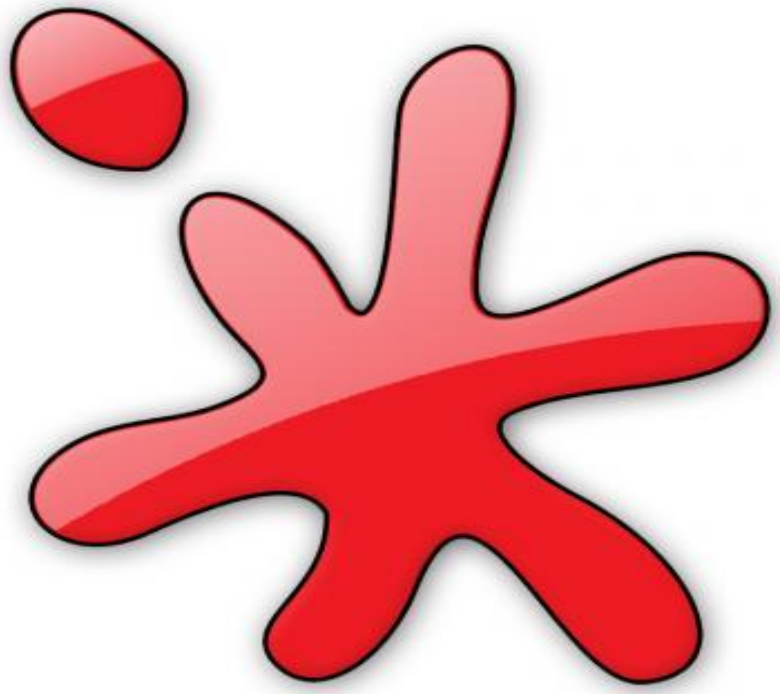
دانش آنکه می داند فراگیر و علم خود را به آنکه نمی داند بیاموز
پس اگر انجام دادی آن را ، داناشوی و به آنچه ندانی و از هرچه
که آموخته ای سودمند شوی (غررالحکم 4567)

2010

Learn OllyIce Debugger - 2nd Edition
Private Learning



HamiD.Rezaei - AHA[godvb]

Sadegh.PM

wWw.4XMen.ir

AHAx92@Yahoo.com

Private: 27/1/2010

Public: 31/8/2013

[سلام. خداوند را شکر که عمری باقی ماند و توانستم این مقاله را کامل تر کنم تا بقولی ویرایش دوم آن ، در اختیار شما قرار بگیرد.

امیدوارم این مقاله هرچند اندک ، شروع خوبی برای شما در این زمینه باشد . هدف اصلی بالابردن سطح علمی و آشنایی با این فنون است و هرگونه سو استفاده بعهده استفاده کننده می باشد.

تنها خواستار بنده از شما در هربار مطالعه این مقاله ، فرستادن سه صلوات است :

1. تعجیل در ظهور آقا امام زمان (عج)
2. برای خودتان
3. برای نویسندگان این مقاله ☺

با تشکر]

در هر زبان برنامه برنامه نویسی به طور حتم از امکانات دیباگرها (Debugger) استفاده شده برای مثال ما از آنها برای کشف و رفع خطا و ..... در برنامه نوشته شده مان استفاده می کنیم .

انواع دیگری از دیباگرها نیز هستند که بین سیستم عامل و برنامه اجرایی قرار گرفته و امکان کنترل ، تحلیل و بررسی کدهای کامپایل شده و فایل های اجرایی است .به طورکلی این نوع Debugger ها به دو سته سیستمی یا ring0 و کاربری یا ring3 تقسیم می شوند. دیباگرهای سیستمی معمولا به منظور انجام بررسی ها بر روی کدهای سیستمی و درایورهای سخت افزاری استفاده می گردند . برای مثال SoftIce یکی از معروف ترین این نوع دیباگرها می باشد.

دیباگرهای مد کاربر نوع دیگری از دیباگرها هستند که برای انجام بررسی ها بر روی روند اجرایی نرم افزارهای کاربردی به کار برده می شود. با توجه به استفاده فراوان از این نوع دیباگرها در ادامه به آموزش یکی از پرکاربردترین و قدرتمندترین این نوع می پردازم .

» نرم افزار OLLY DBG

این نرم افزار یکی از قویترین و پرطرفدارترین دیباگرهای مد کاربر است که بهترین نسخه آن Olly Ice می باشد (که یکی از معروف ترین آنها OLLY ICE می باشد که در واقعه ورژن تغییر یافته OLLY DBG است که پچ های روی آن جهت مخفی کردن دیباگر اعمال شده است. بهترین ورژن و حرفه ای ترین ورژن OLLY DBG که پچ شده است SND DBG است. در اینجا به آموزش قسمت های مختلف Olly ICE می پردازیم. هرچند که ممکن است به باتوجه به گستردگی در نکات وامکانات این دیباگر نتوان تمام قسمت ها را به طور کامل توضیح داد .

اول از همه نگاهی به خصوصیات این نرم افزار می اندازیم :

- دیباگ برنامه های Multi Thread
- توانایی بررسی خطاهایی که در هنگام اجرای فایل اجرایی رخ می دهد
- توانایی دیباگ کردن فایل های Dll به صورت مجزا و فراخوانی توابع داخلی آنها .
- توانایی ذخیره کردن تغییرات ایجاد شده بر روی فایل های اجرایی و Dll در هنگام عملیات دیباگ
- توانایی تشخیص پارامترهای ارسالی به توابع استاندارد API در کدهای اسمبلی
- پشتیبانی از انواع جستجو در کدهای اسمبلی
- پشتیبانی از انواع نقاط توقف (BP) شرطی و غیر شرطی در هنگام دیباگ کردن .
- و . . . . . .

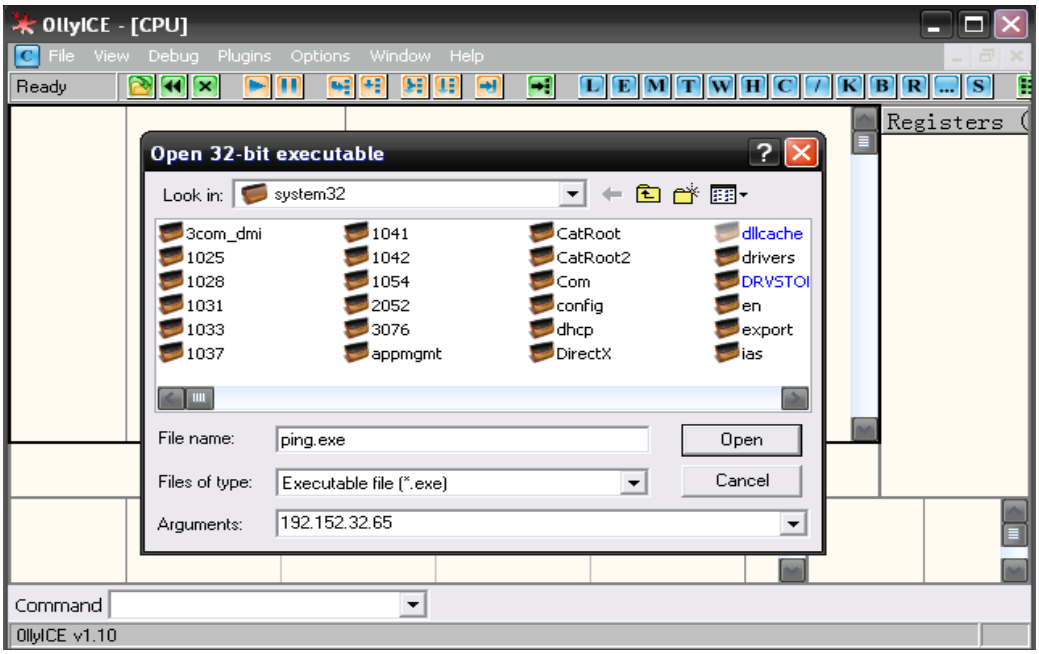
ولی برخی نکات را در مورد این دیباگر باید ذکر کرد :

- می تواند برنامه های کنسول (Console) را دیباگ کند.
- نمی تواند برنامه های .Net را دیباگ کند.
- اگر شما بر روی سیستم عامل های خانواده NT هستید ، ممکن است برای دیباگ برنامه ها نیاز به دسترسی سطح مدیر (administrator) داشته باشید.

» شروع عملیات دیباگ

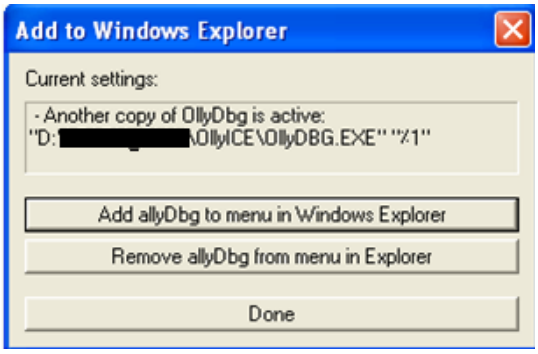
در این ابزار به سه روش میتوان عملیات دیباگ را آغاز کرد که بر حسب نیاز می توان یکی از آنها را انتخاب نمود.

باز کردن فایل از داخل برنامه :



برای این کار از منوی File → Open را میزنیم تا پنجره مربوطه باز شود . دراین پنجره یک بخش اضافی به نام Arguments دیده می شود . اگر می خوام هنگام دیباگ کردن ، فایل با پارامتری خاص اجرا و دیباگ شود باید پارامتر مربوطه را در این قسمت بنویسیم . همانطور که در تصویر می بینید ما فایل ping.exe را در حالتی دیباگ می کنیم که دارد به IP مشخص شده در بخش Argument ، ping می کند.

در روش دوم که راحتتر از قبلی هستش ما فایل مورد نظر رو از روش راست کلیک با دیباگرمان باز می کنیم . برای این کار اول باید گزینه مورد نظر رو به کلیک راست اضافه کنیم . برای این کار به منوی option → add to Explorer رو میزنیم .تا چنین پنجره ای باز شود که شرح این کادر بدین صورت است :



گزینه مورد نظر را به کلیک راست اضافه میکند

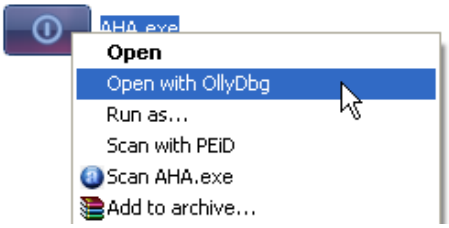
1- گزینه ایجاد شده توسط کلید قبلی از بین میبرد

2- برگشت به محیط قبل

کادر سبز رنگ هم به ما نشان می دهد که این تنظیمات قبلا ایجاد شده است یا خیر ؟ که در کادر سمت راست می بینید ما این تنظیمات را توسط نسخه دیگری تنظیم کرده ایم . این عمل 4 مسیر در رجیستری می سازد که به شرح ذیل است :

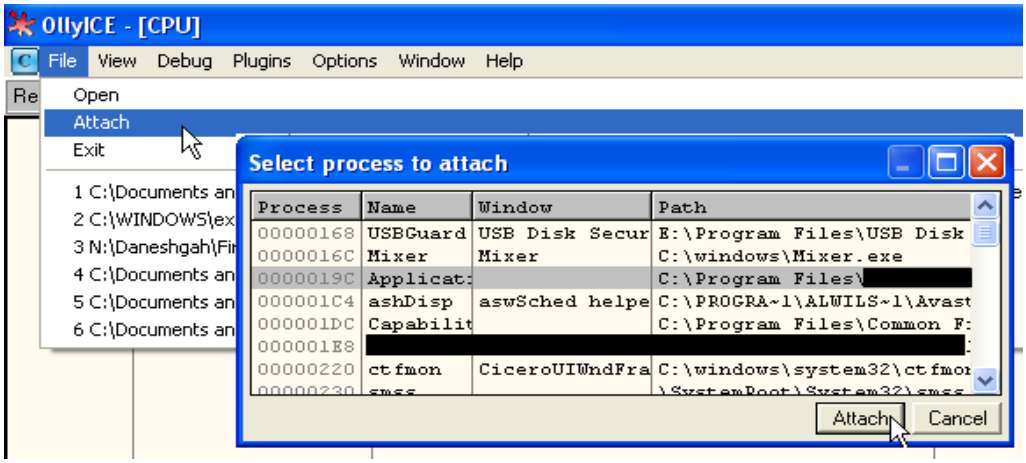
HKEY_CLASSES_ROOT\exefile\shell\Open with OllyDbg
HKEY_CLASSES_ROOT\exefile\shell\Open with OllyDbg\command
HKEY_CLASSES_ROOT\dllfile\shell\Open with OllyDbg
HKEY_CLASSES_ROOT\dllfile\shell\Open with OllyDbg\command

بعد از تنظیم این قابلیت بر روی فایل های خود (این گزینه برای فایل های Dll و Exe فعال می شود) کلیک راست کنید و از گزینه مورد نظر ، فایل خود را به حالت دیباگ ببرید .



و سومین حالت از شروع عملیات دیباگ اتصال در حال اجرا (Attach) به پروسه مورد نظر است . اگر با توابع API آشنا باشید می دانید توابعی وجود دارند که به ما این امکان را بدهد تا پروسه مورد نظر را به حالت دیباگ در بیاوریم و این تابع DebugActiveProcess می باشد که به ما اجازه اتصال به برنامه های درحال اجرا و آوردن آنها به حالت دیباگ را می دهد و

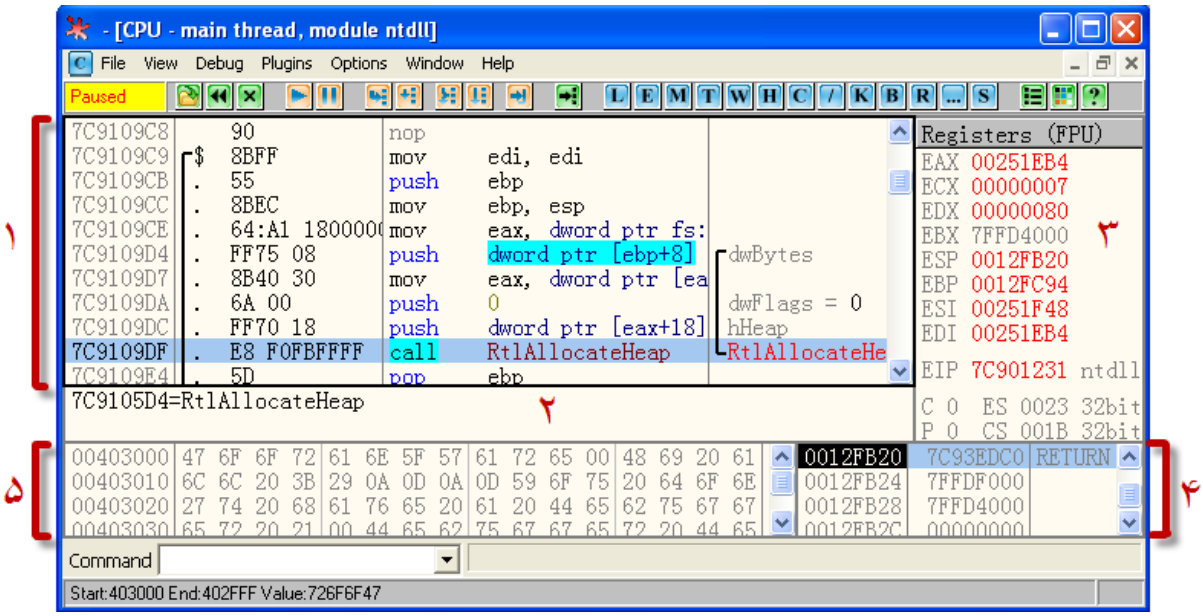
دیباگرها هم از همین توابع استفاده می کنند ☺ (بحث ما طراحی یک دیباگر نیست و بخاطر همین وارد جزییات نمی شویم). برای این کار از منوی File → Attach را انتخاب کنید تا پنجره Select Process باز شود در این پنجره تمام پروسه‌های در حال اجرا درون سیستم را همراه با اطلاعات مختصری نمایش می دهد . در این پنجره پروسه مورد نظر را انتخاب و دکمه Attach را بزنید.



به یاد داشته باشید که بعد از بستن Olly ، پروسه در حال دیباگ نیز بسته خواهد شد. هیچ وقت سعی نکنید به پروسه های سیستم Attach کنید زیرا امکان خراب شدن کامل سیستم عامل هست (برای اطمینان ، سیستم عامل در اکثر موارد اجازه Attach را به پروسه های حساس نمی دهد).

» پنجره اصلی Olly Ice

بعد از بارگذاری فایل مورد نظر توسط روش های قبل ، به پنجره اصلی این برنامه برخورد می کنیم .



همانطور که در تصویر می بینیم پنجره دیباگر دارای 5 قسمت اصلی است (قسمت های دیگر در طول آموزشی توضیح داده خواهد شد) که هرقسمت آن دارای اطلاعات خاصی درمورد فایل ماست که با کلید Tab میتوان بین آنها حرکت کرد و کلیدهای Shift + Tab جهت گردش معکوس (برخلاف جهت حرکت عقربه های ساعت) حرکت می کند . حال به بررسی هر بخش آن می پردازیم .

1– Disassembler :

در این قسمت کدهای Disassemble شده برنامه مورد نظر را نمایش می دهد .این بخش خود شامل 4 قسمت است که هر بخش دارای وظایف خاص خود هست .

1–1 Address :

در این ستون آدرس مجازی سطرها در فایل را نمایش میدهد که با دابل کلیک به آدرس نسبی نسبت به سطر جاری تبدیل می شود.

```
7C9109CC  .   8BEC          mov     ebp, esp
7C9109CE  .   64:A1 180000  mov     eax, dword ptr fs:
7C9109D4  .   FF75 08       push    dword ptr [ebp+8]      dwBytes
7C9109D7  .   8B40 30       mov     eax, dword ptr [ea
7C9109DA  .   6A 00         push    0                      dwFlags = 0
7C9109DC  .   FF70 18       push    dword ptr [eax+18]     hHeap
7C9109DF  .   E8 F0FBFFFF   call    RtlAllocateHeap        RtlAllocateHeap
```

Address

1–2 Hex Dump :

در این قسمت اطلاعات دی اسمبل نشده دستورات به صورت Hex نمایش داده می شود . همینطور که در تصویر ذیل می بینید کنار این عبارات هگز علامت های وجود دارد که این علامت ها بسیار مفیدند . مثلا برای شروع و پایان دستورهای پرشی ، ارجاع ها و . . . . با دابل کلیک در این قسمت می توان بر روی سطر مورد نظر یک BP از نوع معمولی بگذاریم که بعدا به آنها خواهیم پرداخت .

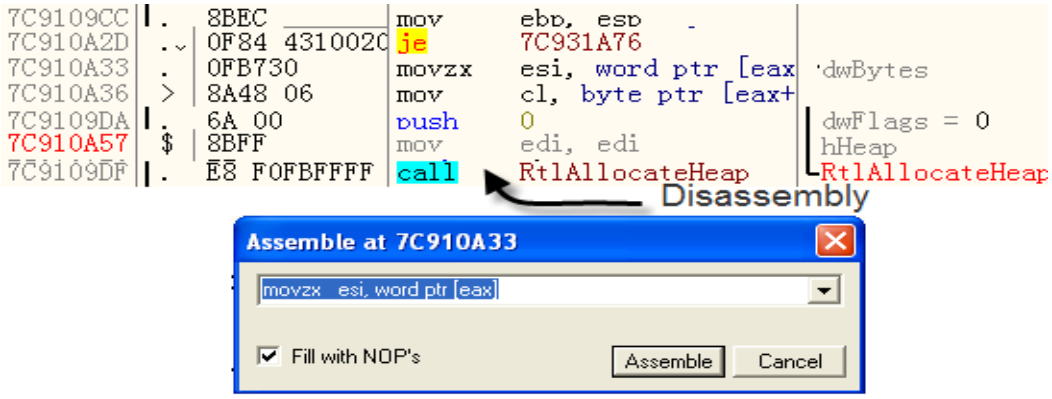
```
7C9109CC  .   8BEC          mov     ebp, esp
7C910A2D  .v  0F84 4310020  je      7C931A76
7C910A33  .   0FB730        movzx   esi, word ptr [eax     dwBytes
7C910A36  >   8A48 06       mov     cl, byte ptr [eax+
7C9109DA  .   6A 00         push    0                      dwFlags = 0
7C910A57  $   8BFF          mov     edi, edi               hHeap
7C9109DF  .   E8 F0FBFFFF   call    RtlAllocateHeap        RtlAllocateHeap
```

Hex Dump

1–3 Disassembly:

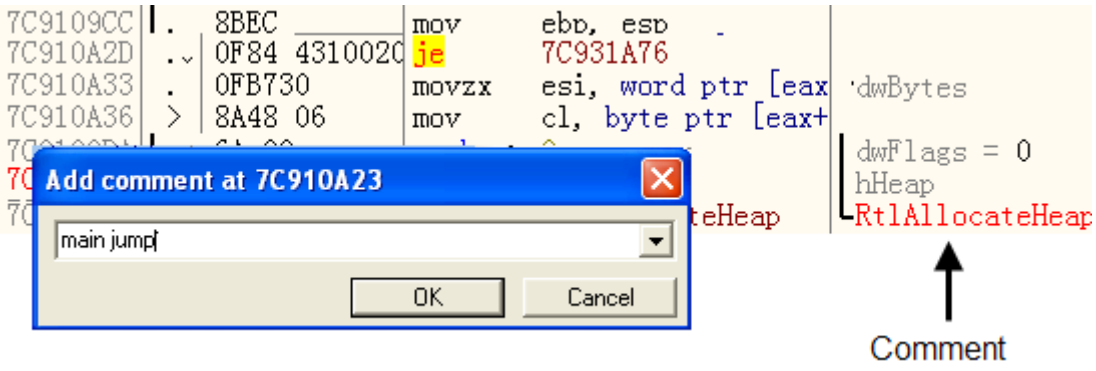
در این ستون کدهای Disassemble شده قرار می گیرد که ما می توانیم این دستورات را باتوجه به نیاز خود آنها را تغییر دهیم برای این کار می توانیم بر روی دستور مورد نظر دابل کلیک کنیم و یا کلیدSpace رافشاردهیم که معادل همان کلیک راست و انتخاب گزینه Assemble است ، تا پنجره Assemble نمایش داده شود. البته باید توجه کرد که دستور جدید دارای فضای بیشتری از

دستور قبلی نباشد چون بیشتر باشد ممکن است دستور بعدی را از بین ببرد. اگر تیک گزینه Fill with NOP را بزنیم درصورتی که دستور جدید دارای فضای کمتری باشد مابقی فضای مانده با NOP پر می شود .(NOP = No Operating یعنی هیچ عملی انجام نمی دهد.) پس همیشه سعی کنید دستورات جدید دارای فضای برابر و یا کمتر از دستور قبلی باشد .



Comment 1–4 :

در این قسمت می توانیم برای دستور مورد نظر توضیحاتی قرار دهیم که در کرک های پیچیده کار مارا راحت می کند . برای اضافه کردن توضیحات می توان بر روی سطر مورد نظر دابل کلیک کرد و یا کلید " ; " را فشار دهیم . در بعضی مواقع مانند تصویر بالا می بینید که خود برنامه توضیحاتی گذاشته است . این نوع توضیحات مربوط به آنالیزر برنامه می شود ، توابعی مانند API را می شناسد و برای راحتی کار ، خودش توضیحاتی میگذارد . اگر سطر ما دارای فراخوانی توابع API باشد ما میتوانیم با فشار دادن کلیدهای Ctrl + F1 توضیحی درباره آن تابع بدست بیاوریم . البته Olly این توضیحات را از درون یک فایل به نام win32.hlp دریافت می کند . شما باید این فایل را جداگانه دریافت و آن را از منوی Help → Select API Help انتخاب کنید . معمولا این فایل در کنار برنامه قرار دارد .

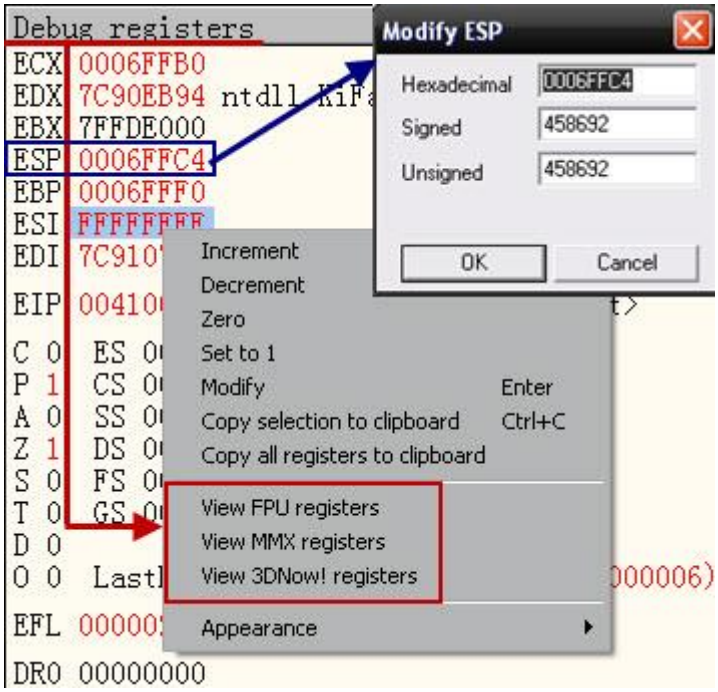


Information–2 :

این قسمت اطلاعات جالبی در مورد دستورالعمل فعلی را نمایش می دهد . این اطلاعات در هنگام اجرای خط به خط راهنمای خوبی است .

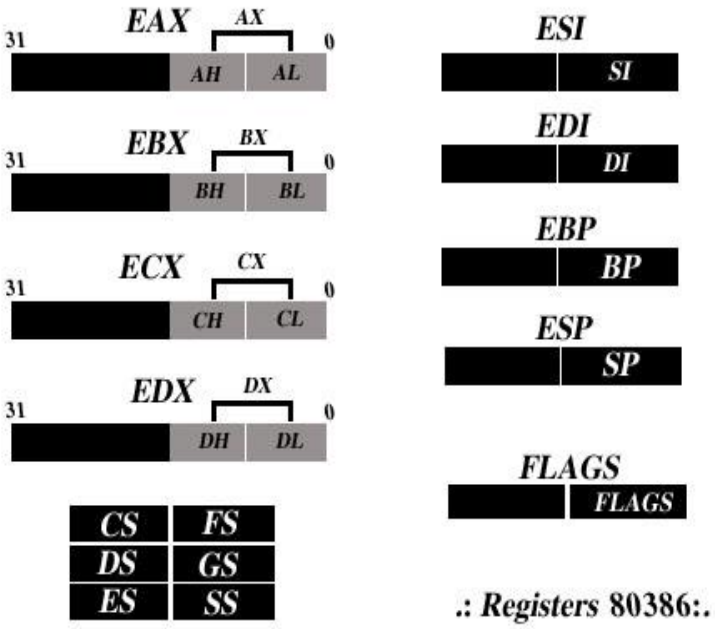
3– Registers:

در ایـن قسـمت مقـادیر فعلـی ثبـات ها(رجیسـتر) سیسـتم را نمـایش مـی دهـد. بـرای رفـتن بـه حالـت نمـایش هـای , FPU Debug ,3DNOW,MMX می توان بر روی نوار عنوان این قسمت کلیک کرده وحالت را تغییـر داد و یـا مـی تـوان بـا اسـتفاده از منوی صفحه هم این کار را کرد در تصویر ذیل می بینید که ما در هرحالتی باشیم آن از منوی صفحه حذف مـی شـود ، الان در حالـت نمایش Debug هستیم و در کادر قرمز رنگ گزینه ای برای نمایش حالت Debug ندارد . برای تغییر دادن مقدار فعلی ثبـات هـا بایـد مقدار آن را انتخاب و بعد Enter را زد ، یا اینکه بر روی آن دابل کلیک کرد تا پنجره Modify به نشان داده شود و یا از منوی صفحه گزینه Modify را انتخاب کند . برای تغییرات کم و سریع می توان از کلیدهای + برای افزایش و – برای کاهش بـه مقـدار یـک واحـد استفاده کرد.



3-1 ثبات ها (Register)

ثبات ها مکان های از CPU هستند که برای ذخیره سازی و استفاده از داده ها (دستیابی به آدرس های حافظه و I/O ، انجام محاسبات ، کنترل اجرای دستورات و .....) استفاده می شود . که در شکل ذیل ثبات های اصلی در پردازنده +80386 را مشاهده می کنید .



.: Registers 80386:.

ثبات ها به گروههای ذیل تقسیم می شوند :

1) ثبات های عمومی :

EDX, DX, ECX, CX, EBX, BX, EAX, AX ثبات های عمومی هستند که میتوان گفت مبنای کار سیستم هستند. این ثبات ها ، که می توانیم نیمی از فضای آن را مورد استفاده قرار دهیم منحصر به فرد هستند . ثبات های 16 بیتی یا همان 2بایتی(حالتDos) (DX , CX , BX , AX) به دوقسمت یک بایتی کم ارزش(Low) و پر ارزش (High) تقسیم میشوند. بعنوان مثال ثبات AX شامل دو قسمت AH (پر ارزش) و AL (کم ارزش) میباشد . بطور کلی ثبات های 16 بیتی اینگونه نام گذاری میشود xH ,xL که متغییر x نشان میدهد ثبات **Accumulator , Base , Count** و یا . . . است و حروف H نمایانگر قسمت پر ارزش (High) و حرف L نمایانگر قسمت کم ارزش (Low) می باشد . بعنوان مثال ثبات AX شامل دو قسمت AH (پر ارزش) و AL (کم ارزش) میباشد . برای درک بهتر صبر کنید یک مثال بهتر بزنم :

اگر AX رو به صورت ABCD نمایش دهیم ، CD در AL و AB در AH قرار میگیرد . حالت ثبات ها در نسخه 32 بیتی اینگونه است : EAX = xxxxABCD (حرف E به معنای Extended می باشد) که AB معرف AH و CD معرف AL می باشد (به تصویر بالا دقت کنید) . حالا یک مثال ساده در ویندوز :

حالت ثبات ها در نسخه 32 بیتی اینگونه است : EAX = xxxxABCD ، که AB معرف AH و CD معرف AL می باشد (به تصویر بالا دقت کنید)

- **ثبات های EAX / AX**

این ثبات در دستور العمل های محاسباتی به عنوان عملگر اصلی محسوب میشود . علاوه بر آن از این ثبات در اعمال ورودی خروجی و رشته ها استفاده می شود .

- **ثبات EBX / BX**

از این ثبات در عملیات محاسباتی و نیز به عنوان شاخص برای آدرس دهی ها استفاده می شود.

- **ثبات ECX/CX**

از این ثبات معمولا برای شمارش گر در حلقه ها استفاده میشود .و در بعضی مواقع در شیفت ها و اعمال محاسباتی استفاده می شود.

- **ثبات EDX/DX**

این ثبات که به ثبات داده معروف هستش که در برخی اعمال ورودی ، خروجی که احتیاج به استفاده از مقدارهای بزرگ هستند استفاده می شود .

2) ثباتهای سگمنت (Segment) :

از این ثباتها معمولا به منظور آدرس دهی نواحی حافظه استفاده می شود .

- **ثبات CS**

آدرس شروع سگمنت کد را در خود دارد . این آدرس به علاوه مقدار آفست در اشاره گر دستورالعمل (IP/EIP)، مشخص کننده آدرس دستورالعملی است که جهت اجرا از حافظه واکشی می شود .

- **ثبات DS**

دارای آدرس شروع سگمنت داده هاست یعنی این آدرس به علاوه یک آفست در یک دستور العمل ، سبب ارجاع به مکان مشخصی از سگمنت داده ها است .

- **ثبات SS**

این ثبات آدرس شروع پشته یا همان Stack را در خود دارد.

- **ثبات ES**

در برخی از عملیات های رشته ای از این ثبات برای دستکاری آدرس دهی حافظه استفاده می شود .

- **ثبات GS , FS**

ثبات های سگمنت اضافی هستند .

3) ثبات های شاخص

- **ثبات SI / ESI**

بعنوان شاخص مبدا شناخته میشود که در بعضی عملیات رشته ای لازم است .

- **ثبات DI / EDI**

بعنوان شاخص مقصد شناخته می شود .

4) ثبات های اشاره گر :

- **ثبات SP / ESP (Stack Pointer) :**

این ثبات همیشه به آخرین عنصری که وارد پشته شده است اشاره میکند که با عمل Push , Pop مقدار آن کم و زیاد می شود .

- **ثبات BP / EBP (Base Pointer) :**

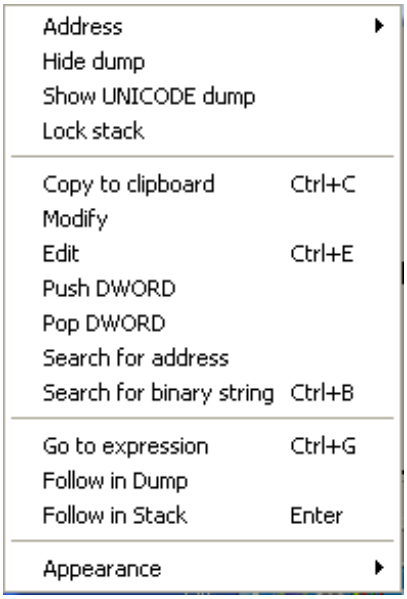
این ثبات همیشه دارای آدرس شروع پشته میباشد که معمولا در عملیات روی پشته استفاده می گردد .

5) ثبات فلگ (Flag Register) :

یک ثبات شانزده بیتی است که فقط دوازده بیت آن مورد استفاده برنامه نویس میباشد .هر کدام از این بیت ها دارای نام خاصی بوده و نشانگر وضعیت خاصی از برنامه است . مانند : ZF, SF, OF, CF, PF, AF و . . .

4– Stack :

می توان گفت Stack یا پشته تکه ای از Ram است که بوسیله دو عمل Push , Pop اطلاعات ذخیره و بازیابی می شود. دستور Push اطلاعات را ذخیره و دستور Pop اطلاعات را بازیابی می کند . همیشه داده ها به اول پشته اضافه می گردد بنابراین آخرین داده وارد شده اولین داده ای است که خارج می شود . مانند گذاشتن بشقاب ها بر روی هم ، اولین بشقابی که گذاشته می شود آخرین بشقابی است که برداشته می شود. در پشته معمولا آدرس های بازگشت در فراخوانی ها (Call) و... ذخیره می گردد که می توان براحتی در آنها به جستجو پرداخت.



چون آموزش کامل و ریزگزینه ها در این مقاله نمی گنجد فقط چند گزینه را توضیح می دهم و ممکن است در حالات مختلف دارای چندگزینه کمتر و یا بیشتر باشد .

Modify: برای تغییر دادن مقدارStack استفاده می شود.

Push Dword: اضافه کردن مقدارهای جدیداستفاده می شود.

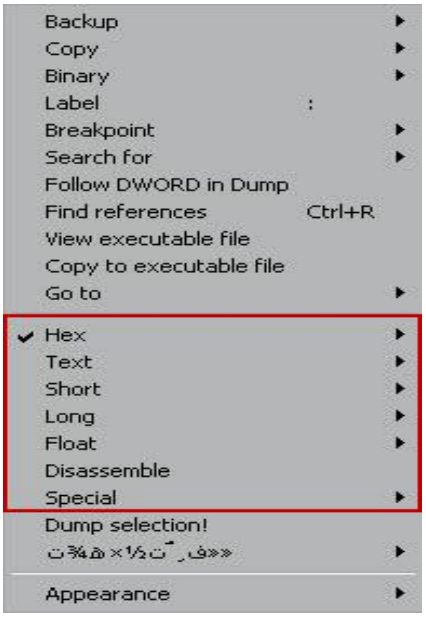
Pop Dword: خارج کردن عناصر داخل پشته استفاده می شود.

Edit: به ما حالت ویرایش بصورت 3 گانه را میدهد (Unicode, ASCII and Hex).

گزینه Lock Stack صفحه ما را ثابت می کند . بدون هیچ حرکتی یعنی اسکرول شدن صفحه را از بین می برد.

5– Dump:

در این قسمت ما می توانیم داده های خام قسمت های مختلف فایل و یا حافظه را بر اساس فرمت های استاندارد (byte , hex , text and etc) نگاه کنیم . با استفاده از منوی صفحه (کلیک راست) می توانید فرمت داده ها را بر حسب نیاز تغییر دهید(کادر قرمز رنگ) .



» کنترل روند اجرایی در 0lly Ice

» اجرای مرحله به مرحله :

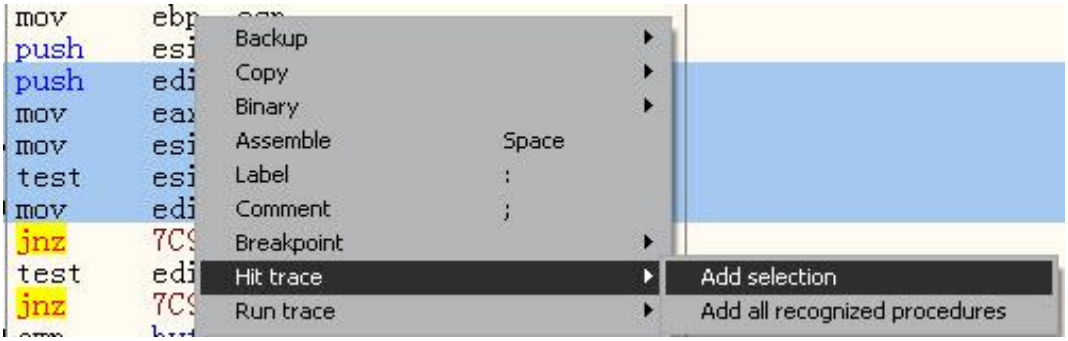
یکی دیگر ازامکانات این برنامه این است که ما می توانیم برنامه Disassemble شده خود را خط به خط اجرا کنیم و تغییرات آن در ثبات ها ، stack، و . . . . مشاهده کنیم . در این نوع اجرا کاربر کنترل برنامه را در دست می گیرد (شبیه زمانی که ، ما برنامه ای را نوشته ایم و میخواهیم باگ های آن را برطرف کنیم و از کلید های مخصوص آن برنامه استفاده می کنیم) که ما می توانیم هم از کلیدهای میانبر یا از منو Debug و یا از دکمه های موجود در تولبار(شکل ذیل) آن استفاده کنیم. در ذیل با حالت های اجرا در این روش آشنا می شویم :

- ✓ **Step Into :** با هر بار فشار دادن این دکمه یک خط از برنامه اجرا می شود و اگر ما به یک فراخوانی (Call) برخورد کنیم با این کلید میتوان وارد رویه فراخوانی شده بشویم و کدهای آن را دید یعنی اشاره گر (ادامه روند اجرا) به تابع فراخوانی شده منتقل می گردد. کلید میانبر این فرمان F7 میباشد و شکل آن در تولبار است .
- ✓ **Step Over:** با هر بار فشار دادن این دکمه یک خط از برنامه اجرا می شود و اگر ما به یک فراخوانی (Call) برخورد کنیم برعکس Step Into عمل می کند و آن را مانند یک دستور عادی انجام میدهد و فقط نتیجه این فراخوانی را مشاهده خواهیم کرد. کلید میانبر F8 و شکل آن در تولبار است .
- ✓ **Animate Into and over:** این روش مانند فشردن پی در پی دو کلید قبلی است که این عمل توسط برنامه شبیه سازی میشود. برای Animate Into از کلیدهای میانبر Ctrl + F7 و برای Animate Over از کلید میانبر Ctrl + F8 استفاده می کینم . درصورتی که بخواهید این عمل انجام نشود و قطع گردد از کلید Esc استفاده کنید .
- ✓ **Run:** برنامه بطور عادی اجرا می شود و ما از نحوه اجرای مرحله به مرحله بی بهره ایم ، دیگر خبری نداریم که کدام دستور Push می کند و یا EAX را Xor می کند و . . . کلید میانبر این گزینه F9 است و شکل آن می باشد.
- ✓ تنها راه توقف این روش استفاده از BP(Break Point) و یا استفاده از گزینه Pause که کلید میانبر آن F12 و شکل آن است .
- ✓ **Execute till return:** وقتی برنامه در حالت توقف (Pause) قرار داشته باشد با زدن این گزینه برنامه تا رسیدن به اولین دستور Retn (Return) به طور عادی اجرا می گردد در صورت رسیدن به دستور مورد نظر برنامه متوقف می شود. کلید میانبر Ctrl + F9 و شکل آن می باشد .
- ✓ **Execute till user code:** در بعضی مواقع که ما برنامه مان را متوقف می کینم ، می بینیم در درون یک فایل Dll و سیستمی ویندوز هستیم ، ما می توانیم تا رسیدن به کدهای خودمان (پروسه اصلی) از این گزینه استفاده کنیم. تمام فایل های درون پوشه System ویندوز بعنوان فایل های سیستمی تلقی می شود . کلید میا نبر Alt + F9 است .

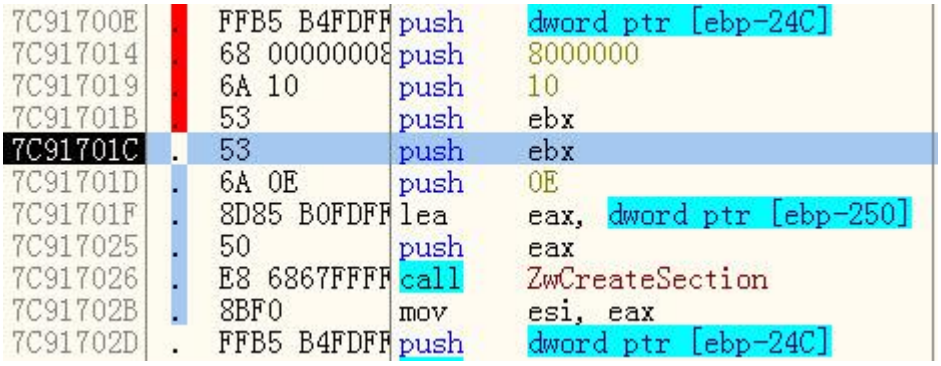
» کنترل دستورهای اجرا شده (Hit Trace) :

در بعضی مواقع دانستن اینکه کدام قسمت ها از برنامه اجرا شده است یا نه ممکن است به ما کمک زیادی در شناسایی کار توابع کند. برای این کار می توانیم بر روی جمعی از دستورات و یا فقط بر روی یک دستور این کار را انجام دهیم . برای این عمل از کلیک راست ،گزینه Hit Trace را انتخاب میکنیم .

طرز کار این گزینه بدین صورت است که برنامه (Olly) یک نقطه توقف از نوع معمولی در آن محدوده ایجاد می کند و بعد از فعال شدن آن را پاک کرده و در قسمت Hex Dump متناظر آن دستور را با رنگ قرمز علامت گذاری می کند . البته این نقاط توقف در محدوده Data باعث خراب شدن آن می شود .

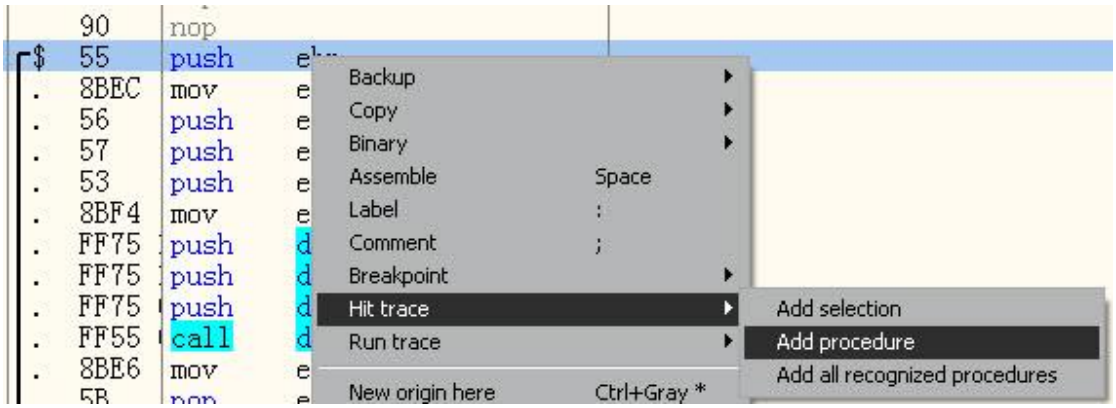


با انتخاب گزینه Add Selection ، عمل تریس بر روی محدوده انتخاب شده اعمال می شود.



Add all recognized procedures : بر روی تمامی محدوده توابع های شناسایی شده عمل Hit Trace انجام می شود.

اگر این عمل را برای یک تابع انجام دهیم ، Hit Trace دارای 3 ذیل منو میشود . گزینه اضافه شده Add Procedure می باشد که کل محدوده آن تابع را به محدوده های عمل ردیابی اضافه می کند.

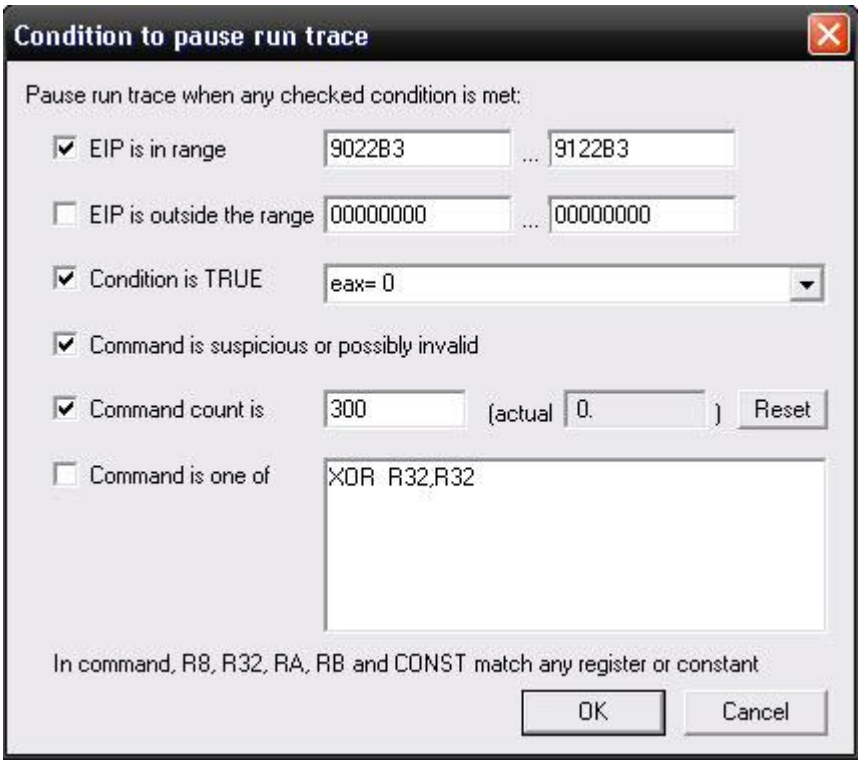


» ردیابی مراحل اجرای برنامه (Run Trace) :

دانستن اینکه یک دستور بعد از اجرا چه تغییراتی را در ثبات ها و حافظه برنامه ایجاد می کند کمک خیلی زیادی در بررسی نحوه عملکرد یک تابع می کند . این روش را Run Trace گویند . عمل کرد این روش هم مانند Hit Trace است ولی در این روش بعد از اجرا هر دستور آدرس فعلی ، وضعیت ثباتها ، ترید و ... را ذخیره میکند که بعدا میتوان آنها را مشاهده کرد. برای مشاهده گزارش این عمل از دکمه در تولبار استفاده نمایید. میتوان برای این عمل شرط هم گذاشت ، که بر اساس آن شرط ها عمل نماید برای این کار به 3 روش می توان عمل نمود :

1. منوی Debug => Set Condition
2. Ctrl + t
3. از کلیک راست در قسمت Disassembler مانند شکل ذیل :

تا پنجره Condition to pause run trace مانند شکل ذیل باز شود :



EIP is in range: توقف در صورتی که دستور العمل فعلی در بازه مشخص شده باشد.

EIP is outside the range : برعکس قبلی ، توقف در صورتی است که دستور العمل فعلی در بازه مشخص شده نباشد.

Condition is TRUE : توقف در صورتی که شرط نوشته شده درست باشد

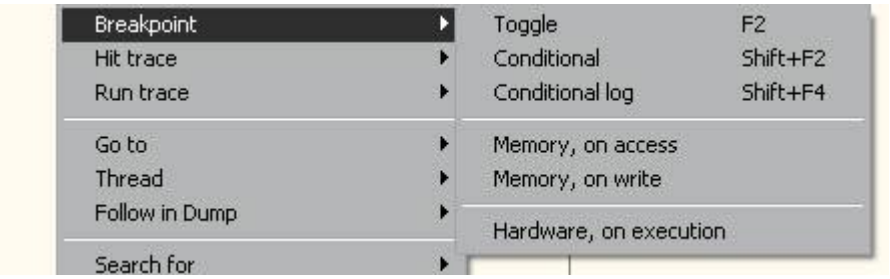
Command count is: توقف در صورتی که تعداد معین شده از از دستورات برنامه اجرا شوند.

Command is one of: توقف در صورتی که دستور بعدی با الگوی مشخص شده تطبیق داشته باشد.

بعد از تعیین شرط می توان با استفاده از کلید های Ctrl + F11 (Trace Into) برای دنبال کـردن و ردیـابی دسـتورات اجـرا شـده توسط دستور Call و یا Ctrl + F12 (Trace Over) اجرای آزاد و عادی دستورها می باشد .

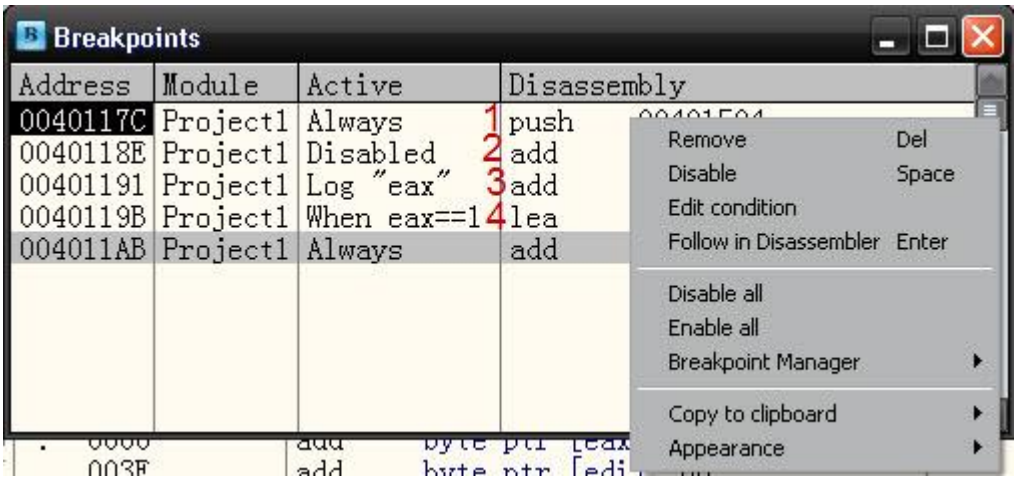
» انواع نقاط توقف (BP[Break Point]) :

نقاط توقف در Olly دو گونه اند : یک سخت افزاری و دومی نرم افزاری . نقاط توقف نرم افزاری دارای نوع های مختلفی است اعم از : نقاط توقف معمولی ، شرطی و ..... که ما در اینجا به توضیح هرکدام می پردازیم . دانستن کاربرد هر نوع نقاط توقف و استفاده درست و بجای آن میتواند به ما خیلی کمک کند و برعکس . مثلا اگر شما به جای نقاط توقف معمولی از نقاط توقف سخت افزاری استفاده کنید و یا ... معلوم نیست که سر از کجا در بیارید !!!! همانطور در تصویر ذیل انواع BP را می بینید ، که برای استفاده از آنها می توان از کلیک راست در قسمت های مربوطه استفاده کرد البته تعدادی از این نقاط توقف دارای کلید میانبر هستند. بعنوان مثال Toggle BP که هم آن نقاط توقف معمولی هستند با کلید F2 فعال می شوند .



» نقاط توقف معمولی (Toggle Break Point) :

پرکاربردترین و ساده ترین نوع هست. در هنگام اجرای آزاد برنامه به محض رسیدن به این نوع نقاط کنترل را به عهده کاربر قرار میدهد . این نقاط توقف با درج دستور وقفه دیباگ (INT3) در دستور ها ایجاد میشود .این نقاط توقف هیچ محدودیتی ندارد در صورتی که نقاط توقف سخت افزاری دارای محدودیت میباشد . این دیباگر نقاط توقف ایجاد شده را در هنگام مراحل دیباگ در فایلی ذخیره می کند این فایل برای هر برنامه و یا فایلی که ما می خواهیم دیباگ کنیم جداگانه می سازد . این فایل باپسوند UDD بطور پیش فرض در کنار خود برنامه و در پوشه UDD می باشد که میتوان این آدرس پیش فرض را هم تغییر داد . (بعضی از Backup ها هم در آنجا ساخته میشود) تا ما وقتی دوباره آن برنامه را خواستیم دیباگ کنیم با همان کارهای قبلی که روی آن انجام دادیم آورده شود . برای ایجاد این نقاط همانطور که قبلا گفتم با فشردن کلید F2 و یا دابل کلیک در قسمت Hex Dump انجام دهیم .برای غیر فعال کردن آن هم دوباره کارهای قبل را میتوان انجام داد . برای دسته بندی و کنترل بر روی BP ، برنامه آنها را درون پنجره Breakpoints قرار میدهد . برای رفتن به این پنجره میتوان از کلید میانبر Alt + B و یا از B در تولبار استفاده کنیم و یا اینکه از منوی View استفاده کنیم .



همانطور که در بالا می بینید یک قسمت Active دارد که در این قسمت 4 حالت ممکن است وجود داشته باشد :

1- BP ما در حالت فعال می‌باشد.

2- BPدر حالت غیر فعال میباشد ، یعنی دیباگر این نقطات را در نظر نمی‌گیرد.

3- این حالت برای نقاط توقف شرطی همراه گزارش میباشد .

4- این حالت هم برای نقاط توقف شرطی (بدون گزارش) می باشد .

در تصویر بالا هم می بینید که دارای منوی کلیک راست مخصوص خود میباشد که این گزینه ها برای حذف و غیر فعال کردن و یا عوض کردن شرط های تعیین شده و . . . . . می باشد .

» نقاط توقف شرطی (Conditional BP) :

این نقاط براساس شرط تعیین شده کاربر می باشد ، که در صورت برقراری شرط BP عمل میکند در غیر این صورت BP را نادیده می گیرد . این نوع سرعت عملیات را پایین آورده و معمولا در پروسیجرهای پنجره استفاده می شود .برای استفاده از این نوع نقطه توقف می توان از کلیدهای Shift + F2 کمک گرفت . تعیین شرط ها در Olly دارای گرامر خاص خود می باشد که میتوان با آن عبارات و شرط های پیچیده ای هم نوشت . در جدول ذیل هم انواع شرط ها را می بینیم که همراه با توضیح میباشند . این جدول برای تمام شرط نویسی و یا اسکریپت نویسی و .... صدق می کند .

Sample	Description
10	constant 0x10 (unsigned). All integer constants are assumed hexadecimal unless followed by a decimal point
10.	decimal constant 10 (signed);
'A'	character constant 0x41;
EAX	contents of register EAX, interpreted as unsigned number;

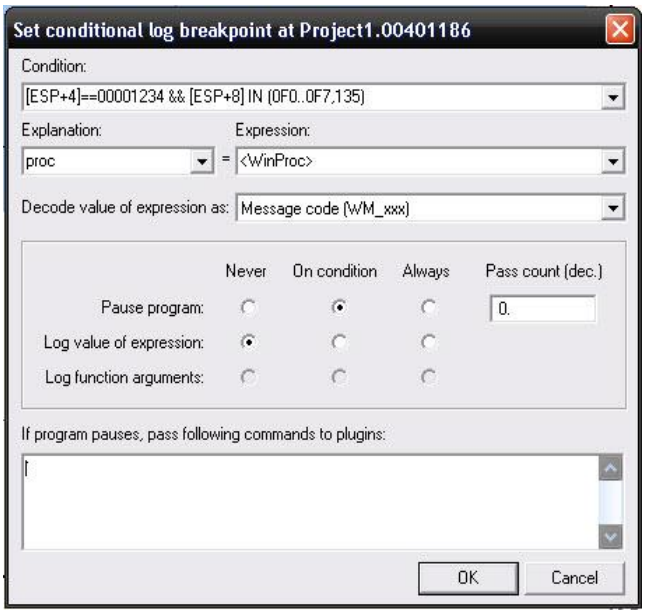
EAX.	contents of register EAX, interpreted as signed number;
[123456]	contents of unsigned doubleword at address 123456. By default, OllyDbg assumes doubleword operands;
DWORD PTR [123456]	same as above. Keyword PTR is optional;
[SIGNED BYTE 123456]	contents of signed byte at address 123456. OllyDbg allows both MASM- and IDEAL-like memory expressions;
STRING [123456]	ASCII zero-terminated string that begins at address 123456. Square brackets are necessary because you display the contents of memory;
[[123456]]	doubleword at address that is stored in doubleword at address 123456;
2+3*4	evaluates to 14. OllyDbg assigns standard C priorities to arithmetical operations;
(2+3)*4	evaluates to 20. Use parentheses to change the order of operations;
EAX.<0.	0if EAX is in range 0..0x7FFFFFFF and 1 otherwise. Notice that constant 0 is also signed. When comparing signed with unsigned, OllyDbg always converts signed operand to unsigned.
EAX<0	always 0 (false), because unsigned numbers are always positive.
MSG==111	true if message is WM_COMMAND. 0x0111 is the code for WM_COMMAND. Use of MSG makes sense only within conditional or conditional logging breakpoint set on call to or entry of known function that processes messages.
[STRING 123456]=="Brown fox"	true if memory starting from address 0x00123456 contains ASCII string "Brown fox", "BROWN FOX JUMPS", "brown fox???" or similar. The comparison is case-insensitive and limited in length to the length of text constant.
EAX=="Brown fox"	same as above, EAX is treated as a pointer.
UNICODE [EAX]=="Brown fox"	OllyDbg treats EAX as a pointer to UNICODE string, converts it to ASCII and compares with text constant.
[ESP+8]==WM_PAINT	in expressions, you can use hundreds of symbolic constants from Windows API.
([BYTE ESI+DWORD DS:[450000+15*(EAX-1)]] & 0F0)!=0	absolutly valid expression.

جدول بالا گرامری بود که برای تعیین شرط ها لازم است و برای اسکریپت نویسی هم صدق می کند . هر عنصر را داخل ({}) فقط یک بار میتواند واقع قرار داده شود و ترتیب آنها مهم نیست . در ذیل هم جدول مخصوص عناصر را می بینید . همانطور هم که قبلا در نکات درون اسکریپ نویسی برای Olly گفتم رشته ها باید درون "" قرار گیرند و این حرف من را هم در جدول ذیل می بینید .

Operations	Type
expression	memterm | memterm <binary operation> memterm
memterm	term | { sigmod sizemod prefix [} expression]
term	(expression) | unaryoperation memterm | signedregister | register | fpuregister | segmentregister | integerconst | floatingconst | stringconst | parameter | pseudovariable
unaryoperation	- | + | ~ | !
signedregister	register.
register	AL | BL | CL ... | AX | BX | CX ... | EAX | EBX | ECX...
fpuregister	ST | ST0 | ST1...
segmentregister	CS | DS | ES | SS | FS | GS
integerconst	<decimal constant>. | <hexadecimal constant> | <character constant> | <symbolic API constant>
floatingconst	<floating constant>
stringconst	"<string constant>"
sigmod	SIGNED | UNSIGNED
sizemod	BYTE | CHAR | WORD | SHORT | DWORD | LONG | QWORD | FLOAT | DOUBLE | FLOAT10 | STRING | UNICODE
prefix	term:
parameter	%A | %B //Allowed in inspectors only
pseudovariable	MSG // Code of window message

» نقاط توقف شرطی با گزارش (Conditional Log BP) :

این نقاط همانند نقاط بدون گزارش است ولی در این نوع عملیات گزارش گیری به صورت های مختلف و با فرمت دلخواه که توسط کاربر قابل تعیین می باشد . کلید میانبر این عمل Shift + F4 میباشد . خروجی این گزارش ها میتوانند در فایل ذخیره شوند و یا در پنجره Log به نمایش در آیند . برای مثال شما می توانید این نوع BP برای رویه پنجره ها و تمام فراخوانی(Call) موجود در آن رویه و یا بر روی Message رسیده WM_COMMAND و یا Message های دیگری تنظیم کنید .



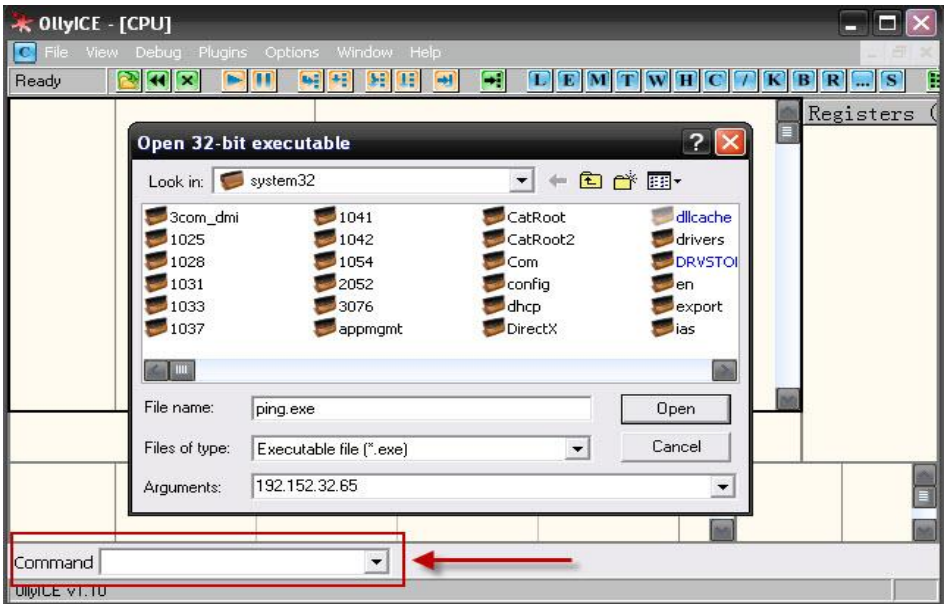
این پنجره دارای قسمت های مختلفی است که بطور خیلی مختصر توضیح می دهم :

- ✓ Condition: در این قسمت باشد شرط مورد نظر را بنویسید. البته با توجه به جدول گرامری که قبلا گذاشتم .
- ✓ Explanation: میتوان توضیح خیلی کوتاهی برای شرط مان بنویسم که بعدا به درد می خورد.
- ✓ Expression: اطلاعاتی را که میخواهیم در گزارش نگاه کنیم.
- ✓ Decode Value Of Expression: فرمت نشان دادن گزارش ها را انتخاب می کنیم.
- ✓ Pause Program: با انتخاب گزینه on condition در هنگام درست بودن شرط برنامه متوقف می شود .می توانیم تعداد دفعاتی که نقطه توقف ما را در نظر نگیرد هم مشخص کنیم این عدد در کادر روبرو نوشته می شود و بر اساس دسیمال است.

Log value of expression: در این قسمت شرایط گزارش گیری مشخص میشود که همانند قبلی است .

Log function arguments: اگر تابع و پارامترهای آن مشخص شده باشند، می تواند گزارشی از تمامی پارامترهای ارسالی به به تابع مورد نظر تهیه کند.

در انتهای پنجره کادری وجود دارد ، اگر برنامه ما بایستد یعنی شرط ما کار کند ، دستورات تایپ شده ما در این قسمت به command bar فرستاده می شود .CommandBar یک پلاگین است که دستوراتی را برای ما انجام می دهد. این امکان در پایین پنجره اصلی قرار دارد .البته این پلاگین مانند خود Olly دارای Help میباشد که همانند Olly باید آن را جداگانه دریافت کنید و برای پلاگین مسیر آن را مشخص کنید . در ذیل هم می توانید دستورات نسبتا کامل و همراه با توضیحات و مثال است را نگاه کنید .



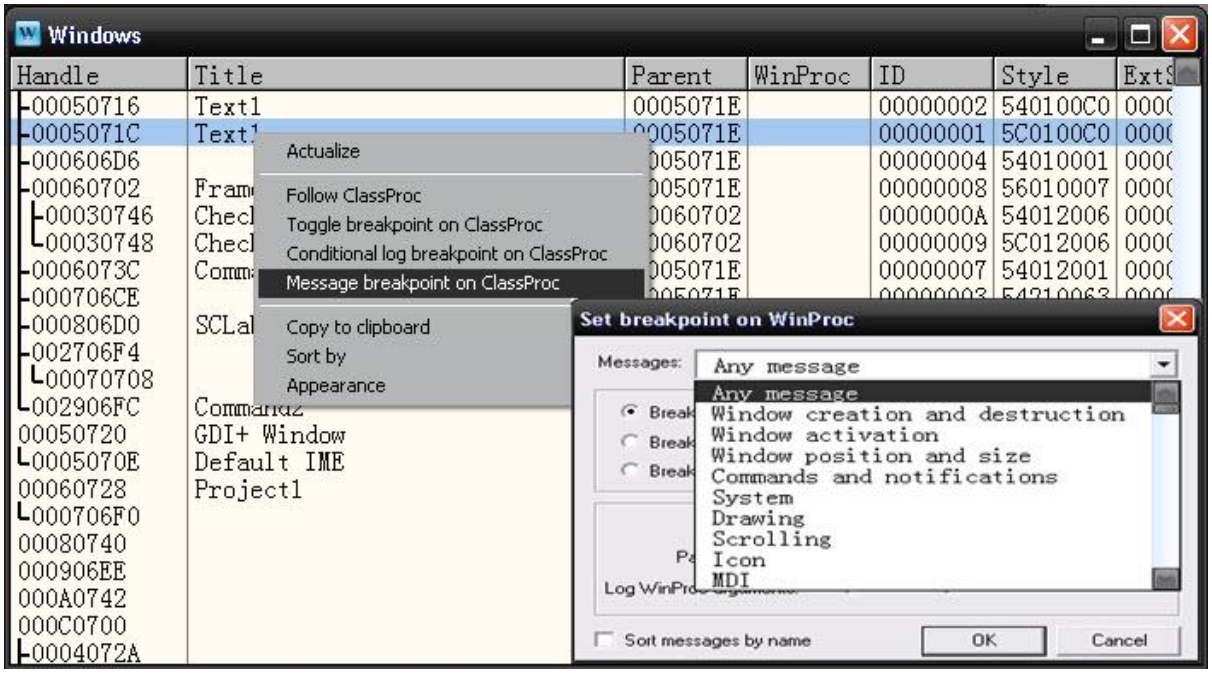
Command	Description	Example
CALC expression	Calculates value of expression	CALC EAX/2+1
? expression	Ditto	
expression(first character is not a letter)	Ditto	2*2
WATCH expression	Add watch	WATCH +[460030+ESI]
W expression	Ditto	
SET reg=expression	Writes value of expression to 8-, 16- or 32-bit general register	SET AL=0 SET ESI=[DWORD EDI-10]
reg=expression	Ditto	
SET memory=expression	Writes value of expression to 8-, 16- or 32-bit memory	SET [410000]=80000001SET [BYTE EAX+ESI*2]=0
AT expression	Follow address in Disassembler	AT 410000
FOLLOW expression	Ditto	FOLLOW EAX
ORIG	Go to actual EIP	
*	Ditto	
D expression	Follow address in dump	D 460000
DUMP expression	Ditto	
DA [expression]	Dump in assembler format	
DB [expression]	Dump in hex byte format	

DC [expression]	Dump as ASCII text	DC EAX
DD [expression]	Dump as addresses (stack format)	
DU [expression]	Dump as UNICODE text	
DW [expression]	Dump in hex word format	
STK expression	Follow address in stack	
A expression [,command]	Assemble at address	A 410000, XOR EAX,EAX
L expression, label	Assign symbolic label to address	L EAX, loopstart
C expression, comment	Set comment at address	C EAX, Here loop starts
BP expression [,condition]	Set INT3 breakpoint at address	BP EAX+10BP 410010, EAX==WM_CLOSEBP Kernel32.GetProcAddress
BPX label	Set breakpoint on each call to external 'label' within the current module	BPX CreateFileA
BC expression	Delete breakpoint at address	BC 410010
MR expression1 [,expression2]	Set memory breakpoint on access to range	
MW expression1 [,expression2]	Set memory breakpoint on write to range	
MD	Remove memory breakpoint	
HR expression	Set 1-byte hardware breakpoint on access to address	
HW expression	Set 1-byte hardware breakpoint on write to address	
HE expression	Set hardware breakpoint on execute at address	
HD [expression]	Remove hardware breakpoint(s) at address	

STOP	Pause execution	
PAUSE	Ditto	
RUN	Run program	
G [expression]	Run till address	
GE [expression]	Pass exception to handler and run till address	
S	Step into	
SI	Ditto	
SO	Step over	
T [expression]	Trace in till address	
TI [expression]	Ditto	
TO [expression]	Trace over till address	
TC condition	Trace in till condition	
TOC condition	Trace over till condition	
TR	Execute till return	
TU	Execute till user code	
LOG	View Log window	
MOD	View Executable modules	
MEM	View Memory window	
CPU	View CPU window	
CS	View Call Stack	
BRK	View Breakpoints window	
OPT	Edit options	
EXIT	Close OllyDbg	
QUIT	Ditto	
OPEN [filename]	Open executable file for debugging	
CLOSE	Close debugged program	
RST	Restart current program	
HELP	Show this help	
HELP OllyDbg	Show OllyDbg help	
HELP APIfunction	Show help on API function	
EXIT	Close OllyDbg	
QUIT	Ditto	

» نقاط توقف برای پروسیجرهای پنجره :

این نوع نقاط شرط شان توسط خود Olly ایجاد می شوند . برای این کار شما باید به پنجره Window بروید ، از تولبار و یا از منوی View استفاده کنید . برای اینکه لیست پنجره ها ایجاد گردد ، باید برنامه را کامل اجرا کنیم .بعد از منوی کلیک راست گزینه Message breakpoint on ClassProc را انتخاب کنید تا پنجره مربوطه باز شود.



بعد از انتخاب گزینه مورد نظر از کلیک راست پنجره Set break Point on WinProc باز می شود . این پنجره دارای یک Combo Box مربوط به Message است و در آن می توانیم پیغام ها را یا به صورت گروهی و یا اینکه پیغامی خاصی را انتخاب کنیم که در جدول ذیل گروهها و Message های مربوط به هر گروه که می توانیم آنها را به صورت تکی بیاوریم توضیح داده شده است .

اگر به دقت در منوی ایجاد شده در تصویر بالا نگاه کنید میبینید که گزینه ای هم برای نقاط توقف شرطی همراه با گزارش دارد . این گزینه همانند نقاط توقف شرطی همراه با گزارش است و هیچ فرقی ندارند فقط در قسمت شرط باید چند نکته را رعایت کنید که آن را با یک مثال توضیح میدهم . تنظیمات پنجره را بدین صورت تنظیم کنید :

Condition: [ESP+4] ==00001234 && [ESP+8] IN (0F0..0F7, 135)

Explanation: <WinProc>

Pause program: On condition

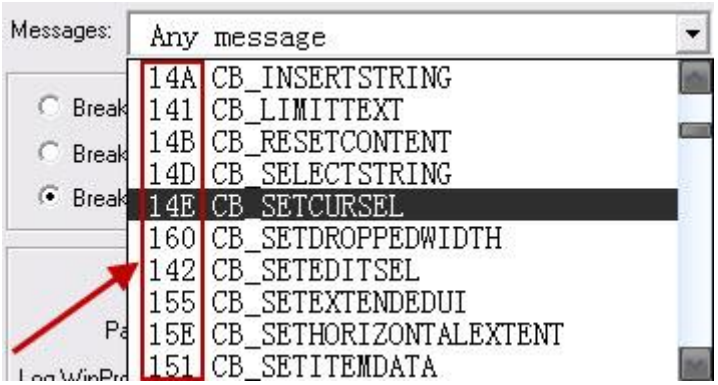
مفهوم متن های سبز رنگ به شرح ذیل است :

[ESP+00] Return address

[ESP+04] Window's handle

[ESP+08] Message
[ESP+0C] wParam
[ESP+10] lParam

متن قرمز رنگ هم مفهوم همان پیغام های ما را می رسانند که برای فهمیدن هر کدام می توانید از Set break Point on WinProc که در کنار هر پیغام مقدار مرتبط با آن نوشته شده است را ببینید . کادر قرمز رنگ را به دقت نگاه کنید :



پیغام های نوشته شده هم اینها هستند :

(BM_GETCHECK... , BM_SETIMAGE, WM_CTLCOLORBTN)

جدول ذیل لیستی از پیغام ها همراه گروه بندی مشخص است :

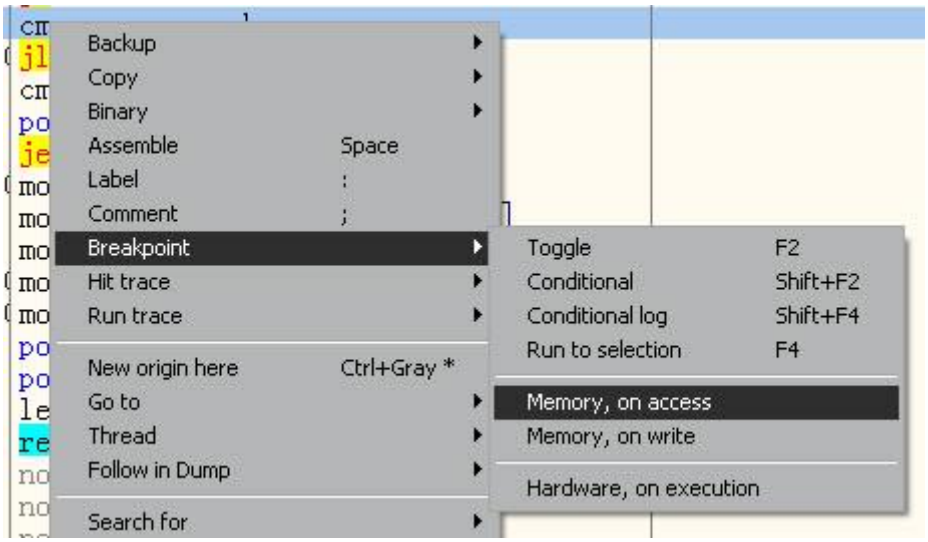
Group	Messages in group
Any message	Any message
Creation and destruction	WM_CREATE, WM_DESTROY, WM_CLOSE, WM_QUERYENDSESSION, WM_QUIT,WM_ENDSESSION, WM_NCCREATE, WM_NCDESTROY, WM_INITDIALOG
Window activation	WM_ACTIVATE, WM_SETFOCUS, WM_KILLFOCUS,WM_ENABLE, WM_SHOWWINDOW, WM_CHILDACTIVATE, WM_QUERYNEWPALETTE
Window position and size	WM_MOVE, WM_SIZE, WM_QUERYOPEN, WM_SHOWWINDOW, WM_GETMINMAXINFO, WM_WINDOWPOSCHANGING, WM_WINDOWPOSCHANGED, WM_NCCALCSIZE, WM_SIZING, WM_MOVING, WM_ENTERSIZEMOVE, WM_EXITSIZEMOVE
Commands and notifications	WM_MEASUREITEM, WM_COMMNOTIFY, WM_NOTIFY,WM_NOTIFYFORMAT, WM_STYLECHANGING, WM_STYLECHANGED, WM_COMMAND, WM_SYSCOMMAND, WM_ENTERIDLE, WM_PARENTNOTIFY, WM_MDIRESTORE

System	WM_SYSCOLORCHANGE, WM_WININICHANGE, WM_DEVMODECHANGE, WM_ACTIVATEAPP, WM_FONTCHANGE, WM_TIMECHANGE, WM_COMPACTING, WM_POWER, WM_USERCHANGED, WM_DISPLAYCHANGE, WM_NCACTIVATE, WM_POWERBROADCAST, WM_DEVICECHANGE, WM_PALETTEISCHANGING, WM_PALETTECHANGED
Drawing	WM_SETREDRAW, WM_PAINT, WM_ERASEBKGND, WM_PAINTICON, WM_ICONERASEBKGND, WM_DRAWITEM, WM_NCPAINT, WM_QUERYNEWPALETTE, WM_PRINT, WM_PRINTCLIENT
Scrolling	WM_HSCROLL, WM_VSCROLL, WM_CTLCOLORSCROLLBAR
Icon	WM_QUERYOPEN, WM_QUERYDRAGICON, WM_GETICON, WM_SETICON
MDI	WM_MDICREATE, WM_MDIDESTROY, WM_MDIACTIVATE, WM_MDIRESTORE, WM_MDINEXT, WM_MDIMAXIMIZE, WM_MDITILE, WM_MDICASCADE, WM_MDIICONARRANGE, WM_MDIGETACTIVE, WM_MDISETMENU
Dialog	WM_CANCELMODE, WM_NEXTDLGCTL, WM_MEASUREITEM, WM_DELETEITEM, WM_GETDLGCODE, WM_CTLCOLORMSGBOX, WM_CTLCOLORDLG
Menu	WM_MEASUREITEM, WM_HELP, WM_CONTEXTMENU, WM_INITMENU, WM_INITMENUPOPUP, WM_MENUSELECT, WM_MENUCHAR, WM_ENTERMENULOOP, WM_EXITMENULOOP, WM_NEXTMENUWM_MDIREFRESHMENU
Text	WM_SETTEXT, WM_GETTEXT, WM_GETTEXTLENGTH, WM_SETFONT, WM_GETFONT
Mouse	WM_SETCURSOR, WM_MOUSEACTIVATE, WM_NCHITTEST, WM_NCMOUSEMOVE, WM_NCLBUTTONDOWN, WM_NCLBUTTONUP, WM_NCLBUTTONDBLCLK, WM_NCRBUTTONDOWN, WM_NCRBUTTONUP, WM_NCRBUTTONDBLCLK, WM_NCMBUTTONDOWN, WM_NCMBUTTONUP, WM_NCMBUTTONDBLCLK, WM_MOUSEMOVE, WM_LBUTTONDOWN, WM_LBUTTONUP, WM_LBUTTONDBLCLK, WM_RBUTTONDOWN,

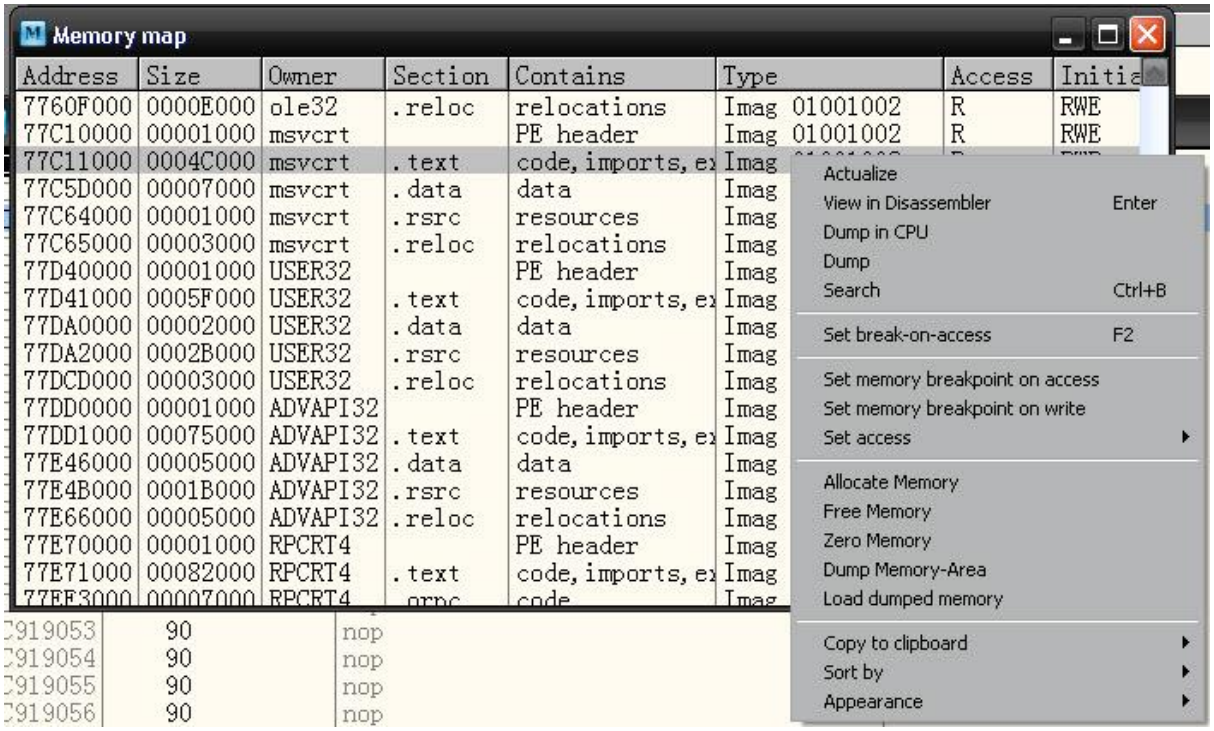
	WM_RBUTTONUP, WM_RBUTTONDBLCLK, WM_MBUTTONDOWN, WM_MBUTTONUP, WM_MBUTTONDBLCLK, WM_MOUSEWHEEL, WM_XBUTTONDOWN, WM_XBUTTONUP, WM_XBUTTONDBLCLK, WM_CAPTURECHANGED
Keyboard	WM_VKEYTOITEM, WM_CHARTOITEM, WM_SETHOTKEY, WM_GETHOTKEY, WM_KEYDOWN, WM_KEYUP, WM_CHAR, WM_DEADCHAR, WM_SYSKEYDOWN, WM_SYSKEYUP, WM_SYSCHAR, WM_SYSDEADCHAR, WM_HOTKEY
Clipboard	WM_CUT, WM_COPY, WM_PASTE, WM_CLEAR, WM_UNDO, WM_RENDERFORMAT, WM_RENDERALLFORMATS, WM_DESTROYCLIPBOARD, WM_DRAWCLIPBOARD, WM_PAINTCLIPBOARD, WM_VSCROLLCLIPBOARD, WM_SIZECLIPBOARD, WM_ASKCBFORMATNAME, WM_CHANGECBCHAIN, WM_HSCROLLCLIPBOARD
Edit control	All EM_xxx messages
Static control	All STM_xxx messages
Button	All BM_xxx messages, WM_CTLCOLORBTN
Combo box	All CB_xxx messages, WM_COMPAREITEM
List box	All LB_xxx messages, WM_COMPAREITEM, WM_CTLCOLORLISTBOX
IME	All WM_IME_xxx messages
User-defined	All messages equal or above WM_USER

» نقاط توقف برای دسترسی به حافظه :

از این نقاط توقف معمولا برای ردیابی عملیات خواندن و نوشتن در یک محدوده از حافظه استفاده می شود . دیباگر Olly اجـازه ایجـاد یک نمونه از این نقاط را به کل برنامه میدهد. بریا ایجاد این نوع ، آدرس را انتخاب و از منوی صفحه (کلیک راست) گزینه Memory, on access و یا Memory, on write را انتخاب کنید.



نوع دیگر این نقاط می توانند از سکشن های فایل اجرایی و یا Dllهای بارگذاری شده به حافظه برنامه پشتیبانی کند . برای این عمل می بایست به پنجره Memory Mapبروید .کلید میانبر Alt + M و یا M را از تولبارانتخاب کنید .



در تصویر بالا می بینید که لیستی از بلوک های موجود در حافظه برنامه همراه با مشخصات هریک به نمایش گذاشته است. بعد از انتخاب بلوک مورد نظر میتوان از گزینه های Set Memory Breakpoint on Access و set memory breakpoint on write استفاده کنید.

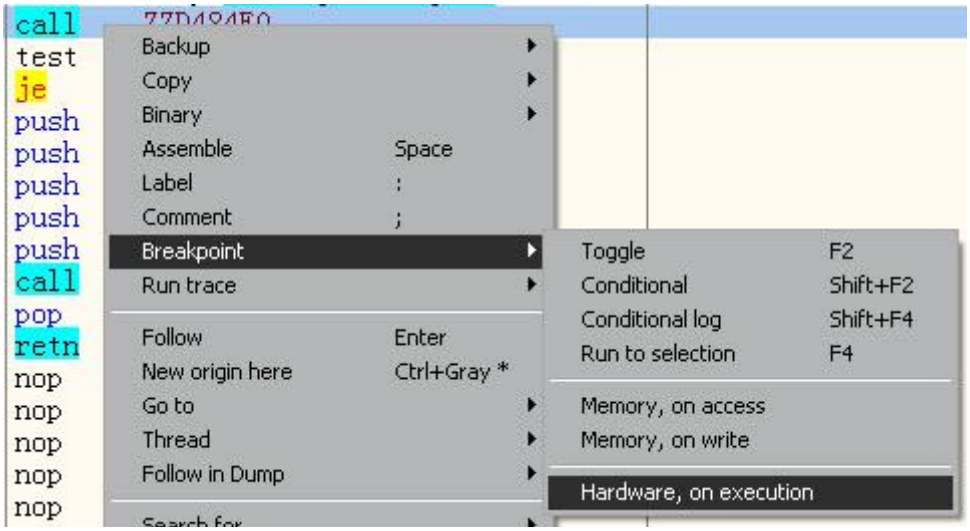
در تصویر بالا گزینه Set break-on-access را می بینید که کلید میانبر آن F2 می باشد . این نقاط به نقاط یکبار مصرف معروفند . یعنی این نقاط پس از فعال شدن بطور خودکار از بین میروند و محدودیتی در استفاده ندارند .معمولا از این نقاط برای فراخوانی ها و یا بازگشت انجام شده به Dll های مورد استفاده برنامه ، استفاده می شوند .

» نقاط توقف سخت افزاری (Hardware BP) :

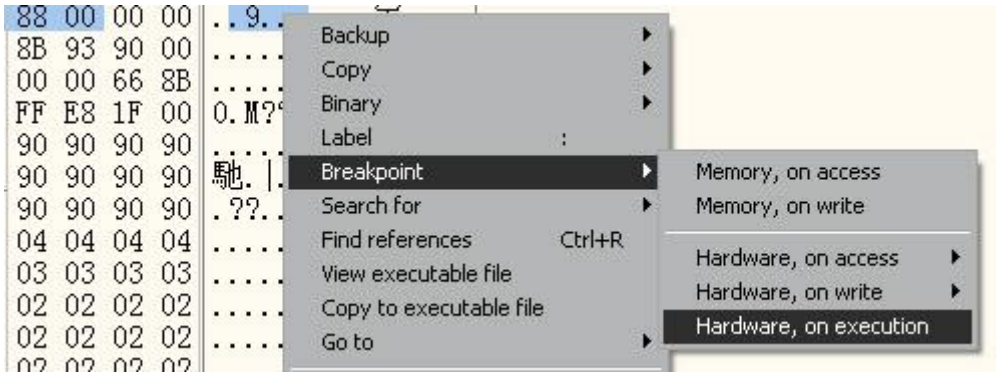
نقاط توقف سخت افزاری از ثبات های خاصی برای ذخیره آدرس خطی استفاده می کند. در این نوع نقاط توقف دارای محدودیت هستیم. اگر شرط مدنظر گرفته شده برای نقاط توقف مقدار درستی بگیرد از طریق وقفه شماره یک کنترل پروسه به دیباگر منتقل خواهد شد که در موارد ذیل این کار انجام می شود:

- اجرا شدن یک دستور
- مقداری در حافظه اصلاح گردد
- فضایی از حافظه خوانده شود و یا به روز گردد
- پورت های ورودی-خروجی اصلی

یکی از فایده های نقاط توقف سخت افزاری این است که توسط نرم افزارها تقریبا قابل تشخیص نمی باشد. اما کار نشد ندارد ☺ بحث بر روی این قضیه مربوط به روشهای ضد مهندسی معکوس می باشد که در مقالات بعدی توضیح داده خواهد شد. معمولا این نقاط برای ردیابی خواندن ، نوشتن و یا اجرای محدوده خاصی از حافظه مورد استفاده برنامه ایجاد می گردند و به دسته های یک بایتی(Byte) ، دو بایتی (Word) و چهار بایتی (DWord) تقسیم میگردند. برای ایجاد این نقاط میتوان در قسمت Disassemble از منوی صفحه گزینه Hardware , on execution را انتخاب کرد .



و برای دسترسی به حافظه و محدوده Data ، از قسمت Dump آدرس مورد نظر را انتخاب و از نقطه توقف مورد نیاز استفاده کنیم



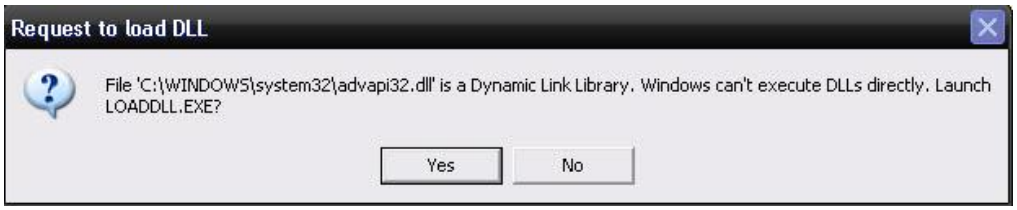
On Access: با هرگونه دسترسی به محدوده مورد نظر فعال می شود.

On Write: موقع نوشتن در محدوده مورد نظر فعال می شود.

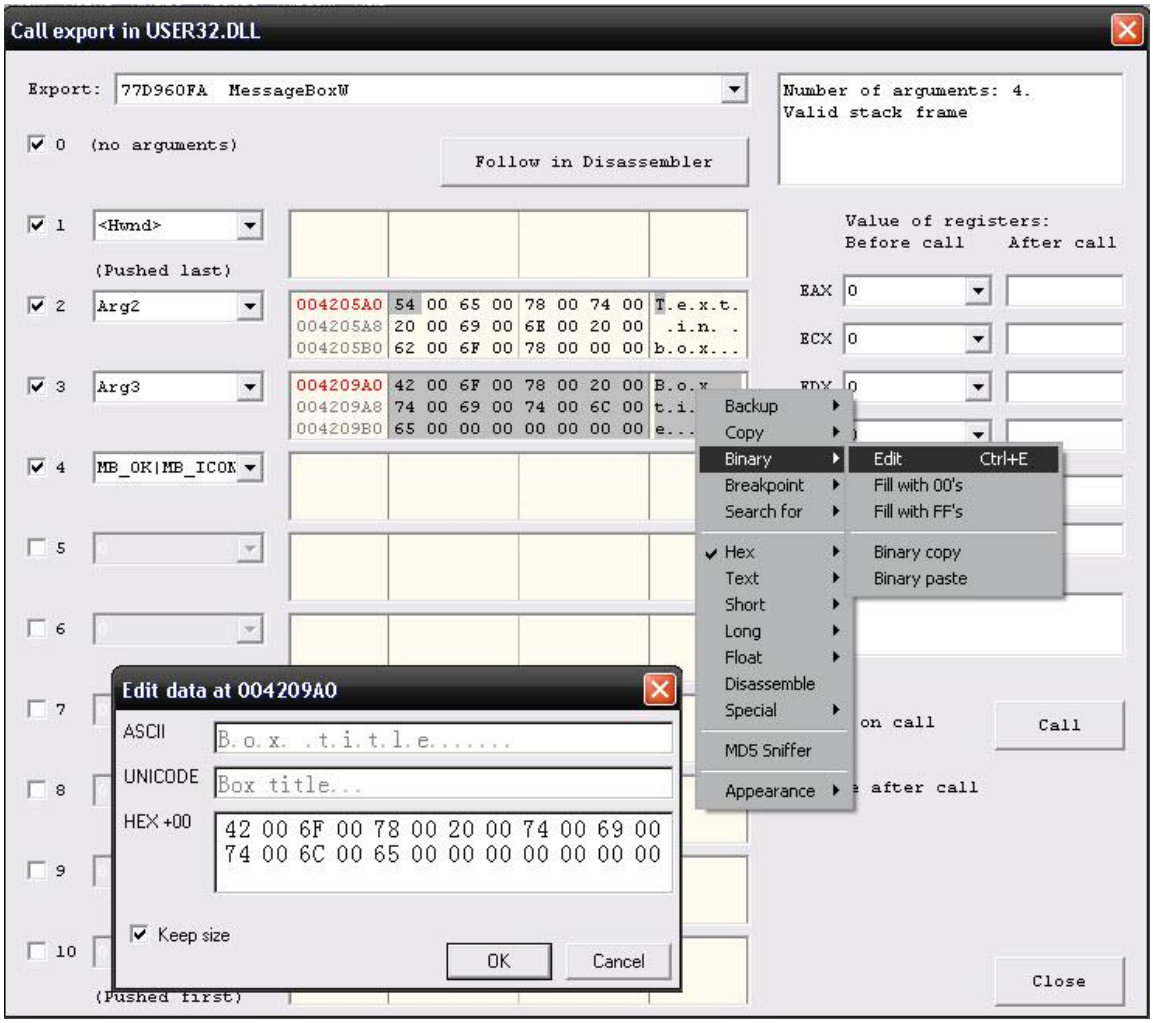
On Execution: در موقع اجرای دستور العمل در محدوده مورد نظر فعال می شود.

» فراخوانی توابع موجود در یک Dll :

دیباگر OllyIce به ما این امکان را میدهد که بتوانیم توابع درون یک فایل Dll را فراخوانی کنیم و به نحوی آن را به تنهایی Debug کنیم. این نوع فایل ها توسط loaddll.exe در برنامه بارگذاری میشود . درصورتی که برای بار اول باشد آن فایل را باز میکنید با پیغامی مواجه میشوید که میگوید فایل مورد نظر نمیتوانند مانند فایل های اجرایی بارگذاری شود آیا با loaddll.exe اجرا شود ؟ که شما باید Ok را بزنید .



در این مثال ما میخواهیم تابع MessageBoxW را از فایل user32.dll فراخوانی کنیم . بعد از بارگذاری آن در برنامه باید از منوی Debug گزینه Call Dll export را بزنیم تا پنجره مربوطه باز شود .



در قسمت Export باید تابع مورد نظرمان را انتخاب کنیم .

دکمه Follow In Disassembler ما را به محدوده تابع مورد نظر در پنجره Disassemble می برد .

پارامتر اول مربوط به هندل پنجره ایت که براحتی میتوانید <HWND> را انتخاب کنید .

پارامتر دوم(Arg2) که مربوط متن پیغام است که میتوانید با Edit متن مورد نظر را در آن بنویسید . ترجیحا در قسمت Unicode بنویسید .

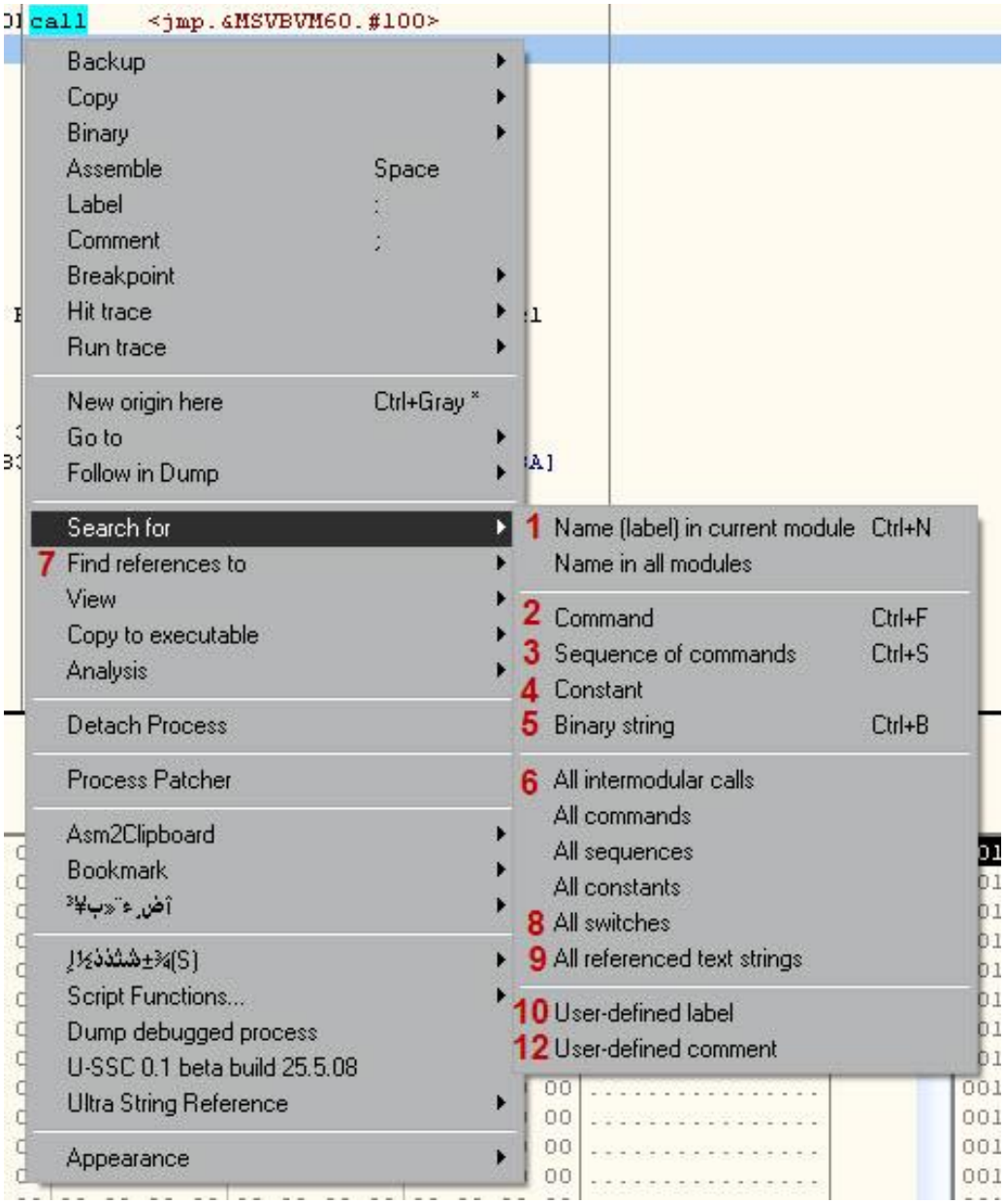
پارامتر سوم (Arg3) مربوط به متن عنوان پیغام است .

پارامتر چهارم مربوط به دکمه ها و آیکون مورد نظر است . که بدین صورت است MB_xxx و ما در این پارامتر مقدار MB_OK|MB_ICONEXCLAMATION را قرار دادیم .

این پنجره به ما 10 آرگومان را می دهد . برای ویرایش هر بخش باید مقدار بایت مورد نیاز را انتخاب و مانند شکل از راست کلیک Binary → Edit را بزنیم تا پنجره مربوط به آن باز شود . بعد بر روی Call کلیک کنید تا پیغام دیده شود .

» جستجو در 0llyIce :

یکی از ویژگی های بارز Olly گستردگی در جستجو می باشد که به ما وسعت کار بیشتری می دهد. ما می توانیم هر کدام را بنابر شرایط ، راحتی کار و بالابردن سرعت عمل انتخاب می کنیم . تمام این گزینه ها در منوی صفحه Disassemble میباشد و گزینه Search for می باشد .



در ذیل لیستی از موردهایی ذکر شده است که می توان براساس آنها جستجو را انجام داد.در ادامه به شرح بیشتر آنها می پردازیم :

1. توابع ورودی - خروجی در Module ها
2. یک دستورالعمل
3. دنباله ای از دستور العمل ها
4. ثابت ها
5. براساس رشته های دودویی (حالت سه گانه)
6. فراخوانی های خارجی انجام شده
7. ارجاع های انجام شده
8. جستجو در Switch ها(دستورات راه گزین)
9. لیست متن های موجود
10. برچسب های ایجاد شده توسط کاربر
11. توضیحات ایجاد شده توسط کاربر (همون شماره 12 در تصویر)

در اینجا نکته ای قابل ذکر است که گزینه هایی نسبتا شبیه هم هستند . طرز کار آنها هیچ فرقی با یکدیگر ندارند و فقط گزینه هایی که دارای کلمه ALL می باشند جستجو را در تمامی ماژول ها انجام می دهند و گزینه هایی که فاقد این کلمه می باشند جستجو را فقط در ماژول جاری انجام می دهند.

» توابع ورودی– خروجی (Name (label) in current module):

با این گزینه تمامی توابع Module جاری برای ما لیست می شود .این توابع دارای نوع های خاص خود می باشد :

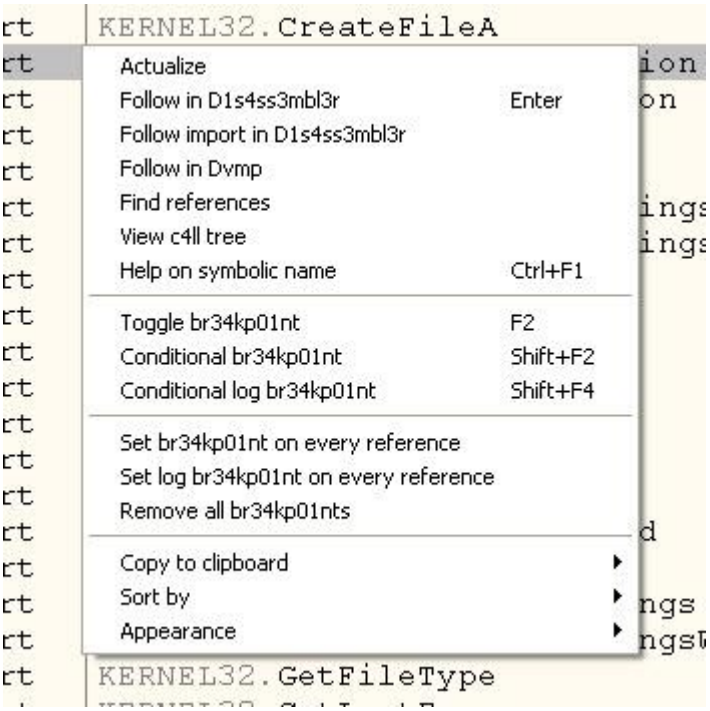
- ✓ Exports: این توابع در خود فایل وجود دارد یعنی کدهای تابع درون فایل نوشته شده است و توسط Module های دیگر قابل استفاده می باشد.
- ✓ Import: توابعی که درون فایل وجود ندارد و از یک فایل دیگر مانند DLL فراخوانی شده است.

بعنوان مثال تصویر ذیل را از فایل GDI32 می بینید که بعضی از توابع را که استفاده می کند بصورت Import است که از یک فایل دیگر فراخوانی کرده است.

Section	Type	Name
.text	Export	DdEntry8
.text	Export	DdEntry9
.text	Export	DeleteColorSpace
.text	Import	KERNEL32.DeleteCriticalSection
.text	Export	DeleteDC
.text	Export	DeleteEnhMetaFile
.text	Import	KERNEL32.DeleteFileW
.text	Export	DeleteMetaFile
.text	Export	DeleteObject
.text	Export	DescribePixelFormat
.text	Export	DeviceCapabilitiesExW
.text	Import	KERNEL32.DisableThreadLibraryCall
.text	Export	DPtoLP

خوب تا به اینجا خوب پیش رفتیم ، خسته نباشید ☺

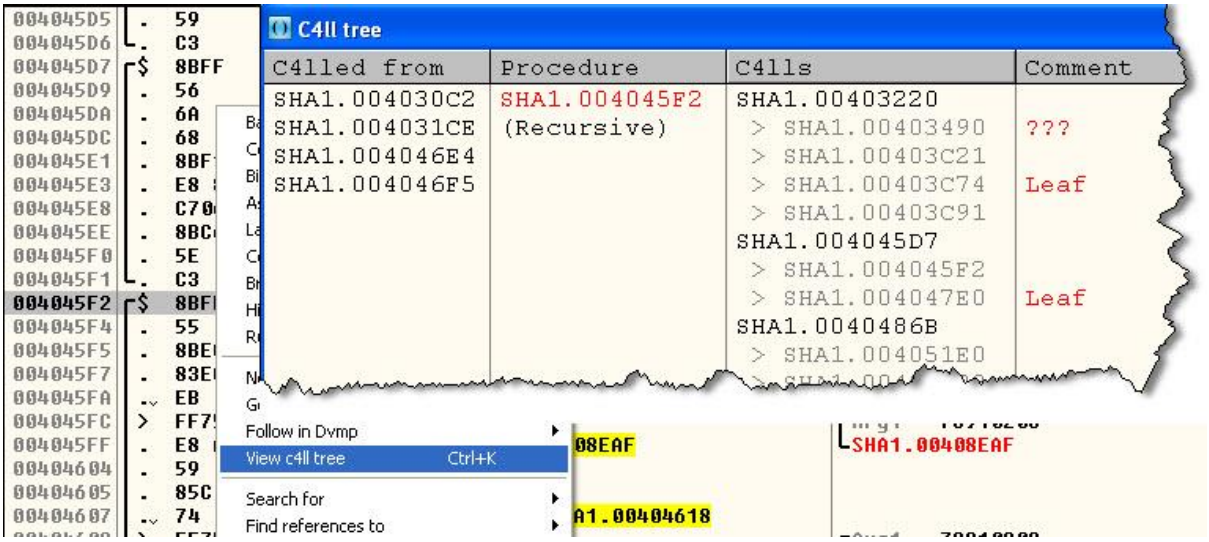
این پنجره هم برای خود دارای دسترسی و امکاناتی می باشد. اگر بر روی سطر دلخواهی کلیک راست کنید تا منوی سیستم بـه نمـایش در آید، چنین منویی مشاهده خواهید کرد. هرکدام از این گزینه های امکانات خاصی به ما میدهد کـه هرکـدام در جـای خـود بهتـرین کمک را به می کند:



در این منو گزینه هایی وجو دارد که نسبتا تکراری است و احتیاجی به توضیح نخواهد داشت و یا حداقل از نام آنهـا مـی تـوان براحتـی فهمید که چکار می کنند، پس به گزینه های اصلی و مهم می پردازیم و آنها را توضیح می دهیم:

❖ **Find References**: با انتخاب این گزینه شما میتوانید لیستی نواحی از برنامه را که چه بصورت مستقیم و یا غیر مستقیم تابع موردنظر ما را فراخوانی کرده اند را مشاهده کنیم.

❖ **View Call Tree**: با انتخاب این گزینه پنجره Call Tree برا ی تابع مورد نظر ما باز خواهد شد. این پنجره لیستی از فراخوانی های انجام شده در پروسیجر مشخص شده را نمایش میدهد.



همانطور که در تصویر می بینید ما بر روی یک پروسیجر عمل Call Tree را انجام دادیم و پنجره مربوط به آن باز شده است.این پنجره بطور کلی دارای 4ستون است.ستون ها به شرح ذیل هستند :

- **Called From**: این ستون به ما نواحی از برنامه را نشان میدهد که پروسیجرمان را فراخوانی می کنند.
- **Procedure**: نام رویه را نشان می دهد. یکی از ویژگی های جالب را میتوان در اینجا دید.اگر دقت کنید در ذیل نام رویه و آدرس آن کلمه Recursive نوشته شده است که معنی آن بازگشتی است، با این توصیفات میتوان فهمید که رویه حالت توابع بازگشتی دارد (خود فراخوانی).
- **Calls**: فراخوانی هایی می باشد که روی انجام می دهد.
- **Comment**: در این بخش هم توضیحاتی در مورد همان تابع نوشته می شود.

❖ **Help on symbolic name**: با انتخاب این گزینه اگر تابع مورد نظر در فایل win32.hlp وجود داشته باشد توضیحات را نمایش می دهد.

در ادامه منو دو بخش موجود می باشد که مربوط به بحث نقاط توقف است و در آن مبحث بطور کامل تمام نقاط توقف را شرح دادم.

» جستجو براساس یک دستورالعمل :

گاهی ممکن است که شما به دنبال یک دستور خاص در کدهای دی اسمبل شده بگردید . برای این کار شما می توانید از گزینه command استفاده کنید و کلید میانبر آن Ctrl + F میباشد .

از دستورات کلیدی می توانید استفاده کنید که در جدول ذیل این دستورات آمده است (صریح و غیرصریح ، دستورات اول صریح هستند و دومی غیر صریح). برای مثال این دستور MOV R32 , [CONST] به جای R32میتواند هر رجیستر (ثبات) 32 بیتی باشد .

Keyword	Matches – PRECISE
R8	Any 8-bit register (AL,BL, CL, DL, AH, BH, CH, DH)
R16	Any 16-bit register (AX, BX, CX, DX, SP, BP, SI, DI)
R32	Any 32-bit register (EAX, EBX, ECX, EDX, ESP, EBP, ESI, EDI)
FPU	Any FPU register (ST0..ST7)
MMX	Any MMX register (MM0..MM7)
CRX	Any control register (CR0..CR7)
DRX	Any debug register (DR0..DR7)
CONST	Any constant
OFFSET	Same as CONST
Command	Matches – IMPRECISE
JCC	Any conditional jump (JE, JC, JNGE...)
SETCC	Any conditional set byte (SETE, SETC, SETNGE ...)
CMOVCC	Any conditional move (CMOVE, CMOVC, CMOVNGE...)

نکته :در بعضی از مواقع شما نیاز دارید که دنبال یک استرینگ یا کدی بگردید که بیت های آن کد را می دانید . در این حالت از search for binary code استفاده می کنید.

» جستجو بر اساس دنباله ای از دستور العمل ها :

معمولا کامپایلرها از دنباله ای ثابت و مساوی از دستورات برای انجام دادن کارها استفاده می کنند. برای مثال معمولا همه پروسیجرها با این دو دستور شروع می شود :

```
PUSH EBP
MOV EBP, ESP
```

هنگام جستجوی دستورالعمل ها ، میتوان از دستورات کلیدی صریح و غیر صریح که در گذشته هم گفته شده است استفاده کنیم. در این نوع جستجو ما دارای محدودیت دستورات تایپ شده هستیم . ما میتوانیم حداکثر 8 دستور بنویسیم که هر خط یک دستور حساب میگردد.علاوه بر دستورات غیرصریح قبلی که گفتم دو دستور اضافه تر در این قسمت وجود دارد که عبارتند از : RA و RB .

این دو ثبات همانند R32 در دستورات قبلی می باشد با این تفاوت که هریک از آنها میتواند در طول دستور فقط یک نوع از ثبات باشند.برای درک بهتر این قضیه از یک مثال استفاده می کنیم . به دستور ذیل توجه کنید :

```
LEA RA, [4*RA+RA]
```

دستور بالا می تواند هریک از این دو جواب باشد :

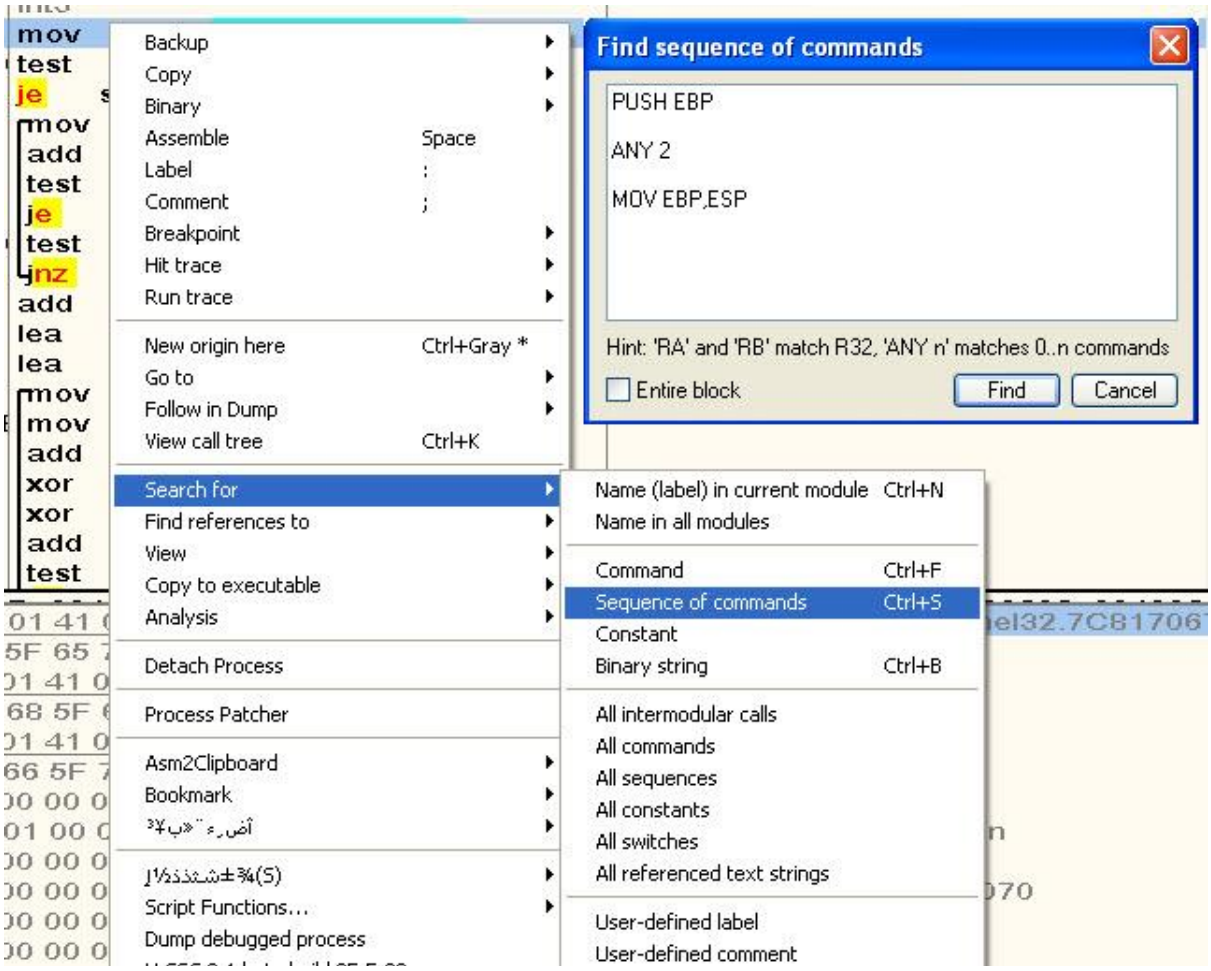
```
LEA EAX,[4*EAX+EAX]  -1
LEA ESI,[4*ESI+ESI]  -2
```

همانطور که می بینید در این دستورات همه ثبات ها یک نوع هستند و نمی تواند این جواب را داشته باشد:

```
LEA ESI, [EAX*4+EBX]
```

می بینید مشکل دستور بالااین است که در طول دستور از چند نوع ثبات استفاده شده است.

به غیر از این دو دستور غیر صریح ، دستور ANY n هم وجود دارد که به منظور پوشش دادن تعداد n دستورالعمل در دنباله مورد جستجو به کار گرفته شود.برای پیدا کردن مورد بعدی (Find Next) هم می توانید از کلیدهای میانبر Ctrl+L استفاده کنید .



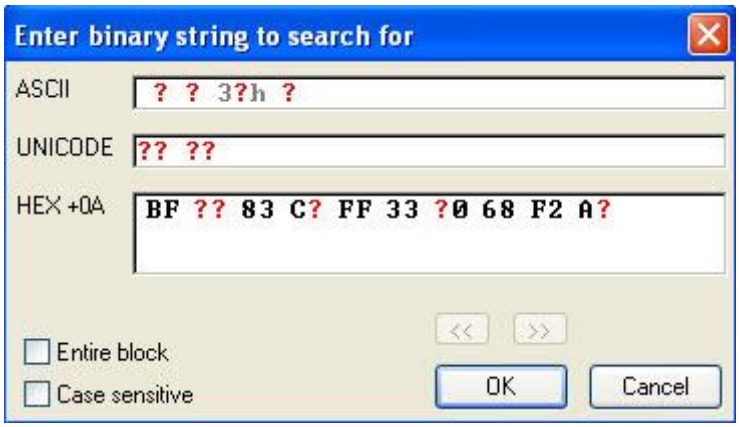
» جستجو بر اساس ثابت ها :

به ما اجازه میدهد تا در بین تمام ثابت (constant)های موجود در سکشن Code از Module جاری به جستجو ثابت مورد نظر بگردیم.

» جستجو براساس رشته های دودویی (حالت سه گانه) :

دیباگر OllyIce به ما این امکان را میدهد که رشته های باینری را جستجو کنیم. کلید میانبر آن Ctrl+B میباشد. در این نوع جستجو ما می توانیم از شیوه های ASCII, UNICODE و قالب هگزادسیمال استفاده کنیم .برای حرکت بین این 3 حالت می توان کلیدهای Ctrl+ArrowUp یا Ctrl+ArrowDn را بکاربرد . همانطور که که می بینید این پنجره دارای 2 دکمه ذیل کادرها می باشد ('»' And '«') . بوسیله این دکمه ها می توان بین 10 جستجوی آخر حرکت کنیم .در مد هگز ما می توانیم نیم بایت در مقایساتمان را مستثنی قائل شویم واز آن در جستجو استفاده نکنیم. برای این کار ما باید از علامت سوال (؟) استفاده کرد یا به قولی

بجای آن نیم بایت (Nibble Byte) هرمقدار قابل مجازی می تواند قرار بگیرد. شما می توانید مقادیر لازم را از درون سیستم خود کپی کرده و در کارد مورد نظر خود درج (Pate) نمایید.



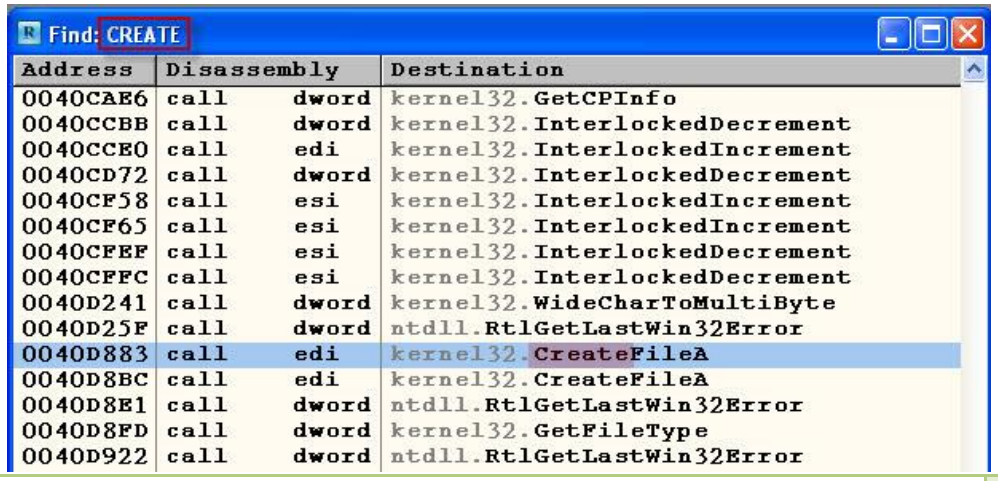
» جستجوی فراخوانی های خارجی انجام شده :

در این نوع جستجو ما میتوانیم بین تمام فراخوانی های خارجی (بعنوان مثال از یک DLL مخصوص فراخوانی شده است مانند: توابع API) استفاده کنیم و بر روی آنها BP بگذاریم. ما می توانیم تمام توابع پیدا شده را بر دو اساس مرتب کنیم :

1. آدرس
2. دستورات

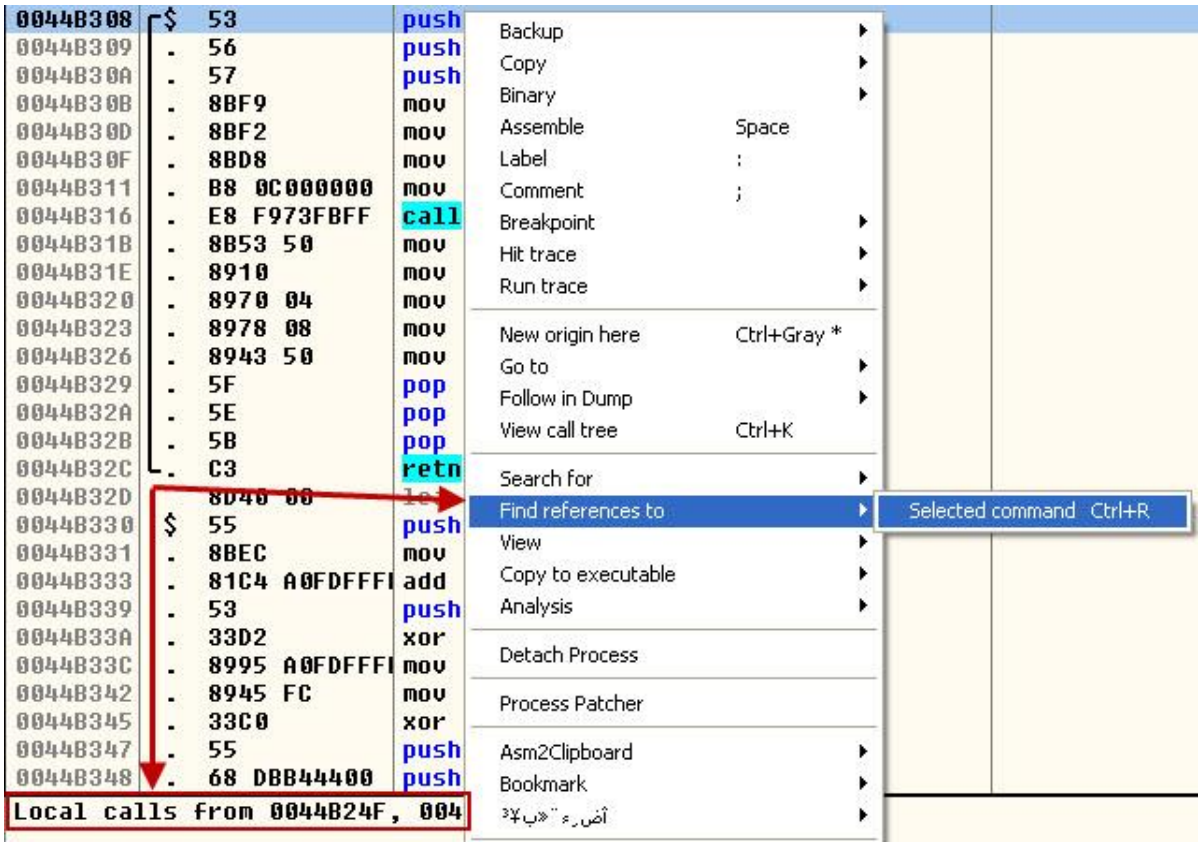
که هرکدام در جای خاص خود کاربرد دارد ولی بهترین نوع مرتب سازی همان مرتب سازی دستورات است. همانطور هم که در تصویر می بینید این دونوع مرتب سازی نام ستون های اول و آخر می باشد. یکی دیگر از قابلیت های خوب که در اینجا نمایان می شود جستجو راحت بین این توابع است. برای این کار فقط نام تابع خود را باید بدانید و آن را تایپ کنید !!! بله شما کلید ها را فشار می دهید: عمل جستجو انجام می شود و هم که کلیدهای فشرده شده در Title Bar نوشته می شود.

به عکس ذیل توجه کنید:



» جستجو ارجاع های انجام شده :

در بعضی از مواقع ما نیاز داریم لیستی از تمامی ارجاع های انجام شده به یک آدرس مشخص را تهیه کنیم. برای مثال موقعه ای که داشتم اسکین کرفتر رو کرک می کردم یک پروسیجر بود که باید حذف می شد بنابراین تمام ارجاع های انجام شده به اون پروسیجر رو پاک کردم این عاقلانه تر هستش تا اون همه دستور رو بخواهیم nop یا کار دیگه ای کنیم !!!! که بعدا دیدم احتیاج به اون کار هم نیست ☺ بهرحال ، برای این کار ما باید ابتدای دستور را انتخاب کنیم و به قسمت Information نگاهی کنیم می بینیم تمام آدرس های که دستورمان را فراخوانی می کنند را نوشته است. میتوان ابتدای دستور را که انتخاب کرده، بر روی آن کلیک راست نموده و مطابق شکل ذیل عمل نمایید .



بعد از آن پنجره ای باز می شود که لیست کامل ارجاع ها را به ما نشان میدهد. البته زیر منوهای این گزینه در مواقعی دارای گزینه های دیگری نیز می باشد که چند گزینه از آنها را توضیح داده خواهد شد:

Selected block: با این گزینه شما یک محدوده خاص را در نظر می گیرید و تمام ارجاع به این محدوده را می توانید پیدا کنید.

Immediate constant: تمام دستوراتی که به مقدار ثابت ذکر شده (ممکن است آدرس یک رشته باشد) در دستور جاری را جستجو می کند.

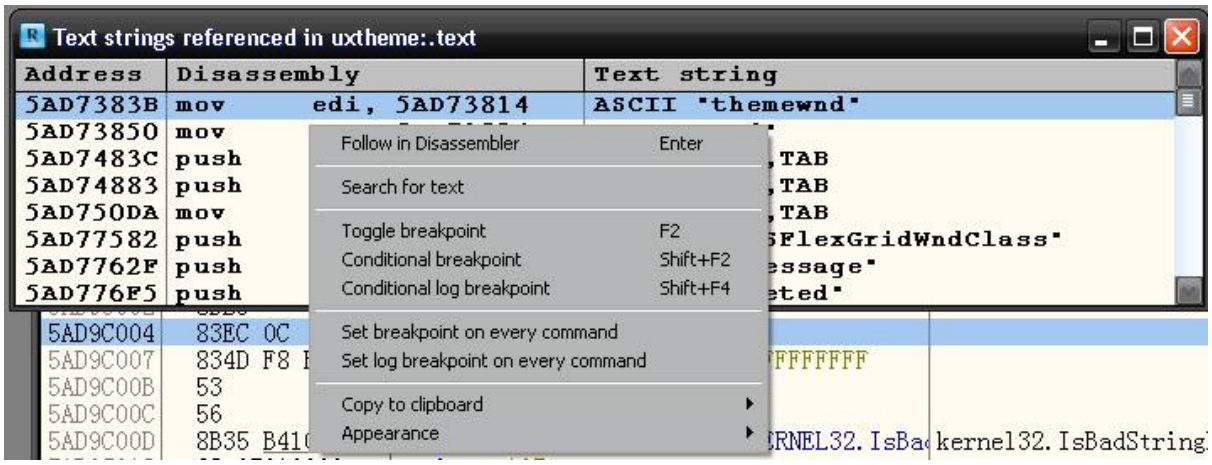
Jump destination: اگر چند پرش (JMP) به یک دستور باشد میتوانید با این گزینه تمامی پرش ها را جستجو کنید.

» جستجو در دستورات switches (راه گزین) :

این گزینه برای ما تمامی دستور Switch تشخیص داده شده را نمایش می دهد. با انتخاب این گزینه پنجره مربوط به آن باز می شود که گزینه خاصی برای توضیح ندارد.

» جستجو در لیست متن های موجود :

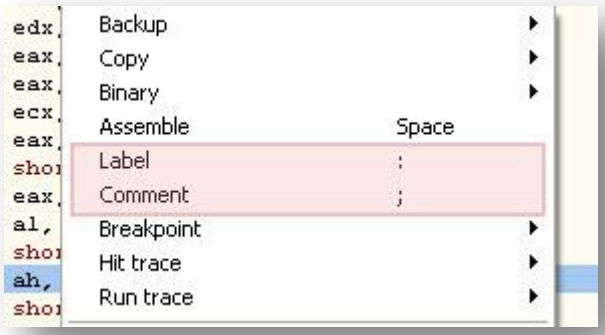
پرکاربرد ترین نوع جستجو می باشد. مثلا برنامه ای را می خواهید کرک کنید و عکس العمل آن در مقابل SN اشتباه پیغامی را نمایش می دهد و شما برای جستجوی این پیغام از این نوع جستجو استفاده می کنید . معمولا همه کرکرها اولین نوع جستجوی که یاد می گیرند این نوع است . بدین منظور می توانید از گزینه All referenced text strings استفاده کنید. پنجره مربوط به این گزینه را در ذیل می بینید :



این پنجره دارای منوی صفحه خاصی می باشد که از آنها می توان برای جستجو ، گذاشتن نقاط توقف بر روی دستور مورد نظر ، رفتن به دستور مورد نظر این متن و ... می باشد .

» جستجو در برچسب ها و توضیحات تعریف شده توسط کاربر:

همانطورکه می دانید و در عکس می بینید ما میتوانیم برای هر خط دستور در پنجره DSM توضیحاتی را بنویسیم و یا برچسب هایی برای آن خط کد در نظر بگیریم. چون ما برنامه را Trace می کنیم معلوم نیست که سر از کجا در بیاوریم به همین دلیل می توانیم برای هر خط توضیحاتی قرار دهیم و اگر آدرس آن را فراموش کردیم توسط این گزینه های جستجو به راحتی می توانیم به آنها دسترسی پیدا کنیم:


Backup
Copy
Binary
Assemble Space
Label :
Comment ;
Breakpoint
Hit trace
Run trace

» بررسی شماره های دسترسی :

هر برنامه که در ویندوز اجرا می شود شماره های دسترسی مخصوص و منحصر به فرد خود را دارد که به آن همان هندل می گوییم (همان hwnd خودمان در VB ☺). این هندل ها می توانند مربوط به فایلهای باز شده توسط برنامه مورد نظر ، کلیدهای رجیستری و ... باشد .

برای دسترسی به لیست این شماره ها می توانید از منوی View→Handles و یا در تولبار انتخاب کنید. همانطور که در تصویر مربوطه می بینید این پنجره دارای ستون هایی می باشد که هرکدام اطلاعاتی را به ما می دهد مانند : شماره هندل،نوع هندل ایجاد شده و ...

اطلاعات این پنجره به طور خودکار بازسازی نمی شود و برای Refresh اطلاعات این صفحه می بایست از منوی صفحه گزینه Actualize را انتخاب کنیم. گزینه های تغییر یافته نسبت به مرحله قبل به رنگ قرمز در می آید.

Handles

Handle	Type	Refs	Access	T	Info	Name
0000000C	File (dir)	2.	0010002			[redacted]
0000015C	File (dir)	2.	0010002			c:\WINDOWS\WinSxS\x86_Micr
00000164	File (dir)	2.	0010002			c:\WINDOWS\WinSxS\x86_Micr
00000188	File (dir)	2.	0010000			e:\Reversing\DBG & DAsem & D
00000030	Key	2.	000F003			HKEY_LOCAL_MACHINE
00000044	Key	2.	0002001			HKEY_LOCAL_MACHINE\SYST
00000048	Key	2.	0002001			HKEY_LOCAL_MACHINE\SYST
0000004C	Key	2.	0002001			HKEY_LOCAL_MACHINE\SYST
00000074	Key	2.	000F003			HKEY_CURRENT_USER
00000080	Key	2.	0002001			HKEY_LOCAL_MACHINE\SYST
00000084	[illegible]	[illegible]	000[illegible]	[illegible]	[illegible]	[illegible]

» بررسی نواحی حافظه برنامه :

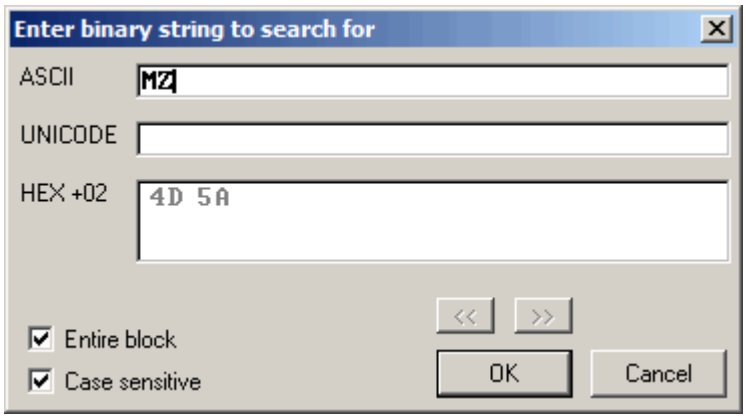
در این قسمت می خواهیم به شرح بیشتر پنجره Memory map بپردازیم .کلید میانبر این پنجره Alt+M می باشد و یا میتوان از تولبار M را انتخاب کرد. این پنجره بلاک های اشغال شده از حافظه توسط برنامه در حال دیباگ را نشان میدهد اگر بلاک های حافظه بخش هایی از مدل قابل اجرا (executable module) باشد ، Olly مشخص می کند که این بلاک شامل کدام نوع از داده هاست : کد(Code) ، داده(Data) ، منابع(Resource) و ... که پیش از این در مبحث نقاط توقف با این پنجره آشنا شدید. این پنجره دارای ستون هایی می باشد که می توان از آنها در مواقع لزوم اطلاعات مفیدی بدست آورد:

- Address : در این ستون ما می توانیم آدرس شروع ناحیه مورد نظر را در حافظه برنامه ببینیم .
- Size: ما میتوانیم مقدار فضای اختصاص داده شده به ناحیه مورد نظر در حافظه را ببینیم که مقدار آن به دو صورت Hex, Dec نمایش داده می شود.
- Owner: اگر ناحیه مورد نظر مربوط به یک فایل اجرایی و یا DLL باشد، نام آن فایل را نمایش میدهد.
- Type: در این ستون اطلاعاتی از نظر ایجاد و استفاده از ناحیه مورد نظر نمایش داده می شود که می تواند اطلاعات ذیل را به خود اختصاص دهد که هرکدام دارای مفهوم خاصی می باشد :
  - Priv: همانطور که از نام آن مشخص است این نواحی برای ذخیره سازی و نگهداری اطلاعات خصوصی برنامه استفاده می شود
  - Map: مربوط به فایل های Map شده (نگاشت) در حافظه پروسه مامی باشد.
  - ImG: تصویری از فایل را نگهداری می کند.
- Access: دسترسی ناحیه مورد نظر از حافظه را از نظر خواندنی،نوشتنی و یا اجرا را مشخص می کند.

در منوی راست کلیک این پنجره گزینه های متفاوتی وجود دارد که در ذیل آنها شرح داده خواهد شد :

- Actualize: عمل این گزینه در این پنجره با توضیحاتی که قبلا داده شده است فرقی ندارد. به روز کردن بلاک های حافظه و حذف نشانه گذاری از بلاک های جدید.
- View in Disassembler: بلاک مورد نظر از حافظه را در در پنجره DSM نمایش میدهد. . این گزینه تنها زمانی موجود هست که بلاک مد نظر شامل کد های اجرایی باشد.
- Dump in CPU: بلاک مورد نظر از حافظه را در بخش Dump از پنجره DSM نمایش خواهد داد.

- Dump: تفاوت این گزینه با قبلی در این است که بجای نمایش در بخش Dump ما بصورت یک پنجره جداگانه می توانیم آن را مشاهده کنیم.اگر نوع داده های بلاک مشخص باشد olly بطور اتوماتیک قالب مخصوص برای نمایش آنها در Dump را انتخاب می کند.
- Search: به ما این امکان را میدهد که از بلاک انتخاب شده به بعد توسط جستجوی سه گانه به جستجوی رشته مورد نظر بپردازیم. اگر رشته مد نظر پیدا شد، Olly بلاک حافظه پیدا شده را نسخه برداری (Dump) می کند. پنجره های Memory map و Dump برای جستجو الگوی یکسان دارند پس شما می توانید جستجو را در پنجره Dump برای موارد دیگر ، ادامه دهید. همچنین شما می توانید با فشردن کلید Esc پنجره Dump را ببندید.
- Search next - Ctrl+L : **تکرار آخرین جستجو**



- Set access: می توانیم صفات یک بلاک در حافظه را تغییر بدهیم ، که میتواند مقادیر ذیل را به خود اختصاص دهد :
  - No access
  - Read only
  - Read/write
  - Execute
  - Execute/read
  - Full access
- Zero Memory: تمام بایت های بلاک مورد نظر را به صفر تبدیل میکند.یعنی از بین بردن داده ها.
- Dump Memory-Area : توسط این گزینه می توان یک نسخه از بلاک مورد نظر را در یک فایل ذخیره نمود.
- Load Dumped Memory : میتوان نسخه گرفته شده از بلاک مورد نظر را بازگردانی نمود .
- Allocate Memory: با این گزینه میتوان به مقدار مورد نیاز یک بلاک در قسمتی از حافظه ایجاد کنیم.

» مدیریت Threads :

در این پنجره به راحتی میتوان ترید های برنامه را مدیریت نمود. در منوی این صفحه می توان گزینه های ذیل را مشاهده کرد :

- Actualize: بروز آوری صفحه مورد نظر و مشخص کردن گزینه های جدید و اضافه شده با رنگی متفاوت.
- Suspend: ترید مورد نظر را به حالت معلق در می آورد.
- Resume: ترید مورد نطر اگر در حالت معلق باشد برای اامه کار و بیرون آمدن از حالت معلق از این گزینه استفاده می شود.
- Set priority: می توان اولویت ترید در پروسه برای پردازش را تعیین کرد.که میتواند مقادیر ذیل را به خود بگیرد :

  - Idle ----------/ پایین ترین اولویت
  - Lowest |
  - Low |
  - Normal |
  - High |
  - Highest |
  - Time critical---\ بالاترین اولویت

- Open in CPU: می توان بجای این گزینه دابل کلیک نمود .می توان موقعیت جاری از ترید مورد نظـر را در پنجـره CPU ببینیم.
- Kill Thread: می توان ترید مورد نظر را از بین ببریم.

» بررسی فایل های DLL مورد استفاده برنامه :

در این پنجره ما میتوانیم تمامی ماژول های اجرایی(DLL) بارگزاری شده درون پروسه در حال دیباگ را مشاهده کنیم. این پنجره به ما اطلاعات مفید دیگری هم میدهد بعنوان مثال سایز ماژول ، آدرس آغاز، مسیر ماژول و یا ورژن آن را می توان مشاهده نمود.

در منوی این صفحه گزینه های ذیل موجود می باشد :

- View memory: با گزینه ما به پنجره Memory و قسمتی که کاژول مورد نظر شروع شده است منتقل می شویم.
- View code in CPU: کدهای ماژول را در پنجره DSM به ما نمایش میدهد
- View names: نمایش توابع تعریف شده (exports, imports, library, user-defined) و یا استفاده شده در ماژول جاری.
- Update .udd file now: برای ماژول مورد نظر یک فایل با پسوند UDD. در دایرکتوری UDD می سازد و اگر وجود داشته باشد آن را به روز رسانی میکند.در این فایل که با نام ماژول مورد نظر ذخیره می شود تمامی اطلاعات مانند نقاط توقف گذاشته شده، توضیحات درج شده چه توسط خود برنامه و چه توسط ما،... را ذخیره میکند.البته Olly بطور اتوماتیک هنگام از بین رفتن ماژول از حافظه فایل UDD را ایجاد می کند.
- View all resources: نمایش تمامی resource های موجود در ماژول مورد نظر به همراه اطلاعات مفید دیگر. شما میتوانید با گزینه dump از ریسورس مورد نظرمان یک نسخه درون سیستم ذخیره کنید و به صورت باینری آن را ویرایش نماییم.

» بررسی فراخوانی های انجام شده :

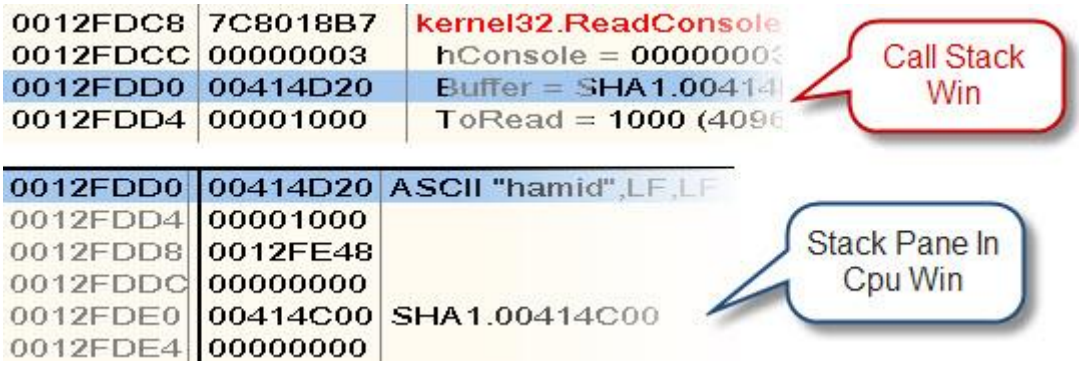
همانطور که می دانید برای فراخوانی توابع و ذخیره آدرس بازگشت (برای دستور Call)از پشته استفاده می شود(دو تا سوال کنکوری هم یاد گرفتید ☺).

گاهی اوقات در جایی از برنامه به هرعلتی توقف می کنید و می خواهید لیستی از فراخوانی های انجام شده تا این لحظه را داشته باشید.دیباگر با تجزیه و تحلیل اطلاعات پشته یک لیست از توابع انجام شده تاکنون را بهمراه اطلاعاتی نظیر : محل قرارگیری تابع در پشته ، آدرس دستورالعملی که تابع مورد نظر را فراخوانی کرده است ، نام تابع و آرگومان های ارسالی و ... را در اختیار ما قرار می دهد که می توانید در تصویر ذیل ببینید.همانطور که گفتم باید برنامه در حالت توقف باشد حالا چه توسط نقاط توقف و یا Pause کردن برنامه و ... این توقف ایجاد شود.برای دسترسی به این گزینه از کلید میانبر Alt+K می توان استفاده نمود و یا از تولبار را انتخاب نمایید تا پنجره Call Stack به نمایش در آید.این پنجره دارای 5ستون است .

Call stack of main thread

Address	Stack	Procedure / arguments	Called from	Frame
0012FC20	7C90DACC	Includes ntdll.KiFastSy	ntdll.7C90DACA	0012FC40
0012FC24	7C912DC8	ntdll.ZwRequestWaitRe	ntdll.7C912DC3	0012FC40
0012FC44	7C872931	ntdll.CsrClientCallServ	kernel32.7C87292B	0012FC40
0012FD40	7C872A78	? kernel32.7C872769	kernel32.7C872A73	0012FD3C
0012FDC8	7C8018B7	kernel32.ReadConsole	kernel32.7C8018B2	0012FDC4
0012FDCC	00000003	hConsole = 00000003		
0012FDD0	00414D20	Buffer = SHA1.00414		
0012FDD4	00001000	ToRead = 1000 (4096		
0012FDD8	0012FE48	pRead = 0012FE48		
0012FDDC	00000000	pReserved = NULL		
0012FE20	004084BE	? kernel32.ReadFile	SHA1.004084B8	0012FE1C
0012FE24	00000003	hFile = 00000003		
0012FE28	00414D20	Buffer = SHA1.00414		
0012FE2C	00001000	BytesToRead = 1000		
0012FE30	0012FE48	pBytesRead = 0012F		
0012FE34	00000000	pOverlapped = NULL		
0012FE64	0040893D	SHA1.004082BB	SHA1.00408938	0012FE60
0012FEA8	00404E9D	? SHA1.0040887D	SHA1.00404E98	0012FEA4
0012FEC4	004035AD	SHA1.00404E20	SHA1.004035A8	0012FEC0
0012FF04	004036C6	? SHA1.0040349F	SHA1.004036C1	0012FF00
0012FF18	004010F1	SHA1.004036B5	SHA1.004010EC	0012FF14
0012FF7C	0040477B	SHA1.00401000	SHA1.00404776	0012FF78

- Address: این ستون شامل آدرس تابع و یا پارامتر های آن در پشته می باشد.
- Stack :آدرس متناظر ، آدرس مورد نظر در پشته را بر می گرداند.به شکل ذیل توجه کنید و دو ستون اول از دو پنجره را بررسی نمایید :



- Procedure (or Procedure / arguments) : در این ستون می توان نام توابع و یا آرگومانهای آن را ببینیم. در تصاویر بالا می بینید که کنار نام توابع علامت ؟ وجود دارد. این بدان معناست که نقطه ورودی پیدا شده برای تابع معتبر نمی باشد.
- Called from: در این ستون آدرس دستور العملی می باشد که تابع مورد نظر را فراخوانی نموده است.
- Frame: این ستون بطور پیش فرض نمایش داده نمی شود و اگر مقدار اشاره گر فریم شناخته شود،آن را نمایش می دهد (register EBP).

این پنجره هم دارای منوی و گزینه هایی می باشد که کار آنها تماما قابل فهم و درک است (اگر تا به اینجا همگام با من بوده باشید براحتی کار گزینه ها را تشخیص می دهید).تنها گزینه Execute to Return نسبتا گنگ می باشد . با انتخاب این گزینه برنامه تا هنگام بازگشت از تابع مورد نظر ادامه می یابد.یعنی ما برنامه را اجرا می کنیم و بعد از آدرسی که تابع مورد نظر را فراخوانی می کند متوقف می شویم.

» کارگاه عملی :

در اینجا قصد داریم به ذکر یک مثال در ارتباط با Olly بپردازیم. این روزها بازار Cryptor ها خیلی داغ است ☺ پس تصمیم گرفتیم سراغ مثالی در این مورد برویم تا کاری کرده باشیم که به اندازه دو کار ارزش داشته باشد (با یک تیر دو نشان زده ایم) ، هم مثالی عملی در رابطه با این دیباگر می باشد و هم بحثی جالب در مورد Cryptor ها و بیرون کشیدن فایل اصلی از دل آنها است پس هدف ما بیرون کشیدن فایل Crypt شده است.

برنامه Notepad ویندوز رو با یه برنامه کد باز Crypt کردیم.سورس این Cryptorدر کنار آموزشی موجود است.

برنامه را در Olly لود کرده و با F9 اجرا می کنیم. می بینیم که برنامه Terminate میشود ، اما برنامه در حال اجرا است !!! خب یه طور دیگر به ماجرا نگاه می کنیم.

به قسمت Attach برنامه میرویم و نگاهی به پروسه ها می اندازیم. با پنجره ای مانند ذیل برخورد می کنیم:

Process	Name	Window	Path
000003D8	Notepad_Crp	Untitle	C:\Documents and Settings\Administrator\Des
0000042C	csrss		\??\C:\WINDOWS\system32\csrss.exe
00000508	winlogon		\??\C:\WINDOWS\system32\winlogon.exe
00000538	spoolsv		C:\WINDOWS\system32\spoolsv.exe
00000548	Notepad_Crp		C:\Documents and Settings\Administrator\Des

برنامه ای که در حال دیباگ است با رنگ قرمز مشخص شده و می توانید ببینید که یه پروسه دیگر با مسیر برنامه در حال دیباگ هم ایجاد شده و دارای PID متفاوتی هست.

نتیجه : احتمال زیاد برنامه اصلی یه پروسه دیگر ایجاد کرده و بعد پروسه خودش را می بندد.

حالا باید دنبال سر نخ باشیم تا برنامه را گیر بیاندازیم. می شود حدس زد که برنامه از تابع CreateProcess برای ایجاد پروسه جدید استفاده می کند پس تو Olly بر روی تابع مورد نظر BreakPoint می گزاریم.

بر روی پنجره CPU از برنامه راست کلیک کرده و از Search for / All Intermodular Calls به دنبال تابع مد نظر می گردیم :

OllyICE - [Find: CREATEPROCESS]

File View Debug Plugins Options Window Help

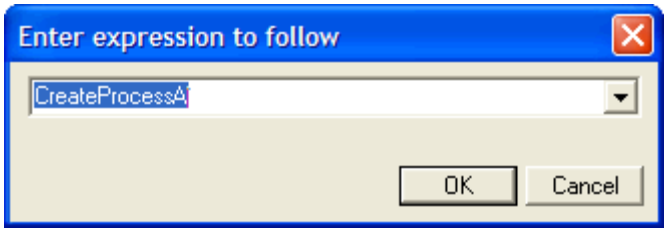
Paused

Address	Disassembly	Destination
004054E8	push ebp	(Initial CPU selection)
00402945	call <jmp.&user32.CharNe	user32.CharNextA
00402981	call <jmp.&user32.CharNe	user32.CharNextA
0040298D	call <jmp.&user32.CharNe	user32.CharNextA

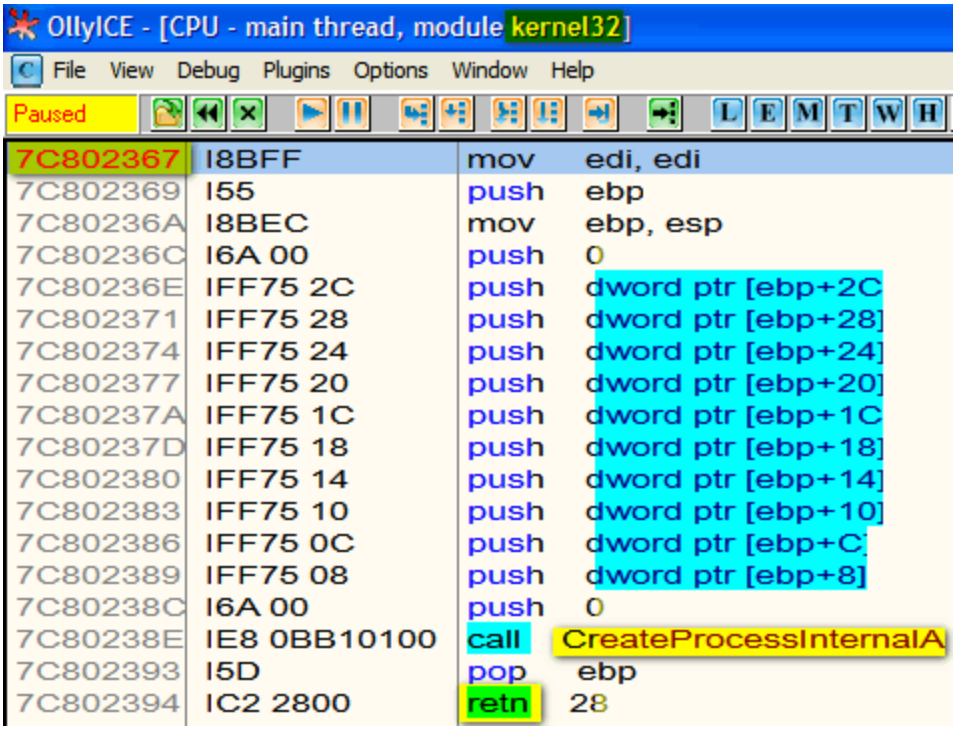
خب چیزی پیدا نمی کنیم ... یعنی برنامه از این تابع استفاده نمی کند!؟

در کل برای فراخوانی توابع دو نوع روش وجود دارد : یکی **استاتیک** و یکی **داینامیک** . توابعی که برای ما لیست شده اند فقط توابعی هستند که برنامه به صورت استاتیک استفاده کرده و امکان دارد برنامه به صورت داینامیک توابع رو فراخوانی کرده باشد.(در فراخوانی داینامیک آدرس توابع به وسیله تابع GetProcAddress به دست می آید) برای این مواقع ما بر روی خود تابع از Dll مربوطه BreakPoint می گزاریم .

دکمه را از نوار ابزار Olly کلیک می کنیم و نام تابع را می نویسیم (تابع CreateProcess ورژن اسکی و یونیکد دارد که ورژن اسکی با اضافه شدن حرف A و یونیکد با W آخر تابع مشخص می شود).



دقت کنید که بزرگی و کوچکی حروف باید رعایت شود!!! OK رو انتخاب می کنیم. Olly ما را به جای مد نظر می برد:



در عنوان پنجره مشخص میشود که تابع از Kernel32.dll فراخوانی شده و کل تابع رو در شکل میبینید که تابع دوباره خود با فراخوانی تابع CreateProcessInternalA و با Retn به کارش خاتمه می دهد . ابتدای تابع یعنی آدرس 7C802367 با F2 یک BP می گذاریم. این BP باعث میشود هر جایی که برنامه به خواهد از این تابع استفاده کند ، Olly برنامه رو متوقف کرده و کنترل برنامه رو بدست ما می دهد.

خب برنامه رو با F9 اجرا می کنیم. می بینیم که Olly متوقف می شود . به قسمت Stack برنامه یه نگاه میندازیم:

```
0040521  CALL to CreateProcessA from Notepad_.00405213
00D5053  ModuleFileName = "C:\Documents and Settings\Administrato
004036F  CommandLine = ""
0000000  pProcessSecurity = NULL
0000000  pThreadSecurity = NULL
0000000  InheritHandles = FALSE
0000000  CreationFlags = CREATE_SUSPENDED
0000000  pEnvironment = NULL
0000000  CurrentDir = NULL
0012FF0  pStartupInfo = 0012FF04
0012FF4  pProcessInfo = 0012FF48
```

در این قسمت اطلاعات ارزشمندی وجود داره : تابع از آدرس 405213 برنامه فراخوانی شده و پروسه جدید در حالت Suspend قرار دارد. بر روی خط اول مشخص شده از شکل دکمه Enter را بزنید تا به محل فراخوانی از برنامه اصلی برویم:

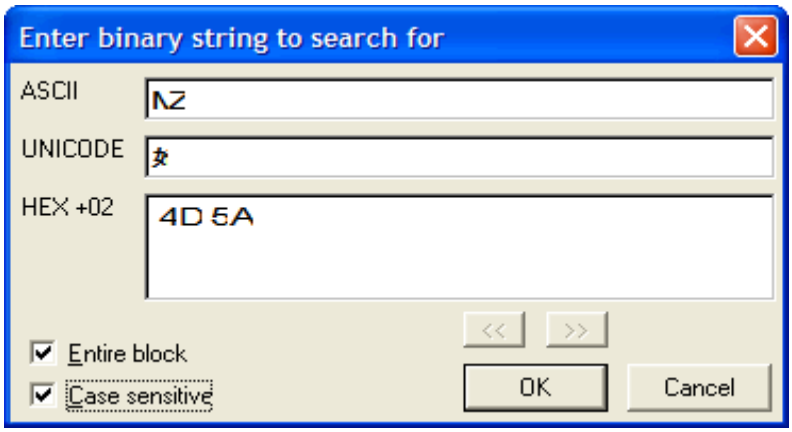
```
00405211  .M3B00   mov   eax, dword ptr [eax]
00405213  .MFFD0   call  eax
00405215  .M35C0   test  eax, eax
```

حالا به راحتی می توانیم از این قسمت برنامه شروع به Trace کنیم. می توان BP قبلی را پاک کنید و روی این قسمت یک BP بزاریم و از اینجا شروع Trace کنیم. تا به اینجا برسیم:

```
00405404  .MFFD0   call  eax   kernel32.ResumeThread
```

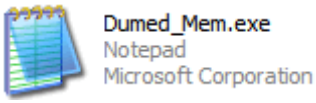
از این قسمت هم با F8 رد شوید. بله به محض رد شدن برنامه اصلی اجرا می شود !!! خب روی این آدرس یک BP می گزاریم و BP های قبلی رو پاک می کنیم. برنامه رو ریست می کنیم و دوباره با F9 برنامه رو اجرا می کنیم. و روی BP هستیم... حالا سراغ بلاک های مورد استفاده برنامه می رویم . با کلیک روی M وارد قسمت Memory Map می شویم. حال دنبال فایل Exe خودمان می گردیم. می دانیم که هر برنامه Exe با رشته MZ شروع می شود. پس در بلاک های حافظه به دنبال این مقدار می گردیم. Ctr+B را می زنیم و وارد پنجره جستجو می شویم.

در قسمت ASCII رشته مد نظر را می نویسیم:



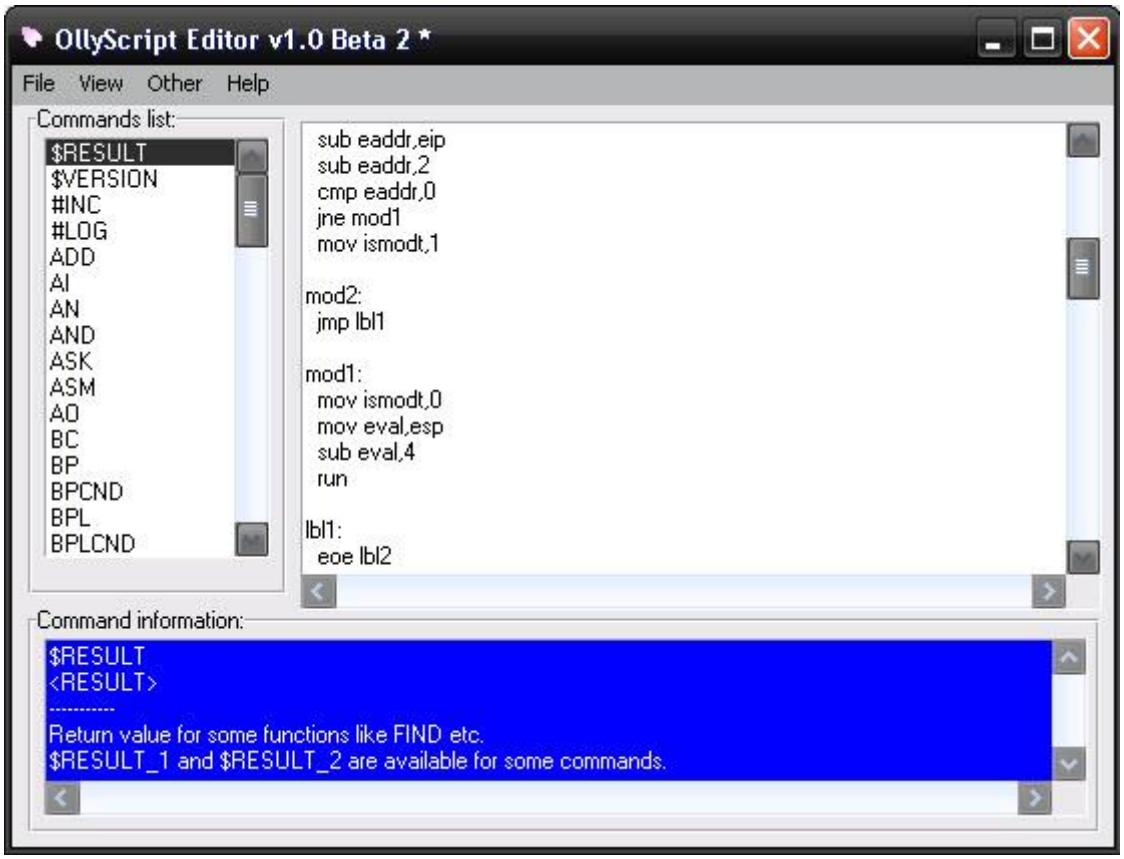
تیک های دو گزینه پایین پنجره رو هم می زنیم. و بعد OK می زنیم. خب برنامه یک مورد پیدا کرد ... کمی Scroll را پایین می کشیم ، می بینیم که حاوی اطلاعاتی نیست ، پس این نیست. پنجره را می بندیم و برای جستجوی دوباره Crt+L را می زنیم.

به یه مورد دیگر بر می خوریم . این بلاک رو بر روی دیسک ذخیره می کنیم تا ببینیم چه شکار کرده ایم (; بر روی پنجره Dump راست کلیک کرده و Backup / Save Data to File را انتخاب می کنیم و اطلاعات را ذخیره می کنیم. فایل ذخیره شده در ابتدای فایل کمی ناخالصی دارد و باید این ناخالصی را تا قسمت MZ پاک می کنید(من این کار را با Notepad++ انجام می دهم) و دوباره ذخیره می کنیم. در انتها خواهید دید که فایل مورد نظر بوده که ذخیره کردیم :



» اسکریپت نویسی برای Olly :

زبان اسکریپت نویسی برای Olly خیلی شبیه به زبان اسمبلی است و بهمین خاطر اسکریپت نویسی خیلی راحت میشود ولی بازهم باید نکاتی را رعایت کرد و دستورات را هم دقیقا دانست . برای اجرای اسکریپت ها باید از منوی Plugins →OllyScript→RunScript را انتخاب کرده و فایل اسکریپت خود را انتخاب میکنیم . این پلاگین توسط 2 نفر نوشته شده است یک نفر با نام E3 ونفر دیگر هم با نام SHaG . پلاگین E3(Epsylon3) دارای بیش از 100دستور است که راهنمای آن بصورت Pdf در قسمت Attachments بانام Epsylon3(E3) موجود است. برنامه OSEditor برای نوشتن اسکریپت میباشد واین برنامه را فردی چینی با نام مستعار LoveBoom براساس پلاگین SHaG که حاوی 74 دستور می باشد (که تمام آن در Help کنار برنامه موجود است) نوشته است و دارای دو زبان چینی و انگلیسی است . این برنامه در قسمت Attachments موجود می باشد . .چیز جالبیه ☺ .



کلید میانبر :

در ذیل کلید میانبرهای موجود برای بخش DSM و در بخش بعد کلید هایی که در تمام پنجره ها کار می کنند و بطور گسترده (Wide) موجود است را می بینید :

Shortcuts DSM :

ENTER - add selected command to the command history and, if current command is a jump, call or part of the switch table, follow address of destination.

BkSpc - remove analysis from the selection, useful if Analyzer recognized code as data. See also decoding hints.

Alt+BkSpc - undo selection, substitutes selected part of the code with the corresponding portion of backup data. Available only when backup data exists and differs from selected code.

Ctrl+F1 - if API help file is selected, open help topic associated with symbolic name in the first selected line.

F2 - toggle INT3 breakpoint on the first selected command. Alternatively, you can double-click line in the second column.

Shift+F2 - set conditional breakpoint on the first selected command. See also Ignore memory access violations in Kernel32.

F4 - run to selection, set one-shot breakpoint on the first selected command and continue execution of debugged program. If OllyDbg catches exception or stops on breakpoint before program reached this command, one-shot breakpoint remains active. If necessary, you can remove it in Breakpoints window.

Shift+F4 - set logging breakpoint (conditional breakpoint with optional logging of value of some expression when condition is met). For more details, see Breakpoints.

Ctrl+F5 - open source file that corresponds to the first selected command.

Alt+F7 - go to the the previous found reference.

Alt+F8 - go to the next found reference.

Ctrl+A - analyse code section of current module.

Ctrl+B - start binary search.

Ctrl+C - copy selection to the clipboard. Copy roughly preserves width of the columns and truncates invisible characters. To exclude some column from the copy, reduce it width to the minimum.

Ctrl+E - edit selection in binary (hexadecimal) format.

Ctrl+F - start command search.

Ctrl+G - go to address. Invokes window asking you to enter address or expression to follow. This command does not modify EIP.

Ctrl+J - list all calls and/or jumps to the current location. You must analyze code before you can use this feature.

Ctrl+K - view Call tree associated with current procedure. You must analyze code before you can use this feature.

Ctrl+L - search next, repeats last search.

Ctrl+N - open list of names (labels) in current module.

Ctrl+O - scan object files. This command displays the Scan object files dialog where you can select object files or libraries and scan them in an attempt to find object modules within the actual code section.

Ctrl+R - find references to selected command. This command scans through the executable code of the active module and finds all references (constants, jumps or calls) to the first selected command. You can use shortcuts Alt+F7 and Alt+F8 to navigate through the references. For your convenience, referenced command is also included into the list of references.

Ctrl+S - search for command. This command displays Find command dialog where you can enter the assembler command and search for the next instance of this command.

Asterisk (*) - go to the origin (contents of EIP of the active thread.(

Ctrl+Gray Asterisk (*) - new origin here, sets EIP of the currently selected thread to the address of the first selected byte. You can undo this operation if you go to Registers pane and select EIP.

Plus (+) - if run trace is inactive, go to the next address from the command history. Otherwise, go to the next record in run trace data.

Ctrl+Plus - go to the beginning of the previous procedure.

Minus (-) - if run trace is inactive, go to the previous address from the command history. Otherwise, go to the next record in run trace data.

Ctrl+Minus - go to the beginning of the next procedure.

Space - assemble. Displays Assemble at dialog where you can edit actual command or enter new commands in Assembler language. These commands substitute actual code. Alternatively, double-click the command you are going to change.

Colon (:) - add label. Displays Add label or Change label window where you enter label (symbolic name) associated with the first byte of the first selected command. Notice that in many languages colon is a part of label.

Semicolon (;) - add comment. Displays Add comment or Change comment window where you enter comment (text string displayed in the last column) associated with the first byte of the first selected command. Notice that many Assembler languages use semicolon to start comment. Alternatively, you can double-click disassembled line in the Comments column.

Shortcuts OllyDbg-wide :

Ctrl+F2 - program reset, starts over debugged program. If there is no active program, OllyDbg restarts first program in the history list. Program reset removes memory and hardware breakpoints.

Alt+F2 - close, closes debugged program. If program is still active, you will be asked to confirm the

action.

F3 - displays „Open 32-bit .EXE file" dialog box where you can select executable file and optionally specify arguments.

Alt+F5 - makes OllyDbg topmost. If debugged program stops on breakpoint while displaying topmost window (usually some kind of modal message or dialog), this window may cover parts of OllyDbg but you are unable to move or minimize it without continuation. Activate OllyDbg (for example, from the taskbar) and press Alt+F5. OllyDbg will get topmost and place itself over the Debuggee. If you press Alt+F5 for the second time, OllyDbg becomes normal (non-topmost) window. Topmost status is preserved between debugging sessions. Actual OllyDbg status is displayed in the status bar.

F7 - step into, executes next single command. If this command is a function call, stops on the call destination. If command has REP prefix, executes single iteration of the command.

Shift+F7 - same as F7, but if debugged program stopped on some exception, Debugger will first try to pass exception to handler specified in the debugged program (see also Ignore memory access violations in Kernel32).

Ctrl+F7 – animate into, executes commands step-by-step, also entering function calls (as if you press and hold F7, only faster). Animation stops when you execute some other stepping or continuation command, program reaches active breakpoint or some exception happens. Each time next step is executed, OllyDbg redraws all windows. To speed up animation, close all windows you don't use and reduce the size of remaining windows. To stop animation, press Esc.

F8 - step over, executes next single command. If this command is a function call, executes called function at once (except when function contains breakpoint or produces exception). If command has REP prefix, executes all iterations and stops on the next command.

Shift+F8 - same as F8, but if debugged program stopped on some exception, Debugger will first try to pass exception to handler specified in the debugged program (see also Ignore memory access violations in Kernel32).

Ctrl+F8 – animate over, executes commands step-by-step, without entering function calls (as if you press and hold F8, only faster). Animation stops when you execute some other stepping or continuation command, program reaches active breakpoint or some exception happens. Each time next step is executed, OllyDbg redraws all windows. To speed up animation, close all

windows you don't use and reduce the size of remaining windows. Press Esc to stop animation.

F9 - continues program execution.

Shift+F9 - same as F9, but if debugged program stopped on some exception, Debugger will first try to pass exception to handler specified in the debugged program (see also Ignore memory access violations in Kernel32).

Ctrl+F9 - execute till return, traces program without entering function calls or updating CPU till the next encountered return. As program is executed step-by-step, this may take some time. Press Esc to stop tracing.

Alt+F9 - execute till user code, traces program without entering function calls or updating CPU and stops when next encountered command belongs to module that doesn't reside in system directory. As program is executed step-by-step, this may take some time. Press Esc to stop tracing.

Ctrl+F11 - run trace into, executes commands step-by-step entering function calls and adding contents of registers to run trace data. Run trace doesn't animate CPU window.

F12 - stops program execution by suspending all threads of debugged program. Don't resume threads manually, rather use ordinary continuation keys and menu items (like F9).

Ctrl+F12 - run trace over, executes commands step-by-step without entering function calls and adds contents of registers to run trace data. Run trace doesn't animate CPU window.

Esc - if animation or tracing is active, stops animation or tracing. If CPU displays trace data, shows actual data.

Alt+B - opens or restores Breakpoints window. Here you can edit, delete or follow breakpoints.

Alt+C - opens or restores CPU window.

Alt+E - opens or restores list of modules.

Alt+K - opens or restores Call stack window.

Alt+L - opens or restores Log window.

Alt+M - opens or restores Memory window.

Alt+O - opens Options dialog.

Ctrl+P - opens Patches window

Ctrl+T - opens Pause run trace dialog

Alt+X - terminates OllyDbg.

Most windows understand following keyboard commands:

Alt+F3 - closes active window.

Ctrl+F4 - closes active window.

F5 - maximizes active window or restores it to normal size.

F6 - activates next window.

Shift+F6 - activates previous window.

F10 - opens pop-up menu associated with acive window or pane.

LeftArrow - shifts contents 1 character to the left

Ctrl+LeftArrow - shifts contents 1 column to the left

RightArrow - shifts contents 1 character to the right

Ctrl+RightArrow - shifts contents 1 column to the right