Tutorial #2 : Intro To Olly Debug

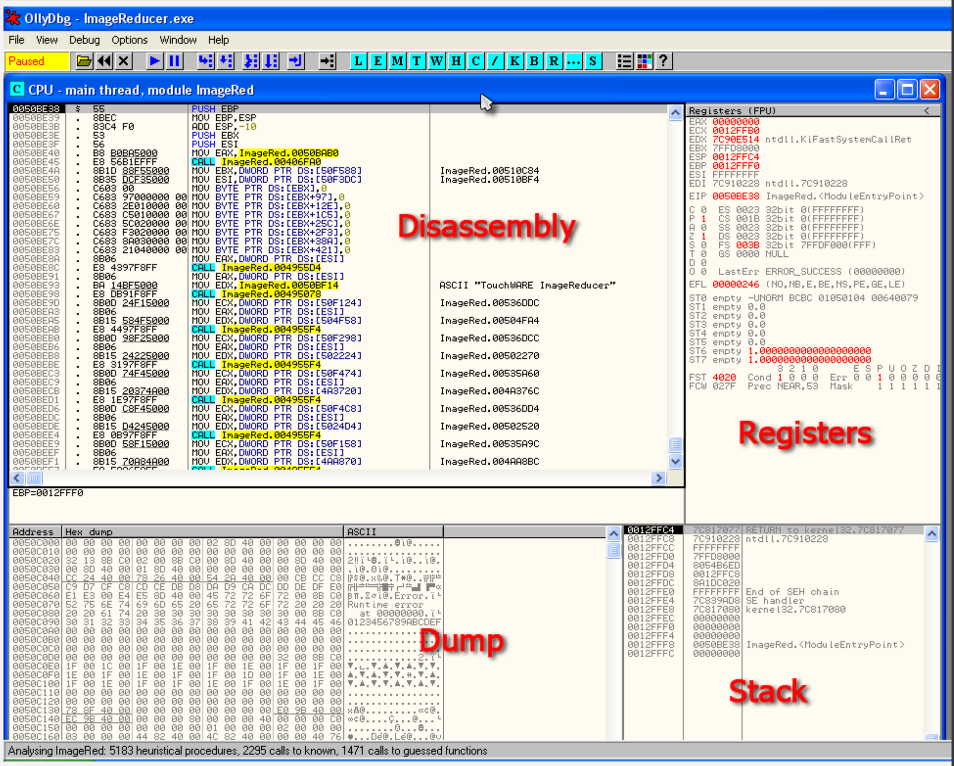
by R4ndom on May.23, 2012, under Reverse Engineering, Tutorials

What is Olly Debugger?

From the author, Oleh Yuschuk, "OllyDbg is a 32-bit assembler level analysing debugger for Microsoft® Windows®. Emphasis on binary code analysis makes it particularly useful in cases where source is unavailable. " Olly is also a "dynamic" debugger, meaning it allows the user to change quite a few things as the program is running. This is very important when experimenting with a binary, trying to figure out how it works. Olly has many, many great features, and that is why it is probably the number one debugger used for reverse engineering (at least in ring 3, but we'll get to that later.)

An Overview

Here is a picture of Olly's main display, along with some labels:


OllyDbg - ImageReducer.exe
File View Debug Options Window Help
Paused
L E M T W H C / K B R ... S
CPU - main thread, module ImageRed
Disassembly
Registers
Dump
Stack

```
0050BE38  $  55             PUSH EBP
0050BE39  .  8BEC           MOV EBP,ESP
0050BE3B  .  83C4 F0        ADD ESP,-10
0050BE3E  .  53             PUSH EBX
0050BE3F  .  56             PUSH ESI
0050BE40  .  B8 B0BA5000    MOV EAX,ImageRed.0050BAB0
0050BE45  .  E8 56B1EFFF    CALL ImageRed.00406FA0
0050BE4A  .  8B1D 88F55000  MOV EBX,DWORD PTR DS:[50F588]          ImageRed.00510C84
0050BE50  .  8B35 DCF35000  MOV ESI,DWORD PTR DS:[50F3DC]          ImageRed.00510BF4
0050BE56  .  C603 00        MOV BYTE PTR DS:[EBX],0
0050BE59  .  C683 97000000 00 MOV BYTE PTR DS:[EBX+97],0
0050BE60  .  C683 2E010000 00 MOV BYTE PTR DS:[EBX+12E],0
0050BE67  .  C683 C5010000 00 MOV BYTE PTR DS:[EBX+1C5],0
0050BE6E  .  C683 5C020000 00 MOV BYTE PTR DS:[EBX+25C],0
0050BE75  .  C683 F3020000 00 MOV BYTE PTR DS:[EBX+2F3],0
0050BE7C  .  C683 8A030000 00 MOV BYTE PTR DS:[EBX+38A],0
0050BE83  .  C683 21040000 00 MOV BYTE PTR DS:[EBX+421],0
0050BE8A  .  8B06           MOV EAX,DWORD PTR DS:[ESI]
0050BE8C  .  E8 4397F8FF    CALL ImageRed.004955D4
0050BE91  .  8B06           MOV EAX,DWORD PTR DS:[ESI]
0050BE93  .  BA 14BF5000    MOV EDX,ImageRed.0050BF14              ASCII "TouchWARE ImageReducer"
0050BE98  .  E8 DB91F8FF    CALL ImageRed.00495078
0050BE9D  .  8B0D 24F15000  MOV ECX,DWORD PTR DS:[50F124]          ImageRed.00536DDC
0050BEA3  .  8B06           MOV EAX,DWORD PTR DS:[ESI]
0050BEA5  .  8B15 584F5000  MOV EDX,DWORD PTR DS:[504F58]          ImageRed.00504FA4
0050BEAB  .  E8 4497F8FF    CALL ImageRed.004955F4
0050BEB0  .  8B0D 98F25000  MOV ECX,DWORD PTR DS:[50F298]          ImageRed.00536DCC
0050BEB6  .  8B06           MOV EAX,DWORD PTR DS:[ESI]
0050BEB8  .  8B15 24225000  MOV EDX,DWORD PTR DS:[502224]          ImageRed.00502270
0050BEBE  .  E8 3197F8FF    CALL ImageRed.004955F4
0050BEC3  .  8B0D 74F45000  MOV ECX,DWORD PTR DS:[50F474]          ImageRed.00535A60
0050BEC9  .  8B06           MOV EAX,DWORD PTR DS:[ESI]
0050BECB  .  8B15 20374A00  MOV EDX,DWORD PTR DS:[4A3720]          ImageRed.004A376C
0050BED1  .  E8 1E97F8FF    CALL ImageRed.004955F4
0050BED6  .  8B0D C8F45000  MOV ECX,DWORD PTR DS:[50F4C8]          ImageRed.00536DD4
0050BEDC  .  8B06           MOV EAX,DWORD PTR DS:[ESI]
0050BEDE  .  8B15 D4245000  MOV EDX,DWORD PTR DS:[5024D4]          ImageRed.00502520
0050BEE4  .  E8 0B97F8FF    CALL ImageRed.004955F4
0050BEE9  .  8B0D 58F15000  MOV ECX,DWORD PTR DS:[50F158]          ImageRed.00535A9C
0050BEEF  .  8B06           MOV EAX,DWORD PTR DS:[ESI]
0050BEF1  .  8B15 70A84A00  MOV EDX,DWORD PTR DS:[4AA870]          ImageRed.004AA8BC

EBP=0012FFF0
```

```
Registers (FPU)
EAX 00000000
ECX 0012FFB0
EDX 7C90E514 ntdll.KiFastSystemCallRet
EBX 7FFD8000
ESP 0012FFC4
EBP 0012FFF0
ESI FFFFFFFF
EDI 7C910228 ntdll.7C910228

EIP 0050BE38 ImageRed.<ModuleEntryPoint>

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 1  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDF000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (00000000)

EFL 00000246 (NO,NB,E,BE,NS,PE,GE,LE)

ST0 empty -UNORM BCBC 01050104 00640079
ST1 empty 0.0
ST2 empty 0.0
ST3 empty 0.0
ST4 empty 0.0
ST5 empty 0.0
ST6 empty 1.0000000000000000000
ST7 empty 1.0000000000000000000
               3 2 1 0      E S P U O Z D I
FST 4020  Cond 1 0 0 0  Err 0 0 1 0 0 0 0 0
FCW 027F  Prec NEAR,53  Mask    1 1 1 1 1 1
```

```
Address   Hex dump                                         ASCII
0050C000  00 00 00 00 00 00 00 00 02 8D 40 00 00 00 00 00  ........☻ì@.....
0050C010  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0050C020  32 13 8B C0 02 00 8B C0 00 8D 40 00 00 8D 40 00  2‼ï└☻.ï└.ì@..ì@.
0050C030  00 8D 40 00 01 8D 40 00 00 00 00 00 00 00 00 00  .ì@.☺ì@.........
0050C040  CC 24 40 00 78 26 40 00 54 2A 40 00 00 CB CC C8  ╠$@.x&@.T*@..╦╠╚
0050C050  C9 D7 CF C8 CD CE DB D8 DA D9 CA DC DD DE DF E0  ╔╫╧╚═╬█╪┌┘╩▄▌▐▀α
0050C060  E1 E3 00 E4 E5 8D 40 00 45 72 72 6F 72 00 8B C0  ßπ.Σσì@.Error.ï└
0050C070  52 75 6E 74 69 6D 65 20 65 72 72 6F 72 20 20 20  Runtime error
0050C080  20 20 61 74 20 30 30 30 30 30 30 30 30 00 8B C0    at 00000000.ï└
0050C090  30 31 32 33 34 35 36 37 38 39 41 42 43 44 45 46  0123456789ABCDEF
0050C0A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0050C0B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0050C0C0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0050C0D0  00 00 00 00 00 00 00 00 00 00 00 00 32 00 8B C0  ............2.ï└
0050C0E0  1F 00 1C 00 1F 00 1E 00 1F 00 1E 00 1F 00 1F 00  ▼.∟.▼.▲.▼.▲.▼.▼.
0050C0F0  1E 00 1F 00 1E 00 1F 00 1F 00 1D 00 1F 00 1E 00  ▲.▼.▲.▼.▼.↔.▼.▲.
0050C100  1F 00 1E 00 1F 00 1F 00 1E 00 1F 00 1E 00 1F 00  ▼.▲.▼.▼.▲.▼.▲.▼.
0050C110  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0050C120  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0050C130  78 8F 40 00 00 00 00 00 00 00 00 00 E0 9B 40 00  xÅ@.........α¢@.
0050C140  EC 9B 40 00 00 00 00 80 00 00 00 40 00 00 00 C0  ∞¢@....Ç...@...└
0050C150  00 00 00 00 00 00 00 00 01 00 00 00 02 00 00 00  ........☺...☻...
0050C160  03 00 00 00 44 82 40 00 4C 82 40 00 00 00 40 76  ♥...Dé@.Lé@...@v
```

```
0012FFC4  7C817077  RETURN to kernel32.7C817077
0012FFC8  7C910228  ntdll.7C910228
0012FFCC  FFFFFFFF
0012FFD0  7FFD8000
0012FFD4  8054B6ED
0012FFD8  0012FFC8
0012FFDC  8A1DC020
0012FFE0  FFFFFFFF  End of SEH chain
0012FFE4  7C839AD8  SE handler
0012FFE8  7C817080  kernel32.7C817080
0012FFEC  00000000
0012FFF0  00000000
0012FFF4  00000000
0012FFF8  0050BE38  ImageRed.<ModuleEntryPoint>
0012FFFC  00000000
```

```
Analysing ImageRed: 5183 heuristical procedures, 2295 calls to known, 1471 calls to guessed functions
```

Main Olly Display

Olly opens with the default window, CPU, open. This is where most of the "big-picture" data is. If you ever close this window, just click the "C" icon in the toolbar. It is separated into 4 main fields; Disassembly, Registers, Stack, and Dump. Here is a description of each section.

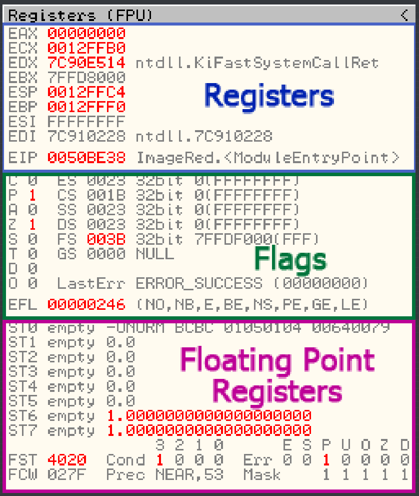
1. Disassembly

This window contains the main disassembly of the code for the binary. This is where Olly displays information in the binary, including the opcodes and translated assembly language. The first column is the address (in memory) of the instruction. The second column is what's called the opcodes- in assembly language, every instruction has at least one code associated with it (many have multiple). This is the code that the CPU really wants and the only code it can read. These opcodes make up ‘machine language’, the language of the computer. If you were to view the raw data in a binary (using a hex editor) you would see a string of these opcodes, and nothing more. One of Olly’s main jobs is to ‘disassemble’ this ‘machine language’ into more human readable assembly language. The third column is this assembly language. Granted, to someone who does not know assembly, it doesn’t look much better than the opcodes, but as you learn more, the assembly offers FAR more insight into what the code is doing.

The last column is Olly’s comments on that line of code. Sometimes this contains the names of API calls (if Olly can figure them out) such as CreateWindow and GetDlgItemX. Olly also tries to help us understand the code by naming any calls that are not part of the API with helpful names, in the case of this picture, “ImageRed.00510C84″ and “ImageRed.00510BF4″. Granted, these are not that helpful, but Olly also allows us to change them into more meaningful names. You may also put your own comments in this column; just double-click on the line in this column and a box pops up allowing you to enter your comment. These comments will then be saved for next time automatically.

2. Registers

Every CPU has in it a collection of registers. These are temporary holders for values, much like a variable in any high-level programming language. Here is a more detailed (and labeled) view of the registers window:



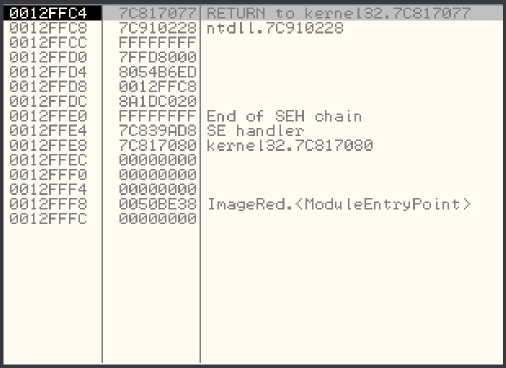
```
Registers (FPU)
EAX 00000000
ECX 0012FFB0
EDX 7C90E514 ntdll.KiFastSystemCallRet
EBX 7FFD8000
ESP 0012FFC4
EBP 0012FFF0
ESI FFFFFFFF
EDI 7C910228 ntdll.7C910228

EIP 0050BE38 ImageRed.<ModuleEntryPoint>
C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 1  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDF000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (00000000)

EFL 00000246 (NO,NB,E,BE,NS,PE,GE,LE)
ST0 empty -UNORM BCBC 01050104 00640079
ST1 empty 0.0
ST2 empty 0.0
ST3 empty 0.0
ST4 empty 0.0
ST5 empty 0.0
ST6 empty 1.0000000000000000000
ST7 empty 1.0000000000000000000
               3 2 1 0      E S P U O Z D
FST 4020  Cond 1 0 0 0  Err 0 0 1 0 0 0 0
FCW 027F  Prec NEAR,53  Mask    1 1 1 1 1
```

On the top is the actual CPU Registers. The registers will change color if they have been changed from black to red (makes it really easy to watch for changes). You can also double click on any of the registers to change their contents. These registers are used for many things, and we will have much to say about them later.

The middle section are flags, used by the CPU to flag the code that something has happened (two numbers are equal, one number is greater than another, etc). Double clicking one of the flags changes it. These will also play an important part in our journey.

The bottom section are the FPU, or Floating Point Unit registers. These are used whenever the CPU performs any arithmetic involving decimal points. These are rarely used by reversers, mostly when we get into encryption.

3. The Stack



```
0012FFC4  7C817077 RETURN to kernel32.7C817077
0012FFC8  7C910228 ntdll.7C910228
0012FFCC  FFFFFFFF
0012FFD0  7FFD8000
0012FFD4  8054B6ED
0012FFD8  0012FFC8
0012FFDC  8A1DC020
0012FFE0  FFFFFFFF End of SEH chain
0012FFE4  7C839AD8 SE handler
0012FFE8  7C817080 kernel32.7C817080
0012FFEC  00000000
0012FFF0  00000000
0012FFF4  00000000
0012FFF8  0050BE38 ImageRed.<ModuleEntryPoint>
0012FFFC  00000000
```

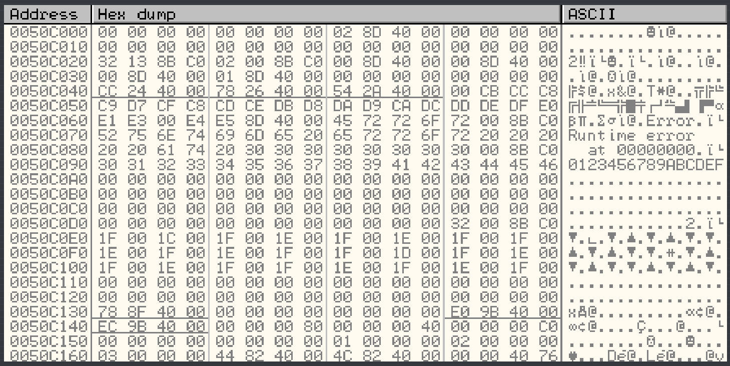
The stack is a section of memory reserved for the binary as a ‘temporary’ list of data. This data includes pointers to addresses in memory, strings, markers, and most importantly, return addresses for the code to return to when calling a function. When a method in a program calls another method, control needs to be shifted to this new method so that it can retun. The CPU must keep track of where this new method was called from so that when this new method is done, the CPU can return to where it was called and continue executing the code after the call. The stack is where the CPU will hold this return address.

One thing to know about the stack is that it is a a “First In, Last Out” data structure. The metaphor normally used is one of those stacks of plates in a cafeteria that are spring loaded. When you ‘push’ a plate onto the top, all of the plates underneath are pushed down. When you remove (‘pop’) a plate off the top, all of the plates that were underneath raise up one level. We will see this in action in the next tutorial, so don’t worry if it’s a little hazy.

In this picture, the first column is the address of each data member, the second column is the hex, 32-bit representation of the data, and the last column is Olly’s comments about this data item, if it can figure them out. If you notice the first row, you will see a “RETURN to kernel…” comment. This is an address that the CPU has placed on the stack for when the current function is done, so that it will know where to return to.

In Olly, you can right click on the stack and choose ‘modify’ to change the contents.

4 The Dump

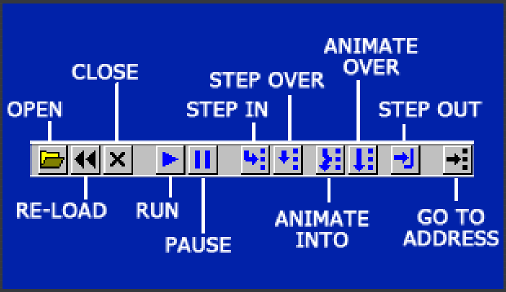


```
Address  Hex dump                                         ASCII
0050C000 00 00 00 00 00 00 00 00 02 8D 40 00 00 00 00 00  ........☻ì@.....
0050C010 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0050C020 32 13 8B C0 02 00 8B C0 00 8D 40 00 00 8D 40 00  2‼ï└☻.ï└.ì@..ì@.
0050C030 00 8D 40 00 01 8D 40 00 00 00 00 00 00 00 00 00  .ì@.☺ì@.........
0050C040 CC 24 40 00 78 26 40 00 54 2A 40 00 00 CB CC C8  ╠$@.x&@.T*@..╦╠╚
0050C050 C9 D7 CF C8 CD CE DB D8 DA D9 CA DC DD DE DF E0  ╔╫╧╚═╬█╪┌┘╩▄▌▐▀α
0050C060 E1 E3 00 E4 E5 8D 40 00 45 72 72 6F 72 00 8B C0  ßπ.Σσì@.Error.ï└
0050C070 52 75 6E 74 69 6D 65 20 65 72 72 6F 72 20 20 20  Runtime error
0050C080 20 20 61 74 20 30 30 30 30 30 30 30 30 00 8B C0    at 00000000.ï└
0050C090 30 31 32 33 34 35 36 37 38 39 41 42 43 44 45 46  0123456789ABCDEF
0050C0A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0050C0B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0050C0C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0050C0D0 00 00 00 00 00 00 00 00 00 00 00 00 32 00 8B C0  ............2.ï└
0050C0E0 1F 00 1C 00 1F 00 1E 00 1F 00 1E 00 1F 00 1F 00  ▼.∟.▼.▲.▼.▲.▼.▼.
0050C0F0 1E 00 1F 00 1E 00 1F 00 1F 00 1D 00 1F 00 1E 00  ▲.▼.▲.▼.▼.↔.▼.▲.
0050C100 1F 00 1E 00 1F 00 1F 00 1E 00 1F 00 1E 00 1F 00  ▼.▲.▼.▼.▲.▼.▲.▼.
0050C110 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0050C120 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0050C130 78 8F 40 00 00 00 00 00 00 00 00 00 E0 9B 40 00  xÅ@.........α¢@.
0050C140 EC 9B 40 00 00 00 00 80 00 00 00 40 00 00 00 C0  ∞¢@....Ç...@...└
0050C150 00 00 00 00 00 00 00 00 01 00 00 00 02 00 00 00  ........☺...☻...
0050C160 03 00 00 00 44 82 40 00 4C 82 40 00 00 00 40 76  ♥...Dé@.Lé@...@v
```

Earlier in this tutorial, when we talked about the raw ‘opcodes’ that the CPU reads inside a binary, I mentioned that you could see this raw data in a hex viewer. Well, in Olly, you don’t have to. The dump window is a built-in hex viewer that lets you see the raw binary data, only in memory as opposed to on disk. Usually it shows two views of the same data; hexadecimal and ASCII. These are represented in the two right-hand columns in the previous picture (the first column is the address in memory that the data resides.) Olly does allow these representations of data to be changed, and we will see this later in the tutorials.

The Toolbar

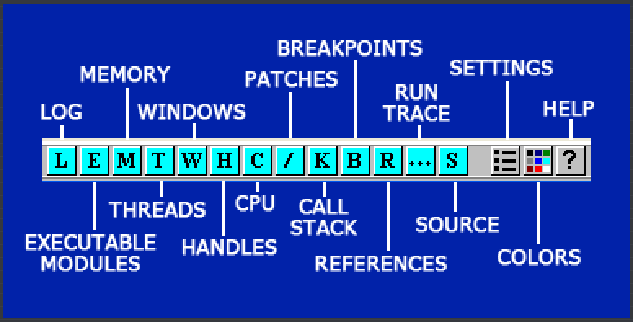
Unfortunately, the Olly toolbar leaves a little much to be desired (especially as English is not the author’s first tongue.) I have labeled the left hand toolbar icons to help:



These are your main controls to run code. Keep in mind that, especially as you start using Olly, all of these buttons are also accessible from the “Debug” drop down menu, so if you don’t know what something is, you can look in there.

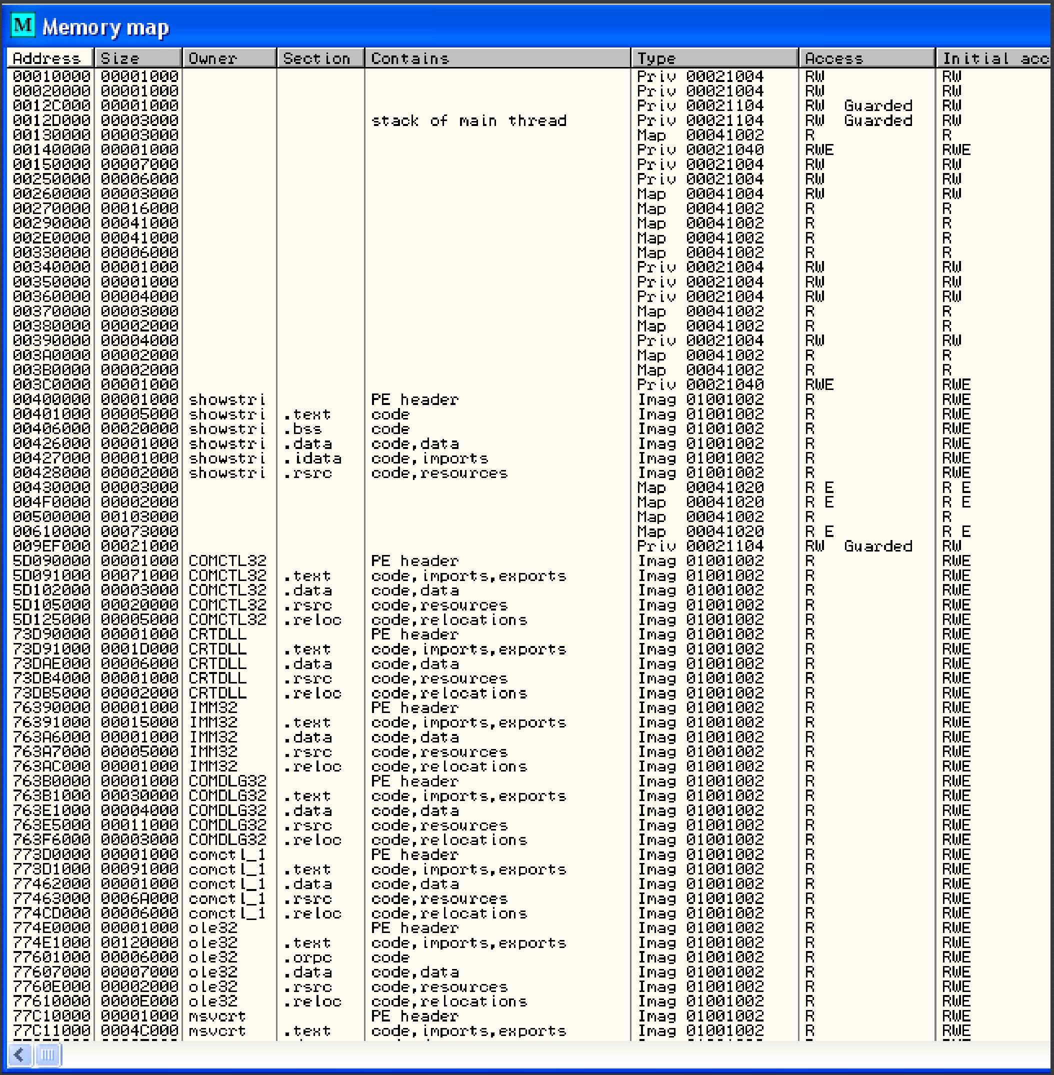
I will make a couple of remarks about some of the icons. “Re-load” is basically to restart the app and pause it at the entry point. All patches (see later) will be removed, some breakpoints will be disabled, and the app will not have run any code yet, well, most of the time anyway. “Run” and “Pause” do just that. “Step In” means run one line of code and then pause again, calling into a function call if there was one. “Step Over” does the same thing, but jumps over a call to another function. “Animate” is just like Step In and Over except it does it slowly enough that you can watch it. You won't use this much, but sometimes it's fun to watch code run, especially if it's a polymorphic binary and you can watch the code change. But I'm getting ahead of myself…

Next is the (even more cryptic) windows icons:



Each of these icons opens a window, some of which you will use often, some rarely. Seeing as they are not the most intuitive letters, you can also do like I did and just start clicking them all until you find what you want. Each of these are also accessible in the "View" menu, so you can get some help when first starting out.I will go over some of the more common windows right now:

1. (M)emory

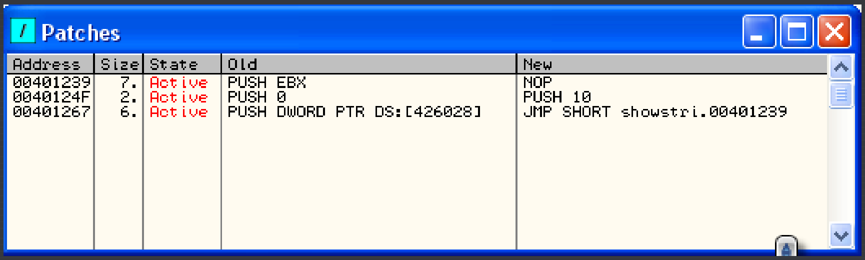


Memory map

Address	Size	Owner	Section	Contains	Type	Access	Initial acc
00010000	00001000				Priv 00021004	RW	RW
00020000	00001000				Priv 00021004	RW	RW
0012C000	00001000				Priv 00021104	RW Guarded	RW
0012D000	00003000			stack of main thread	Priv 00021104	RW Guarded	RW
00130000	00003000				Map 00041002	R	R
00140000	00001000				Priv 00021040	RWE	RWE
00150000	00007000				Priv 00021004	RW	RW
00250000	00006000				Priv 00021004	RW	RW
00260000	00003000				Map 00041004	RW	RW
00270000	00016000				Map 00041002	R	R
00290000	00041000				Map 00041002	R	R
002E0000	00041000				Map 00041002	R	R
00330000	00006000				Map 00041002	R	R
00340000	00001000				Priv 00021004	RW	RW
00350000	00001000				Priv 00021004	RW	RW
00360000	00004000				Priv 00021004	RW	RW
00370000	00003000				Map 00041002	R	R
00380000	00002000				Map 00041002	R	R
00390000	00004000				Priv 00021004	RW	RW
003A0000	00002000				Map 00041002	R	R
003B0000	00002000				Map 00041002	R	R
003C0000	00001000				Priv 00021040	RWE	RWE
00400000	00001000	showstri		PE header	Imag 01001002	R	RWE
00401000	00005000	showstri	.text	code	Imag 01001002	R	RWE
00406000	00020000	showstri	.bss	code	Imag 01001002	R	RWE
00426000	00001000	showstri	.data	code,data	Imag 01001002	R	RWE
00427000	00001000	showstri	.idata	code,imports	Imag 01001002	R	RWE
00428000	00002000	showstri	.rsrc	code,resources	Imag 01001002	R	RWE
00430000	00003000				Map 00041020	R E	R E
004F0000	00002000				Map 00041020	R E	R E
00500000	00103000				Map 00041002	R	R
00610000	00073000				Map 00041020	R E	R E
009EF000	00021000				Priv 00021104	RW Guarded	RW
5D090000	00001000	COMCTL32		PE header	Imag 01001002	R	RWE
5D091000	00071000	COMCTL32	.text	code,imports,exports	Imag 01001002	R	RWE
5D102000	00003000	COMCTL32	.data	code,data	Imag 01001002	R	RWE
5D105000	00020000	COMCTL32	.rsrc	code,resources	Imag 01001002	R	RWE
5D125000	00005000	COMCTL32	.reloc	code,relocations	Imag 01001002	R	RWE
73D90000	00001000	CRTDLL		PE header	Imag 01001002	R	RWE
73D91000	0001D000	CRTDLL	.text	code,imports,exports	Imag 01001002	R	RWE
73DAE000	00006000	CRTDLL	.data	code,data	Imag 01001002	R	RWE
73DB4000	00001000	CRTDLL	.rsrc	code,resources	Imag 01001002	R	RWE
73DB5000	00002000	CRTDLL	.reloc	code,relocations	Imag 01001002	R	RWE
76390000	00001000	IMM32		PE header	Imag 01001002	R	RWE
76391000	00015000	IMM32	.text	code,imports,exports	Imag 01001002	R	RWE
763A6000	00001000	IMM32	.data	code,data	Imag 01001002	R	RWE
763A7000	00005000	IMM32	.rsrc	code,resources	Imag 01001002	R	RWE
763AC000	00001000	IMM32	.reloc	code,relocations	Imag 01001002	R	RWE
763B0000	00001000	COMDLG32		PE header	Imag 01001002	R	RWE
763B1000	00030000	COMDLG32	.text	code,imports,exports	Imag 01001002	R	RWE
763E1000	00004000	COMDLG32	.data	code,data	Imag 01001002	R	RWE
763E5000	00011000	COMDLG32	.rsrc	code,resources	Imag 01001002	R	RWE
763F6000	00003000	COMDLG32	.reloc	code,relocations	Imag 01001002	R	RWE
773D0000	00001000	comctl_1		PE header	Imag 01001002	R	RWE
773D1000	00091000	comctl_1	.text	code,imports,exports	Imag 01001002	R	RWE
77462000	00001000	comctl_1	.data	code,data	Imag 01001002	R	RWE
77463000	0006A000	comctl_1	.rsrc	code,resources	Imag 01001002	R	RWE
774CD000	00006000	comctl_1	.reloc	code,relocations	Imag 01001002	R	RWE
774E0000	00001000	ole32		PE header	Imag 01001002	R	RWE
774E1000	00120000	ole32	.text	code,imports,exports	Imag 01001002	R	RWE
77601000	00006000	ole32	.orpc	code	Imag 01001002	R	RWE
77607000	00007000	ole32	.data	code,data	Imag 01001002	R	RWE
7760E000	00002000	ole32	.rsrc	code,resources	Imag 01001002	R	RWE
77610000	0000E000	ole32	.reloc	code,relocations	Imag 01001002	R	RWE
77C10000	00001000	msvcrt		PE header	Imag 01001002	R	RWE
77C11000	0004C000	msvcrt	.text	code,imports,exports	Imag 01001002	R	RWE

The memory window displays all of the memory blocks that the program has allocated. It includes the main sections of the running app (in this case, the "Showstr " items in the Owner column. You can also see a lot of other sections down the list; these are DLL's that the program has loaded into memory and plans on using. If you double-click on any of these lines, a window will open showing a disassembly (or hex dump) of that section. This window also shows the type of block, the access rights, the size and the memory address where the section is loaded.

2. (P)atches

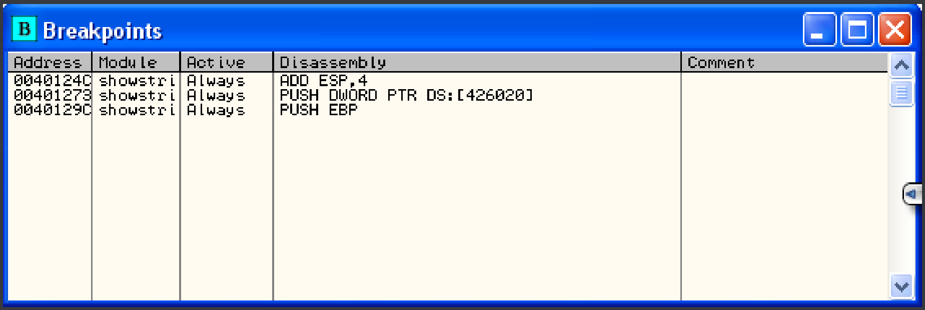


Patches

Address	Size	State	Old	New
00401239	7.	Active	PUSH EBX	NOP
0040124F	2.	Active	PUSH 0	PUSH 10
00401267	6.	Active	PUSH DWORD PTR DS:[426028]	JMP SHORT showstri.00401239

This window displays any "patches" you have made, ie. any changes to the original code. Notice that the state is set as Active; if you re-load the app (by clicking the re-load icon) these patches will become disabled. In order to re-enable them (or disable them) simply click on the desired patch and hit the spacebar. This toggles the patch on/off. Also notice that in the "Old" and "New" columns it shows the original instructions as well as the changed instructions.

3. (B)reakpoints



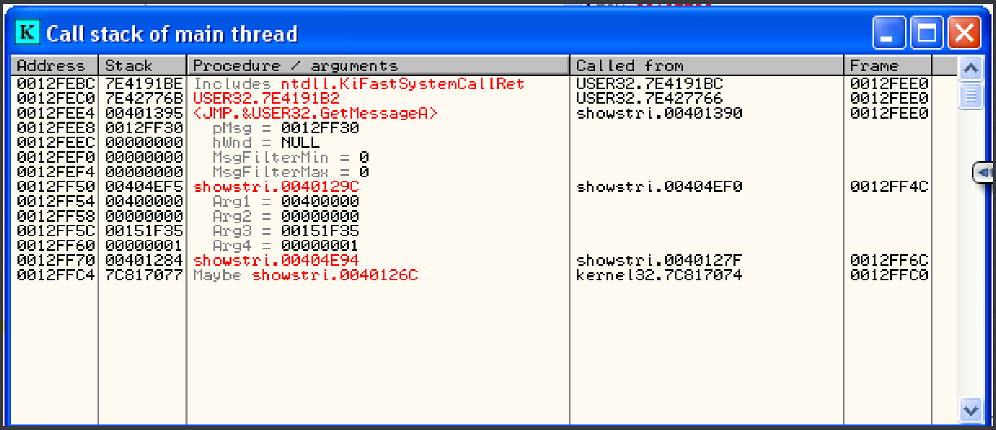
Breakpoints

Address	Module	Active	Disassembly	Comment
0040124C	showstri	Always	ADD ESP,4	
00401273	showstri	Always	PUSH DWORD PTR DS:[426020]	
0040129C	showstri	Always	PUSH EBP	

This window shows where all of the current breakpoints are set. This window will be your friend 😀

3. (K)all Stack

(Gee, I wonder why beginners have a hard time remembering these icons…)

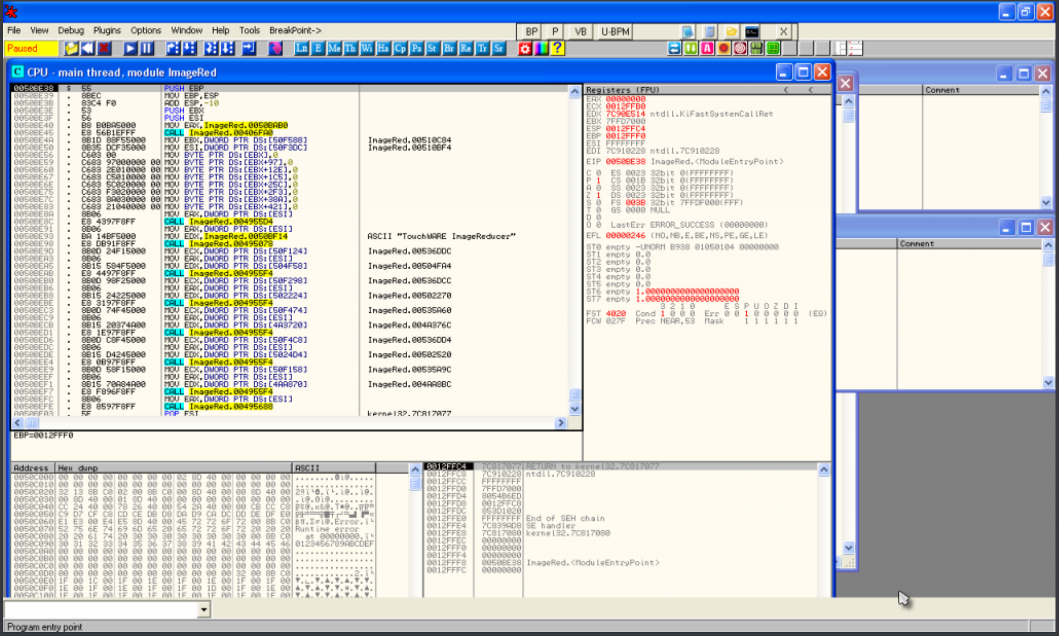


Call stack of main thread

Address	Stack	Procedure / arguments	Called from	Frame
0012FEBC	7E4191BE	Includes ntdll.KiFastSystemCallRet	USER32.7E4191BC	0012FEE0
0012FEC0	7E42776B	USER32.7E4191B2	USER32.7E427766	0012FEE0
0012FEE4	00401395	<JMP.&USER32.GetMessageA>	showstri.00401390	0012FEE0
0012FEE8	0012FF30	pMsg = 0012FF30		
0012FEEC	00000000	hWnd = NULL		
0012FEF0	00000000	MsgFilterMin = 0		
0012FEF4	00000000	MsgFilterMax = 0		
0012FF50	00404EF5	showstri.0040129C	showstri.00404EF0	0012FF4C
0012FF54	00400000	Arg1 = 00400000		
0012FF58	00000000	Arg2 = 00000000		
0012FF5C	00151F35	Arg3 = 00151F35		
0012FF60	00000001	Arg4 = 00000001		
0012FF70	00401284	showstri.00404E94	showstri.0040127F	0012FF6C
0012FFC4	7C817077	Maybe showstri.0040126C	kernel32.7C817074	0012FFC0

This window is different from the "Stack" see earlier. It shows a lot more info about calls being made in the code, the values sent to those functions, and more. We will see more of this shortly.

* In the next tutorial I will be including my version of Olly with many 'upgrades', some of which are buttons that you can actually understand. Here, you can see a picture of it *


File View Debug Plugins Options Window Help Tools BreakPoint->
Paused
BP P VB U-BPM
CPU - main thread, module ImageRed
Program entry point

```
0050BE38  $  55               PUSH EBP
0050BE39  .  8BEC             MOV EBP,ESP
0050BE3B  .  83C4 F0          ADD ESP,-10
0050BE3E  .  53               PUSH EBX
0050BE3F  .  56               PUSH ESI
0050BE40  .  B8 B0BA5000      MOV EAX,ImageRed.0050BAB0
0050BE45  .  E8 56B1EFFF      CALL ImageRed.00406FA0
0050BE4A  .  8B1D 88F55000    MOV EBX,DWORD PTR DS:[50F588]       ImageRed.00510C84
0050BE50  .  8B35 DCF35000    MOV ESI,DWORD PTR DS:[50F3DC]       ImageRed.00510BF4
0050BE56  .  C603 00          MOV BYTE PTR DS:[EBX],0
0050BE59  .  C683 97000000 00 MOV BYTE PTR DS:[EBX+97],0
0050BE60  .  C683 2E010000 00 MOV BYTE PTR DS:[EBX+12E],0
0050BE67  .  C683 C5010000 00 MOV BYTE PTR DS:[EBX+1C5],0
0050BE6E  .  C683 5C020000 00 MOV BYTE PTR DS:[EBX+25C],0
0050BE75  .  C683 F3020000 00 MOV BYTE PTR DS:[EBX+2F3],0
0050BE7C  .  C683 8A030000 00 MOV BYTE PTR DS:[EBX+38A],0
0050BE83  .  C683 21040000 00 MOV BYTE PTR DS:[EBX+421],0
0050BE8A  .  8B06             MOV EAX,DWORD PTR DS:[ESI]
0050BE8C  .  E8 4397F8FF      CALL ImageRed.004955D4
0050BE91  .  8B06             MOV EAX,DWORD PTR DS:[ESI]
0050BE93  .  BA 14BF5000      MOV EDX,ImageRed.0050BF14           ASCII "TouchWARE ImageReducer"
0050BE98  .  E8 DB91F8FF      CALL ImageRed.00495078
0050BE9D  .  8B0D 24F15000    MOV ECX,DWORD PTR DS:[50F124]       ImageRed.00536DDC
0050BEA3  .  8B06             MOV EAX,DWORD PTR DS:[ESI]
0050BEA5  .  8B15 584F5000    MOV EDX,DWORD PTR DS:[504F58]       ImageRed.00504FA4
0050BEAB  .  E8 4497F8FF      CALL ImageRed.004955F4
0050BEB0  .  8B0D 98F25000    MOV ECX,DWORD PTR DS:[50F298]       ImageRed.00536DCC
0050BEB6  .  8B06             MOV EAX,DWORD PTR DS:[ESI]
0050BEB8  .  8B15 24225000    MOV EDX,DWORD PTR DS:[502224]       ImageRed.00502270
0050BEBE  .  E8 3197F8FF      CALL ImageRed.004955F4
0050BEC3  .  8B0D 74F45000    MOV ECX,DWORD PTR DS:[50F474]       ImageRed.00535A60
0050BEC9  .  8B06             MOV EAX,DWORD PTR DS:[ESI]
0050BECB  .  8B15 20374A00    MOV EDX,DWORD PTR DS:[4A3720]       ImageRed.004A376C
0050BED1  .  E8 1E97F8FF      CALL ImageRed.004955F4
0050BED6  .  8B0D C8F45000    MOV ECX,DWORD PTR DS:[50F4C8]       ImageRed.00536DD4
0050BEDC  .  8B06             MOV EAX,DWORD PTR DS:[ESI]
0050BEDE  .  8B15 D4245000    MOV EDX,DWORD PTR DS:[5024D4]       ImageRed.00502520
0050BEE4  .  E8 0B97F8FF      CALL ImageRed.004955F4
0050BEE9  .  8B0D 58F15000    MOV ECX,DWORD PTR DS:[50F158]       ImageRed.00535A9C
0050BEEF  .  8B06             MOV EAX,DWORD PTR DS:[ESI]
0050BEF1  .  8B15 70A84A00    MOV EDX,DWORD PTR DS:[4AA870]       ImageRed.004AA8BC
0050BEF7  .  E8 F896F8FF      CALL ImageRed.004955F4
0050BEFC  .  8B06             MOV EAX,DWORD PTR DS:[ESI]
0050BEFE  .  E8 8597F8FF      CALL ImageRed.00495688
0050BF03  .  5F               POP ESI                             kernel32.7C817077
EBP=0012FFF0
```

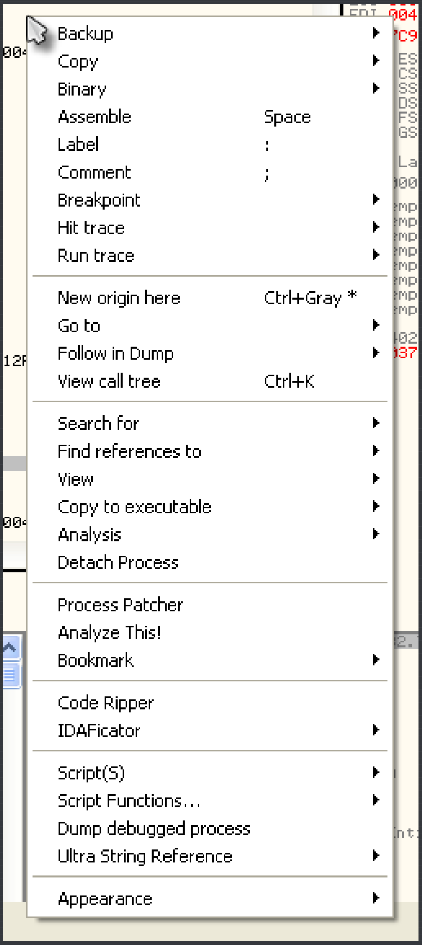
```
Registers (FPU)
EAX 00000000
ECX 0012FFB0
EDX 7C90E514 ntdll.KiFastSystemCallRet
EBX 7FFD7000
ESP 0012FFC4
EBP 0012FFF0
ESI FFFFFFFF
EDI 7C910228 ntdll.7C910228
EIP 0050BE38 ImageRed.<ModuleEntryPoint>
C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 1  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDF000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (00000000)
EFL 00000246 (NO,NB,E,BE,NS,PE,GE,LE)
ST0 empty -UNORM B938 01050104 00000000
ST1 empty 0.0
ST2 empty 0.0
ST3 empty 0.0
ST4 empty 0.0
ST5 empty 0.0
ST6 empty 1.0000000000000000000
ST7 empty 1.0000000000000000000
               3 2 1 0      E S P U O Z D I
FST 4020  Cond 1 0 0 0  Err 0 0 1 0 0 0 0 0  (EQ)
FCW 027F  Prec NEAR,53  Mask    1 1 1 1 1 1
```

```
Address   Hex dump                                         ASCII
0050C000  00 00 00 00 00 00 00 00 02 8D 40 00 00 00 00 00  .........@.@.....
0050C010  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0050C020  32 13 8B C0 02 00 8B C0 00 8D 40 00 00 8D 40 00  2‼ï└@.ï└.ì@..ì@.
0050C030  00 8D 40 00 01 8D 40 00 00 00 00 00 00 00 00 00  .ì@.Oì@.........
0050C040  CC 24 40 00 78 26 40 00 54 2A 40 00 00 CB CC C8  ╠$@.x&@.T*@..╦╠╚
0050C050  C9 D7 CF C8 CD CE DB D8 DA D9 CA DC DD DE DF E0  ╔╫╧╚═╬█╪┌┘╩▄▌▐▀α
0050C060  E1 E3 00 E4 E5 8D 40 00 45 72 72 6F 72 00 8B C0  ßπ.Σσì@.Error.ï└
0050C070  52 75 6E 74 69 6D 65 20 65 72 72 6F 72 20 20 20  Runtime error
0050C080  20 20 61 74 20 30 30 30 30 30 30 30 30 00 8B C0    at 00000000.ï└
0050C090  30 31 32 33 34 35 36 37 38 39 41 42 43 44 45 46  0123456789ABCDEF
0050C0A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0050C0B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0050C0C0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0050C0D0  00 00 00 00 00 00 00 00 00 00 00 00 32 00 8B C0  ............2.ï└
0050C0E0  1F 00 1C 00 1F 00 1E 00 1F 00 1E 00 1F 00 1F 00  ▼.∟.▼.▲.▼.▲.▼.▼.
0050C0F0  1E 00 1F 00 1E 00 1F 00 1F 00 1D 00 1F 00 1E 00  ▲.▼.▲.▼.▼.↔.▼.▲.
```

```
0012FFC4  7C817077  RETURN to kernel32.7C817077
0012FFC8  7C910228  ntdll.7C910228
0012FFCC  FFFFFFFF
0012FFD0  7FFD7000
0012FFD4  8054B6ED
0012FFD8  0012FFC8
0012FFDC  853D1020
0012FFE0  FFFFFFFF  End of SEH chain
0012FFE4  7C839AD8  SE handler
0012FFE8  7C817080  kernel32.7C817080
0012FFEC  00000000
0012FFF0  00000000
0012FFF4  00000000
0012FFF8  0050BE38  ImageRed.<ModuleEntryPoint>
0012FFFC  00000000
```

The Context Menu

For the last item of this tutorial, I wanted to quickly introduce you to the right-click menu in Olly. It is where a lot of action happens, so you should at least be familiar with it. Right-clicking anywhere in the disassembly section brings it up:



I will only go over the most popular items now. As you gain experience, you will end up using some of the less used options.

“Binary” allows editing of the binary data on a byte-by-byte level. This is where you may change a “Unregistered” string buried in a binary to “Registered” 😀 . “Breakpoint” allows you to set a breakpoint.

There are several types of breakpoints and we will be going over them in the next tutorial. “Search For” is a rather large sub-menu, and it’s where you search the binary for data such as strings, function calls etc. “Analysis” forces Olly to re-analyze the section of code you are currently viewing. Sometimes Olly gets confused as to whether you are viewing code or data (remember, they’re both just numbers) so this forces Olly to consider where you are in the code and attempt to guess what this section should look like.

Also notice that my menu will look different from yours in that I have some plugins installed and they add some functionality. Don’t worry, we will be going over all of these in future tuts.

-Well, till next time.

R4ndom