Project 1
Buffer Overflow

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

January 19, 2010
Project Overview

• Due: Jan 28, 2010

• Given
  ‣ Vulnerable Program: cse544-victim.c
  ‣ Attack Program: cse544-attack.c

• Configure attack program to overflow buffer and take control of victim program
  ‣ Identify vulnerable buffer
  ‣ Overflow buffer to call function “shell”
  ‣ Correct injection causes the execution of a new shell
The Players

- A buffer in the victim
  - Allocated on the stack
- Attack code
  - Call the victim with inputs necessary to overflow buffer
  - Overwrites the return address on the stack
- Exploit
  - In a specific manner necessary to gain control of the execution
  - Jump-to-libc (actually a local function) attack
The Players

• A buffer in the victim
  ‣ Allocated on the stack

• Attack code
  ‣ Call the victim with inputs necessary to overflow buffer
  ‣ Overwrites the return address on the stack

• Exploit
  ‣ In a specific manner necessary to gain control of the execution
  ‣ Jump-to-libc (actually a local function) attack
Vulnerability

- Find an appropriate buffer in cse544-victim.c
  - Not hard as there are not that many
Determine how to attack

- Examine stack as victim runs
  - Build: make victim
  - Run: ./victim foo bar

```c
...
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;
```
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

```
buf

ebp
rtn addr
ct

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
```

V
ictim function
– Dumps contents of stack

```
... printf(BEFORE picture of stack);
for (i=((unsigned) buf-8); i<((unsigned) (char*)&ct)+8; i++)
  printf(%p: 0x%x
, (void*)i, *(unsigned char*) i);
/* run overflow */
  for (i=1; i<tmp; i++)
    printf(i = %d; tmp = %d; ct = %d; &tmp = %p
, i, tmp, ct, (void*) &tmp);
  strcpy(p, inputs[i]);
/* print stack after the fact */
  printf(AFTER iteration %d
, i);
for (j=((unsigned) buf-8); j<(unsigned) (char*)&ct)+8; j++)
  printf(%p: 0x%x
, (void*)j, *(unsigned char*) j);
p += strlen(inputs[i]);
if (i+1!= tmp)
  *p++ = ' ';
}
printf(buf = %s
, buf);
printf(victim: %p
, (void*) &victim);
return 0;
```
Find Return Address Offset

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

```c
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)
Exploit

• Run code determined by attacker

• Jump-to-libc attack
  ‣ Configure the stack to run code in the victim’s address space

• Choose code: function shell (see cse544-victim.c)
  ‣ Need to get the address to call this function
  ‣ Need to prepare the stack for the call
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

```assembly
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
In cse544-victim.s

Before

```assembly
victim:
pushl  %ebp
movl   %esp, %ebp
subl   $40, %esp
movl   $0, 20(%ebp)
jmp    .L4

.L5:
movl   20(%ebp), %eax
movb   $0, -12(%ebp,%eax)
addl   $1, 20(%ebp)

.L4:
cmpl   $11, 20(%ebp)
jle     .L5
leal   -12(%ebp), %eax
movl   %eax, 16(%ebp)
movl   8(%ebp), %eax
movl   %eax, -16(%ebp)
movl   $main, 4(%esp)
movl   $.LC0, (%esp)
call    printf
movl   $shell, 4(%esp)
movl   $.LC1, (%esp)
call    printf
```
Build and Run Modified Victim

• Build
  – ‘make victim-label’
• Run
  – ‘./victim-label foo bar’

In shell
i = 3; inputs = 0xbfa3b944
&main = 0x8048424
&shell = 0x80486df
&inputs[0] = 0xbfa3b944
&buf[0] = 0xbfa3b85c
BEFORE picture of stack
0xbfa3b854: 0x3
0xbfa3b855: 0x0
0xbfa3b856: 0x0
0xbfa3b857: 0x0

Now ‘shell’ prints to the call address (JMP_ADDR)
Modify Attack Program

- Add return address offset to `rtn_addr_distance`
- Add replacement return address (for calling shell) to `rtn1` to `rtn4`. 8-bit hex values.
- Add code to build buffer replaces the return address
- Need to include the two arguments to `shell` function in the buffer (0, address of argv) after the return address
Modify Attack Program

- 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
Run the Attack

• Build
  – ‘make attack’

• Run
  – ‘./attack’

• Result
  – Should open a shell
  – Stop at a prompt: ‘$’
  – You can run shell commands and exit

• Don’t worry about seg fault after exit