آموزش های R4NDOM -فارسی

Tutorial #4: Using Olly, Part 2

این قسمت درواقع ادامه ی قسمت قبل است و ما با توجه به آموخته های قبلی سعی خواهیم کرد آن را گسترش دهیم

DLLs

فایل های DLL یک فرمت استاندارد برای ویندوز مایکروسافت بوده و مخفف عبارت Dynamic-Link Library است. فایل‌های lib و ocx و drv هم تقریباً مشابه DLL هستند. یک فایل DLL شامل توابع، کدها، منابع (تصویر، آیکون و ...) و داده‌هایی است که به توسعه دهندگان و برنامه نویسان این امکان را فراهم می‌کند که به آن‌ها لینک نموده و از توابعشان در برنامه‌های خود استفاده کنند. یکی از خواص مهم فایل‌های کتابخانه‌ای DLL این است که در یک زمان چندین برنامه می‌توانند از آن‌ها استفاده کنند درحالی که کدهایشان در یک مکان ثابت قرار داشته و نیازی به گرفتن فضای بیشتر برای هر برنامه نیست.از طرفی فرض کنید درحال نوشتن برنامه ایی بزرگ هستیم و مدام باید یک کار تکراری مثه تبدیل رشته را انجام دهیم ،بدیهی است که این کار خسته کننده است،اما با به کار گیری Dll فقط آن تابع را فراخوانی کنیم.ویندوز کالکشنی عظیم از توابع را دارا است که هر کدام کارهای بخصوصی انجام میدهند.به طور کلی اگر برنامه ی شما از توابع استفاده خواهد کرد شما دو انتخاب خواهید داشت :

۱- شما کد مروط به هر عملیات را باید شخصا بنویسید و مدام در حال Copy/Past در طول کد نویسی باشید ؛این واقعا هم خسته کننده است و هم حجم کلی برنامه را افزایش میدهد و مستند سازی آن هم مشکل خواهد شد

۲- شما میتوانید توابع را درون یک Dll قرار داده و موقع نیاز آنها را به سادگی فراخوانی کنید(در هرزبان برنامه نویسی)

چرا این فایل‌ها را کتابخانه نامیده اند؟

آشپزی و در هر یک از این کتاب‌های آشپزی یک دستور پخت بگذارید مثالی بزنم: یک کتابخانه عمومی را فرض کنید که پر از کتاب‌های نیز این چنین هستند؛ یعنی پر از توابع DLL افراد به کتابخانه سر زده و از آن کتاب‌ها استفاده می‌کنند. فایل‌های نوشته شده است و می‌دهند و برنامه‌ها می‌توانند از آن‌ها استفاده کنند مختلف که هر یک کار خاصی را انجام.

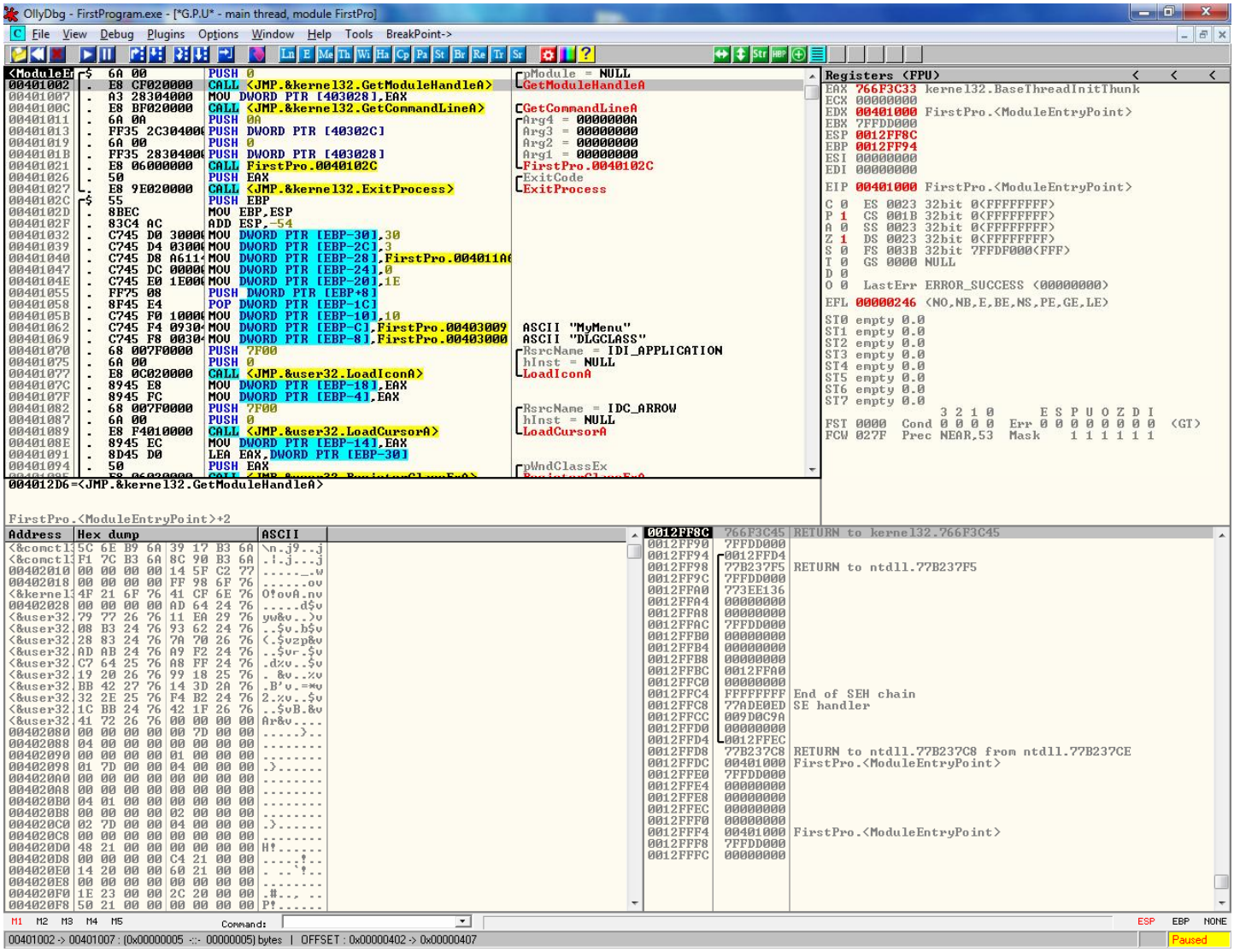
Dll ها چگونه استفاده میشوند

حال که با Dll آشنا شدیم ،بگذارید ببینیم چگونه مورد استفاده قرار میگیرد،وقتی که شما برنامه ی تان را اجرا میکنید،ویندوز لودر سکشن خاصی در PE هدر را چک میکند (یادتان هست؟) چه چیزی را؟ در واقع Dll ها و توابع مورد استفاده در آنها و همچنین آدرس آنها را ،وقتی که برنامه در آدرس حافظه لود شد.توابع لازم و ضروری را در فضای حافظه ی مختص برنامه ی شما با آدرس صحیح ،هرکجا که برنامه از توابع استفاده کردهInject میکند.

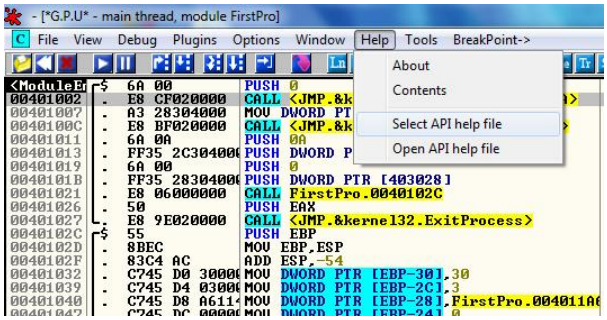
مثلا تابع StrToUpper واقع در Kernel32.dll را در نظر بگیرد ،کار این تابع نمایش رشته ها با حروف بزرگ است ،حال اگر برنامه شما از این تابع استفاده کرده باشد به این شکلCall StrToUpper،ویندوز لودر ابتدا Kernel32.dll را وبعد آدرس تابع StrToUpper را پیدا کرده و آدرس مورد نظر را در فایل ما Inject میکند. و سپس به برنامه باز میگردد.

بیایید تمام توضیحات بالا را به صورت عملی ببینیم در ضمیمه ی این آموزش فایلی با نام "FirstProgram.exe " موجود است ،آن را در OllyDBGبارگذاری کنید.OllyDBG در خط اول متوقف میشود که به آن Entry Point یا به اختصار EP گفته میشود.EPمحلی است که ویندوز لودر بعد از اتمام کار خود به آن قسمت پرش میکند و کنترل به دست برنامه جهت اجرا می افتد.

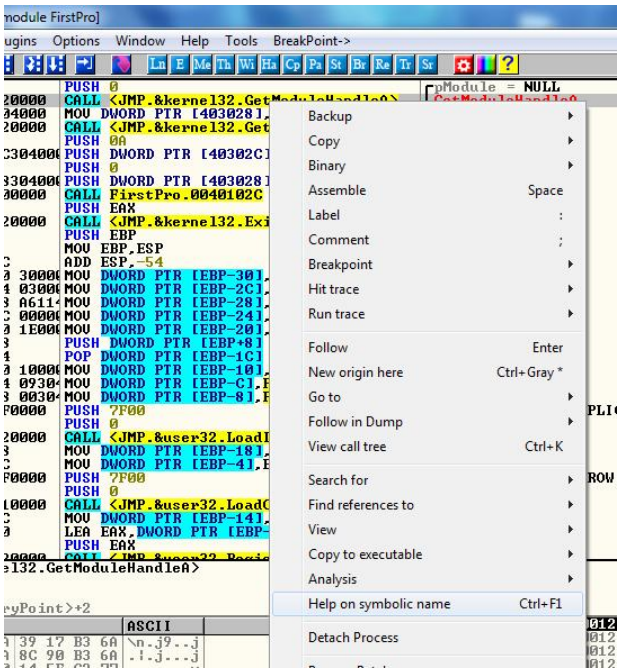
اگر به خط دوم نگاه کنید میبینید که تابعی با نام kernel32.GetModuleHandleA فراخوانی شده:



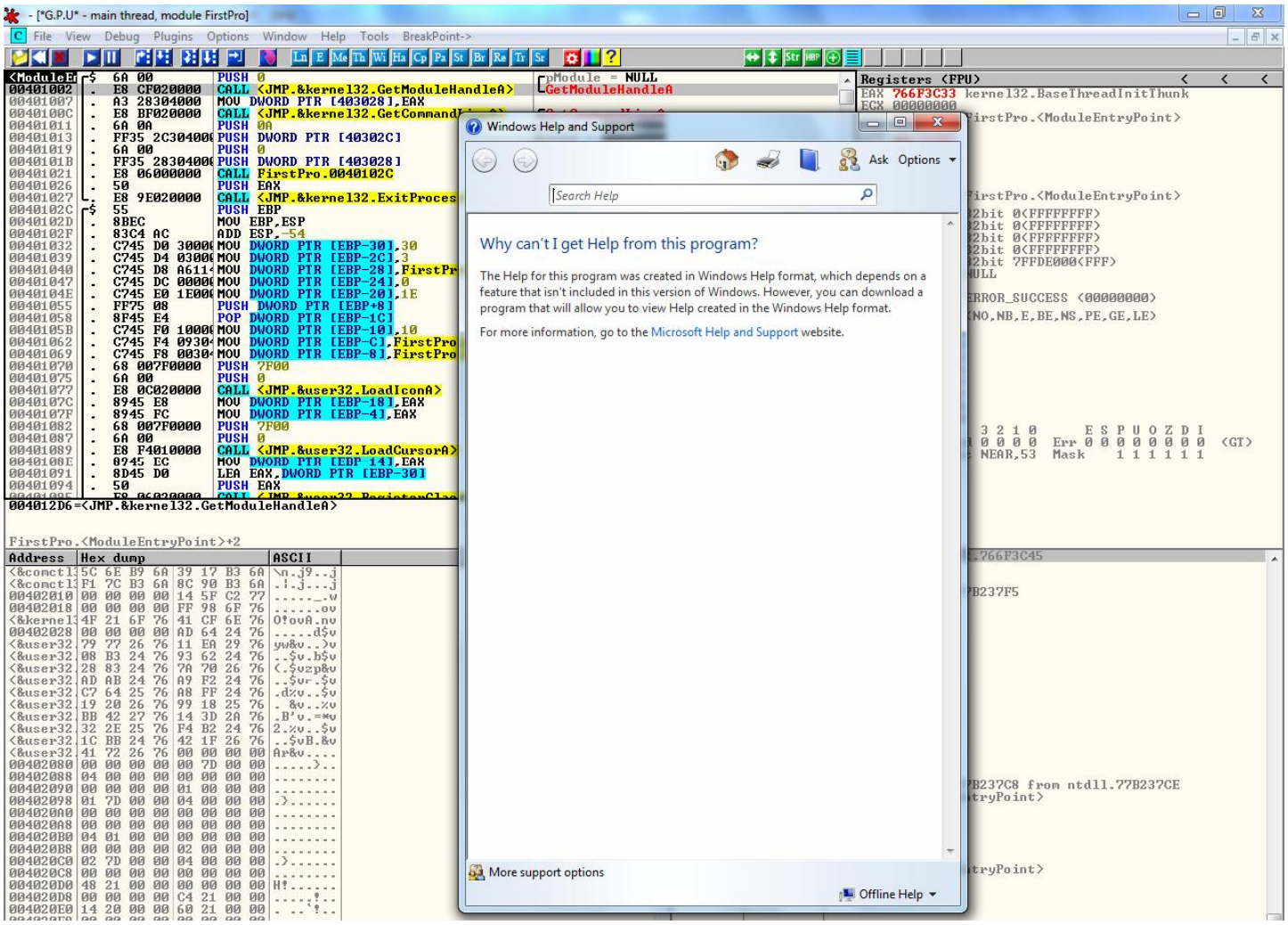
اول ازهمه بگذارید ببینیم این تابع چکاری انجام میدهد، در این آموزش فایل WIN32.HLP ضمیمه شده ،این فایل قابلیت نمایش نحوه ی عملکرد تمامی توابع ویندوز را که تا کنون از طرف مایکروسافت معرفی و سند دار هستند را نشان میدهد،برای اینکار مانند تصویر زیر عمل کنید و فایل WIN32.HLP را انتخاب کنید:



بعد از انتخاب اگر بروی هر تابع راست کلیک کنید (آیتم Help on symbolic name به منو اضافه شده)



با انتخاب آن اطلاعات مربوط به تابع نمایش داده میشود،اگر از ویندوز ۷ به بالا استفاده میکنید ممکن است با صفحه ی :



رو برو شوید برای حل این مشکل به آدرس اینترنتی زیر بروید :

https://support.microsoft.com/en-us/kb/917607

با توجه به نسخه ی سیستم عامل مورد استفاده فایل مذکور را دانلود کرده و نصب کنید:

When this problem occurs, you receive one of the following error messages.

Expand to see the full error message

To resolve this problem, download and install the Windows Help program (WinHlp32.exe) for your version of Windows.

- WinHlp32.exe for Windows 8.1
- WinHlp32.exe for Windows 8
- WinHlp32.exe for Windows 7
- WinHlp32.exe for Windows Server 2008
- WinHlp32.exe for Windows Server 2008 R2
- WinHlp32.exe for Windows Vista

Validation Required

For more information about the validation process click here

File Name:	Size:
Windows6.1-KB917607-x64.msu	702 KB
Windows6.1-KB917607-x86.msu	688 KB

Continue

Windows Help program (WinHlp32.exe) for Windows 7

Select Language: English

Download

808x187 25kb JPEG

نکته:لازم به ذکر است تمامی فایل های HLP از جمله Helpمربوط به Olly یا هر HLP دیگر اجرا خواهند شد،بعد از نصب ollydbg را ریستارت کنید.

و پس از نصب داریم:

همانطور که میبینید این تابع هندل (شناسه)را از ویندوز میگیرد. هر آبجکت یک شناسه ی دسترسی داردو برای ویندوز بسیار مهم هست ،اساسا با همین شناسه است که ویندوز میفهمد شما به کدام آبجکت اشاره میکنید

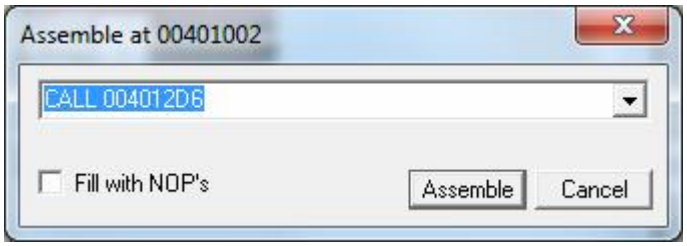
پنجره ی HLP را ببندید، و به فراخوانی (خط دوم) نگاه کنید که به کدام آدرس اشاره دارد،که به چند روش قابل فهمیدن است :

۱-بروی خط مذکور Enter کرده

۲-از قسمت Info قابل مشاهده است

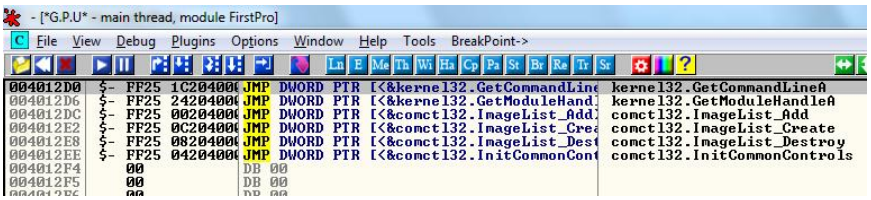
۳-با فشرن دکمه ی Space

۴-با فشردن کلید ترکیبی Ctrl+G و وارد کردن آدرس



نکته: کلید Space دوکاره است ،هم برای ویرایش کد و هم برای دریافت آدرس فراخوانی ها

بسیار خوب ، با فشردن کلید Enter بروی خط دوم "CALL GetModuleHandleA" به آدرس فراخوانی میرویم :



جدول آدرس پرش ها

همانطور که در تصویر بالا میبینید این یک پرش به سمت تابع GetModuleHandleA واقع در kernel32.dll است.با فشردن Enter بروی آن به ابتدای این تابع ارجاع داده خواهیم شد.چیزی که درباره Dll ها باید بدانید این است که همیشه در همان آدرس مشخص شده که در ImageBaseهست ، لود نمیشوند!!ویندوز لودر مسئول بارگذاری و همچنین وفراهم کردن تمام توابع و dll های مورد نیاز برنامه است،همچنین میتواند آدرس لود dll ها و حتی خوده برنامه را تغییر دهد!!.فرض کنید که dll ایی داریم که میخواهد در آدرس 80000000 بارگذاری شود و از قضا dll دیگری داریم که میخواهد در همان آدرس لود شود!!تکلیف چیست؟از آنجایی که دو dll نمیتوانند در یک آدرس مشابه بارگذاری شوند.ویندوز لودر باید یکی از آنها را به آدرس دیگری ببرد.که در اکثر مواقع اتفاق می افتد که به آن relocation گفته میشود.

وقتی ما برنامه ایی مینویسیم و به فرض تابع GetModuleHandleA را درون آن فراخوانی مکنیم ،کمپایلر آدرس دستیابی به آن را نمیداند ،حال وقتی برنامه اجرا شود ،و اگر ویندوز لودر آن در آدرس 80000E300 بارگذاری کند چه اتفاقی می افتد؟ مشخص است خطا رخ میدهد ، فایل های اجرایی این مشکل را با به کارگیری جدوال پرش حل کرده اند،این یعنی هنگام کمپایل برنامه هرگونه ارجاع مثلا به تابع GetModuleHandleA علامت گذاری میشود و به محض فراخوانی ، آدرس مورد نظر جایگزین میشود و در نهایت به آدرس مناسب تبدیل میشوند. در واقع همه ی dll ها از این متد استفاده میکنند.

```
- [*G.P.U* - main thread, module FirstPro]
File View Debug Plugins Options Window Help Tools BreakPoint->

00401247  .  C9            LEAVE
00401248  .  C2 1000       RET 10
0040124B     CC            INT3
0040124C  $- FF25 14204000 JMP DWORD PTR [<&gdi32.DeleteObject>]      gdi32.DeleteObject
00401252  $- FF25 74204000 JMP DWORD PTR [<&user32.CreateDialogPara   user32.CreateDialogParamA
00401258  $- FF25 70204000 JMP DWORD PTR [<&user32.DefWindowProcA>]   user32.DefWindowProcA
0040125E  $- FF25 6C204000 JMP DWORD PTR [<&user32.DestroyWindow>]    user32.DestroyWindow
00401264  $- FF25 68204000 JMP DWORD PTR [<&user32.DispatchMessageA   user32.DispatchMessageA
0040126A  $- FF25 60204000 JMP DWORD PTR [<&user32.GetDlgItem>]       user32.GetDlgItem
00401270  $- FF25 64204000 JMP DWORD PTR [<&user32.GetDlgItemTextA]   user32.GetDlgItemTextA
00401276  $- FF25 5C204000 JMP DWORD PTR [<&user32.GetMessageA>]      user32.GetMessageA
0040127C  $- FF25 58204000 JMP DWORD PTR [<&user32.IsDialogMessageA   user32.IsDialogMessageA
00401282  $- FF25 40204000 JMP DWORD PTR [<&user32.LoadCursorA>]      user32.LoadCursorA
00401288  $- FF25 2C204000 JMP DWORD PTR [<&user32.LoadIconA>]        user32.LoadIconA
0040128E  $- FF25 30204000 JMP DWORD PTR [<&user32.LoadImageA>]       user32.LoadImageA
00401294  $- FF25 34204000 JMP DWORD PTR [<&user32.MessageBoxA>]      user32.MessageBoxA
0040129A  $- FF25 38204000 JMP DWORD PTR [<&user32.PostQuitMessage]   user32.PostQuitMessage
004012A0  $- FF25 3C204000 JMP DWORD PTR [<&user32.RegisterClassExA   user32.RegisterClassExA
004012A6  $- FF25 78204000 JMP DWORD PTR [<&user32.SendDlgItemMessa   user32.SendDlgItemMessageA
004012AC  $- FF25 44204000 JMP DWORD PTR [<&user32.SetDlgItemTextA]   user32.SetDlgItemTextA
004012B2  $- FF25 48204000 JMP DWORD PTR [<&user32.SetFocus>]         user32.SetFocus
004012B8  $- FF25 4C204000 JMP DWORD PTR [<&user32.ShowWindow>]       user32.ShowWindow
004012BE  $- FF25 50204000 JMP DWORD PTR [<&user32.TranslateMessage   user32.TranslateMessage
004012C4  $- FF25 54204000 JMP DWORD PTR [<&user32.UpdateWindow>]     user32.UpdateWindow
004012CA  .- FF25 20204000 JMP DWORD PTR [<&kernel32.ExitProcess>]    kernel32.ExitProcess
004012D0  $- FF25 1C204000 JMP DWORD PTR [<&kernel32.GetCommandLine   kernel32.GetCommandLineA
004012D6  $- FF25 24204000 JMP DWORD PTR [<&kernel32.GetModuleHand]   kernel32.GetModuleHandleA
004012DC  $- FF25 00204000 JMP DWORD PTR [<&comctl32.ImageList_Add]   comctl32.ImageList_Add
004012E2  $- FF25 0C204000 JMP DWORD PTR [<&comctl32.ImageList_Crea   comctl32.ImageList_Create
004012E8  $- FF25 08204000 JMP DWORD PTR [<&comctl32.ImageList_Dest   comctl32.ImageList_Destroy
004012EE  $- FF25 04204000 JMP DWORD PTR [<&comctl32.InitCommonCont   comctl32.InitCommonControls
004012F4     00            DB 00
```

شاید هنوز متوجه نشده باشید ،بگذارید مثالی بزنم ،در این مثال ما از kernel32.dll تابع فرضی ShowMarioBrosPicture را فراخوانی میکنیم :

```
main()
{
call ShowMarioBrosPicture();
call ShowDoYouLikeDialog()
exit();
}
ShowDoYouLikeDialog()
{
If ( user clicks yes )
{
call ShowMarioBrosPicture();
Call ShowMessage( "Yes, it's our favorite too!")
}
else
{
call showMessage( "You obviously never played Super Mario Bros.");
}
}
```

بعد از کمپایل تکه کد بالا فراخوانی تابع ها با آدرس واقعی جایگزین میشوند :

```
401000 call 402000 // Call ChowMarioBrosPicture
401002 call 401006 // Call showDoYouLikeDialog
401004 call ExitProcess
401006 Code for "Do You like It" dialog
.
40109A if (user clicks yes)
40109C call 402000 // call showMarioBrosPicture
40109E call 4010FE // call show message
4010a1 call ExitProcess
4010a3 if (user clicks no)
4010a5 call 4010FE // call show message
4010a7 call ExitProcess
4010FE code for show message
...40110A retn
```

همینطور Jump Table ما برای تابع ShowMarioBrosPicture اینگونه خواهد بود :

```
402000 JMP XXXXXXXX
```

وقتی ویندوز لودر برنامه را بارگذاری میکند به سراغ Jump Table رفته و آن را پر میکند ،اما هنوز Jump Table هیچ آدرس واقعی ندارد!! و بعد Dll را در فضای برنامه ما لود میکند،وقتی لود کامل شد ،سپس آدرس تابع ما را (مثلا:ShowMarioBrosPicture) را پیدا میکند و به Jump Table رجوع مکند و آدرس را بجای "XXXXXXXX" مینویسد،فرض کنیدکه آدرس 77CE550A را نوشته:

402000 JMP 77CE550A.

و در نهایت که Jamp Table اینگونه خواهد بود:

402000 JMP DWORD PTR DS:[<&kernel32.showMarioBrosPicture>]

بیایید دوباره به برنامه خودمان برگردیم:

```
- [*G.P.U* - main thread, module FirstPro]
File View Debug Plugins Options Window Help Tools BreakPoint->

00401247  .  C9            LEAVE
00401248  .  C2 1000       RET 10
0040124B     CC            INT3
0040124C  $- FF25 14204000 JMP DWORD PTR [<&gdi32.DeleteObject>]      gdi32.DeleteObject
00401252  $- FF25 74204000 JMP DWORD PTR [<&user32.CreateDialogPara   user32.CreateDialogParamA
00401258  $- FF25 70204000 JMP DWORD PTR [<&user32.DefWindowProcA>]   user32.DefWindowProcA
0040125E  $- FF25 6C204000 JMP DWORD PTR [<&user32.DestroyWindow>]    user32.DestroyWindow
00401264  $- FF25 68204000 JMP DWORD PTR [<&user32.DispatchMessageA   user32.DispatchMessageA
0040126A  $- FF25 60204000 JMP DWORD PTR [<&user32.GetDlgItem>]       user32.GetDlgItem
00401270  $- FF25 64204000 JMP DWORD PTR [<&user32.GetDlgItemTextA]   user32.GetDlgItemTextA
00401276  $- FF25 5C204000 JMP DWORD PTR [<&user32.GetMessageA>]      user32.GetMessageA
0040127C  $- FF25 58204000 JMP DWORD PTR [<&user32.IsDialogMessageA   user32.IsDialogMessageA
00401282  $- FF25 40204000 JMP DWORD PTR [<&user32.LoadCursorA>]      user32.LoadCursorA
00401288  $- FF25 2C204000 JMP DWORD PTR [<&user32.LoadIconA>]        user32.LoadIconA
0040128E  $- FF25 30204000 JMP DWORD PTR [<&user32.LoadImageA>]       user32.LoadImageA
00401294  $- FF25 34204000 JMP DWORD PTR [<&user32.MessageBoxA>]      user32.MessageBoxA
0040129A  $- FF25 38204000 JMP DWORD PTR [<&user32.PostQuitMessage]   user32.PostQuitMessage
004012A0  $- FF25 3C204000 JMP DWORD PTR [<&user32.RegisterClassExA   user32.RegisterClassExA
004012A6  $- FF25 78204000 JMP DWORD PTR [<&user32.SendDlgItemMessa   user32.SendDlgItemMessageA
004012AC  $- FF25 44204000 JMP DWORD PTR [<&user32.SetDlgItemTextA]   user32.SetDlgItemTextA
004012B2  $- FF25 48204000 JMP DWORD PTR [<&user32.SetFocus>]         user32.SetFocus
004012B8  $- FF25 4C204000 JMP DWORD PTR [<&user32.ShowWindow>]       user32.ShowWindow
004012BE  $- FF25 50204000 JMP DWORD PTR [<&user32.TranslateMessage   user32.TranslateMessage
004012C4  $- FF25 54204000 JMP DWORD PTR [<&user32.UpdateWindow>]     user32.UpdateWindow
004012CA  .- FF25 20204000 JMP DWORD PTR [<&kernel32.ExitProcess>]    kernel32.ExitProcess
004012D0  $- FF25 1C204000 JMP DWORD PTR [<&kernel32.GetCommandLine   kernel32.GetCommandLineA
004012D6  $- FF25 24204000 JMP DWORD PTR [<&kernel32.GetModuleHandl   kernel32.GetModuleHandleA
004012DC  $- FF25 00204000 JMP DWORD PTR [<&comctl32.ImageList_Add]   comctl32.ImageList_Add
004012E2  $- FF25 0C204000 JMP DWORD PTR [<&comctl32.ImageList_Crea   comctl32.ImageList_Create
004012E8  $- FF25 08204000 JMP DWORD PTR [<&comctl32.ImageList_Dest   comctl32.ImageList_Destroy
004012EE  $- FF25 04204000 JMP DWORD PTR [<&comctl32.InitCommonCont   comctl32.InitCommonControls
004012F4     00            DB 00
```

همانطور که گفتیم ،وقتی برنامه ایی را مینویسیم (هنوز آن را اجرا نکرده ایم) قاعدتا کمپایلر نمیداند که توابع به کار رفته درون این برنامه در چه آدرس هستند،حال برنامه را درون ollydbg باز کنید و وقتی به آدرس 401002 رسیدید کلید F7 را بفشارید تا به درون این call برویم :

```
- [*G.P.U* - main thread, module FirstPro]
File View Debug Plugins Options Window Help Tools BreakPoint->
00401247  .  C9            LEAVE
00401248  .  C2 1000       RET 10
0040124B     CC            INT3
0040124C  $- FF25 14204000 JMP DWORD PTR [<&gdi32.DeleteObject>]      gdi32.DeleteObject
00401252  $- FF25 74204000 JMP DWORD PTR [<&user32.CreateDialogPara   user32.CreateDialogParamA
00401258  $- FF25 70204000 JMP DWORD PTR [<&user32.DefWindowProcA>]   user32.DefWindowProcA
0040125E  $- FF25 6C204000 JMP DWORD PTR [<&user32.DestroyWindow>]    user32.DestroyWindow
00401264  $- FF25 68204000 JMP DWORD PTR [<&user32.DispatchMessageA   user32.DispatchMessageA
0040126A  $- FF25 60204000 JMP DWORD PTR [<&user32.GetDlgItem>]       user32.GetDlgItem
00401270  $- FF25 64204000 JMP DWORD PTR [<&user32.GetDlgItemTextA]   user32.GetDlgItemTextA
00401276  $- FF25 5C204000 JMP DWORD PTR [<&user32.GetMessageA>]      user32.GetMessageA
0040127C  $- FF25 58204000 JMP DWORD PTR [<&user32.IsDialogMessageA   user32.IsDialogMessageA
00401282  $- FF25 40204000 JMP DWORD PTR [<&user32.LoadCursorA>]      user32.LoadCursorA
00401288  $- FF25 2C204000 JMP DWORD PTR [<&user32.LoadIconA>]        user32.LoadIconA
0040128E  $- FF25 30204000 JMP DWORD PTR [<&user32.LoadImageA>]       user32.LoadImageA
00401294  $- FF25 34204000 JMP DWORD PTR [<&user32.MessageBoxA>]      user32.MessageBoxA
0040129A  $- FF25 38204000 JMP DWORD PTR [<&user32.PostQuitMessage]   user32.PostQuitMessage
004012A0  $- FF25 3C204000 JMP DWORD PTR [<&user32.RegisterClassExA   user32.RegisterClassExA
004012A6  $- FF25 78204000 JMP DWORD PTR [<&user32.SendDlgItemMessa   user32.SendDlgItemMessageA
004012AC  $- FF25 44204000 JMP DWORD PTR [<&user32.SetDlgItemTextA]   user32.SetDlgItemTextA
004012B2  $- FF25 48204000 JMP DWORD PTR [<&user32.SetFocus>]         user32.SetFocus
004012B8  $- FF25 4C204000 JMP DWORD PTR [<&user32.ShowWindow>]       user32.ShowWindow
004012BE  $- FF25 50204000 JMP DWORD PTR [<&user32.TranslateMessage   user32.TranslateMessage
004012C4  $- FF25 54204000 JMP DWORD PTR [<&user32.UpdateWindow>]     user32.UpdateWindow
004012CA  .- FF25 20204000 JMP DWORD PTR [<&kernel32.ExitProcess>]    kernel32.ExitProcess
004012D0  $- FF25 1C204000 JMP DWORD PTR [<&kernel32.GetCommandLine   kernel32.GetCommandLineA
004012D6  $- FF25 24204000 JMP DWORD PTR [<&kernel32.GetModuleHand]   kernel32.GetModuleHandleA
004012DC  $- FF25 00204000 JMP DWORD PTR [<&comctl32.ImageList_Add]   comctl32.ImageList_Add
004012E2  $- FF25 0C204000 JMP DWORD PTR [<&comctl32.ImageList_Crea   comctl32.ImageList_Create
004012E8  $- FF25 08204000 JMP DWORD PTR [<&comctl32.ImageList_Dest   comctl32.ImageList_Destroy
004012EE  $- FF25 04204000 JMP DWORD PTR [<&comctl32.InitCommonCont   comctl32.InitCommonControls
004012F4     00            DB 00
```

حالا یک بار دیگر کلید F7 را بفشارید تا به آدرس واقعی تابع (GetModuleHandleA) در آدرس (7780B741) منتقل شویم :

خب اگر به تایتل پنجره ی CPU نگاه کنید میبینید که وارد ماژول Kernel32.dll شده ایم :

```
File View Debug Plugins Options Windo
*G.P.U* - main thread, module kernel32
tModule  8BFF      MOV EDI,
6ECF43   55        PUSH EBP
6ECF44   8BEC      MOV EBP,
6ECF46   5D        POP EBP
6ECF47 ^ EB ED     JMP SHOR
6ECF49   3935 80677676  CMP DWOR
6ECF4F v 75 05     JNZ SHOR
```

دوباره برنامه را ری استارت کنید و اینبار از روی تابع (GetModuleHandleA) با کلید F8 عبور کنید وبر بروی خط توقف کنید کلید Spaceرا بفشارید ما آدرس 4012D0 را میبینیم

```
- [*G.P.U* - main thread, module FirstPro]
File View Debug Plugins Options Window Help Tools BreakPoint->
<ModuleEr $  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>    GetCommandLineA
```

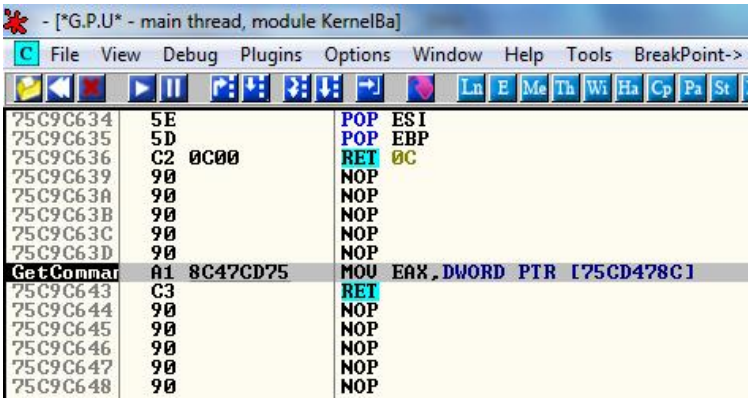
```
Assemble at 0040100C
CALL 004012D0
Fill with NOP's        Assemble        Cancel
```

به صورت دستی میخواهیم وارد آدرس بالا شویم (برای آشنایی بهتر با OllyDBG)،کلید های Ctrl+Gرا بفشارید یا آیکـن کلیـک کنید.و آدرس را وارد و کلید Ok را بفشارید:

ما بروی تابع GetCommandLineA قرار داریم :

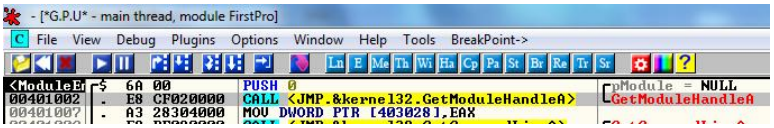
```
004012CA  .- FF25 20204000 JMP DWORD PTR [<&kernel32.ExitProcess>]   kernel32.ExitProcess
004012D0  $- FF25 1C204000 JMP DWORD PTR [<&kernel32.GetCommandLine  kernel32.GetCommandLineA
004012D6  $- FF25 24204000 JMP DWORD PTR [<&kernel32.GetModuleHand]  kernel32.GetModuleHandleA
004012DC  $- FF25 00204000 JMP DWORD PTR [<&comctl32.ImageList_Add]  comctl32.ImageList_Add
```

و با کلیک F7 به ابتدای تابع"GetCommandLineA " در آدرس (7C812FBD) ارجاع داده میشویم :



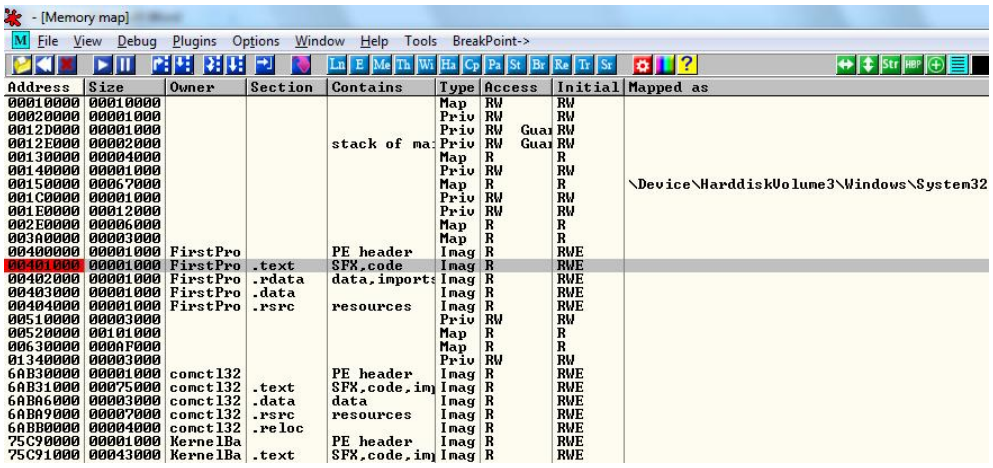
ارجاع به داخل و بیرون Dll

گاهی اوقات شما در حال شکستن حفاظت برنامه ایی هستید که ناگهان درون Dll های سیستمی غوطه ور میشوید و نمیدانید چگونه به برنامه ی اصلی بازگردید برای این کار من دو روش را پیشنهاد میکنم ،برای شروع ابتدا برنامه را ری استارت کنید و با کلید F7 وارد jump Table و سپس وارد تابع "GetCommandLineA " در kernel32.dll شوید حال با کلید Ctrl+F9 یا "Execute till user code" میتوانید به دستورالعمل بعد از CALL <JMP.&kernel32.GetModuleHandleA> یعنی MOV DWORD PTR [403028],EAX بازگردید



این دستور یعنی هرجا که هستی دستورات را اجرا کن تا به کد برنامه ی ما برسی و آنجا توقـف کـن.بـا اینکـار میتـوان رونـد را ادامـه داد،روش دیگر استفاده از Memory Map است :

با فشردن کلید Alt+M وارد پنجره ی مذکور میشویم و سپس بروی سکشن کد که در اینجا آدرس 401000 اسـت یـک نقطـه توقـف معمولی یا حافظه ایی گذاشته :

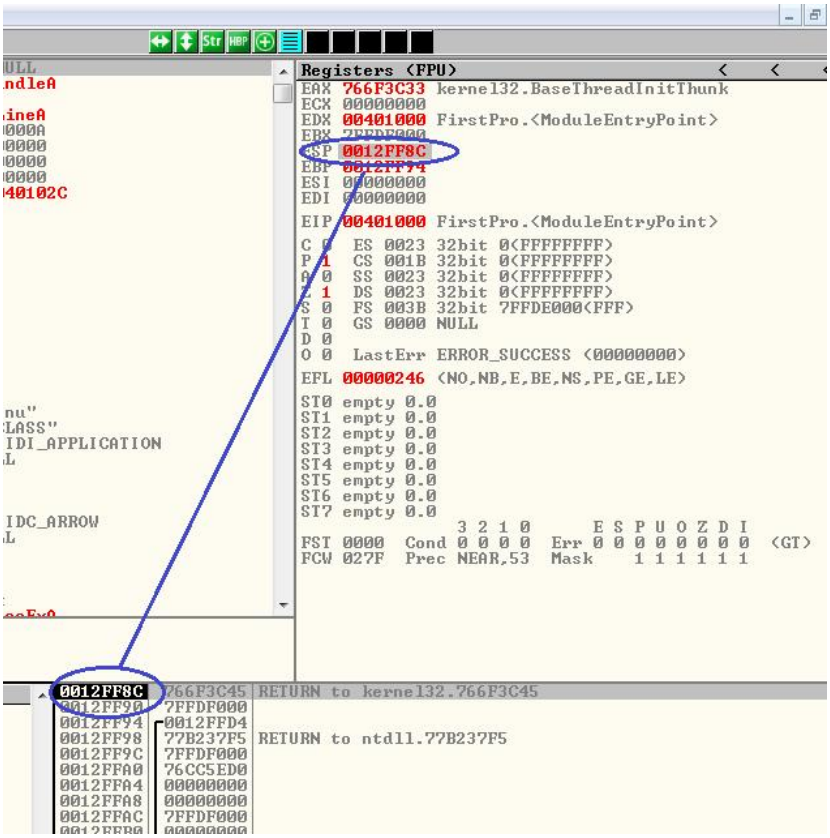


سپس برنامه را با کلید F9 اجرا کنید ،همانطور که میبینید برنامه در آدرس 401000 متوقف میشود ،حال اگر نقطه ی توقف شما از نوع حافظه ایی است آن را کلا بردارید چرا که با هر بار اجرا برنامه در آن سکشن متوقف خواهد شد ☺

اطلاعات بیشتر در مورد Stack

واقعا Stack در مهندسی معکوس بسیار مهم است ،و اگر درمورد آن آگاهی نداشته باشید مطمئن باشید هیچ وقت یک کرکر واقعی نخواهید شد.سعی میکنم بسیار ساده و روان توضیح دهم

بعد از اینکه برنامه را دوباره ری استارت کردید به پنجره ی ثبات ها نگاه کنید ،ثبات ESP را نگاه کنید که قرمز شده است،این ثبات به بالای استک اشاره میکند،و مقدار آن 12FF8C است(در کامپیوتر شما ممکن است آدرس فرق کند). حالا به پنجره ی پائین استک نگاه کنید ،آدرس 12FF8C در ثبات ESP با آدرس پنجره ی استک یکی است:



حالا یک بار کلید F7 یا F8 را بفشارید، و خوب به پنجره ی استک نگاه کنید که مقدار صفر روی آن ریخته میشود:

```
0012FF88  00000000  hModule = NULL
0012FF8C  766F3C45  RETURN to kernel32.766F3C45
0012FF90  7FFDF000
0012FF94  0012FFD4
0012FF98  77B237F5  RETURN to ntdll.77B237F5
0012FF9C  7FFDF000
0012FFA0  76CC5ED0
0012FFA4  00000000
0012FFA8  00000000
0012FFAC  7FFDF000
0012FFB0  00000000
0012FFB4  00000000
0012FFB8  00000000
0012FFBC  0012FFA0
0012FFC0  00000000
0012FFC4  FFFFFFFF  End of SEH chain
0012FFC8  77ADE0ED  SE handler
0012FFCC  016FB37C
0012FFD0  00000000
0012FFD4  0012FFEC
0012FFD8  77B237C8  RETURN to ntdll.77B237C8 from ntdll.77B237CE
0012FFDC  00401000  FirstPro.<ModuleEntryPoint>
0012FFE0  7FFDF000
0012FFE4  00000000
0012FFE8  00000000
0012FFEC  00000000
0012FFF0  00000000
0012FFF4  00401000  FirstPro.<ModuleEntryPoint>
0012FFF8  7FFDF000
0012FFFC  00000000
```

حالا به پنجره ی ثبات ها وبه ثبات ESP نگاه کنید :

```
Registers (FPU)
EAX 766F3C33 kernel32.BaseThreadInitThunk
ECX 00000000
EDX 00401000 FirstPro.<ModuleEntryPoint>
EBX 7FFD9000
ESP 0012FF88
EBP 0012FF94
ESI 00000000
EDI 00000000

EIP 00401002 FirstPro.00401002

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
```

12FF88-12FF8C=4

میبینیم که ۴ بایت Push شده،حال یک بار F8را بفشارید و از تابع GetModuleHandleAعبور کنید و سپس به پنجره استک نگاه کنید:

```
0012FF8C  766F3C45  RETURN to kernel32.766F3C45
0012FF90  7FFD9000
0012FF94  0012FFD4
0012FF98  77B237F5  RETURN to ntdll.77B237F5
0012FF9C  7FFD9000
0012FFA0  76DA4FFC
0012FFA4  00000000
0012FFA8  00000000
0012FFAC  7FFD9000
0012FFB0  00000000
0012FFB4  00000000
0012FFB8  00000000
0012FFBC  0012FFA0
0012FFC0  00000000
0012FFC4  FFFFFFFF  End of SEH chain
0012FFC8  77ADE0ED  SE handler
0012FFCC  0179A250
0012FFD0  00000000
0012FFD4  0012FFEC
0012FFD8  77B237C8  RETURN to ntdll.77B237C8 from ntdll.77B237CE
0012FFDC  00401000  FirstPro.<ModuleEntryPoint>
0012FFE0  7FFD9000
0012FFE4  00000000
0012FFE8  00000000
0012FFEC  00000000
0012FFF0  00000000
0012FFF4  00401000  FirstPro.<ModuleEntryPoint>
0012FFF8  7FFD9000
0012FFFC  00000000

ESP  EBP  NONE
```

همانطور که میبینید صفر از پشته POP شد و پشته به سمت پائین آمد ،حال با کلید F8 پائین بیایید تا به اولین Pushبر سید ،میبینید تا رسیدن به اولین Push هیچ تغیری روی استک صورت نمیگیرد که دلیل آن هم این است که چیزی روی آن Push نشده

اکنون دستور العمل PUSH 0A اجرا شده و 0A در بالای پشته قرار داده شده،و ثبات ESPچهار بایت کم شد(هر موقع عملیاتPush صورت میگیرد این ثبات کم میشود و هرگاه POP صورت میگیرد ،افزایش پیدا میکند) دوباره کلید F8 را بفشارید ،اگر به بالای پشته نگاه کنید میبینید که مقدار ۴ بایتی بروی پشته قرار دارد.

به پشته نگاه کنید،مقدار 00000000 دروبالای پشته قرار دارد ،چرا؟ به خط زیر نگاه کنید:

```
00401013  .  FF35 2C304000  PUSH DWORD PTR [40302C]
00401019  .  6A 00          PUSH 0
```

معنی خط بالا این است که ، مقدار ۴ بایتی را از آدرس 40302C را بگیر و درون پشته قرار بده.مقدار 40302C چیست؟00000000 ،بله ، البته (به بالای پنجری استک نگاه کنید) برای متوجه شدن برروی همان دستور العمل (00401013) راست کلیک کنید Follow in Dump” -> “Memory Address” :

Address	Hex dump	ASCII
0040302C	00 00 00 00 00 00 00 00	.....
00403034	00 00 00 00 00 00 00 00	.....
0040303C	00 00 00 00 00 00 00 00	.....
00403044	00 00 00 00 00 00 00 00	.....
0040304C	00 00 00 00 00 00 00 00	.....
00403054	00 00 00 00 00 00 00 00	.....
0040305C	00 00 00 00 00 00 00 00	.....
00403064	00 00 00 00 00 00 00 00	.....
0040306C	00 00 00 00 00 00 00 00	.....
00403074	00 00 00 00 00 00 00 00	.....
0040307C	00 00 00 00 00 00 00 00	.....
00403084	00 00 00 00 00 00 00 00	.....
0040308C	00 00 00 00 00 00 00 00	.....
00403094	00 00 00 00 00 00 00 00	.....
0040309C	00 00 00 00 00 00 00 00	.....
004030A4	00 00 00 00 00 00 00 00	.....
004030AC	00 00 00 00 00 00 00 00	.....
004030B4	00 00 00 00 00 00 00 00	.....
004030BC	00 00 00 00 00 00 00 00	.....
004030C4	00 00 00 00 00 00 00 00	.....
004030CC	00 00 00 00 00 00 00 00	.....
004030D4	00 00 00 00 00 00 00 00	.....
004030DC	00 00 00 00 00 00 00 00	.....
004030E4	00 00 00 00 00 00 00 00	.....
004030EC	00 00 00 00 00 00 00 00	.....
004030F4	00 00 00 00 00 00 00 00	.....
004030FC	00 00 00 00 00 00 00 00	.....
00403104	00 00 00 00 00 00 00 00	.....
0040310C	00 00 00 00 00 00 00 00	.....
00403114	00 00 00 00 00 00 00 00	.....
0040311C	00 00 00 00 00 00 00 00	.....
00403124	00 00 00 00 00 00 00 00	.....

همانطور که میبینید مقادیر حافظه از آدرس 40302C شروع شده،(بجزچهار بایت اول push شده) بقیه ، فضایی است که کامپایلر برای متغییر های ما در نظر گرفته ولی فعلا همگی باصفر مقدار دهی اولیه شده اند ،اگر میخواهید ببینید ،ابتدا برنامه را کامل اجرا کنید و متنی در برنامه بنویسید و سپس به آدرس مذکور نگاه کنید:

Address	Hex dump	ASCII
0040302C	00 00 00 00 57 57 57 2E	....WWW.
00403034	49 52 41 4E 4C 45 44 2E	IRANLED.
0040303C	43 4F 4D 00 00 00 00 00	COM.....
00403044	00 00 00 00 00 00 00 00	.....
0040304C	00 00 00 00 00 00 00 00	.....
00403054	00 00 00 00 00 00 00 00	.....
0040305C	00 00 00 00 00 00 00 00	.....
00403064	00 00 00 00 00 00 00 00	.....
0040306C	00 00 00 00 00 00 00 00	.....
00403074	00 00 00 00 00 00 00 00	.....
0040307C	00 00 00 00 00 00 00 00	.....
00403084	00 00 00 00 00 00 00 00	.....
0040308C	00 00 00 00 00 00 00 00	.....

Dialog As Main

Options

WWW.IRANLED.COM

خیلی خب ،مجددا کلید F8 را بفشارید تا به Push بعدی رسیم که مقدار 0را درون استک میریزد:

```
00401019  .  6A 00          PUSH 0
```

و خط بعدی:

```
0040101B  .  FF35 28304000  PUSH DWORD PTR [403028]
```

میبینید که اشاره به آدرس 403028 دارد،بیایید به این آدرس برویم :

همانطور که میبینیدآدرس 403028 حاوی مقدار 400000 که با فشردن کلید F8 در پشته Push خواهد شد , همه ی آرگومانها کامل در پشته قرار گرفتند.حال کلید F7را یکبار فشار دهید :

درست است ،یک مقدار به پشته اضافه شد!!که در حقیقت این آدرس بازگشتی (401021) توسط دستور العمل Call درون پشته Push شده است و توسط دستور العمل RET بازیابی یا POP میشودو پس از خاتمه ی برنامه به آدرس401021 ارجاع داده میشود :

```
004011A2  .  C9          LEAVE
004011A3  .  C2 1000     RET 10
004011A6  .  55          PUSH EBP
```

10 مقابل دستور ret به معنای پاک کردن ۱۰ بایت از بالای استک است ،برای بازگشتن به خط مورد نظر بروی ثبات EIP دبل کلیک کنید.