The Legend of Random

Task 01

Reverse Engineering

What is Reverse Engineering?

Reverse engineering is the process of taking a compiled binary and attempting to recreate the original way the program works.

What is reverse engineering used for?

Reverse engineering can be applied to many areas of computer science, but here are a couple of generic categories;

1. Making it possible to interface to legacy code (where you do not have the original code source).
2. Breaking copy protection (ie. Impress your friends and save some $$).
3. Studying virii and malware.
4. Evaluating software quality and robustness.
5. Adding functionality to existing software.

The first category is reverse engineering code to interface with existing binaries when the source code is not available.

The second category is the biggest. It disables time trials, defeat registration, and basically everything else to get commercial software for free.

Reverse Engineering needs for studying virus and malware code because not a lot of virus coders out there don't send instructions on how they wrote the code, what it is supposed to accomplish, and how it will accomplish this (unless they are really dumb).

In the fourth category, when creating large systems like Operating Systems, reverse engineering is used to make sure that the system does not contain any major vulnerabilities, security flaws, and frankly, to make it as hard as possible to allow crackers to crack the software.

The final category is adding functionality to existing software. As examples, graphics used in web design software and when want to change it.

What knowledge is required?

Basic understanding of how program flow works
Familiarization with Assembly Language
Reverse engineering Tools

Experimentation (playing with different packers/protectors/encryption schemes, learning about programs originally written in different programming languages (even Delphi), deciphering anti-reverse engineering tricks)

What kinds of tools are used?

1. Disassemblers
Disassemblers attempt to take the machine language codes in the binary and display them in a friendlier format. They also extrapolate data such as function calls, passed variables and text strings.

2. Debuggers
Debuggers first analyze the binary, much like a disassembler Debuggers then allow the reverser to step through the code, running one line at a time and investigating the results. This is invaluable to discover how a program works.

3. Hex editors
Hex editors allow you to view the actual bytes in a binary, and change them. They also provide searching for specific bytes, saving sections of a binary to disk, and much more.

4. PE and resource viewers/editors
Every binary designed to run on a windows machine (and Linux for that matter) has a very specific section of data at the beginning of it that tells the operating system how to set up and initialize the program. It tells the OS how much memory it will require, what support DLLs the program needs to borrow code from, information about dialog boxes and such. This is called the Portable Executable, and all programs designed to run on windows needs to have one.

5. System Monitoring tools
When reversing programs, it is sometimes important (and when studying virii and malware, of the utmost importance) to see what changes an application makes to the system; are there registry keys created or queried? are there .ini files created? are separate processes created, perhaps to thwart reverse engineering of the application? Examples of system monitoring tools are procmon, regshot, and process hacker.

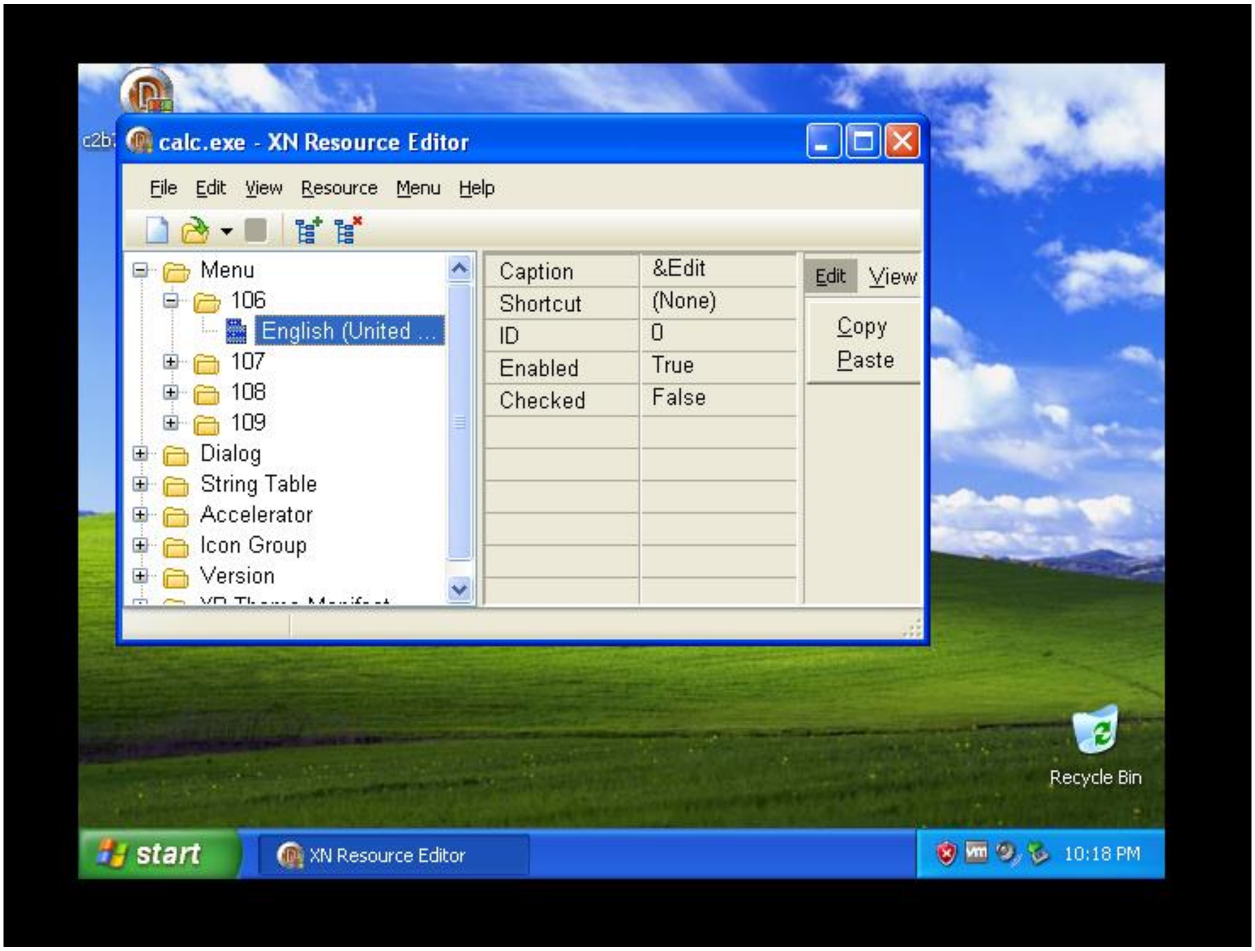
6. Miscellaneous tools and information
There are tools such as scripts, unpackers, packer identifiers etc and also Windows API helps to know exactly what called functions are doing.

7. Beer.

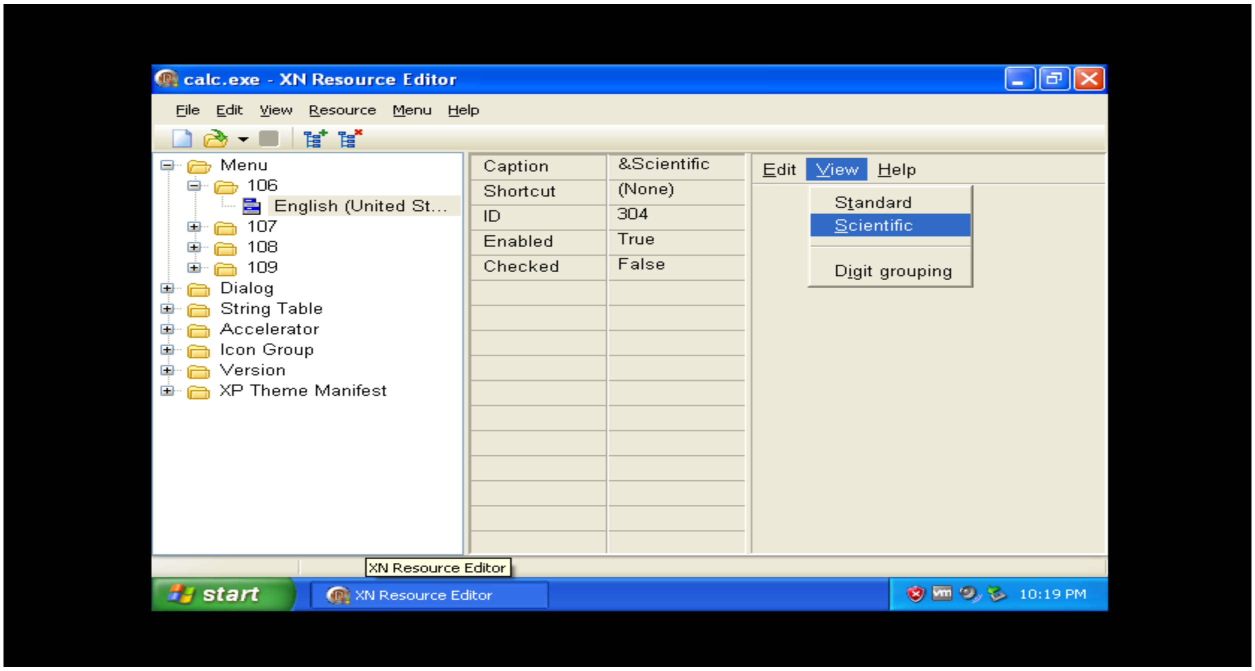
Let’s get Start

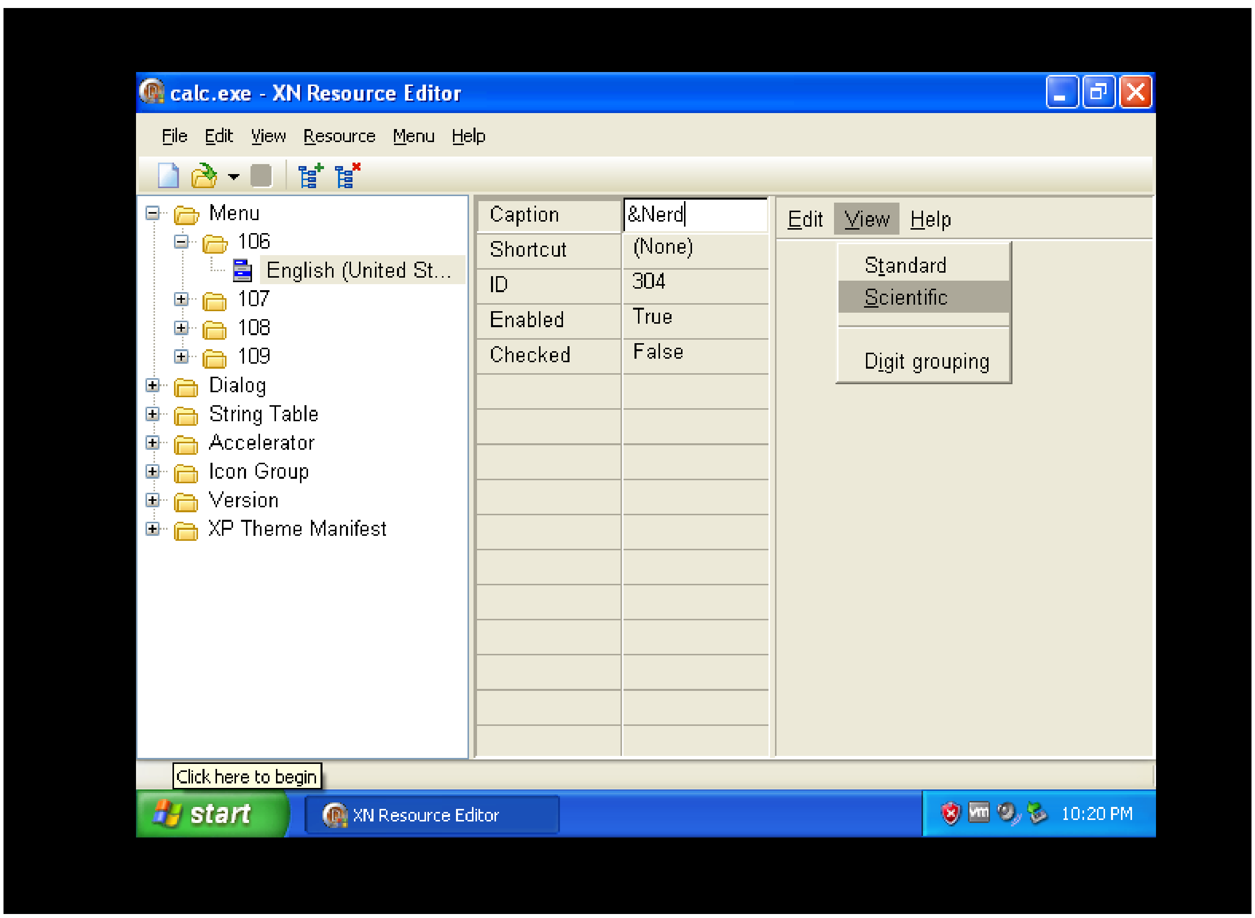
Here I have used a resource viewer/editor called XN Resource Editor. It is freeware. Basically, this program allows to see the resource section in an exe file, as well as modify these resources. In here this file allows to change the menus, icons, graphics, dialogs, you name it, in programs. Let's try one ourselves.
First, run XN. Click on the load icon on top, and click over to Windows\System32\ and load calc.exe. There are bunch of folders available.

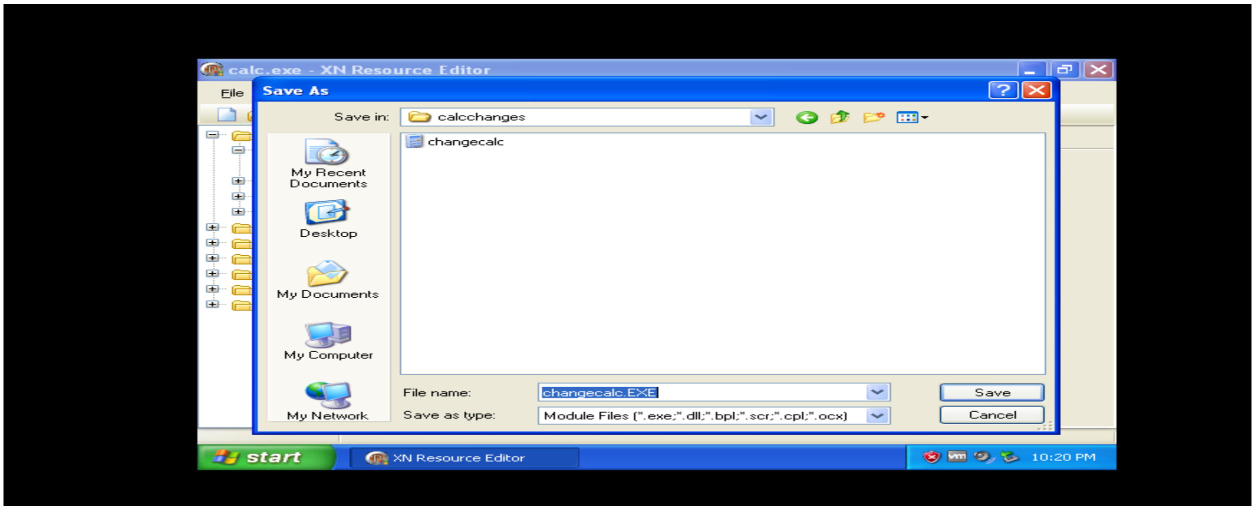
In Windows\System32\ there are folders for Bitmaps, Menu, Dialog, String Table, IconGroup etc. To do changes, click on the plus next to Menu. Then there's a folder with a number as a name. This is the ID that windows will use to access this resource in the program. In side this folder there's an icon for "English (United States)". When click on it and see a diagram of menu.

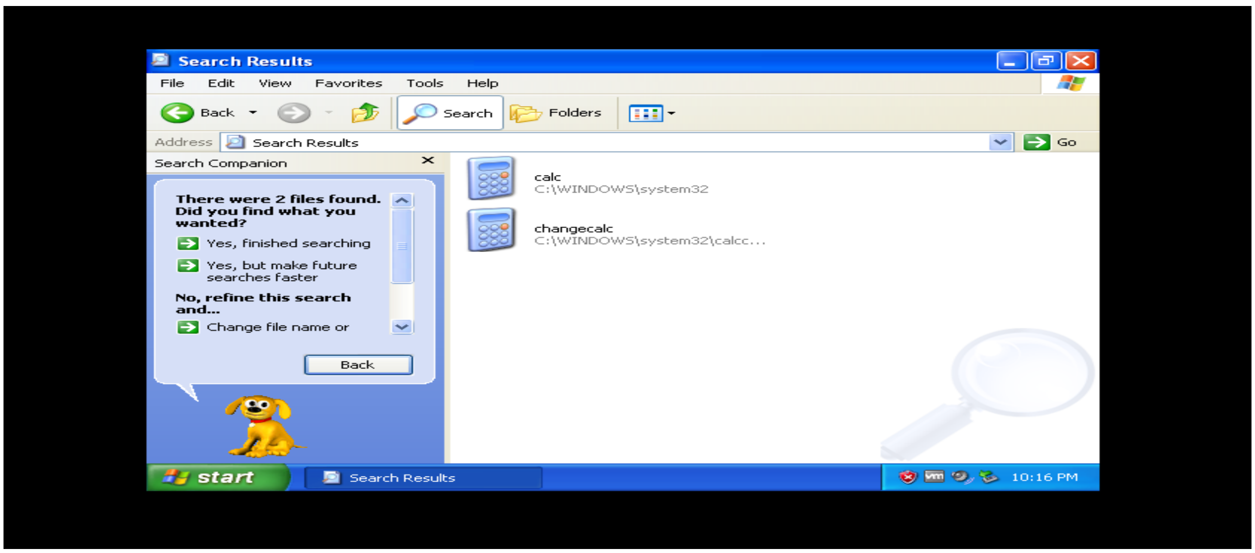


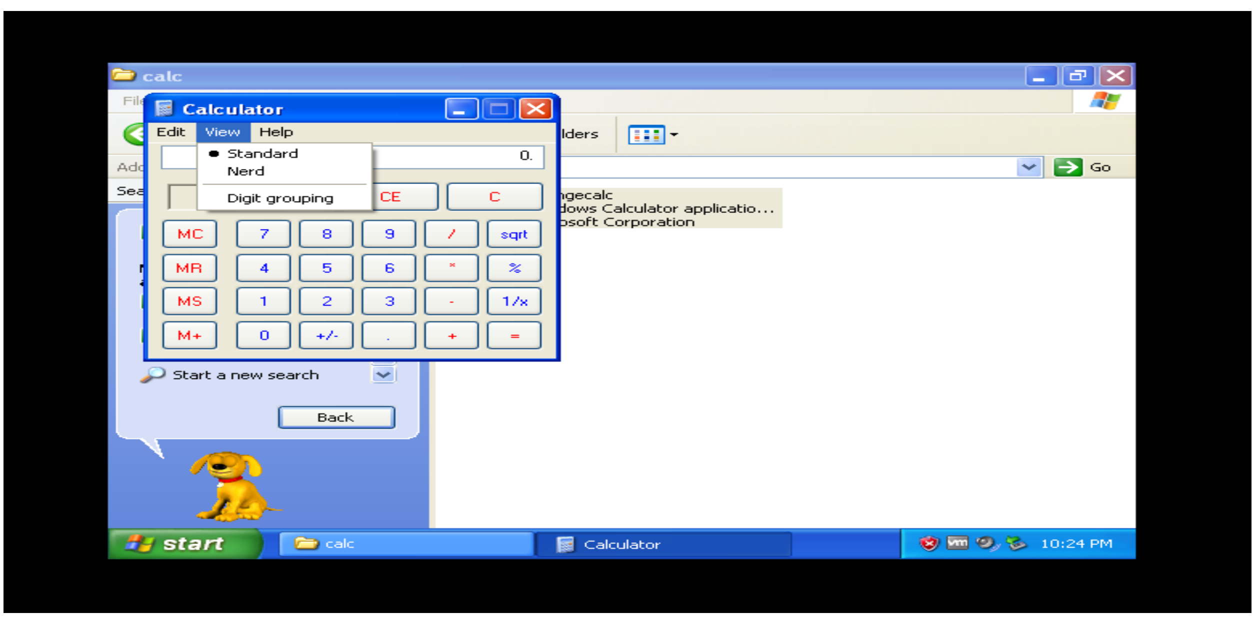
Now, cli ck on the menu option "Scientific". Change caption field to "&Scientific". "&" shows 'Hot-Key' in this case it's "S". If instead wanted the 'e' to be the hot-key, it would look like this "Sci&entific". In the Caption field, replace the &Scientific with "&Nerd". This will now change the menu option to "Nerd" and use the hot-key 'N'. Now, go up to File (in XN Resource) and choose "Save As…" Save new version of calc to "calcchange" and saved in another new folder names as "change calc" and then run it.


calc.exe - XN Resource Editor
File Edit View Resource Menu Help
Menu
106
English (United St...
107
108
109
Dialog
String Table
Accelerator
Icon Group
Version
XP Theme Manifest
Caption
&Scientific
Shortcut
(None)
ID
304
Enabled
True
Checked
False
Edit View Help
Standard
Scientific
Digit grouping
XN Resource Editor
start
XN Resource Editor
10:19 PM


calc.exe - XN Resource Editor
File Edit View Resource Menu Help
Menu
106
English (United St...
107
108
109
Dialog
String Table
Accelerator
Icon Group
Version
XP Theme Manifest
Caption
&Nerd
Shortcut
(None)
ID
304
Enabled
True
Checked
False
Edit View Help
Standard
Scientific
Digit grouping
Click here to begin
start
XN Resource Editor
10:20 PM


calc.exe - XN Resource Editor
Save As
Save in: calcchanges
changecalc
My Recent Documents
Desktop
My Documents
My Computer
My Network
File name: changecalc.EXE
Save as type: Module Files (*.exe;*.dll;*.bpl;*.scr;*.cpl;*.ocx)
Save
Cancel
start
XN Resource Editor
10:20 PM


Search Results
File Edit View Favorites Tools Help
Back Search Folders
Address Search Results
Go
Search Companion
There were 2 files found. Did you find what you wanted?
Yes, finished searching
Yes, but make future searches faster
No, refine this search and...
Change file name or
Back
calc
C:\WINDOWS\system32
changecalc
C:\WINDOWS\system32\calcc...
start
Search Results
10:16 PM


calc
Calculator
Edit View Help
Standard
Nerd
Digit grouping
0.
CE
C
MC 7 8 9 / sqrt
MR 4 5 6 * %
MS 1 2 3 - 1/x
M+ 0 +/- . + =
Start a new search
Back
Go
start
calc
Calculator
10:24 PM

Task 02

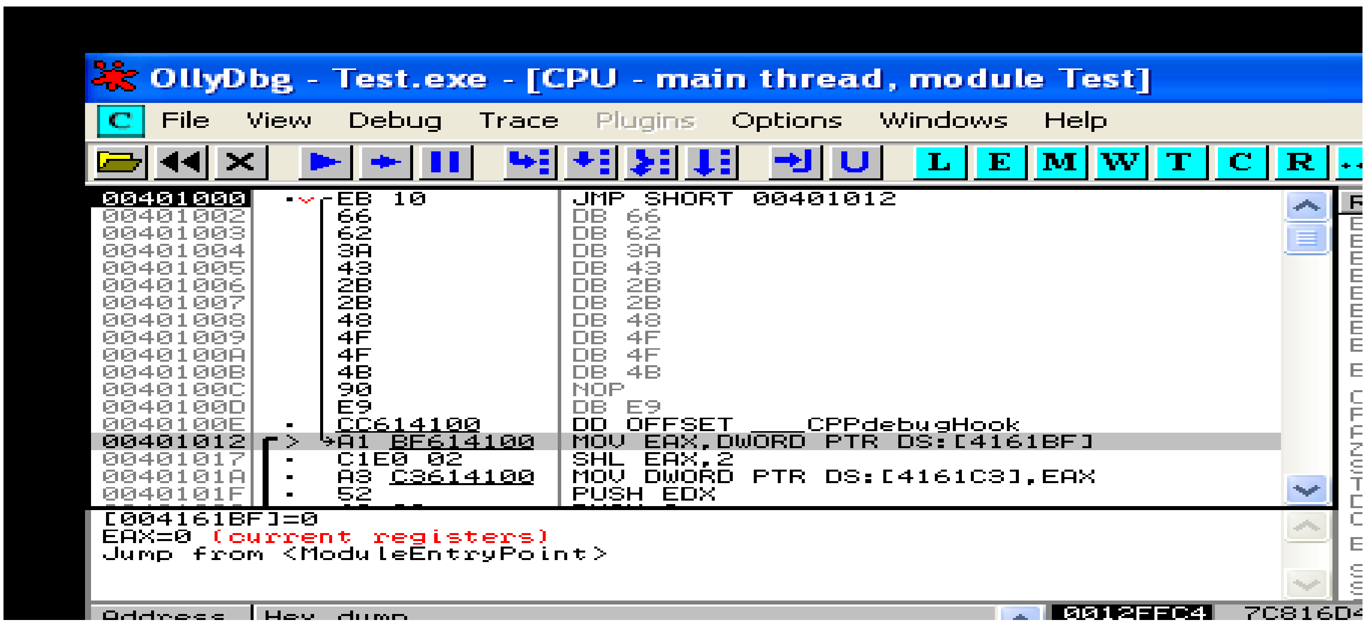
Olly Debugger

What is Olly Debugger?

” Olly is also a “dynamic” debugger, it allows the user to change quite a few things while running the program. This is very important when experimenting with a binary, when trying to figure out how it works. Olly has many, many great features, and this is the main debugger used for reverse engineering.

An Overview

Here is the Olly’s main display, along with some labels.



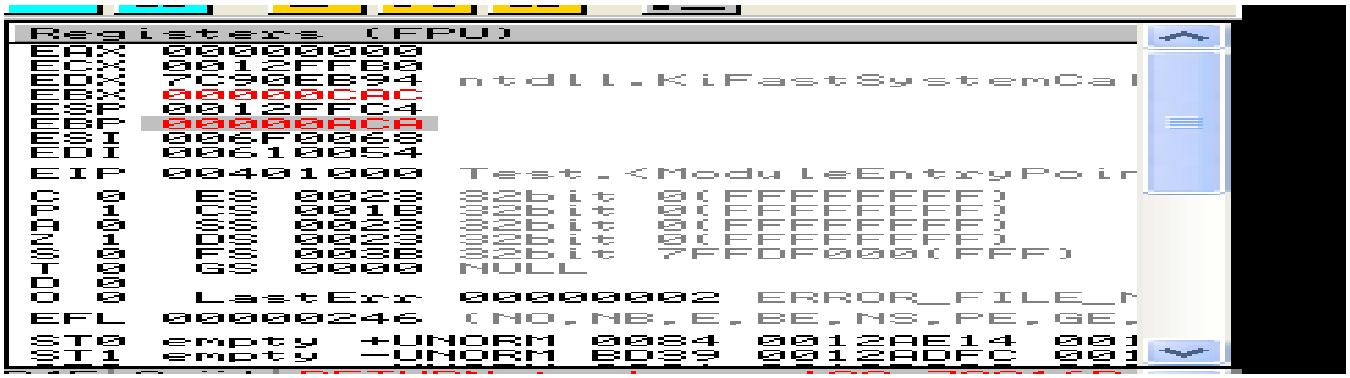
There are 4 separated main fields;

1. Disassembly
2. Registers
3. Stack
4. Dump

1. Disassembly

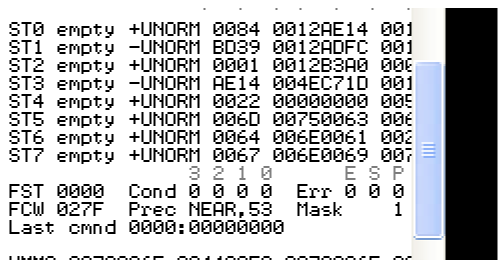
Window contains the main disassembly of the code for the binary. This is where Olly displays information in the binary, including the opcodes and translated assembly language.

2. Registers



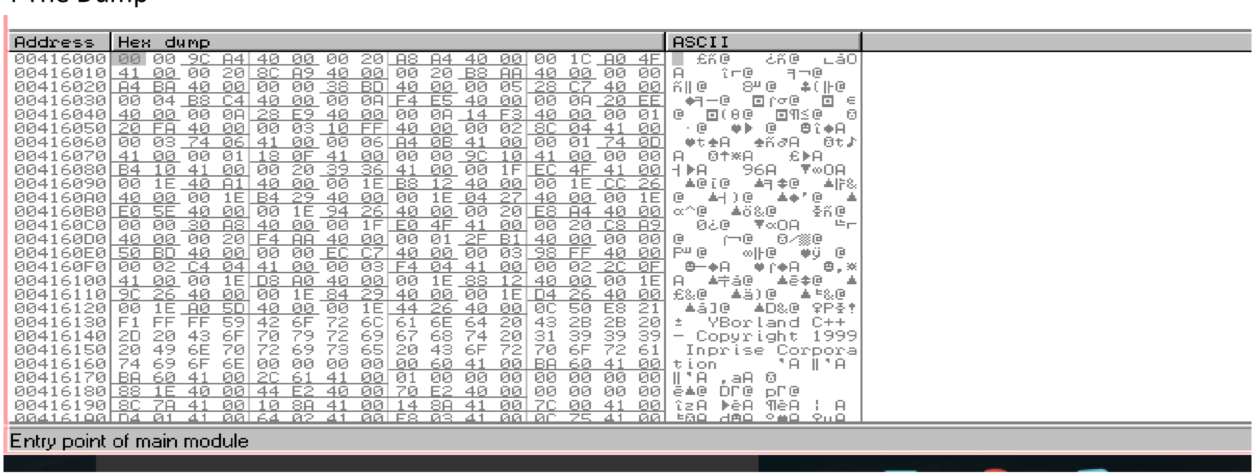
Registers are temporary holders for values, much like a variable in any high-level programming language.

3. The Stack



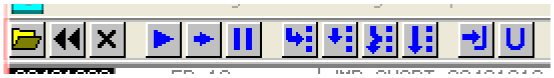
The stack is a section of memory reserved for the binary as a 'temporary' list of data. This data includes pointers to addresses in memory, strings, markers, and most importantly, return addresses for the code to return to when calling a function.In the stack, there's a "First In, Last Out" data structure.
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



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.

The Toolbar



All of the buttons are accessible from the "Debug" drop down menu.

Re-load

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

Run one line of code and then pause again, calling into a function call if there was one.

Step Over

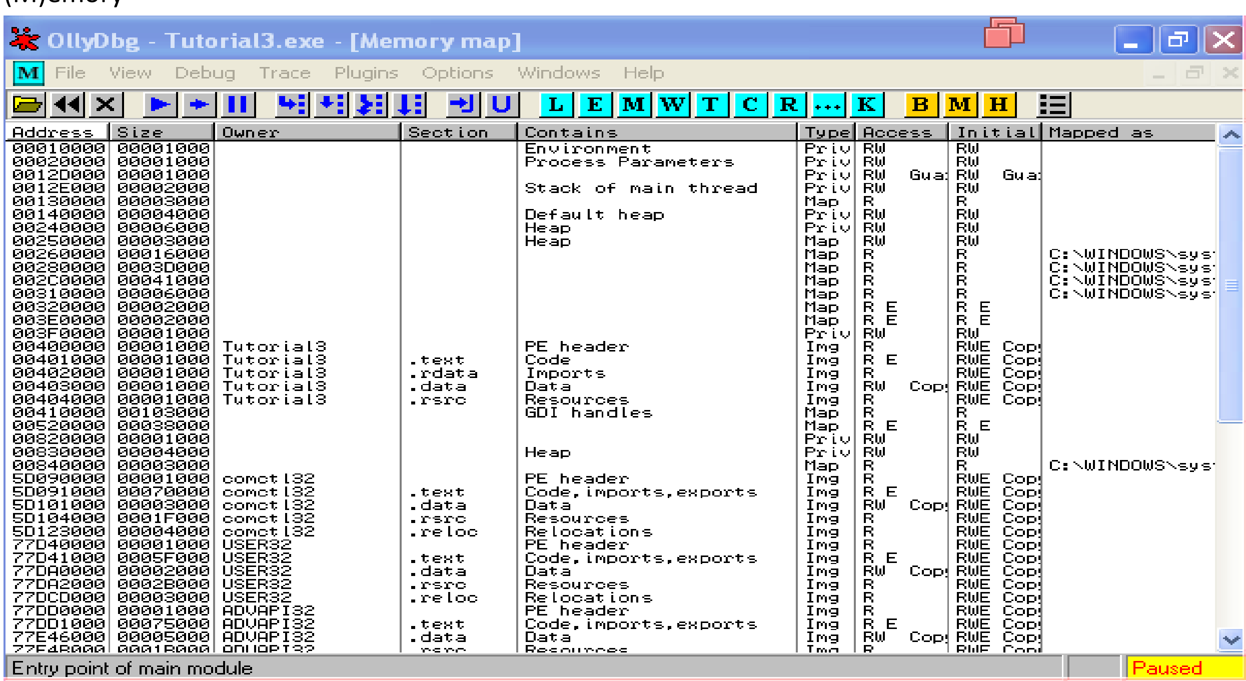
Does the same thing, but jumps over a call to another function.

Animate

Just like Step In and Over except it does it especially if it's a polymorphic binary and can watch the code change.

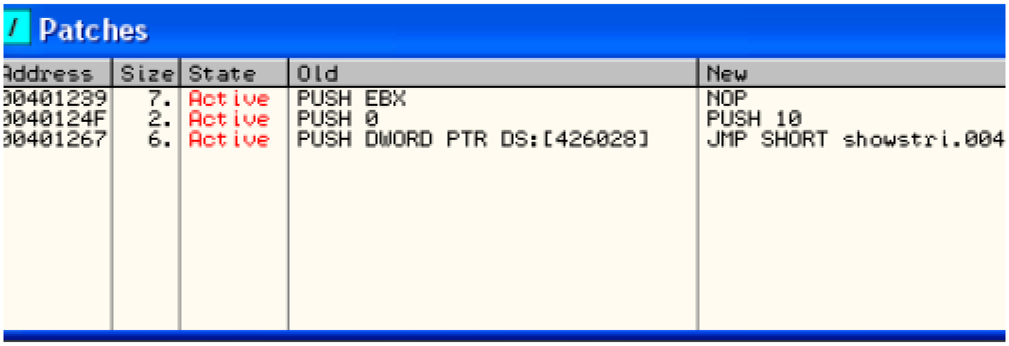
Some of the more common windows right now:

1. (M)emory



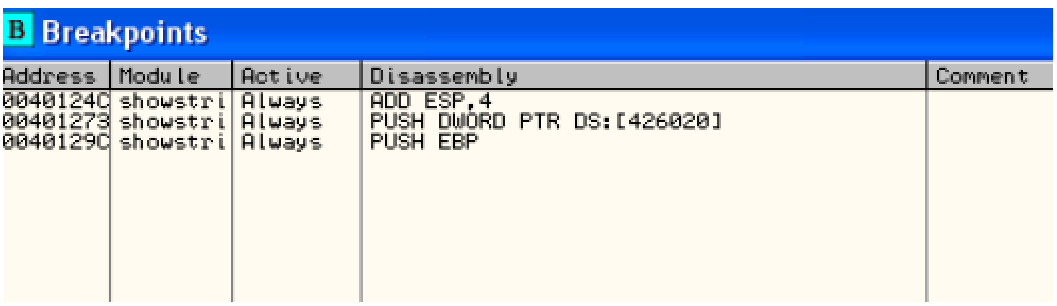
The memory window displays all of the memory blocks that the program has allocated. It includes the main sections of the running app.This window also shows the type of block, the access rights, the size and the memory address where the section is loaded.

2. (P)atches



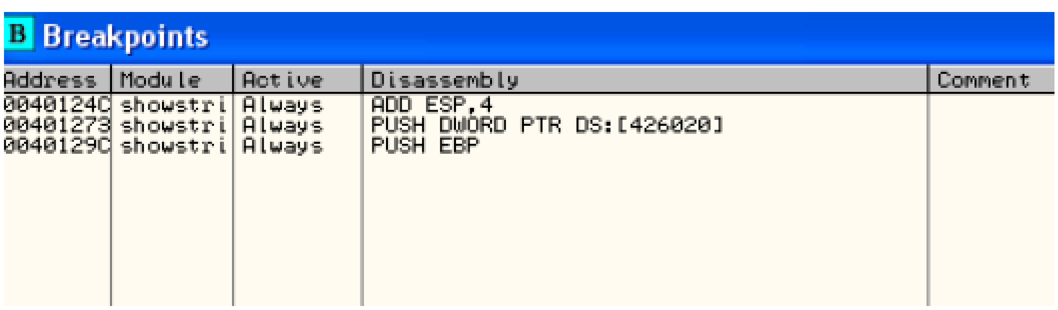
This window displays any “patches” you have made, any changes to the original code. Notice that the state is set as “Active”. If you re-load the app, patches will become disabled. In order to re-enable them (or disable them) simply click on the desired patch and hit the spacebar. This toggles the patch on/off.

3. (B)reakpoints



Shows where all of the current breakpoints are set.

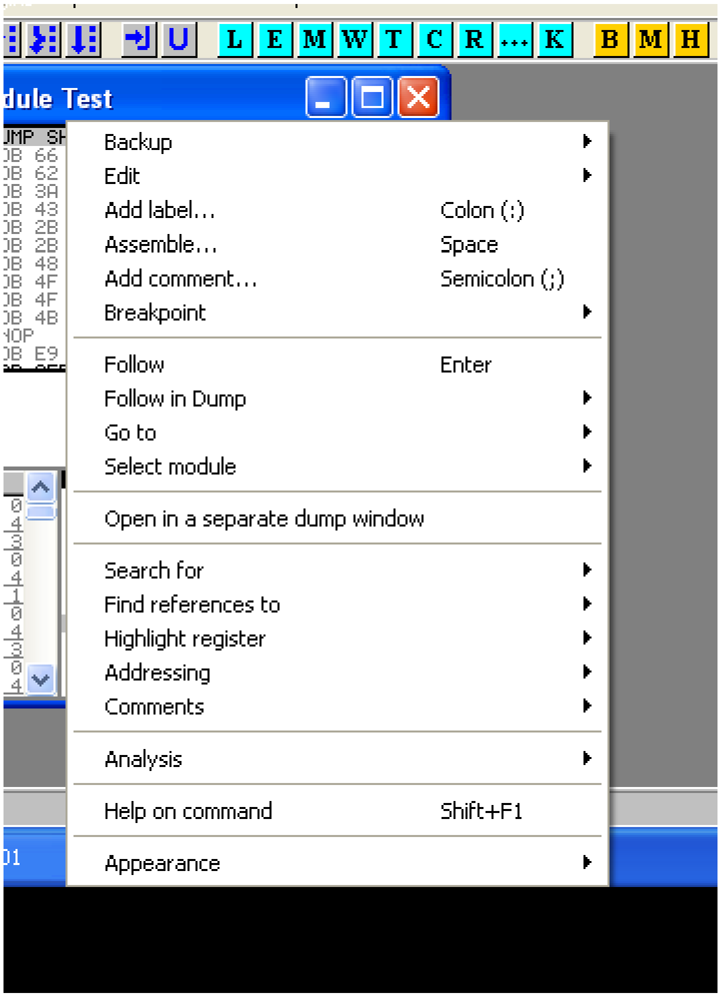
4. (K)all Stack



This window is different from the “Stack”. It shows a lot more info about calls being made in the code, the values sent to those functions, and more.

The Context Menu

Right-clicking anywhere in the disassembly section brings it up:



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

A rather large sub-menu, and it’s where you search the binary for data such as strings, function calls etc.

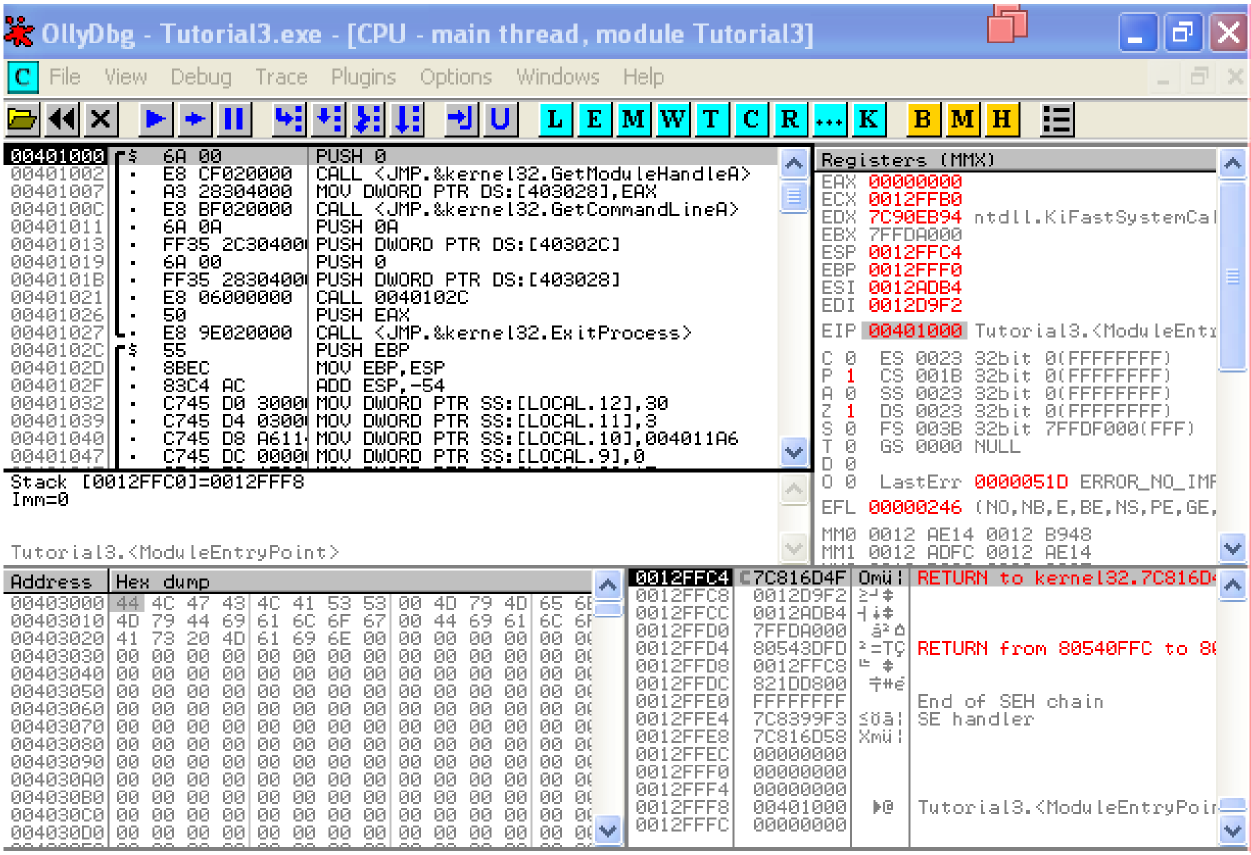
Analysis

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.

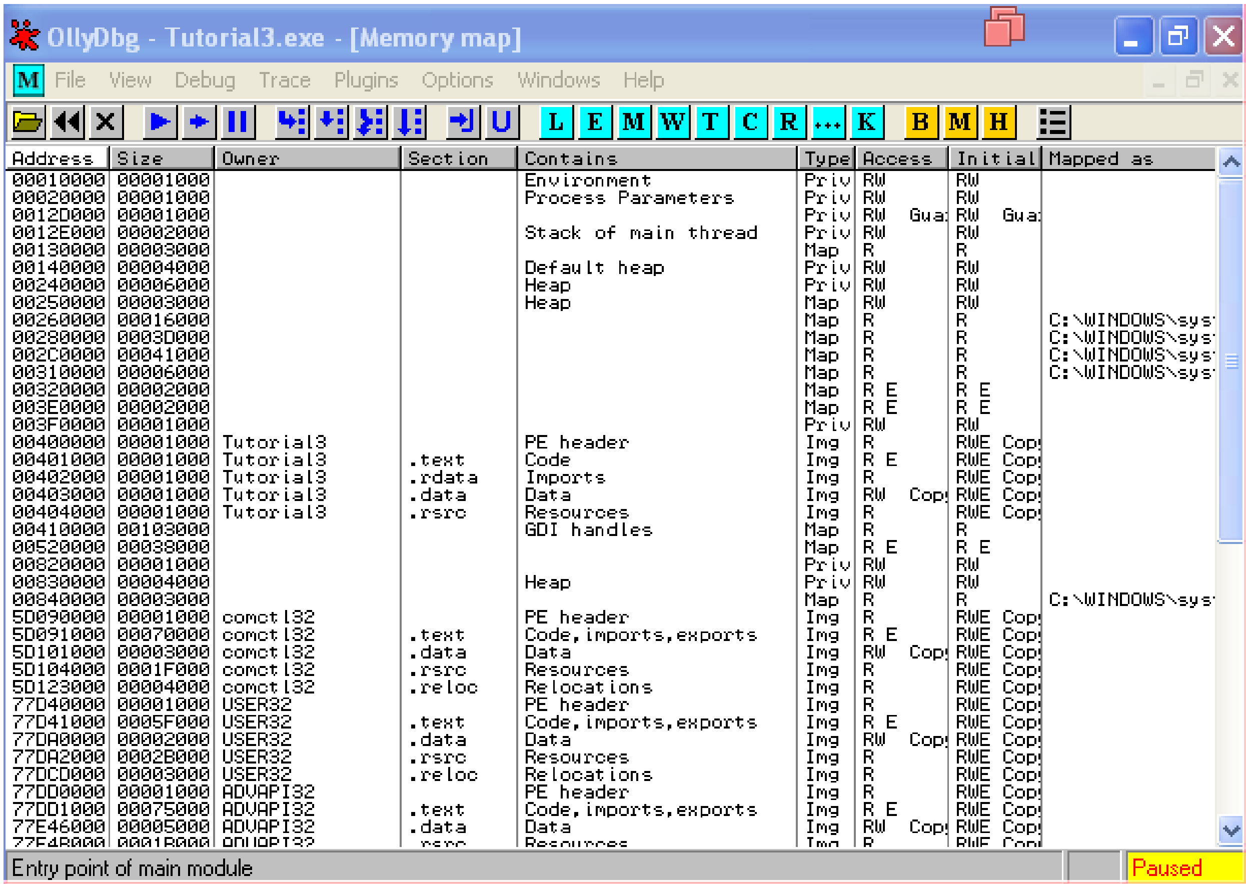
Task 03

Loading the app

As the first step load “FirstProgram.exe”, file to the Olly and it completed analysis and stop at the programs Entry Point (EP):



Here in the first column the EP is at address 401000.This is a pretty standard starting point for an executable.



Address	Size	Owner	Section	Contains	Type	Access	Initial	Mapped as
00010000	00001000			Environment	Priv	RW	RW	
00020000	00001000			Process Parameters	Priv	RW	RW	
0012D000	00001000				Priv	RW Gua	RW Gua	
0012E000	00002000			Stack of main thread	Priv	RW	RW	
00130000	00003000				Map	R	R	
00140000	00004000			Default heap	Priv	RW	RW	
00240000	00006000			Heap	Priv	RW	RW	
00250000	00003000			Heap	Map	RW	RW	
00260000	00016000				Map	R	R	C:\WINDOWS\sys
00280000	0003D000				Map	R	R	C:\WINDOWS\sys
002C0000	00041000				Map	R	R	C:\WINDOWS\sys
00310000	00006000				Map	R	R	C:\WINDOWS\sys
00320000	00002000				Map	R E	R E	
003E0000	00002000				Map	R E	R E	
003F0000	00001000				Priv	RW	RW	
00400000	00001000	Tutorial3		PE header	Img	R	RWE Cop	
00401000	00001000	Tutorial3	.text	Code	Img	R E	RWE Cop	
00402000	00001000	Tutorial3	.rdata	Imports	Img	R	RWE Cop	
00403000	00001000	Tutorial3	.data	Data	Img	RW Cop	RWE Cop	
00404000	00001000	Tutorial3	.rsrc	Resources	Img	R	RWE Cop	
00410000	00103000			GDI handles	Map	R	R	
00520000	00038000				Map	R E	R E	
00820000	00001000				Priv	RW	RW	
00830000	00004000			Heap	Priv	RW	RW	
00840000	00003000				Map	R	R	C:\WINDOWS\sys
5D090000	00001000	comctl32		PE header	Img	R	RWE Cop	
5D091000	00070000	comctl32	.text	Code,imports,exports	Img	R E	RWE Cop	
5D101000	00003000	comctl32	.data	Data	Img	RW Cop	RWE Cop	
5D104000	0001F000	comctl32	.rsrc	Resources	Img	R	RWE Cop	
5D123000	00004000	comctl32	.reloc	Relocations	Img	R	RWE Cop	
77D40000	00001000	USER32		PE header	Img	R	RWE Cop	
77D41000	0005F000	USER32	.text	Code,imports,exports	Img	R E	RWE Cop	
77DA0000	00002000	USER32	.data	Data	Img	RW Cop	RWE Cop	
77DA2000	0002B000	USER32	.rsrc	Resources	Img	R	RWE Cop	
77DCD000	00003000	USER32	.reloc	Relocations	Img	R	RWE Cop	
77DD0000	00001000	ADVAPI32		PE header	Img	R	RWE Cop	
77DD1000	00075000	ADVAPI32	.text	Code,imports,exports	Img	R E	RWE Cop	
77E46000	00005000	ADVAPI32	.data	Data	Img	RW Cop	RWE Cop	

In the address column, at location 401000, the row contains,

“FirstPro” (First Program) →1000

“.text, →“SFX, code”

This is the 1000h bytes long “code” for the program and it starts at address 401000 in memory.

“.rdata”→ contains data and imports at address 402000,

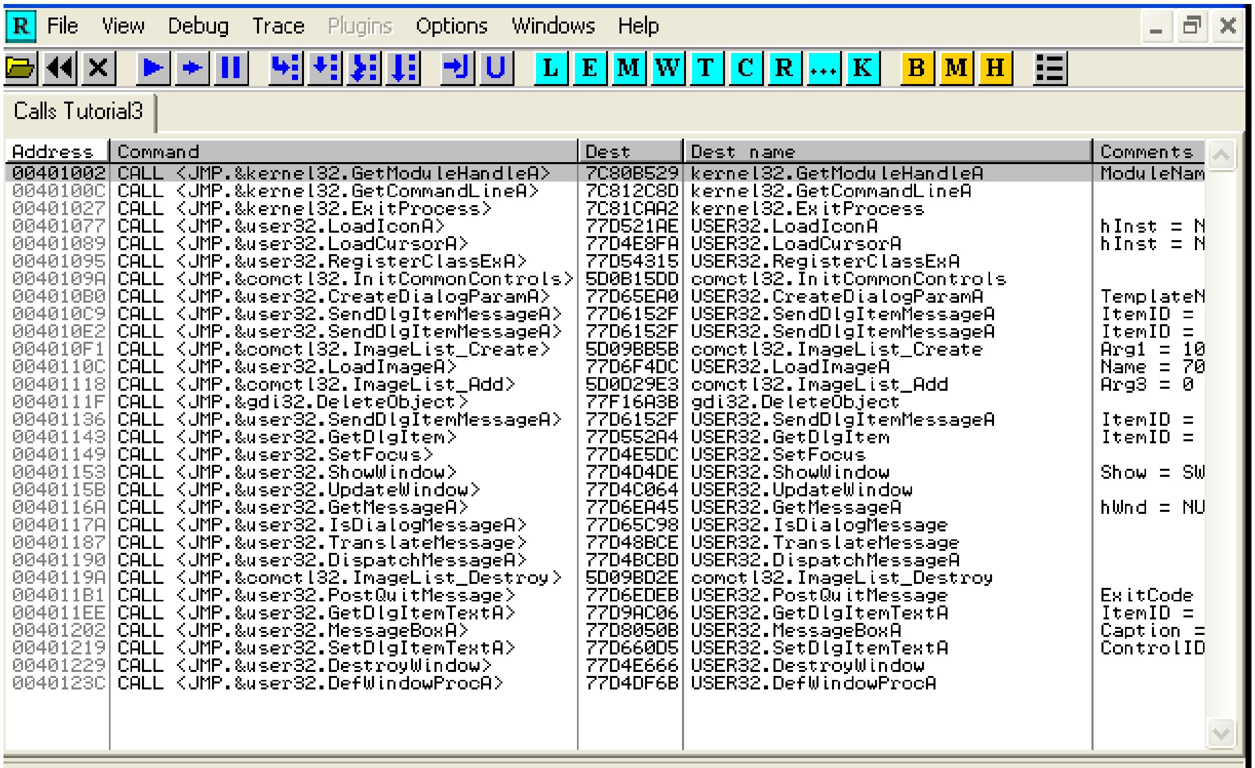
“.data” → contains nothing at address 403000(actually contains global variables and random data but Olly doesn’t what kind of data it stored)

“.rsrc” → contains resources (eg: dialog boxes, images, text etc.)

To see exactly which functions program calls,

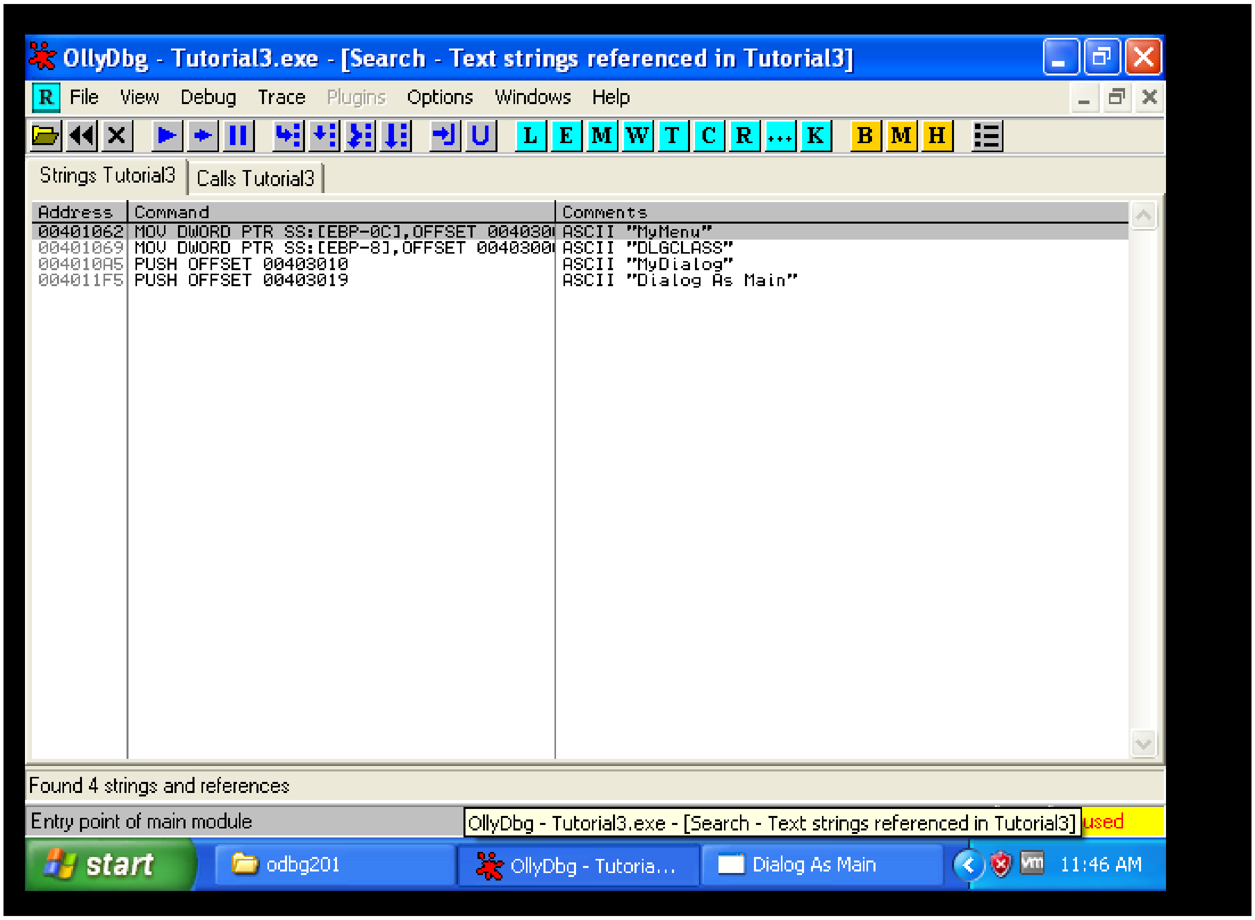
Right-click in Olly’s disassembly window and select “Search For” -> “All Intermodular Calls.

This shows something like the following:



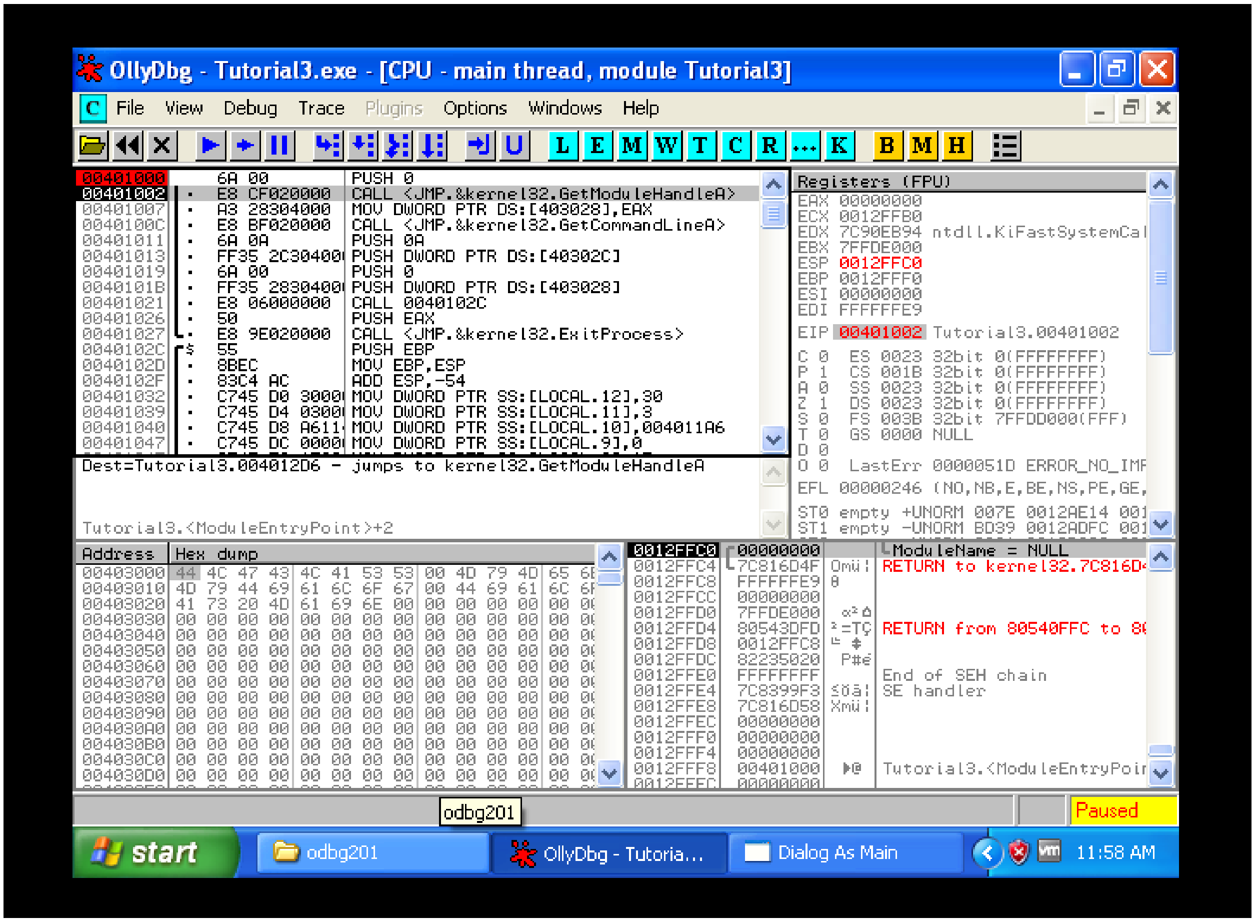
To search for all strings in the app,

Right-click the disassembly window and choose “SearchFor” -> “All Referenced Text Strings



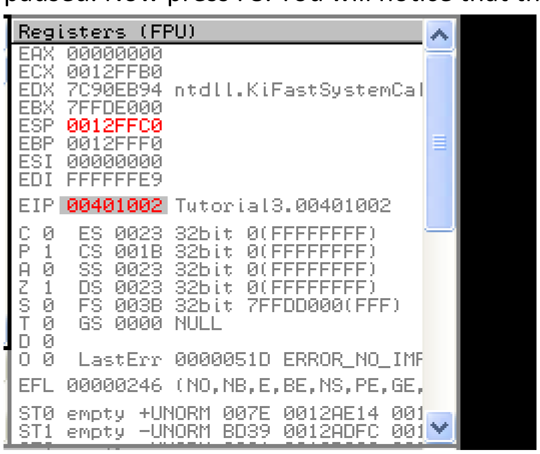
This window shows all text strings it could find in our app.

Try hitting F9 (or choose “Run” from the “Debug” menu option). After a second, program will pop up a dialog box (it may open behind Olly, so minimize Olly’s window to make sure.)

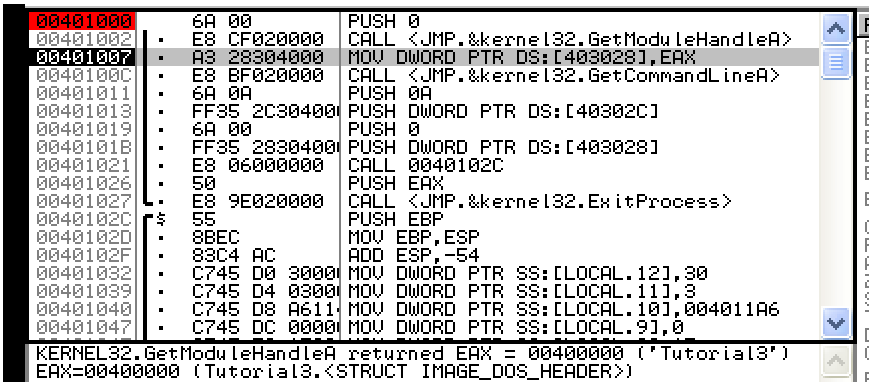


To do single-stepping,

Reload the application (reload icon, ctrl-F2, or debug->restart) at the start of the application paused. Now press F8. You will notice that the current line selector has gone down one line.



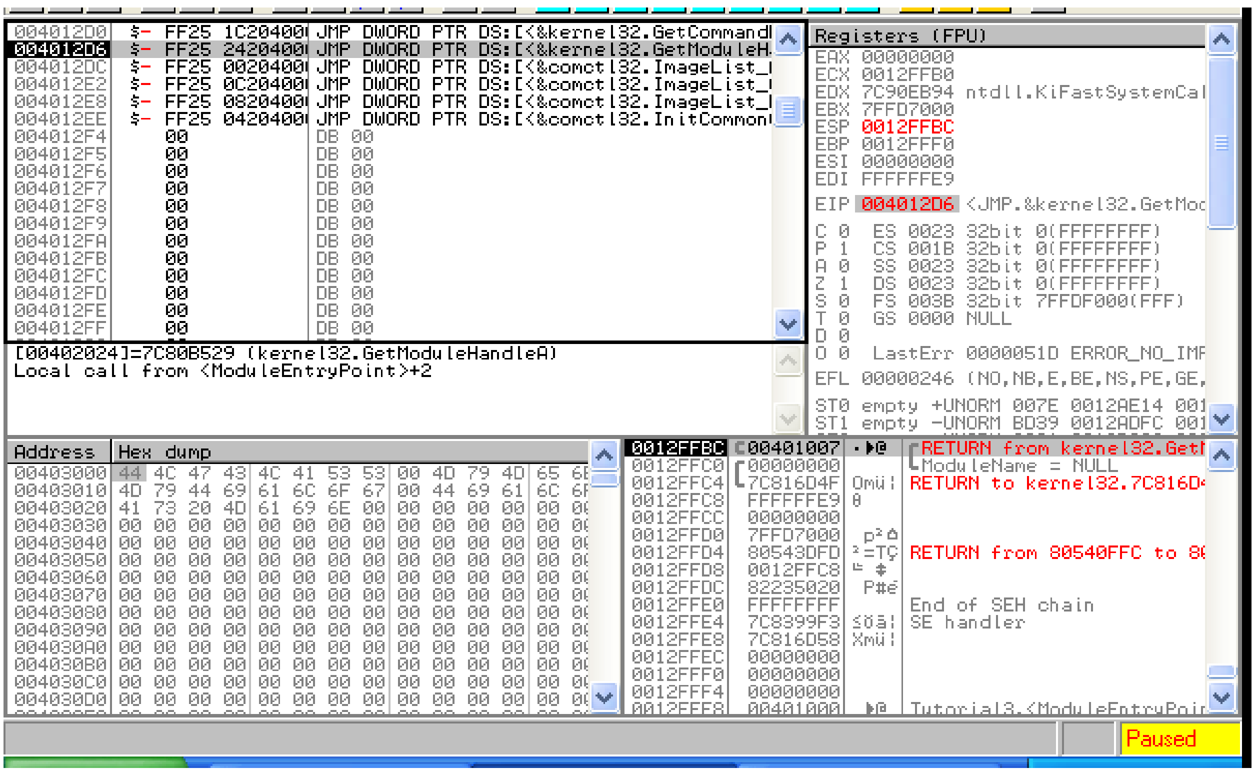
This is because the instruction that we performed, PUSH 0" 'pushed' a zero onto the stack. This shows up on the stack as "pModule = NULL". NULL is another name for zero. You also may have noticed that in the registers window, the ESP and EIP registers have turned red.



A register turns red means that the last instruction run changed that register.

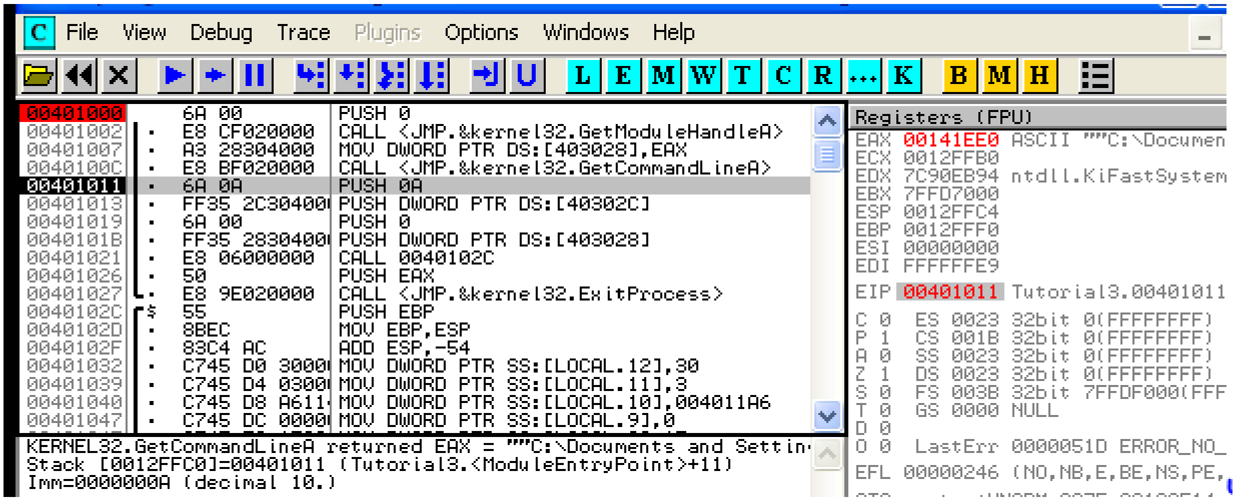
In this case, the **ESP register** (which points to the address of the top of the stack) was incremented by one since we pushed a new value onto the stack.

The **EIP register**, which points to the current instruction that is being run, has also increased by two. This is because we are no longer on address 401000, but 401002.Running that last instruction was two bytes long and now paused on the next instruction. This instruction is at 401002, the current value of EIP.



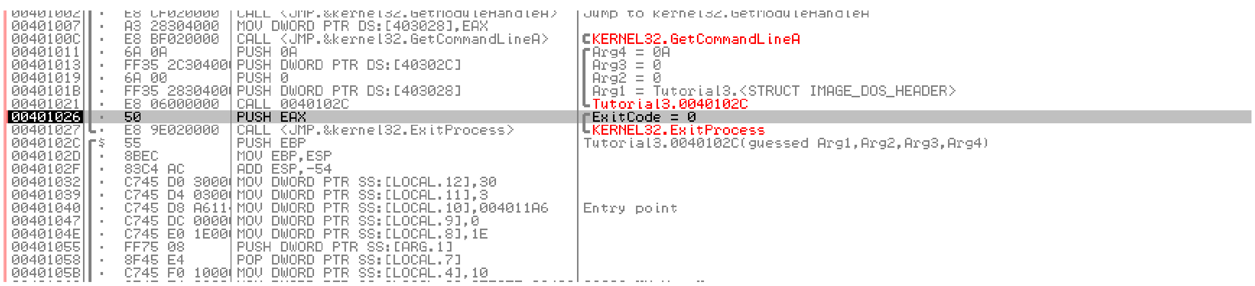
Now press F8 once more for "Step-Over". The current line indicator will move down one, the EIP register will stay red and increase by 5 (as the instruction that ran was 5 bytes long) and the stack was brought back to what it originally showed.

To see the other option, re-start the program (ctrl-F2), press F8 to step over the first instruction, but this time hit F7 on the call instruction. Then the entire window looks different.



This is because F7 is “Step-In”, means the call jumped to a new area of memory (EIP = 4012d6).

Now the program paused at the beginning of the program, hit F8 (Step-Over) 4 times and landed on this statement.

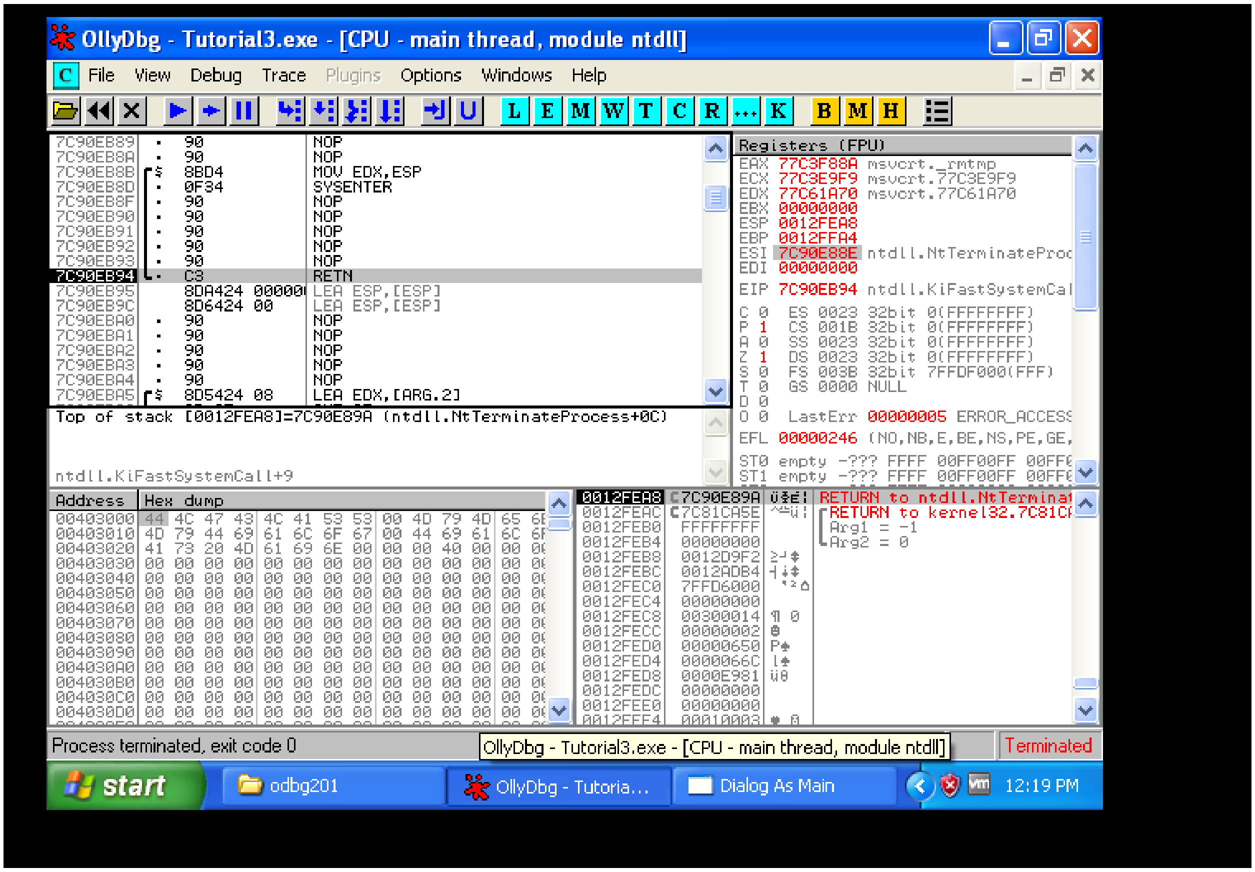


There are 4 PUSH statements in a row. This time, watch the stack window as you hit F8 4 times and watch the stack grow, it actually grows down.

In this case it is because these 4 numbers are being passed as parameters to a function.

Now press F8 one more time and noticed that it will say “Running” in the active bar in Olly and dialog box will show up. This is because we stepped-over the call that actually has most of the program in it.

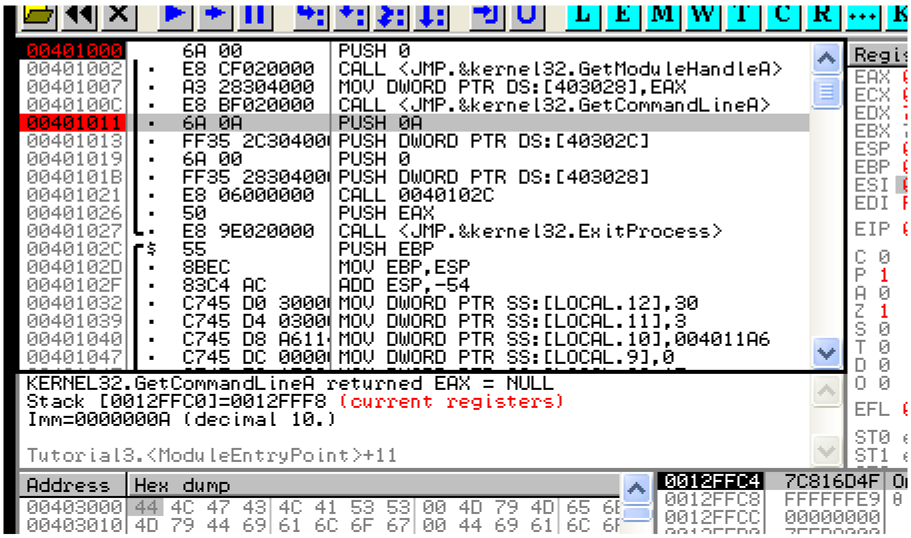
To fix that, Click over the program and hit the ‘close’ button to end the app. Olly will immediately pause on the next line after the call.



You will also notice that our program has disappeared. This is the Windows API that stops an application, Olly has paused program after it has closed the window. But before it has actually been terminated! If now press F9, the program will terminate, the active bar in Olly will say “Terminated” and no longer debugging anything.

Breakpoints

Re-load the app (ctrl-F2) and double-click on the line at address 401011 in the second column. Address 401011 will now turn red. This means set a breakpoint on address 401011. Breakpoints force Olly to pause execution when it reaches it. There are different types of breakpoints in order to stop execution on different events.



Software Breakpoints

A software breakpoint replaces the byte at your breakpoint address with a 0xCC opcode, which is an int 3.This is a special interrupt that tells the operating system that a debugger wishes to pause here and to give control over to the debugger before executing the instruction.

To set a software breakpoint,

Double-click on the opcode column, or highlight the row which want the breakpoint on, right-click on it, and choose Breakpoints->Toggle (or hit F2).

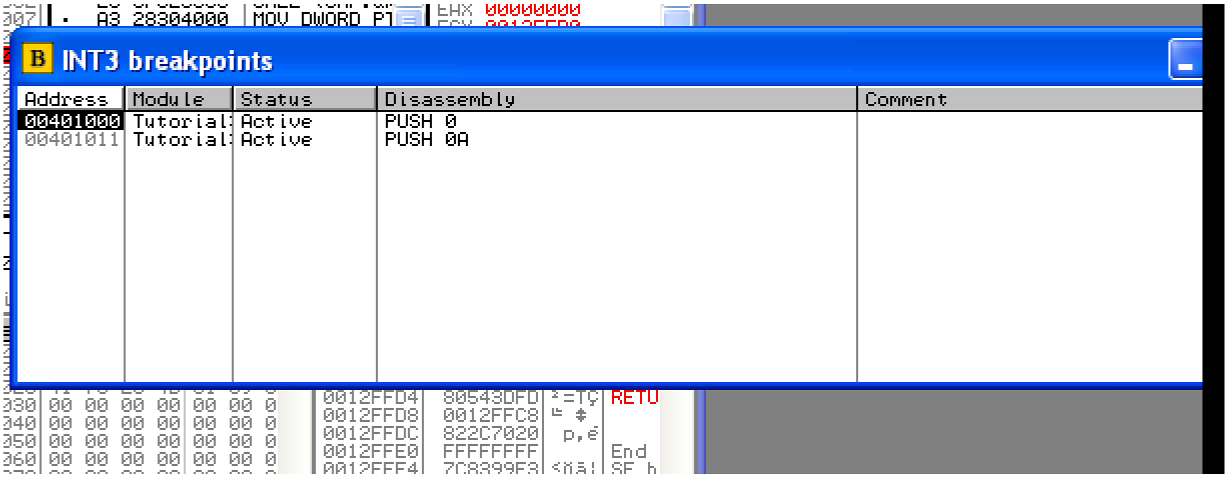
To remove the breakpoint,

Double-click on the same line or right-click and select Breakpoint->Remove Software Breakpoint (or hit F2 again).

To see an entry in the breakpoint window that shows currently set BP,

Click on the "Br" toolbar icon or select View->Breakpoints

As a practical I've a BP (breakpoint) set at address 401011 and program is paused at the first instruction, hit F9 (run) now the program will run but will pause at the line with our BP on it.



If once highlight a breakpoint and click the space bar, the breakpoint will toggle between enabled and disabled. You can also highlight a breakpoint row and hit “DEL” key which will remove the breakpoint.

Lastly, restart the program, go into the breakpoints window, highlight the breakpoint we set at address 401011, and hit the space bar. The “Active” column will change to “Disabled”. Now run the program (F9). You will notice that Olly did not stop at our BP because it was disabled.

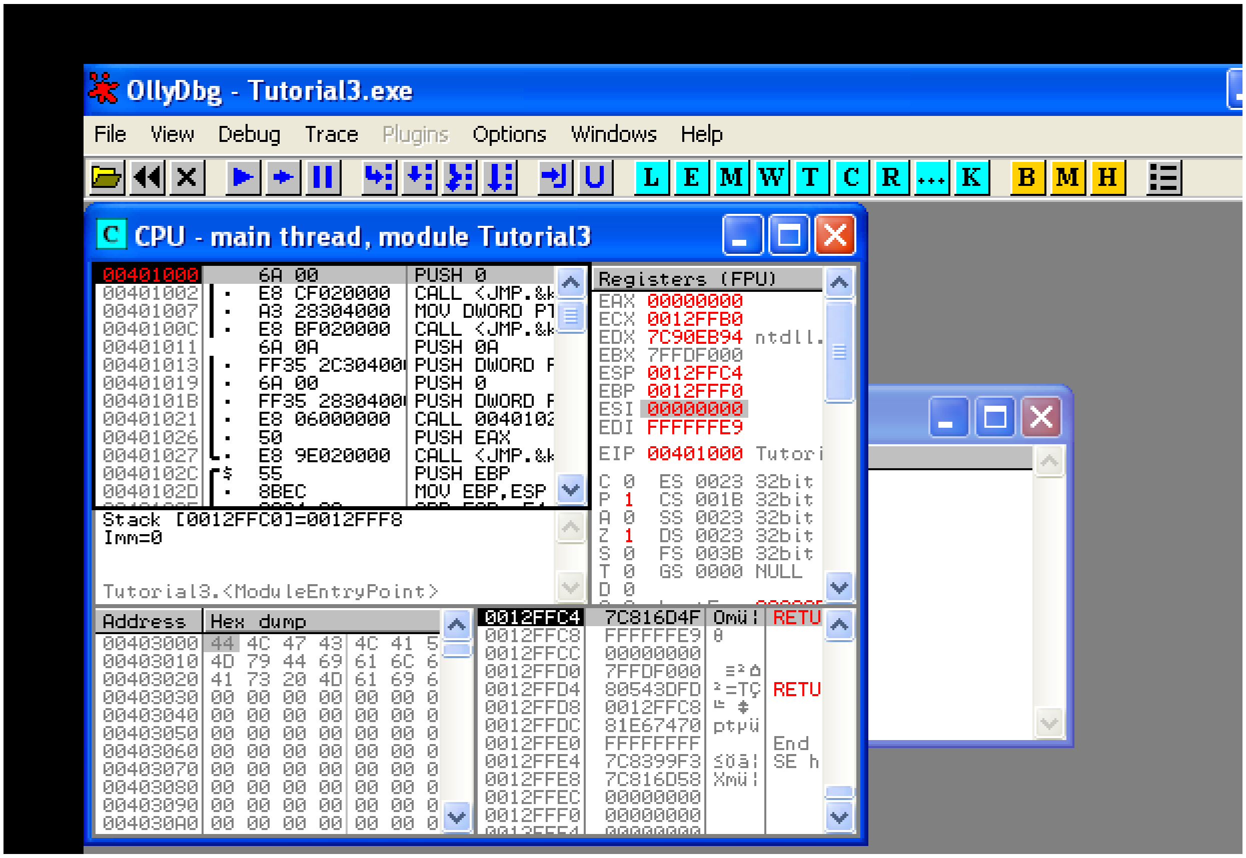
Hardware Breakpoints

A hardware breakpoint uses the CPU’s debug registers. There are 8 of these registers built into the CPU, R0-R7. Even though there are 8 built into the chip, we can only use four of them to breaking on reading, writing or executing a memory section.

The difference between hardware and software breakpoints is hardware BP’s don’t change the process’s memory, so they can be more reliable, especially in programs that are packed or protected.

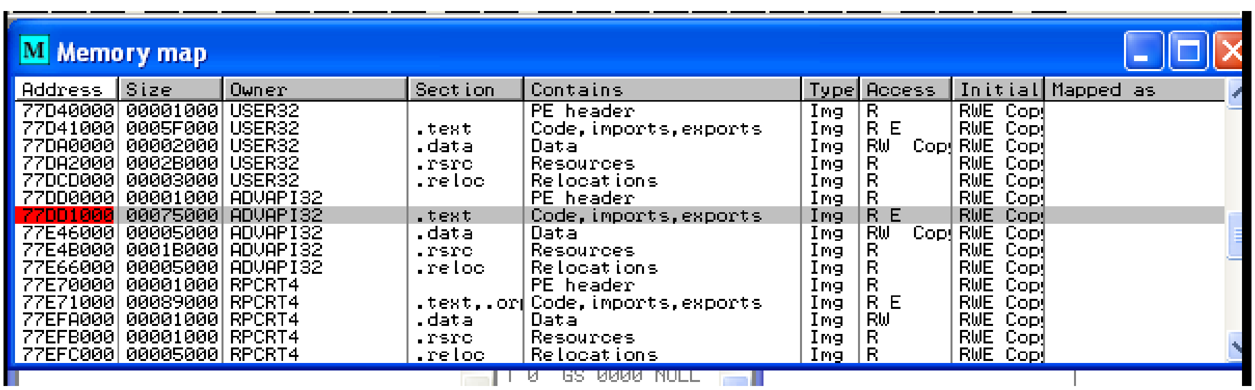
To set a hardware breakpoint,

Right-clicking on the desired line, selecting Breakpoint, and then choosing Hardware, on Execution.



To see what memory BP’s have set,

Select “Debug” and “Hardware Breakpoints”



Memory Breakpoints

Sometimes you may find a string or a constant in the program’s memory, but you don’t know where in the program it is accessed. Using a memory breakpoint tells Olly that you want to pause whenever ANY instruction in the program reads or writes to that memory address (or groups of addresses.)

To select a memory breakpoint:

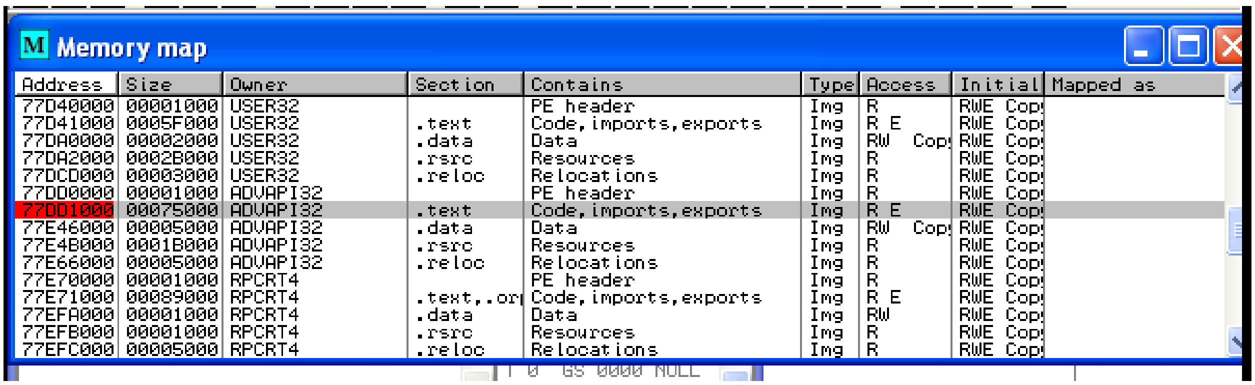
Right-click on the desired line and select Breakpoint->Memory, On Access or Memory, On Write

To set a BP on an address in the memory dump,

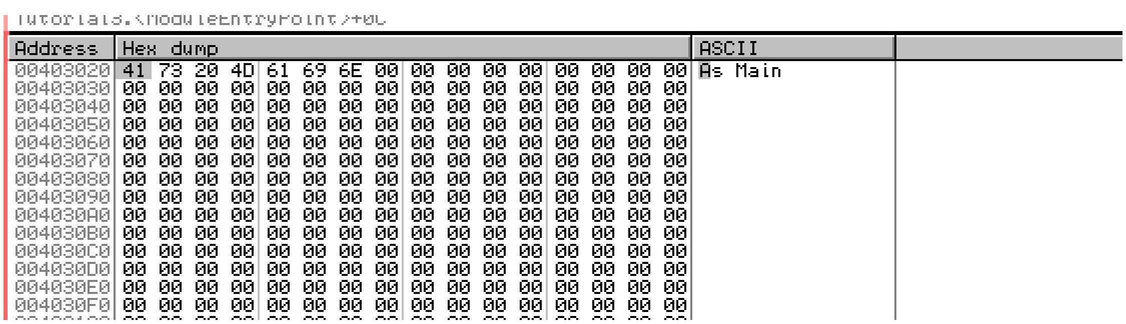
Highlight one or more bytes in the dump window, right click on them and select Breakpoint->Memory, On Access or Memory, On Write

To set a BP for an entire section of memory,

Open Memory window (“Me” icon or View->Memory), right click the section of memory you desire, and right-click and choose “Set Break On Access for either Access or Write.



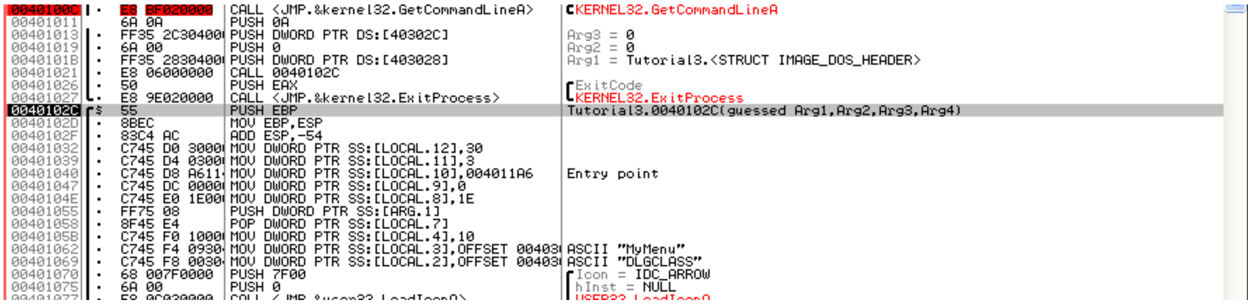
There are three ways to set a memory break point.



Using the Dump Pane

Right click on reference and select “Follow in Dump” and the dump pane will show you the address section. You can also right-click anywhere in the dump pane and select “Go To” to enter an address to view.

Now, press F8 eight times and it shows the current instruction at address 401021 that says CALL FirstPro.40102c. This is a CALL instruction, so eventually come back to 401021.



Now, step the code (F8) until get to address 401062. Also just set a breakpoint on this line and hit F9 to run to it. Now program paused at address 401062.

This instruction moves whatever is at address 00403009 onto the stack. Right-click on the instruction and select "Follow in Dump" and then noticed couple options here.

To see the address is being affected in the instruction select "Immediate Constant". If you had chosen "Selection"→shows the address of the highlighted line, in this case 401062

"Memory address"→ shows the memory for LOCAL.3

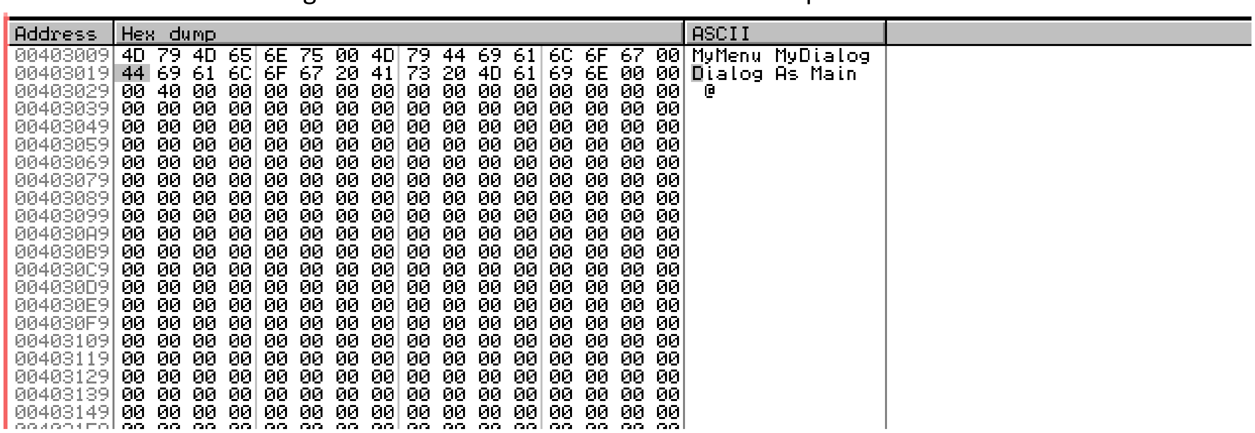
The dump looks like after selecting "Immediate Constant",

Address 403009→ the address of the instruction Olly was loading the ASCII string from.

Let's edit the binary to display our own message!

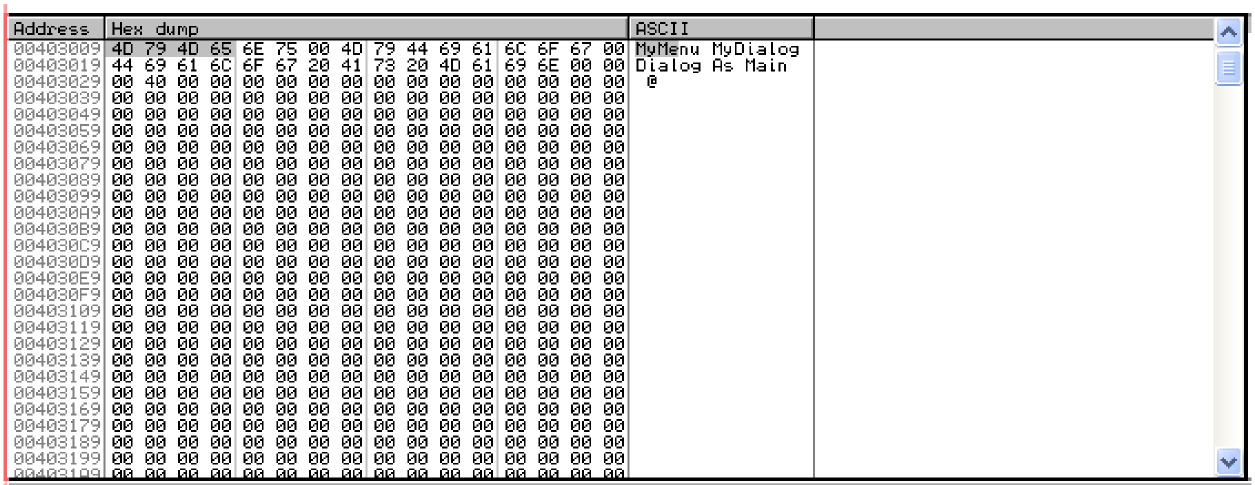
Change the "Dialog As Main" string to something of our own and then see what happens.

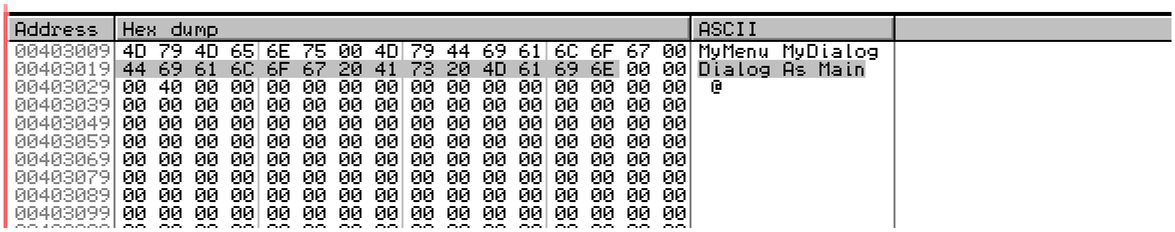
01. Click on the "D" of "Dialog As Main" in the ASCII section of the dump window.



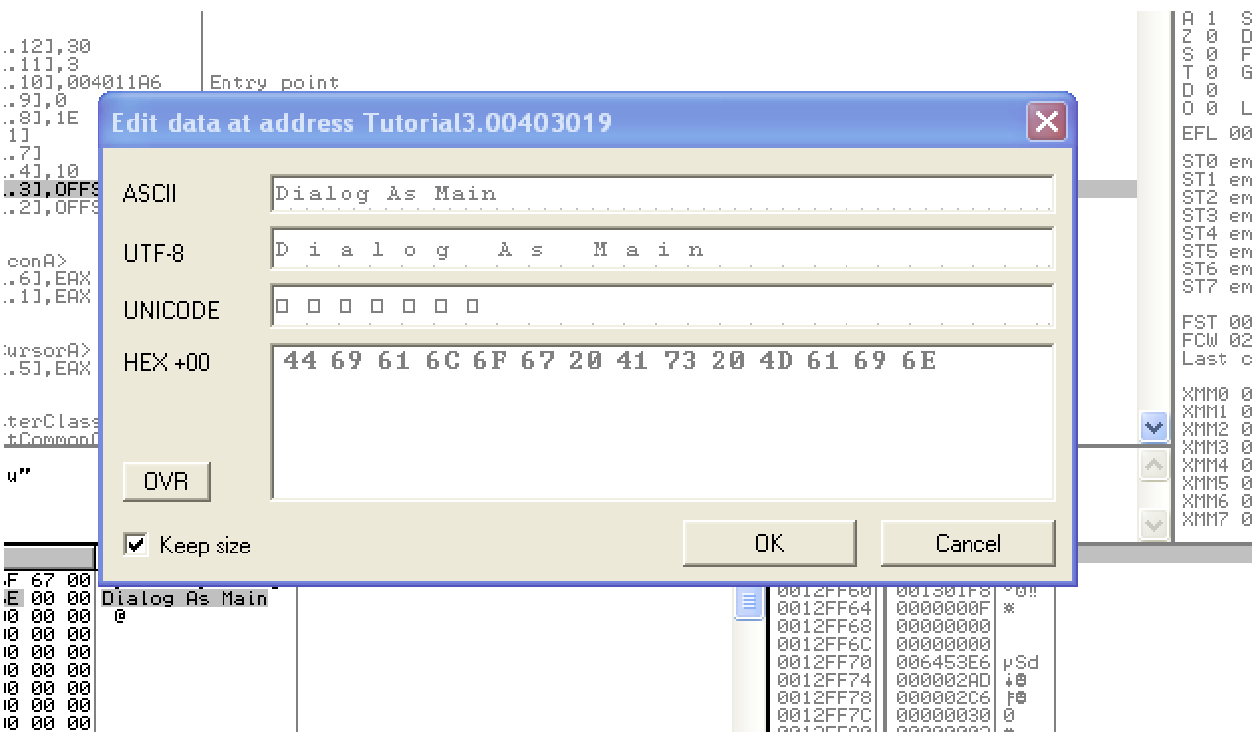
Here noticed that the first hex digit is also highlighted on the left.

02. Then clicked and dragged to select the entire string "Dialog As Main",





03. Right-click anywhere on the selection and choose “Binary” -> “Edit” to change the memory contents of program.



Now the screen looked.

Here I typed “waruna ” over top of the “Dialog As Main” string.

04. Now click OK and run the app. Switch over to our program, type something in No tags and select “Options” -> “Get Text”. Now look at the dialog box.