنامه ای هست که برنامه اصلی را بصورت یک لفافه دربر میگیرد و هدف اصلی از انجام اینکار خنثی کردن دستکاری فایل اصلی و یا دیباگ کردن فایل اصلی توسط یک کرکر میباشد.
هنگامی که از یک packer استفاده میکنید،اینکار باعث میشود که OEP به جای دیگر منتقل شود در واقع بعد از برنامه packer و Entry Point مربوط به packer جایگزین شود.
هر packer هنگام اجرا شدن عملیات های زیر را بابت بدست آوردن فایل اجرایی اصلی انجام میدهد:

۱. فایل اصلی را از حالت کد کردن خارج میکند.
۲. Import Table را دوباره بازسازی میکند.
۳. به OEP پرش میکند.

اولین مرحله یعنی decrypt کردن فایل اصلی زیاد مهم نیست اگر از آموزشهای قبلی به خاطر داشته باشید میتوانیم صبر کنیم تا عملیات decrypt کردن تمام شود و آنوقت برنامه را از روی حافظه dump کنیم.
اما عملیات های ۲ و ۳ بسیار مهم میباشند ، هر چه این دو عملیات با دقت و پیچیدگی بیشتری انجام شود عملیات بازگرداندن فایل اصلی مشکل تر و گاه غیر ممکن است. اما در دنیای واقعی بسیاری از packerها از این نظر ضعف دارند.
اما شما از یک packer قوی همیشه باید انتظار اینرا داشته باشید که عملیات بسیاری پیچیده ای برروی IAT انجام دهد.
این عملیات میتواند شامل موارد زیر باشد:

۱. IAT از فایل اصلی حذف شده و packer فقط اسمهای توابع API و آدرس آنها را بصورت درهم(hash) در یک جای امن در حافظه ذخیره کرده است.
۲. از الگوریتم بسیار پیچیده ای برای بدست آوردن IAT استفاده کرده است و این الگوریتم دارای روشهایی برای مقابله با کرکرها میباشد مانند anti-debugوanti-trace.
۳. packer از تابع GetProcAddress استفاده نکرده است و بجای آن از روش خود برای بدست آوردن آدرس توابع API استفاده میکند.
۴. IAT بطور کلی به جای دیگر ارجاع داده شده است.

اما جالب است بدانید که بسیار از packerها تجاری که امروزه استفاده میشود هیچکدام از موارد ۱ و ۳ را رعایت نمیکنند.
در مورد مورد شماره ۴ چه ؟
بیاید ببینیم منظور از ارجاع IAT به جای دیگر یعنی چه؟
هنگامی که یک برنامه را disassemble میکنید و اگر آن برنامه از توابع API استفاده کرده باشد حتماً از تابع kernel32.exitproccess استفاده کرده است (این تابع برای تمام کردن و خروج از یک پردازش بکار میرود) این ارجاع در برنامه بصورت زیر میباشد:

```
call [XXXXXXXXh]      ; call to kernel32.ExitProcess
XXXXXXXXh inside the IAT
  ...                   ;
XXXXXXXXh: YYYYYYYYh  ; address of Kernel32.ExitProcess
```

و یا راه دیگر بدین صورت میباشد:

```
call XXXXXXXXh
```

اما پارامتر همان میباشد(XXXXXXXX).

بارکننده (Loader) ویندوز این ارجاعات را که در Imports Table قرار دارد چک کرده و IAT را با آدرس توابع API آدرس دهی میکند.

Packerها اصلاعات Import Table برنامه پک شده را از بین میبرند و هنگامی که برنامه اجرا میشود بارکننده ویندوز آدرسهای غلط را در IAT قرار میدهد. نتیجه این میشود که آدرس دهی درست IAT برعهده packer قرار میگیرد.

اگر شما یک برنامه pack شده را که از ویژگی شماره ۴ استفاده کرده باشد را در disassembler باز کنید IAT آن مانند زیر خواهد بود:

```
  XXXXXXXXh: ZZZZZZZZh      ; ZZZZZZZZh is inside a
buffer dynamically
                            ; allocated by the packer, so
it will
                            ; not exist if you remove the
packer.
  ZZZZZZZZh: push ebp       ; (*)
             ror eax, 16h
             pushf
             popf
             mov ebp, esp   ; (*)
             call @@1
             db 68h
        @@1:
             add eax, 134h
             ....
             jmp ACTUAL_ENTRY_POINT_OF_API + k
```

(برنامه ای که با Asprotect پک شده است)

Packer دستورالعمل اول API را با کدهای اضافی درهم میکند (در مثال بالا کدهای اضافی با علامت * مشخص شده اند) البته گاهی این کدهای اضافی میتوانند حاوی کدهای anti-debug هم باشند و در آخر سر یک پرش (و یا چیزی شبیه به پرش مانند ارجاع) به نقطه شروع اصلی(EOP) تابع API بعدی خواهد داشت (api + k) که k اینجا منظور شماره آدرس اصلی میباشد که در میان کدهای اضافی پنهان شده است.

بوسیله این ترفند یافتن توابع API که در آدرس ZZZZZZZZ قرار دارند غیرممکن میشود.

(imports table ساختار ساده ای ندارد و بحث در مورد جزئیات آن خارج از حوزه این کتاب میباشد)

برخی از برنامه هایی که به اصطلاح به آنها "Import Rebuilder" میگویند قادرند برخی از کدهای اضافی و الگوریتمهای جاسازی آنها را شناسایی کرده و توابع اصلی API را شناسایی کرده و آدرسهای واقعی آنها را برای ما پیدا کنند. اما معمولاً اینگونه برنامه ها عملکرد بسیار محدودی دارند و همیشه نمیتوانند پا به پای آخرین ویرایش packerها حرکت کنند.

حال بیایید ببینیم چه اتفاقی برسر OEP می افتد.
همنطور که قبلاً اشاره کردم هنگامی که یک packer در زمان با موفقیت کار خود را انجام داد (فایل مورد نظر راحالت کد درآورد و جدول IAT آنرا از نو ساخت) باید کنترل را به برنامه اصلی واگذار کند. این کار میتواند با ساخت یک thread(پردازش جدید) انجام شود.(البته راه حل ها دیگری نیز وجود دارد ولی برای اینکه یک کرکر کارکشته شوید همیشه بدترین حالت را در نظر بگیرید.)و بعد از واگذاری کنترل به برنامه اصلی packer به OEP پرش میکند.

Stolen Bytes(بایتها دزدیده شده)

راه دیگر اینکه بتوان برنامه را از دسترس کرکر نگهداشت استفاده از ترفند stolen bytes میباشد.
در این روش packer یکسری از توابع API از پیش تعریف شده را از برنامه حذف میکند.(یکی از رایج ترین توابع ، تابع GetModuleHandleA میباشد) . مانند زیر:

- Calls `GetModuleHandleA` and stores the return.
- Looks inside the protected app for patterns like:

```
call XXXXXXXXh
     ; call to kernel32.GetModuleHandleA
mov [handle], eax
```

Packer ارجاع مورد نظر را برداشته و دستور:

Mov [handle],eax

را با دستوری مشابه دستور زیر جایگزین میکند:

Mov [handle],hardcode_value

که در اینجا مقدار hardcoded_value بوسیله تابع GetModuleHandleA که قبلاً توسط pakcer برداشته شده است ، برگشت داده میشود.
اگر کرکر متوجه این موضوع نشود برنامه در هنگام کرک شدن بر روی هیچکدام از سیستم عاملهای موجود اجرا نخواهد شد.
این بایتهایی که توسط packer از برنامه برداشته میشود به اصطلاح stolen byte نامیده میشوند.

درک مشکلات stolen byte

همانگونه در مطالب بالا گفته شده packer اولین دستورالعمل k را که مربوط به یک تابع API میباشد را بعد از ادغام با کدهای اضافی به سمت K+1مین دستور حرکت میکند.
دومین K همانند زیر میشود:

```
; not changed entry point of
; kernel32.ExitProcess inside
; KERNEL32.DLL.

77E55CB5 kernel32.ExitProcess    push ebp
```

```
77E55CB6                            mov ebp,esp
77E55CB8                            push -1

; example of instructions you could
; find at the buffer:
          xchg eax, esp
          sub eax, 4
          jmp @@1
          db 68h
      @@1:xchg eax, esp
          mov dword ptr [esp], ebp
          push esp
          pop ebp
          push (77E55CB8+RANDOM_VALUE)
          sub dword ptr [esp], RANDOM_VALUE
          ret
```

هر دو k مانند همدیگر با کدهای اضافی ادغام میشوند و پیدا کردن توابع API میان این همه کد اضافی کار آسان و راحتی نیست و بعد از مقوله بهم ریختگی کدهای API بازیابی buffer میباشد و از آن سخت تر ساخت مجدد imports table.
حال بیاید فرض کنیم شما یک نقطه توقف (Break Point) برروی تابع kernel32.ExitProcess گذاشته اید که خیلی راحت بوسیله packer دور زده میشود. البته شما میتوانید هرکجا که خواستید Bp بگذارید اما اینکه پارامترهای توابع API از کجا به آنها ارجاع داده میشود کار بسیار مشکل و طاقت فرسایی میباشد.
امکان گذاشتن Bp برروی آدرسهای درست توابع API یک مسئله بسیار مهمی است، در واقع کنترل کردن توابع API بصورت درست و کارآمد میتواند کنترل برنامه را از packer خارج کردن و آن را در دستان شما قرار دهد.
معمولاً برنامه های تجاری متوسط (منظور از نظر کاربرد و قیمت آن میباشد) packerها را ترجیح میدهند.شناسایی و کارکرد درونی packerها یکی از دغدغه های ذهنی هر کرکر میباشد.

Asprotect چگونه کار میکند:

نرم افزار asprotect از سازنده aspack که در دو فصل قبل با نحوه آنپک آن آشنا شدید میباشد)www.aspack.com(
این packer فقط یک نرم افزار صد کرک تجاری صرف محسوب نمیشود بلکه تکنولوژی بکار رفته در این نرم افزار یک انقلاب در این صنعت به شمار میرود. در واقع این محصول یک packer آماده بوده که کسانی که نمیخواهند وقت خود را در مورد برنامه نویسی anti-crack صرف کنند مفید است.

سازنده asprotect آقای Alexey Solodovnikov درسهای بسیار زیادی را از نرم افزار قبلی خود aspack برای ساخت این برنامه یادگرفته است.به جرات میتوان گفت تمامی روشهای کرک کردن توسط این packer شناسایی شده و توسط این نرم افزار پیاده میشود ، تنها ضعف این packer استفاده نکردن از روشهای anti debug- میباشد.

در مقایسه با FLEXlm این برنامه بسیار زیبا و خوش ساخت نوشته شده است و از الگوریتم بسیار قوی بابت فشرده کردن فایلها استفاده میکند (الگوریتم مربوط به Aspack) و حجم برنامه را در حد چشمگیری کاهش داده و فقط ۶۰ کیلوبایت به header فایل اضافه میکند.
تا زمانی که برنامه از حالت فشرده خارج نشود ، برنامه اصلی قابل دسترسی نیست. روالی که برای از حالت فشرده سازی درآوردن استفاده میشود فایل را در مورد تغییرات احتمالی چک میکند.

هنگامی که برنامه در حافظه قرار میگیرد یک روال برای چک کردن حافظه وجود دارد که جلوی تغییراتی که در حافظه اعمال میشود را بگیرد(منظور تغییراتی که در دامنه حافظه برنامه مورد نظر است و نه کل سیستم)
Asprotect از decompress کردن برنامه جلوگیری میکند ، این عمل باعث میشود عملیات dumping بوسیله نرم افزارهایی مانند ProcDump سخت تر و یا غیرممکن شودو هنگامی که قسمت Import از حافظه Dump شود فایل اجرایی (PE) درست نخواهد شد!
اما هنوز راههایی برای Decompress برنامه هایی که با asprotect محافظت میشوند وجود دارد اما نه فقط با یک پرش. ولی با این حال حتی بعد از dumping و decompress فایلهای asprotect این فایلها کامل نبوده و بعضی از کرکرها ترجیح میدهند خود یک بارکننده برای این نوع packer بنویسند.

مانند FLEXlm ،Asprotect از اجرای توابعیی که در زمان ثبت نشدن برنامه نباید اجرا شوند جلوگیری میکند برای اینکار asprotect از روش بسیار جالبی استفاده میکند.
برای مثال:
اگر برنامه نویسی بخواهد قسمت پیش نمایش ، برنامه خود را فقط در حالت ثبت شده برای کاربر نشان دهد فقط کافیست کدهای مربوط به اینکار را در جاهای مناسب قرار دهد و بعد از اینکه کاربر برنامه را خرید و آنرا ثبت کرد این تابع بوسیله یک registration key که ثابت میباشد بکار می افتد.(decode کردن اینگونه روالها بدون registration key غیرممکن است).
Asprotect از تکنیکهای بسیاری پیچیده ای برای جلوگیری از روشهای پیدا کردن رمز مانند Brute Force استفاده میکند.

سه روش برای محافظت از برنامه توسط asprotect وجود دارد:

- استفاده از متد کلاسیک فشرده سازی میباشد. این راه بدلیل آسانی بازیافت آن توسط دیباگر توصیه نمیشود.
- دومین روش که یکی از روشهای جذاب میباشد ولی بهترین روش نیست. بوسیله این روش فقط یک لفافه برروی نرم افزار شما قرار خواهد گرفت (البته با احتساب ویژگی روش اول) و فقط یک registration key ثابت در اختیار شما میگذارد و تمامی برنامه اصلی بوسیله همان کلید محافظت میشود. در واقع بوسیله این روش برنامه نویس خود مبادرت به ایجاد یک روش دستی ضد کرک میکند و از asprotect فقط برای محافظت هرچه بیشتر کد ضد کرکی که خود نوشته است استفاده میکند.امنیت این روش بیشتر بستگی به تکنیکها و ترفندهایی بستگی دارد که برنامه نویس در کد ضد کرک خود بکار برده.
- آخرین روش و بهترین روش.در این روش هیچگونه روال محافظتی به برنامه شما اضافه نمیشود(البته در صورت تمایل شما هنوز میتوانید روالهای ضد کرک خود را پیاده سازی کنید).
هنگامی که شما از این روش استفاده میکنید در قسمت مربوط به registration درasprotect شما یک register key ثابت انتخاب میکنید سپس asprotect یک ثابت پایه برای برنامه شما میسازد که مینای ساخت دیگر registration keyهای نقاط مختلف برنامه شما میشود.حتی شما میتوانید کلیدهای مورد نظر خود را برای هر کاربر ایجاد کرده و آنها را مدیریت کنید و حتی شما میتوانید جایی که کلیدها باید در رجیستری ویندوز ذخیره شوند را مشخص کنید.

قدم بعدی چک کردن شماره سریالهایی میباشد که شما با asprotect برای کاربرانتان ساخته اید.برای مثال اگر برنامه شما از قابلیت اتصال به اینترنت و چک کردن نسخه های جدید برنامه را پشتیبانی میکند.و اگر کاربری شماره سریال خود را به شخص دیگری هم داده باشد شما به راحتی میتوانید هر دو شخص را از طریق اینترنت شناسایی کرده در asprotect مشخص کنید که در نسخه های بعدی شماره سریال برای شخص خاطی ساخته نشود.

در کد نمونه زیر یک مثال از نحوه پیاده سازی asprotect در Visual C++ آمده است:
البته نمونه کدهای Delphi,VB و دیگر زبانهای برنامه نویسی را میتوانید از سایت aspack.com دریافت نمایید.

```
include <windows.h>
#include "include\asprotect.h"
char *message;
void RegisterAction ()
{
   REG_CRYPT_BEGIN
   message = "Registered version !";
   REG_CRYPT_END
}
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE
hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
   message = "Unregistered version !";
   RegisterAction();
   MessageBox (0,message,"",0);
   return 0;
}
```

همانطور که در بالا مشاهده میکنید شما باید کد REG_CRYPT_BEGIN را در اول و کد REG_CRYPT_END را در آخر روالی که میخواهید بصورت ثبت شده توسط کاربر استفاده شود قرار دهید.

و یا به معنای دیگر در اول روال ، کد:

```
0EBh, 04h, 0EBh, 05h, 89h, 89h, 0E9h, 0, 0, 0, 0
```

و در آخر روال ، کد :

```
0EBh, 04h, 0EBh, 05h, 99h, 99h
```

را باید قرار دهید.

بوسیله این اطلاعات asprotect روال مورد نظر شما را encode میکند.

و تنها کاری که شما باید انجام بدهید صدا کردن روال مربوطه میباشد بقیه کارها(منظور چک کردن اعتبار کاربر و ثبت شده بودن برنامه)بر عهده asprotect میباشد.

شما میتوانید اسم کاربر را بوسیله تابع apiGetRegInfo() از سیستم بگیرید. امکان دارد که شما بخواهید registration keyهای بسیار زیادی را در آن واحد تولید کنید.asprotect اینکار را بوسیله فایل کتابخانه ای keygen.dll انجام میدهد.شما میتوانید کلیدهای بسیاری را بوسیله دو تابع GenerateKeyAsRegFile() و GenerateKeyAsString()بسازید. البته تابع دوم اشاره گر کلید را که در حافظه قرار دارد به شما میدهد.

همچنین asprotect به شما امکان تعداد دفعات باز شدن برنامه را میدهد چه بصورت تاریخ و چه بصورت تعداد اجرای برنامه.

در آخر باید بگویم با اینکه سازنده asprotect ادعا کرده است packer ضدنفوذی ساخته است ولی اگر شما یک شماره سریال معتبر داشته باشید به راحتی میتوانید برنامه را dump کرده و جدول IAT آنرا ترمیم کرده و برنامه را کرک کنید.

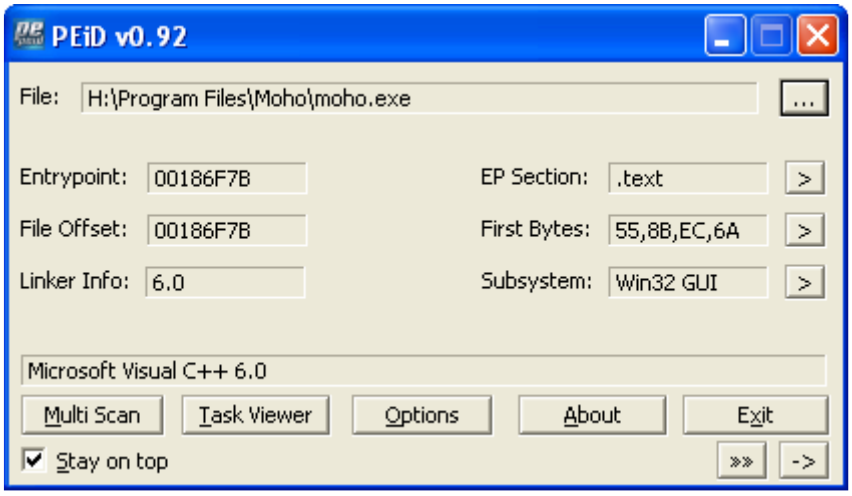
پایان

آموزش کرکینگ با ollydbg قسمت هفتم
تاریخ : اردیبهشت ۱۳۸۵ 2006 May
نوع حمله Memory Breakpoints
نرم افزار هدف : Lost Marble's Moho v5.1
نرم افزارهای مورد استفاده : ollydbg 1.10,PEiD 0.93
سطح : متوسط

مقدمه:

در این فصل قصد دارم یکی دیگر از تکنینکهای مهندسی معکوس را به شما نشان دهم.
تکنیکی به نام Memory Breakpoints ، بوسیله این تکنینک دیگر نیازی نیست برای یافتن شماره سریال برنامه در تمامی برنامه از Bp های متعددی استفاده کنید با کمی دقت و فکر میتوانید به راحتی برنامه را طوری تغییر دهید که همیشه ثبت شده به نظر برسد.
البته احتمالاً از فصول قبل فهمیده اید که تمامی روشها کلی نبوده و نمیتوان آنرا برای تمامی برنامه ها تجویز کرد و باید شرایط را سنجیده و راه مناسب انتخاب کرد.

خب اول نرم افزار رو نصب میکنیم.
بعد از نصب همیشه اولین کاری که باید بکنیم این هست که اول بفهمیم برنامه رو با چی نوشتن و با چی پک شده برای اینکار میتوانید از نرم افزارهایی مانند PEid و یا RDG Packer Detector استفاده کنید من اینجا از PEid استفاده کردم.
برنامه PEid رو باز کرده و فایل UDRulerp.exeرو بندازید توش و یا از دکمه ...(open) برای تستِ فایل استفاده کنید



شکل ۱

برنامه از هیچگونه packer و یا protector استفاده نکرده است.

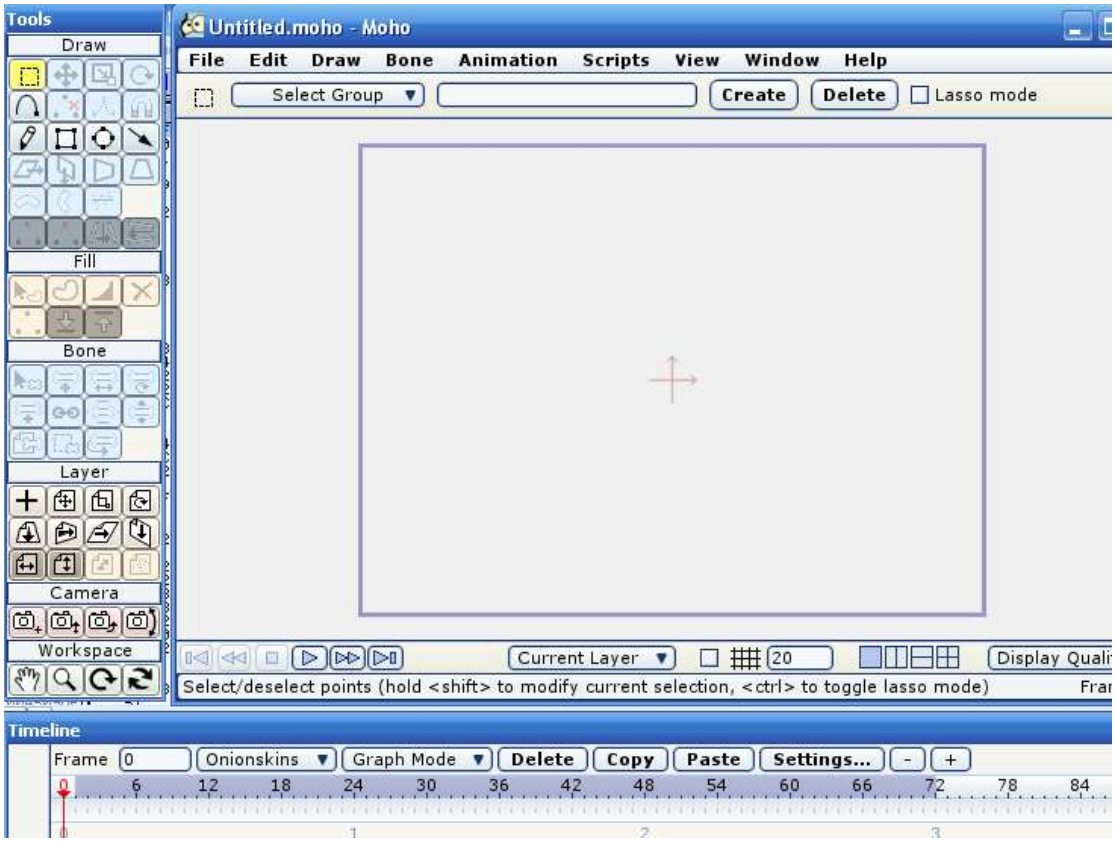
برنامه را با ollydbg باز میکنیم

```
00586F7B  $  55           PUSH EBP
00586F7C  .  8BEC         MOV EBP,ESP
00586F7E  .  6A FF        PUSH -1
00586F80  .  68 40655B00  PUSH moho.005B6540
00586F85  .  68 98AB5800  PUSH moho.0058AB98                    SE handler installation
00586F8A  .  64:A1 000000 MOV EAX,DWORD PTR FS:[0]
00586F90  .  50           PUSH EAX
00586F91  .  64:8925 0000 MOV DWORD PTR FS:[0],ESP
00586F98  .  83EC 58      SUB ESP,58
00586F9B  .  53           PUSH EBX
00586F9C  .  56           PUSH ESI
00586F9D  .  57           PUSH EDI
00586F9E  .  8965 E8      MOV DWORD PTR SS:[EBP-18],ESP
00586FA1  .  FF15 9C305A0 CALL DWORD PTR DS:[<&KERNEL32.GetVersio kernel32.GetVersion
00586FA7  .  33D2         XOR EDX,EDX
00586FA9  .  8AD4         MOV DL,AH
00586FAB  .  8915 9809780 MOV DWORD PTR DS:[780998],EDX
00586FB1  .  8BC8         MOV ECX,EAX
00586FB3  .  81E1 FF00000 AND ECX,0FF
00586FB9  .  890D 9409780 MOV DWORD PTR DS:[780994],ECX
```

برنامه را با زدن کلید F9 اجرا میکنیم.

بعد از چند لحظه پنجره ای برای گرفتن شماره سریال نمایان میشود.

کلید Demo را بزنید تا برنامه بطور کامل اجرا شود.

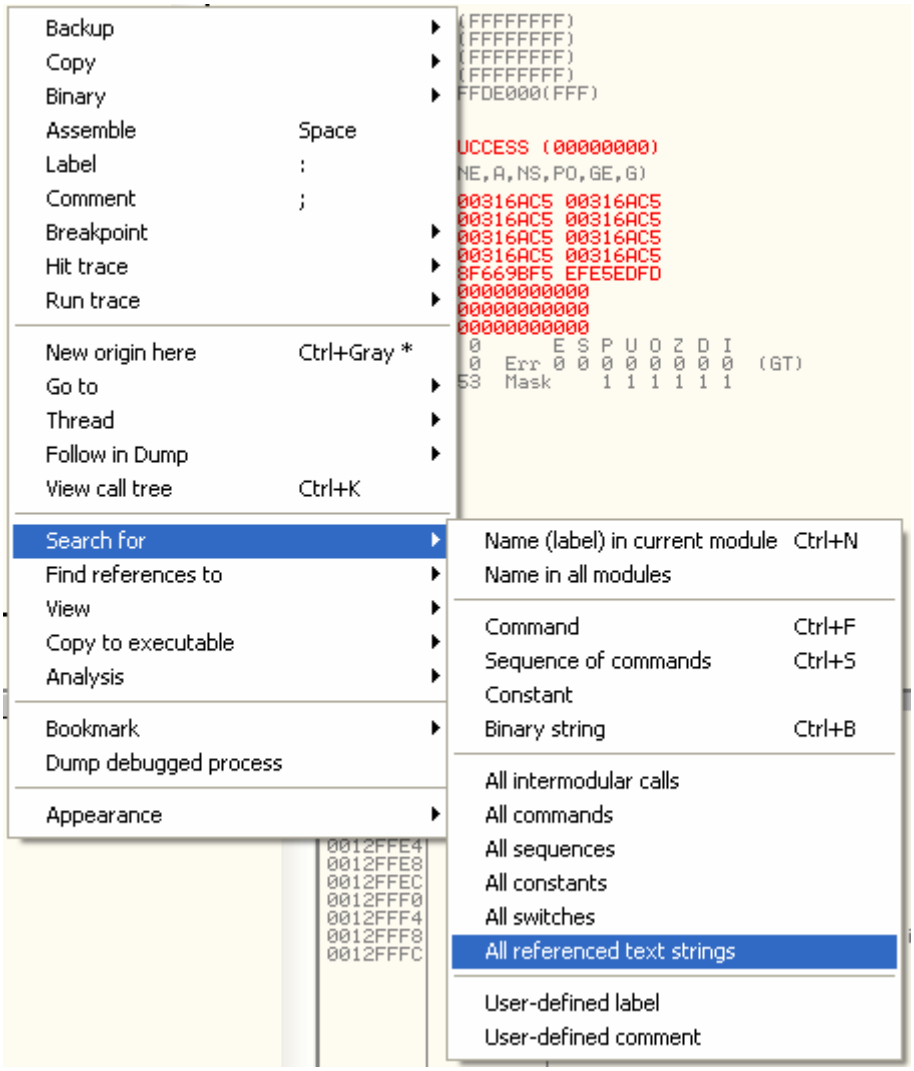


حال بیاید ببینیم برنامه در حالت Demo (ثبت نشده) چه محدودیتهایی را دارا میباشد.
تمام گزینه ها فعال است و کار میکند ولی هنگام خروجی گرفتن از برنامه به دلیل اینکه برنامه خریداری نشده است در تمامی خروجیها بصورت background اسم lost marble moho نوشته میشود.

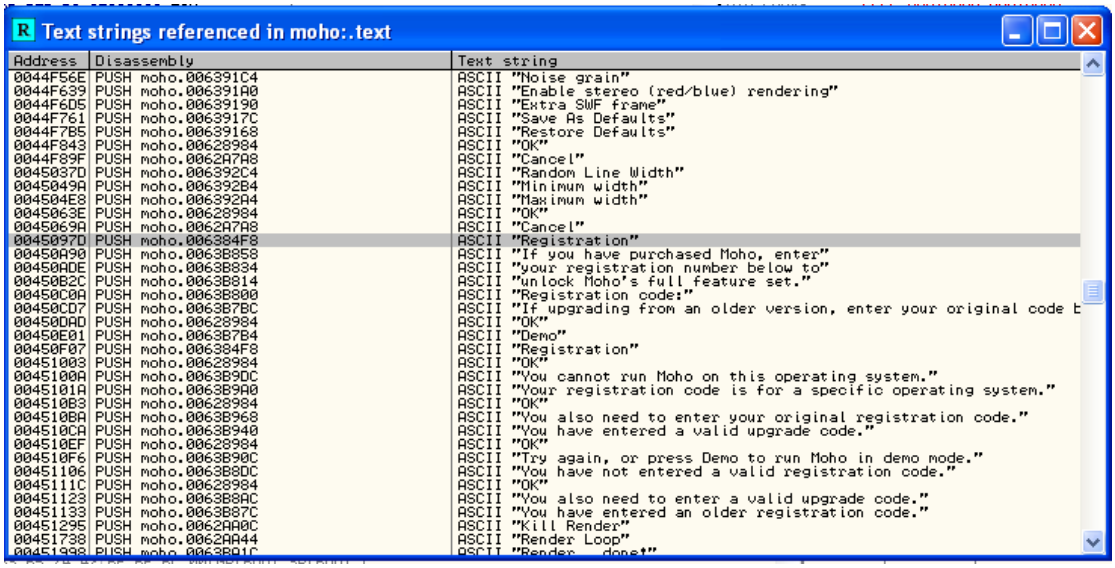
حال دوباره برنامه با ctrl+F2 ریستارت کنید و دوباره برنامه را با F9 باز کنید اینبار در پنجره ای که از کاربر شماره سریال مخواهد عددی را وارد نمایید من عدد ۹۸۸۹۹ را زدم حال دکمه ok را فشار دهید.

با یک پیغام مواجه میشوید که میگوید شماره سریالی که شما وارد کرده اید نامعتبر میباشد.این پیغام را یادداشت کرده و به ollydbg بازگردید.

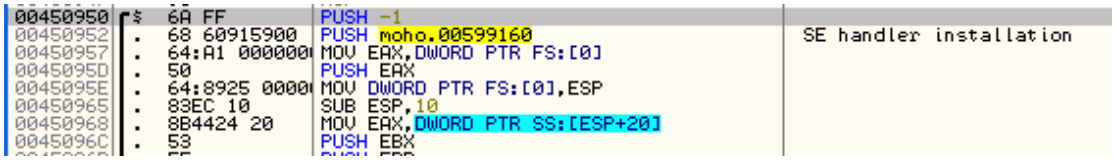
در ollydbg در پنجره code دکمه سمت راست موس را زده و گزینه search for و بعد گزینه س all referenced text string را انتخاب کنید.



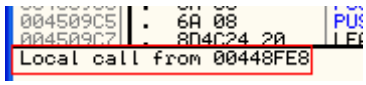
در پنجره باز شده دکمه سمت راست موس را زده و کلمه registration را وارد نمایید دقت کنید که گزینه case sensitive فعال نباشد، برروی دکمه ok کلیک کنید تا ollydbg بتواند رشته مورد نظر را بیابد.احتمالاً مجبور خواهید شد عملیات جستجو را دوباره تکرار کنید برای اینکار میتواند ctrl+L را فشار دهید تا کلمه مورد نظر شما در باقیمانده رشته ها جستجو شود.



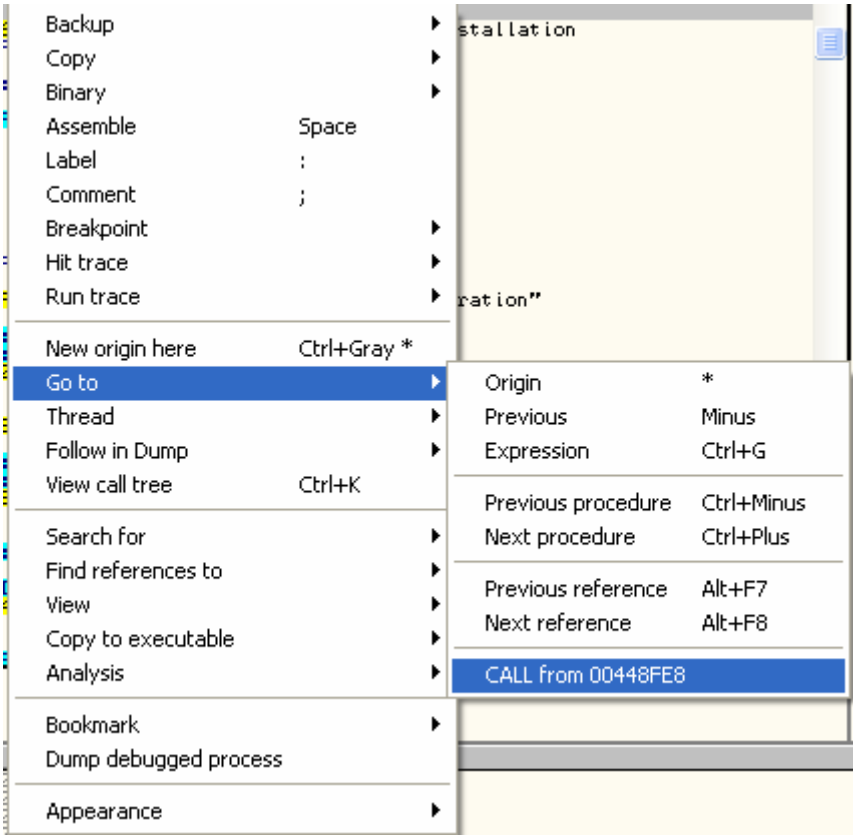
شما به آدرس 45097D خواهید رسید بر روی آن دابل کلیک کنید
کمی در پنجره کد نوار پیمایش را بالاتر برده تا به اول روال چک کردن شماره سریال برسید.



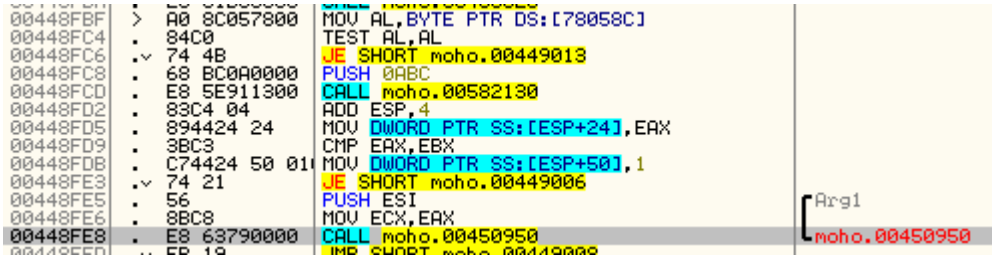
به پنجره info که زیر پنجره code هست دقت کنید.



این روال از آدرسی دیگر صدا زده شده است در واقع از آدرس 448FE8 . حال بر روی اولین دستور دکمه سمت راست موس را زده و گزینه goto و بعد آخرین گزینه یعنی CALL from 448FE8 را انتخاب کنید.



هنگامی که به آدرس مورد نظر رفتیم در واقع جایی که شماره سریال ما توسط برنامه چک میشود با کمی دقت خواهید دید که دو پرش شرطی وجود دارد.

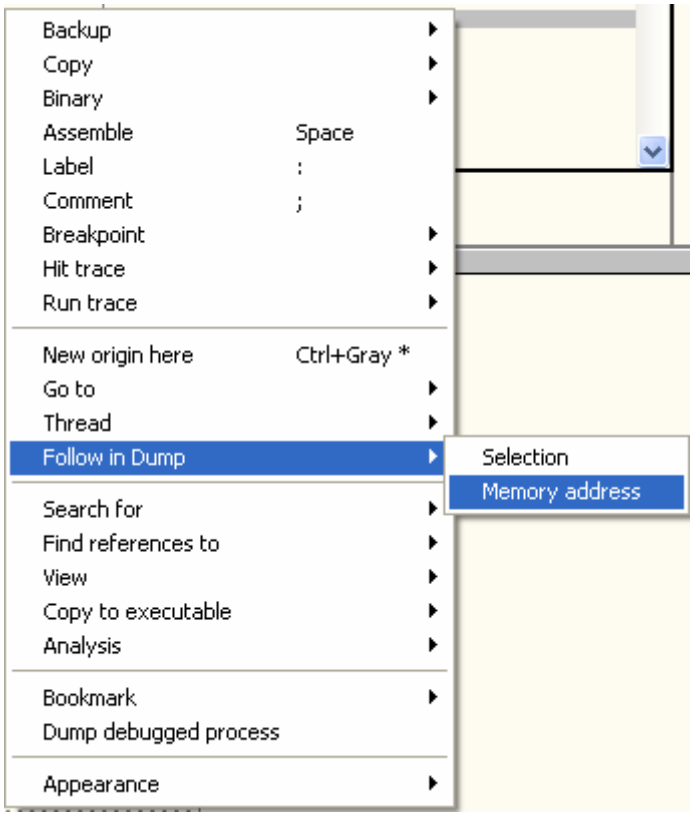


در آدرس 448FE3 پرش شرطی JE SHORT moho.449006

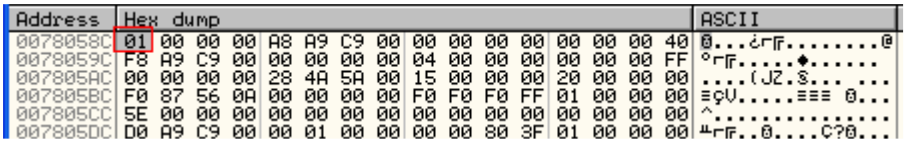
و در آدرس 448FC6 پرش شرطی JE SHORT moho.449013 که در بالای آن پرش ، شرط پرش جک میشود یعنی TEST AL,AL . یکی از رایجترین شرط پرشهایی که باید به آن دقت کنیم و در این مثال شرط کلیدی میباشد.

اگر به آدرس 448FBF دقت کنید میبینید که AL در این آدرس مقدار دهی میشود در واقع مقدار AL از آدرس 78058C که در حافظه قرار دارد در درون AL قرار میگیرد.

برای اینکه متوجه شویم الان AL دارای چه مقداری هست بر روی کد آدرس 448FBF دکمه سمت راست موس را زده و گزینه follow in dump و بعد گزینه memory address را انتخاب کنید



حال به پنجره dump دقت کنید:



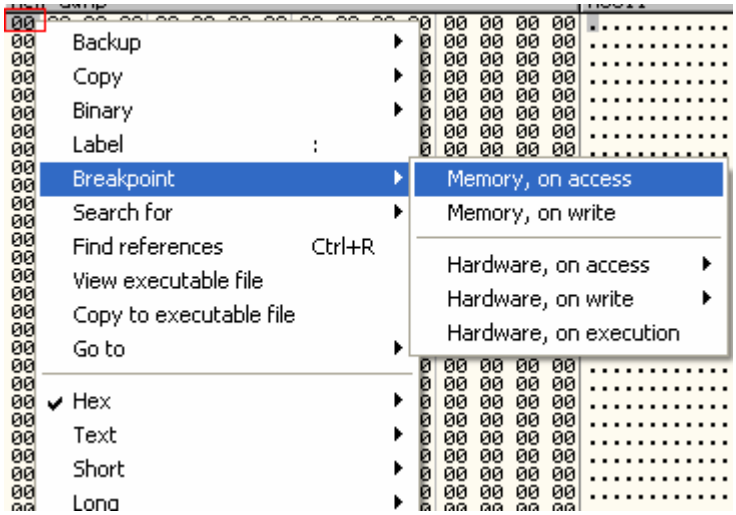
همانطور که مشاهده میکنید مقدار AL برابر 01 میباشد.

تنها کاری که ما باید انجام دهیم اینست که همیشه مقدار AL را در این مرحله برابر 00 نگه داریم.

برای اینکه بفهمیم کی و کجا مقدار AL برابر 01 میشود و از آن عمل جلوگیری نماییم باید از Memory Breakpoint استفاده کنیم.

حال مراحل زیر را انجام دهید:

- در پنجره dump کلید ctrl+g را فشار داده و آدرس 78058C را در درون پنجره باز شده وارد کنید.
- برنامه را با کلید ctrl+F2 ریستارت کنید.
- دوباره در پنجره dump کلید ctrl+g را وارد نمایید و اینبار فقط کلید enter را فشار دهید(به این دلیل که آدرسی که قبل در این پنجره وارد کرده اید در حافظه موقت ollydbg باقی می ماند).
- مانند شکل زیر مقدار آدرس 78058C را انتخاب کرده و دکمه سمت راست موس را برروی آن زده و گزینه Breakpoint و بعد گزینه Memory on access را انتخاب کنید.

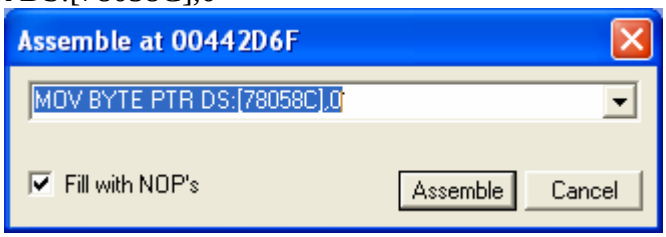


- حال برنامه را با زدن دکمه F9 اجرا کنید

بعد از چند لحظه خواهید دید که ollydbg در آدرس442D6F متوقف میشود.

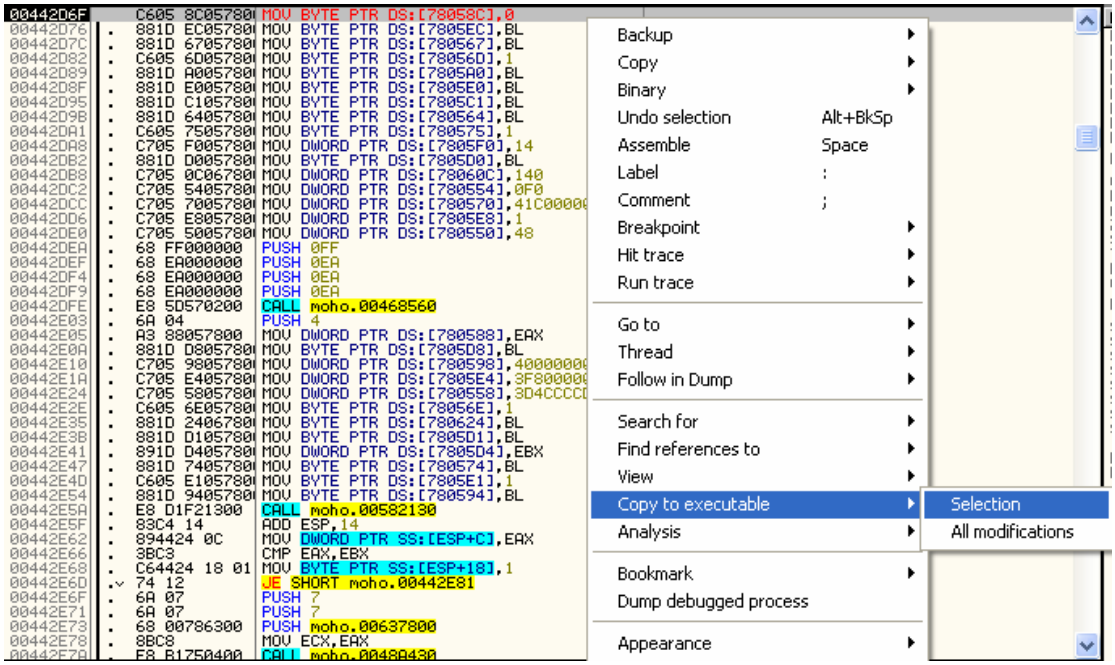
این آدرس همان آدرسی هست که مقدار AL را برار 01 میکند و به این معنا که برنامه خریداری نشده است بر روی این آدرس دکمه space را فشار دهید کد مورد نظر را مانند زیر تغییر دهید:

MOV BYTE PTR DS:[78058C],0



حال دکمه assemble را فشار دهید.

برای ذخیره کردن برنامه برروی همان آدرسی که آنرا تغییر داده اید ایستاده و مانند زیر عمل کنید.



و در پنجره بعدی دکمه سمت راست را زده و گزینه save file را انتخاب کنید و برنامه را ذخیره کنید.

همانطور که مشاهده کردید ، یکی از روشهای موثر و کارا برای کرک کردن برنامه ها استفاده درست و بجا از Memory Breakpointها میباشد.البته این کار نیاز به تجربه کافی دارد و در هر برنامه ای که میخواهید این عملیات را انجام دهید باید تمامی شرایط را سنجیده و دقیقاً چک کنید و بعد عملیات را انجام دهید گاهی ممکن است برنامه آنطور که بنظر میرسد عمل نکند و شما نتوانید از این حالت استفاده کنید.

پایان

آموزش کرکینگ با ollydbg قسمت هشتم
تاریخ : اردیبهشت ۱۳۸۵ 2006 May
تئوری breakpointها
نرم افزار هدف : ندارد
نرم افزارهای مورد استفاده : ندارد
سطح : متوسط

مقدمه:

اگر از اول کتاب تا کانون قدم به قدم کتاب را خوانده باشید حتماً دیده اید که ما از نقاط انفصال ها(Breakpoint) استفاده زیادی کرده ایم ولی تاکنون بصورت صریح تعریفی از چگونگی کارکرد آنها و نوع آنها نداشته اید در این فصل در مورد breakpointها بحث خواهم کرد.
در کل دو نوع breakpoint وجود دارد:

- نقاط انفصال سخت افزاری (Hardware Breakpoint)
- نقاط انفصال نرم افزاری (Software Breakpoint)

نقاط انفصال سخت افزاری:

اخیراً پرادزنده ها خود نقاط انفصال سخت افزاری را بوسیله ثباتهای خاص که به آنها ثباتهای debug میگویند پشتیبانی میشود.

در کل چهار ثبات debug وجود دارد:
DR0,DR1,DR2,DR3
این ثباتها آدرس خطی چهار breakpoint را در خود ذخیره میکنند.
شرط انفصال که باعث متوقف شدن برنامه در آن نقطه میشود در یک ثبات مخصوص دیگر بنام DR7 ذخیره میشود.
هنگامی که شرط برقرار شود پردازنده یک استثناء وقفه INT 1 را صادر میکند و بعد از آن کنترل به دست debugger مربوطه می افتد.
چهار شرط ممکن که برای HW BP پیش بینی شده است:

- هنگامی که یک دستورالعمل اجرا شود.
- محتویات مکانی از حافظه که مشخص شده است تغییر کند.
- محتویات مکانی از حافظه که مشخص شده است خوانده شود و یا بروز شود.
- یکی از ورودی و یا خروجی درگاهها را مشخص کرده باشیم.

هنگامی که شما یک Hardware Breakpoint را مشخص میکنید debugger بیت trap را که مربوط به flag(نشانه) ثباتها میباشد را چک میکند ، اگر شرط برقرار باشد استثناء INT 1 بوسیله پردازنده صادر میشود و کنترل بدست debugger می افتد.

ملاحضات Hardware Breakpoint:

همانطور که تاکنون متوجه شده اید نقاط انفصال سخت افزاری محدود به چهار عدد میشوند (شاید این یک ضعف بنظر برسد) اما فایده نقاط انفصال سخت افزاری در این است که تقریباً غیر قابل تشخیص بوسیله نرم افزارهای هدف میباشند.
تنهای راهی که میتوان یک Hardware Breakpoint را شناسایی کرد اینست که بوسیله یک نرم افـزار، مقـدار ثباتهـای DR0 تا DR7 را خوانده و flag های ثبات DR7 را تجزیه و تحلیل کرد.البتـه ایـن راه فقـط در RING 0 امکـان پـذیر است.

برای اطلاع بیشتر از falgهای ثبات DR7 میتوانید به مستندات شرکت اینتل مراجعه کنید در آدرس:
http://developer.intel.com/design/pentium4/manuals/253668.htm

کد زیر یک نمونه از سوئیچ کردن به RING 0 میباشد که برای پیدا کردن مقادیر flag های DR7 به کار میرود.

```
.386
.MODEL FLAT,STDCALL locals
jumps
UNICODE=0
include w32.inc

Extrn SetUnhandledExceptionFilter : PROC
Interrupt equ 5                                 ;the interrupt numbers 1 or 3
will make
                                                ;debugging more difficult
.DATA

message1 db "Debug breakpoint detection",0
message2 db "Debug breakpoint not found",0
message3 db "Debug breakpoint found",0
delayESP dd 0                                   ;the ESP register saves here
previous dd 0                                   ;the ESP register will save the
address
                                                ;of the previous SEH service
here

.CODE
Start:
;-------------------------------------------------------------------------------
-----------------------
;Sets SEH in case of an error
;-------------------------------------------------------------------------------
-----------------------
     mov [delayESP], esp
     push offset error
     call SetUnhandledExceptionFilter
     mov [previous], eax
;-------------------------------------------------------------------------------
-----------------------

     push edx
     sidt [delayesp-2]                          ;reads IDT into the stack
     pop edx
     add edx, (Interrupt*8)+4                   ;reads the vector of the
required interrupt
     mov ebx,[edx]
     mov bx,word ptr [edx-4]                    ;reads the address of the old
service of the
                                                ;required interrupt
     lea edi,InterruptHandler
     mov [edx-4],di
     ror edi,16                                 ;sets the new interrupt service
```

```
     mov [edx+2],di
     push ds                                ;saves registers for security
     push es
     int Interrupt                          ;jumps into Ring0 (a newly
defined INT 5h service)
     pop es                                 ;restores the registers
     pop ds
     mov [edx−4],bx                         ;sets the original INT 5h
interrupt service
     ror ebx,16
     mov [edx+2],bx
     push eax                               ;saves the return value

;−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
−−−−−−−−−−−−−−−−−−−−−−
;Sets the previous SEH service
;−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
−−−−−−−−−−−−−−−−−−−−−−
     push dword ptr [previous]
     call SetUnhandledExceptionFilter
;−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
−−−−−−−−−−−−−−−−−−−−−−

     pop eax                                ;restores the return value
     test eax,eax                           ;tests to see if eax=0
     jnz jump                               ;if not, the program has found a
debug
                                            ;breakpoint and it ends

continue:
     call MessageBoxA,0, offset message2,\
     offset message1,0
     call ExitProcess, −1

jump:
     call MessageBoxA,0, offset message3,\
     offset message1,0
     call ExitProcess, −1

error:                                      ;sets a new SEH service if there
is an error
     mov esp, [delayESP]
     push offset continue
     ret

;−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
−−−−−−−−−−−−−−−−−−−−−−
;Your new service INT 5h (runs in Ring0)
;−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
−−−−−−−−−−−−−−−−−−−−−−
InterruptHandler:
     mov eax, dr0                          ;reads a value from the DR0 debug
register
     test ax,ax                            ;tests to see if a breakpoint was
set
     jnz Debug_Breakpoint                  ;if so, the program jumps
     mov eax,dr1                           ;reads a value from the DR1 debug
register
     test ax,ax                            ;tests to see if a breakpoint was
set
     jnz Debug_Breakpoint                  ;if so, the program jumps
     mov eax,dr2                           ;reads a value from the DR2 debug
register
     test ax,ax                            ;tests to see if a breakpoint was
set
     jnz Debug_Breakpoint                  ;if so, the program jumps
     mov eax,dr3                           ;reads a value from the DR3 debug
register
```

```
     test ax,ax                              ;tests to see if a breakpoint was
set
     jnz Debug_Breakpoint                    ;if so, the program jumps
     iretd                                   ;if a breakpoint was not set the
program will
                                             ;return 0 into eax

Debug_Breakpoint:
     mov eax,1                               ;sets the value 1 into eax to
show that
                                             ;breakpoints are active
     iretd                                   ;jump back into Ring3

ends
end Start
```

این تکنیک یکی از معدود تکنیکهای شناسایی Hardware Breakpoint میباشد.این تکنیک میتواند نقاط انفصال سخت افزاری را پاک کرده بدون اینکه کنترل برنامه بدست debugger بیفتد و یا حتی هنگامی که یک نقطه انفصال سخت افزاری را مشاهده کرد به نقطه دیگری در برنامه رفته و باعث سردرگمی کرکر شود.
ولی متاسفانه این تکنیک فقط در ویندوز 9x جوابگو میباشد چراکه در ویندوزهای NT based(NT 4.0,2000,XP,2003) شما نمیتوانید به RING 0 سوئیچ کنید به این دلیل که بسیاری از ویروسهای مخربی که برای سیستم عامل ویندوز وجود دارند میتوانند به این لایه حرکت کرده و باعث خرابیهای جبران ناپذیری بشوند و به همین دلیل شرکت مایکروسافت تصمیم گرفت که حق دسترسی به RING 0 را کاملاً از برنامه های سطح بالا بگیرد.

عوض کردن ثباتهای دیباگ در RING 3:

با توجه به تجربیاتی که من در این زمینه کسب کرده ام این منطقی نیست که شما بطور کامل ودستی به RING 0 سوئیچ کرده و مقادیر ثبات DR7 را تغییر دهید.
برنامه هایی که بعنوان debugger کار میکنند میتوانند به توابع API مانند GetThreadContext() و SetThreadContext() دسترسی داشته باشند.این توابع بوسیله فایل NTDLL.DLL اجرا میشوند و بعنوان توابع سیستمی شناخته میشوند(وقفه 2E) که باعث میشود که پردازنده شما را به RING 0 سوئیچ دهد.

شما خود میتوانید برنامه زیر را که بوسیله assembly نوشته شده است تست کنید. این برنامه از مکانیسم SEH برای پاک کردن مقادیر ثباتهای debug استفاده میکند و سپس به برنامه را به حالت عادی ادامه میدهد.
برای نمونه در debugger یک Hardware Breakpoint برروی یکی از دستورات NOP و یک Software Breakpoint بر روی اولین دستور SHE بگذارید و ببینید چه اتفاقی می افتد.

```
.686
.model flat, stdcall ;32 bit memory model
option casemap :none ;case sensitive
assume fs:nothing ;MASM feature (otherwise FS assumed to be ERROR)
include EraseDrx.Inc

.code

start:
     ; ### set the S.E.H ###
     push offset mySEH
     push dword ptr fs:[0]
     mov dword ptr fs:[0],esp
```

```
     ;*** now everything will be covered by our SEH ***
     ; raise an invalid opcode exception
     UD2

@@SafeOffset: ; this is where we can safely return from our SEH
     fnop

     ;try to hardware BP one of those NOP
     nop
     nop
     nop
     nop

     ;*** now this is this end of the SEH ***
     pop dword ptr fs:[0]
     add esp,4
     ret ;return to ExitThread

;-------------------------------------------------------------------------
----------------------
; OUR SEH handler, which erases the debug registers
;-------------------------------------------------------------------------
----------------------
mySEH proc C lpExcept:DWORD, lpFrame:DWORD, lpContext:DWORD,
lpDispatch:DWORD

     mov ecx,[lpContext]

     ; push all linear addresses of drx (from Dr0 to Dr3)
     ; you should see your hardware BP there (just to demonstrate where they
are)
     push [ecx][CONTEXT.iDr0]
     push [ecx][CONTEXT.iDr1]
     push [ecx][CONTEXT.iDr2]
     push [ecx][CONTEXT.iDr3]
     add esp,4*4 ; skip them

     ;erase DR0 to DR3
     push 0
     push 0
     push 0
     push 0

     pop [ecx][CONTEXT.iDr0]
     pop [ecx][CONTEXT.iDr1]
     pop [ecx][CONTEXT.iDr2]
     pop [ecx][CONTEXT.iDr3]

     ;erase also DR7
     push 0
     pop [ecx][CONTEXT.iDr7]

     ;now set EIP to our SafeOffset
     push offset @@SafeOffset
     pop [ecx][CONTEXT.regEip]

     mov eax,FALSE
     ret

mySEH endp

end start
```

در پایان، برای اینکه یک debugger بتواند خود را از اینگونه کدها پنهان کند باید تمام flagهای ثباتها را خوانده و روند اجرایی آنها را دنبال کند و همیشه هنگامی که قرار است مقداری از طرف این flagها به سمت برنامه ها تشخیص debugger فرستاده شود مقادیر را با 0 و یا مقدار False جایگزین کند که شناسایی نشود که البته اینکار بسیار مشکل میباشد.

استفاده از نقاط انفصال سخت افزار در OllyDBG:

حال این سوال پیش می آید که HW Breakpointها کجا به کار می آیند؟
شاید با خواندن بخش نقاط انفصال نرم افزاری بتوانید به این سوال جواب بدهید ولی درهر صورت نقاط انفصال سخت افزاری را میتوان در دو حالت کلی زیر بکار برد:

- برنامه ای که در حال دیباگ شدن است در مقابل تغییراتی که در درون آن انجام میشود حساس هست و اجازه اینکار را نمیدهد(برنامه هایی که خودشان ،خود را کنترل میکنند)
- برنامه ای که در حال دیباگ شدن میباشد روالی برای تشخیص دیباگر دارد و نقاط انفصال دیباگر را پیدا کرده و آنها را از بین میبرد(برنامه هایی که از ویژگیهای برنامه های خودکنترل بعلاوه ویژگیهای صدتحریف استفاده میکنند)

این ویژگیها در برنامه های فشرده سازی بسیار پیچیده و یا packerهایی مانند Armadilo,Asprotect,Execryptor و یا ActiveMark وجود دارد.

یکی دیگر از ویژگیهای نقاط انفصال سخت افزاری این هست که ثباتها هیچ ربطی به هیچگونه نرم افزار debugger ندارند و جزو ویژگیهای نرم افزاری اینگونه نرم افزارها به حساب نمی آیند.
برای نمونه یک برنامه ای که با asprotect پک شده است را بوسیله ollydbg باز کنید و بر روی یکی از دستورات آن یک HW Breakpoint بگذارید حال دوباره یک نسخه دیگر از ollydbg را باز کنید و همان برنامه را در درون آن باز کنید و آنرا اجرا کنید میبینید که نقطه انفصالی که بر روی اولین نمونه ollydbg گذاشته اید برروی دومین نمونه هم کار میکند(بدون اینکه شما HW Bp را در نسخه دوم مشخص کرده باشید).

نقاط انفصال نرم افزاری:

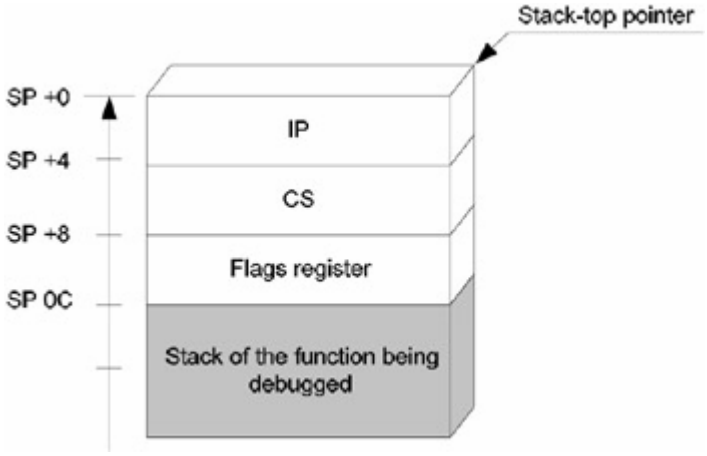
Software Breakpoint تنها مدل از نقاط انفصال میباشد که بدون نوشتن برنامه کامل شبیه ساز پردازنده قابلیت پنهان شدن از برنامه های تشخص دیباگر را ندارد.
برای تست این Bp میتوانید در درون دستورالعملهای یک برنامه کد 0xCC را جایگزین کنید و هنگامی که این خط اجرا شود استثنا INT 3 صادر خواهد شد و کنترل بدست debugger خواهد افتاد.

INT 3 کنترل را بدست میگیرید و شما هرکاری که بخواهید میتوانید با برنامه انجام دهید اما قبل از اینکه کنترل برنامه بوسیله INT 3 واگذار شود روال رسیدگی کننده به وقفه ها صدا زده میشود و مقادیر جاری تمامی flagهای ثباتها و اشاره گر ثبات CS و اشاره گر ثبات IP را در پشته قرار میدهد.

در واقع تمام وقفه ها غیر فعال است (IF flag مقداری ندارد) و مقدار flag مربوط به trap نیز پاک شده است.بنابراین بین وقفه دیباگ و بقیه وقفه تفاوتی وجود ندارد.

دیباگر برای اینکه بتواند در نقطه انفصالی که توسط کاربر مشخص شده است توقف نماید مقادیر تمامی ثباتها را در پشته ذخیره میکند و برای اینکه دیباگر بتواند در تمامی جاهای برنامه توقف کرده و کنترل آن را بدست بگیرد علاوه بر اینکه مقادیر را باید

در پشته ذخیره کند باید مقدار 0xCC را در جایی که کاربر مشخص کرده است جایگزین کند و آنرا اجرا کند و بعد از اتمام کار و هنگامی که کنترل میخواهد به برنامه هدف بازگردد دوباره تمامی مقادیر از پشته باید بازخوانی شود.



ملاحصات:

عیت نقاط انفصال نرم افزاری در پردازنده های خانواده x86 دراینست که هنگامی که Bp بوسیله کاربر مشخص میشود دیباگر باید کد برنامه هدف را تغییر دهد.
این مکانیسم در بعضی حالات باعث شناسایی دیباگر توسط برنامه هدف میشود.

در کد زیر یک نمونه از روال تشخیص دیباگر را میبینید.

```
int main(int argc, char* argv[])
{
    // The ciphered string "Hello, Free World!"
    char s0[]="\x0C\x21\x28\x28\x2B\x68\x64\x02\x36\
               \x21\x21\x64\x13\x2B\x36\x28\x20\x65\x49\x4E";

    __asm
    {
        BeginCode:                  ; The beginning of the code being
debugged
            pusha                   ; All general-purpose registers are
saved.
            lea ebx, s0             ; ebx=&s0[0]
            GetNextChar:            ; do
                xor eax, eax        ; eax = 0;
                lea esi, BeginCode  ; esi = &BeginCode
                lea ecx, EndCode    ; The length of code
                sub ecx, esi        ; being debugged is computed.
                HarvestCRC:         ; do
                    lodsb           ; The next byte is loaded into al.
                    add eax, eax    ; The checksum is computed.
                loop HarvestCRC     ; until(--cx>0)
                xor [ebx], ah       ; The next character is decrypted.
                inc ebx             ; A pointer to the next character
                cmp [ebx], 0        ; Until the end of the string
            jnz GetNextChar         ; Continue decryption
            popa                    ; All registers are restored.
        EndCode:                    ; The end of the code being debugged
            nop                     ; A breakpoint is safe here.
    }
    printf(s0);                     //The string is diplayed.
```

```
    return 0;
}
```

هنگامی که برنامه را بدون حضور دیباگر اجرا کنید برنامه کلیمه "Hello,Free World!" را برروی صـفحه نمـایش چـاپ میکند ولی مادامی که برنامه را با یک دیباگر باز کنید و بر روی یکی از دستورالعملهای آن یک SW Bp بگذارید کلمه دیگری چاپ خواهد شد!

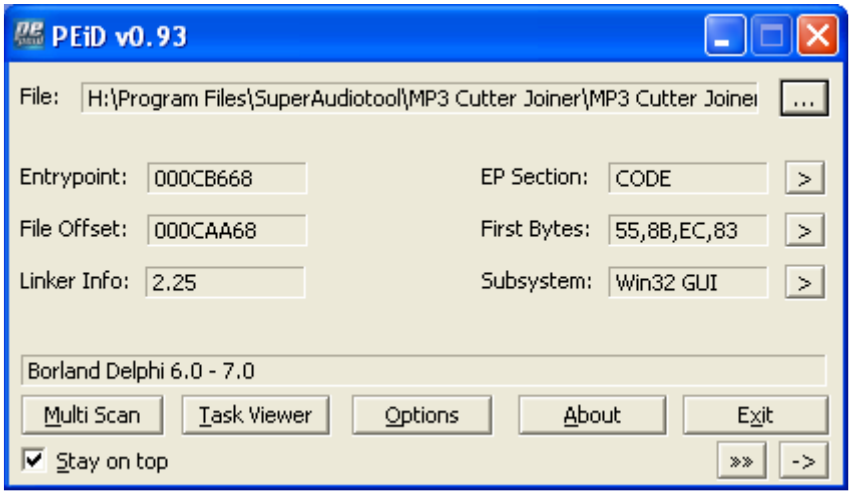
میبینید که بوسیله یکسری روال تشخیص دیباگر ، کرکر به راحتی میتواند اشتباه کند به همین دلیل همیشه باید از ابزاری بـرای پنهان کردن دیباگر استفاده نمود تا راه را اشتباه طی ننمونند.

پایان

آموزش کرکینگ با ollydbg قسمت نهم
تاریخ : اردیبهشت ۱۳۸۵ April 2006
نوع حمله Serial,inline patching
نرم افزار هدف : MP3 Cutter Joiner v1.8
نرم افزارهای مورد استفاده : ollydbg 1.10,PEiD 0.93
سطح : متوسط

مقدمه:
در این آموزش به نوع دیگری از محافظت از نرم افزار آشنا میشوید که اصطلاحاً به آن بایت جادویی (magic byte) میگویند.
در واقع این بایت جادویی جایی در حافظه و یا هارددیسک میباشد که مقداری برابر با 0 و یا 1 که همان True، False میباشد را دارد میباشد که اگر برابر Trueویا 1 باشد شما اجازه دسترسی به تمام امکانات برنامه را خواهید داشت.
قبل از شروع این فصی به شما پیشنهاد میکنم فصلهای ۷ و ۸ را به دقت بخوانید.

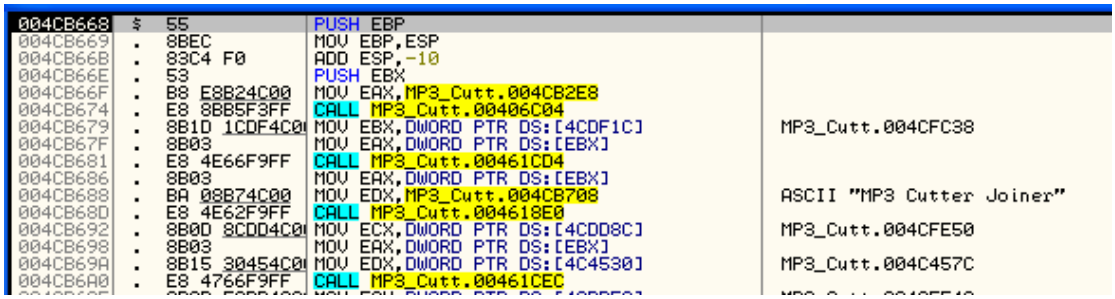
خب اول نرم افزار رو نصب میکنیم.
بعد از نصب همیشه اولین کاری که باید بکنیم این هست که اول بفهمیم برنامه رو با چی نوشتن و با چی پک شده برای اینکار میتوانید از نرم افزارهایی مانند PEid و یا RDG Packer Detector استفاده کنید من اینجا از PEid استفاده کردم.
برنامه PEid رو باز کرده و فایل MP3 Cutter Joiner.exeرو بندازید توش و یا از دکمه ...(open) برای تستِ فایل استفاده کنید



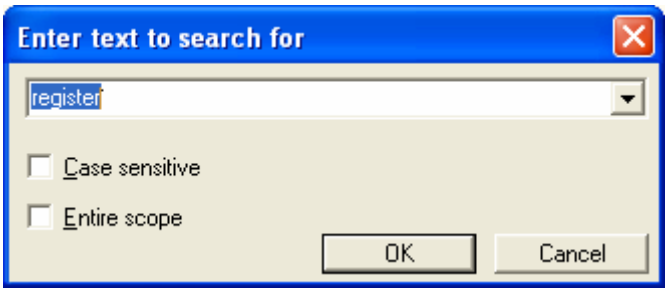
شکل ۱

پیدا کردن بایت جادویی:

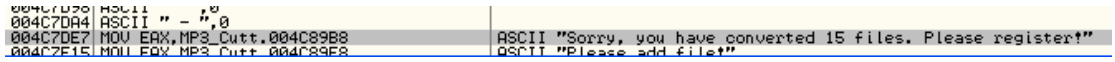
خب برنامه را بوسیله ollydbg باز کنید.



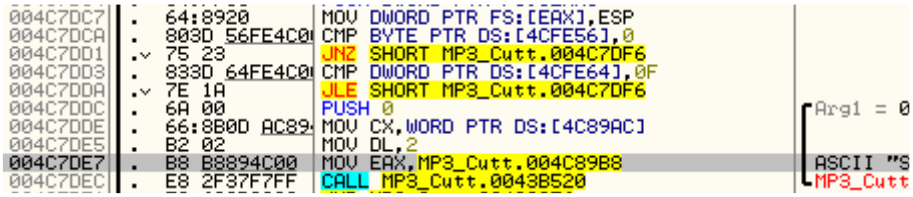
در اینجا فرض من بر اینست که شما فصلهای قبلی را بادقت مطالعه کرده اید و به همین خاطر سریعتر به اصل مطلب میپردازم.
دکمه سمت راست موس را زده و گزینه search for و بعد گزینه search referenced text strings را انتخاب کنید
در پنجره ظاهر شده نوار پیماش را کاملاً به سمت بالا بیاورید و بعد دکمه سمت راست موس را زده و گزینه search for text را انتخاب کنید و مانند شکل زیر گزینه register را در آن وارد کنید:



حال دکمه ok را بزنید ، میبینید که ollydbg شما را بر روی خطی که نوشته شده است RegisterAutomation متوقف میکند این آدرس ، آدرسی نیست که برای ما مفید باشید برای ادامه جستجو بوسیله کلیدهای ctrl+L جستجو را ادامه دهید تا به آدرس 4C7DE7 برسید یعنی جایی که نوشته شده است
Sorry, you have converted 15 files. Please register!



خب حال برروی همین آدرس دابل کلیک کرده تا در پنجره کد ollydbg بر روی این آدرس برویم:

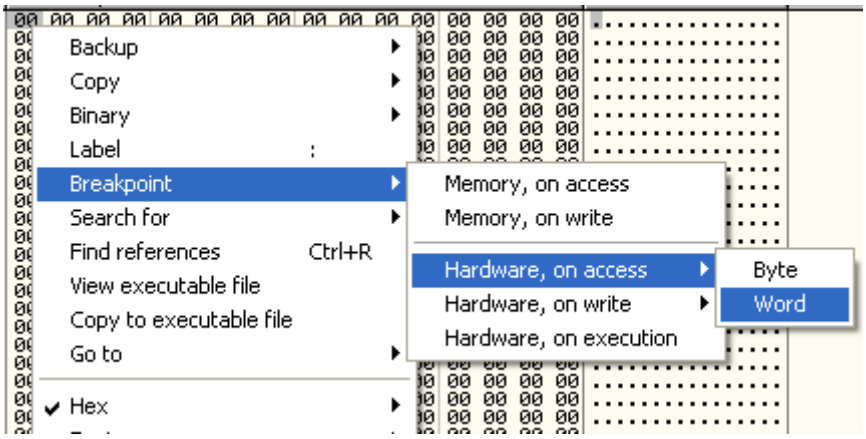


میبینید که در بالای این آدرس دو شرط پرش وجود دارد
من برای راحتی کار شما آنها را توضیح میدهم.

- در آدرس 4C7DDA نوشته شده است .... JLE short که شرط پرش بدین صورت میباشد که مقداری که در آدرس حافظه 4CFE64 وجود دارد با مقدار 0F مقایسه میشود و اگر مقداری آدرس حافظه 4CFE64 کمتر از F شود (F در دستگاه ده دهی به معنا 15 میباشد). پرش اجرا میشود.

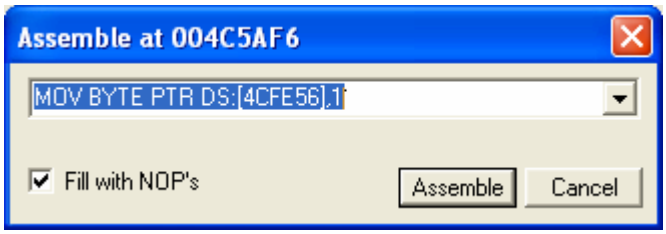
- در آدرس 4C7DD1 نوشته شده است JNZ short … که شرط پرش بوسیله دستور CMP که آدرس حافظه 4CEFE56 با مقداری 0 مقایسه میشود اگر مقدار آدرس حافظه 4CFE56 برابر با 0 نباشد شری برقرار میشود و پرش انجام میشود و دیگر شرط آدرس 4C7DDA هم اجرا نمیشود.

نتیجه گیری اینکه شرطی که در آدرس 4C7DD1 میباشد شرط کنترل این میباشد که آیا برنامه ثبت شده است و یا نه؟ و جواب اینکه آیا شرط باید اجرا شود یا نه در آدرس حافظه 4CFE56 قرار دارد. خب این آدرس را یادداشت کنید. در پنجره dump کلیدهای ctrl+g را زده و آدرس یادداشت شده را وارد نمایید و دکمه ok را فشار دهید تا ollydbg شما را به آدرس 4CFE56 ببرد مقدار این آدرس را انتخاب کرده و دکمه سمت راست موس را زده و گزینه Breakpoint و بعد گزینه hardware,on access و بعد گزینه word را انتخاب کنید. در واقع این آدرس ، بایت جادویی میباشد که ما بدنبال آن میباشیم و با گذاشتن یک نقطه انفصال برروی آن میتوانیم این بایت جادویی را کنترل کنیم و ببینیم که چه موقع و چطور تغییر میکند.دلیل اینکه از یک نقطه انفصال سخت افزاری استفاده میکنم به این دلیل میباشد که اگر از نقطه انفصال نرم افزاری در این برنامه استفاده کنم برنامه crash میکند و اجرا نمیشود.

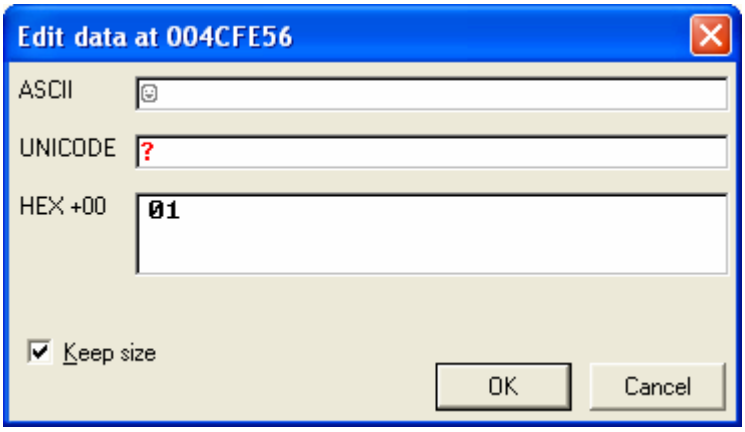


حال دکمه F9 را بزنید تا برنامه اجرا شود:
شما در آدرس 4C5AFD توقف خواهید کرد به آدرس بالای این نقطه توجه کنید. این نقطه همان جایی میباشد که بایت جادویی مقداردهی میشود.

خب برروی آدرس 4C5AF6 یک بار کلیک کرده و بعد دکمه space را بزنید و مقدار آن را بدین صورت تغییر دهید:
MOV BYTE PTR DS:[4CFE56],1



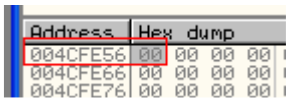
حال در پنجره dump آدرس 4CFE56 را پیدا کرده و برروی مقدار آن یکبار کلیک کرده و سپس کلید space را فشار دهید حال مقدار آنرا به 01 تغییر دهید:



برنامه را با زدن کلید F8 قدم به قدم دنبال میکنیم تا ببینیم کجا مقدار جادویی ما تغییر خواهید کرد البته هنگامی که اینکار را انجام میدهید حواستان باید کاملاً به پنجره dump و مقدار آدرس 4CFE56 باشد که مقدار آن از 01 به 00 تغییر نکند.خب اینکار انجام میدهیم تا به آدرس 4C5B9A برسیم.

```
004C5B95  .  A1 70FE4C00   MOV EAX,DWORD PTR DS:[4CFE70]
004C5B9A  .  E8 5559F5FF   CALL MP3_Cutt.0041B4F4
004C5B9F  .  803D 56FE4C0( CMP BYTE PTR DS:[4CFE56],0
004C5BA6  .v 75 0F         JNZ SHORT MP3_Cutt.004C5BB7
004C5BA8  .  A1 F0DB4C00   MOV EAX,DWORD PTR DS:[4CDBF0]
```

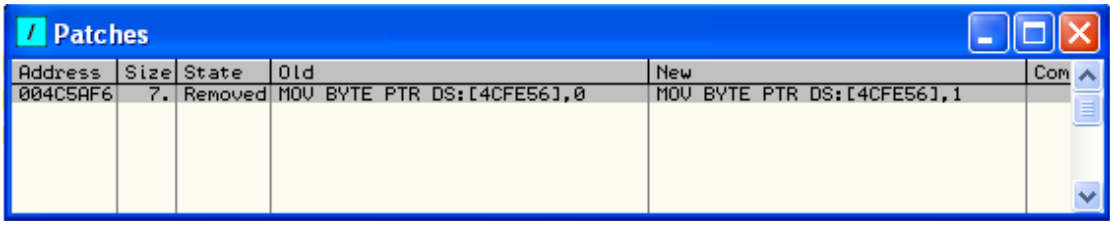
یکبار دیگر کلید F8 را بزنید خواهید دید که مقدار آدرس 4CFE56 در پنجره dump دوباره به 00 تغییر میکند.



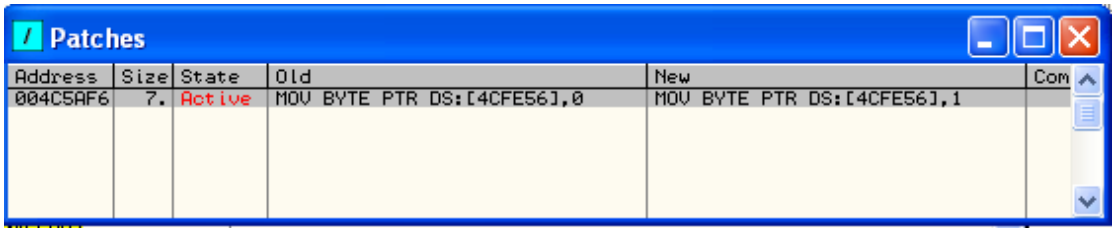
حال بر روی آدرس 4C5B9A رفته و دکمه F2 را بزنید تا یک Bp برروی آن قرار گیرد.

برنامه را با کلید ctrl+F2 ریستارت کرنید برای اینکه میخواهیم روالی که مقدار آدرس 4CFE56 را تغییر میدهد دنبال کنیم.

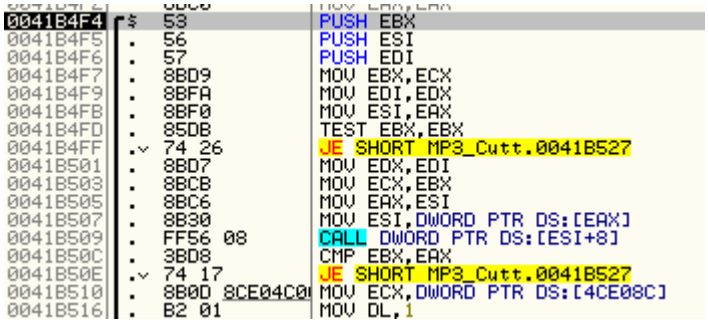
در ollydbg از منوی view گزینه patch را انتخاب کنید و یا کلید ctrl+P را فشار دهید



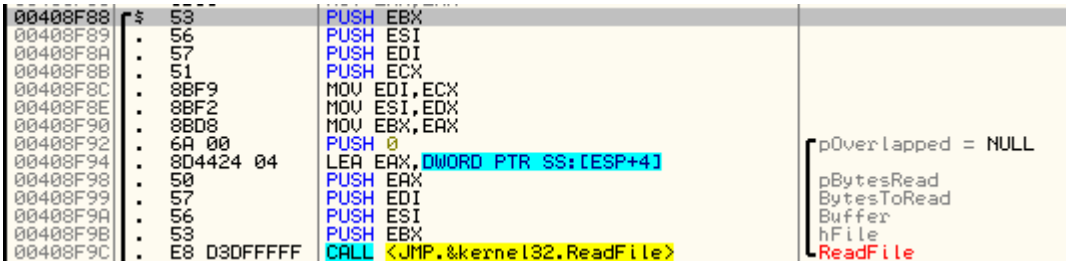
پنجره ای مانند پنجره بالا خواهید دید.
در واقع هنگامی که شما برنامه را بوسیله ctrl+F2 ریستارت میکنید تمامی patch هایی که انجام داه اید بصورت غیرفعال درمی آیند برای اینکه بتوانید patchهای خود را فعال کنید ollydbg تمامی آنها را حفظ میکند و شما بوسیله این پنجره میتوانید هرکدام را بخواهید فعال کنید کافیست برروی یکی از آنها دکمه سمت راست موس را زده و گزینه apply patch را انتخاب کنید و یا برروی آن patch یکبار کلیک کرده و کلید space را فشار دهید.



از منوی debug گزینه hardware breakpoint را انتخاب کنید و نقطه انفصالی را که قبلاً گذاشته بودیم را حذف کنید.
برنامه را با F9 اجرا کنید بعد از چند لحظه شما بر روی آدرس 4C5B9A خواهید ایستاد.
برنامه را با F7 دنبال کنید مانند شکل زیر در آدرس 41B4F4 متوقف میشوید

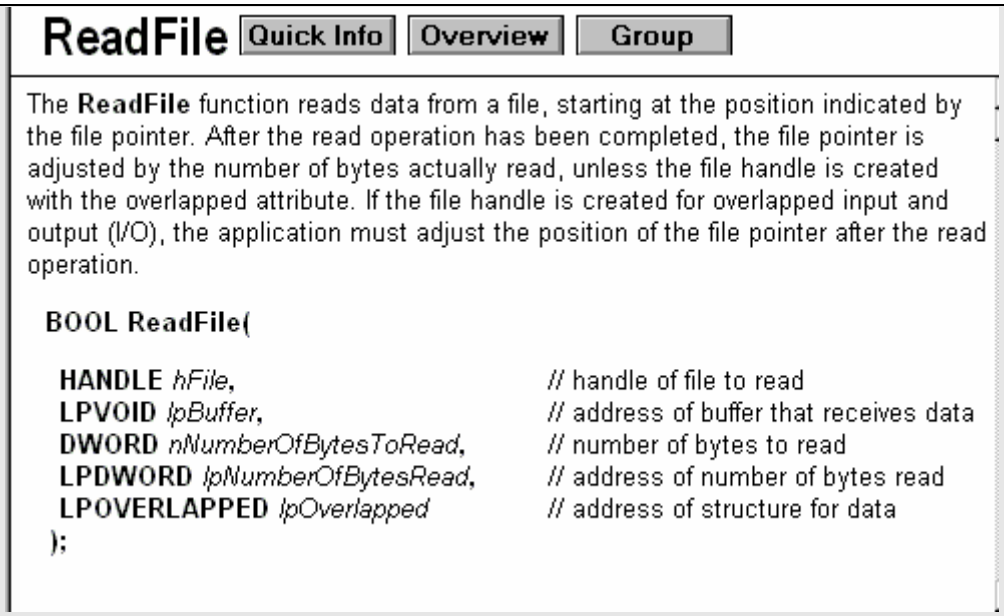


حال دوباره کلید F7 را بزنید تا به آدرس 41B509 برسید (اگر پنجره کد ollydbg شما مانند شکل نیست کافیست ctrl+a را فشار دهید تا ollydbg کدهای اسمبلی را ارزیابی کند و معادل آنها را دقیقتر به شما نشان دهد.این یک اشکال در ollydbg میباشد چراکه گاهی نمیتواند کدهای اسمبلی را بصورت کامل ارزیابی کند و گاهی کدهای اشتباه نشان میدهد).
دوباره آنقدر کلید F7 را فشار دهید تا مانند شکل زیر در آدرس 408F88 قرار گیرید



در پنجره dump دوباره کلید ctrl+g را فشار دهید و آدرس 4CFE56 را در درون پنجره بزنید و دکمه ok را فشار دهید حال به مقدار این آدرس توجه کنید هنوز 01 میباشد حال با دقت به زدن کلید F7 ادامه بدهید تا به آدرس 408F94 برسید همانطور که از شکل متوجه شده اید ما داخل یک تابع API هستیم به نام ReadFile .

تابع ReadFile:
این تابع مقدار مشخصی بایت را از یک فایل خوانده و در جای دیگری کپی میکند به شکل زیر توجه کنید:



ReadFile Quick Info | Overview | Group

The **ReadFile** function reads data from a file, starting at the position indicated by the file pointer. After the read operation has been completed, the file pointer is adjusted by the number of bytes actually read, unless the file handle is created with the overlapped attribute. If the file handle is created for overlapped input and output (I/O), the application must adjust the position of the file pointer after the read operation.

```
BOOL ReadFile(

  HANDLE hFile,                // handle of file to read
  LPVOID lpBuffer,             // address of buffer that receives data
  DWORD nNumberOfBytesToRead,  // number of bytes to read
  LPDWORD lpNumberOfBytesRead, // address of number of bytes read
  LPOVERLAPPED lpOverlapped    // address of structure for data
);
```

این تابع nNumberOfBytesToRead را از lpAddressOfBytesRead خوانده در داخل lpBuffer میریزد
ویندوز چگونه میداند که تابع ReadFile چه بایتی را باید از کجا خوانده؟
همانطور که میدانید nNumberOfBytesToRead, lpAddressOfBytesRead, lpBuffer, hFile و lpOverlapped تماماً پارامترهای تابع ReadFile میباشند که هنگامی که تابع ReadFile صدا میشود این پارامترها هم باید به دنبال آن مقدار دهی شوند و به همراه تابع صدا شوند.

در رابطه نحوه ارجاع پارامترها به توابع API:

در کل دو روش برای ارجاع پارامترها به یک تابع وجود دارد:

- ارجاع توابع بصورت C Calling
- ارجاع توابع بصورت Pascal Calling

تابع زیر را در نظر بگیرید:
ReadFile(hFile, lpBuffer, nNumberOfBytesToRead, lpNumberOfBytesRead, lpOverlapped)

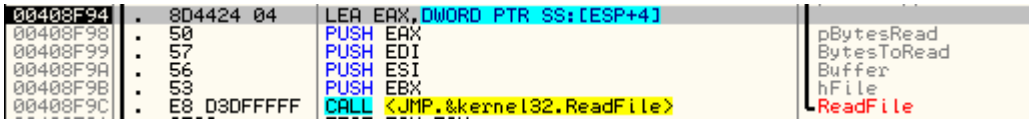
برای اینکه بتوان پارامترهای این تابع را به آن اجرجاع داد ابتدا باید باید آنها را در پشته قرار دهیم (PUSH) بسته به نحوه استفاده ما از نوع ارجاع مشخص میشود که کدامین پارامتر در ابتدا باید از حافظه خوانده شود.
اگر نحوه ارجاع ما بصورت C Call Convention باشد اول پارامتر lpOverlapperd بازخوانی میشود.
به صورت زیر:

```
PUSH lpOverlapped
PUSH lpNumberOfBytesRead
PUSH nNumberOfBytesToRead
PUSH lpBuffer
PUSH hFile
CALL Kernel32.ReadFile
```

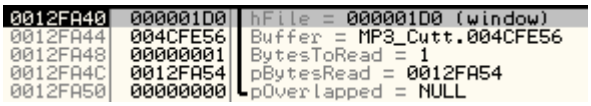
(C Call Convention پارامترها را از راست به چپ میخواند)
اگر نحوه ارجاع پارامترها بصورت Pascall Call Convention باشد اول پارامتر hFile خواهنده میشود.

```
PUSH hFile
PUSH lpBuffer
PUSH nNumberOfBytesToRead
PUSH lpNumberOfBytesRead
PUSH lpOverlapped
CALL Kernel32.ReadFile
```
(Pascall Call Convention پارامترها را از چپ به راست میخواند)

با توضیحاتی که در بالا داده شد حالا ما میفهمیم که نحوه صدا کردن پارامترها در این برنامه بصورت C Call Convention میباشد.

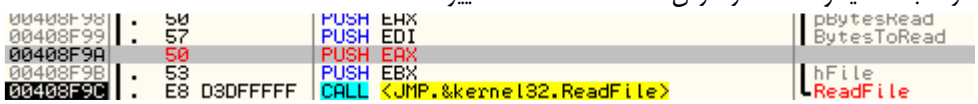


در آدرس 408F98 آدرسی که باید بایتها از آن خوانده شود در پشته قرار میگیرد ، این آدرس را میتوانید در پنجره Info که در زیر پنجره کد قرار دارد پیدا کنید در کامپیوتر من این آدرس :12FA54 میباشد.
در آدرس 408F99 مقدار بایتی که باید خوانده شود در پشته قرار میگیرد که این مقدار برابر است با 1 بایت.
در آدرس 408F9A مقدار بایتی که باید نوشته شود در ESI قرار میگیرد(این مقدار در آدرس 4CFE56 که همان بایت جادویی میباشد قرار دارد).
در آدرس 408F9B مقدار Handler فایل قرار دارد این مقدار برابر یک شماره یکتا میباشد که به جای اسم فایلی مورد نظر در طول برنامه از آن استفاده میشود توجه داشته باشید که این مقدار میتواند در هر کامپیوتر متغیر باشد در کامپیوتر من این مقدار برابر است با 1D0.
حال دکمه F8 را آنقدر فشار دهید تا به آدرس 408F9C برسید یعنی جایی که تابع ReadFile صدا زده میشود حال به پنجره stack نگاه کنید:



میبینید که این تابع فقط یک بایت را از فایلی که handler آن 1D0 میباشد خوانده و آنرا در آدرس 4CFE56 مینویسد.
اگر یادتان باشد در اول فصل گفتم که اگر از نقاط انفصال نرم افزاری در آدرس 4CFE56 استفاده شود برنامه crash خواهد نمود و این به دلیل میباشد که تابع ReadFile میخواهد در این آدرس چیزی بنویسد و مادامیکه ما از یک نقطه انفصال نرم افزاری استفاده کنیم اینکار صورت نمیگیرد.

حال ما نمیخواهیم که مقدار 00 از فایل مورد نظر خوانده شود و در آدرس حافظه 4CFE56 نوشته شود
برنامه را با ctrl+f2 ریستارت کنید و روال را دوباره تا آدرس 408F94 پیگیری کنید.
کاری که من میخواهم در اینجا انجام دهم اینست که مقدار buffer که در ثبات ESI ذخیره میشود را به تابع ReadFile ندهم تا این تابع نتواند مقداری 01 را از آدرس 4CFE56 حافظه پاک کند.
برروی آدرس 408F9A ایستاده(PUSH ESI) حال دکمه space را بزنید و مقدار آنرا با مقدار PUSH EAX عوض کنید اینکار ما باعث میشود که مقدار آدرس حافظه 4CFE56 تغییر نکند



میتوانید در پنجره dump آنرا مشاهده کنید که هنوز آدرس 4CFE56 دارای مقدار 01 میباشد.

حال برنامه را اجرا میکنیم.

حال میتوانید همانطور از فصلهای قبل به خاطر دارید patch خود را ذخیره کنید.

Patch کردن برنامه MP3 Cutter Joiner v1.8 از طریق پیدا کردن فایل Magic Byte:

در قسمت اول دیدید که چگون توانستیم حالت حفاظتی magic byte را شکسته و برنامه را ثبت شده جلوه دهیم حال بیایید نگاهی دقیقتر به کاری که انجام داده ایم داشته باشیم و روش دیگری را برای اینکه این قفل را از بین ببیریم بکارگیریم.

- برنامه هدف از یک مقدار Boolean برای کنترل کردن اینکه آیا برنامه ثبت شده است و یا نه استفاده میکند.
- مقدار Boolean در آدرس 4CFE56 ذخیره میشود
- برنامه در هنگام اجرا فرض میکند که برنامه ثبت نشده است و مقدار 00 را در آدرس 4CFE56 مینویسد
- سپس برنامه ۱ بایت را از فایلی خوانده و آنرا در آدرس 4CFE56 قرار میدهد.

برنامه هدف دارای ضعفهای بسیار زیادی میباشد و ما توانستیم برنامه را به گونه ای تغییر دهیم که ثبت شده به نظر برسد.
حال میخواهم بجای آنکه برنامه هدف را patch کنم فایلی را که مقدار magic byte در آن ذخیره میشود را پیدا کنم و آنجا را تغییر دهم.
اولین چیزی که نیاز داریم اینست که اسم فایل چیست ؟ و اینکه کجا ذخیره شده است؟

اگر به خاطر داشته باشید گفتیم که تابع ReadFile برای اینکه بتواند مقداری را بخواند از یک Handler استفاده مکیند.این Handler از کجا آمده؟
برای اینکه ReadFile بتواند یک محتویات یک فایل را بخواند احتیاج به این دارد که فایل را باز کند.

فرض کنید که شما یک برنامه ای نوشته اید که File1 را باز کرده و محتویات آنرا میخواند و در داخل File2 میریزد.ویندوز File1 را باز میکند و یک عدد یکتا به آن اختصاص میدهد که به آن Handler گفته میشود و نیز همین کار با File2 انجام میشود.این handler نیز همانطور در چند صفحه قبل اشاره شد در درون برنامه بجای اسم فایل بین روالهای برنامه در حال حرکت میباشد و اعمالی خاصی که شما در نظر دارید بر روی فایل انجام دهید بوسیله این Handler شناسایی شده و انجام میشود.

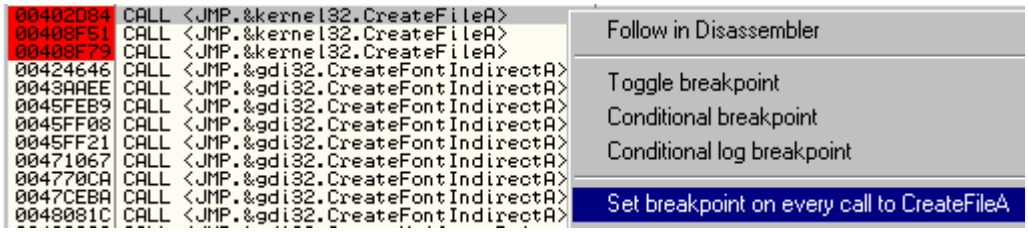
حال که به اهمیت Handler پی بردیم میخواهیم ببینیم که چه موقع ویندوز این Handler را در فایل هدف ما مشخص میکند.اینکار بوسیله تابع CreateFile انجام میشود(این تابع فقط برای ساخت یک File بکار نمیرود) برای اطلاعات بیشتر در مورد تابع CreateFile میتوانید به آدرس:

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/fileio/fs/createfile.asp

مراجعه کنید.

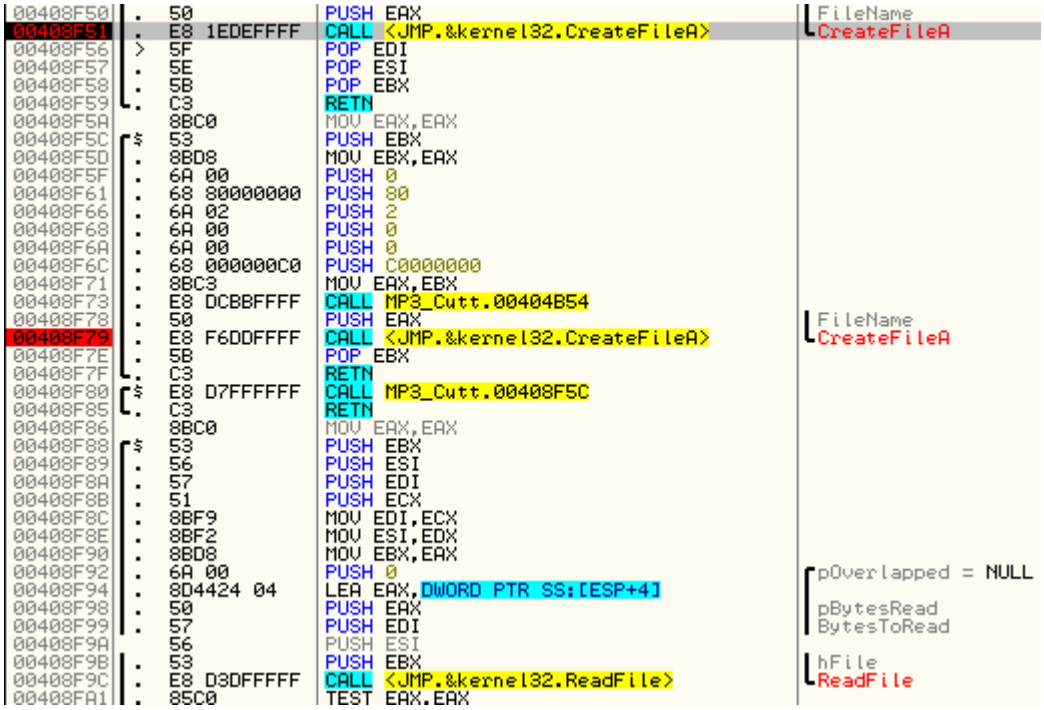
- تابع CreateFile یک File را ایجاد کرده و یا درصورت وجود باز میکند و یک عدد یکتا بنام Handler که شناسه File باز شده میباشد را بعنوان برگشتی تابع برمیگرداند

حال که چیزهای زیادی در مورد Fileها فهمیده ایم به برنامه هدف برمگیردیم حال دوباره برنامه هدف را در ollydbg باز میکنیم در پنجره کد دکمه سمت راست موس را زده و گزینه search for و بعد گزینه all intermodualar call را انتخاب میکنیم در پنجره ظاهر شده کلمه CreateFileA را تایپ کنید و بعد مانند شکل زیر بر روی یکی از توابع CreateFileA که دیده میشود دکمه سمت راست موس را زده و بعد گزینه Set BreakPoint on every call to CreateFileA را انتخاب کنید.

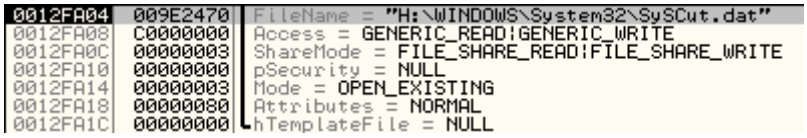


حال برنامه را با F9 اجرا کنید

بعد از چند لحظه برنامه بر روی آدرس 408F51 توقف میکند



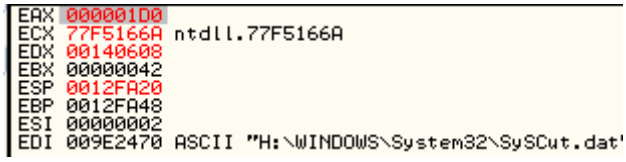
اگر کمی پایین تر از جایی که Ollydbg متوقف شده است را نگاه کنید خواهید دید که مقدار magic byte ما بوسیله تابع ReadFile بازنویسی میشود یعنی در آدرس 408F9C . این نشان میدهد که ما در جای درست قرار داریم یعنی اولین جایی که فایل magic byte باز میشود به پنجره پشته دقت کنید:



اسم File مورد نظر ما SySCut.dat میباشد که در آدرس $windir$/system32 ذخیره میشود.خب حال ببینیم که handler فایل کجا ذخیره شده است؟

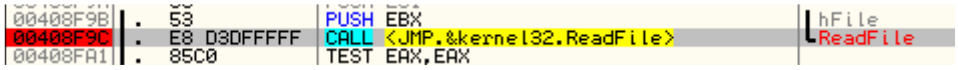
در اکثر اوقات مقادیر برگشتی توسط توابع API در ثبات EAX ذخیره میشوند.

یکبار کلید F8 را فشار دهید و به پنجره registers توجه کنید:

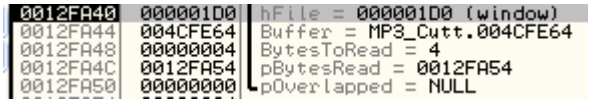


مقدار handler در کامپیوتر من 1D0 میباشد.

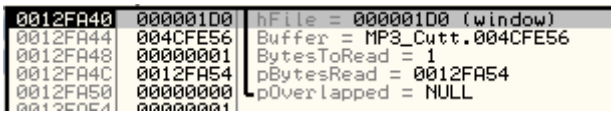
خب ما باید صد در صد مطمئن شویم که این فایل همان جایی هست که مقدار Magic Byte در آن ذخیره میشود به آدرس 408F9C بروید یعنی جایی که هنگامی که فایل بوسیله تابع CreatFile باز میشود محتویات آن خوانده میشودو کلید F2 را بزنید.



دوباره کلید F9 را بزنید تا برنامه اجرا شود اینبار شما دقیقاً در آدرس 408F9C توقف کرده اید به پنجره پشته نگاه کنید:



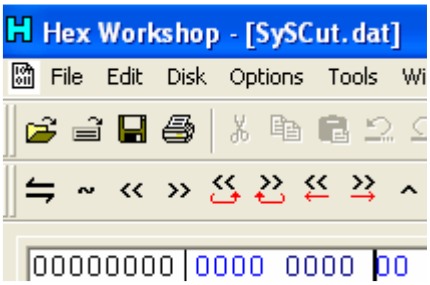
میبینید که handler مورد نظر همان 1D0 میباشد پس SyScut.dat همان file مورد نظر میباشد.
نگاهی به مقدار BytesToRead بیاندازید برابر با ۴ است این یعنی اینکه ۴ بایت باید از SySCut.dat خوانده شود ولی Magic Byte مورد نظر ما فقط ۱ بایت میباشد! خب پس الان موقع نوشتن برروی File مورد نظر نیست یکبار دیگر کلید F9 را بزنید تا برنامه به روند کار خود ادامه دهد ، شما دوباره در آدرس 408F9C توقف خواهید نمود حال دوباره به پنجره stack نگاه کنید:

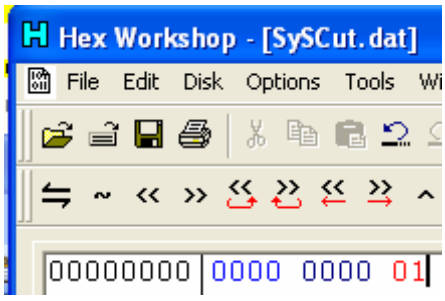


این دفعه مقدار BytesToRead برابر 1 میباشد در واقع بدین صورت میباشد:

در بار اول برنامه syscut.dat باز شده و مقدار آن خوانده شده و با مقدار 01 کنترل میشود اگر برابر شد با مقدار 01 همان مقدار دوباره در syscut.dat نوشته میشود و در غیراینصورت 00 دوباره در این فایل نوشته میشود.

حال بوسیله hex Editor پرونده syscut.dat را باز کنید و مقدار بایت آخر را مانند شکل زیر به 01 تغییر دهید و دکمه ذخیره را بزنید.





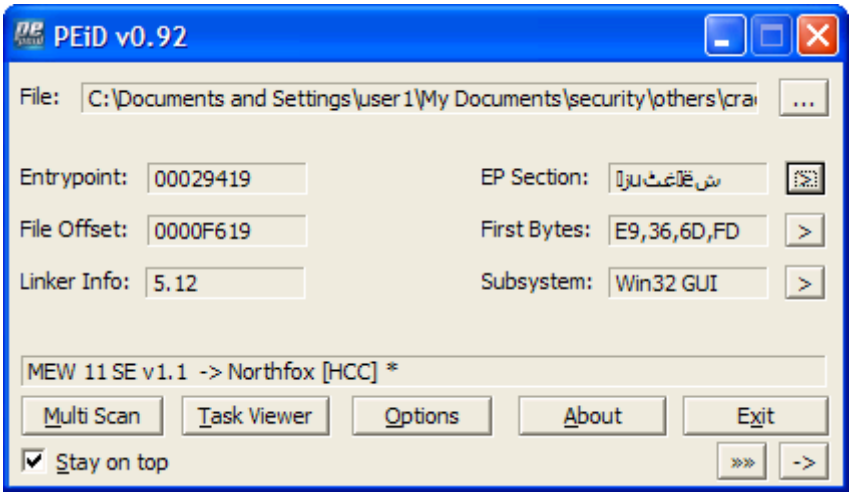
حال برنامه را اجرا کنید.میبینید که برنامه ثبت شده بنظر میرسد
پایان

ضمیمه ها:

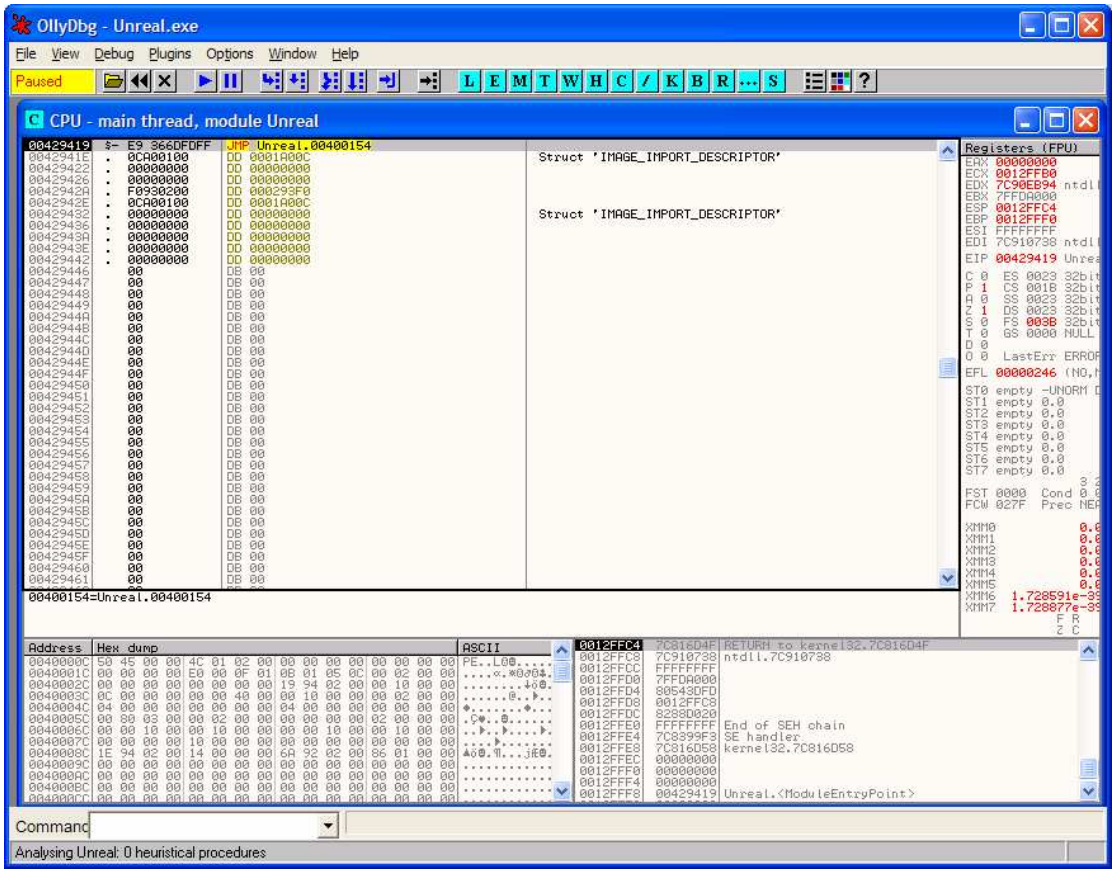
ضمیمه ۱ :

آموزش آنپک دستی MEW 11 SE v1.2
سطح : مبتدی
فایل هدف : unreal.exe
ابزارها :ollydbg 1.10,ollydump plugin,ImpREC 1.6f,PEiD

برنامه را اول با PEiD چک میکنیم:



برنامه مورد نظر را در ollydbg باز میکنیم. با دو پیغام روبرو میشویم با هر دو موافقت کرده تا ollydbg برنامه را باز کند.



برنامه را با F8 اجرا میکنیم. برنامه از آدرس 400154 شروع به اجرا شدن میکند .

```
00400154   BE 1CA04100     MOV ESI,Unreal.0041A01C
00400159   8BDE            MOV EBX,ESI
0040015B   AD              LODS DWORD PTR DS:[ESI]
0040015C   AD              LODS DWORD PTR DS:[ESI]
0040015D   50              PUSH EAX
0040015E   AD              LODS DWORD PTR DS:[ESI]
0040015F   97              XCHG EAX,EDI
00400160   B2 80           MOV DL,80
00400162   A4              MOVS BYTE PTR ES:[EDI],BYTE PTR DS:[ESI
00400163   B6 80           MOV DH,80
00400165   FF13            CALL DWORD PTR DS:[EBX]
00400167 ^ 73 F9           JNB SHORT Unreal.00400162
00400169   33C9            XOR ECX,ECX
0040016B   FF13            CALL DWORD PTR DS:[EBX]
0040016D v 73 16           JNB SHORT Unreal.00400185
0040016F   33C0            XOR EAX,EAX
00400171   FF13            CALL DWORD PTR DS:[EBX]
00400173 v 73 21           JNB SHORT Unreal.00400196
00400175   B6 80           MOV DH,80
00400177   41              INC ECX
00400178   B0 10           MOV AL,10
0040017A   FF13            CALL DWORD PTR DS:[EBX]
0040017C   12C0            ADC AL,AL
0040017E ^ 73 FA           JNB SHORT Unreal.0040017A
00400180 v 75 3E           JNZ SHORT Unreal.004001C0
00400182   AA              STOS BYTE PTR ES:[EDI]
00400183 ^ EB E0           JMP SHORT Unreal.00400165
00400185   E8 769E0100     CALL Unreal.0041A000
0040018A   02F6            ADD DH,DH
0040018C   83D9 01         SBB ECX,1
0040018F v 75 0E           JNZ SHORT Unreal.0040019F
00400191   FF53 FC         CALL DWORD PTR DS:[EBX-4]
00400194 v EB 26           JMP SHORT Unreal.004001BC
00400196   AC              LODS BYTE PTR DS:[ESI]
00400197   D1E8            SHR EAX,1
00400199 v 74 2F           JE SHORT Unreal.004001CA
0040019B   13C9            ADC ECX,ECX
0040019D v EB 1A           JMP SHORT Unreal.004001B9
0040019F   91              XCHG EAX,ECX
```

میبینیم که بعد از چندین بار اجرای برنامه با کلید F8 برنامه در آدرس 400167 و دیگر آدرسها در حال اجرای حلقه هست. با توجه به پنجره Register میتوانید حدس بزنید که تمامی توابع توکار برنامه(در اینجا برنامه با ویژال بیسیک نوشته شده است) در حال باز شدن میباشد.

```
EAX 7C910738 ntdll.7C910738
ECX 0012FFB0
EDX 7C9080C4 ASCII "NtQuerySemaphore"
EBX 0041A01C Unreal.0041A01C
ESP 0012FFC0
EBP 0012FFF0
ESI 0041A02C Unreal.0041A02C
EDI 00419C31 Unreal.00419C31

EIP 00400167 Unreal.00400167
```

برای اینکه اینکار بسیار وقت گیر میباشد نوار پیمایش را کمی پایین تر بیاورید تا به آدرس4001FD برسید یعنی آخر تابع پکر یک خط بالاتر یعنی در آدرس 4001FB دستور JNZ وجود دارد بر روی این خط کلیک کرده و بعد دکمه F2 را بزنید حال یک خط پایین تر آمده یعنی در آدرس 4001FD آنجا هم F2 بزنید.

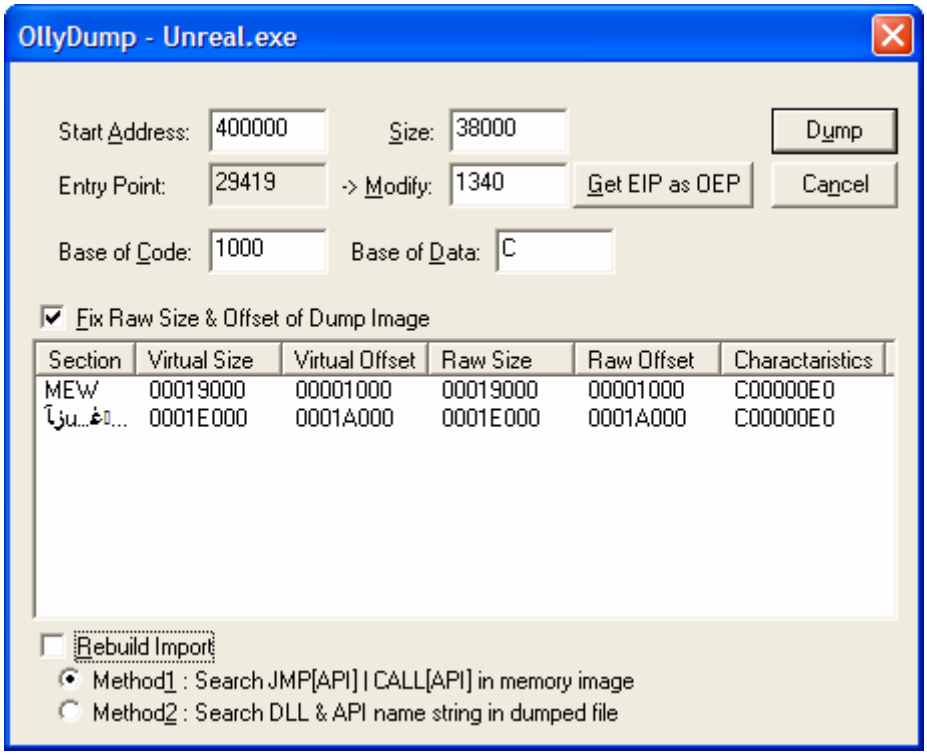
```
004001E2   56          PUSH ESI
004001E3   AD          LODS DWORD PTR DS:[ESI]
004001E4   0FC8        BSWAP EAX
004001E6   40          INC EAX
004001E7   59          POP ECX
004001E8 ^ 74 EC       JE SHORT Unreal.004001D6
004001EA v 79 07       JNS SHORT Unreal.004001F3
004001EC   AC          LODS BYTE PTR DS:[ESI]
004001ED   3C 00       CMP AL,0
004001EF ^ 75 FB       JNZ SHORT Unreal.004001EC
004001F1   91          XCHG EAX,ECX
004001F2   40          INC EAX
004001F3   50          PUSH EAX
004001F4   55          PUSH EBP
004001F5   FF53 F4     CALL DWORD PTR DS:[EBX-C]
004001F8   AB          STOS DWORD PTR ES:[EDI]
004001F9   85C0        TEST EAX,EAX
004001FB ^ 75 E5       JNZ SHORT Unreal.004001E2
004001FD   C3          RETN
```

خب حال برنامه را با F9 دنبال کنید بعد از چند لحظه ollydbg بر روی آدرس 4001FB متوقف خواهد شد حال آنقدر کلید F9 را فشار دهید تا ollydbg بر روی آدرس 4001FD متوقف شود. به پنجره register توجه کنید تمامی توابع داخلی برنامه را خواهید دید.

بعد از اینکه ollydbg برروی آدرس 4001FD ایستاد برنامه را با F8 اجرا کنید. هنگامی که یکبار کلید F8 را فشار دهید در آدرس 401340 خواهید ایستاد.

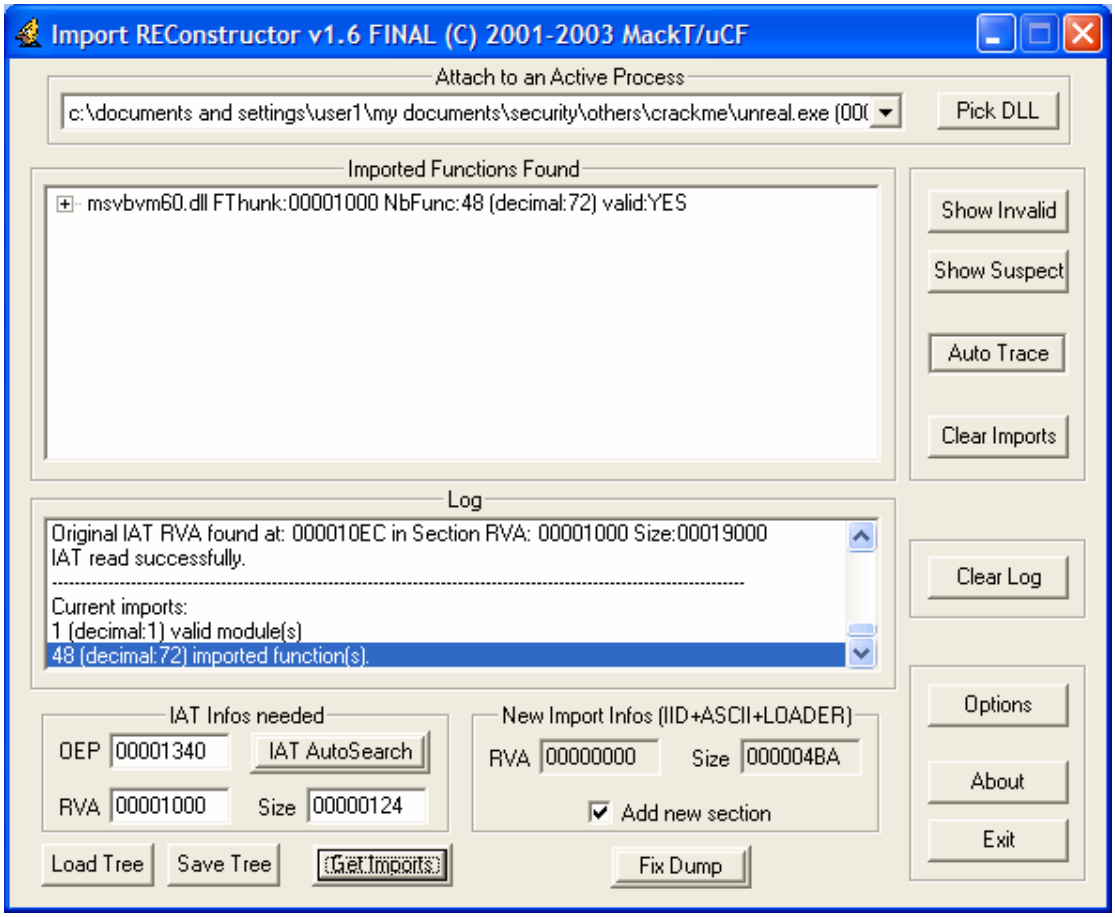
```
00401340   68          DB 68          CHAR 'h'
00401341   38          DB 38          CHAR '8'
00401342   6B          DB 6B          CHAR 'k'
00401343   40          DB 40          CHAR '@'
00401344   00          DB 00
00401345   E8          DB E8
00401346   F0          DB F0
```

حال plugin برنامه ollydump را اجرا کنید و مانند شکل زیر تنظیمات را انجام دهید.



توجه کنید آدرس OEP در آدرس 1340 میباشد آنرا یادداشت کرده و دکمه dump را زده و برنامه dump شده را با نام mup_dump.exe ذخیره کنید .

حال ollydbg را ببندید و برنامه اصلی را که pack شده میباشد اجرا کنید و برنامه ImpREC را اجرا کنید و از لیست پایین افتادنی فایل unreal.exeرا انتخاب کنید .



در textbox مربوط به IAT آدرس OEP که قبلاً یادداشت نموده اید وارد کنید و بعد از آن بر روی دکمه Get Imports کلیک کنید مانند شکل بالا.

حال بر روی دکمه Fix Dump کلیک کنید و برنامه mup_dump.exe را انتخاب کنید.

برنامه بصورت کامل آنپک شد.

ضمیمه ۲:

Ring 0

پردازنده ۴ حالت دسترسی(privilege) دارد به نامهای :Ring 0,Ring 1,Ring 2,Ring3
برنامه های عادی در Ring 3 اجرا میشوند.Ring 3 دارای محدودیتهای بسیاری میباشد برای مثال در این حالت ما نمیتوانیم ثباتهای دیباگ را خوانده و بالطبع دستورالعملهای داخل این ثباتها کار نخواهند کرد(مثال mov eax,dr7).
برای یک کرکر بسیار بهتر است که حق دسترسی بیشتری داشته(منظور اینکه نیازمند Ring 0 هستیم).اما یک مشکلی وجود دارد و آن اینکه فقط برنامه های مخصوصی به Ring 0 دسترسی دارند برای مثال device driver ها در Ring 0 اجرا میشوند اما برنامه نویسی device driver ساده و سریع نمیباشد.
خوشبختانه ویندوز دارای حفره های امنیتی بسیاری میباشد مخصوصاً ویندوز 9x راهها بسیاری برای سوئیچ کردن از Ring 3 به Ring 0 در ویندوز 9x وجود دارد.این راهها بوسیله ویروسهای برای نفوذ به سیستم مورد استفاده قرار میگیرد.
متدهای توضیح داده شده در این مقاله فقط قابلیت استفاده در ویندوزهای 9x را دارا میباشد.
بسیاری از این حفره در ویندوزهای NT based شناسایی شده و مشکلات آنها برطرف گردیده است. برای اینکه در ویندوزهای NT based بتوانید وارد Ring 0 بشوید باید یک Device Driver بسازید در ضمن در ویندوزهای NT based کاربر برای اینکه بتواند کد شما را اجرا کند باید حتماً حق دسترسی admin را دارا باشد در غیر اینصورت کدهای شما اجرا نخواهد شد.

سوئیچ کردن از Ring 3 به Ring 0 بوسیله LDT(جدول توصیفگر محلی):

قدیمیترین روشی که به ندرت از آن استفاده میشد.این بهترین راه نمیباشد ولی بهتر از روش IDT میباشد بدین دلیل که افراد کمی از آن باخبرند:

```
.386p
.MODEL FLAT,STDCALL
locals
jumps
UNICODE=0
include w32.inc

Extrn SetUnhandledExceptionFilter : PROC

.data

msg1            db "Switch to Ring0 by LDT",0
msg2            db "Ring0 activated",0

gdt_            df 0
call_           dd 00
                dw 0Fh
o_gate          dw 0
                dw 028h             ;segment for RING0
                dw 0EC00h
                dw 0
.code

Start:
```

```
                mov  eax, offset ring0
                                ;our Ring0 routine

                mov  [o_gate],ax   ;set address of our new Ring0
service to our "callgate"
                shr  eax,16
                mov  [o_gate+6],ax

                xor  eax, eax
                sgdt fword ptr gdt_
                                ;save GDT
                mov  ebx,dword ptr [gdt_+2]
                                ;GDT base address
                sldt ax
                add  ebx,eax       ;discriptor address

                mov  al,[ebx+4]
                mov  ah,[ebx+7]
                shl  eax,16        ;LDT address

                mov  ax,[ebx+2]    ;callgate's discriptor address

                add  eax,8
                mov  edi,eax       ;set in callgate for changes
                mov  esi,offset o_gate
                                ;our "callgate" address
                movsd              ;move it to real callgate
                movsd              ;for jump to Ring0

                call fword ptr [call_]
                                ;jump to Ring0 to our Ring0
service

                xor  eax, eax
                sub  edi,8         ;delete our changes in callgate
                stosd
                stosd


                call MessageBoxA,0, offset msg2, offset msg1,0

                call ExitProcess, -1


;-------------------------------------------------------------
;Our new Ring0 service
;-------------------------------------------------------------

ring0:
                mov eax, dr7       ;test for Ring0

                retf               ;back to RING3

ends
end Start
```

من یک روش دیگر در اینترنت پیدا کردم که بوسیله SoPinKy نوشته شده و همانند کد بالا میباشد ولی بزبان C:

```
Main.CPP
-----------------------------------CUT------------------------------
--------
#include <WINDOWS.h>
#include "DirectHackers.h"


//it is a example of a proc in Ring 0
Ring0Proc()
{
InitRing0();
__asm
     {
          int 20h                        //get current vm
         _emit 0x01                      //Function ID
             _emit 0x00
         _emit VMM_ID                    //VXD ID
             _emit 0x00
                                         //in ebx i have the handle
                                         //of virtual machine
     }

RetCallback;
};

int WINAPI WinMain(HINSTANCE hInstance,HINSTANCE hPrevInstance,
                                    PSTR lpCmdLine,int nCmdShow)
    {
        MSG msg ;
        DWORD a;
        int x;
        __asm pusha
        InitDirectH();
        CallRing0((unsigned int)Ring0Proc);
                __asm popa
        return 0;
    }

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;A .h
Files;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
DirectHackers.h
-----------------------------------CUT------------------------------
--------
#ifndef __DirectHackers_h
#define __DirectHackers_h

#include "VMMStruct.h"
                                         //Data
DWORD VM=0,TR=0;
Control_Block *VMCBSystem=0,*VMCB=0;
DWORD esp3;
WORD cs3,ds3,es3,sp3,fs3,gs3;            //State of ring 3 register
WORD cs0,ds0,es0,fs0,gs0;                //State of ring 0 register

Comp *Callb,Callbcpy;                    //a callbacks


                                         //to return of ring 0
#define RetCallback  \
    _asm sti      \
    _asm pop edi  \
    _asm pop esi  \
```

```
    _asm pop ebx  \
    _asm leave    \
    _asm retf;

InitDirectH()
{
FPWORD gdt;                             //Base of GDT
Descriptor *gdtdesc;
word a;
        __asm  sgdt  gdt;               //get the addres of GDT
        gdtdesc=(Descriptor *)gdt.base;

        __asm   //Save the ring 3 Segments selectors
        {
                mov cs3,cs
                mov ds3,ds
                mov es3,es
                mov sp3,sp
                mov fs3,fs
                mov gs3,gs
                mov esp3,esp
        }

                                        //Serch for the adecuate CS
for(a=0;a<(gdt.limite>>3);a++)
{
gdtdesc=(Descriptor *)(gdt.base+((DWORD)0x08*a));
if(gdtdesc->limit_l==0xffff &&
   gdtdesc->base_l==0x0000  &&
   gdtdesc->base_m==0x00    &&
   gdtdesc->access==0x9b    &&
   gdtdesc->limit_h== 0xcf  &&
   gdtdesc->base_h==0x00)break;


}
cs0=a<<3;

                                        //Serch for the adecuate DS,
ES, Etc
for(a=0;a<(gdt.limite>>3);a++)
{
gdtdesc=(Descriptor *)(gdt.base+((DWORD)0x08*a));
        if(gdtdesc->limit_l==0xffff &&
        gdtdesc->base_l==0x0000  &&
        gdtdesc->base_m==0x00    &&
        gdtdesc->access==0x93    &&
        gdtdesc->limit_h== 0xcf  &&
        gdtdesc->base_h==0x00)break;
        }
ds0=a<<3;
es0=a<<3;
fs0=a<<3;
gs0=a<<3;
}

                                        //Call a proc and switch to
ring 0
CallRing0(DWORD PUNTERO)
{
FPWORD gdt;
```

```
Descriptor *gdtdesc;
Comp *Callb,Callbcpy;
FARJMP salto;
WORD h,l;
salto.offset32=0;
salto.seg=0x08;

        __asm  sgdt  gdt;
         gdtdesc=(Descriptor *)(gdt.base+8);
         Callb=(Comp *)(gdt.base+8);

         Callbcpy.sel=Callb->sel;       //make a copy
         Callbcpy.attrib=Callb->attrib;
         Callbcpy.offs_l=Callb->offs_l;
         Callbcpy.offs_h=Callb->offs_h;

         Callb->sel=cs0;
         Callb->attrib=0xec00;;

 __asm
         {
         mov eax,PUNTERO
         mov l,ax
         shr eax,16
         mov h,ax
     }
         Callb->offs_l=l;
         Callb->offs_h=h;
 __asm {
           push ds
                   push es
                   push gs
                   push fs
       }                                //save the ring 3 segment
selectors


     __asm                              //Call the CALL GATE!!!!
         {
     cli
         call FWORD PTR salto
         }
                                        //restore de segment
selectors in ring 3
         __asm
         {
          cli
          pop fs
          pop gs
          pop es
          pop ds
     sti
         }
return;
}


InitRing0()
{
        FPWORD gdt;
        Comp *Callb;
```

```
        __asm  sgdt  gdt;
        __asm  cli
         Callb=(Comp *)(gdt.base+8);
         Callb->sel=Callbcpy.sel;
         Callb->attrib=Callbcpy.attrib;
         Callb->offs_l=Callbcpy.offs_l;
         Callb->offs_h=Callbcpy.offs_h;
         __asm
         {
         mov ds,ds0
         mov es,es0
         mov fs,fs0
         mov gs,gs0
         sti

                                        //int 3h
         int 20h
         _emit 0x08                     //get the thead handle
         _emit 0x01
         _emit VMM_ID
         _emit 0x00
          mov TR,edi
         int 20h
         _emit 0x01                     //get current vm
         _emit 0x00
         _emit VMM_ID
         _emit 0x00
          mov  VM,ebx                   //current VM handle, osea de
sistema
         sti
         }
}
#endif

-----------------------------------CUT-----------------------------
--------
VMMStruct.h
-----------------------------------CUT-----------------------------
--------
#ifndef __vmmstruct_h
#define __vmmstruct_h

                                        //definitions
#define Get_Cur_VM_Handle 0x01
#define Get_VMM_Version   0x00
#define VMM_ID 0x01
#define VDD_ID 0x0a
#define VFD_ID 0x0011f;
#define VWIN32_ID  0x0002A
#define SHELL_ID   0x00017
#define word  unsigned short
#define dword unsigned int
#define DWORD  unsigned int
#define WORD  unsigned short
#define byte  unsigned char
#define BYTE  unsigned char

                                        //Structs
#pragma pack(1)
typedef struct
```

```
{
    word  limite;
        dword base;
}FPWORD;

typedef struct
{
        dword offset32;
        word  seg;
}FARJMP;

                                        //struct of descriptors
typedef struct
{
WORD limit_l;
WORD base_l;
BYTE base_m;
BYTE access;
BYTE limit_h;
BYTE base_h;
}Descriptor;

typedef struct
{
WORD desp_l;
WORD sel;
BYTE tipo_l;
BYTE tipo_h;
BYTE desp_h;
}Idt_Descriptor;

                                        //compuertas del 386
typedef struct
{
WORD offs_l;
WORD sel;
WORD attrib;
WORD offs_h;
}Comp;

                                        //Description Block
typedef struct  {
    ULONG DDB_Next;                     /* VMM RESERVED FIELD */
    USHORT DDB_SDK_Version;             /* INIT <DDK_VERSION>
RESERVED FIELD */
    USHORT DDB_Req_Device_Number;       /* INIT <UNDEFINED_DEVICE_ID>
*/
    UCHAR DDB_Dev_Major_Version;        /* INIT <0> Major device
number */
    UCHAR DDB_Dev_Minor_Version;        /* INIT <0> Minor device
number */
    USHORT DDB_Flags;                   /* INIT <0> for init calls
complete */
    UCHAR DDB_Name[8];                  /* AINIT <"        "> Device
name */
    ULONG DDB_Init_Order;               /* INIT
<UNDEFINED_INIT_ORDER> */
    ULONG DDB_Control_Proc;             /* Offset of control
procedure */
    ULONG DDB_V86_API_Proc;             /* INIT <0> Offset of API
procedure */
```

```
    ULONG DDB_PM_API_Proc;              /* INIT <0> Offset of API
procedure */
    ULONG DDB_V86_API_CSIP;             /* INIT <0> CS:IP of API
entry point */
    ULONG DDB_PM_API_CSIP;              /* INIT <0> CS:IP of API
entry point */
    ULONG DDB_Reference_Data;           /* Reference data from real
mode */
    ULONG DDB_Service_Table_Ptr;        /* INIT <0> Pointer to
service table */
    ULONG DDB_Service_Table_Size;       /* INIT <0> Number of
services */
    ULONG DDB_Win32_Service_Table;      /* INIT <0> Pointer to Win32
services */
    ULONG DDB_Prev;                     /* INIT <'Prev'> Ptr to prev
4.0 DDB */
    ULONG DDB_Size;                     /* INIT
<SIZE(VxD_Desc_Block)> Reserved */
    ULONG DDB_Reserved1;                /* INIT <'Rsv1'> Reserved */
    ULONG DDB_Reserved2;                /* INIT <'Rsv2'> Reserved */
    ULONG DDB_Reserved3;                /* INIT <'Rsv3'> Reserved */
}Desc_Block;



                                        //Control block
typedef struct   {
    ULONG Client_EDI;                   /* Client's EDI */
    ULONG Client_ESI;                   /* Client's ESI */
    ULONG Client_EBP;                   /* Client's EBP */
    ULONG Client_res0;                  /* ESP at pushall */
    ULONG Client_EBX;                   /* Client's EBX */
    ULONG Client_EDX;                   /* Client's EDX */
    ULONG Client_ECX;                   /* Client's ECX */
    ULONG Client_EAX;                   /* Client's EAX */
    ULONG Client_Error;                 /* Dword error code */
    ULONG Client_EIP;                   /* EIP */
    USHORT Client_CS;                   /* CS */
    USHORT Client_res1;                 /*   (padding) */
    ULONG Client_EFlags;                /* EFLAGS */
    ULONG Client_ESP;                   /* ESP */
    USHORT Client_SS;                   /* SS */
    USHORT Client_res2;                 /*   (padding) */
    USHORT Client_ES;                   /* ES */
    USHORT Client_res3;                 /*   (padding) */
    USHORT Client_DS;                   /* DS */
    USHORT Client_res4;                 /*   (padding) */
    USHORT Client_FS;                   /* FS */
    USHORT Client_res5;                 /*   (padding) */
    USHORT Client_GS;                   /* GS */
    USHORT Client_res6;                 /*   (padding) */
    ULONG Client_Alt_EIP;
    USHORT Client_Alt_CS;
    USHORT Client_res7;
    ULONG Client_Alt_EFlags;
    ULONG Client_Alt_ESP;
    USHORT Client_Alt_SS;
    USHORT Client_res8;
    USHORT Client_Alt_ES;
    USHORT Client_res9;
    USHORT Client_Alt_DS;
    USHORT Client_res10;
```

```
    USHORT Client_Alt_FS;
    USHORT Client_res11;
    USHORT Client_Alt_GS;
    USHORT Client_res12;
}Client_Reg_Struc;

typedef struct Thread_Control_Block {
    ULONG   TCB_Flags;                  /* Thread status flags */
    ULONG   TCB_Reserved1;              /* Used internally by VMM */
    ULONG   TCB_Reserved2;              /* Used internally by VMM */
    ULONG   TCB_Signature;
    ULONG   TCB_ClientPtr;              /* Client registers of thread
*/
    ULONG   TCB_VMHandle;               /* VM that thread is part of
*/
    USHORT  TCB_ThreadId;               /* Unique Thread ID */
    USHORT  TCB_PMLockOrigSS;           /* Original SS:ESP before
lock stack */
    ULONG   TCB_PMLockOrigESP;
    ULONG   TCB_PMLockOrigEIP;          /* Original CS:EIP before
lock stack */
    ULONG   TCB_PMLockStackCount;
    USHORT  TCB_PMLockOrigCS;
    USHORT  TCB_PMPSPSelector;
    ULONG   TCB_ThreadType;             /* dword passed to
VMMCreateThread */
    USHORT  TCB_pad1;                   /* reusable; for dword align
*/
    UCHAR   TCB_pad2;                   /* reusable; for dword align
*/
    UCHAR   TCB_extErrLocus;            /* extended error Locus */
    USHORT  TCB_extErr;                 /* extended error Code */
    UCHAR   TCB_extErrAction;           /*      "   "   Action */
    UCHAR   TCB_extErrClass;            /*      "   "   Class */
    ULONG   TCB_extErrPtr;              /*      "   pointer */
}Thread_Control_Block;

typedef struct
{
DWORD CB_VM_Status      ;
DWORD CB_High_Linear    ;
DWORD CB_Client_Pointer ;
DWORD CB_VMID           ;
DWORD CB_Signature      ;
}Control_Block;
#endif
-------------------------------------CUT------------------------------
--------
```

بنظر من مثالهای اسمبلر خوانا تر بوده این هم روش سوئیچ کردن به Ring 0 بوسیله IDT(جدول توصیفگر وقفه) نویسنده aka EliCZ:

این روش معروفترین روش میباشد.این کد را برای اولین بار نزد دوست خود EliCZ دیدم و چندی بعد ویروس CIH نیز از این روش بهره گرفت.
بسیاری از برنامه نویسانی که میخواهند به Ring 0 سوئیچ کنند از این روش بهره میگیرند اما بعضی از ابزارهای anti-debug این روش را شناسایی میکنند(مانند frog-ice,icedump)

```
.386p
.MODEL FLAT,STDCALL
locals
jumps
UNICODE=0
include w32.inc


Extrn SetUnhandledExceptionFilter : PROC

Interrupt       equ 5              ;interrupt number which we will
use
                                   ;if you use Int 1h or 3h, it will
be
                                   ;more harder debugg your program
.DATA

msg1            db "Switch to Ring0 by IDT",0
msg2            db "Ring0 activated",0


.CODE
Start:

                push edx
                sidt [esp-2]       ;read IDT to stack
                pop  edx           ;address of Interrupt table
                add  edx,(Interrupt*8)+4
                                   ;Interrupt table base+Int
number+size for
                                   ;Int in Interrupt table=Int vector
address
                mov  ebx,[edx]
                mov  bx,word ptr [edx-4]
                                   ;read old address our interrupt
(INT 5h)

                lea  edi,InterruptHandler
                mov  [edx-4],di
                ror  edi,16        ;set our new interrupt handler
                mov  [edx+2],di

                push ds            ;save registers
                push es


                int  Interrupt     ;jump to Ring0 (our int 5h
handler)

                pop  es            ;restore registers
                pop  ds


                mov  [edx-4],bx    ;set old int 5h handler
                ror  ebx,16
                mov  [edx+2],bx

                call MessageBoxA,0, offset msg2, offset msg1,0
                call ExitProcess, -1
```

```
;-------------------------------------------------------------------
---------
;OUR NEW INT 5h HANDLER (it run in Ring0)
;-------------------------------------------------------------------
---------

InterruptHandler:

                mov  eax,dr7       ;test for Ring0
                iretd              ;jump back to Ring3



ends
end Start
```