Sri Lanka Institute of Information Technology

Offensive Computer Security

Year 4, Semester 1– 2016

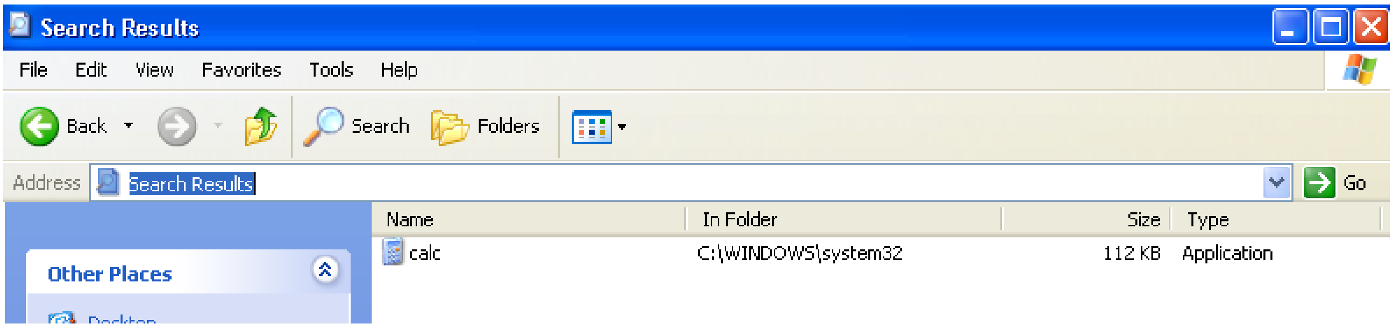
Ollydebug Exercises

V.V.Y.Wickramanayake

IT 13106966

Step 01:

1. Run XN Resource Editor:



Name	In Folder	Size	Type
calc	C:\WINDOWS\system32	112 KB	Application

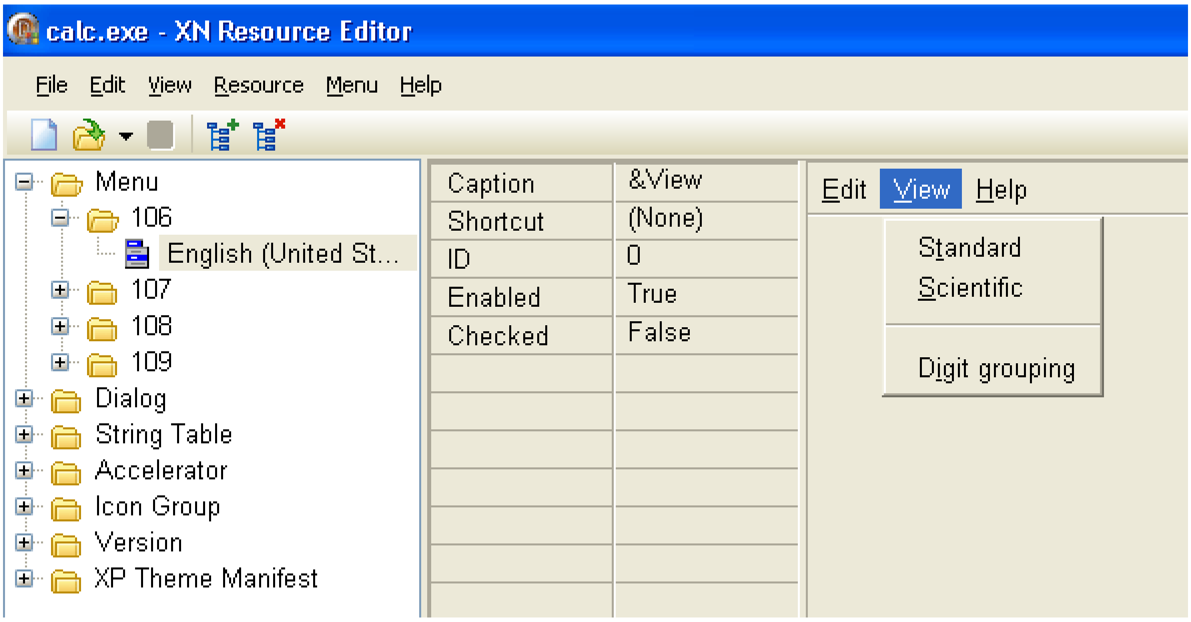
2. Click on the load icon on top, and click over to Windows\System32\ and load calc.exe

3. Click on the plus next to Menu

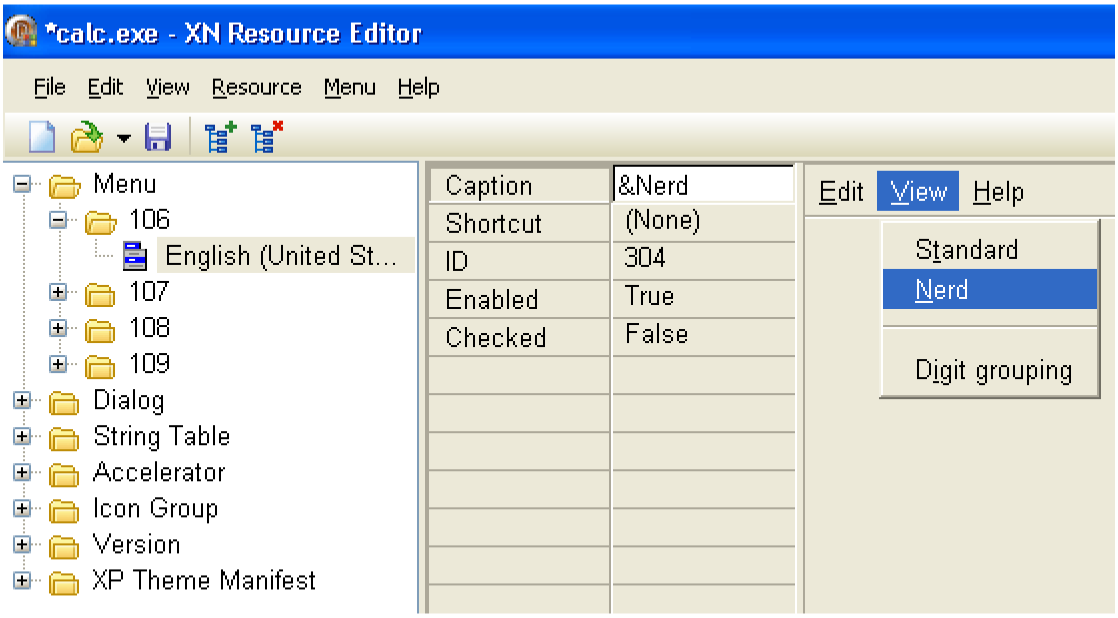
You will then see a folder with a number as a name. This is the ID that windows will use to access this resource in the program. Open this folder as well. You should now see an icon for "English (United States)" or something like this. If you click on this you will see a diagram of what the menu will look like (you can even click around- it works just like a real menu).

4. Click on the menu option "Scientific":

The Caption field should change to "&Scientific". The ampersand is there to tell you what the 'Hot-Key' is, in this case 'S'. If instead we wanted the 'e' to be the hot-key, it would look like this "Sci&entific".



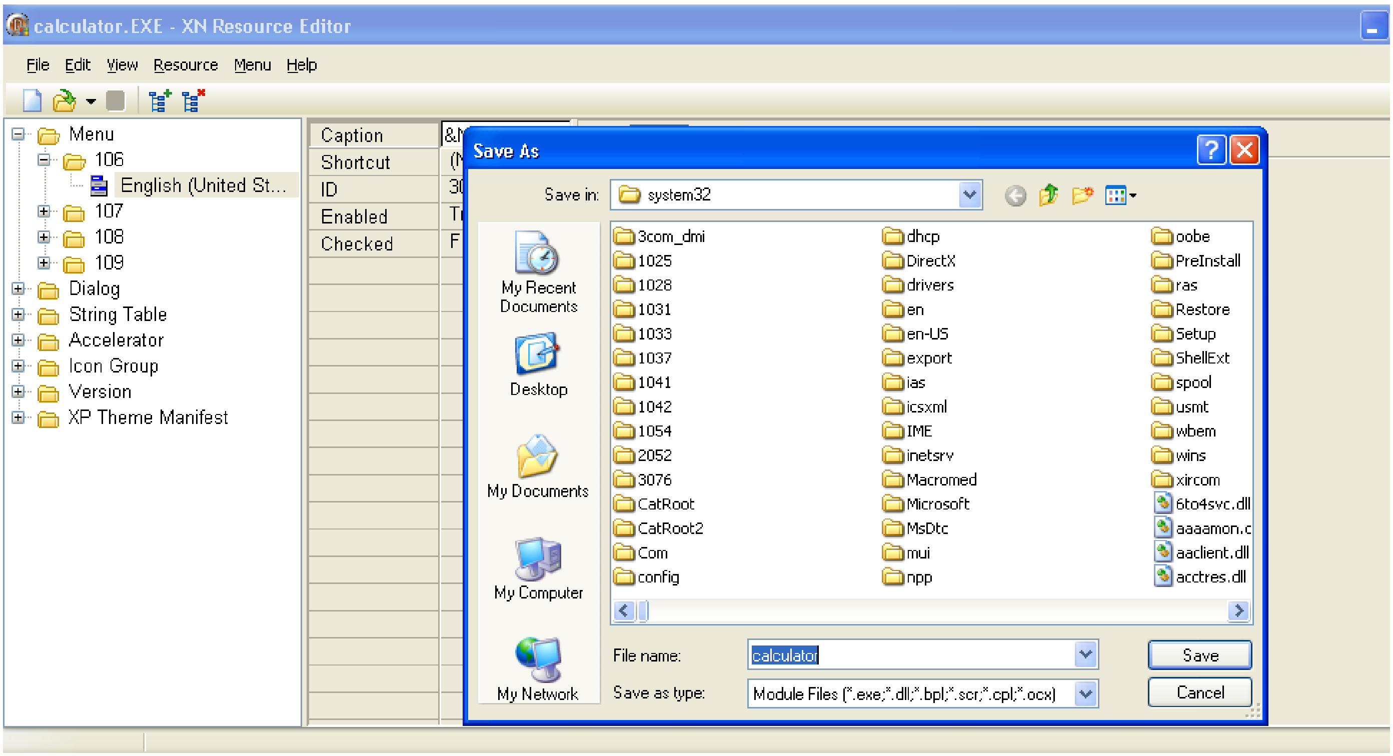
Caption	&View
Shortcut	(None)
ID	0
Enabled	True
Checked	False



Caption	&Nerd
Shortcut	(None)
ID	304
Enabled	True
Checked	False

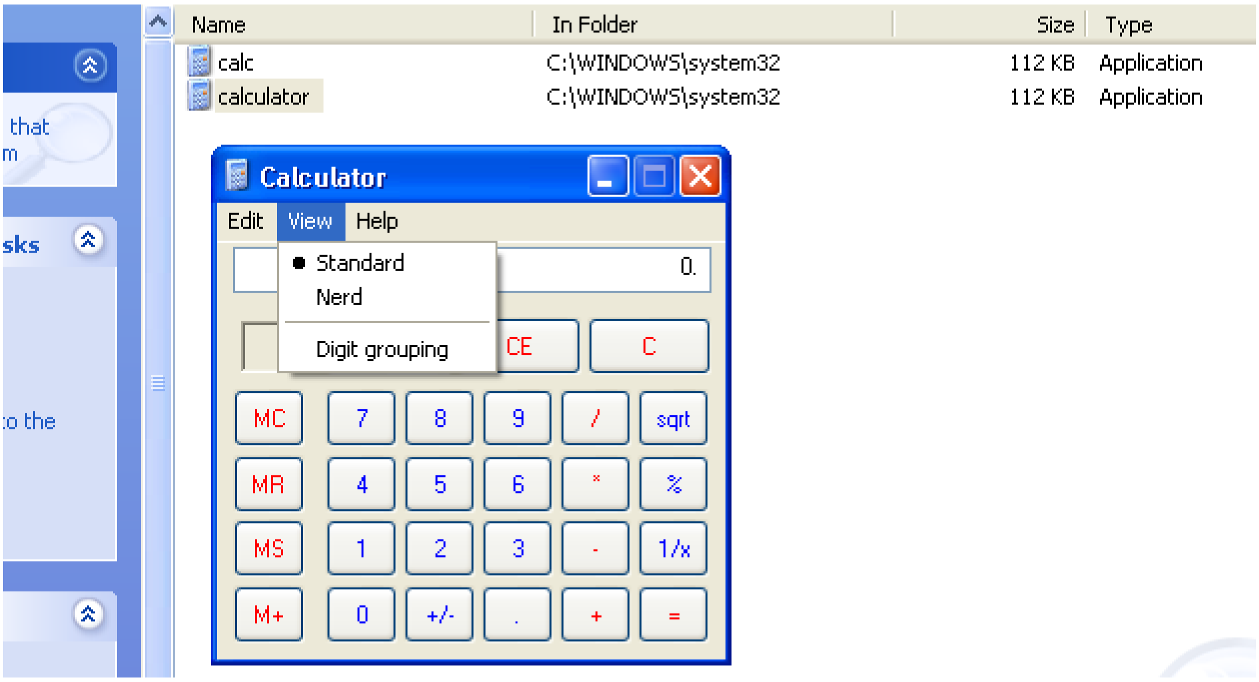
5. Go up to File (in XN Resource) and choose “Save As…”:

Save your new version of calc to a different name (and preferably a different location) and then run it.

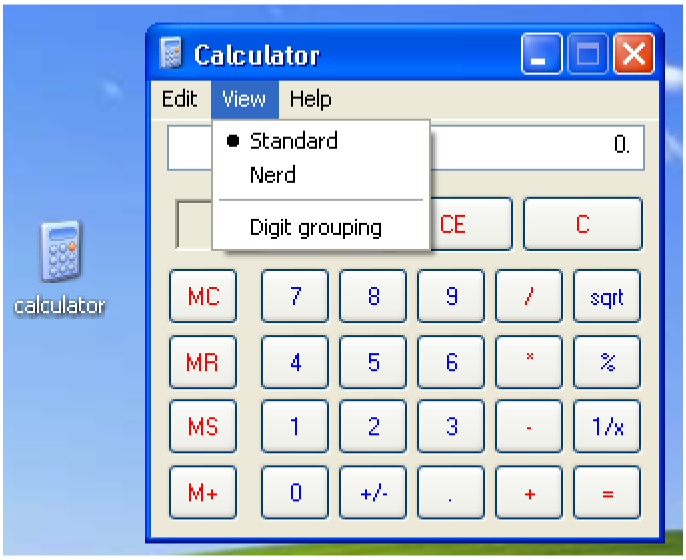


File name: calculator

Save as type: Module Files (*.exe;*.dll;*.bpl;*.scr;*.cpl;*.ocx)



Name	In Folder	Size	Type
calc	C:\WINDOWS\system32	112 KB	Application
calculator	C:\WINDOWS\system32	112 KB	Application



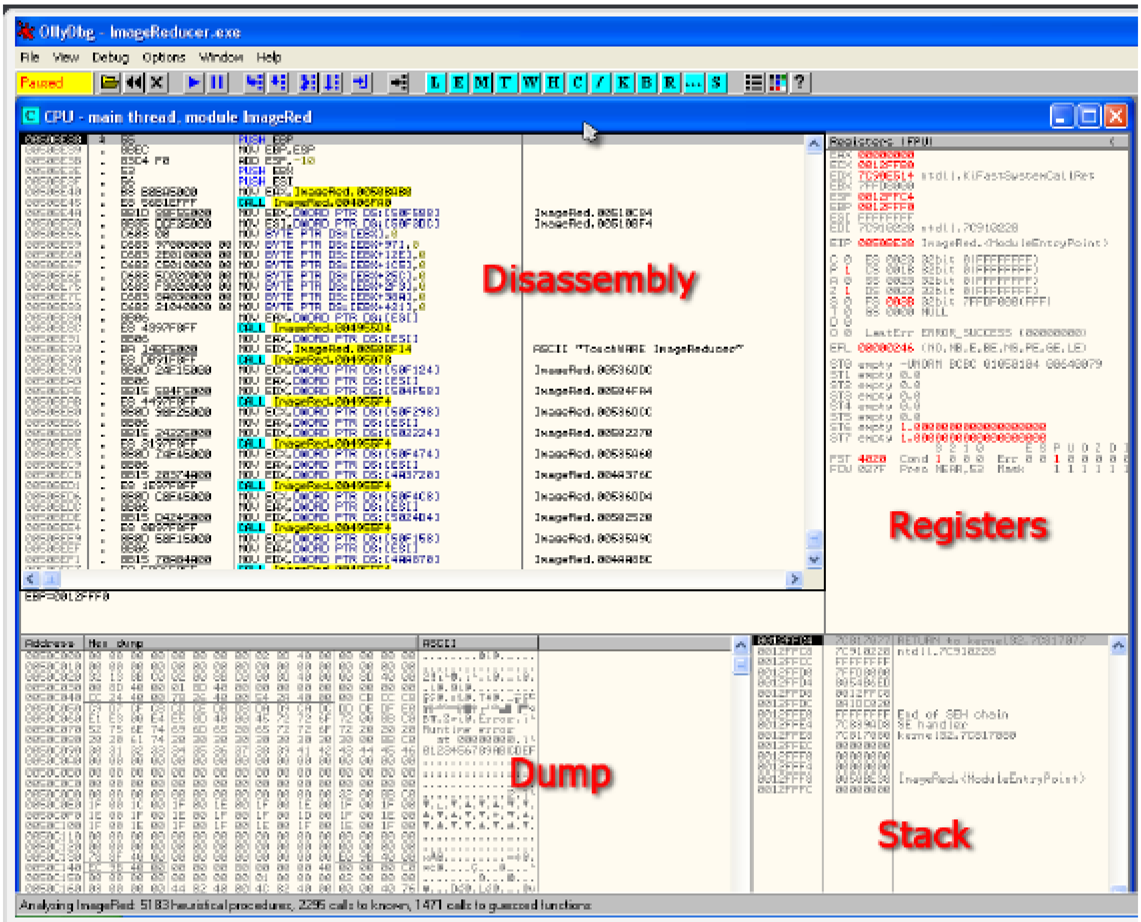
Step 02: Introduction to Olly Debug

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.

An Overview

If the web developer added some regular expressions, to prevent the simple XSS payload from working, we can see that and are filtered. One of the most basic ways to bypass these types of filters is to play with the case: if we try and for example, we should be able to get the alert box.



OllyDbg - ImageReducer.exe
File View Debug Options Window Help
Paused
CPU - main thread, module ImageRed
Disassembly
Registers
Dump
Stack

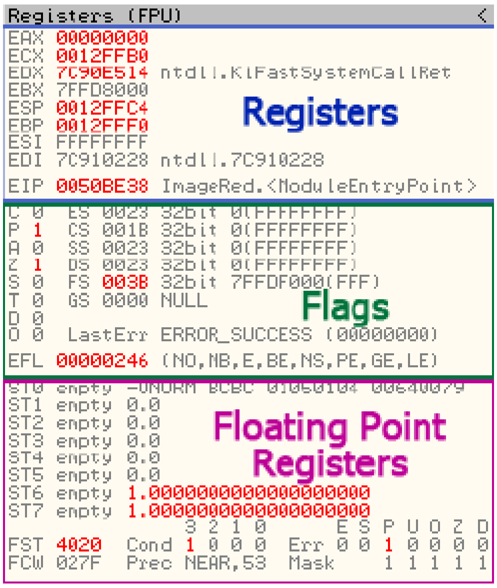
Analysing ImageRed: 5183 heuristical procedures, 2295 calls to known, 1471 calls to guessed functions

Main Olly Display

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 associate 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:



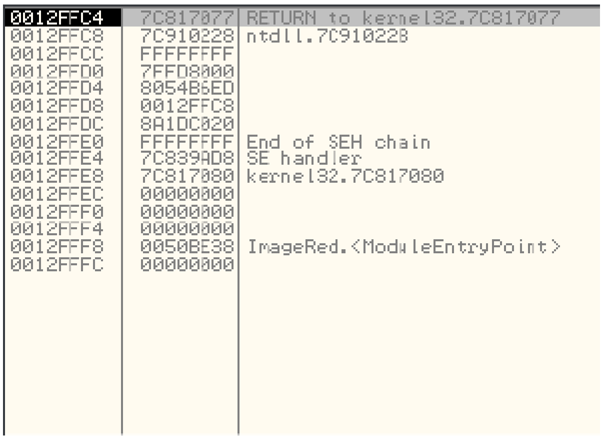
```
EAX 00000000
ECX 0012FFB0
EDX 7C90E514 ntdll.KiFastSystemCallRet
EBX 7FFD8000
ESP 0012FFC4
EBP 0012FFF0
ESI FFFFFFFF
EDI 7C910228 ntdll.7C910228

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 01060104 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
```

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.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
```

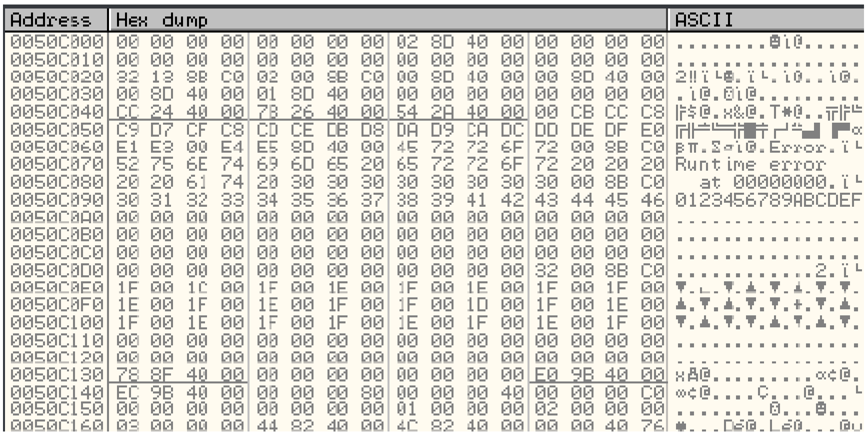
The stack is a section of memory reserved for the binary as a ‘temporary’ list of data. This data includes RSS Feed WordPress.org Subscribe Enter your email to subscribe to future updates pointers to addresses in memory, strings, markers, and most importantly, return addresses for the code to return 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.

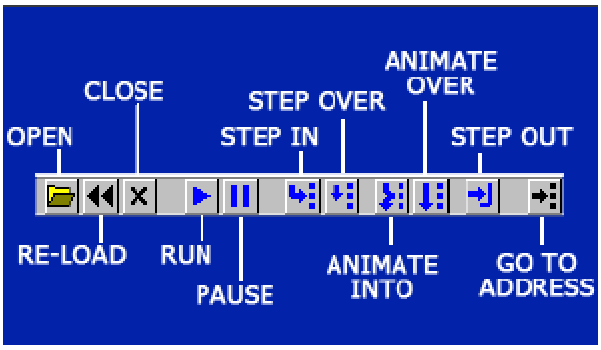
4 The Dump:



```
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
```

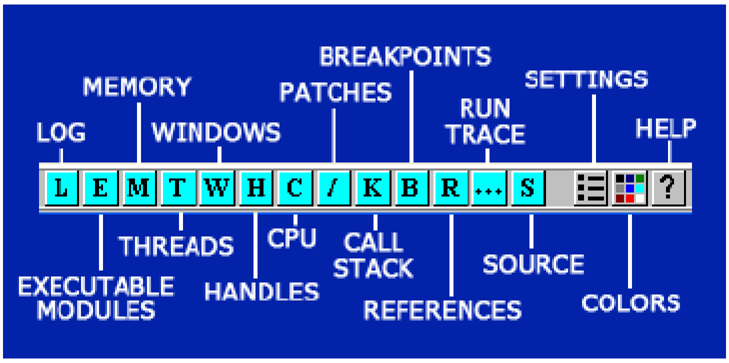
The dump window is a built-in hex viewer that lets you see the raw binary data, only in memory as opposed to on disk. Usually it shows two views of the same data; hexadecimal and ASCII. These are represented in the two right-hand columns in the previous picture (the first column is the address in memory that the data resides.) Olly does allow these representations of data to be changed.

The Toolbar



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.

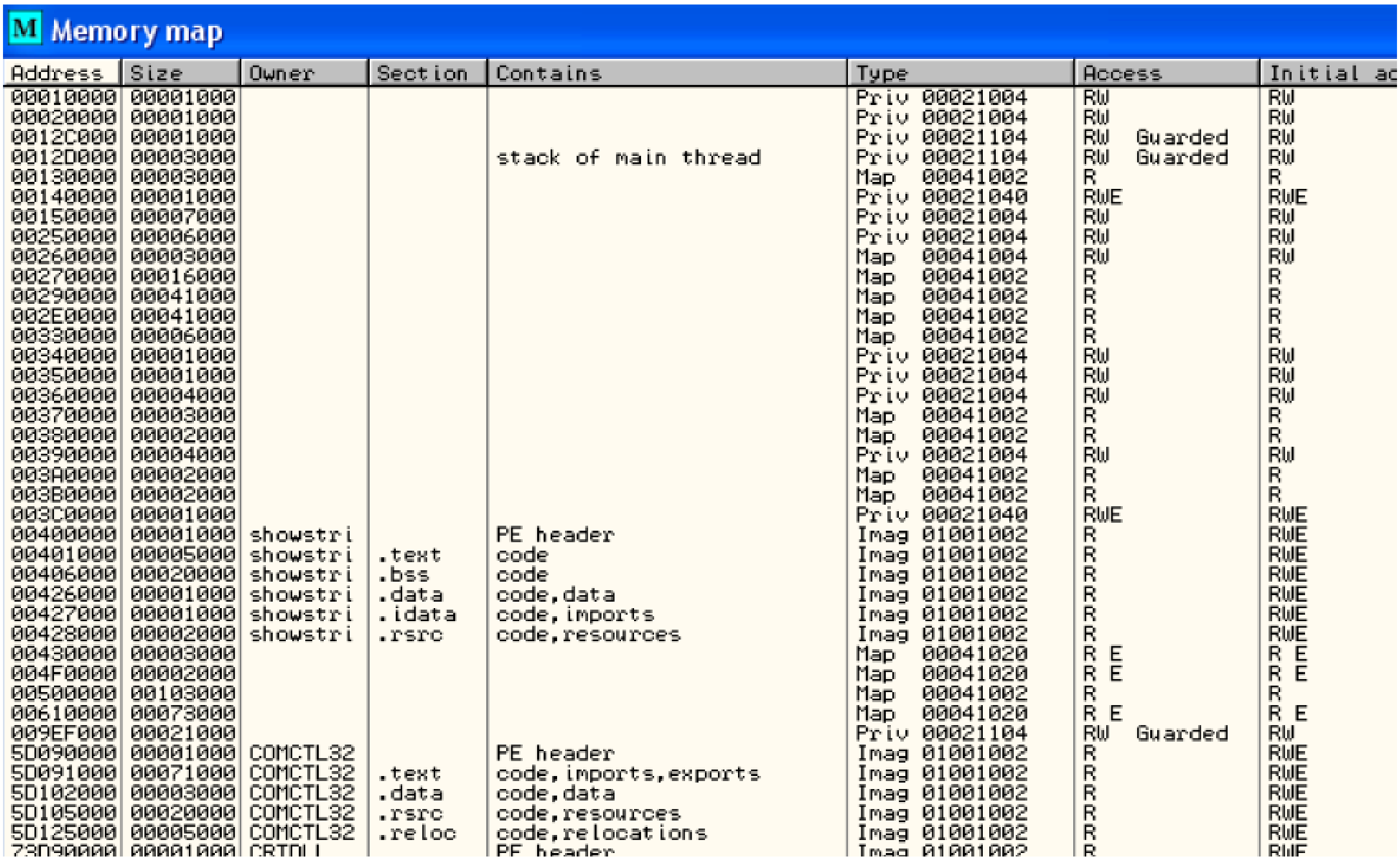
“Re-load” is basically to restart the app and pause it at the entry point. All patches (see later) will be removed, some breakpoints will be disabled, and the app will not have run any code yet, well, most of the time anyway. “Run” and “Pause” do just that. “Step In” means run one line of code and then 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.



Each of these icons opens a window, some of which you will use often, some rarely.

1. (M)emory :

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.

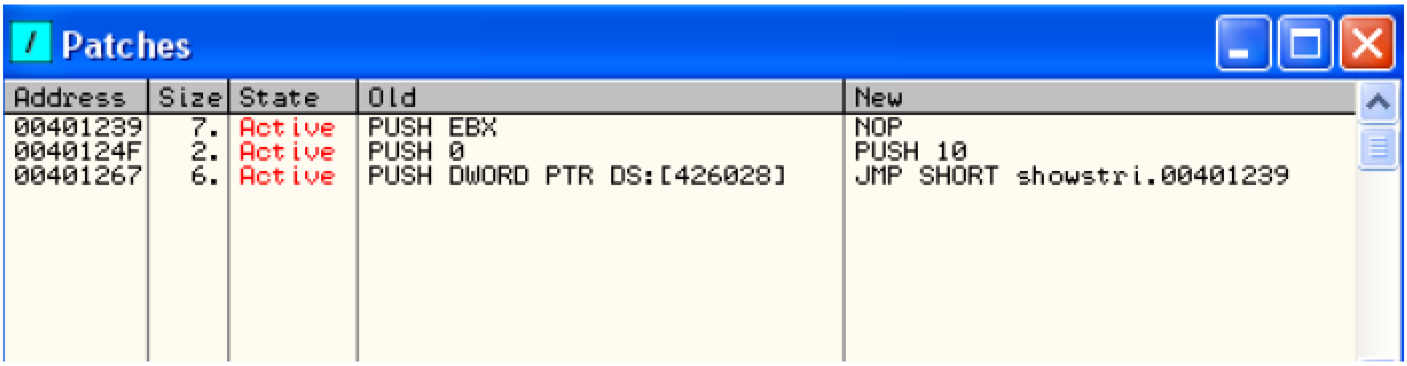


Memory map

```
Address   Size      Owner      Section   Contains                  Type           Access          Initial ac
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
```

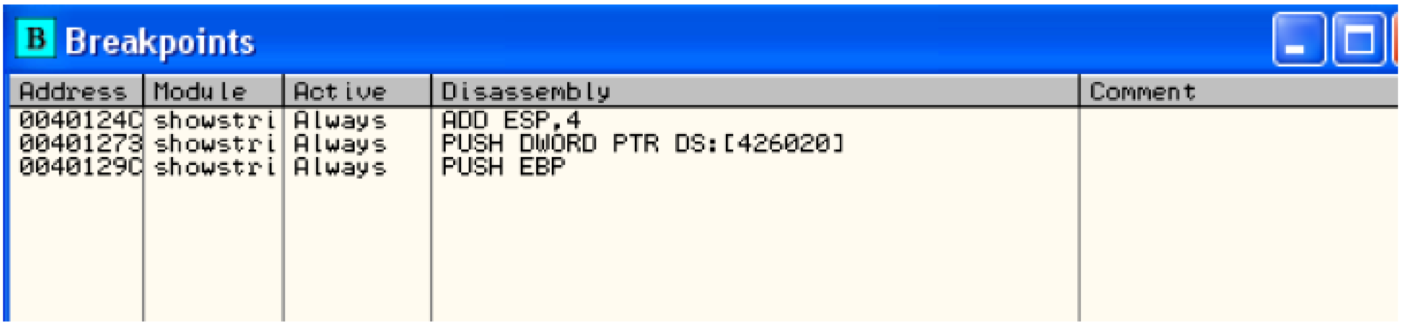
2. Patches :

This window displays any “patches” you have made. 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.



```
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
```

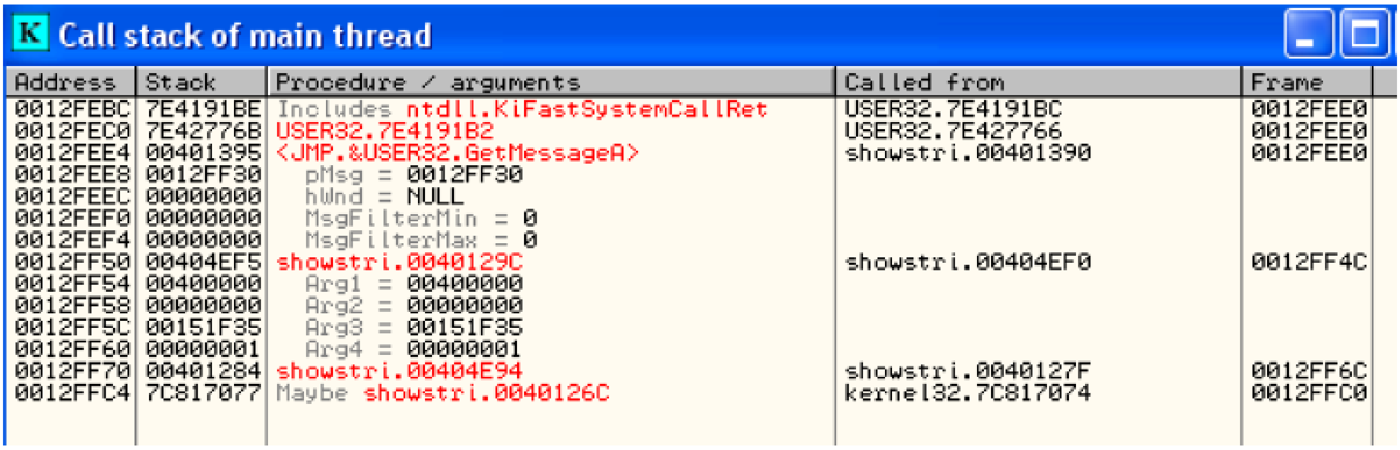
3. (B)reakpoints :



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

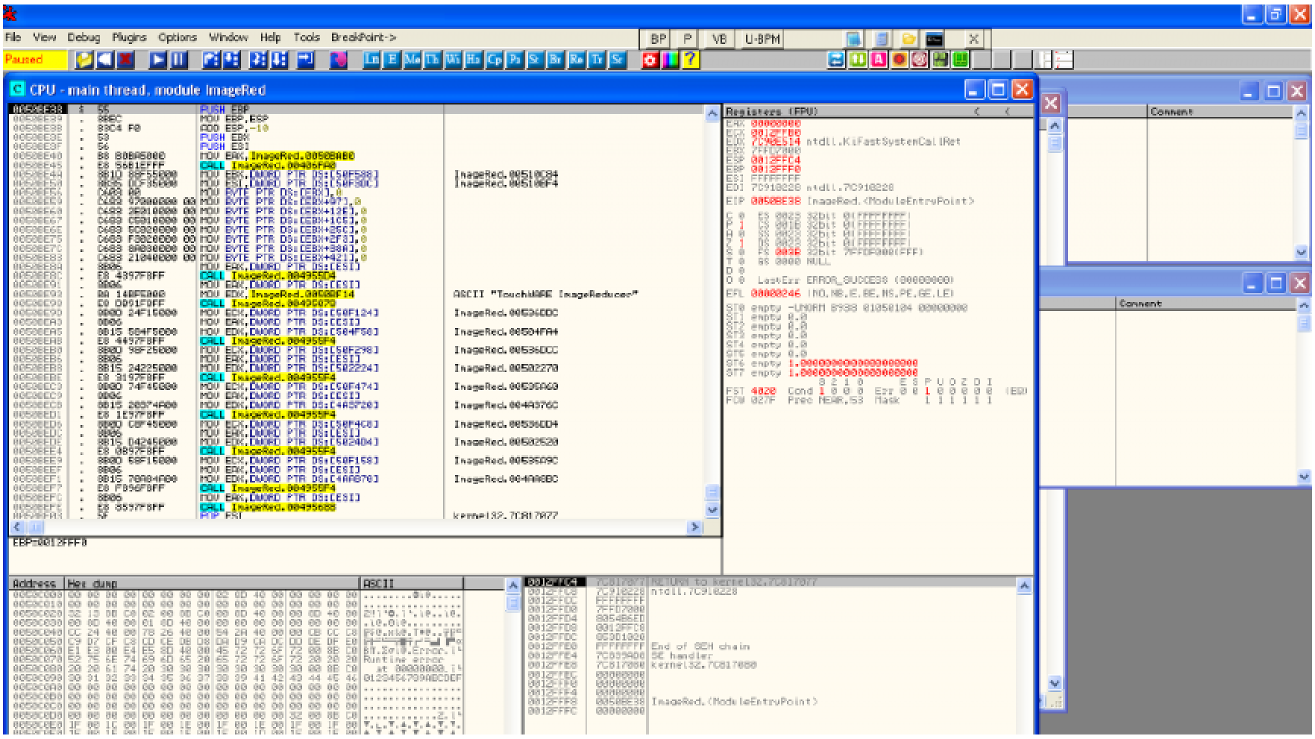
This window shows where all of the current breakpoints are set.

4. (K)all Stack :

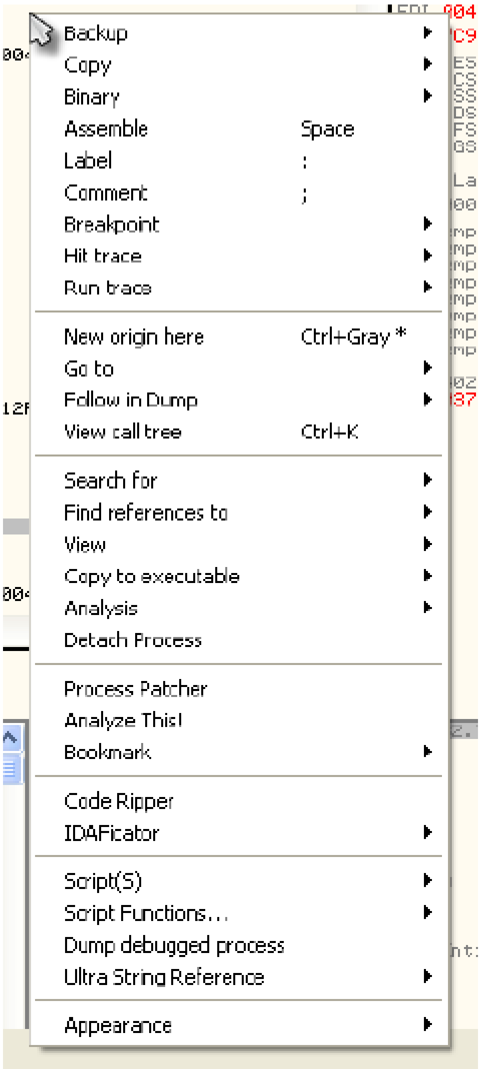


```
Address  Stack    Procedure / arguments                Called from        Frame
0012FEBC 7E4191BE Includes ntdll.KiFastSystemCallRet   USER32.7E4191BC    0012FEE0
0012FEC0 7E42776B USER32.7E4191B2                      USER32.7E427766    0012FEE0
0012FEE4 00401395 <JMP.&USER32.GetMessageA>            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.


File View Debug Plugins Options Window Help Tools BreakPoint->
Paused
CPU - main thread, module ImageRed
Registers (FPU)
Address Hex dump ASCII
Comment

The Context Menu



“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. “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.