SEH Based Exploitation

By Umair Manzoor (UmZ).

About Me

- Software Engineer at NADRA.
- Involved in Network & Information security since childhood.
- Published few exploits. (against Microsoft too) & paper at NEbraskaCERT USA.
- Contact & List of publications.

umz32.dll [at] gmail.com

http://www.milw0rm.com/author/701

Agenda

- Brief overview of buffer overflow in applications.
- SEH basics.
- SEH based exploitation.

What is BOF????!

- An error caused when a program tries to store too much data in a temporary storage area. This can be exploit by hackers to execute malicious code.

  -- smoothwall

- In computer security and programming, a buffer overflow or buffer overrun is a programming error which may result in a memory access exception and program termination or in the event of user being malicious, a breach of system security.

  -- wikipedia

SEH Prerequisites

- Should have good understanding of stack based exploitation.
- Use of debuggers like Olly.
- Assembly language.
- Concept of shellcodes.

What is SEH?

- Structured Exception Handling.
- What is an Exception?
  - An unexpected situation caused within the software.
  - What if it was really unexpected by the developer too??
  - OS takes over the program and handles it by itself.

Types of Exceptions

- Hardware
  - Raised by the OS in response to processor or virtual machine events (e.g. divide by zero or invalid memory access).
- Software
  - Raised by you or the Windows API or a component you are using.

WHY SEH??

- Windows is a smart platform that prevents owning the IP REG. while overflowing the buffer specially in SP2.
- As a result even if you have placed a shellcode successfully you won't be able to execute it.

WHY SEH (cont.)

- Any occurrence of access violation hands over the charge to SEH.

- SEH then tries to clean up the exception either by invoking some more exceptions in chain or by terminating the program cleanly.

Anatomy of SEH……

- Disassembly of a program reveals that
  - There are 3 important pointers.
    - Pointer to next SEH.
    - SE Handler.
    - End of SEH Chain.

Anatomy (cont.)

- Pointer to next SEH record.
  - The address to which the CPU transfers the control when the current SE handler is unable to handle the exception. (resulting in invocation of next SEH)
- SE Handler.
  - The address of piece of code that handles or tries to handle the exception.
- End of SEH chain.
  - If the chain is unable to handle the exception it results in an improper termination of the program.

Anatomy (cont.)

- **Stack dump**
- First the Pointer to next SEH record is placed on the stack.
- Then there is the address of SE handler.
- If exception is handled correctly then the control is transferred back to program.

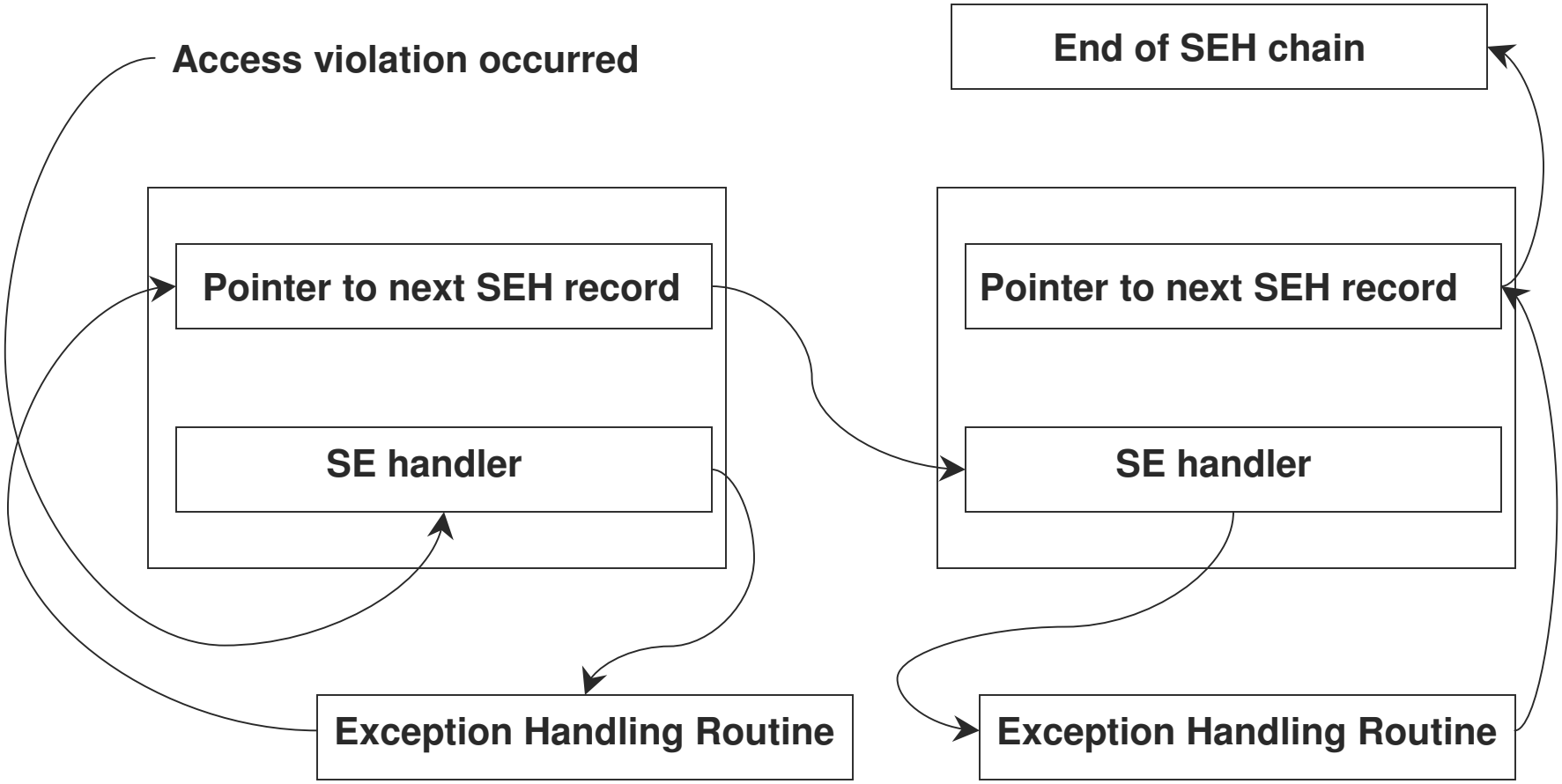
- 0012FFA4 |FFFFFFFF
- 0012FFA8 |0012FF4C
- 0012FFAC |0012FFF0
- 0012FFB0 |0012FFE0 Pointer to next SEH record
- 0012FFB4 |0040A1C4 SE handler
- 0012FFB8 |0040E3F8 PMSystem.0040E3F8
- 0012FFBC |00000000
- 0012FFC0 \0012FFF0

Anatomy (cont.)

- **Stack dump**
- Here **0xFFFFFFFF** represents the End of SEH.

- 0012FFC4 7C816D4F RETURN to kernel32.7C816D4F
- 0012FFC8 7C910738 ntdll.7C910738
- 0012FFCC FFFFFFFF
- 0012FFD0 7FFD6000
- 0012FFD4 8054B038
- 0012FFD8 0012FFC8
- 0012FFDC 821A17C0
- **0012FFE0 FFFFFFFF End of SEH chain**
- 0012FFE4 7C8399F3 SE handler
- 0012FFE8 7C816D58 kernel32.7C816D58

Anatomy of SEH



Taking over SEH!!!

- What advantage do we get by taking over SEH??? anyone?
  - You don't have to worry about the access violation exceptions when you have owned the SEH.
  - As a matter of fact we need the access violation.

Why access violation???

- If any thing bad happens the SE handler will execute.
- If we have already owned that SE handler then we can jump to some fake exception.
- Fake Exception (POP, POP, RET).
- Pointer to next SEH must point to shellcode.

POP, POP, RET???

- What is POP, POP, RET?? Anyone??
  - It's a magic routine to fool the OS.
  - OS understands that exception handling routine has been executed and now move to next SEH or End of SEH chain.
  - Fake exception should be in some binary dll / exe except the stack.

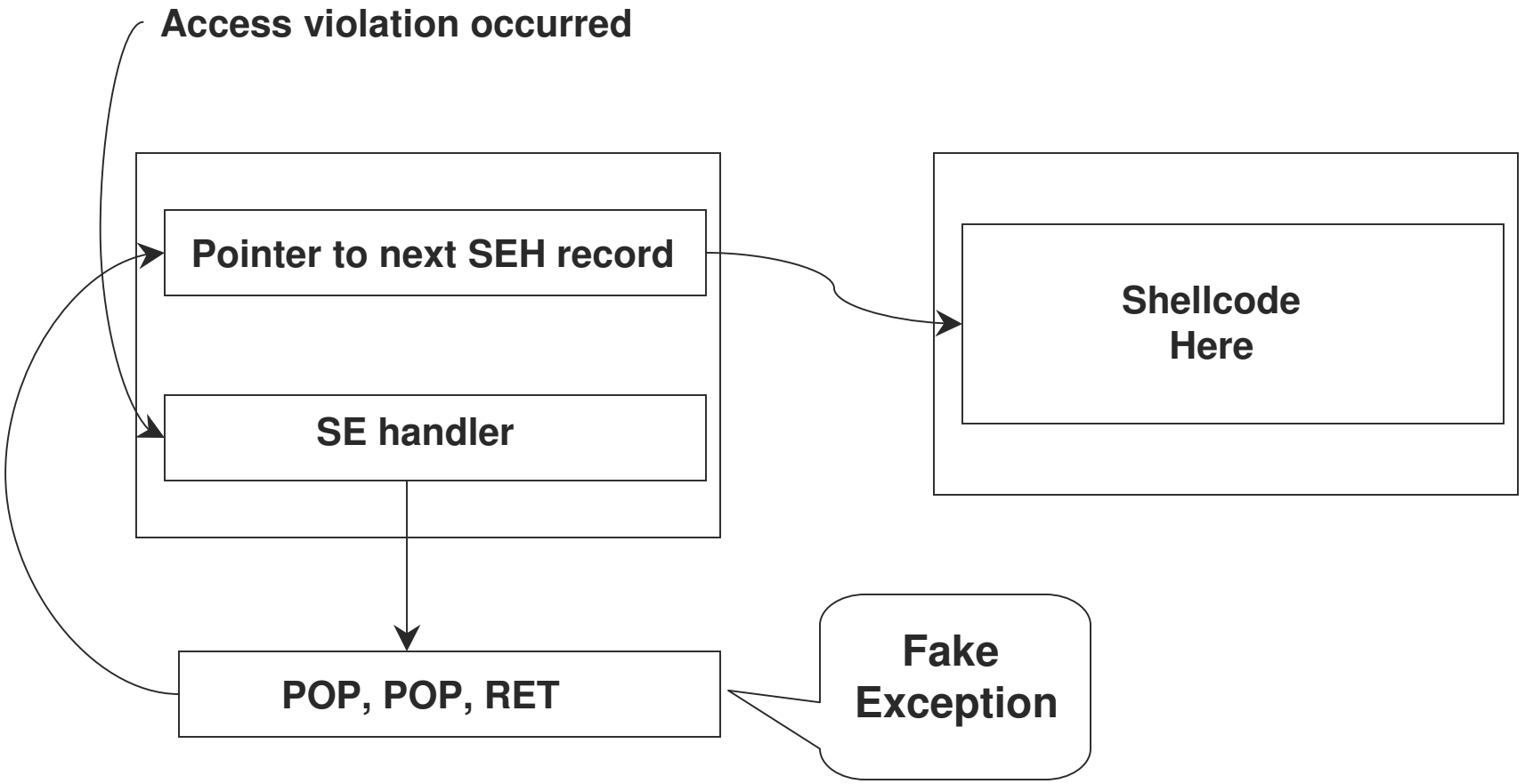
POP, POP, RET??? (cont.)

- Then where is pop pop ret???
  - Windows has it for us in ntdll.dll
  - NTDLL is "**NT LAYER DLL**" that contains NT Kernel functions.
  - We can dump the entire memory area and search for magic routine.(use memdump).
  - Use msfpescan to find the pop pop ret sequence.

POP, POP, RET??? (cont.)

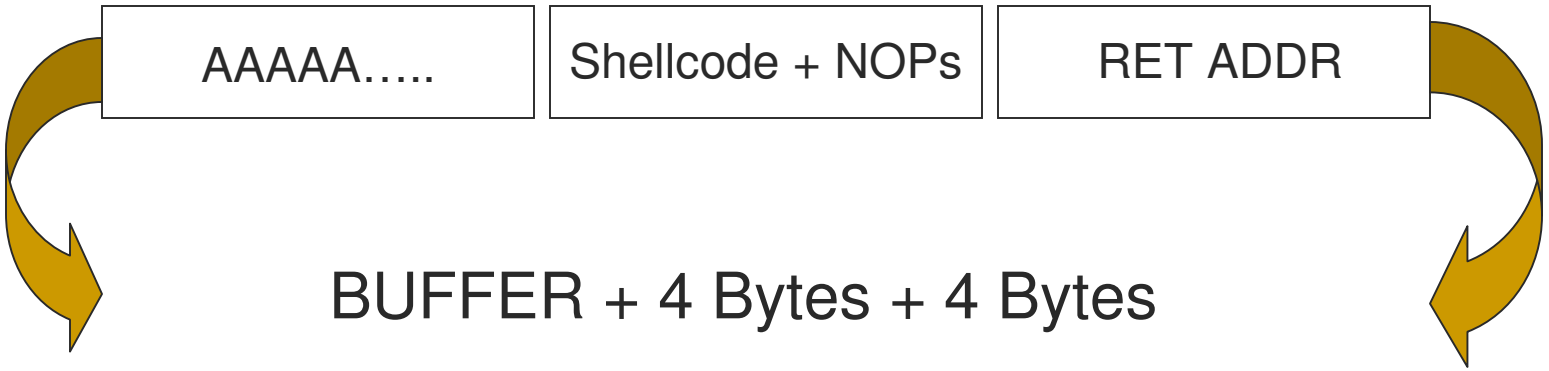
- But…….
  - Searching an unmapped memory area will cause an access violation that might kill the application.

Owned SEH!!!!


Access violation occurred
Pointer to next SEH record
SE handler
Shellcode
Here
POP, POP, RET
Fake
Exception

Using SEH for code execution

- Successfully exploitation depends upon the well structured payload.
- Payloads are like…….



Payloads

- Many approaches for a payload.
- An easy approach for a good payload is usage of NOPs.
- In such a way you can avoid absolute addressing (return address).

eXample

- Nops are very usefull here.
- CPU process NOPs as No Operation command.
- Hence it jumps to the next statement.
- This process continues until it get a operational statement.

- 010EFC14 90909090
- 010EFC18 90909090
- 010EFC1C 90909090
- 010EFC20 90909009
- 010EFC24 90909090
- 010EFC28 90909090
- 010EFC2C 0128FC90 Pointer to next SEH record
- 010EFC30 7C901010 SE handler
- 010EFC34 90909090
- 010EFC38 90909090
- 010EFC3C 90909090
- 010EFC40 90909090
- 010EFC44 90909090
- 010EFC48 90909090
- 010EFC4C 90909090

eXample (cont.)

- After NOPs the shellcode starts.
- So we don't need to specify the exact address of **shellcode** in pointer to next SEH record.
- For example rather then providing **0x010EFCA0,** you can provide **010EFC90-9C**
- Wooops!! We avoided the absolute addressing.

- **010EFC90 90909090**
- **010EFC94 90909090**
- **010EFC98 90909090**
- **010EFC9C 90909090**
- **010EFCA0 E983C931**
- **010EFCA4 D9EED9DB**
- **010EFCA8 5BF42474**
- **010EFCAC D8137381**
- **010EFCB0 83E47222**
- **010EFCB4 F4E2FCEB**
- **010EFCB8 E434CA24**
- **010EFCBC A1F922D8**
- **010EFCC0 E10EA9E4**
- **010EFCC4 6F9D23A0**
- **010EFCC8 BBF93A97**

Things to remember.

- Be careful while making payloads.
- Reduce the no. of NOPs if required to adjust the size of shellcode.
- Your shellcode may require encoding schemes such as Alpha2 etc.

If you spend more on coffee than on IT security, you will be hacked. What's more, you deserve to be hacked.

— White House Cyber security Advisor, Richard Clarke

All Done.

- Thanks.
- ----------X----------X----------

- Questions ?