

اگر برنامه در دیباگر شما در آدرس مذکور متوقف نشد، از منوی Option > Debugging Options>Events>WinMain(If Location is Known) را تیک بزنید و برنامه را Restart کنید

فب مالا بیاييد به Memory با انتخاب دکمه ی "Me" (در نسخه های دیگر Olly دکمه ی "M") کلیک کنید:

Address	Size	Owner	Section	Contains	Type	Access	Initial	Mapped as
00010000	00010000				Map	RW	RW	
00020000	00010000				Priv	RW	RW	
0012D000	00010000				Priv	RW	Guar	
0012E000	00002000			stack of ma	Priv	RW	Guar	
00130000	00004000				Map	R	R	
00140000	00001000				Priv	RW	RW	
00150000	00067000				Map	R	R	
001C0000	00001000				Priv	RW	RW	
00210000	00007000				Priv	RW	RW	
00310000	00005000				Map	R	R	
003D0000	00003000				Map	R	R	
00400000	00001000			PE header	Image	R	RWE	
00401000	00001000	Tutorial	.text	code	Image	R	RWE	
00402000	00001000	Tutorial	.pdata	imports	Image	R	RWE	
00403000	00001000	Tutorial	.data	data	Image	R	RWE	
00404000	00001000	Tutorial	.rsrc	resources	Image	R	RWE	
004A0000	00003000				Priv	RW	RW	
004B0000	00101000				Map	R	R	
005C0000	00086000				Map	R	R	
01260000	00003000				Priv	RW	RW	
6B460000	00001000	comctl32		PE header	Image	R	RWE	
6B461000	00075000	comctl32	.text	code, imports	Image	R	RWE	
6B4D6000	00003000	comctl32	.data		Image	R	RWE	
6B4D9000	00007000	comctl32	.rsrc	resources	Image	R	RWE	
6B4E0000	00004000	comctl32	.reloc	relocations	Image	R	RWE	
754E0000	00001000	KERNELBA		PE header	Image	R	RWE	
754E1000	00043000	KERNELBA	.text	code, imports	Image	R	RWE	
75524000	00002000	KERNELBA	.data	data	Image	R	RWE	
75526000	00001000	KERNELBA	.rsrc	resources	Image	R	RWE	
75527000	00003000	KERNELBA	.reloc	relocations	Image	R	RWE	
756F0000	00001000	ADVAPI32		PE header	Image	R	RWE	
756F1000	00072000	ADVAPI32	.text	code, imports	Image	R	RWE	
75763000	00004000	ADVAPI32	.data	data	Image	R	RWE	
75767000	00024000	ADVAPI32	.rsrc	resources	Image	R	RWE	
7578B000	00005000	ADVAPI32	.reloc	relocations	Image	R	RWE	
75790000	00001000	MSCIF		PE header	Image	R	RWE	
75791000	00083000	MSCIF	.text	code, imports	Image	R	RWE	
75814000	00002000	MSCIF	.data	data	Image	R	RWE	
75816000	00041000	MSCIF	.rsrc	resources	Image	R	RWE	
75857000	00005000	MSCIF	.reloc	relocations	Image	R	RWE	
75860000	00001000	gdi32		PE header	Image	R	RWE	
75861000	00048000	gdi32	.text	code, imports	Image	R	RWE	
758A9000	00002000	gdi32	.data	data	Image	R	RWE	
758AB000	00001000	gdi32	.rsrc	resources	Image	R	RWE	
758AC000	00002000	gdi32	.reloc	relocations	Image	R	RWE	
758B0000	00001000	kernel32		PE header	Image	R	RWE	
758B1000	000C5000	kernel32	.text	code, imports	Image	R	RWE	
75976000	00001000	kernel32	.data	data	Image	R	RWE	
75977000	00001000	kernel32	.rsrc	resources	Image	R	RWE	
75978000	0000C000	kernel32	.reloc	relocations	Image	R	RWE	
75C50000	00001000	RPCRT4		PE header	Image	R	RWE	
75C51000	00073000	RPCRT4	.text	code, imports	Image	R	RWE	
75CE4000	00003000	RPCRT4	.orpc	code	Image	R	RWE	
75CE7000	00001000	RPCRT4	.data	data	Image	R	RWE	
75CE8000	00004000	RPCRT4	.rsrc	resources	Image	R	RWE	
75CEC000	00005000	RPCRT4	.reloc	relocations	Image	R	RWE	
75E40000	00001000	sechost		PE header	Image	R	RWE	

اگر به قسمت سبز رنگ نگاه کنید، در قسمت آدرس 401000 و در قسمت سایز 1000 با نام Tutorial (کوتاه شده ی اسم برنامه Tutorial3) را داریم و در سکشن "Text". ما "code" را داریم. در قسمت های زیر شما سکشن های دیگری از برنامه میبیند مانند ".rdata" ".data" ".rsrc". با آدرس شروع در حافظه و سایز آن بر مسمب بایت و نام و خصوصیت و نوع بلوک های حافظه، در ستون Contains اطلاعاتی در بایه ی سکشن های مپ شده در حافظه میتوان مشاهده کرد برای مثال:

".rsrc": حاوی دیا لوگ باکس و دکمه و متن ها و عکس های بکار رفته در برنامه قرار دارد

".Text": حاوی کد برنامه است

".rdata": شامل داده ها و ارجاعات برنامه است

".data": شاید تعجب کنید ،این سکشن چیزی در خود ندارد!!

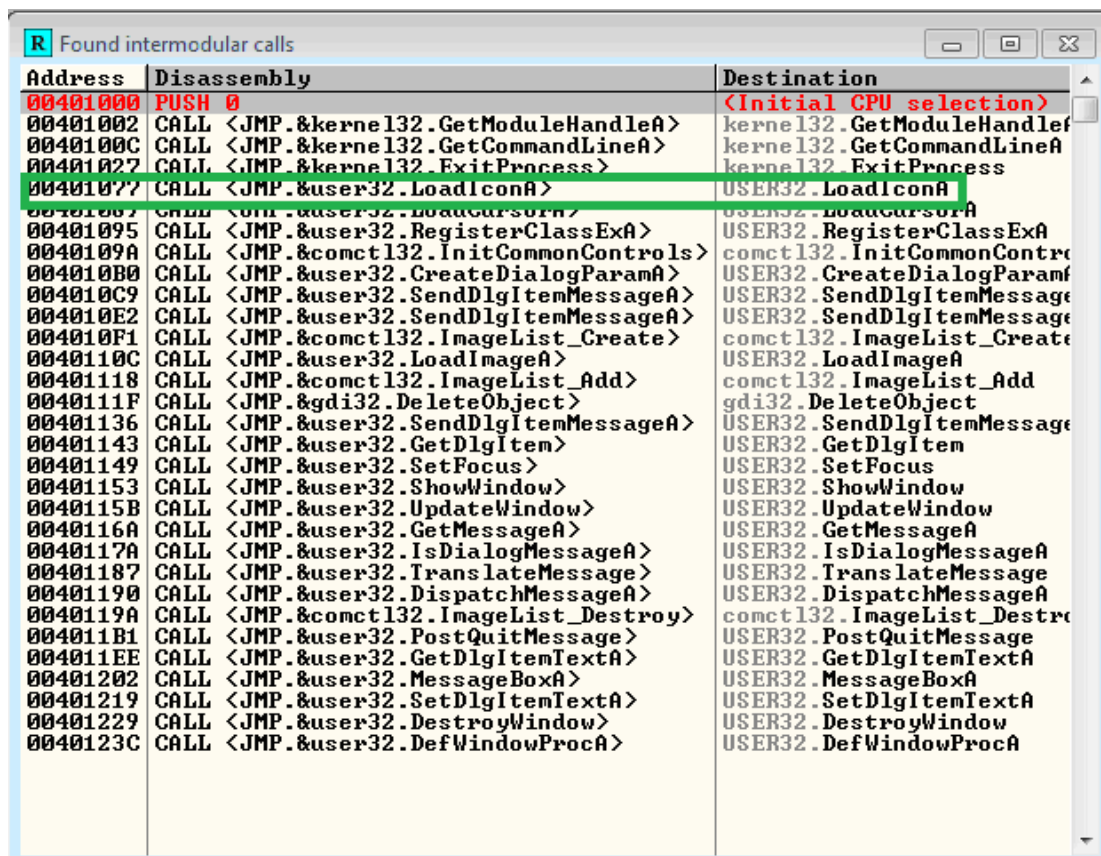
به یاد داشته باشید که اسامی این سکشن ها قابل تغییر است و برنامه نویس هنگام نوشتن برنامه میتواند آنها را تغییر دهد

در صدر این سکشن ها “PE Header” وجود دارد این سکشن بسیار مهم است، در هر آموزش مقداری از مباحث آن را فرا خواهیم گرفت، فعلا همین قدر بدانید که ویندوز لودر برای بارگذاری یک فایل اجرایی در حافظه به این قسمت رجوع میکند و اطلاعاتی را مانند میزان حافظه ی مورد نیاز و... را از همین سکشن میگرد

اگر به تصویر بالا نگاه کنید سکشن های دیگری را هم میبینید مانند: comctl32،

imm32, gdi32, kernel32 این ها Dll هایی هستند که در خود، مجموعه توابعی دارند که برای برنامه ما نیاز هستند و برنامه به آنها رجوع خواهد کرد. کارهای متفاوتی اعم از باز کردن دیالوگ باکس ها، مقایسه رشته ها، سافت پنجره ها، خواندن و نوشتن در فایل ها، و امثال اینها انجام میدهند. دلیل استفاده از این مجموعه توابع این است که اگر قرار نبود آنها را فراخوانی کنیم باید خودمان آنها را مینوشتیم!، که صد البته کاری وقت گیر و بسیار دشوار خواهد بود و قطعا از هر سیستم به سیستم دیگر هم برنامه دچار مشکل میشد.

شاید از خود بپرسید که بطور ویندوز تشخیص میدهد که کدام Dll و با کدام آدرس برای برنامه ی ما نیاز است؟ جواب در همان PE Header است، در آنجا ذکر شده. وقتی ویندوز لودر برنامه را در حافظه بارگذاری میکند ابتدا به سراغ این قسمت رفته و نام Dll ها و توابعی که فعلا مورد نیاز برنامه هست را در فضای حافظه ی مربوط به برنامه لیست میکند به این صورت برنامه به راحتی میتواند آنها را فراخوانی کند. اگر میخواهید ببینید که برنامه ی شما از کدام توابع استفاده میکند ابتدا بروی پنجره ی **disassembly** راست کلیک کنید و سپس **“All Intermodular Calls” -> “Search For”**:

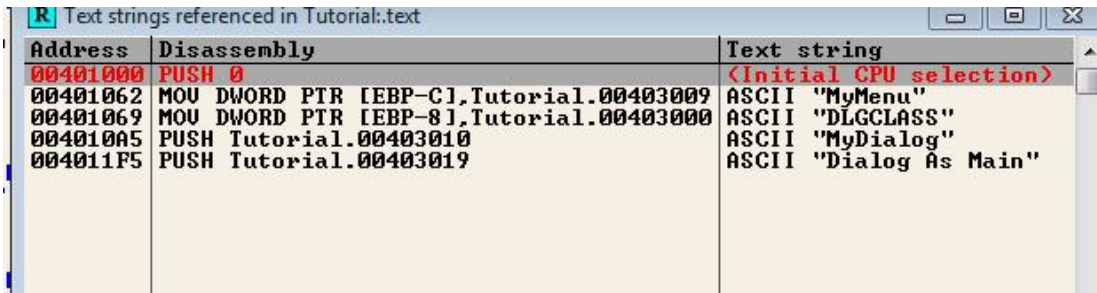


Address	Disassembly	Destination
00401000	PUSH 0	<Initial CPU selection>
00401002	CALL <JMP.&kernel32.GetModuleHandleA>	kernel32.GetModuleHandleA
0040100C	CALL <JMP.&kernel32.GetCommandLineA>	kernel32.GetCommandLineA
00401027	CALL <JMP.&kernel32.ExitProcess>	kernel32.ExitProcess
00401077	CALL <JMP.&user32.LoadIconA>	USER32.LoadIconA
00401087	CALL <JMP.&user32.LoadCursorA>	USER32.LoadCursorA
00401095	CALL <JMP.&user32.RegisterClassExA>	USER32.RegisterClassExA
0040109A	CALL <JMP.&comctl32.InitCommonControls>	comctl32.InitCommonControls
004010B0	CALL <JMP.&user32.CreateDialogParamA>	USER32.CreateDialogParamA
004010C9	CALL <JMP.&user32.SendDlgItemMessageA>	USER32.SendDlgItemMessageA
004010E2	CALL <JMP.&user32.SendDlgItemMessageA>	USER32.SendDlgItemMessageA
004010F1	CALL <JMP.&comctl32.ImageList_Create>	comctl32.ImageList_Create
0040110C	CALL <JMP.&user32.LoadImageA>	USER32.LoadImageA
00401118	CALL <JMP.&comctl32.ImageList_Add>	comctl32.ImageList_Add
0040111F	CALL <JMP.&gdi32.DeleteObject>	gdi32.DeleteObject
00401136	CALL <JMP.&user32.SendDlgItemMessageA>	USER32.SendDlgItemMessageA
00401143	CALL <JMP.&user32.GetDlgItem>	USER32.GetDlgItem
00401149	CALL <JMP.&user32.SetFocus>	USER32.SetFocus
00401153	CALL <JMP.&user32.ShowWindow>	USER32.ShowWindow
0040115B	CALL <JMP.&user32.UpdateWindow>	USER32.UpdateWindow
0040116A	CALL <JMP.&user32.GetMessageA>	USER32.GetMessageA
0040117A	CALL <JMP.&user32.IsDialogMessageA>	USER32.IsDialogMessageA
00401187	CALL <JMP.&user32.TranslateMessage>	USER32.TranslateMessage
00401190	CALL <JMP.&user32.DispatchMessageA>	USER32.DispatchMessageA
0040119A	CALL <JMP.&comctl32.ImageList_Destroy>	comctl32.ImageList_Destroy
004011B1	CALL <JMP.&user32.PostQuitMessage>	USER32.PostQuitMessage
004011EE	CALL <JMP.&user32.GetDlgItemTextA>	USER32.GetDlgItemTextA
00401202	CALL <JMP.&user32.MessageBoxA>	USER32.MessageBoxA
00401219	CALL <JMP.&user32.SetDlgItemTextA>	USER32.SetDlgItemTextA
00401229	CALL <JMP.&user32.DestroyWindow>	USER32.DestroyWindow
0040123C	CALL <JMP.&user32.DefWindowProcA>	USER32.DefWindowProcA

البته این لیست کوچکی است و برای نرم افزارهای بزرگ میتواند بسیار بزرگتر از این مرف ها باشد، در تصویر بالا قسمت سبز رنگ user32.dll را میبینیم که تابع loadIconA فراخوانی شده، وظیفه ی این تابع بارگذاری آیکن بالای کپشن برنامه است.

حال بیایید به جستجوی رشته های موجود در برنامه بپردازیم، این یکی از تکنیک های مهندسی معکوس است که به در رسیدن به هدف کمک میکند، برای مثال پیدا کردن پیام خطا و

بروی پنجره ی **disassembly** راست کلیک کنید و سپس **Strings** "All Referenced Text:" -> "Search For" :



Address	Disassembly	Text string
00401000	PUSH 0	<Initial CPU selection>
00401062	MOV DWORD PTR [EBP-C],Tutorial.00403009	ASCII "MyMenu"
00401069	MOV DWORD PTR [EBP-8],Tutorial.00403000	ASCII "DLGCLASS"
004010A5	PUSH Tutorial.00403010	ASCII "MyDialog"
004011F5	PUSH Tutorial.00403019	ASCII "Dialog As Main"

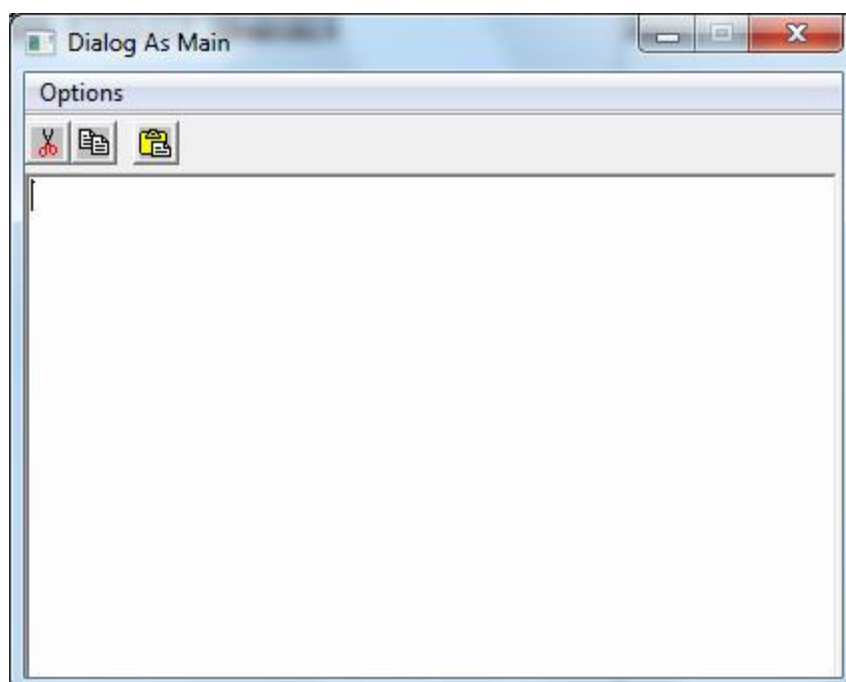
این پنجره تمام رشته های پیدا شده در برنامه را نشان میدهد. توجه داشته باشید این برنامه بسیار کوچک است. برنامه های بزرگ هزاران رشته را ممکن است درون خود داشته باشند. در مواقعی هم این پنجره حاوی رشته های بسیار کمی است با اینکه مثلا برنامه ی بزرگی است در این حالت ممکن است برنامه پک یا مبهم سازی شده باشد. کرکر های تازه کار به شدت متکی بر یافتن رشته ها هستند تا محدودیت های برنامه را از بین ببرند اما همیشه این امر ممکن نیست.

اجرای برنامه

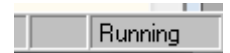
اگر به قسمت پائین سمت راست Olly نگاه کنید عبارت "Paused" را میبینید، این به این معنی است که برنامه متوقف شده و آماده ی ادامه عملیات دیباگینگ است :



با فشردن کلیک F9 میتوان برنامه اجرا کرد. اگر برنامه کامل اجرا شده و آن را ندیدید، ممکن است برنامه پشت پنجره ی Olly اجرا شده باشد، پس دقت کنید) :



فب برنامه اجرا شده به پنجره ی وضعیت نگاه ی بیاندازید :



میبینید که در حالت اجراست این یعنی برنامه درون Olly در حال اجراست. حالا کلید **Ctrl+2** یا  مربوط به راه اندازی مجدد را بفشارید ، بله ، روند دیباگ دوباره انجام شد و دوباره **F9** را بفشارید تا برنامه مجددا اجرا شود.

حال با فشردن کلید **F12 (Debug->Pause)** برنامه را Pause کنید. با انجام این کار برنامه در هر کجای حافظه در حال اجرا باشد متوقف فواید شد. و کنترل به دست دیبگر فواید افتاد ، دوباره کلید **F9** را بفشارید ، تا مجددا برنامه اجرا شود ، اگر هر اتفاقی افتاد برنامه را با **ctrl-F2** مجددا بارگذاری کنید .

ورود به درون برنامه

برنامه کامل اجرا شد ، ولی ما نفهمیدیم که دقیقا چه شد و درون آن چه اتفاقاتی رخ داد!!

بیا یی با هم پله به پله به درون کد های برنامه برویم و سر در بیاوریم. ابتدا برنامه را با **ctrl-F2 (یا debug->restart)** دوباره بارگذاری کنید. همانطور که قبلا هم گفتیم برنامه بروی **EP** توقف فواید کرد. حال کلید **F8** را بفشارید، تا اولین گام خود را درون برنامه بگذاریم. با هر بار فشردن یک دستور العمل اجرا فواید شد و Olly متوقف میشود و منتظر حرکت بعدی شما میماند. و حالت وضعیت پنجره به :



تغییر میکند. و پنجره ی استک در آن زمان به شکل زیر است:

0012FF88	00000000	LpModule = NULL
0012FF8C	75903C45	RETURN to kernel32.75903C45
0012FF90	7FFD9000	
0012FF94	0012FFD4	
0012FF98	771637F5	RETURN to ntdll.771637F5
0012FF9C	7FFD9000	
0012FFA0	762A7446	
0012FFA4	00000000	
0012FFA8	00000000	
0012FFAC	7FFD9000	
0012FFB0	00000000	
0012FFB4	00000000	
0012FFB8	00000000	
0012FFBC	0012FFA0	ASCII "Ft*"
0012FFC0	00000000	
0012FFC4	FFFFFFFF	End of SEH chain
0012FFC8	7711E0ED	SE handler
0012FFCC	012D99EA	
0012FFD0	00000000	
0012FFD4	0012FFEC	
0012FFD8	771637F5	RETURN to ntdll.771637F5 from ntdll.771637F5

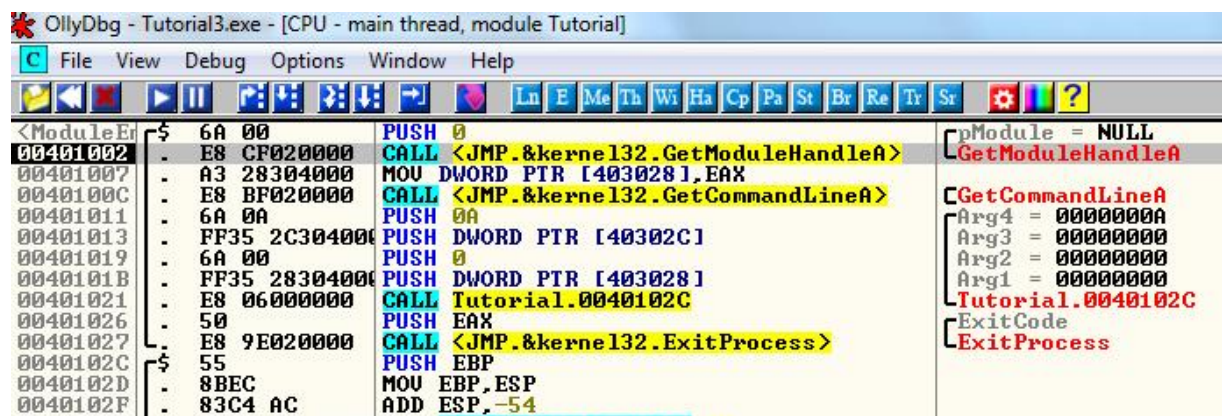
این به آن خاطر است که با اجرای اولین دستور العمل **PUSH 0** ، مقدار 0 درون پشته گذاشته میشود. و به صورت : **pModule = NULL** نمایش داده میشود. در حقیقت این نام دیگر "صفر" است.

همچنین ممکن است متوجه شده باشید که در پنجره ی رجیسترس، ثبات های **ESP, EIP** به رنگ قرمز درآمده باشند.

Registers <FPU>		
EAX	75903C33	kernel32.BaseThreadInitThunk
ECX	00000000	
EDX	00401000	Tutorial.<ModuleEntryPoint>
EBX	7FFD9000	
ESP	0012FF88	
EBP	0012FF94	
ESI	00000000	
EDI	00000000	
EIP	00401002	Tutorial.00401002

این یعنی اجرای دستور العمل قبلی تأثیری بر روی این ثبات ها گذاشته ، عدد درون ثبات **EIP** همیشه اشاره به آدرسی که در حال اجرا هست دارد **401000** و دو مقدار افزایش داشته **401002** که نشان دهنده ی این است که طول دستور ۲ بایتی بوده . و ثبات **ESP** اشاره به آدرس بالای پشته دارد. و به ازای **PUSH** یک وامد افزایش پیدا کرد.

ما اکنون در فط بعدی هستیم ، و بر روی تابع زیر ایستاده ایم.



اگر کمی با دانش برنامه نویسی آشنا باشید، این فراخوانی چنین خواهد بود :

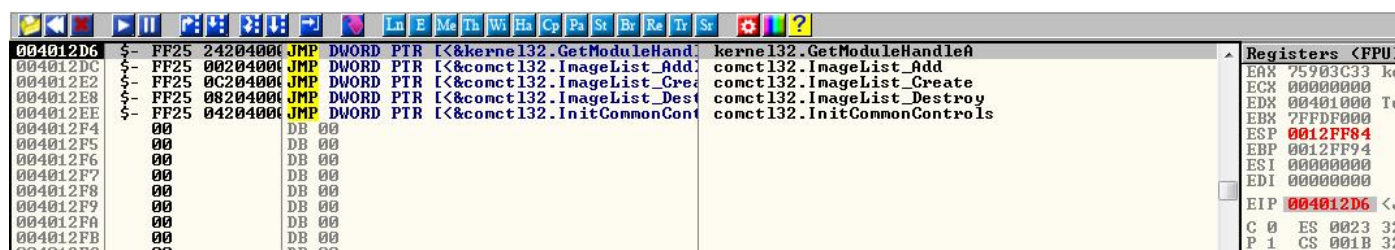
```
int main()
{
    int x = 1;
    call doSomething();
    x = x + 1;
}
```

int x = 1; متغیر **X** را با مقدار **1** ارزش گذاری کرده ایم و پس از اینکه تابع **call doSomething();** اجرا بشود سپس در فط بعد یک وامد به **X** اضافه خواهد شد.

فب در زبان اسمبلی در فط اول ما مقدار ۰ را در استک **push** کردیم . و در بعد تابعی از فایل **Kernel32** به نام **GetModuleHandleA()**: فراخوانی و اجرا خواهد شد، درست مثله مثال

حال دوباره کلید **F8** را بفشارید تا این تابع اجرا شود ، همانطور که میبینید ثبات **EIP** همواره قرمز خواهد ماند و به مقدار **5** وامد افزایش داشت که به معنی طول دستور اجرا شده است. کلید **F8** به معنای **Step-Over** است این بدین معنی است که توابع را یکباره اجرا میکند و نتیجه ی کلی را ثبت میکند. و پشته هم حاوی مقادیر خودش است، و در نهایت در فط بعدی **Olly** متوقف شده است و منتظر مرکت بعدی است.

فب برنامه را مجدداً **(ctrl-F2)** Restart کنید . و کلید **F8** را یک بار بفشارید و وقتی به تابع مورد نظر **GetModuleHandleA()** رسیدید ، کلید **F7** را بفشارید. با فشردنش شما وارد این تابع میشوید:



در این حالت شما درمقیقت به قسمت دیگری از حافظه پرش خواهید کرد گواه این جمله مقدار ثبات **EIP** است،خب برنامه را دوباره (**ctrl-F2**) , **Restart** کنید ،البته با کلید های میان بر میشود راهی را که آماده ایم را برگردیم ولی بگذارید اصولی پیش برویم،بعد از بارگذاری ، **۴** بار پشت سر هم **F8** را بفشارید تا به خط زیر برسید:

<ModuleE	\$	6A 00	PUSH 0	pModule = NULL
00401002	-	E8 CF020000	CALL <JMP.&kernel32.GetModuleHandleA>	GetModuleHandleA
00401007	-	A3 28304000	MOV DWORD PTR [403028],EAX	
0040100C	-	E8 BF020000	CALL <JMP.&kernel32.GetCommandLineA>	CGetCommandLineA
00401011	-	6A 0A	PUSH 0A	Arg4 = 0000000A
00401013	-	FF35 2C304000	PUSH DWORD PTR [40302C]	Arg3 = 00000000
00401019	-	6A 00	PUSH 0	Arg2 = 00000000
0040101B	-	FF35 28304000	PUSH DWORD PTR [403028]	Arg1 = 00400000
00401021	-	E8 06000000	CALL Tutorial.0040102C	Tutorial.0040102C
00401026	-	50	PUSH EAX	ExitCode
00401027	-	E8 9E020000	CALL <JMP.&kernel32.ExitProcess>	ExitProcess
0040102C	\$	55	PUSH EBP	
0040102D	-	8BEC	MOV EBP,ESP	

اگر به تصویر بالا نگاه کنید میبینید که چهار دیتور **PUSH** پشت سرهم وجود دارند با کلید **F8** آنها را اجرا کنید ؛شاید از خود پرسید که این چهار **PUSH** چه معنی دارند ،اینها پارامترهای تابع در خط 401021 هستند.اگر بفوهم مثالی مشابه جهت درک بهتر در زبان برنامه نویسی سطح بالا بزنم به این شکل خواهد بود:

```
int main()
{
int x = 1;
int y = 0;
call doSomething( x, y );
x = x + 1;
}
```

در اینجا ما ۲ متغیر با نام های **X,Y** داریم که درون تابع **call doSomething(x, y);** هم هستند.هنگامی که تابع اجرا میشود مقادیر بازگشتی را درون این متغیر ها قرار میدهد ،پشته یکی از راه های اصلی برای انتقال متغیر های تابع است.و با دستور **POP** این مقادیر قابل دسترسی هستند.اما پشته فقط مقادیر این متغیر ها را نگه نمیدارد بلکه میتواند مقادیر ثبات ها هم نگهداری کند. به هر حال اگر یک بار دیگر **F8** را بفشاریم برنامه اجرا خواهد شد.

Breakpoints

برنامه را با کلید (**ctrl-F2**) ری استارت کنید و با کلید **F2** یا با دبل کلیک بروی ماشین کد ("**6A 0A**" opcodes)بروی آدرس **401011** یک نقطه ی توقف ایجاد کنید:

00401007	-	A3 28304000	MOV DWORD PTR [403028],EAX	GetModuleHandleA
0040100C	-	E8 BF020000	CALL <JMP.&kernel32.GetCommandLineA>	CGetCommandLineA
00401011	-	6A 0A	PUSH 0A	Arg4 = 0000000A
00401013	-	FF35 2C304000	PUSH DWORD PTR [40302C]	Arg3 = 00000000
00401019	-	6A 00	PUSH 0	Arg2 = 00000000
0040101B	-	FF35 28304000	PUSH DWORD PTR [403028]	Arg1 = 00400000
00401021	-	E8 06000000	CALL Tutorial.0040102C	Tutorial.0040102C
00401026	-	50	PUSH EAX	ExitCode
00401027	-	E8 9E020000	CALL <JMP.&kernel32.ExitProcess>	ExitProcess
0040102C	\$	55	PUSH EBP	
0040102D	-	8BEC	MOV EBP,ESP	

درواقع با درج نقطه ی توقف بروی آدرس **401011** دیباگر به محض برخورد با آن متوقف شده،اساسا ما چند نوع نقطه ی توقف داریم

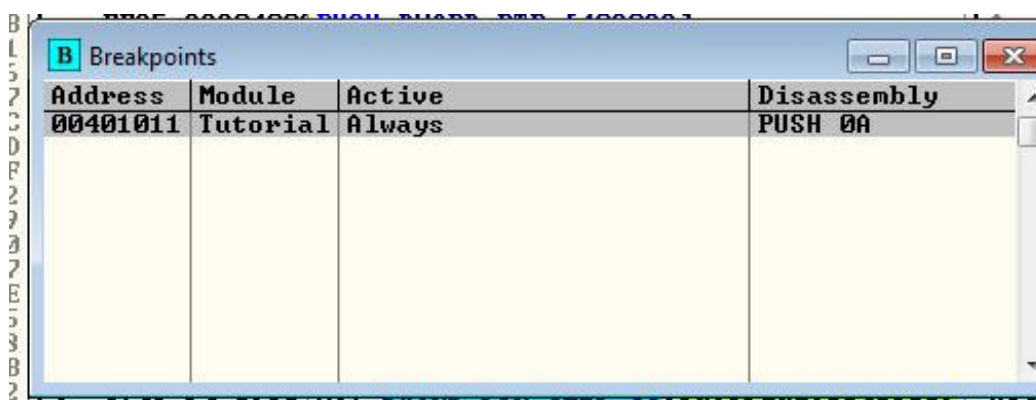
نقاط توقف نرم افزاری :

ساده ترین و پرکاربردترین نقاط توقف هستند. پس از قراردادن یکی از نقاط در آدرس مورد نظر، روند اجرایی برنامه با رسیدن به دستورالعمل متناظر با آن آدرس متوقف شده و کنترل اجرا به **Debugger** منتقل می شود. از این مرحله به بعد کاربر می تواند بررسی ها و یا تغییرات دلفواه خود را انجام داده، به اجرای خط به خط برنامه بپردازد و یا روند اجرای را ادامه دهد. این نقاط توقف با درج دستورالعمل وقفه دیباگ (**INT3**) در دستورالعمل ها ایجاد می شوند. تعداد این گونه نقاط توقف محدودیت خاصی نداشته و می توانید هر تعداد از آنها را ایجاد کرده و از آنها استفاده کنید

OllyDbg نقاط توقف تعیین شده در هنگام مراحل دیباگ را در فایل های UDD ذخیره می کند. این فایل ها برای هر فایل باز شده به وسیله این نرم افزار به طور جداگانه ایجاد شده (بجز فایل های ثابت) و مورد استفاده قرار می گیرد. در صورتی که فایل مورد نظر دوباره توسط **OllyDbg** بارگذاری شود، این نقاط توقف به طور خودکار دوباره اعمال می گردند .

ساده ترین روش ایجاد این نقاط توقف، انتخاب آدرس مورد نظر در قسمت **Disassembler** و فشردن کلید **F2** است **Double Click** . کردن در قسمت **Hex Dump** از آدرس مورد نظر نیز اثر مشابهی خواهد داشت .
به منظور غیرفعال کردن نقاط توقف ایجاد شده می توانید عملیات فوق را تکرار کنید .

به منظور مشاهده و مدیریت نقاط توقف ایجاد شده می توانید از پنجره Breakpoints استفاده کنید. این پنجره از طریق گزینه **Breakpoints** از منوی **View** قابل دسترسی است. همان طور که در شکل زیر مشاهده می کنید در این پنجره لیست از نقاط توقف به همراه توضیحات کوتاهی در مورد هر یک نمایش داده شده و امکان ایجاد تغییرات و غیر فعال کردن آنها به کاربر داده شده است.

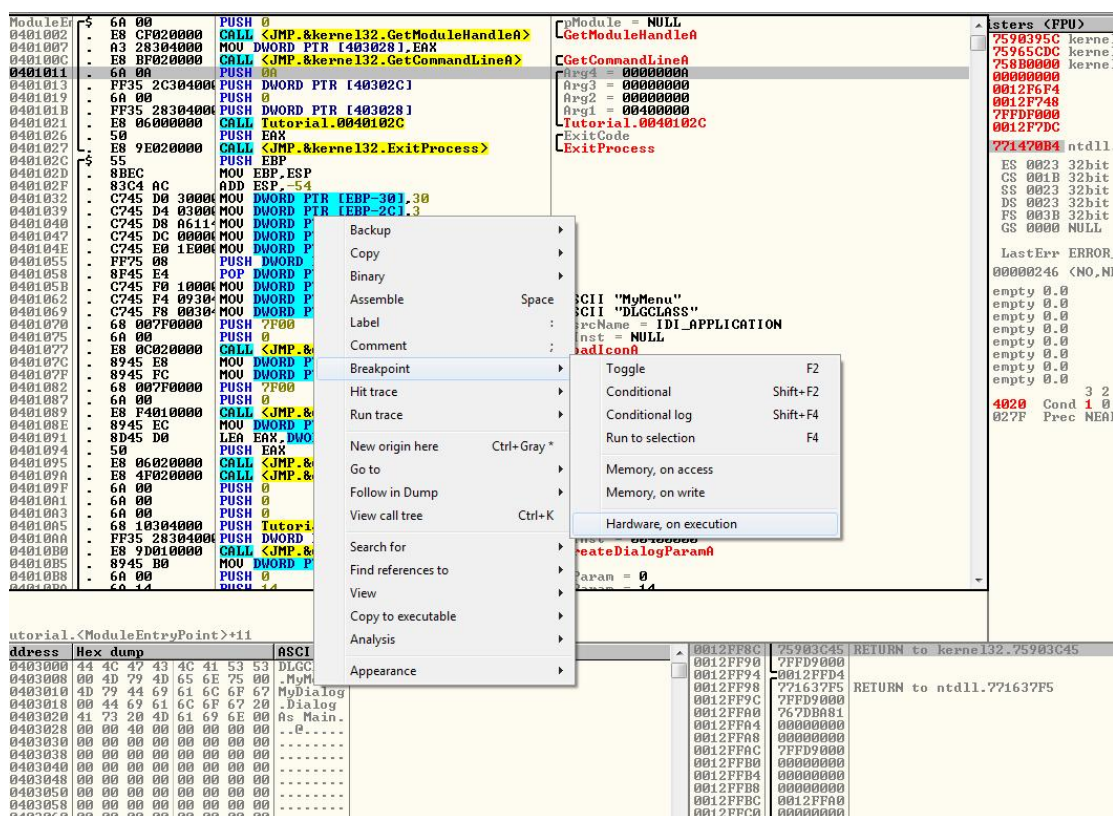


نقاط توقف سخت افزاری

این نقاط توقف مستقیماً توسط پردازنده های **x86** پشتیبانی می شوند و می توانند عملکردی مشابه نقاط توقف معمولی (**INT3**) و یا نقاط توقف حافظه ای داشته باشند. معمولاً از آنها به منظور ردیابی عملیات خواندن، نوشتن و یا اجرای محدوده خاصی از فضای حافظه برنامه استفاده می شود. به علت محدودیت های سخت افزاری، تعداد کل نقاط توقف سخت افزاری مورد استفاده به تعداد **۴** عدد و اندازه آنها به محدوده های **یک بایتی (Byte)**، **دو بایتی (Word)** و **چهار بایتی (Dword)** محدود می شود. با توجه به پشتیبانی سخت افزار، استفاده از آنها باعث کند شدن سرعت اجرای برنامه نمی گردد.

توجه داشته باشید که این گونه نقاط توقف تنها توسط سیستم های عامل Windows 2000, XP, 2003 و بالاتر پشتیبانی می شوند و در نسخه های قدیمی تر امکان استفاده از آنها وجود ندارد.

برای ایجاد این گونه نقاط توقف در قسمت کد، ابتدا آدرس دستورالعمل مورد نظر را در قسمت **Disassembler** انتخاب کرده و سپس همانند شکل زیر از قسمت **Breakpoints** از منوی **Disassembler**، گزینه **On Execution** و **Hardware** را انتخاب کنید:



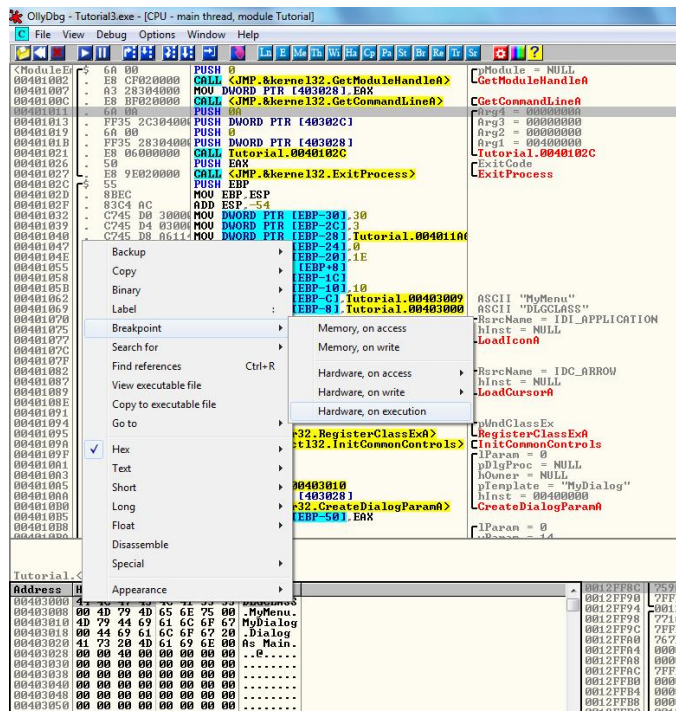
به منظور ایجاد توقف برای دسترسی ها به حافظه و محدوده های **data**، ابتدا آدرس مورد نظر را در قسمت **Dump** انتخاب کرده و سپس از قسمت **Breakpoint** از منوی **Dump** نقطه توقف دلفواه را انتخاب کنید. همان طور که در شکل زیر مشاهده می کنید انواع مختلف این گونه نقاط توقف عبارتند از: **On access**، **On write**، **On execute**.

On write: در هنگام نوشتن داده در محدوده مورد نظر فعال می شود.

On execute: در قسمت های کد و در هنگام اجرای دستورالعمل در محدوده مورد نظر فعال می شود.

On access: با هر گونه دسترسی به محدوده مورد نظر فعال می شود که می تواند خواندن، نوشتن و یا اجرا باشد.

همان طور که ذکر شد این گونه از نقاط توقف می توانند محدوده های ۱ و ۲ و ۴ بایتی را تحت پوشش قرار دهند که توسط منوی **Dump** قابل انتخاب است.



نقاط توقف برای دسترسی ها به حافظه

از این گونه توقف معمولاً به منظور ردیابی عملیات خواندن و نوشتن در محدوده خاصی از فضای حافظه برنامه استفاده می شود. **OllyDbg** اجازه تعریف یک نمونه از این نقاط توقف را در کل برنامه می دهد. به علت محدودیت های سفت افزاری اندازه این محدود ۴۰۹۶ بایت و ثابت است. به علت مدیریت نرم افزاری در نظر گرفته شده برای این نقاط توقف، معمولاً استفاده از آنها در محدوده هایی با دسترسی زیاد می تواند کار آیی سیستم و سرعت اجرا را به شدت تحت تاثیر قرار دهد. استفاده از این گونه نقاط توقف در سیستم های عامل Windows 95/98 با اختلالات زیادی همراه است. عدم توانایی کاربر در تعیین محدوده مورد نظر نیز معمولاً با مشکلات زیادی همراه بوده و باعث توقف های غیردلفواه در محدوده ۴۰۹۶ بایتی می گردد. در قسمت بعد نقاط توقف سفت افزاری معرفی خواهد شد که می توانند جایگزینی دقیق و مناسب برای این گونه نقاط توقف باشند.

به منظور ایجاد این نقاط توقف ابتدا آدرس شروع مورد نظر را در قسمت Disassembler و یا Dump انتخاب کرده و سپس از قسمت Breakpoints از منوی صفحه، گزینه Memory on Write و یا Memory on Access را انتخاب کنید.

نوع دیگری از اینگونه نقاط توقف توانایی پشتیبانی از یک بلوک از حافظه برنامه را دارند. این بلوک ها معمولاً Section های فایل های اجرایی و یا dll های بارگذاری شده به فضای حافظه برنامه هستند. این نوع از نقاط توقف در ردیابی دسترسی ها به منابع فایل های اجرایی نقش کلیدی را ایفا می کنند.

به منظور ایجاد این نوع از نقاط توقف ابتدا گزینه Memory را از منوی View انتخاب کرده و یا از کلیدهای Alt+M استفاده کنید. با این عمل پنجره Memory map همانند شکل زیر ظاهر شده و لیستی از بلوک های موجود در حافظه را به همراه مشخصات هر یک به نمایش می گذارد. در ادامه جزئیات این پنجره و نحوه عملکرد آن را مورد بررسی قرار خواهیم داد.

Address	Disassembly	Comment	Access	Breakpoint	Module
00401000	PUSH 0				C:\Module = NULL
00401002	CALL <JMP.<kernel32.GetModuleHandleA>				C:\Module = NULL
00401007	MOV DUWORD PTR [403028], EAX				C:\Module = NULL
0040100C	CALL <JMP.<kernel32.GetCommandLineA>				C:\Module = NULL
00401011	PUSH 0				C:\Module = NULL
00401013	PUSH DUWORD PTR [40302C]				C:\Module = NULL
00401019	PUSH 0				C:\Module = NULL
0040101B	PUSH DUWORD PTR [403028]				C:\Module = NULL
00401021	CALL Tutorial.0040102C				C:\Module = NULL
00401026	PUSH EAX				C:\Module = NULL
00401027	CALL <JMP.<kernel32.ExitProcess>				C:\Module = NULL
0040102C	PUSH EBP				C:\Module = NULL
0040102D	8BEC				C:\Module = NULL
0040102F	ADD ESP, -4				C:\Module = NULL
00401032	MOV DUWORD PTR [EBP-30], 30				C:\Module = NULL
00401039	C745 D8 0300				C:\Module = NULL
00401040	C745 D8 0611				C:\Module = NULL
00401047	Backup				C:\Module = NULL
00401055	Copy				C:\Module = NULL
0040105B	Binary				C:\Module = NULL
00401062	Label				C:\Module = NULL
00401070	Breakpoint				C:\Module = NULL
00401075	Search for				C:\Module = NULL
0040107F	Find references				C:\Module = NULL
00401082	View executable file				C:\Module = NULL
00401087	Copy to executable file				C:\Module = NULL
00401091	Go to				C:\Module = NULL
00401095	Hex				C:\Module = NULL
0040109A	Test				C:\Module = NULL
004010A1	Short				C:\Module = NULL
004010A5	Long				C:\Module = NULL
004010B0	Float				C:\Module = NULL
004010B5	Disassemble				C:\Module = NULL
004010B8	Special				C:\Module = NULL
004010C0	Appearance				C:\Module = NULL

از لیست نمایش داده شده بلوک مورد نظر را انتخاب کرده و از گزینه های Set memory breakpoint on access یا Set memory write breakpoint on استفاده کنید. توجه داشته باشید که غیر از محدودیت اندازه نامیه، محدودیت های این نوع دقیقاً همانند نوع قبل است.

نقاط توقف یکبار مصرف برای بلوک های حافظه

عملکرد این نوع از توقف دقیقاً مشابه نوع قبل است با این تفاوت که پس از یک بار فعال شدن به طور خودکار از بین می روند. همچنین این نوع محدودیتی برای تعداد ندارند و می توانید بر خلاف نوع قبلی به هر تعداد از آنها استفاده کنید. توجه داشته باشید که این نوع از نقاط توقف تنها در ویندوزهای خانواده (NT, 2000, XP, 2003) قابل تعریف و استفاده هستند. معمولاً از آنها به منظور کنترل فراخوانی ها و یا بازگشت های انجام شده به dll های مورد استفاده برنامه استفاده می شود.

به منظور ایجاد این نوع از نقاط توقف ابتدا گزینه Memory را از منوی View انتخاب کرده و پس از انتخاب بلوک مورد نظر از کلید F2 استفاده کنید.

به منظور حذف این نقاط توقف عملیات مذکور را تکرار کنید.

استفاده از پنجره ی دامپ

اگر هر دستور العملی اعم از ثبات ها و هر آیتمی در پنجره ی Assembly دارای رفرنس و یا مقداری باشد میتوان در این پنجره آن را مشاهده کرد. با کلیک بروی آیتم مورد نظر و انتخاب **Follow in Dump** به محل مورد نظر رفته، همچنین در همین قسمت قابلیت پرش و جستجوی آدرس وجود دارد

برنامه را درون Olly بارگذاری کنید و با کلید F8 به آدرس 471021 بیایید، ببینید که این تابع به آدرس 40102C اشاره دارد:

The screenshot shows the OllyDbg interface with the following details:

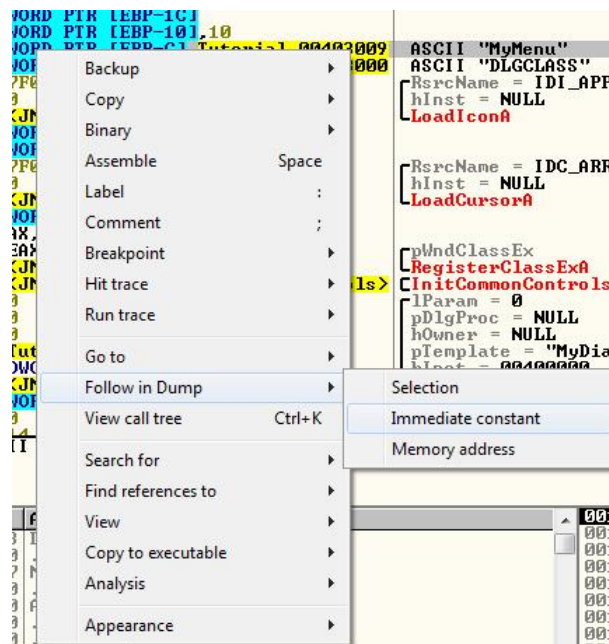
- Assembly Window:**
 - Address 00401002: CALL <JMP.&kernel32.GetModuleHandleA>
 - Address 00401007: MOV DWORD PTR [403028], EAX
 - Address 0040100C: CALL <JMP.&kernel32.GetCommandLineA>
 - Address 00401011: PUSH 0A
 - Address 00401013: PUSH DWORD PTR [40302C]
 - Address 00401019: PUSH 0
 - Address 0040101B: PUSH DWORD PTR [403028]
 - Address 00401021: CALL Tutorial.0040102C (highlighted)
 - Address 00401026: PUSH EAX
 - Address 00401027: CALL <JMP.&kernel32.ExitProcess>
 - Address 0040102C: PUSH EBP
 - Address 0040102D: MOV EBP, ESP
 - Address 0040102F: ADD ESP, -54
 - Address 00401032: MOV DWORD PTR [EBP-30], 30
 - Address 00401039: MOV DWORD PTR [EBP-2C], 3
 - Address 00401040: MOV DWORD PTR [EBP-28], Tutorial.004011A0
 - Address 00401047: MOV DWORD PTR [EBP-24], 0
 - Address 0040104E: MOV DWORD PTR [EBP-20], 1E
- CPU Registers Window:**
 - EAX: 00000000
 - ECX: 00000000
 - EDX: 00000000
 - EBX: 00000000
 - ESP: 0040102C
 - EBP: 0040102C
 - ESI: 00000000
 - EDI: 00000000
 - EIP: 00401021
- Memory Dump Window:**
 - pModule = NULL
 - GetModuleHandleA
 - GetCommandLineA
 - Arg4 = 00000000
 - Arg3 = 00000000
 - Arg2 = 00000000
 - Arg1 = 00400000
 - Tutorial.0040102C
 - ExitCode
 - ExitProcess

با فشردن کلید F7 به درون آن میرویم، و دوباره با کلید F8 ادامه میدهیم تا به خط 401062 برسیم :

00401039	. C745 D4 0300	MOV	DWORD PTR [EBP-2C], 3		
00401040	. C745 D8 A611	MOV	DWORD PTR [EBP-28], Tutorial.004011A6		
00401047	. C745 DC 0000	MOV	DWORD PTR [EBP-24], 0		
0040104E	. C745 E0 1E00	MOV	DWORD PTR [EBP-20], 1E		
00401055	. FF75 08	PUSH	DWORD PTR [EBP+8]		
00401058	. 8F45 E4	POP	DWORD PTR [EBP-1C]		
0040105B	. C745 F0 1000	MOV	DWORD PTR [EBP-10], 10		
00401062	. C745 F4 0930	MOV	DWORD PTR [EBP-C], Tutorial.00403009	ASCII "MyMenu"	
00401069	. C745 F8 0030	MOV	DWORD PTR [EBP-8], Tutorial.00403000	ASCII "DLGCLASS"	
00401070	. 68 007F0000	PUSH	7F00	RsrcName = IDI_APPLICATION	
00401075	. 6A 00	PUSH	0	hInst = NULL	
00401077	. E8 0C020000	CALL	<JMP.&user32.LoadIconA>	LoadIconA	
0040107C	. 8945 E8	MOV	DWORD PTR [EBP-18], EAX		
0040107F	. 8945 FC	MOV	DWORD PTR [EBP-4], EAX		

همچنین شما میتوانید با گذاشتن نقطه ی توقف و فشردن کلید F9 مستقیم به این خط بیاید

به خط MOV [LOCAL.3], FirstPro.00403009 نگاه بیندازید، این خط مقداری را از آدرس 403009 رادرون استک انتقال میدهد. اگر به ستون Comment نگاه کنید رشته ی ASCII حاوی MyMenu را خواهید دید. با این حال روی دستورالعمل راست کلیک کنید
:“Follow in Dump”



بروی “Immediate Constant” کلیک کنید، این گزینه تمامی مقادیر در آدرس دستورالعمل را نشان میدهد
اگر شما گزینه ی Section را انتخاب کنید مقادیر آدرس دستورالعمل جاری یعنی 401062 نشان داده خواهد شد
و اگر گزینه ی Memory Address را انتخاب کنید ممتوای پشته را که محل نگهداری متغییر های را نشان میدهد
به هر حال بروی “Immediate Constant” کلیک میکنیم :

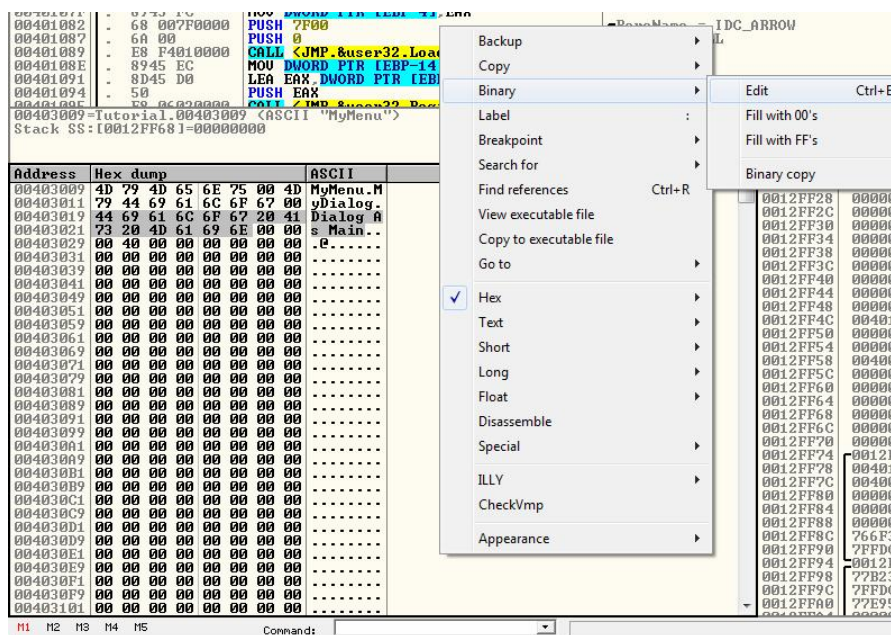
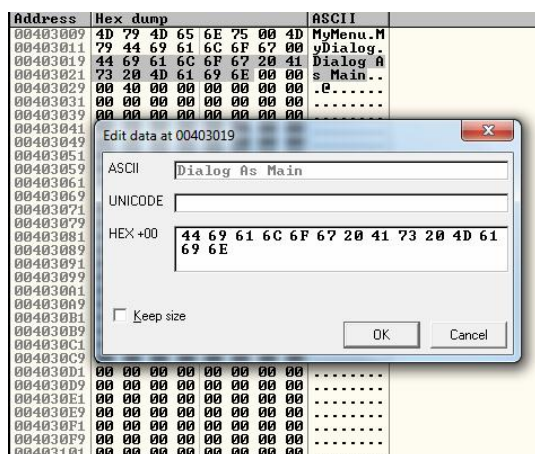
004010D0	. 0H 00	push	0		
004010D0	. 60 14	push	14		
00403009=Tutorial.00403009 <ASCII "MyMenu">					
Stack SS:[0012FF69]=00000000					
Address	Hex	dump	ASCII		
00403009	4D	79 4D 65 6E 75 00 4D	MyMenu.M		
00403011	79	44 69 61 6C 6F 67 00	yDialog.		
00403019	44	69 61 6C 6F 67 20 41	Dialog A		
00403021	73	20 4D 61 69 6E 00 00	s Main..		
00403029	00	40 00 00 00 00 00 00	.e.....		
00403031	00	00 00 00 00 00 00 00			
00403039	00	00 00 00 00 00 00 00			
00403041	00	00 00 00 00 00 00 00			
00403049	00	00 00 00 00 00 00 00			
00403051	00	00 00 00 00 00 00 00			
00403059	00	00 00 00 00 00 00 00			
00403061	00	00 00 00 00 00 00 00			
00403069	00	00 00 00 00 00 00 00			
00403071	00	00 00 00 00 00 00 00			
00403079	00	00 00 00 00 00 00 00			
00403081	00	00 00 00 00 00 00 00			

همانطور که میبینید پنجره ی دامپ از آدرس 403009 شروع شده.

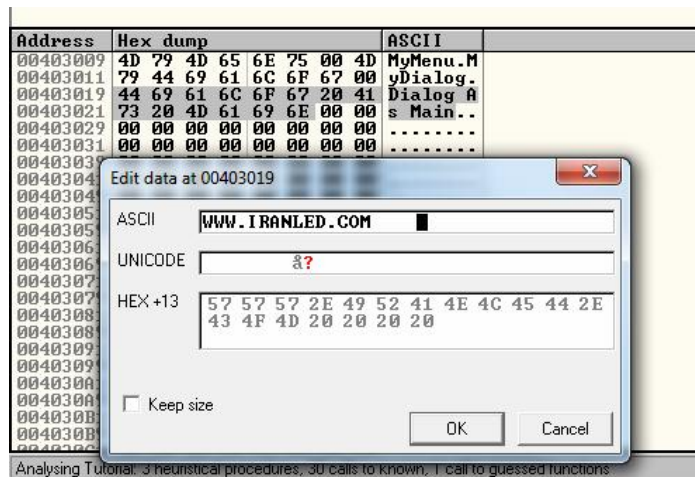
بازی:

به قسمت پایانی این آموزش رسیدیم و قصد داریم کمی تفریح کنیم به این شکل که میفواهیم عبارت "Dialog As Main" را به دلفواه تغییر دهیم .

برای شروع بروی کد اسکی " D " را انتخاب کنید ، همانطور که میبینید کد هگزای این کاراکتر 44 است متن مورد نظر را انتخاب میکنیم و راست کلیک کرده Binary>Edit را انتخاب کرده :

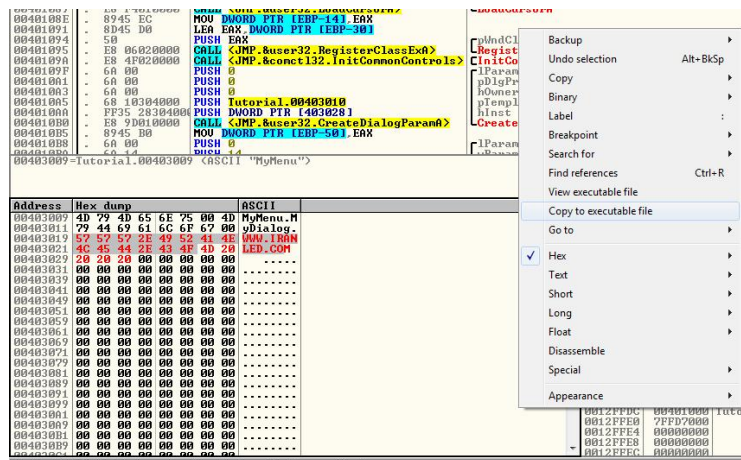


در قسمت ASCII متن دلفواه را نوشته و کلید Ok را بفشارید:

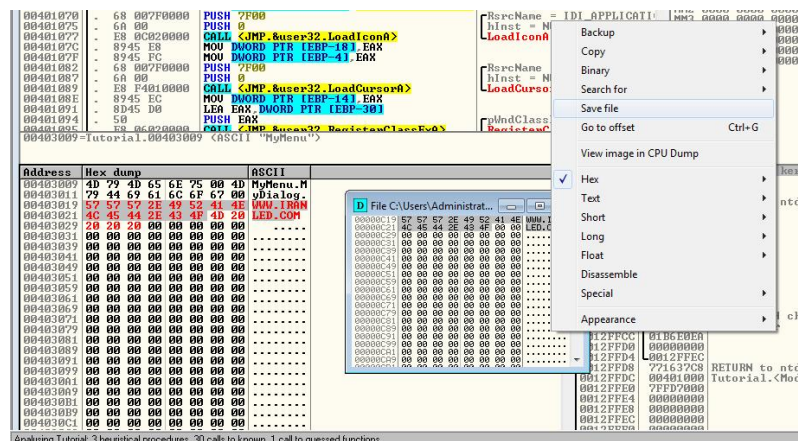


Analysing Tutorial: 3 heuristical procedures, 30 calls to known, 1 call to guessed functions

سپس راست کلیک کرده :



سپس فایل نهایی را ذخیره کنید :



Analysing Tutorial: 3 heuristical procedures, 30 calls to known, 1 call to guessed functions

