

# CS 47

## Last Topics from Chapter 3

### Topics

- gdb
- Buffer overflow and security

class13b.ppt

CS 47 Spring 2008

## Debugging Assembly Language Programs

```
int fac(int n);

int main(int argc, char* argv[])
{
    int i = 1, n;
    if(argc < 2) {printf("Usage: fac i1 i2 i3 ... (list of
non-negative integers)");}
    while (i < argc)
    {
        n = atoi(argv[i]);
        if(n < 0) printf("%d out of range\n", n);
        else printf("%d! = %d\n", n, fac(n));

        i++;
    };
    return(0);
}
-2-
```

CS 47 Spring, 2008

## Assembly Procedure

```
.file "fac.s"
.section .text
.p2align 4,,15
.globl _fac
_fac:
    pushl %ebp
    movl %esp, %ebp
    xorl %eax, %eax
    incl %eax          /* set result to 1 */
    movl %eax, %ecx   /* set i to 1 */
    cmpl 8(%ebp), %ecx /* 8(%ebp) is n */
    jg L2             /* if (i > n) */
L1:
    imull %ecx, %eax  /* result *= i */
    incl %ecx         /* i += 1 */
    cmpl 8(%ebp), %ecx
    jle L1           /* while i <= n */
L2: leave
    ret
```

- 3 -

CS 47 Spring, 2008

## Compiling and Assembling

```
gcc -o fac.exe [-O2] [-Wall] run_fac.c fac.s
```

**Combines both source files into fac.exe**

**Use gdb, if necessary, to debug**

**Instructions are on p. 205 of the text**

- 4 -

CS 47 Spring, 2008

## Debugging

```
gdb fac.exe           /* start debugger,  
                        new prompt: (gdb) */  
(gdb) quit           /* stop debugger */  
(gdb) run 5 10       /* run with these inputs*/  
(gdb) disas fac      /* disassemble function fac*/  
(gdb) break *0x1efd  /* set a break point */  
(gdb) print $eax     /* print register in decimal */  
(gdb) print /x $eax /* print register in hex */  
(gdb) x /20b fac     /* examine 20 bytes of fac */  
(gdb) stepi          /* execute one instruction */
```

- 5 -

CS 47 Spring, 2008

## Debugging (cont)

```
(gdb) info registers /* show all registers */  
(gdb) info frame    /* describe current stack frame */  
(gdb) info stack    /* shows chain of calls */  
  
(gdb) continue     /* continue after break */  
(gdb) delete 1      /* delete breakpoint 1 */  
(gdb) help          /* get more information */
```

- 6 -

CS 47 Spring, 2008

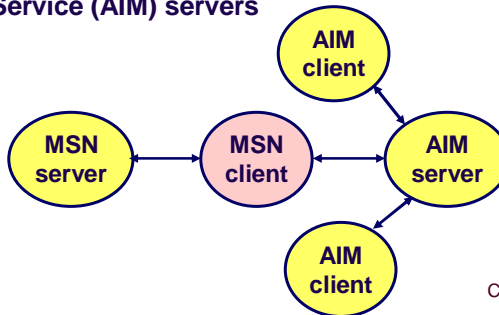
## Internet Worm and IM War

### November, 1988

- Internet Worm attacks thousands of Internet hosts.
- How did it happen?

### July, 1999

- Microsoft launches MSN Messenger (instant messaging system).
- Messenger clients can access popular AOL Instant Messaging Service (AIM) servers



- 7 -

CS 47 Spring, 2008

## Internet Worm and IM War (cont.)

### August 1999

- Mysteriously, Messenger clients can no longer access AIM servers.
- Microsoft and AOL begin the IM war:
  - AOL changes server to disallow Messenger clients
  - Microsoft makes changes to clients to defeat AOL changes.
  - At least 13 such skirmishes.
- How did it happen?

**The Internet Worm and AOL/Microsoft War were both based on *stack buffer overflow* exploits!**

- many Unix functions do not check argument sizes.
- allows target buffers to overflow.

- 8 -

CS 47 Spring, 2008

## String Library Code

- Implementation of Unix function gets
  - No way to specify limit on number of characters to read

```
/* Get string from stdin */
char *gets(char *dest)
{
    int c = getc();
    char *p = dest;
    while (c != EOF && c != '\n') {
        *p++ = c;
        c = getc();
    }
    *p = '\0';
    return dest;
}
```

- Similar problems with other Unix functions
  - strcpy: Copies string of arbitrary length
  - scanf, fscanf, sscanf, when given %s conversion specification

- 9 -

CS 47 Spring, 2008

## Vulnerable Buffer Code

```
/* Echo Line */
void echo()
{
    char buf[4]; /* Way too small! */
    gets(buf);
    puts(buf);
}
```

```
int main()
{
    printf("Type a string:");
    echo();
    return 0;
}
```

- 10 -

CS 47 Spring, 2008

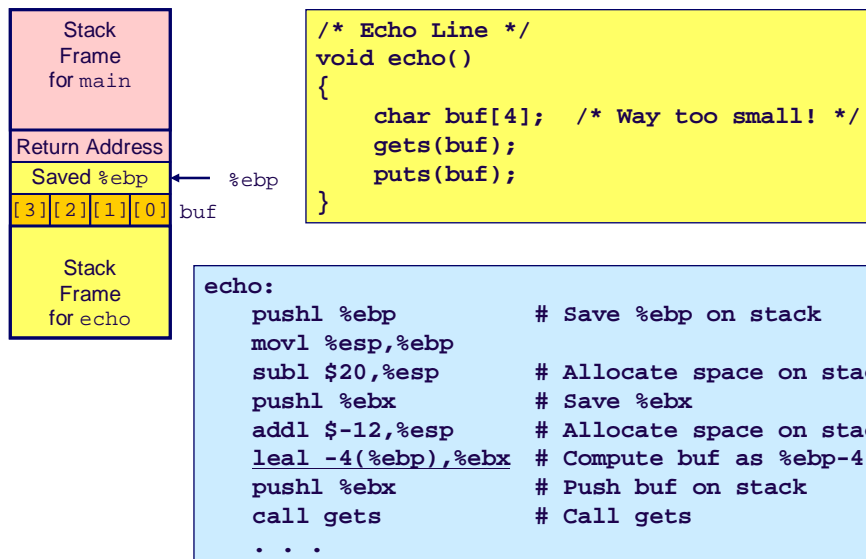
# Buffer Overflow Executions

```
unix> ./bufdemo
Type a string:123
123
```

```
unix> ./bufdemo
Type a string:12345
Segmentation Fault
```

```
unix> ./bufdemo
Type a string:12345678
Segmentation Fault
```

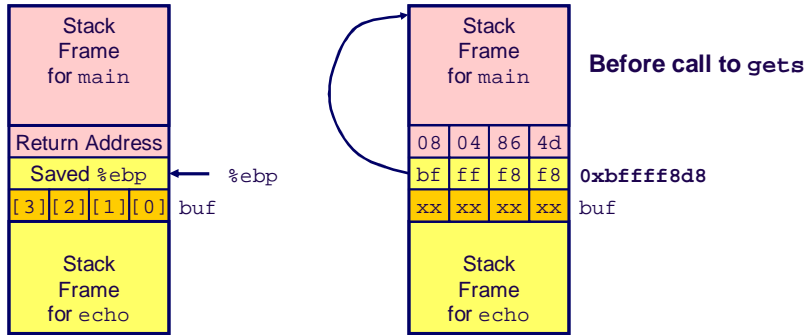
# Buffer Overflow Stack



# Buffer Overflow Stack Example

```

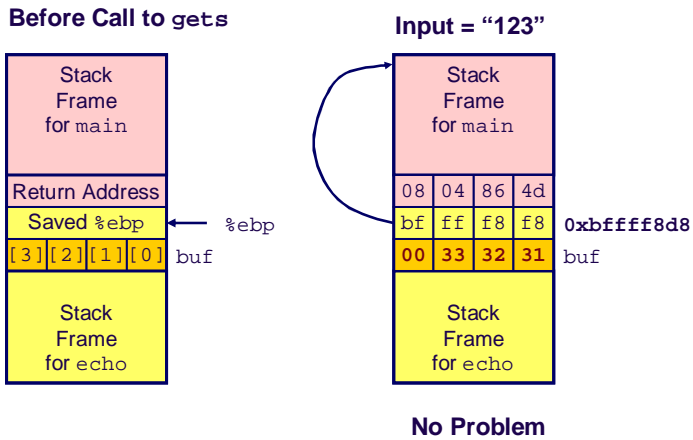
unix> gdb bufdemo
(gdb) break echo
Breakpoint 1 at 0x8048583
(gdb) run
Breakpoint 1, 0x8048583 in echo ()
(gdb) print /x *(unsigned *)$ebp
$1 = 0xbffff8f8
(gdb) print /x *((unsigned *)$ebp + 1)
$3 = 0x804864d
    
```



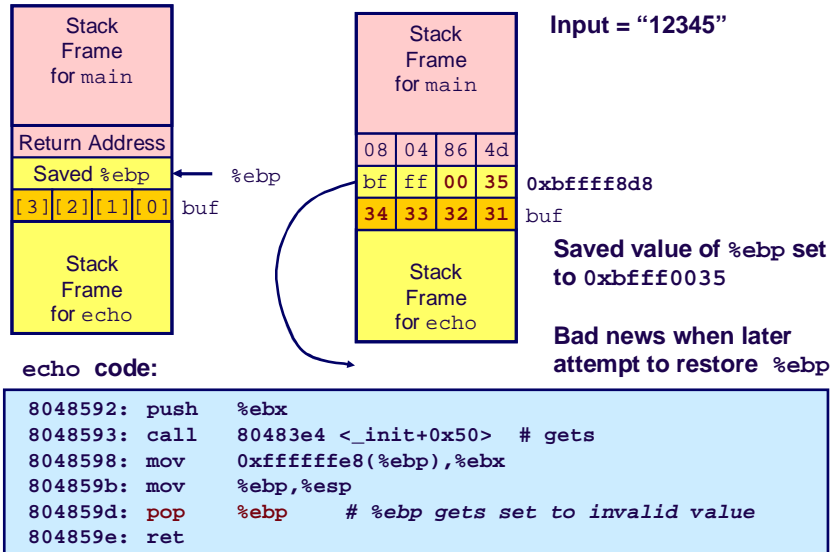
```

8048648: call 804857c <echo>
804864d: mov 0xfffffe8(%ebp),%ebx # Return Point
    
```

# Buffer Overflow Example #1



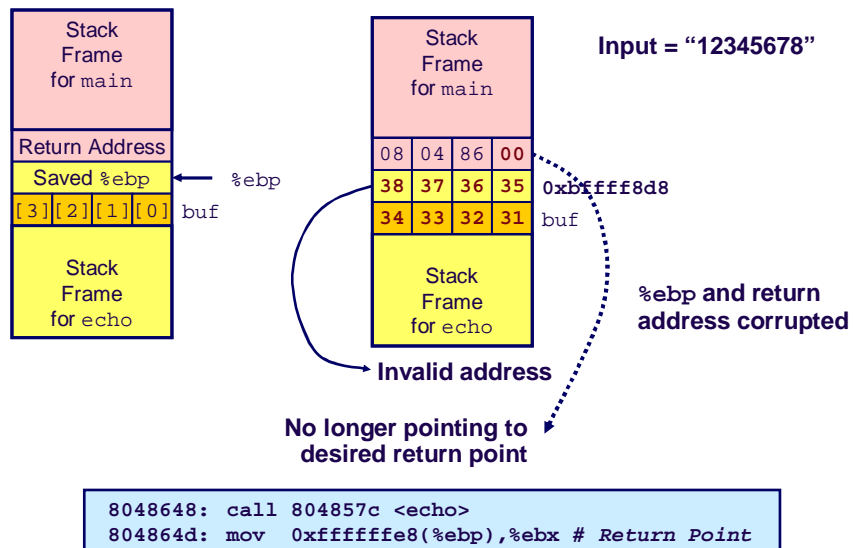
## Buffer Overflow Stack Example #2



- 15 -

CS 47 Spring, 2008

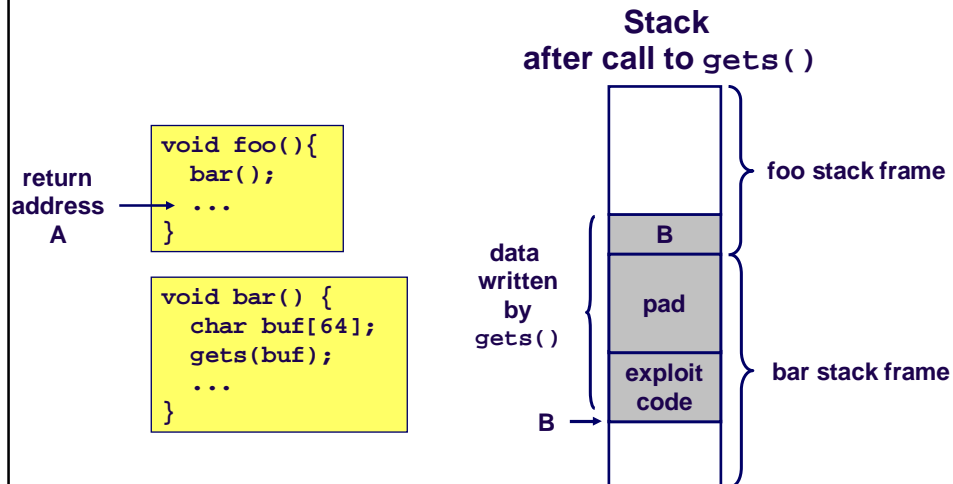
## Buffer Overflow Stack Example #3



- 16 -

CS 47 Spring, 2008

## Malicious Use of Buffer Overflow



- Input string contains byte representation of executable code
- Overwrite return address with address of buffer
- When `bar()` executes `ret`, will jump to exploit code

- 17 -

CS 47 Spring, 2008

## Exploits Based on Buffer Overflows

*Buffer overflow bugs allow remote machines to execute arbitrary code on victim machines.*

### Internet worm

- Early versions of the finger server (`fingerd`) used `gets ( )` to read the argument sent by the client:
  - `finger droh@cs.cmu.edu`
- Worm attacked `fingerd` server by sending phony argument:
  - `finger "exploit-code padding new-return-address"`
  - exploit code: executed a root shell on the victim machine with a direct TCP connection to the attacker.

- 18 -

CS 47 Spring, 2008

# Exploits Based on Buffer Overflows

*Buffer overflow bugs allow remote machines to execute arbitrary code on victim machines.*

## IM War

- AOL exploited existing buffer overflow bug in AIM clients
- exploit code: returned 4-byte signature (the bytes at some location in the AIM client) to server.
- When Microsoft changed code to match signature, AOL changed signature location.

Date: Wed, 11 Aug 1999 11:30:57 -0700 (PDT)  
From: Phil Bucking <philbucking@yahoo.com>  
Subject: AOL exploiting buffer overrun bug in their own software!  
To: rms@pharlap.com

Mr. Smith,

I am writing you because I have discovered something that I think you might find interesting because you are an Internet security expert with experience in this area. I have also tried to contact AOL but received no response.

I am a developer who has been working on a revolutionary new instant messaging client that should be released later this year.

...

It appears that the AIM client has a buffer overrun bug. By itself this might not be the end of the world, as MS surely has had its share. But AOL is now \*exploiting their own buffer overrun bug\* to help in its efforts to block MS Instant Messenger.

....

Since you have significant credibility with the press I hope that you can use this information to help inform people that behind AOL's friendly exterior they are nefariously compromising peoples' security.

Sincerely,  
Phil Bucking  
Founder, Bucking Consulting  
philbucking@yahoo.com

**It was later determined that this email originated from within Microsoft!**