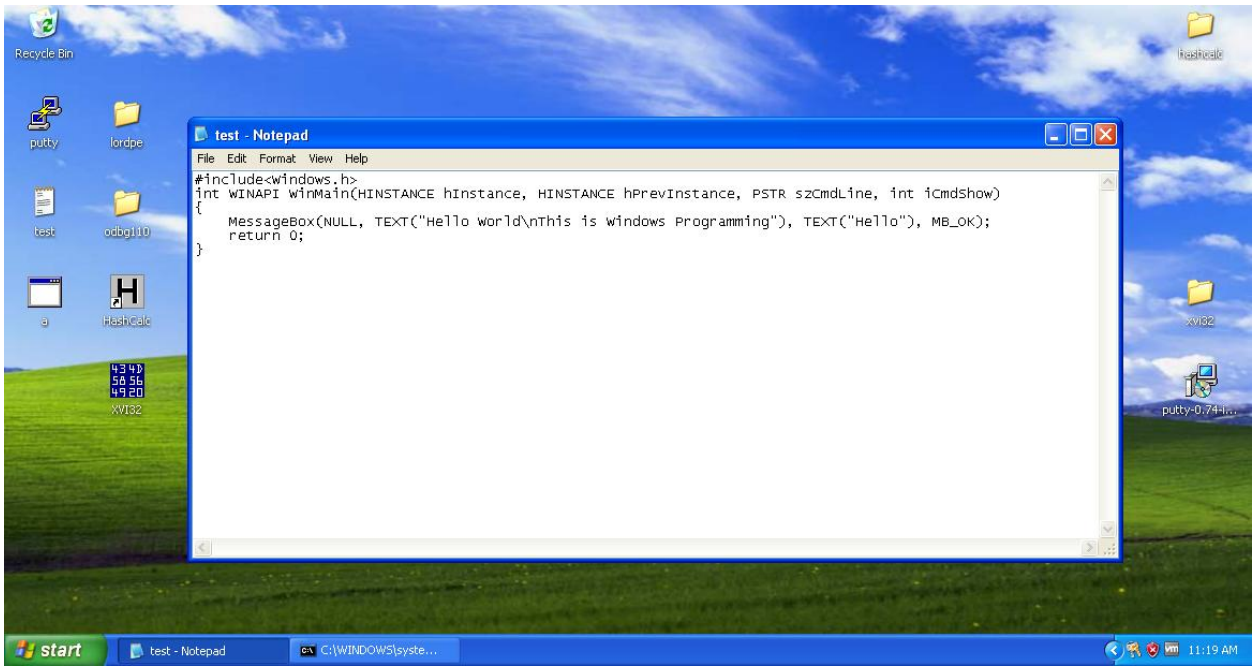
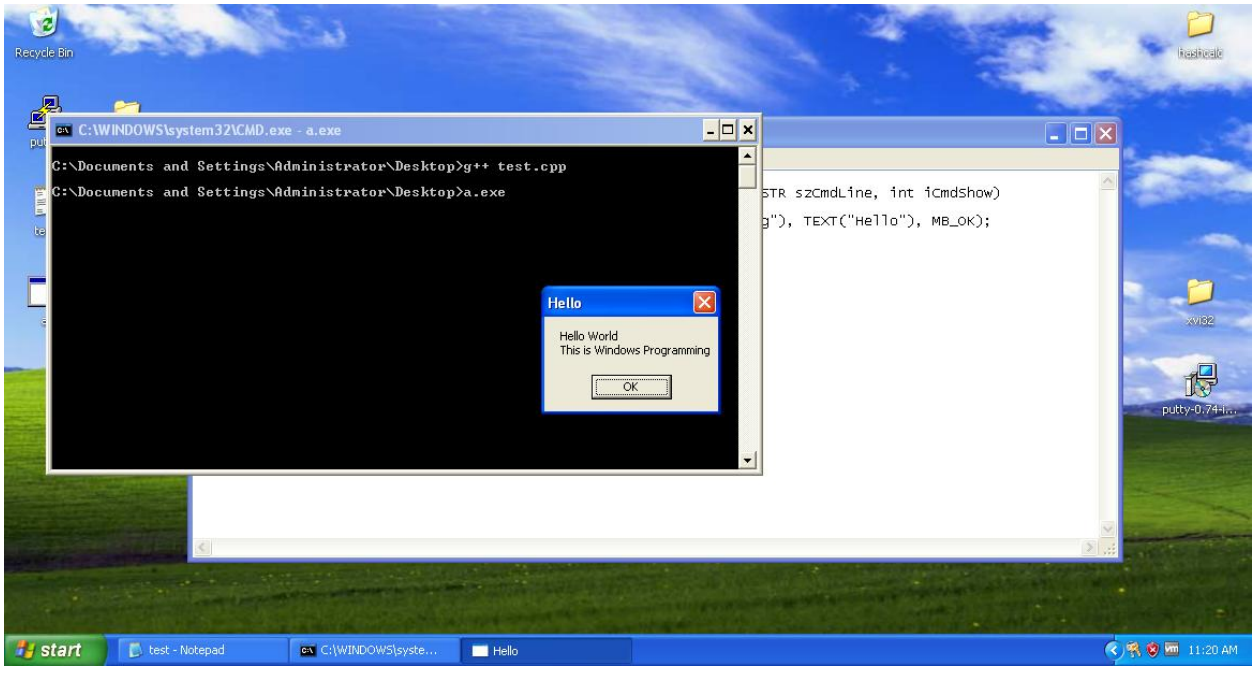
Here's the windows program (test.cpp) that display message box.



Compile the program (g++ test.cpp). And then run the executable file (a.exe).

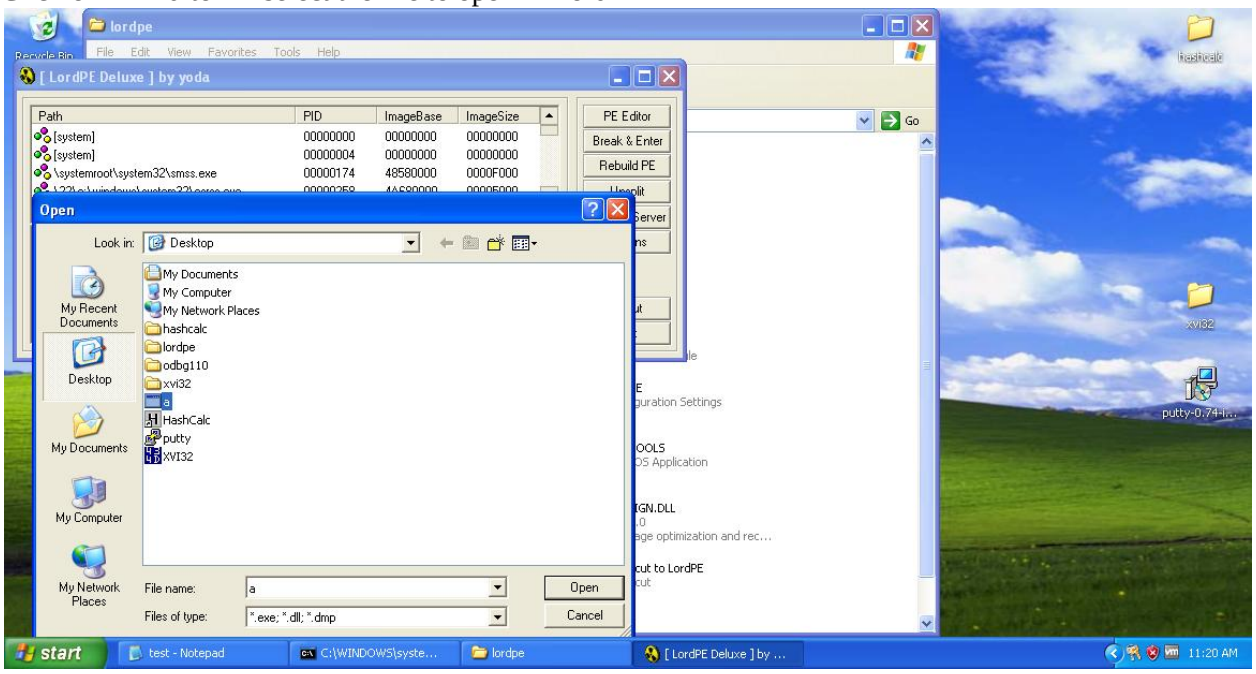


After running the file, it shows the above message box and after clicking OK it returns control back to console (or cmd).

Adding a New Section to the PE Header

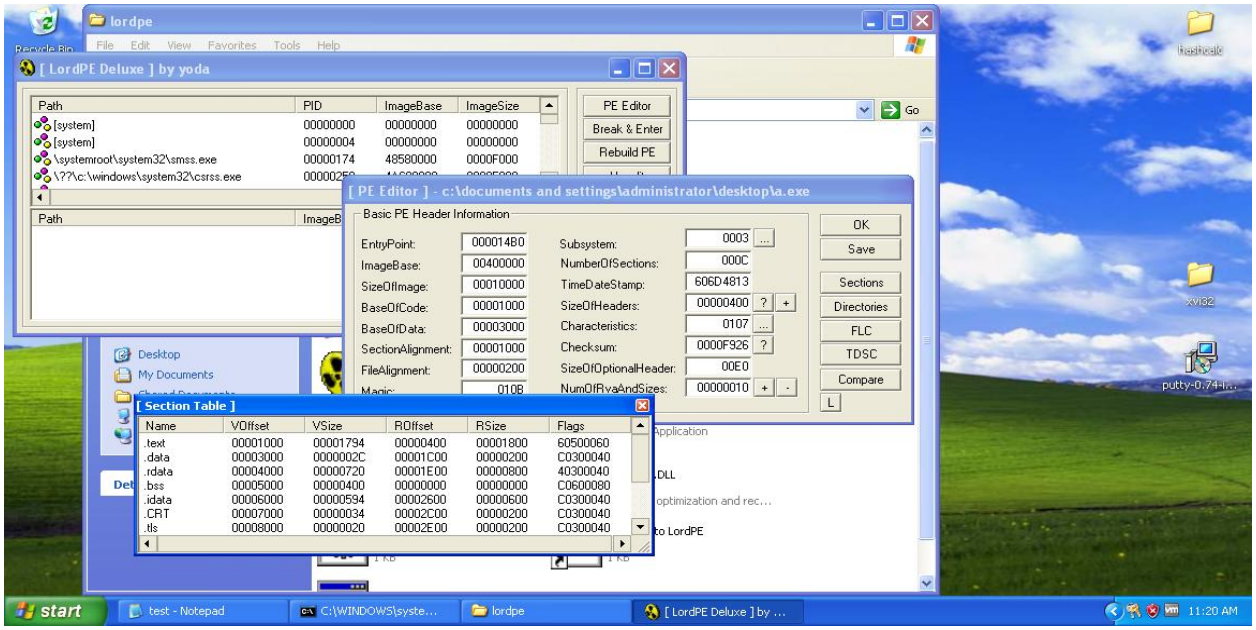
Start the LordPE application.
LordPE is a tool for system programmers/reverse engineers which is able to edit/view many parts of PE (Portable Executable) files, dump them from memory, optimize them, validate, analyze, edit, etc.

Click on PE Editor -> select the file to open in LordPE



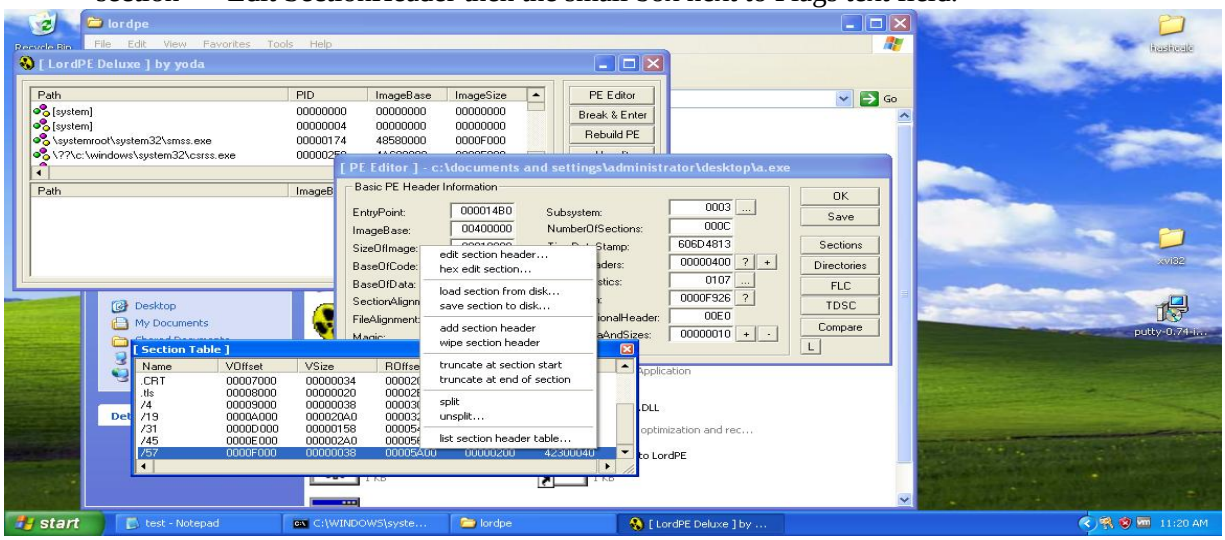
A "PE Editor" box opens, showing general information about a.exe, as shown below.
To get the sections involved in this executable by clicking on ‘Sections’ option. A section Tale appears with the sections and related information (Virtual offset, Virtual Size, Real Offset, Real Size and Flags).

• EntryPoint: Virtual offset from base address that points to the first command to be executed (ModuleEntryPoint).
• ImageBase: Preferred base address to map the executable to, although default value is 0x00400000, this value can be overridden. Ignored if compiled with ASLR.
• SectionAlignment: Alignment of the sections when loaded in memory, cannot be less than page size (4096 bytes). Sections have to occupy space of multiples of SectionAlignment in memory.
• FileAlignment: Alignment of the sections in the raw file, usually 512.
• Magic: Slightly overhyped term for File Signature (Sorry, nothing magical here).
• NumberOfSections: Number of sections defined after header, discussed later.
• SizeOfHeaders: Combined size of all headers (including DOS header, PE header, PE optional header and section headers).
• Checksum: The image file checksum.
• SizeOfOptionalHeader: As it says. Optional header contains data like preferred ImageBase, EntryPoint, Checksum and many other fields.

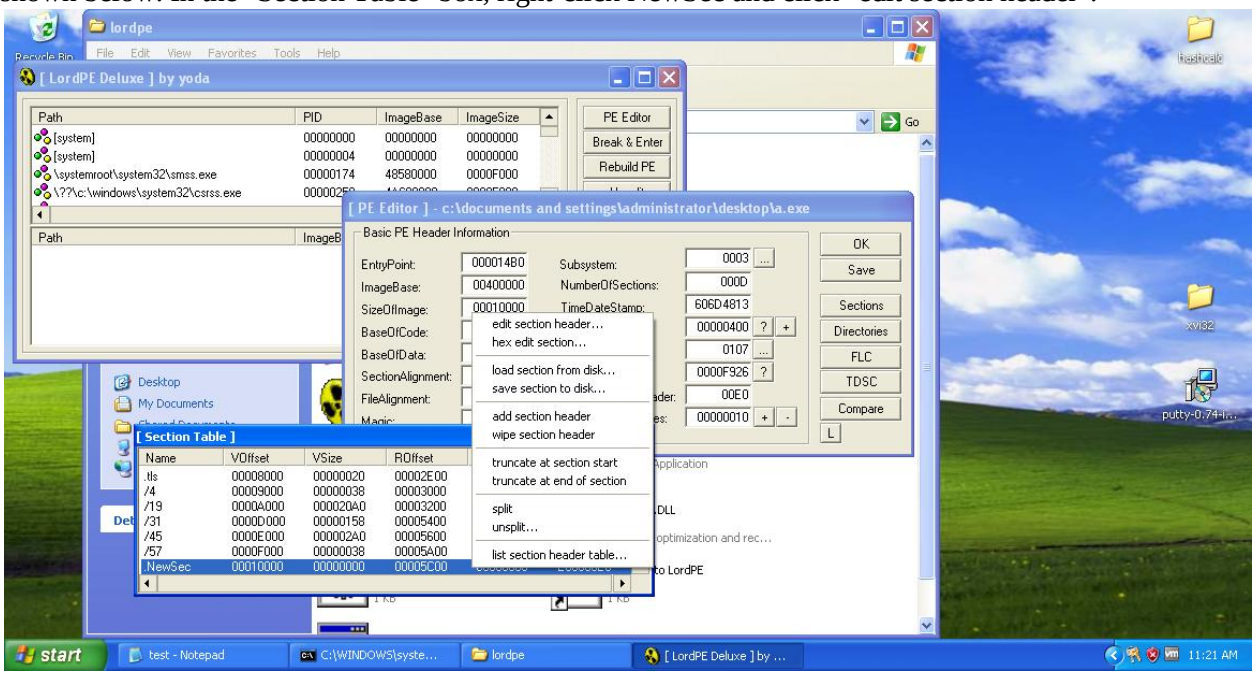


Right-click one of the sections and click "add section header", as shown below.
As NumberOfSections shows, we have 12 sections.
The .text section contains the executable code, so by default it needs to be readable and executable.
.data and .rdata contains read-only data, executing content inside this section is possible by setting the Executable flag.
.rsrc contains resource data,
.reloc section is usually not needed unless there are base address conflicts in memory.
These are standard sections present in almost every executable.

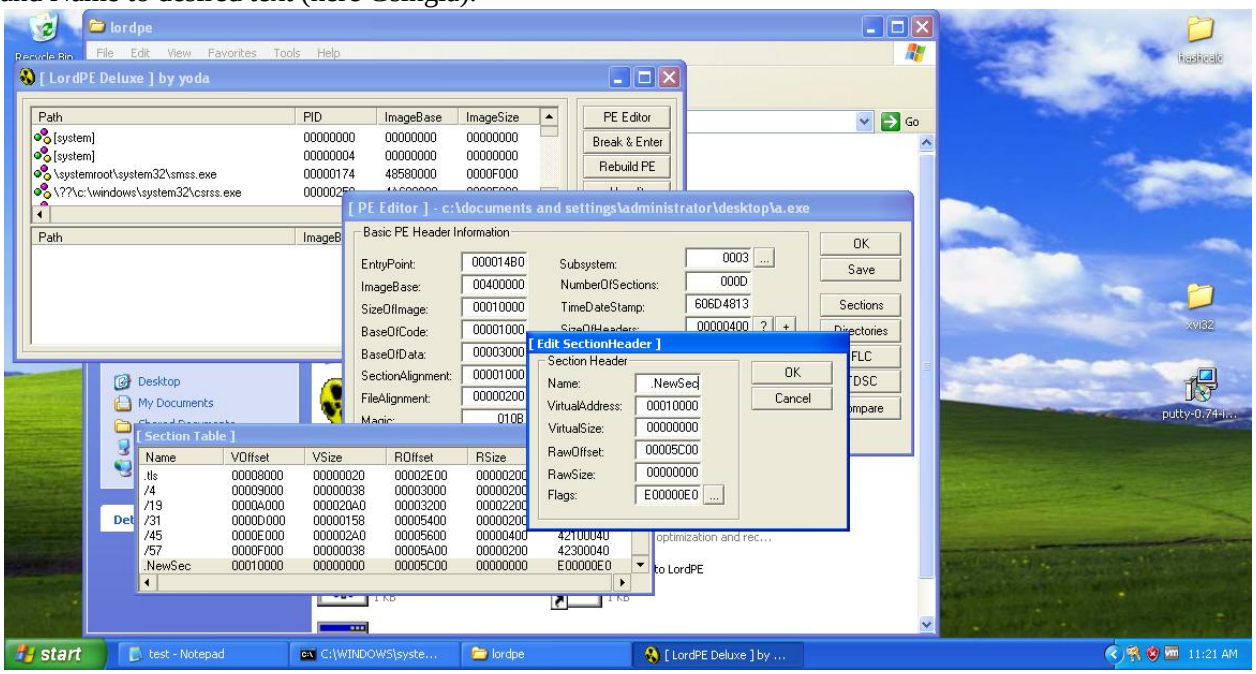
- Voffset: Offset of the section from the ImageBase when loaded into memory.
- VSize: Size of the section when loaded into memory.
- ROffset: Real file offset on disk, this can be verified using your preferred HEX editor tool.
- RSize: Real size of the section on disk.
- Flags: Contains flags defining "permissions" on sections. For easy viewability, right click a section - > Edit SectionHeader then the small box next to Flags text field.

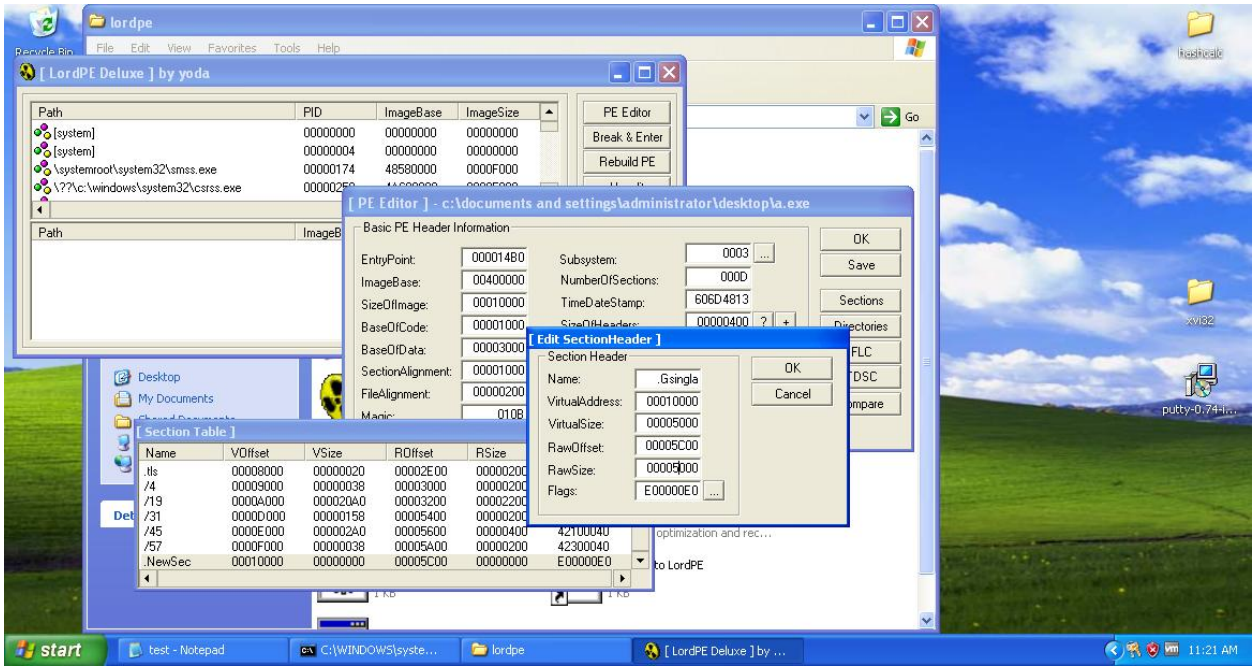


A new section named "NewSec" appears. Currently, this section has "VSize" and "RSize" values of 0, as shown below. In the "Section Table" box, right-click NewSec and click "edit section header".



In the "[Edit SectionHeader]" window, change the VirtualSize and RawSize to 00005000 as shown below and Name to desired text (here Gsingla).

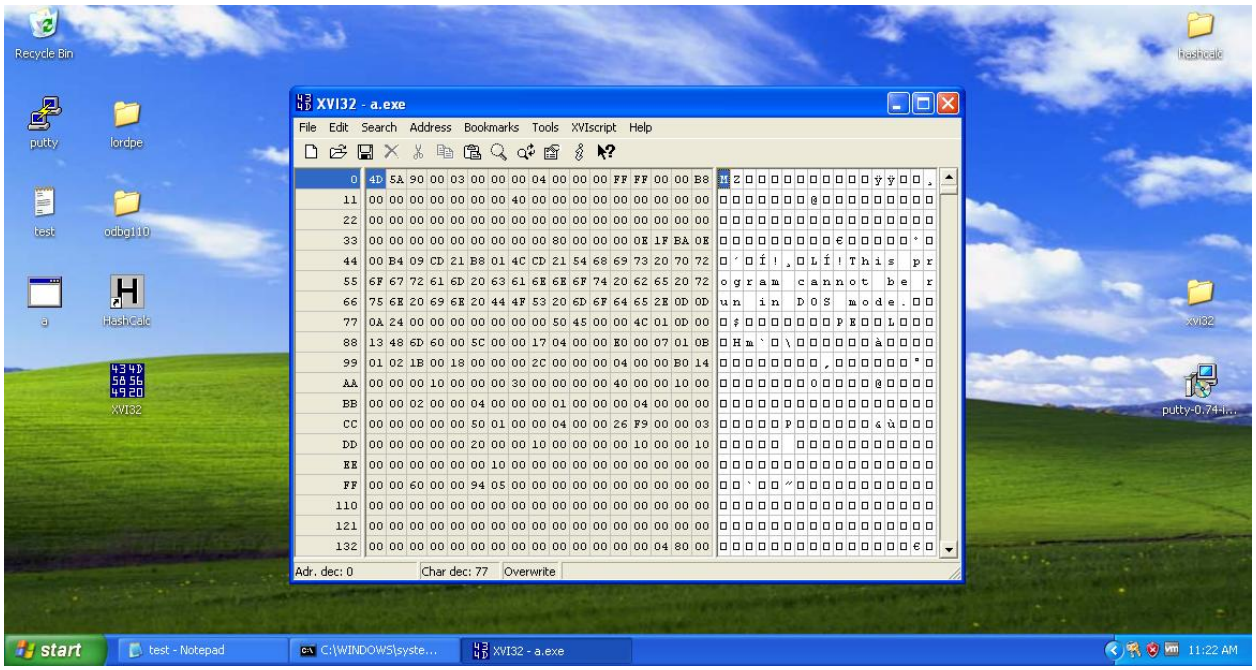




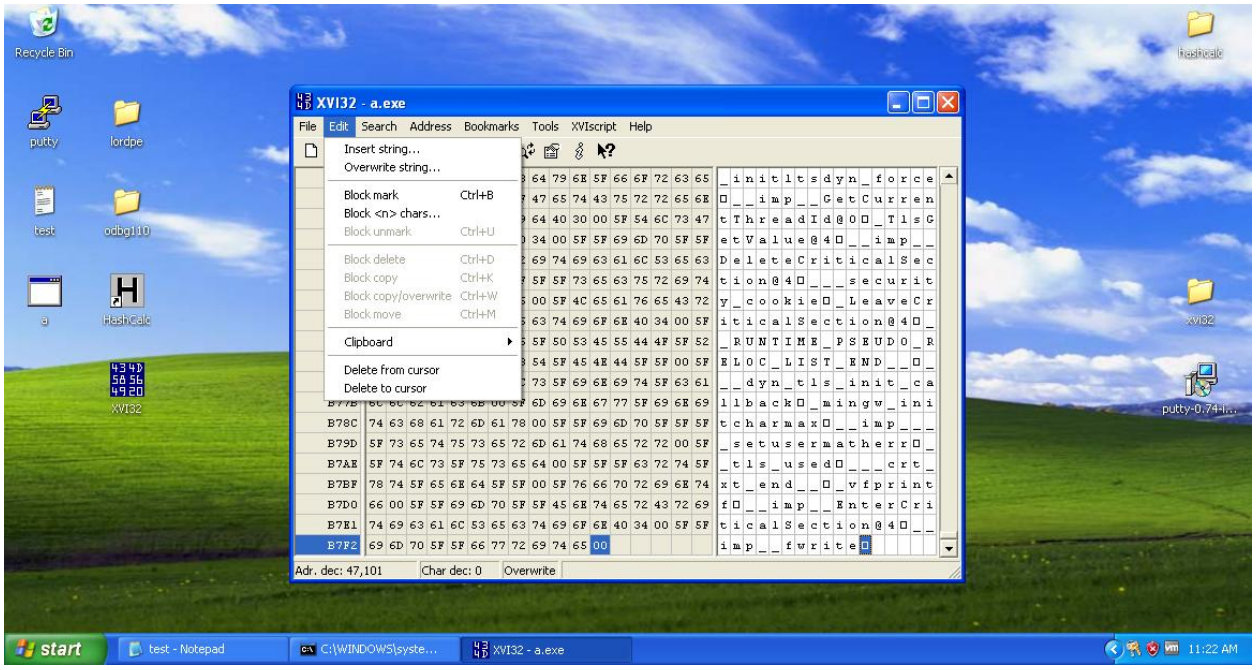
Close the "Section Table" box.
In the "PE Editor" box, click the Save button.
In the "PE Editor" box, click the OK button.
Close the LordPE window.

As a new section has been added, fill that section with desired strings using XVI32 to recognize the section.

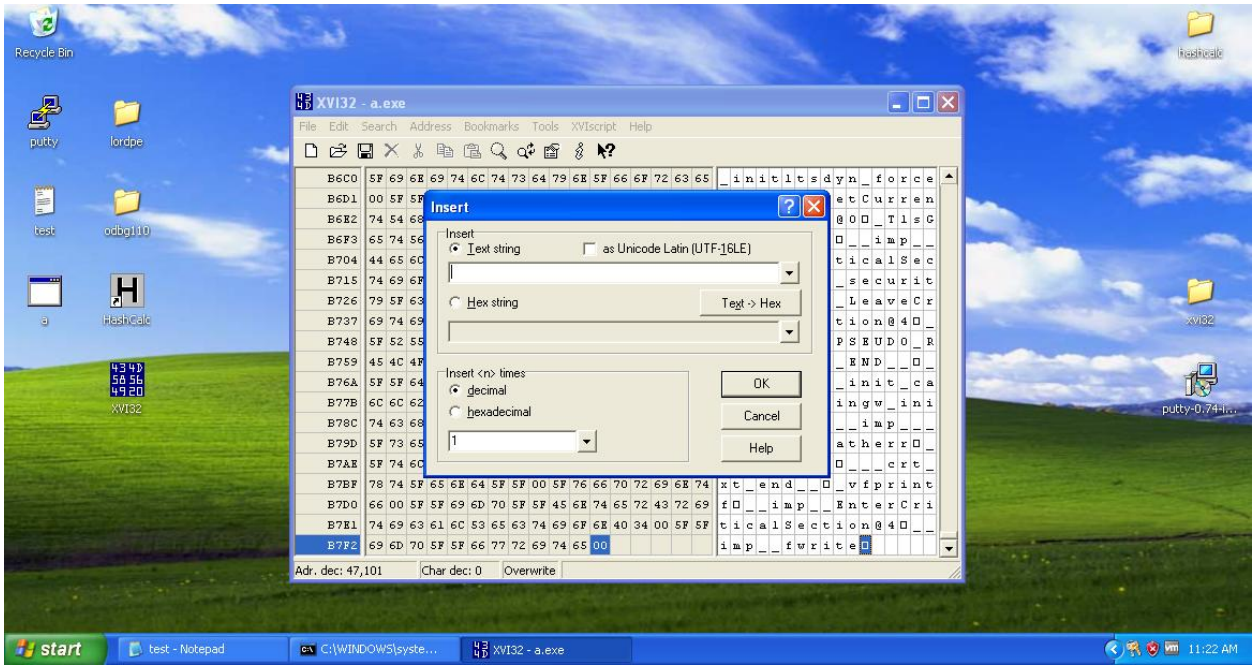
Open a.exe in XVI32 as shown.



Get to the end of the memory dump as the new section was added in the end. And then insert the string from Edit header.

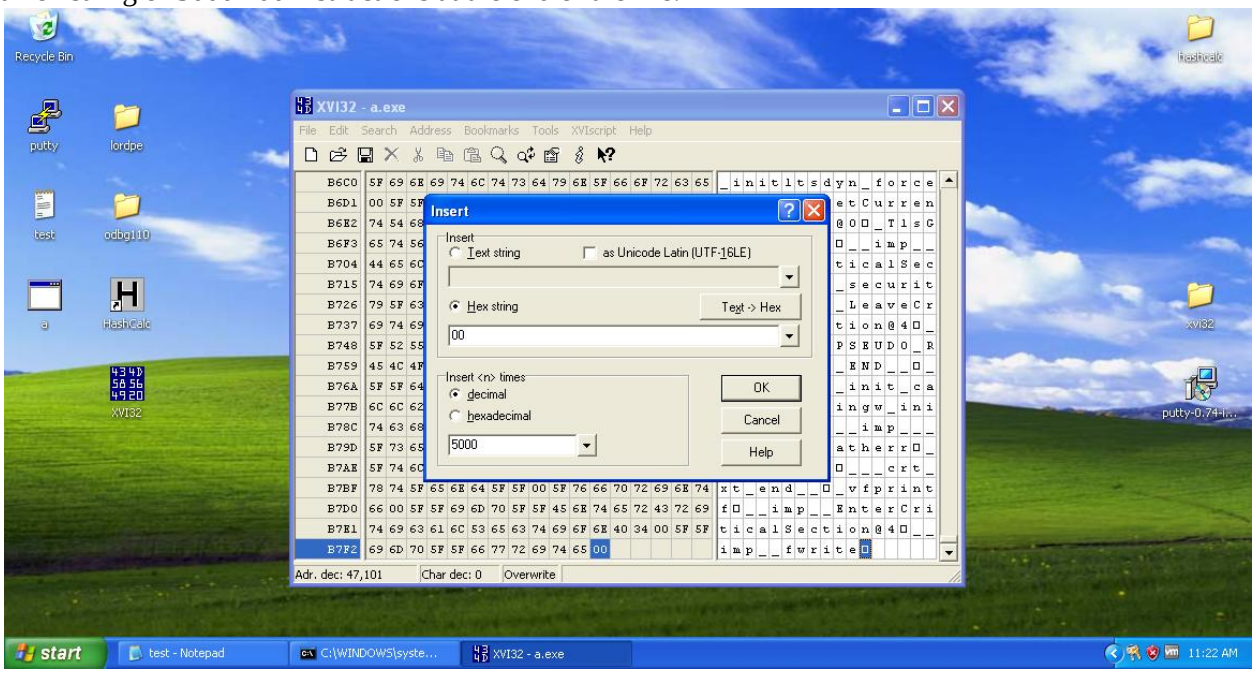


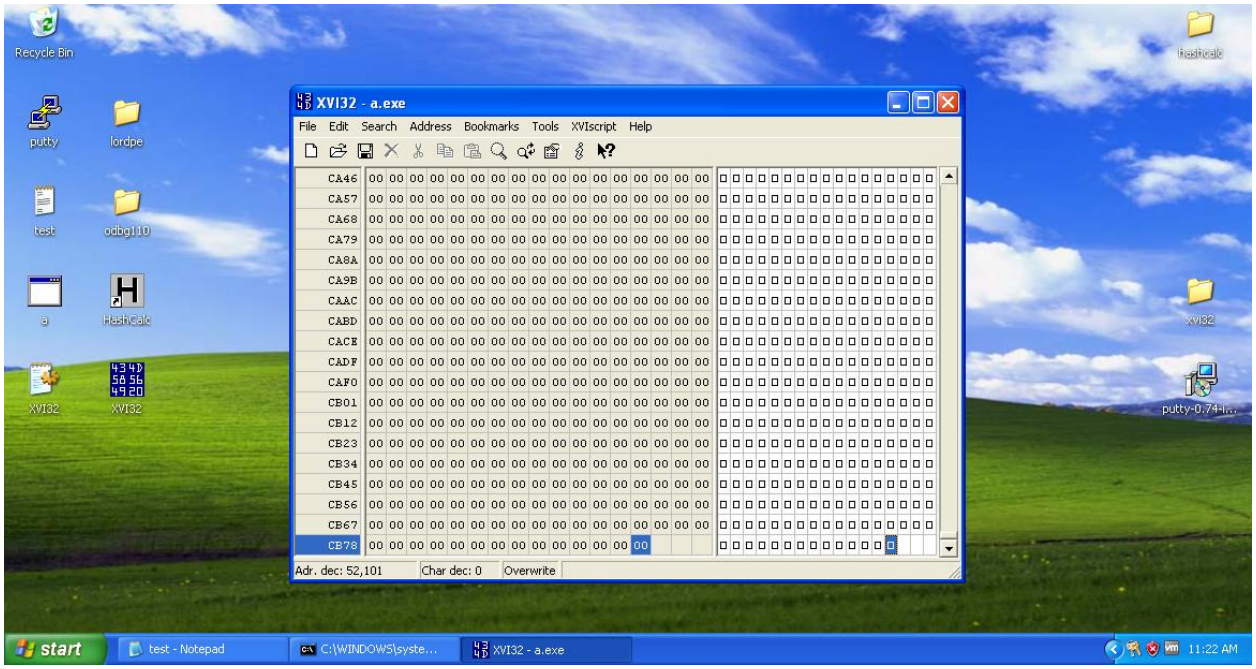
A Insert dialog box appears with options of string to be included.



Since we have added a new section to the binary and left it empty. So now we will use a hex-editor to add the extra 5000 bytes to the binary. We will open the executable file in a hex editor (like XVI32) and insert

a hex string of 5000 x00 instructions at the end of the file.





Task 2: Redirecting Code Execution with Ollydbg

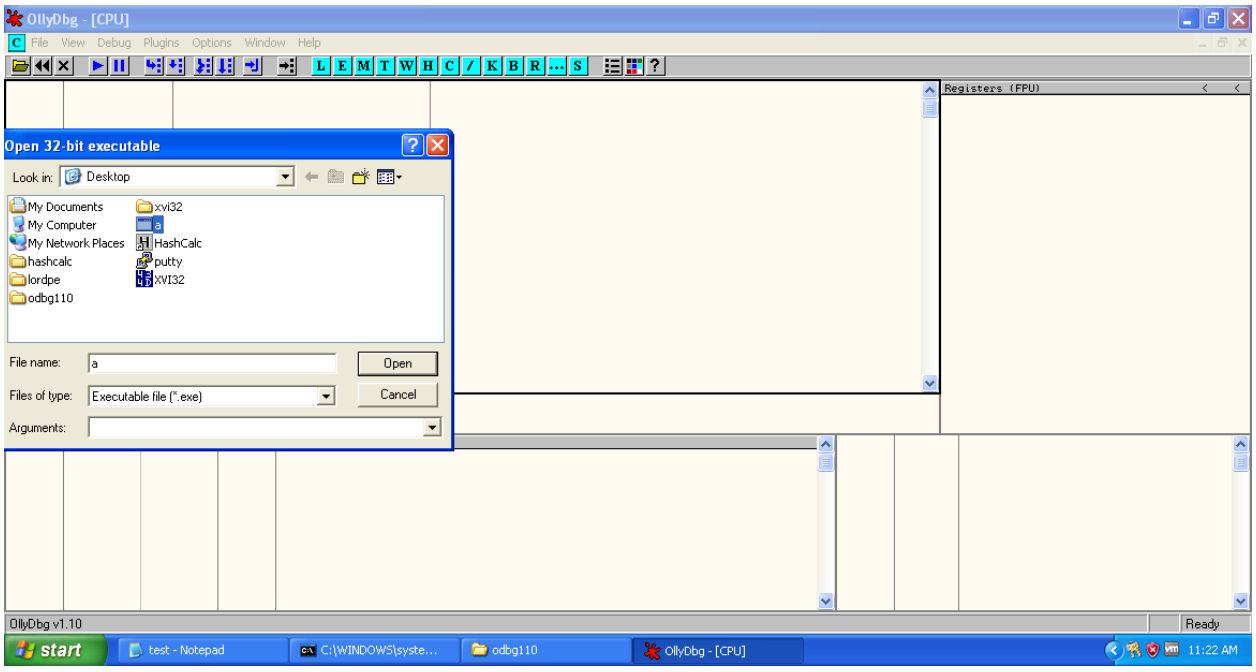
In context of backdooring, a code cave is a new or unused dead space where we can put custom code and redirect the execution to it, without breaking the actual executable.
There are a Couple of techniques:

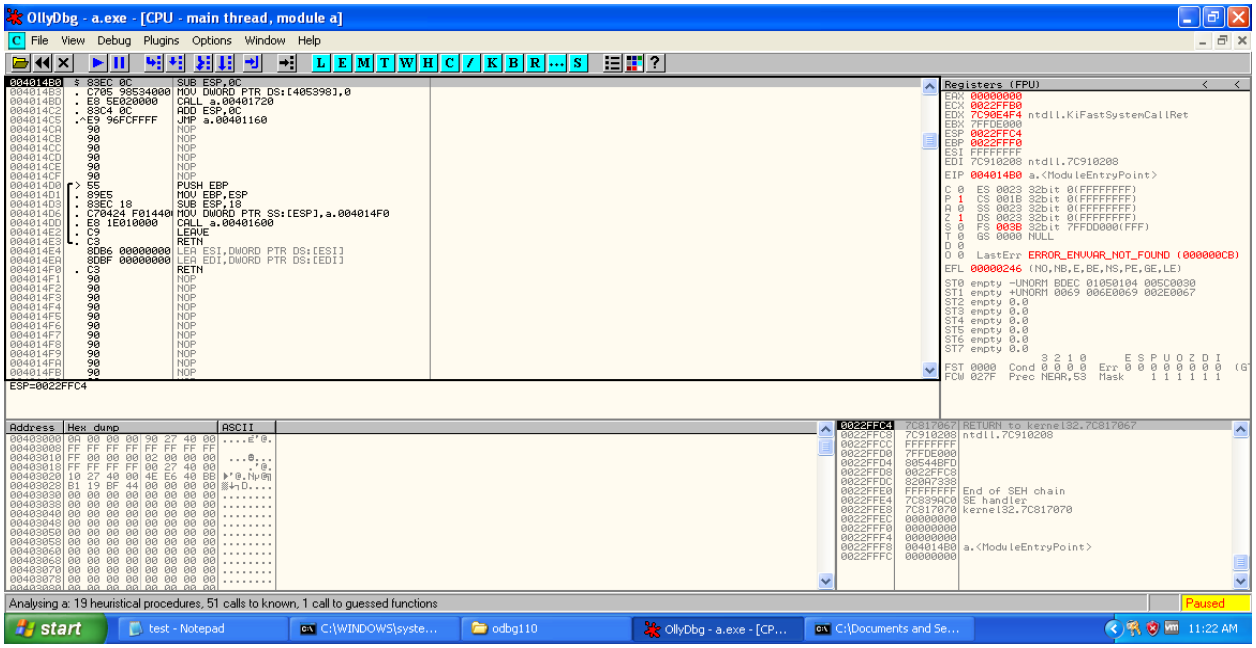
- Adding a new section
  - Pros: Lots of space.
  - Cons: Binary size increases, more susceptible to get flagged as malicious.
- Using existing dead space
  - Pros: File size doesn't change, less susceptible to get flagged as malicious.
  - Cons: Might be very low on space, section permissions might need to change to allow code execution.
- Extending last section
  - Pros: Number of sections doesn't change.
  - Cons: Binary size increases, more susceptible to get flagged as malicious, heavy dependency on the last section. Doesn't perform better than adding a new section.
- Cave jumping
  - Pros: Flexible, can utilize a single or a mix of existing techniques. Possibly stealthier.
  - Cons: Tricky to break payload into smaller parts, might require changing permissions on multiple sections.

Let’s hijack the entry point of the program and then redirect the execution of the a.exe executable to the newly added space which holds the Trojan code and after executing the that code, we will redirect the flow back to the normal execution of the program.

Using Ollydbg to Examine the .Gsingla Section

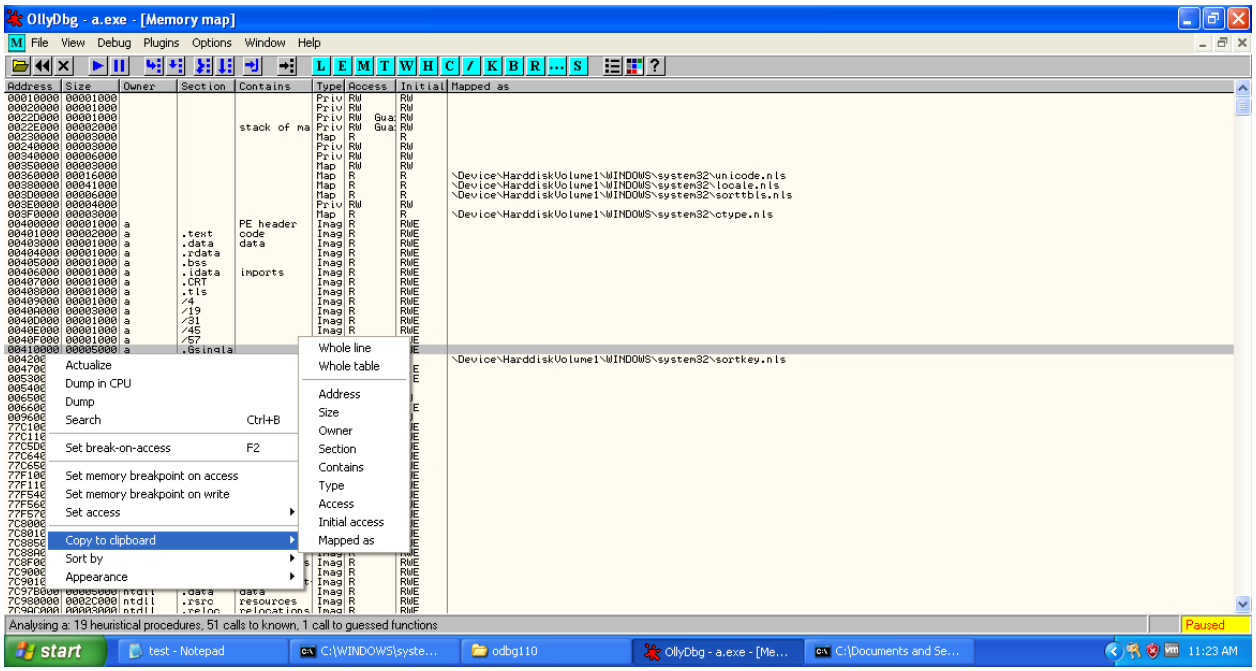
Launch Ollydbg.
In Ollydbg, from the menu bar, click File -> Open. Navigate to a.exe and open it.



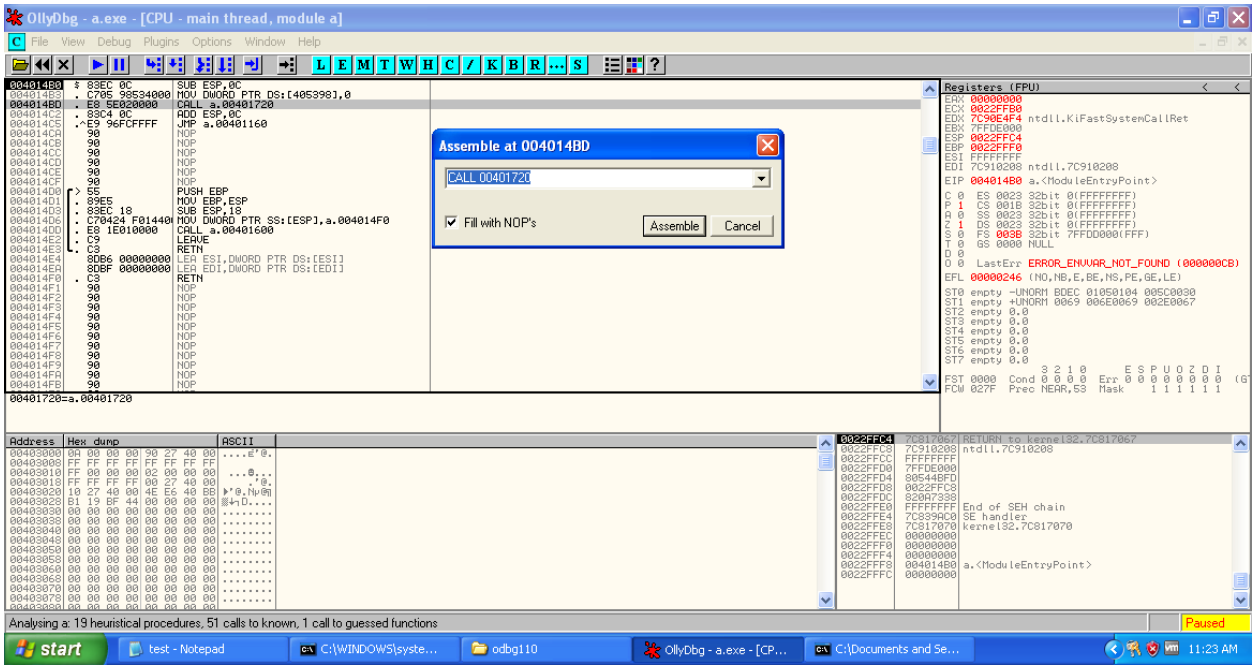


From the Ollydbg menu bar, click View, Memory.

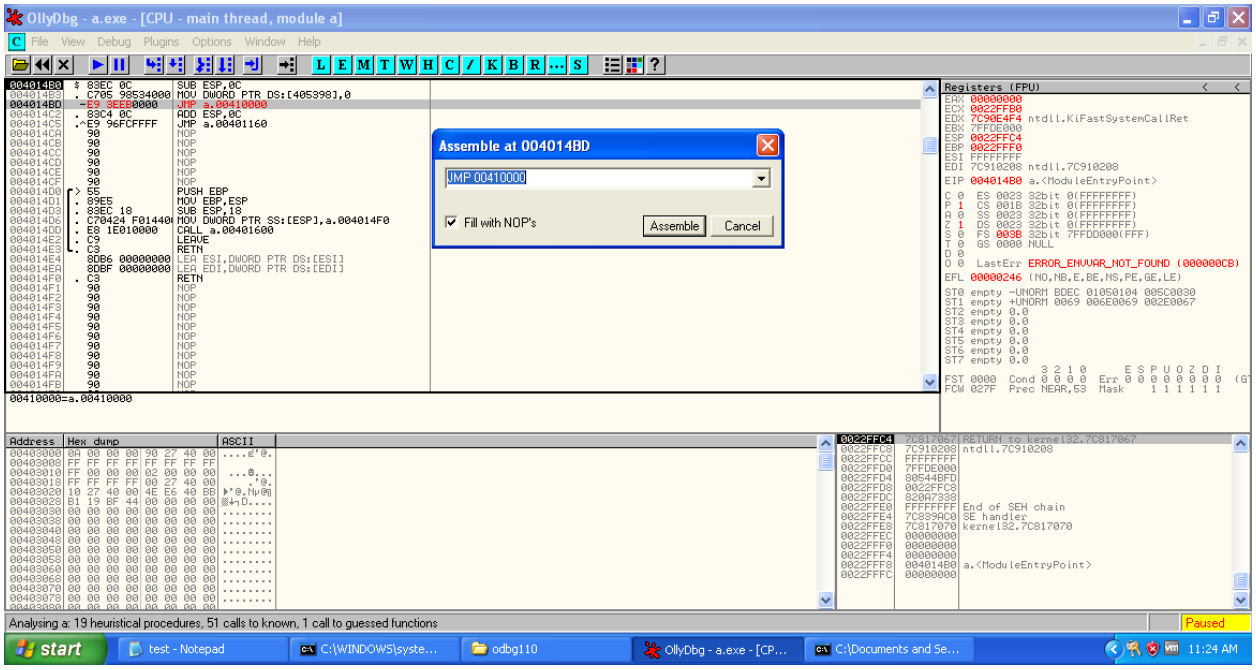
Ollydbg shows the memory layout of putty. As highlighted in the image below, the ".Gsingla" section begins at address 00410000. Copy the address by right clicking on the section.



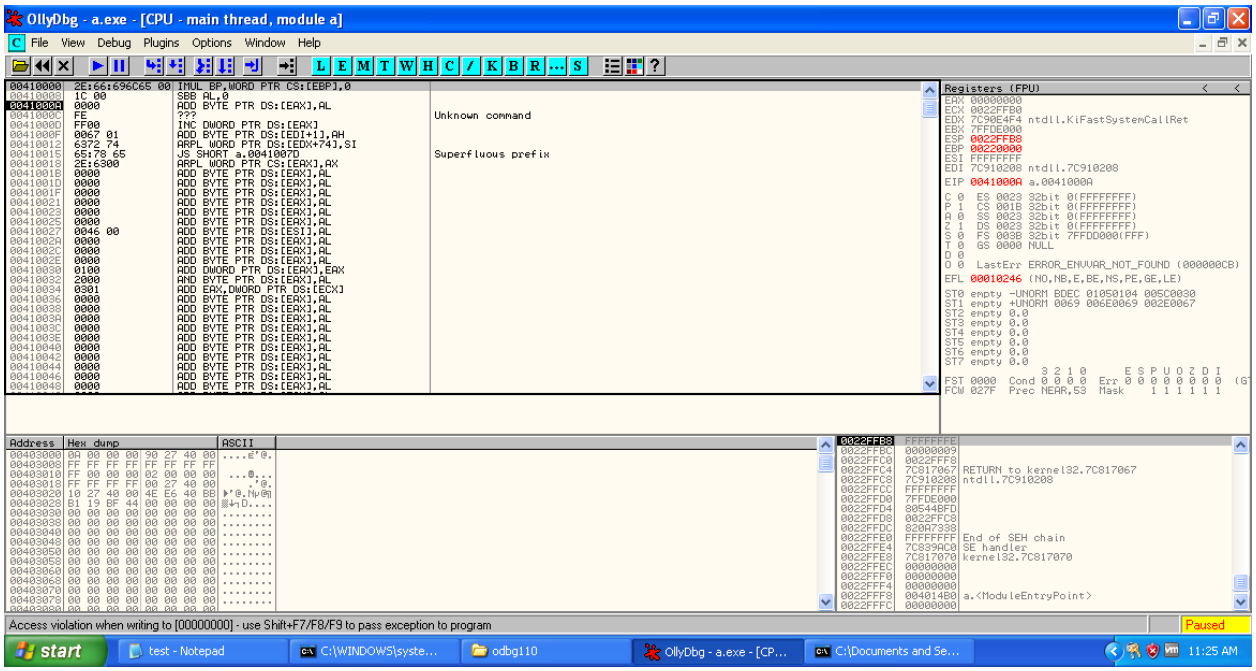
Get back to the a.exe main thread window and select the first address of a.exe ‘004014BD’ right click -> Assemble or press spacebar.



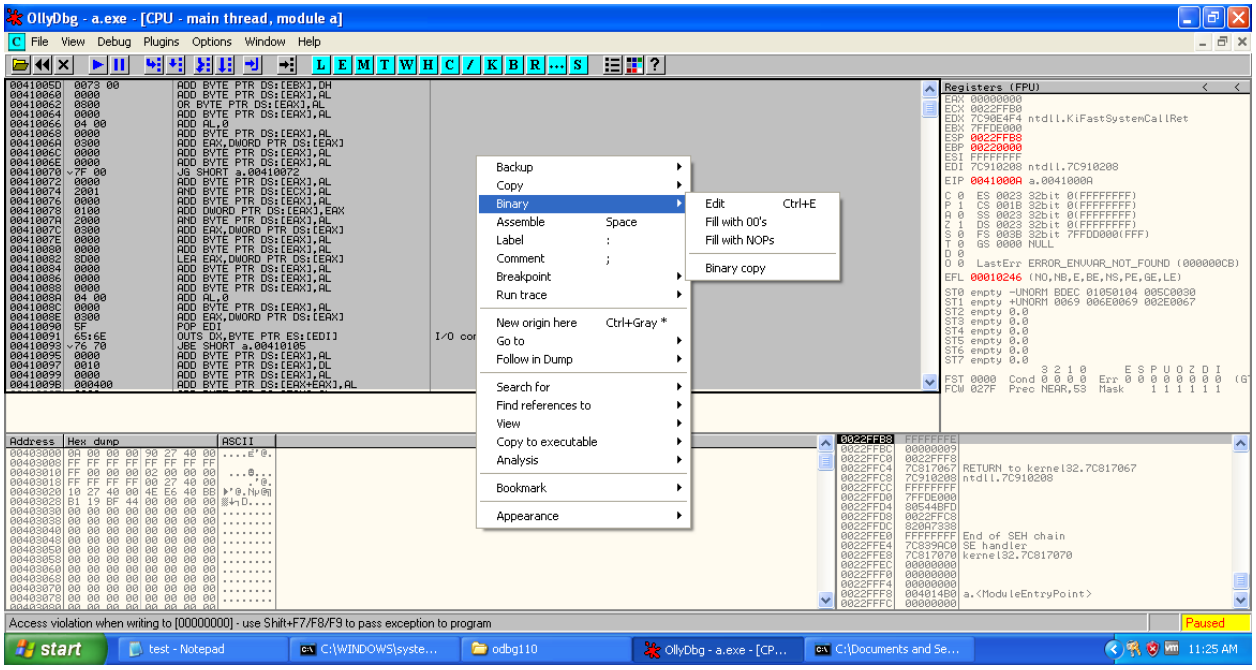
In the "Assemble" box, enter this command, as shown below:
JMP 00410000
Click the Assemble button.
Click the Cancel button.
The MOV instruction has been replaced by this instruction, as shown below:
JMP a.00410000

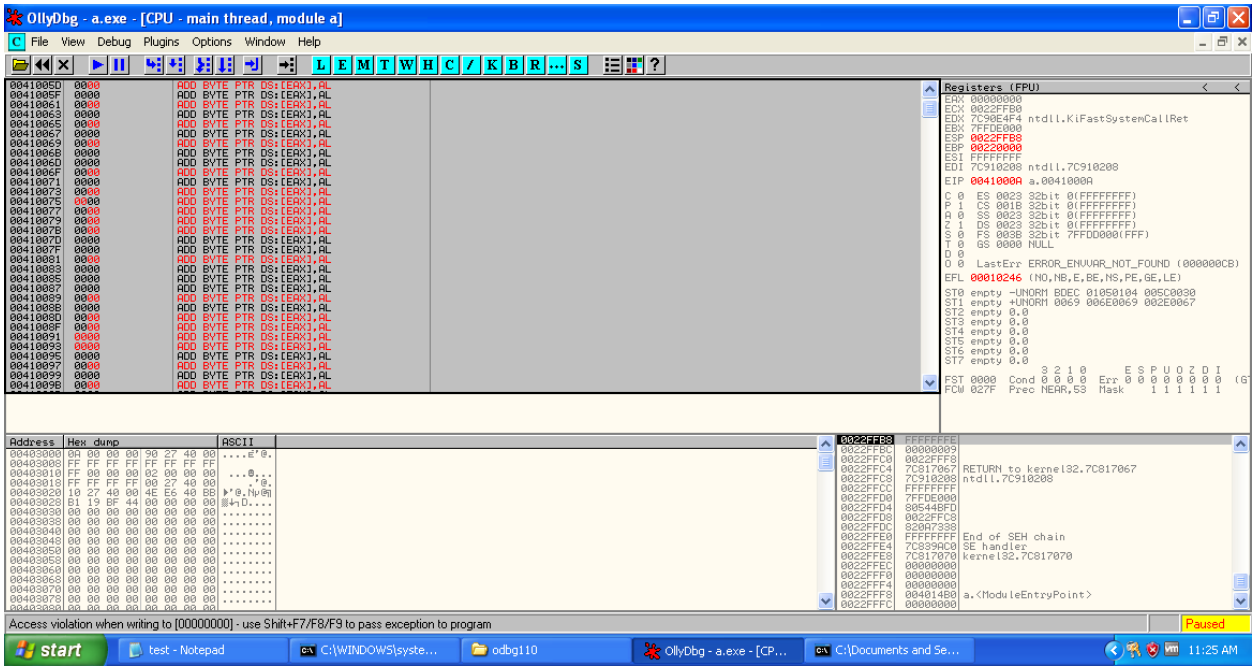


Let’s step into (F7) this instruction to jump to our code cave.

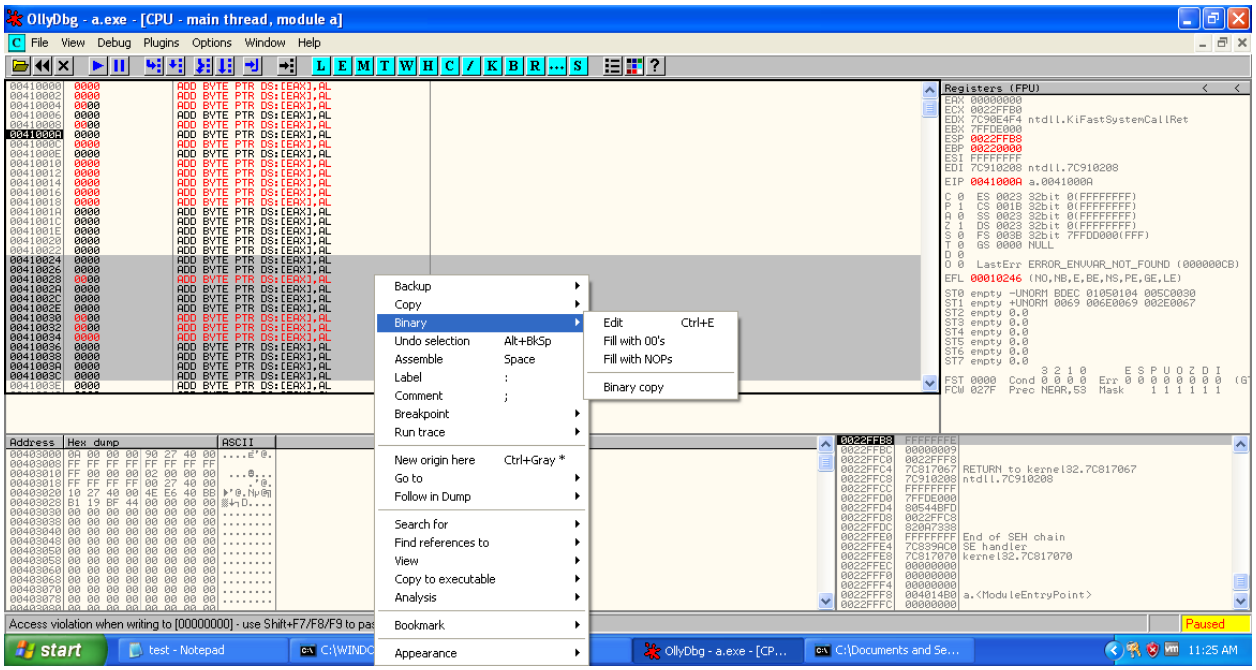


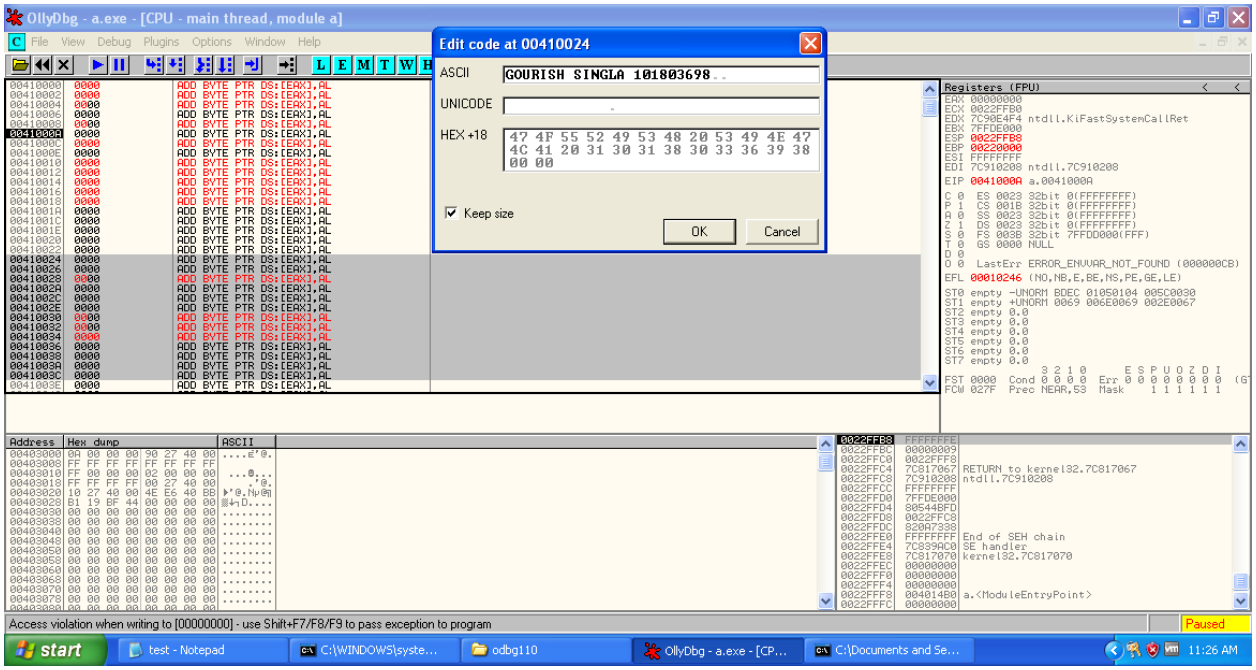
Select the sizeable amount of instructions from 00410000 and fill with 00’s to just identify which section is to be altered.



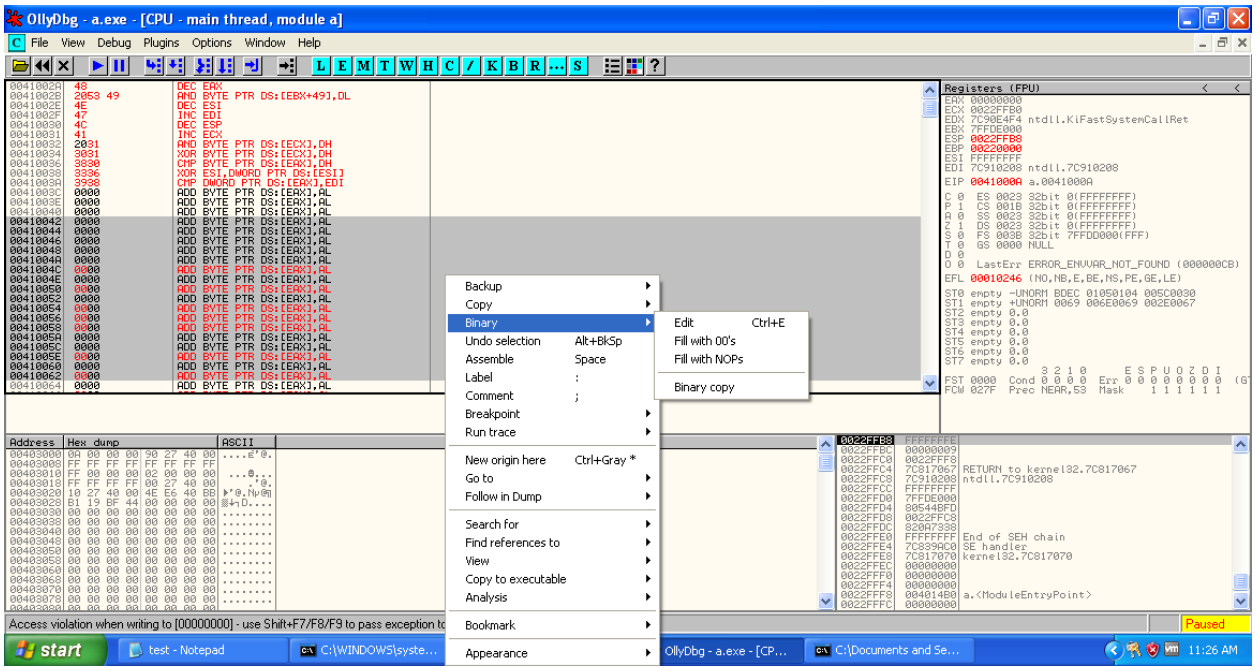


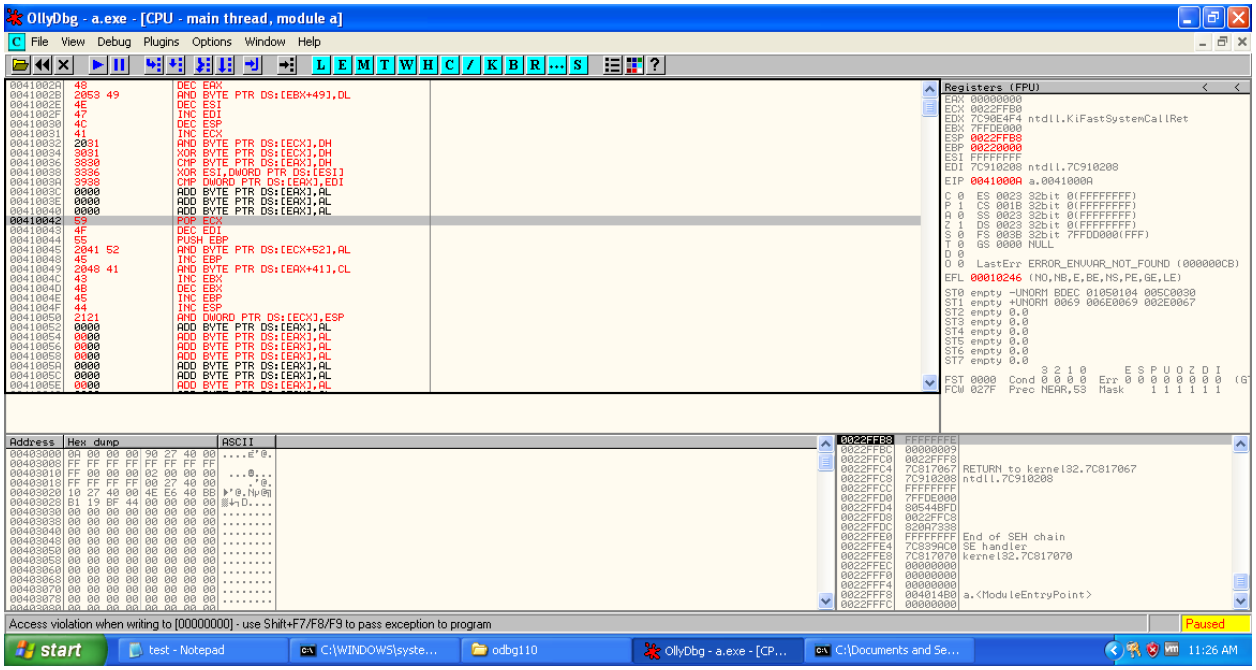
Select some portion and Edit the binary with the string ‘GOURISH SINGLA 101803698’ and note down the starting address 00410024.





Select another chunk of addresses and edit the string with ‘YOU ARE HACKED!!’ and note down the starting address 00410042.





Now, code for Message Box.

Syntax

```
C++
```

Copy

```
int MessageBox(
  HWND    hWnd,
  LPCTSTR lpText,
  LPCTSTR lpCaption,
  UINT    uType
);
```

Parameters

hWnd Type: **HWND**

A handle to the owner window of the message box to be created. If this parameter is **NULL**, the message box has no owner window.

lpText Type: **LPCTSTR**

The message to be displayed. If the string consists of more than one line, you can separate the lines using a carriage return and/or linefeed character between each line.

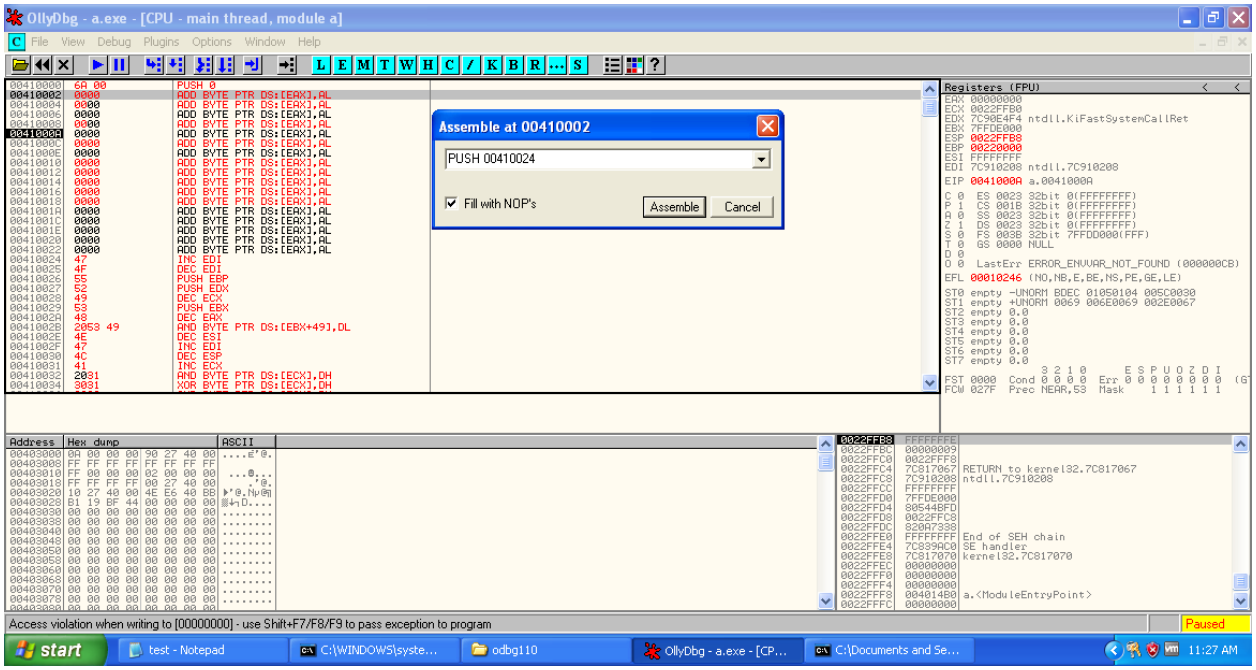
lpCaption Type: **LPCTSTR**

The dialog box title. If this parameter is **NULL**, the default title is **Error**.

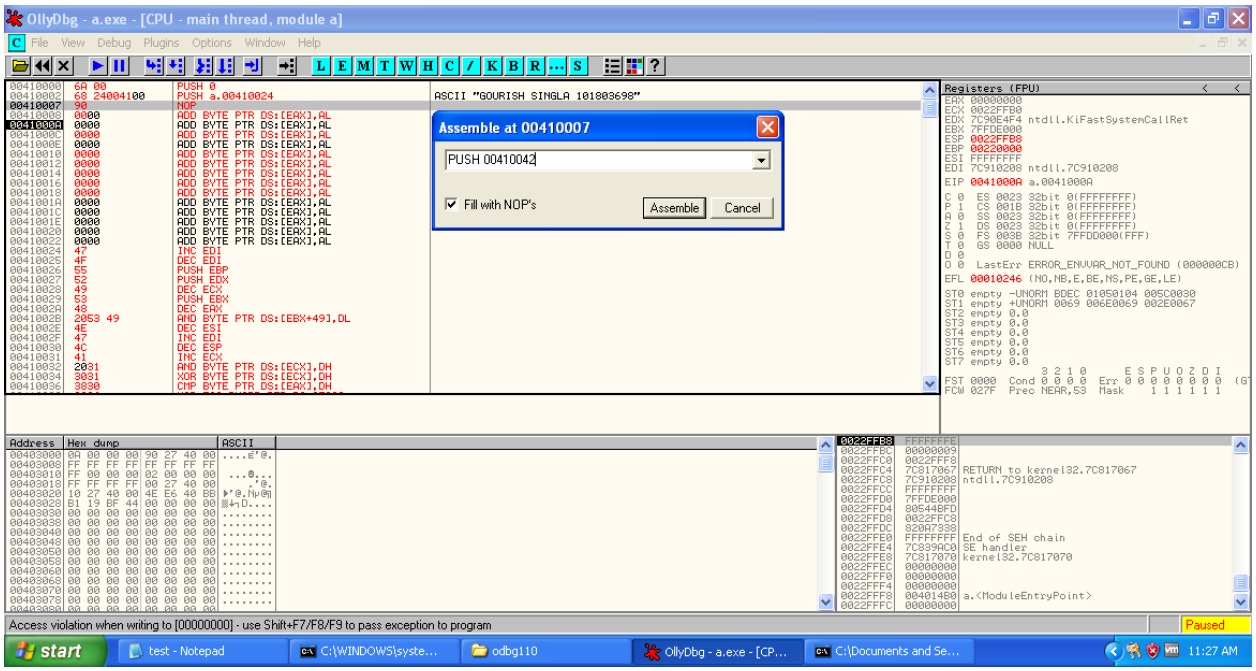
uType Type: **UINT**

The contents and behavior of the dialog box. This parameter can be a combination of flags from the following groups of flags.

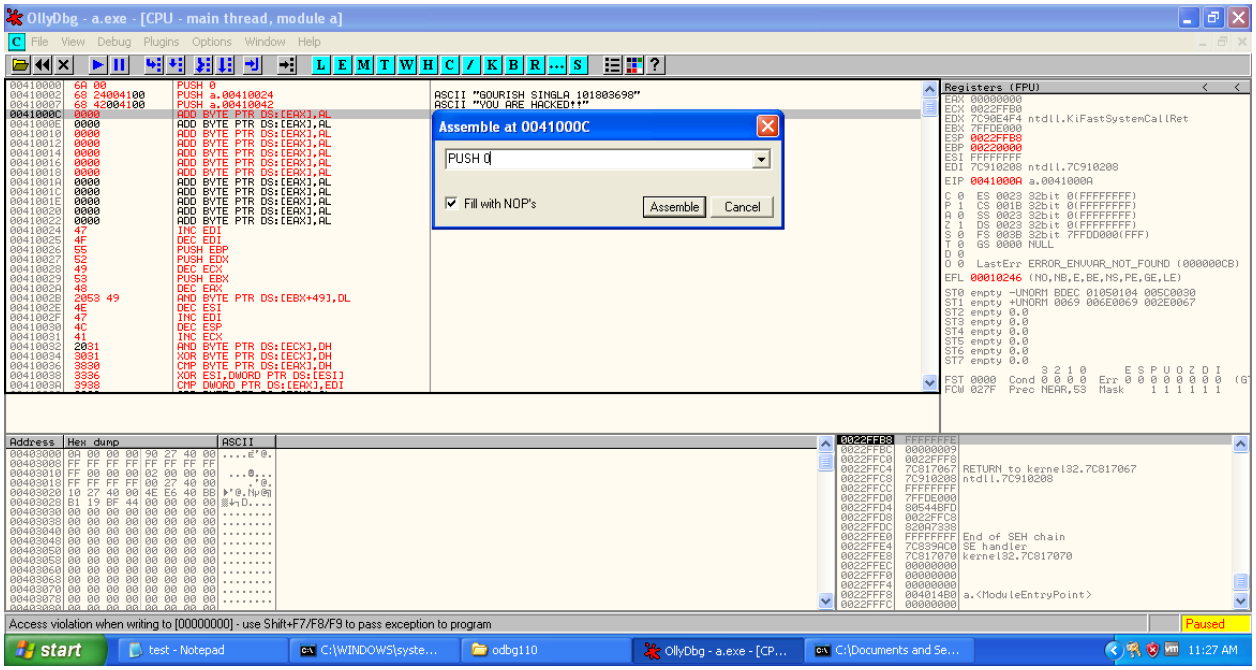
First PUSH 0 (hwnd)



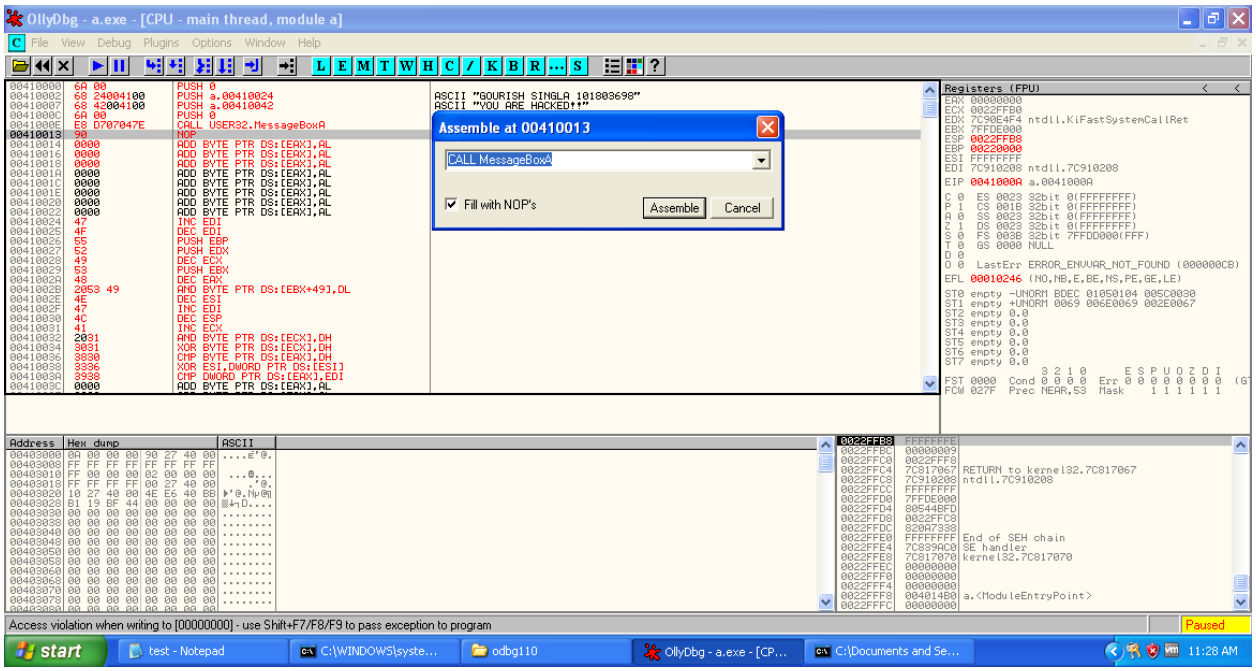
The push 00410024 (starting address of caption string)



The push 00410042 (starting address of message string)

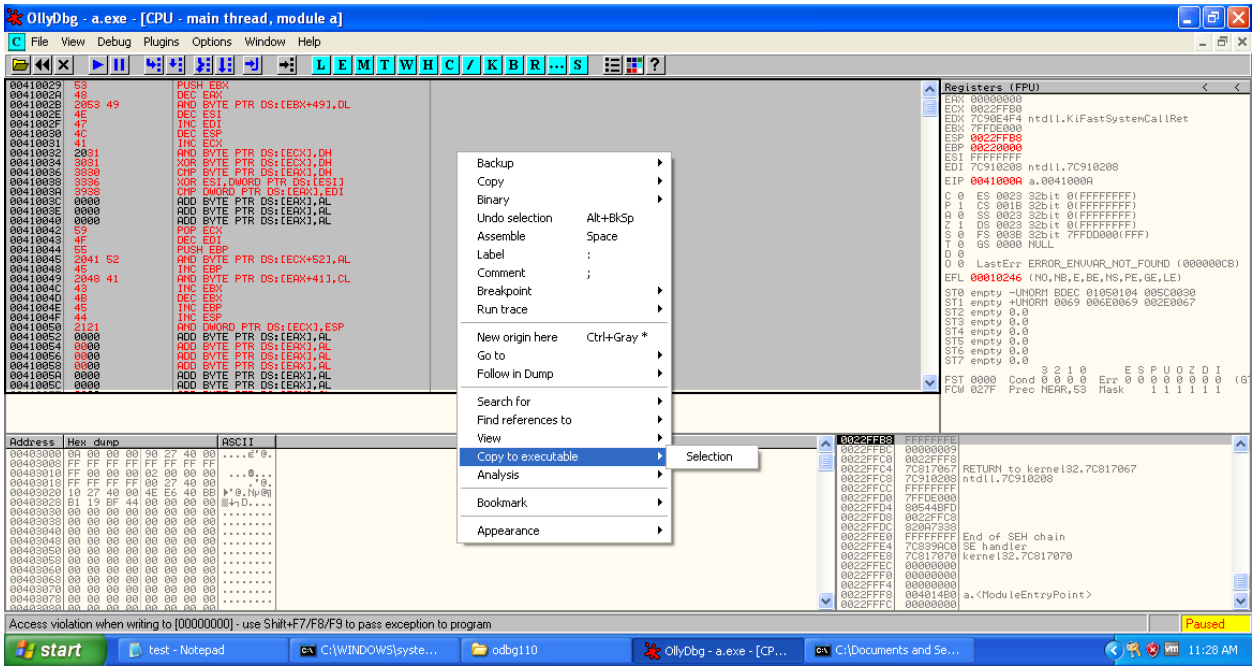


Then push 0 for message box type.

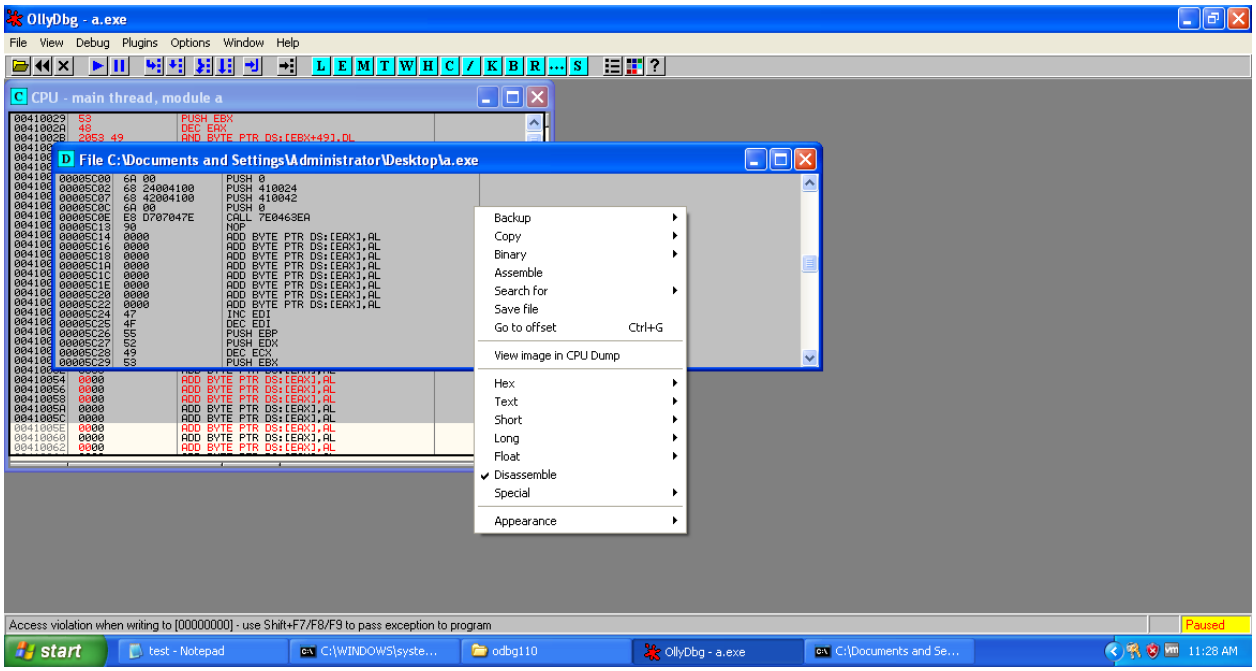


Then assemble CALL MessageBoxA to call the message box with above four parameters.

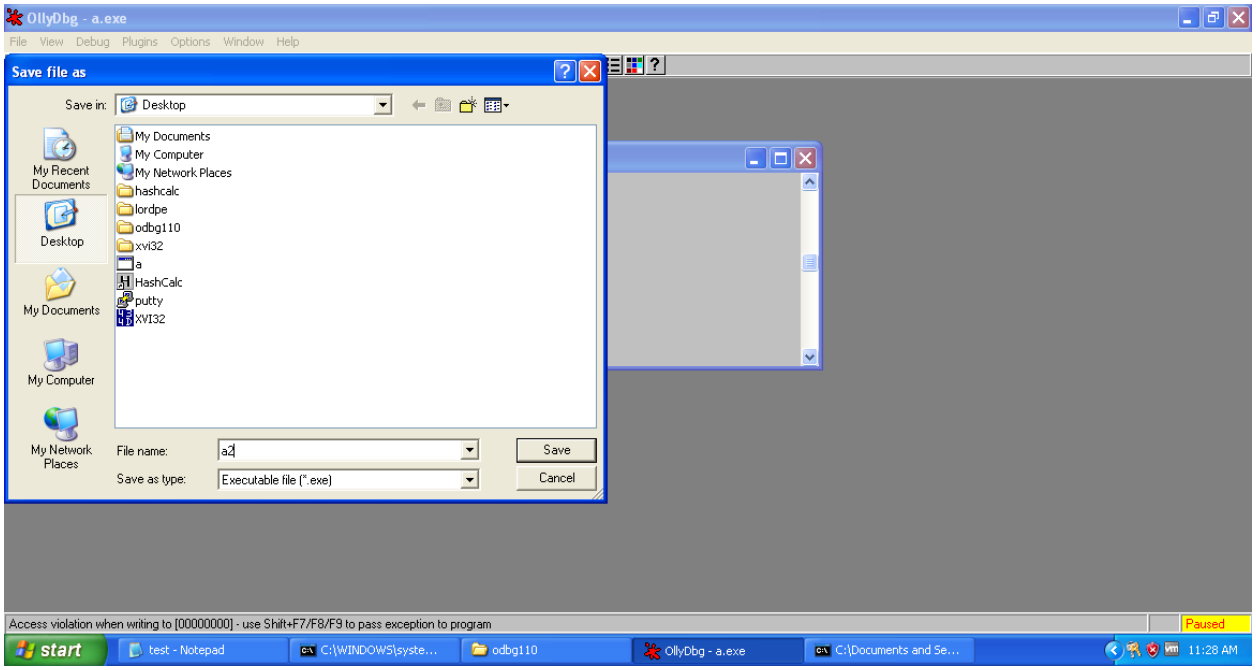
Now select all the modified assembly code to executable.



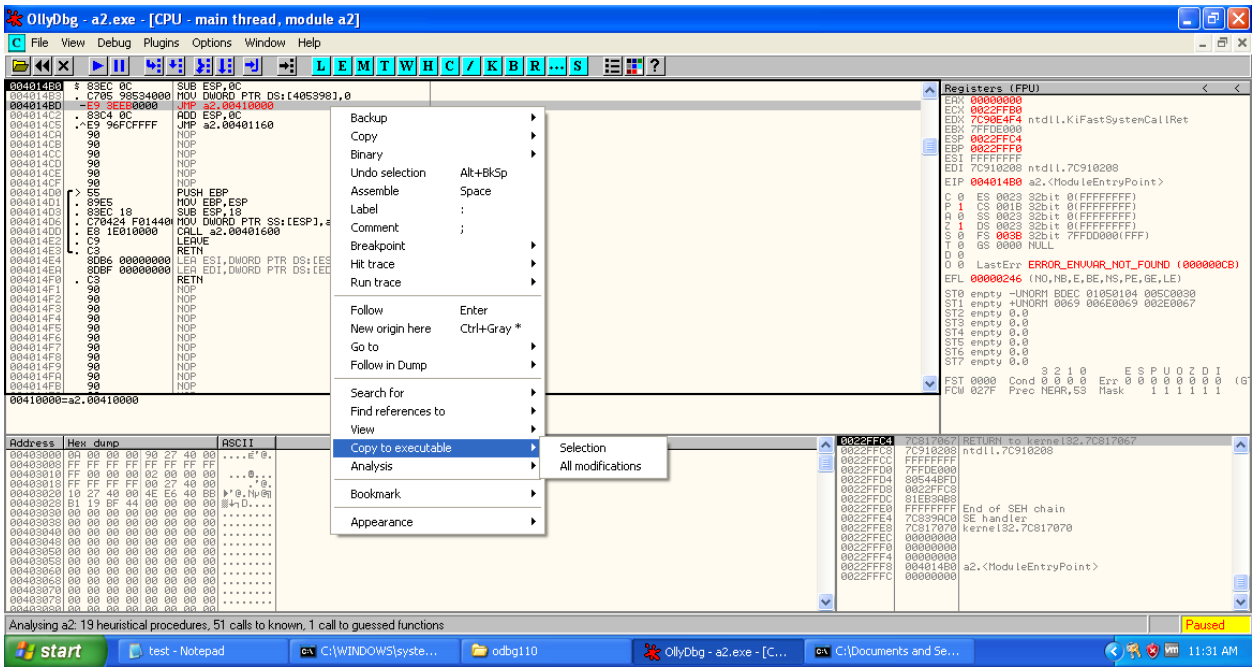
Now save the selection to get modified executable file.

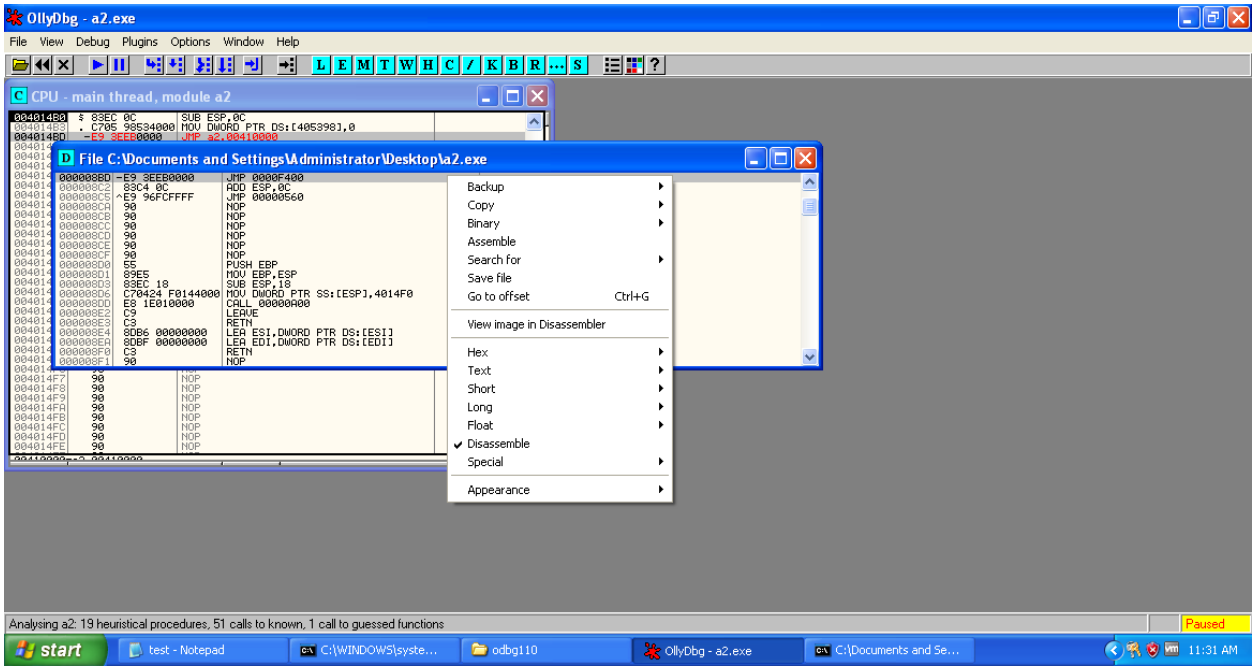


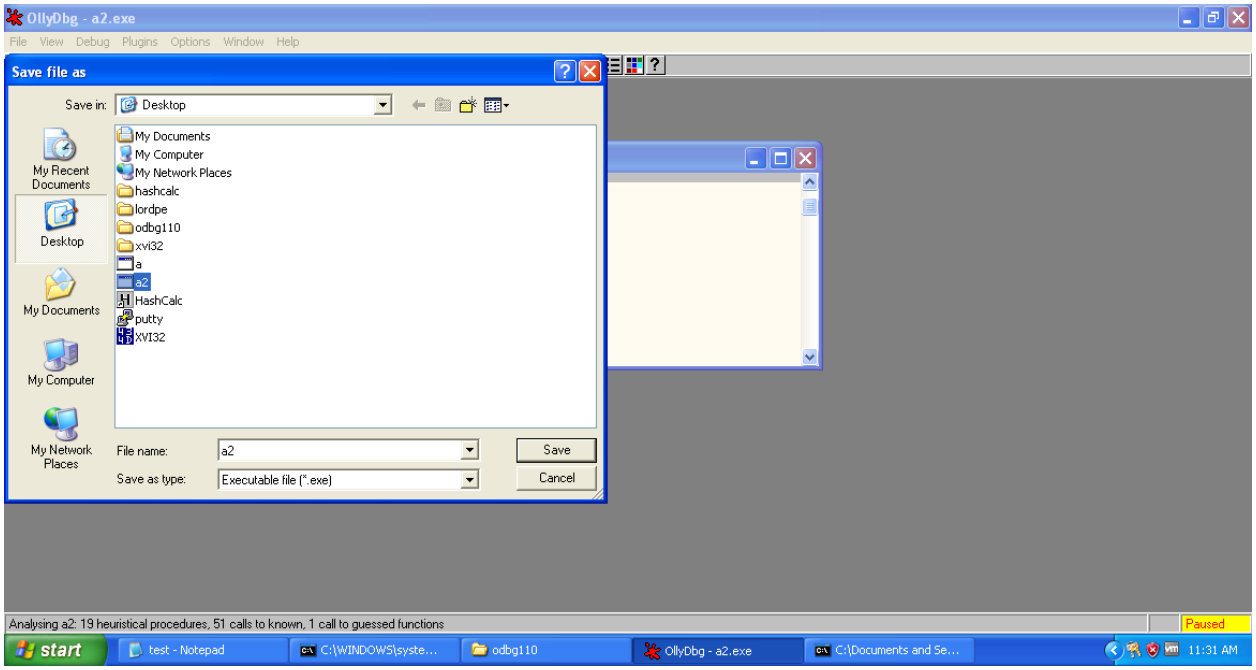
Save it as a2.exe



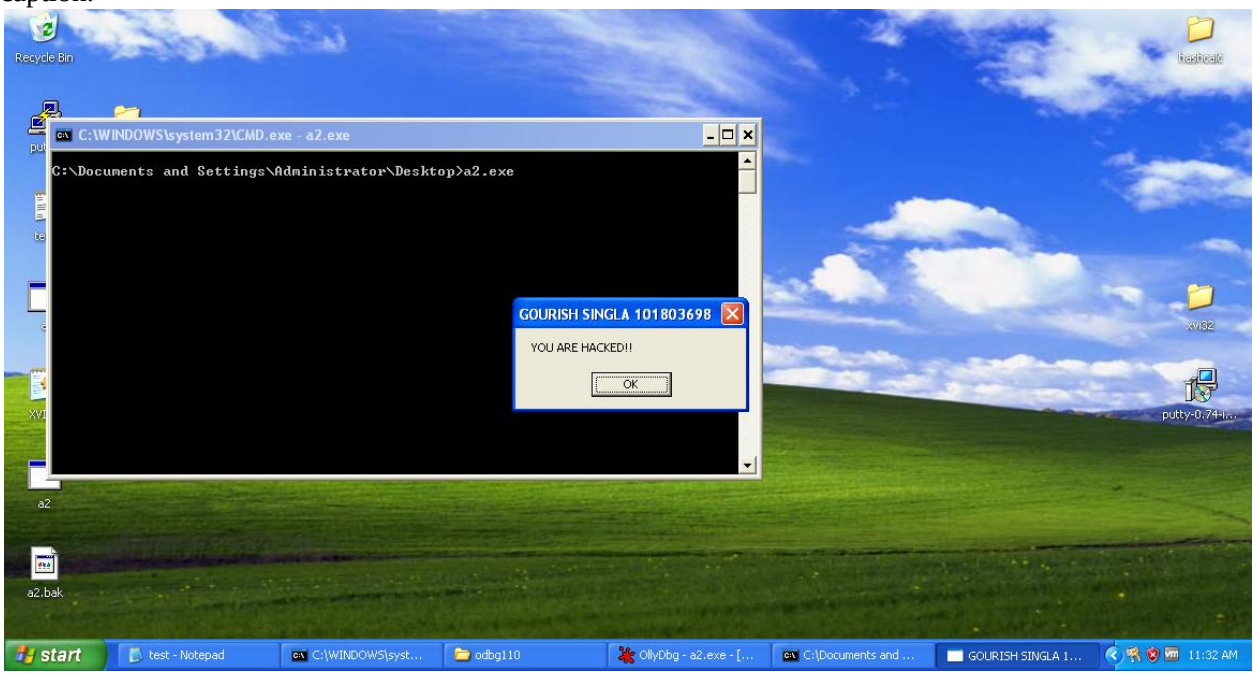
Also save the jmp statement modified at the beginning of the process to the same a2.exe.



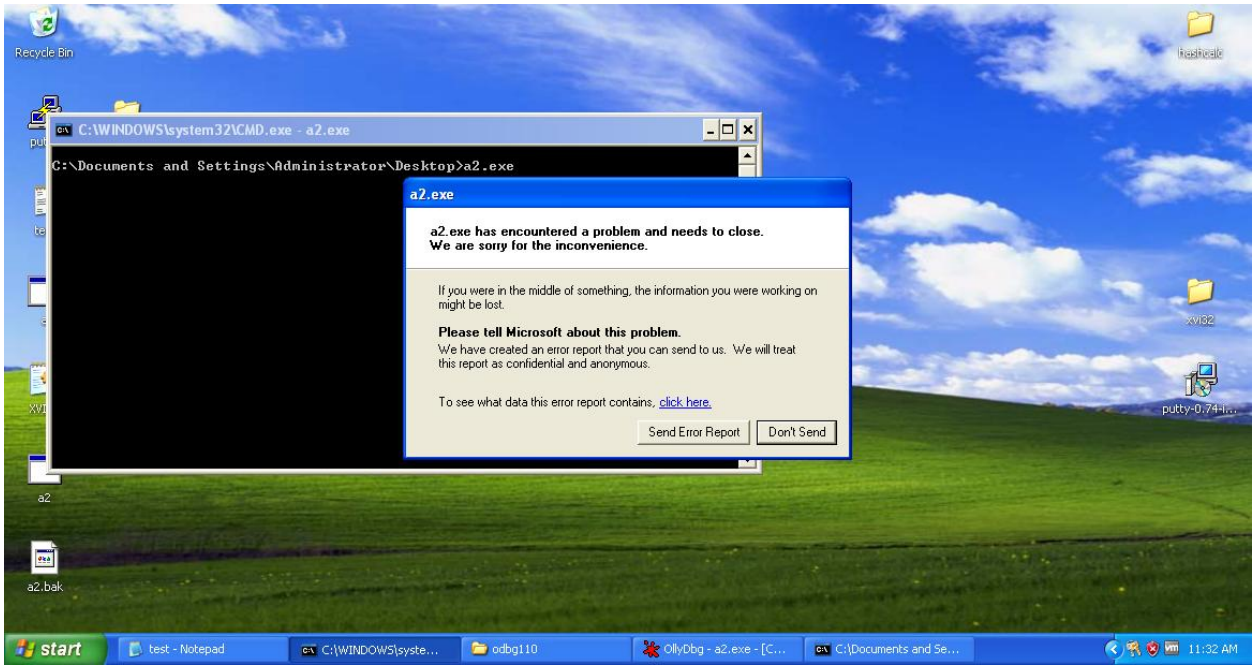

OllyDbg - a2.exe
File View Debug Plugins Options Window Help
CPU - main thread, module a2
File C:\Documents and Settings\Administrator\Desktop\a2.exe
Backup
Copy
Binary
Assemble
Search for
Save file
Go to offset Ctrl+G
View image in Disassembler
Hex
Text
Short
Long
Float
Disassemble
Special
Appearance
Analysing a2: 19 heuristical procedures, 51 calls to known, 1 call to guessed functions
Paused
start
test - Notepad
C:\WINDOWS\syste...
odbg110
OllyDbg - a2.exe
C:\Documents and Se...
11:31 AM


OllyDbg - a2.exe
Save file as
Save in: Desktop
My Documents
My Computer
My Network Places
hashcalc
lordpe
odbg110
xvi32
a
a2
HashCalc
putty
XVI32
My Recent Documents
Desktop
My Documents
My Computer
My Network Places
File name: a2
Save as type: Executable file (*.exe)
Save
Cancel
Analysing a2: 19 heuristical procedures, 51 calls to known, 1 call to guessed functions
Paused
start
test - Notepad
C:\WINDOWS\syste...
odbg110
OllyDbg - a2.exe
C:\Documents and Se...
11:31 AM

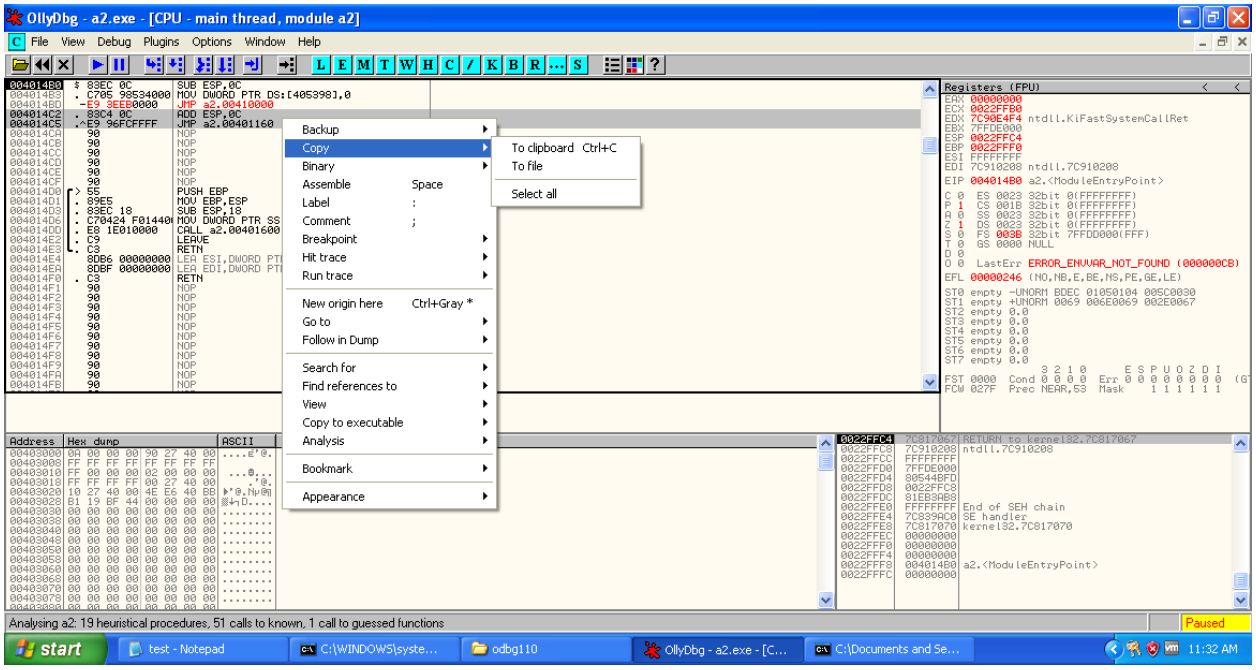
Now as we try to run the a2.exe from command line it shows message box with desired message and caption.



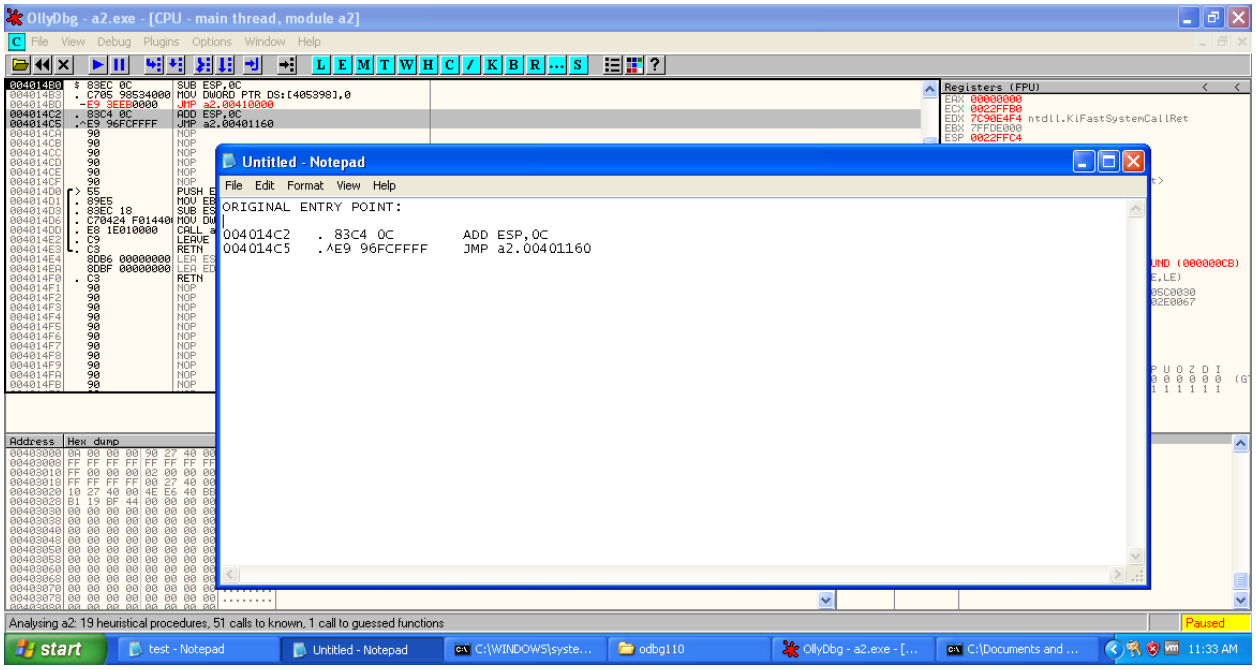
As soon as we click OK it tends to show erroroneus display because the control has not been sent to original code after execution of Trojan code.



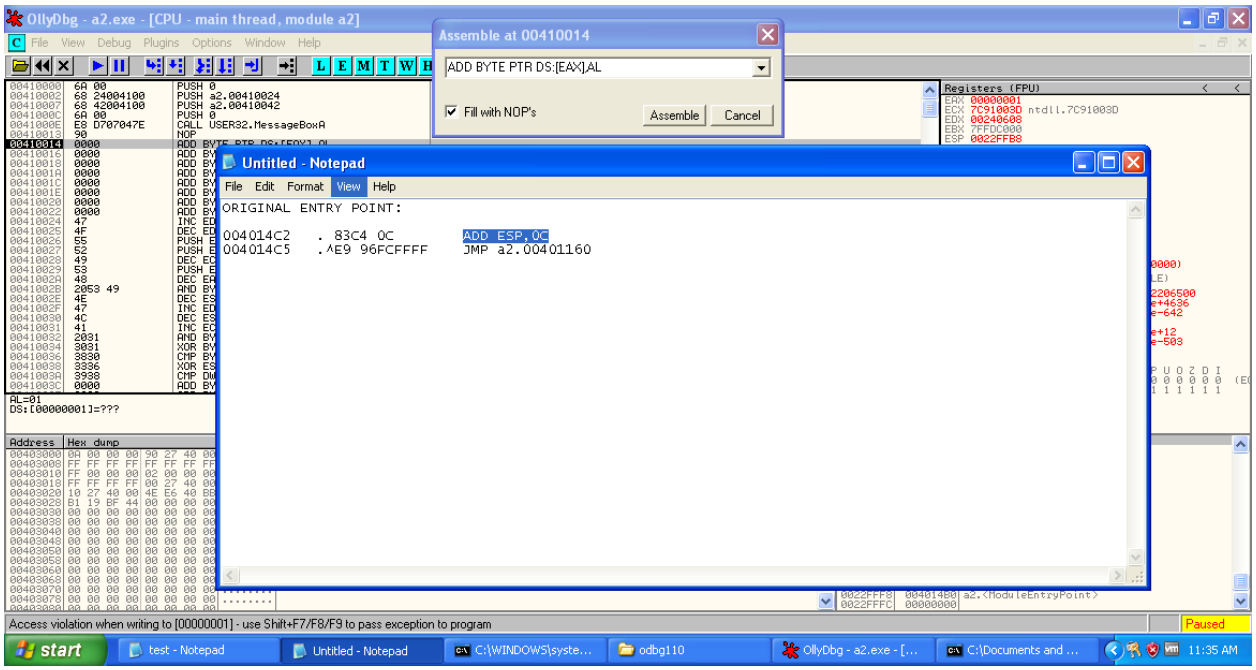
Lets the following instructions of the starting of the a2.exe as these are assembly to which control has to be returned.

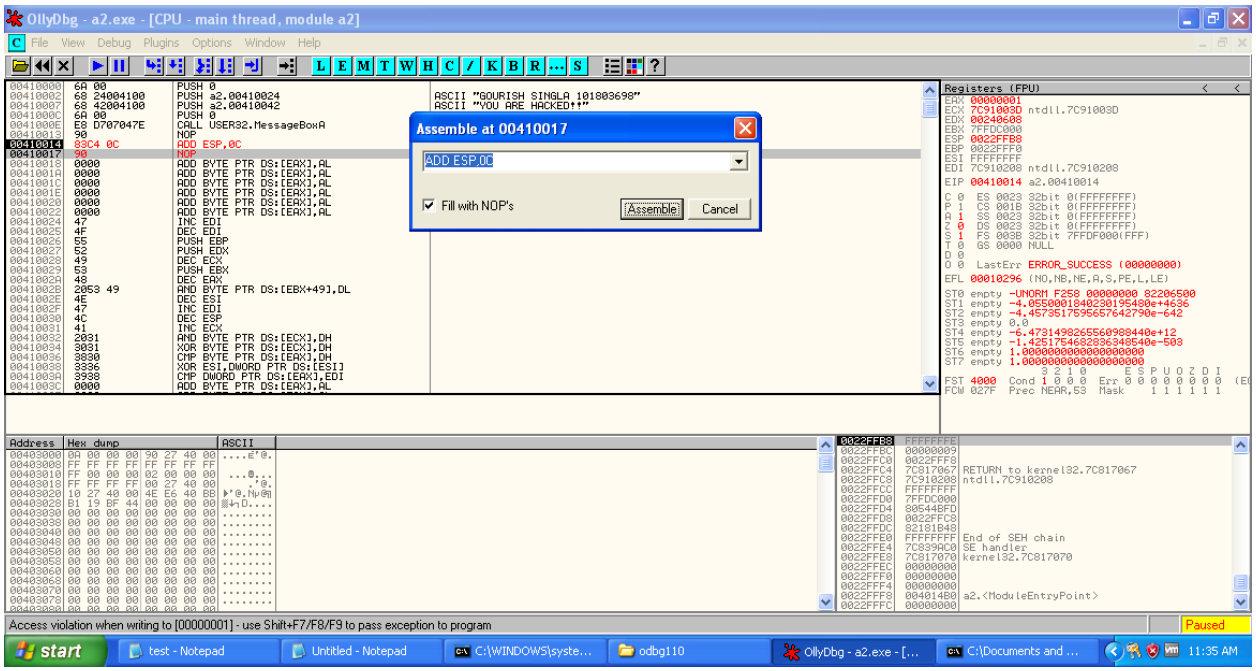


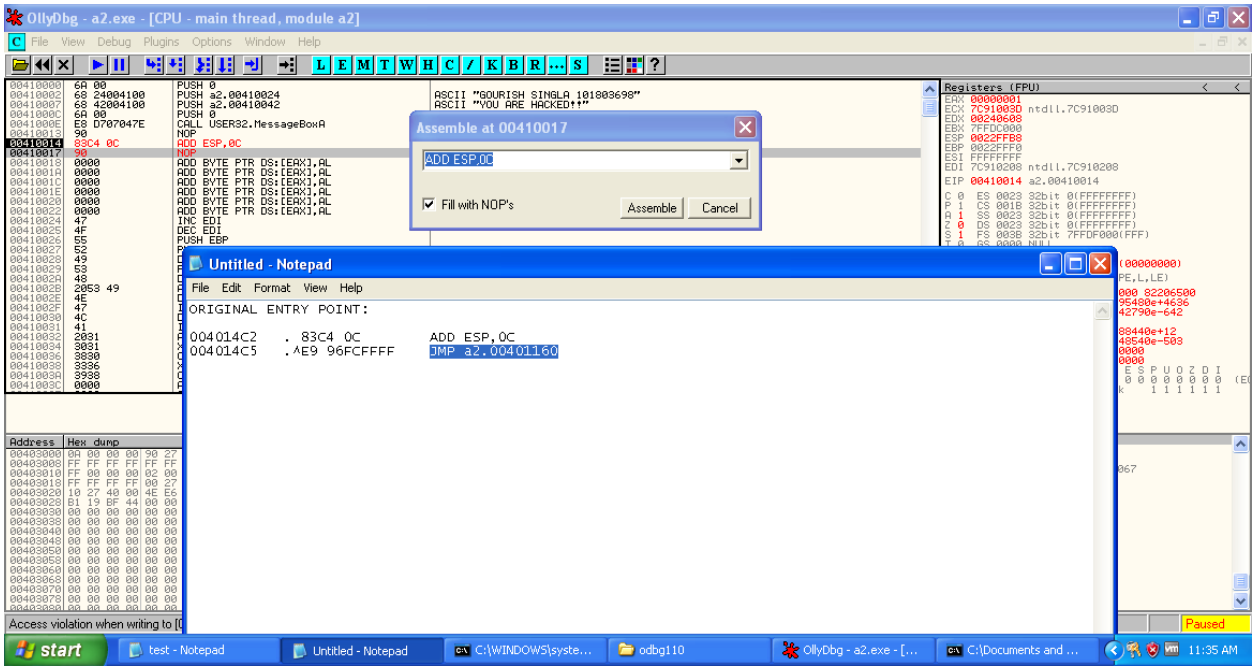
Lets save it in notepad for reference.

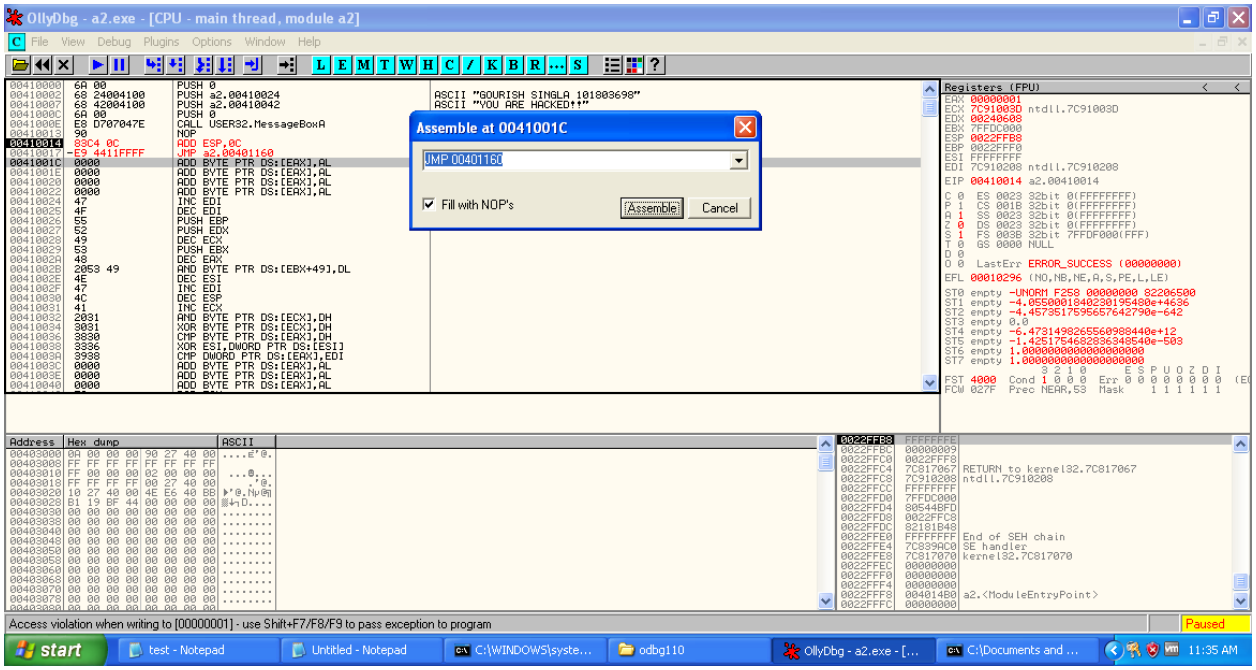


Add the assembly at location just after the calling function of messagebox.

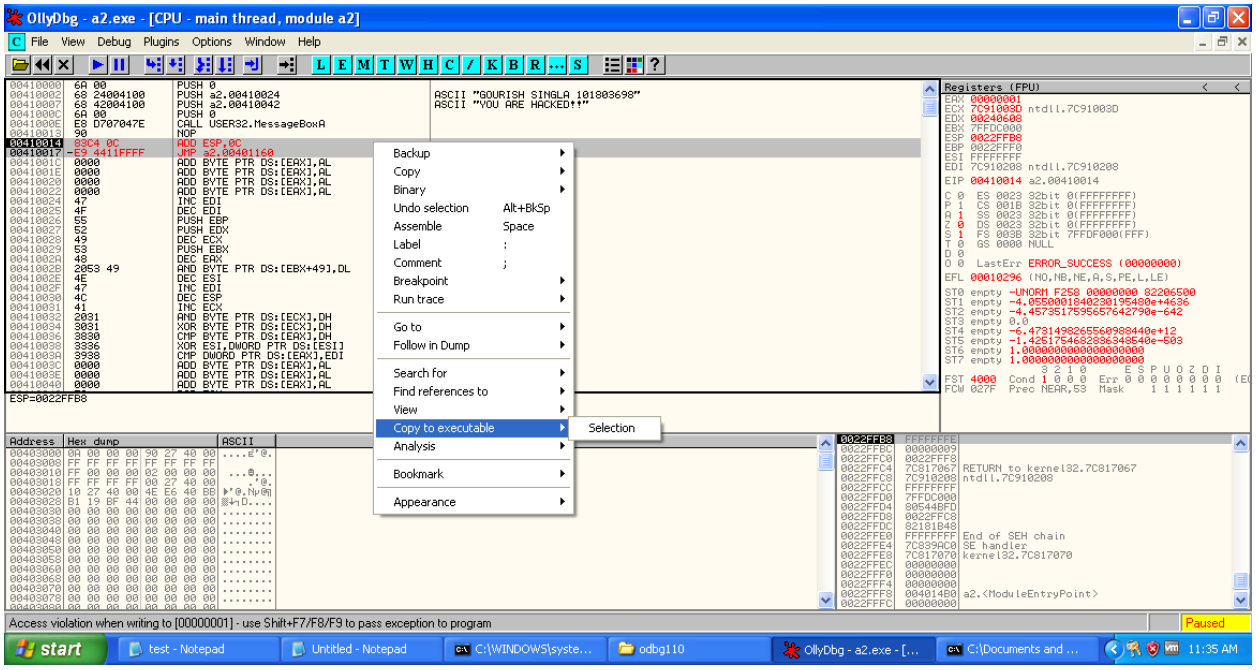


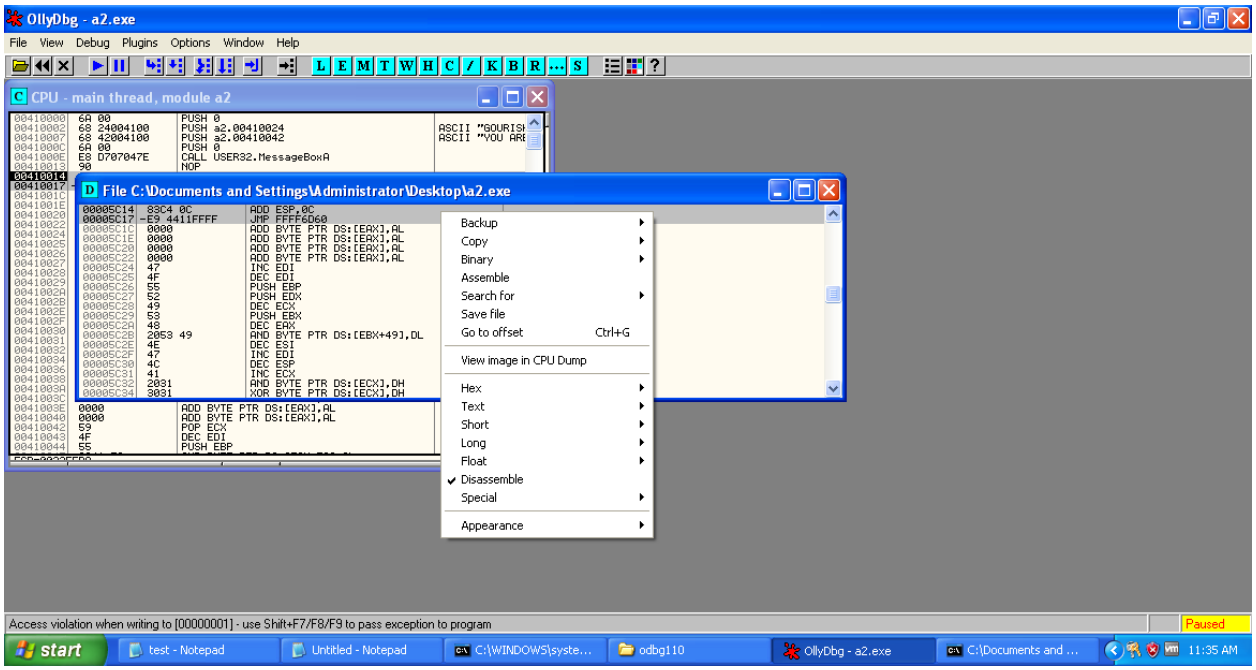


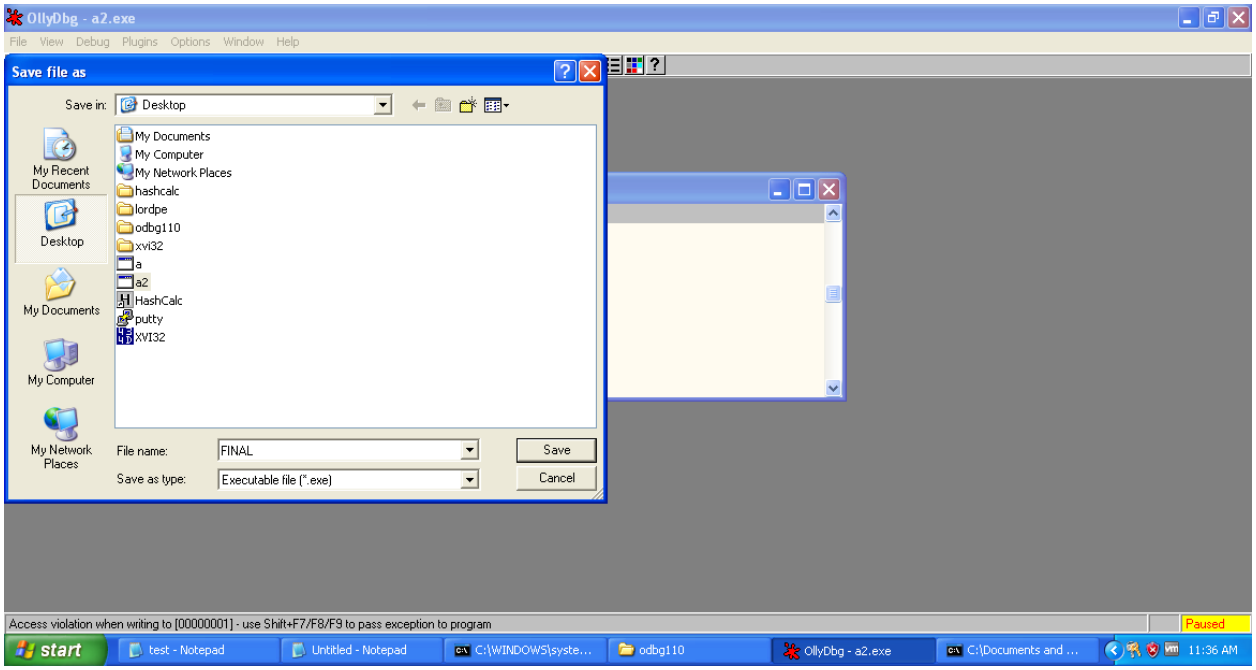

OllyDbg - a2.exe - [CPU - main thread, module a2]
Assemble at 00410017
ADD ESP,0C
Fill with NOP's
Assemble
Cancel
Untitled - Notepad
File Edit Format View Help
ORIGINAL ENTRY POINT:
004014C2 . 83C4 0C ADD ESP,0C
004014C5 .AE9 96FCFFFF JMP a2.00401160
Access violation when writing to [
Paused
start
test - Notepad
Untitled - Notepad
C:\WINDOWS\syste...
odbg110
OllyDbg - a2.exe - [...
C:\Documents and ...
11:35 AM


OllyDbg - a2.exe - [CPU - main thread, module a2]
Assemble at 0041001C
JMP 00401160
Fill with NOP's
Assemble
Cancel
Access violation when writing to [00000001] - use Shift+F7/F8/F9 to pass exception to program
Paused
start
test - Notepad
Untitled - Notepad
C:\WINDOWS\syste...
odbg110
OllyDbg - a2.exe - [...
C:\Documents and ...
11:35 AM

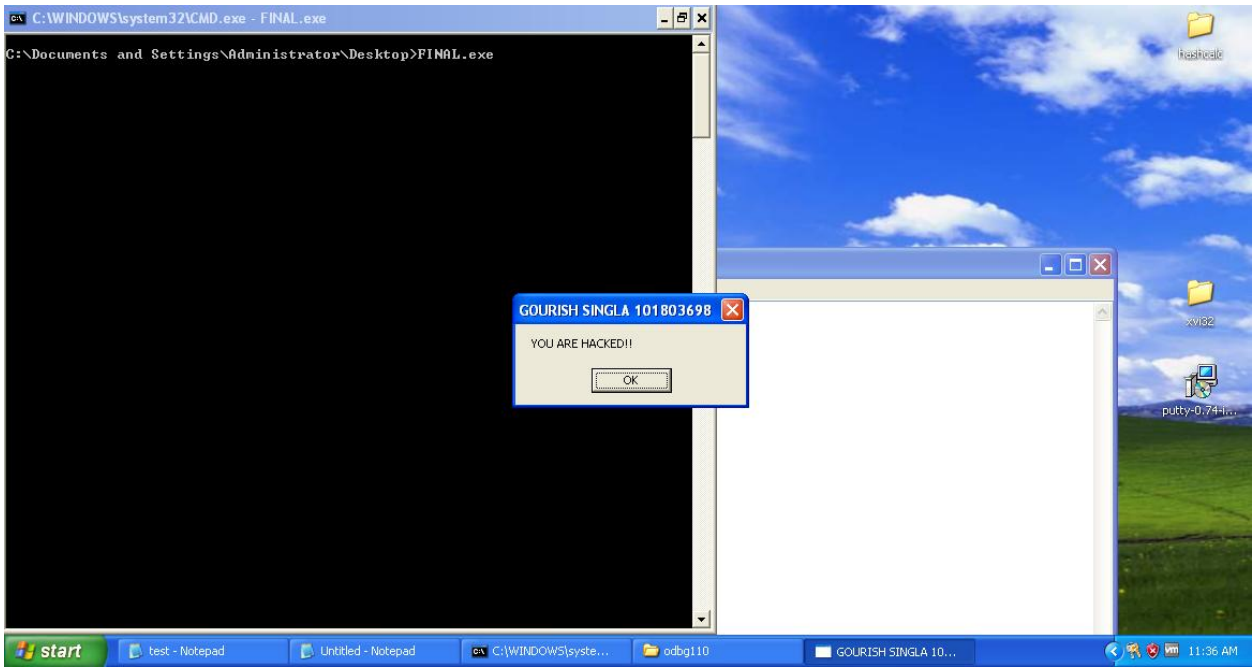
After adding both instructions, select them and copy and save to the executable to permanent the change in FINAL.exe.







Now when we run the FINAL.exe from cmd it first displays the modified Trojan messagebox.



When we click OK it shown the message box originally associated with the test.c or a.exe