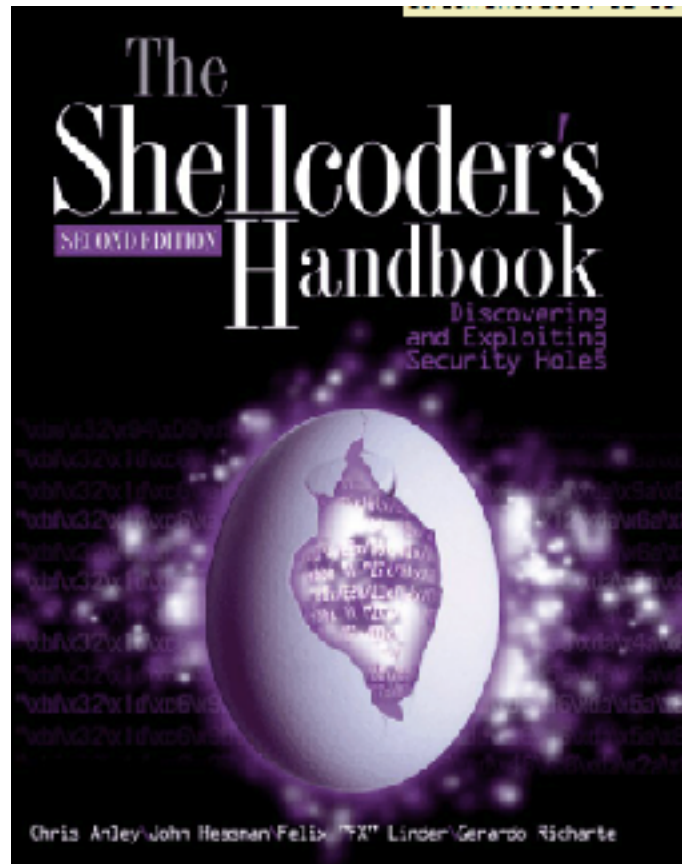


# CNIT 127: Exploit Development

## Ch 8: Windows Overflows

### Part 1



Rev. 3-15-17

# Topics

- Thread Environment Block and Process Environment Block
- Stack-Based Buffer Overflows
- Defeating ASLR (Address Space Layout Randomization)
- Frame-Based Exception Handlers
- SEH Protections
- Defenses in Server 2008 (not in textbook)

# Thread Environment Block and Process Environment Block

# Thread Environment Block (TEB)

- Stores information about the currently executing thread
- Also called the Thread Information Block
- Officially undocumented for Win 9x
- Partially documented for Win NT
- So many Win32 programs use undocumented TEB fields that they are effectively part of the API

# Thread Environment Block (TEB)

- TEB can be used to get a lot of information about a process without calling the Win32 API
- Can get
  - Last error
  - Import Address Table
  - Process startup arguments
  - and more

# Thread Environment Block (TEB)

- SEH pointer
- Info about the stack
  - Much more (Link Ch 8a)

## Contents of the TIB (32-bit Windows) [\[ edit \]](#)

Position	Length	Windows Versions	Description
FS:[0x00]	4	Win9x and NT	Current <a href="#">Structured Exception Handling (SEH)</a> frame
FS:[0x04]	4	Win9x and NT	Stack Base / Bottom of stack (high address)
FS:[0x08]	4	Win9x and NT	Stack Limit / Ceiling of stack (low address)

# FS: and GS:

- Segment Registers
- Left over from very early operating systems
- Not used much anymore for their original purpose
- On Windows 32-bit x86, FS: is used to point to the TEB
  - Links Ch 8e, 8y

# Process Environment Block (PEB)

- An opaque data structure
  - Details are hidden from its users
  - Values are only intended to be manipulated by calling subroutines that can access the hidden information
- Used by Win NT internally
- Most fields intended only for internal OS use



# Process Environment Block (PEB)

- Only a few fields are documented by Microsoft
- Contains data structures that apply across a whole process

# Process Environment Block (PEB)

Fields of the PEB that are documented by Microsoft<sup>[2]</sup>

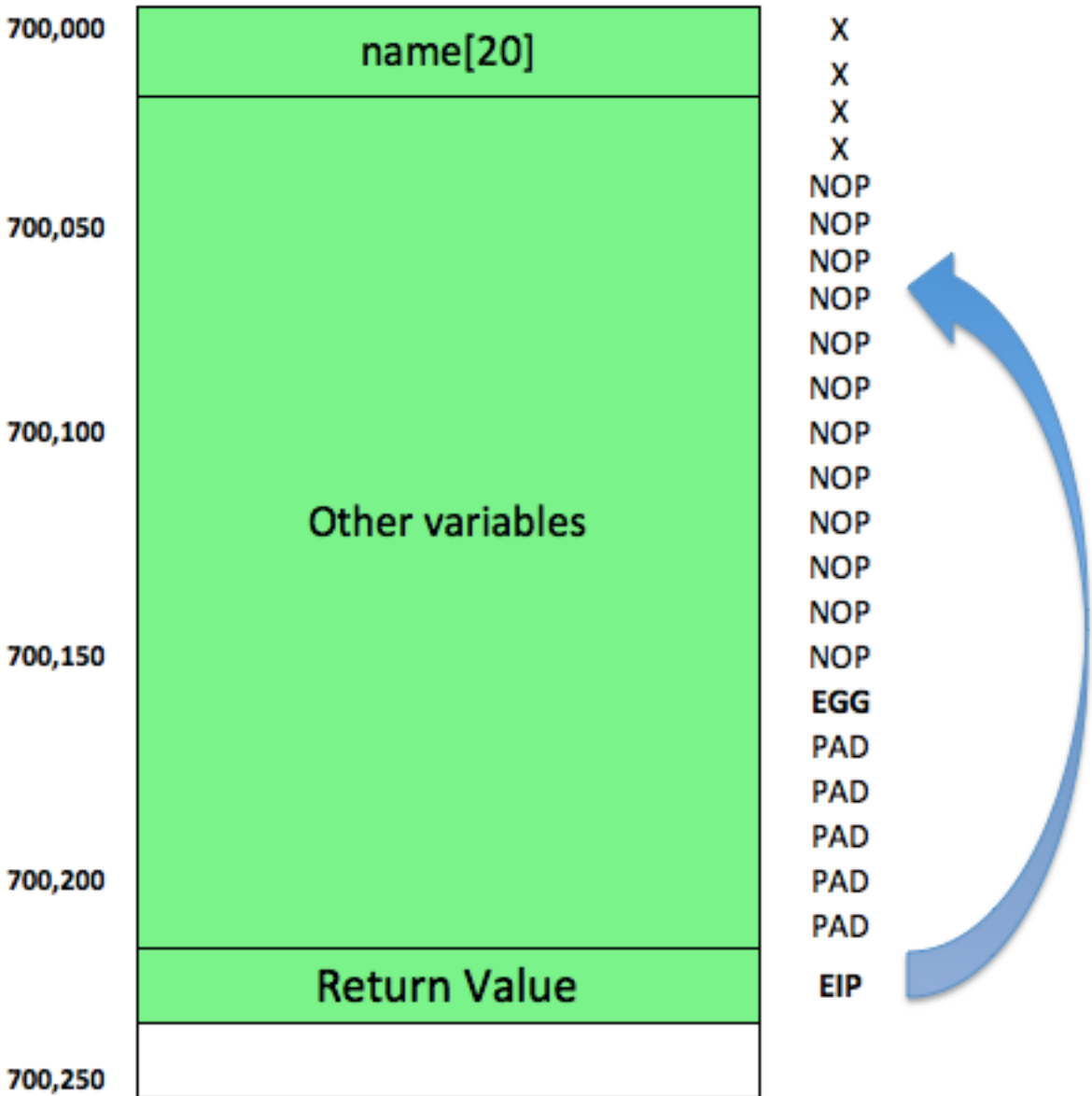
Field	meaning	notes
BeingDebugged	Whether the process is being debugged	Microsoft recommends not using this field but using the official Win32 <code>CheckRemoteDebuggerPresent()</code> library function instead. <sup>[2]</sup>
Ldr	A pointer to a <code>PEB_LDR_DATA</code> structure providing information about loaded modules	Contains the base address of <code>kernel32</code> and <code>ntdll</code> .
ProcessParameters	A pointer to a <code>RTL_USER_PROCESS_PARAMETERS</code> structure providing information about process startup parameters	The <code>RTL_USER_PROCESS_PARAMETERS</code> structure is also mostly opaque and not guaranteed to be consistent across multiple versions of Windows. <sup>[4]</sup>
PostProcessInitRoutine	A pointer to a callback function called after DLL initialization but before the main executable code is invoked	This callback function is used on <code>Windows 2000</code> , but is not guaranteed to be used on later versions of Windows NT. <sup>[2]</sup>
SessionId	The session ID of the Terminal Services session that the process is part of	The <code>NtCreateUserProcess()</code> system call initializes this by calling the kernel's internal <code>MmGetSessionId()</code> function. <sup>[3]</sup>

# Stack-Based Buffer Overflows

# Classic Technique

- Overwrite the saved return address
  - With a value that points back into the stack
- When the function returns, it copies the return address into EIP
- Return address points to NOP Sled

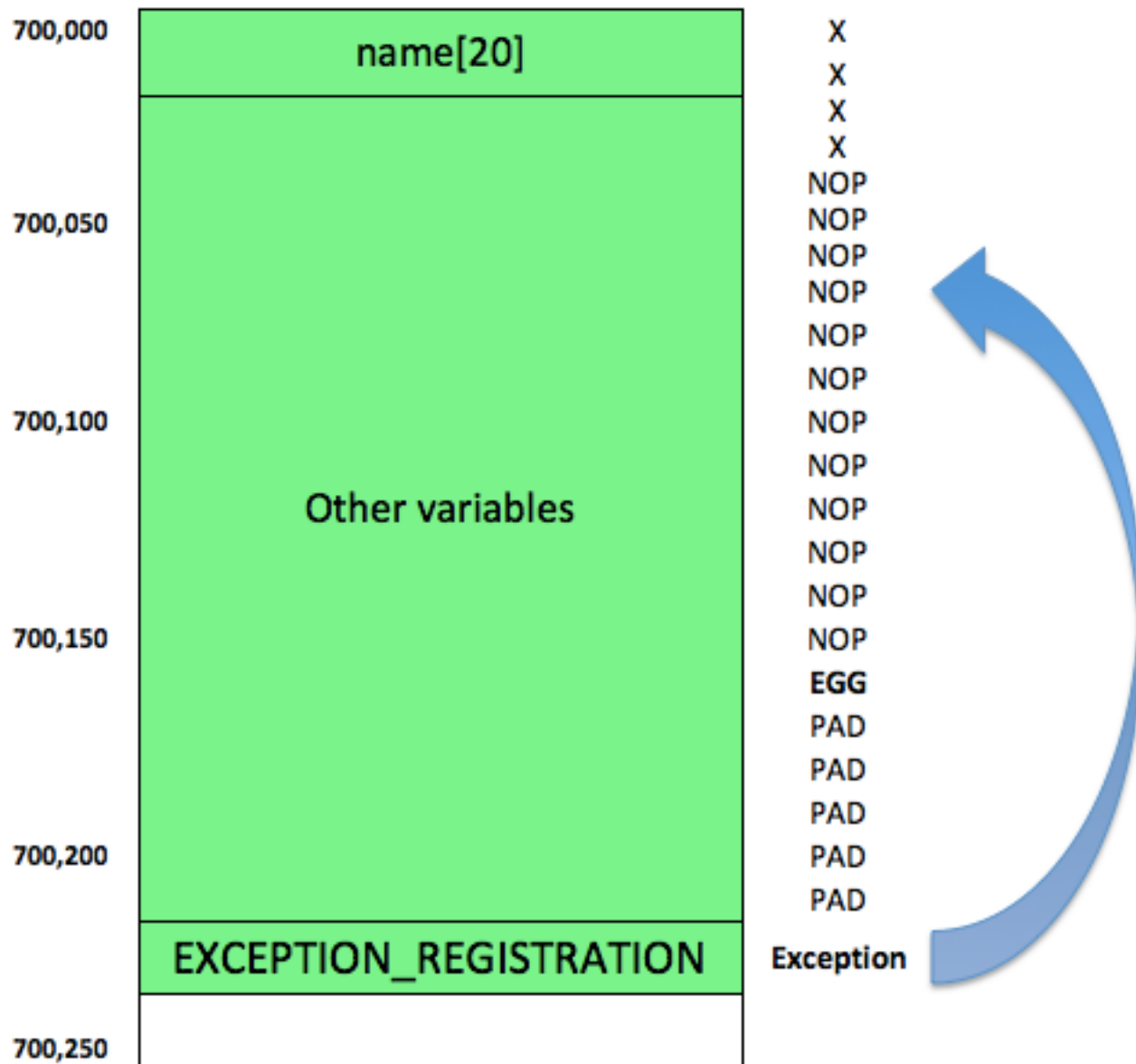
# Buffer Overflow Using Return Value



# SEH Technique

- Overwrite the SEH
  - With a value that points back into the stack
- Trigger an exception
- Modified exception handler points to NOP Sled

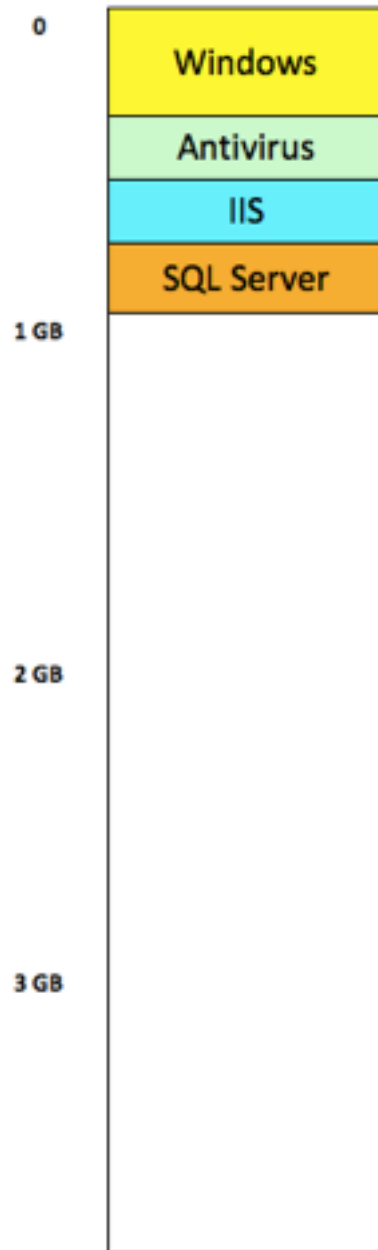
# Buffer Overflow Using SEH



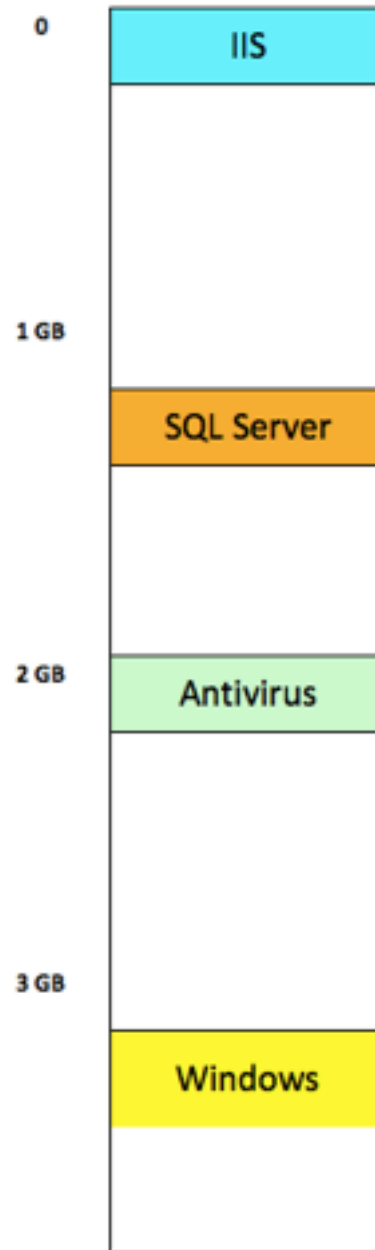
# Defeating ASLR (Address Space Layout Randomization)



### No ASLR



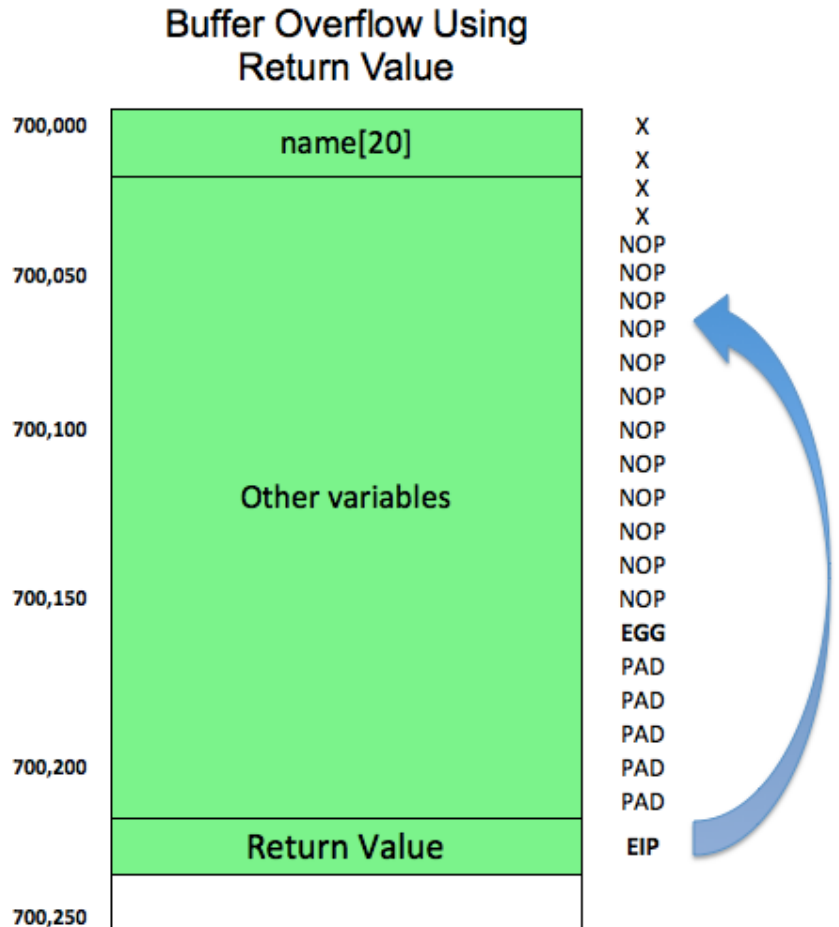
### With ASLR



RAM

# Using Return Value

- We're inside the function
- ESP points to the stack
- Find a JMP ESP and insert its address into the return value





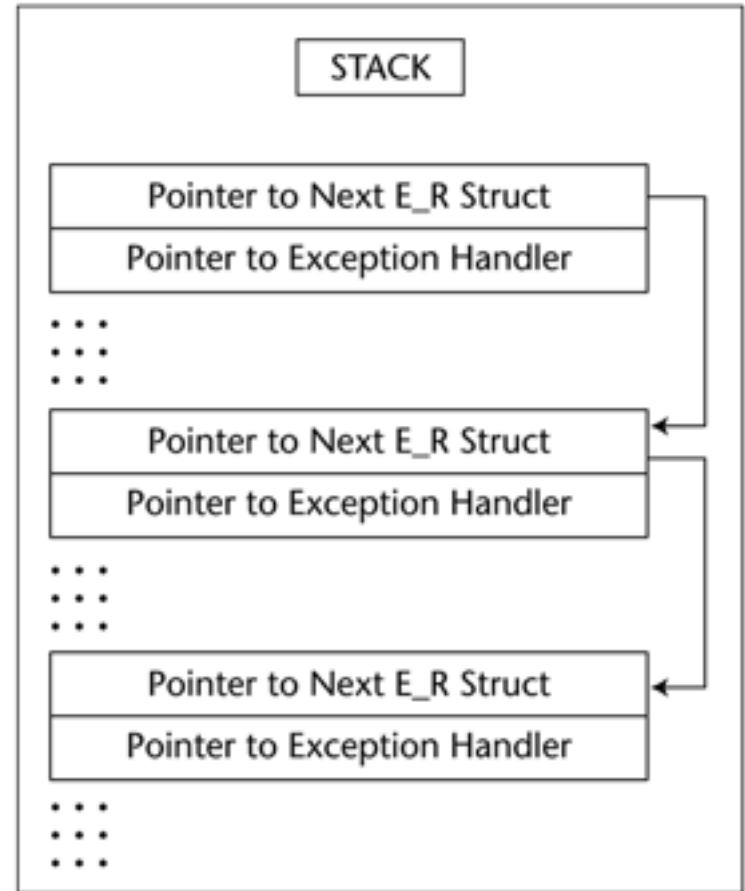
# Frame-Based Exception Handlers

# Exception Handler

- Code that deals with problems from within a running process
  - Such as access violation or divide by zero
- Frame-Based Exception Handler
  - Associated with a particular procedure
  - Each procedure sets up a new stack frame
- Every thread in a Win32 process has at least one frame-based exception handler

# EXCEPTION\_REGISTRATION

- Each thread's TEB has the address of the first EXCEPTION\_REGISTRATION structure at fs:[0]
- When an exception occurs, the OS walks through the list until a suitable handler is found



# SEH Example Code

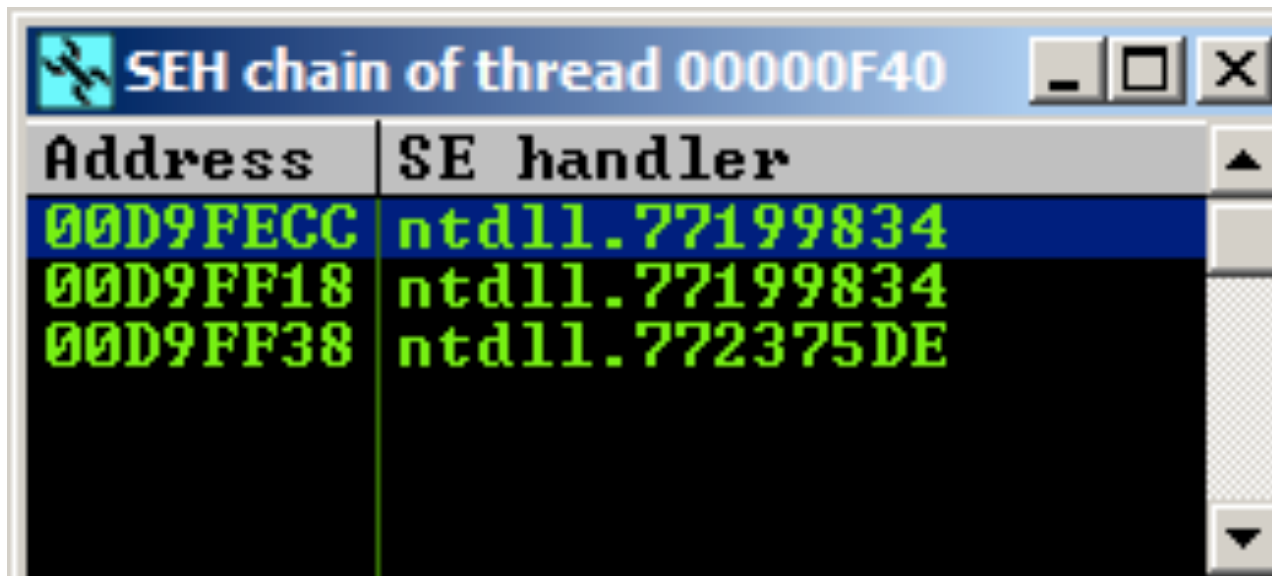
```
#include <stdio.h>
#include <windows.h>

DWORD MyExceptionHandler(void)
{
    printf("In exception handler....");
    ExitProcess(1);
    return 0;
}

int main()
{
    __try
    {
        __asm
        {
            // Cause an exception
            xor eax,eax
            call eax
        }
    }
    __except(MyExceptionHandler())
    {
        printf("oops...");
    }
    return 0;
}
```

# SEH Chain in Immunity

- View, SEH Chain
- This is Notepad's SEH Chain



Address	SE handler
00D9FECC	ntd11.77199834
00D9FF18	ntd11.77199834
00D9FF38	ntd11.772375DE



# Follow Address in Stack

00D9FECC	00D9FF18	↑ J .	Pointer to next SEH record
00D9FED0	77199834	4j↓w	SE handler
00D9FED4	003AD1CC	T: .	
00D9FED8	00000000	... .	
00D9FEDC	00D9FEE8	8j↓ .	
00D9FEE0	76294911	←I>v	RETURN to kernel32.76294911
00D9FEE4	00000000	... .	
00D9FEE8	00D9FF28	< J .	
00D9FEEC	771CE4B6	Σ←w	RETURN to ntdll.771CE4B6
00D9FEF0	00000000	... .	
00D9FEF4	77FC1BFC	n←nw	
00D9FEF8	00000000	... .	
00D9FEFC	00000000	... .	
00D9FF00	00000000	... .	
00D9FF04	00000000	... .	
00D9FF08	00000000	... .	
00D9FF0C	00000000	... .	
00D9FF10	00D9FEF4	f ↓ .	
00D9FF14	00000000	... .	
00D9FF18	00D9FF38	8 J .	Pointer to next SEH record
00D9FF1C	77199834	4j↓w	SE handler
00D9FF20	003ADB2C	, : .	
00D9FF24	00000000	... .	
00D9FF28	00D9FF40	@ J .	
00D9FF2C	771CE489	ëΣ←w	RETURN to ntdll.771CE489 from ntdll.771C
00D9FF30	7721D094	öM?w	ntdll.DbgUiRemoteBreakin
00D9FF34	00000000	... .	
00D9FF38	FFFFFFFF		End of SEH chain
00D9FF3C	772375DE	t#w	SE handler
00D9FF40	00000000	... .	
00D9FF44	00000000	... .	
00D9FF48	7721D094	öM?w	ntdll.DbgUiRemoteBreakin
00D9FF4C	00000000	... .	

# Exceptions in Stack Overflows

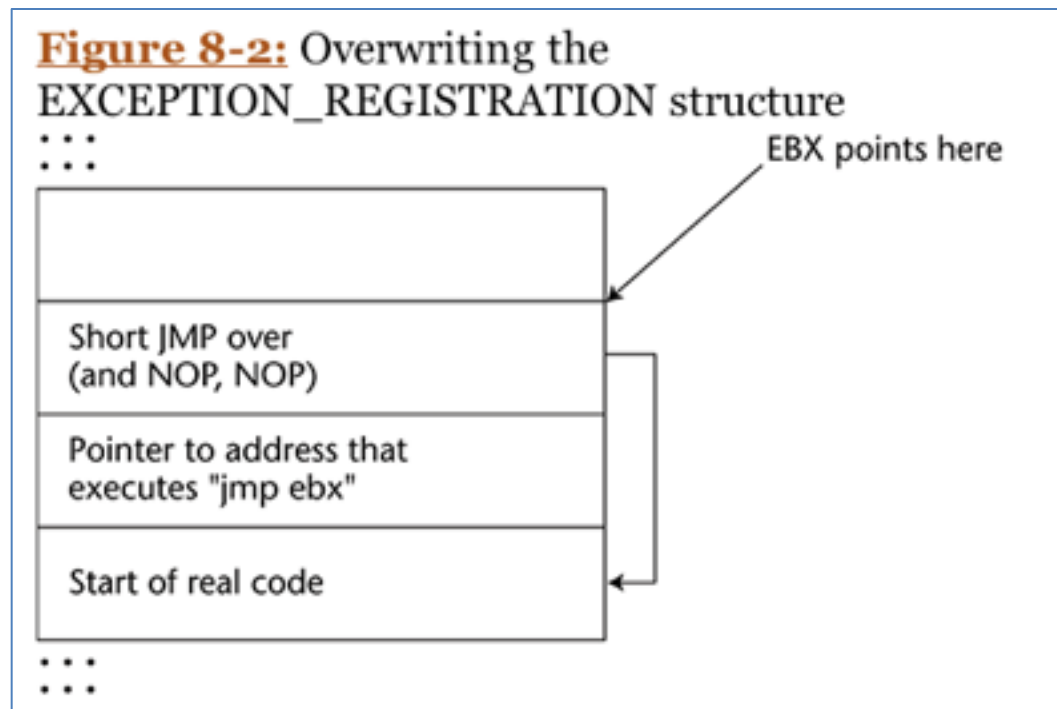
- Suppose we overflow the stack to overwrite the stored **return address**
- Unfortunately, we also overwrite other variables by accident (collateral damage)
- Those may lead to exceptions, as other instructions become invalid, before the function returns

# Overwriting EXCEPTION\_REGISTRATION

- To gain control of exception handling
- On Windows 2000 and Win XP (before SP1)
  - EBX points to the current EXCEPTION\_REGISTRATION structure
- On later Windows versions, all the registers are zeroed when the SEH is called
  - In order to make exploitation more difficult

# Old Windows Process

- At crash, EBX points to EXCEPTION\_REGISTRATION structure
- Overwrite With "JMP EBX"

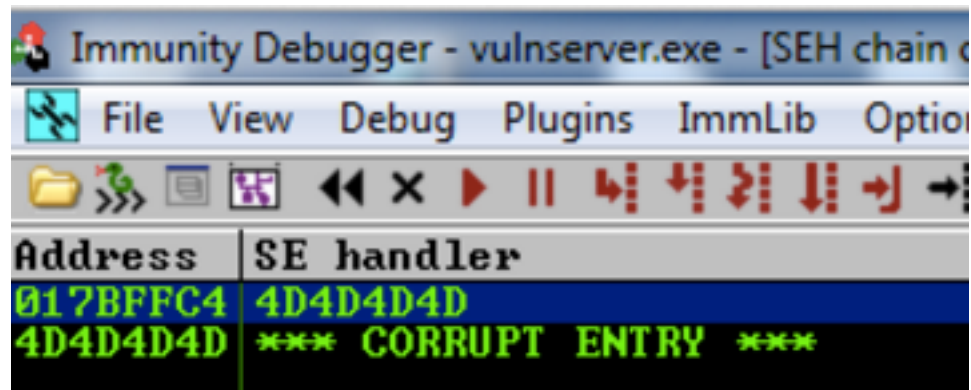


# Modern Windows Process

- Stack at crash

```
017CEC98 77BC71F9 -qJw RETURN to ntdll.77BC71F9
017CEC9C 017CED80 C∞!@
017CECA0 017CFFC4 - !@
017CECA4 017CED9C £∞!@
```

- 3<sup>rd</sup> value points to EXCEPTION\_REGISTRATION structure
- Use POP, POP, RET



# SEH Protections

# Win 2003 Server

- Attempts to ensure that handler is valid before using it, with these steps
  - 1.If handler is on the stack -- INVALID
    - According to the TEB's entries FS:[4] and FS:{8]
  - 2.If handler is in any loaded EXE or DLL -  
MAYBE
    - Otherwise VALID

# Win 2003 Server

3. If the module is marked as "not allowed - INVALID
4. If a module has no "Load Configuration Directory", or one with a small size: VALID



# Three Ways to Exploit SEH on Windows Server 2003

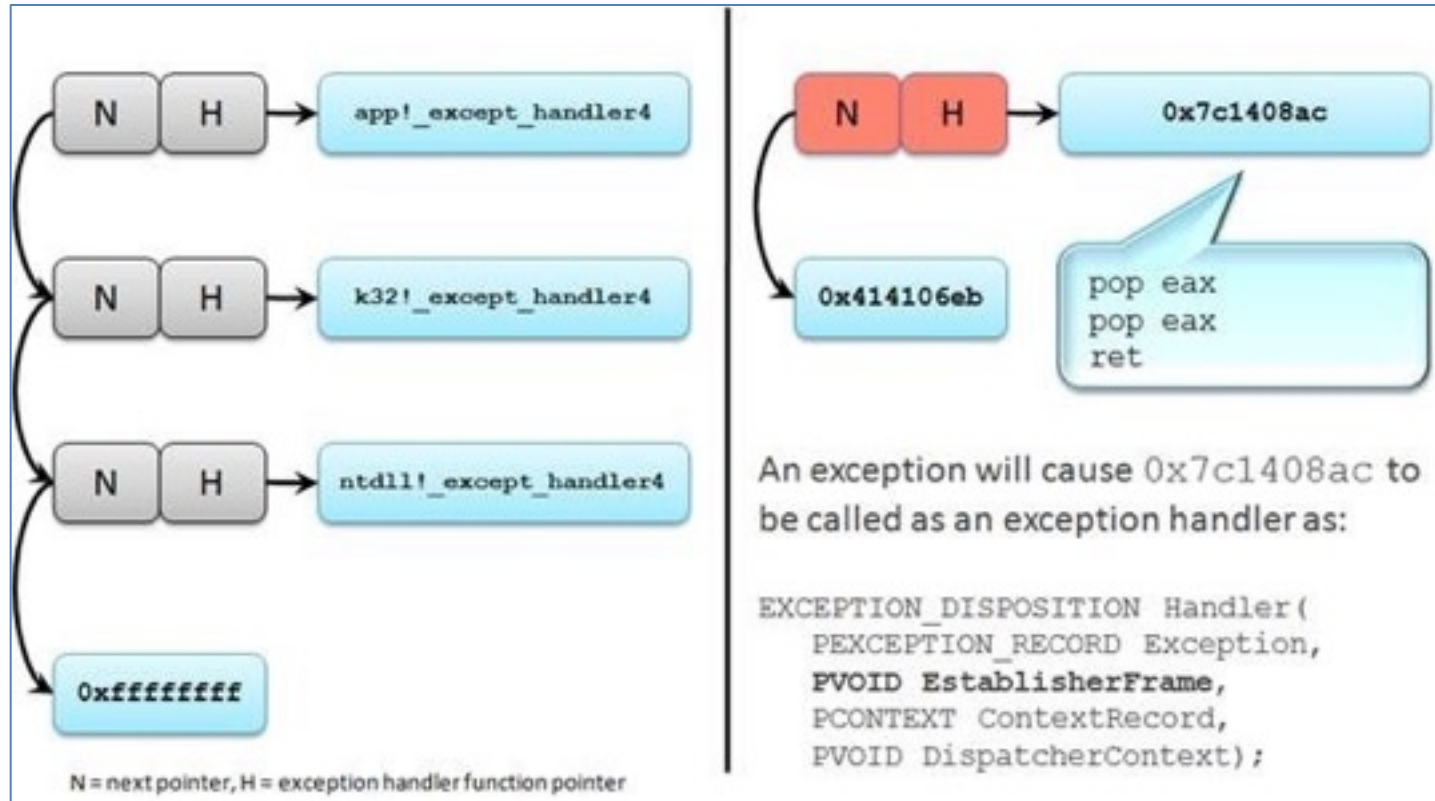
1. Abuse an existing handler
2. Use code in an address outside all modules
3. Use code in a module without a "Load Configuration Directory"

# Defenses in Win Server 2008

Not in Book

# Microsoft's Defenses

- Normal SEH attack (link Ch 8f)



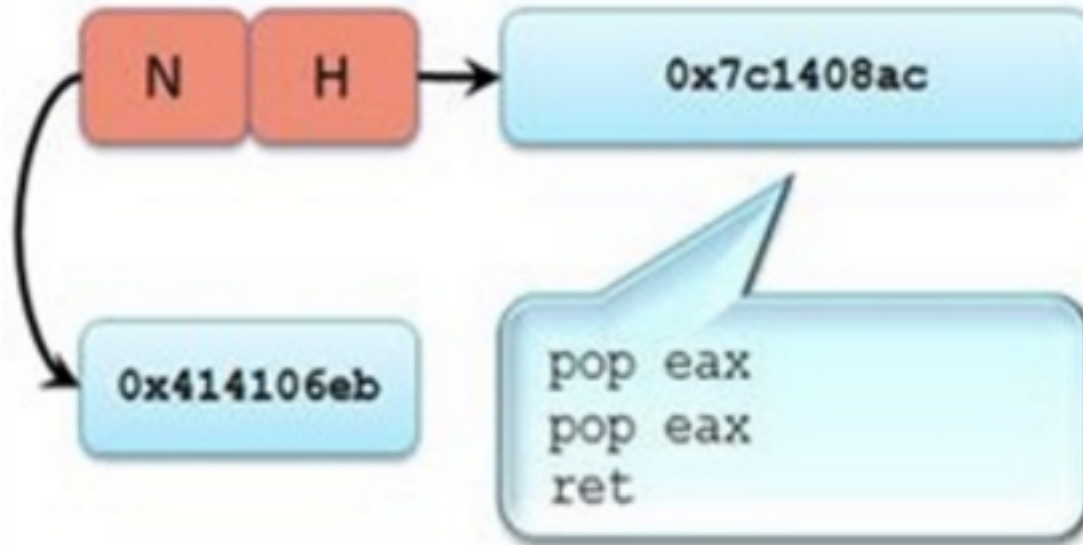
# SAFESEH

- Microsoft added this compiler switch to Visual Studio 2003
- It creates a whitelist of exception handler addresses
- BUT this depends on the developer using the switch
  - All legacy code must be recompiled

# SEHOP

- Verifies that the exception handler is intact before using it
- The pointer to the next handler comes before the pointer to the registration record
- So an exploit will usually damage the \*Next pointer

# EXCEPTION\_REGISTRATION\_RECORD

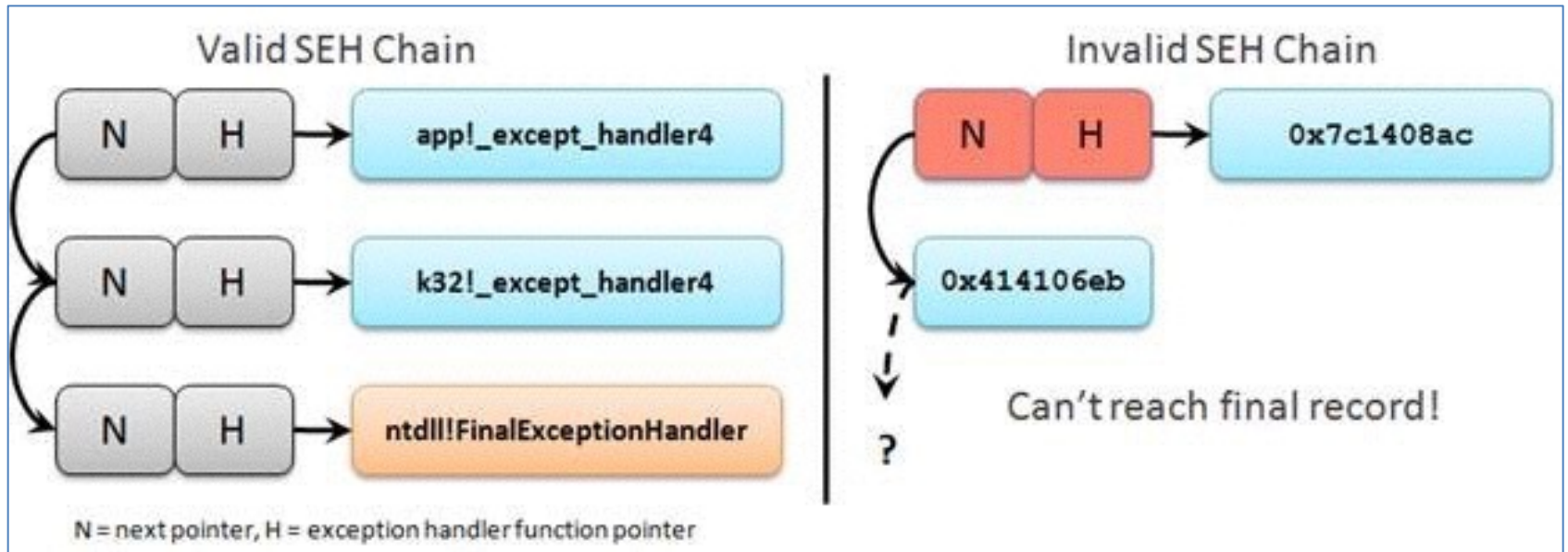


```
typedef struct _EXCEPTION_REGISTRATION_RECORD  
{  
    struct _EXCEPTION_REGISTRATION_RECORD *Next;  
    PEXCEPTION_ROUTINE                      Handler;  
} EXCEPTION_REGISTRATION_RECORD, *PEXCEPTION_REGISTRATION_RECORD;
```

# How SEHOP Works

- Adds an extra registration record at the end of the list
- Walks the exception handler list to ensure that the added record can be reached
- Will detect corruption of \*Next pointers

# SEHOP





# Windows Versions

- SEHOP is enabled by default in Windows Server 2008
- Disabled by default in Windows Vista
  - Because it breaks some legacy code
- Also disabled by default on Windows 7
  - But it can be enabled in the Registry
  - Link Ch 8k

Registry Editor

File Edit View Favorites Help

- GRE\_Initialize
- ICM
- Image File Execution Options
  - DIINXOptions
  - ExtExport.exe
  - ie4uinit.exe
  - IEInstal.exe
  - ielowutil.exe
  - ieUnatt.exe
  - iexplore.exe
  - msfeedssync.exe
  - mshta.exe
  - vulnserver.exe
- IniFileMapping
- InstalledFeatures
- KnownFunctionTableDlls
- KnownManagedDebuggingDlls
- LanguagePack
- MCI
- MCI Extensions
- MCB2
- MiniDumpAuxiliaryDlls

Name	Type	Data
(Default)	REG_SZ	(value not set)
DisableExceptionChainValidation	REG_DWORD	0x00000000

Computer\HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\vulnserver.exe

```
00403090 00 00 00 00 00 00 00 00 00 .....  
00403098 00 00 00 00 00 00 00 00 00 .....  
004030A0 00 00 00 00 00 00 00 00 00 .....  
004030A8 00 00 00 00 00 00 00 00 00 .....  
004030B0 00 00 00 00 00 00 00 00 00 .....  
004030B8 00 00 00 00 00 00 00 00 00 .....
```

Debugged program was unable to process exception