
# Attacks

*Trent Jaeger*
*Systems and Internet Infrastructure Security (SIIS) Lab*
*Computer Science and Engineering Department*
*Pennsylvania State University*

August 29, 2011

# Introduction

- Problem – Attacks on software and systems
- Classical attack – Buffer overflow
- Attack: (1) Change control and (2) Run code
- Other forms of attack
- Return-oriented attacks
- Stuxnet

# Our Goal

- In this course, we want to develop techniques to detect vulnerabilities and fix them automatically
- What's a vulnerability?
- How to fix them?



- *We will examine the first question today*

# Vulnerability

- How do you define computer ‘vulnerability’?

# Buffer Overflow

- First and most common way to take control of a process
- Attack code
  - Call the victim with inputs necessary to overflow buffer
  - Overwrites the return address on the stack
- Exploit
  - Jump to attacker chosen code
  - Run that code

# Determine what to attack

- Local variable that is a char buffer
  - Called buf

```
...
printf("BEFORE picture of stack\n");
for ( i=((unsigned) buf-8); i<((unsigned) ((char *)&ct)+8); i++ )
  printf("%p: 0x%x\n", (void *)i, *(unsigned char *) i);

/* run overflow */
for ( i=1; i<tmp; i++ ){
  printf("i = %d; tmp= %d; ct = %d; &tmp = %p\n", i, tmp, ct, (void *)&tmp);
   strcpy(p, inputs[i]);

  /* print stack after the fact */
  printf("AFTER iteration %d\n", i);
  for ( j=((unsigned) buf-8); j<((unsigned) ((char *)&ct)+8); j++ )
    printf("%p: 0x%x\n", (void *)j, *(unsigned char *) j);

   p += strlen(inputs[i]);
  if ( i+1 != tmp )
    *p++ = ' ';
 }
printf("buf = %s\n", buf);
printf("victim: %p\n", (void *)&victim);

return 0;
}
```

```
BEFORE picture of stack
0xbfa3b854: 0x3
0xbfa3b855: 0x0
0xbfa3b856: 0x0
0xbfa3b857: 0x0
0xbfa3b858: 0x3
0xbfa3b859: 0x0
0xbfa3b85a: 0x0
0xbfa3b85b: 0x0
0xbfa3b85c: 0x0
0xbfa3b85d: 0x0
0xbfa3b85e: 0x0
0xbfa3b85f: 0x0
0xbfa3b860: 0x0
0xbfa3b861: 0x0
0xbfa3b862: 0x0
0xbfa3b863: 0x0
0xbfa3b864: 0x0
0xbfa3b865: 0x0
0xbfa3b866: 0x0
0xbfa3b867: 0x0
0xbfa3b868: 0xa8
0xbfa3b869: 0xb8
0xbfa3b86a: 0xa3
0xbfa3b86b: 0xbf
0xbfa3b86c: 0x71
0xbfa3b86d: 0x84
0xbfa3b86e: 0x4
0xbfa3b86f: 0x8
0xbfa3b870: 0x3
0xbfa3b871: 0x0
0xbfa3b872: 0x0
0xbfa3b873: 0x0
```

buf

ebp

rtn addr

ct

# Configure Attack

- Configure following
  - Distance to return address from buffer
    - Where to write?
  - Location of start of attacker's code
    - Where to take control?
  - What to write on stack
    - How to invoke code (jump-to existing function)?
  - How to launch the attack
    - How to send the malicious buffer to the victim?

# Return Address

- x86 Architecture
  - Build 32-bit code for Linux environment
- Remember integers are represented in "little endian" format
- Take address 0x8048471
  - See trace at right

```
BEFORE picture of stack
0xbfa3b854: 0x3
0xbfa3b855: 0x0
0xbfa3b856: 0x0
0xbfa3b857: 0x0
0xbfa3b858: 0x3
0xbfa3b859: 0x0
0xbfa3b85a: 0x0
0xbfa3b85b: 0x0
0xbfa3b85c: 0x0
0xbfa3b85d: 0x0
0xbfa3b85e: 0x0
0xbfa3b85f: 0x0
0xbfa3b860: 0x0
0xbfa3b861: 0x0
0xbfa3b862: 0x0
0xbfa3b863: 0x0
0xbfa3b864: 0x0
0xbfa3b865: 0x0
0xbfa3b866: 0x0
0xbfa3b867: 0x0
0xbfa3b868: 0xa8
0xbfa3b869: 0xb8
0xbfa3b86a: 0xa3
0xbfa3b86b: 0xbf
0xbfa3b86c: 0x71
0xbfa3b86d: 0x84
0xbfa3b86e: 0x4
0xbfa3b86f: 0x8
0xbfa3b870: 0x3
0xbfa3b871: 0x0
0xbfa3b872: 0x0
0xbfa3b873: 0x0
```

buf

ebp

rtn addr

ct

# Find Return Address Offset

- Build and run victim
  - 'make victim'
  - './victim foo bar'
- Find buffer address
  - printed at start of victim output

```
In shell
i = 3; inputs = 0xbfa3b944
&main = 0x8048424
&shell = 0x8048648
&inputs[0] = 0xbfa3b944
&buf[0] = 0xbfa3b854
BEFORE picture of stack
```

- To start of return address
  - read from stack
  - 0xbfa3b86c
- How do we know its the rtn_addr?
  - Must be an address in caller (main)

```
BEFORE picture of stack
0xbfa3b854: 0x3
0xbfa3b855: 0x0
0xbfa3b856: 0x0
0xbfa3b857: 0x0
0xbfa3b858: 0x3
0xbfa3b859: 0x0
0xbfa3b85a: 0x0
0xbfa3b85b: 0x0
0xbfa3b85c: 0x0
0xbfa3b85d: 0x0
0xbfa3b85e: 0x0
0xbfa3b85f: 0x0
0xbfa3b860: 0x0
0xbfa3b861: 0x0
0xbfa3b862: 0x0
0xbfa3b863: 0x0
0xbfa3b864: 0x0
0xbfa3b865: 0x0
0xbfa3b866: 0x0
0xbfa3b867: 0x0
0xbfa3b868: 0xa8
0xbfa3b869: 0xb8
0xbfa3b86a: 0xa3
0xbfa3b86b: 0xbf
0xbfa3b86c: 0x71
0xbfa3b86d: 0x84
0xbfa3b86e: 0x4
0xbfa3b86f: 0x8
0xbfa3b870: 0x3
0xbfa3b871: 0x0
0xbfa3b872: 0x0
0xbfa3b873: 0x0
```

buf

ebp

rtn addr

ct

# Exploits

- Run code determined by attacker
- Old way
  - Include attack code in buffer value
  - Prevented by modern defenses: NX and randomized stack base
- Modern way
  - Return-to-libc attack
  - Configure the stack to run code in the victim's address space

# Find Addr to Call Shell Fn

- Jump to location where call to shell function occurs (In main function)
- What address is this at?
  - Need to look at assembly code
- Step 1:
  - Build victim in assembly
  - ‘make victim.s
- Step 2:
  - Insert label before call to shell and rerun
  - ‘make victim-label’

# Add Label before Call

- In cse544-victim.s

```
main:
    leal    4(%esp), %ecx
    andl    $-16, %esp
    pushl   -4(%ecx)
    pushl   %ebp
    movl    %esp, %ebp
    pushl   %ebx
    pushl   %ecx
    subl    $48, %esp
    movl    %ecx, %ebx
    movl    4(%ebx), %eax
    movl    %eax, 4(%esp)
    movl    (%ebx), %eax
    movl    %eax, (%esp)
JMP_ADDR:
    call    shell
    movl    $0, 16(%esp)
    movl    $0, 12(%esp)
    movl    -12(%ebp), %eax
    movl    %eax, 8(%esp)
    movl    4(%ebx), %eax
    movl    %eax, 4(%esp)
    movl    (%ebx), %eax
    movl    %eax, (%esp)
            call        victim
```

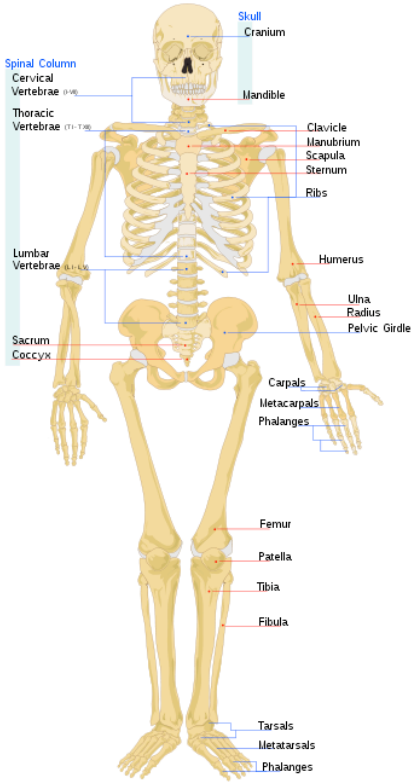
(1) Find ‘call shell’
(2) Add ‘JMP_ADDR:’ to the prior line

# Launch Attack

- Execute the victim program with the malicious buffer
  - From the attack program
  - Use the *system* system call to involve the *exec* system call on victim

# Anatomy of Control Flow Attacks

- Two steps
- First, the attacker changes the control flow of the program
  - In buffer overflow, overwrite the return address on the stack
  - What are the ways that this can be done?
- Second, the attacker uses this change to run code of their choice
  - In buffer overflow, inject code on stack
  - What are the ways that this can be done?



# Return-oriented Programming

- General approach to control flow attacks
- Demonstrates how general the two steps of a control flow attack can be
- First, change program control flow
  - In any way
- Then, run any code of attackers' choosing, including the code in the existing program

# Return-oriented Programming

PENNSTATE 1855

- ROP slides

# Stuxnet

- Stuxnet slides

# Summary

- The types of attacks that we must defend against are becoming more complex
- Return-oriented programming shows us that *any* attacker-dictated change in program control flow can lead to arbitrary malice
- Stuxnet shows that ad hoc system defenses can be evaded by an adversary
- We must apply principled approaches to defense to make significant strides in defense