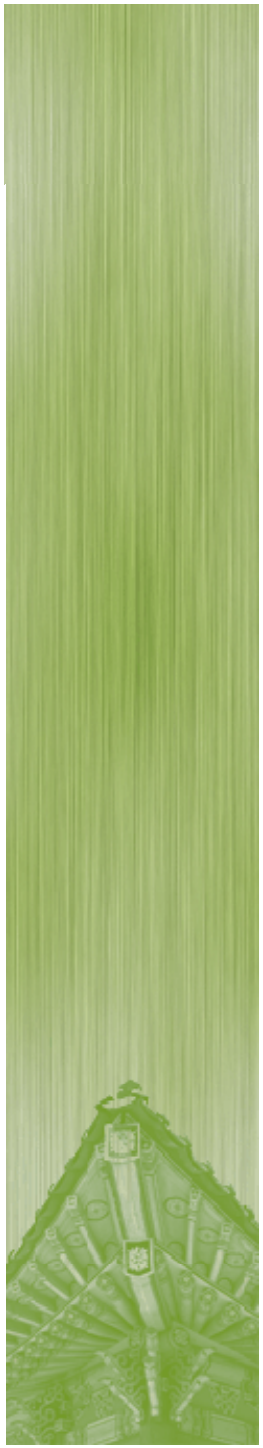


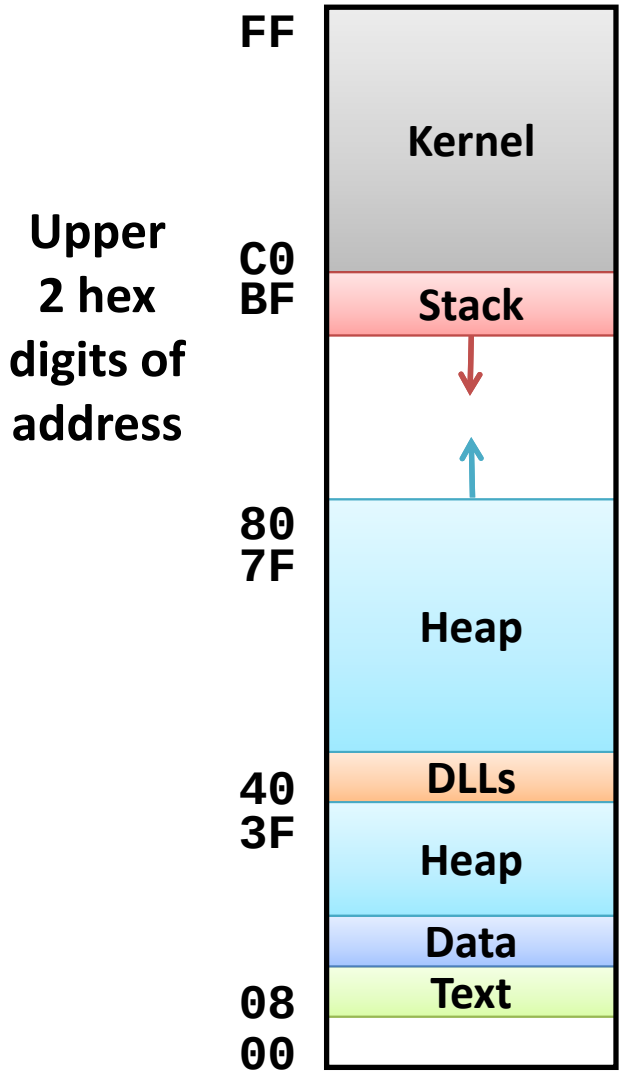
Buffer Overflow

Jin-Soo Kim (jinsookim@skku.edu)
Computer Systems Laboratory
Sungkyunkwan University
http://csl.skku.edu

SKK UNIVERSITY

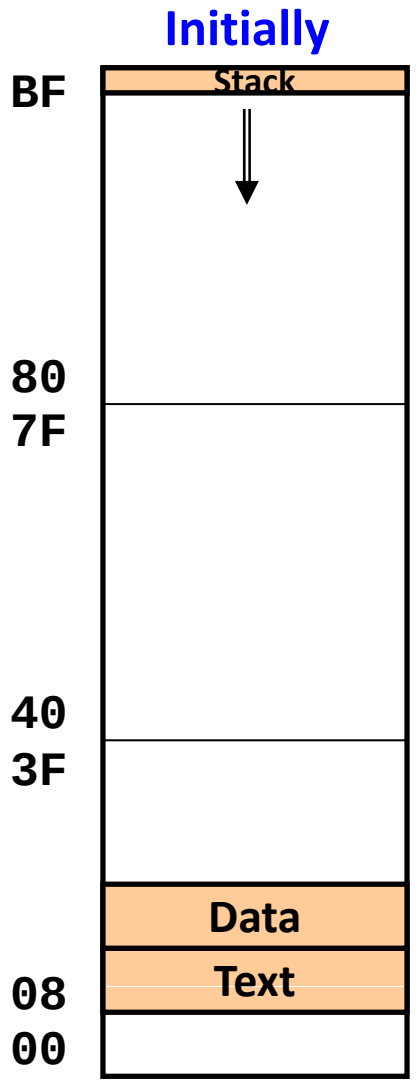
IA-32/Linux Memory Layout

- **Stack**
  - Runtime stack (8MB limit)
- **Heap**
  - Dynamically allocated storage
  - When call malloc(), calloc(), new()
- **DLLs (shared libraries)**
  - Dynamically linked libraries
  - Library routines (e.g., printf, gets)
  - Linked into object code when first executed
- **Data**
  - Statically allocated data
  - e.g., arrays & strings declared in code
- **Text**
  - Executable machine instructions
  - Read-only



Text & Stack Example

```
(gdb) break main
(gdb) run
  Breakpoint 1, 0x804856f in main ()
(gdb) print $esp
  $3 = (void *) 0xbffffc78
```



Vulnerable Buffer Code

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

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

```
$ ./bufdemo
Type:123
123

$ ./bufdemo
Type: 12345
Segmentation Fault

$ ./bufdemo
Type: 12345678
Segmentation Fault
```

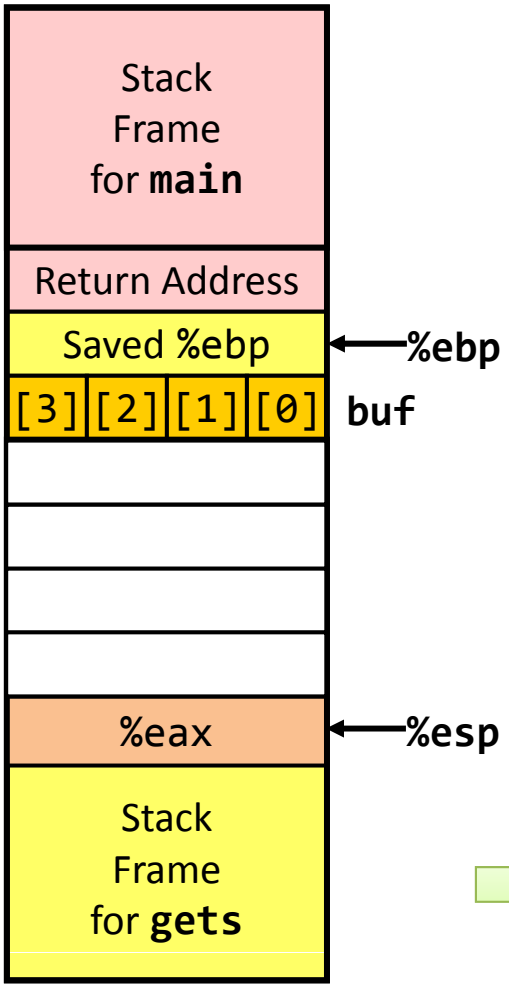
String Library Code

- **Implementation of Unix function gets()**
  - No way to specify limit on # 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**, given **%s** conversion specification

Buffer Overflow (1)

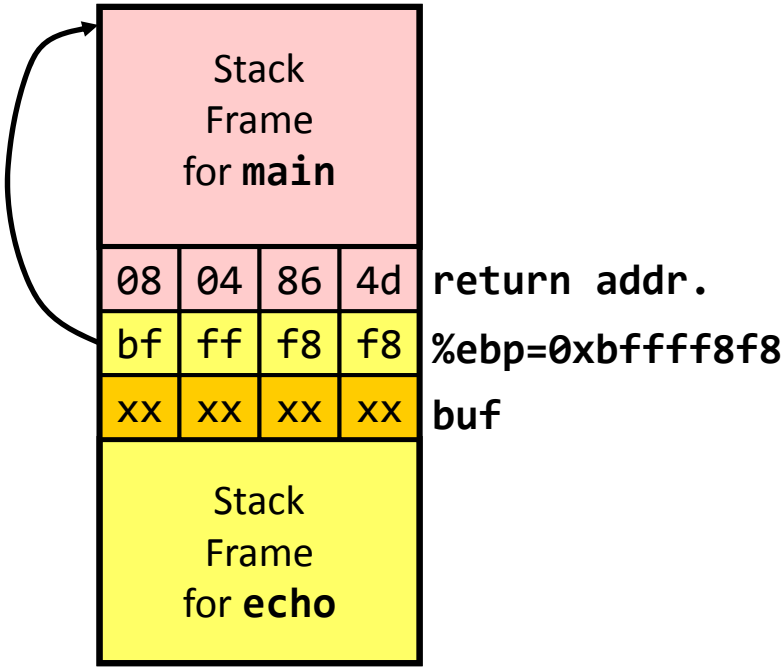



```
/* Echo Line */
void echo()
{
    char buf[4];      /* Way too small! */
    gets(buf);
    puts(buf);
}
```

```
echo:
 pushl %ebp              # Save %ebp on stack
 movl %esp,%ebp
 subl $24,%esp           # Allocate space
 leal -4(%ebp),%eax      # compute buf as %ebp-4
 movl %eax,(%esp)        # save to the stack
 call gets               # Call gets
   . . .
```

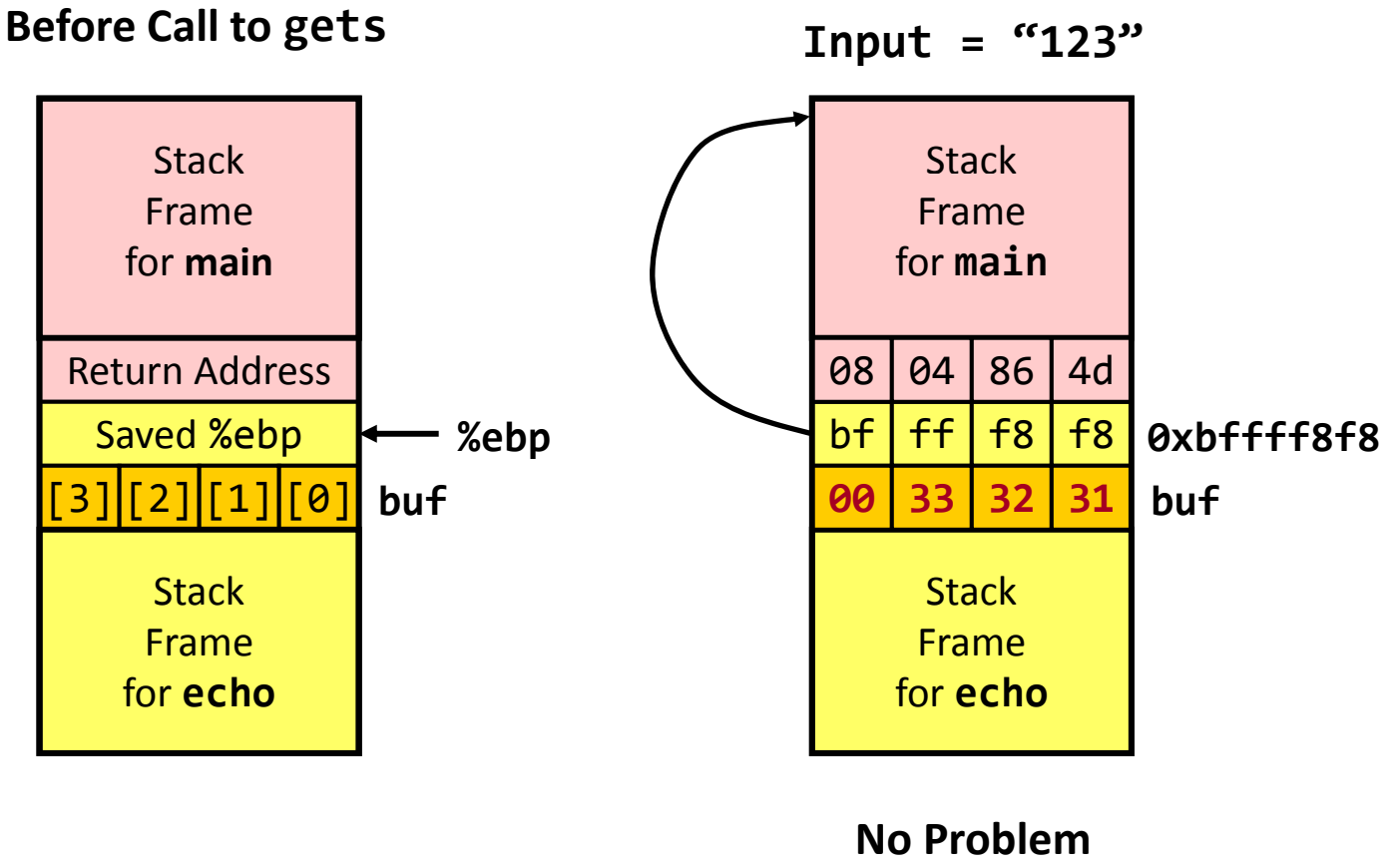
Buffer Overflow (2)

```
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
```

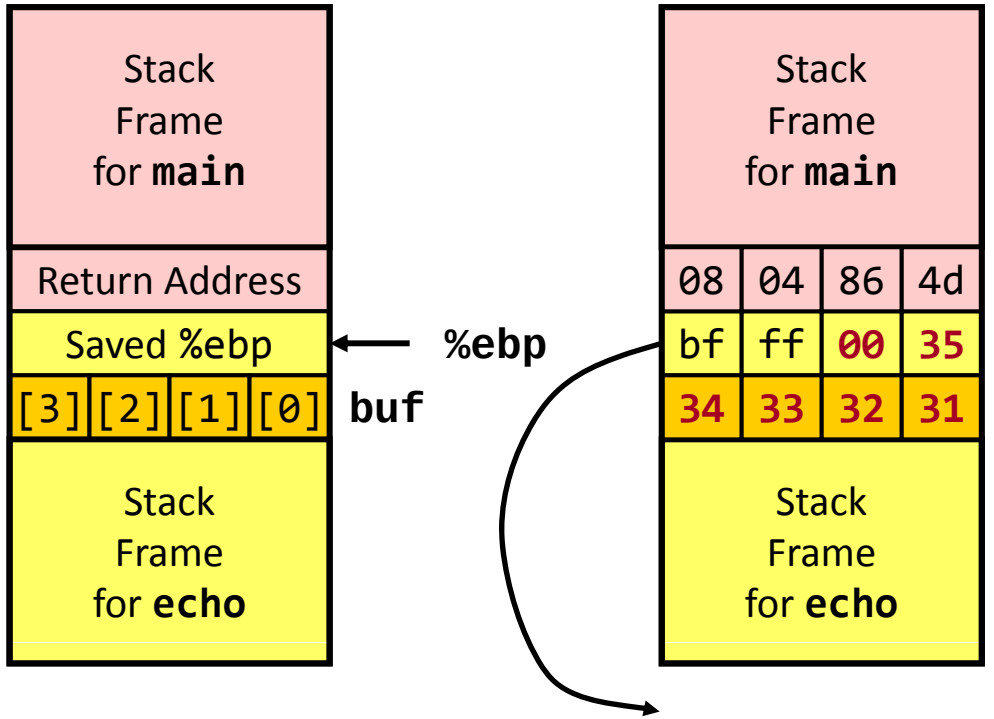



```
8048648:   call 804857c <echo>
804864d:   mov  $0,%eax          # Return Point
```

Buffer Overflow (3)



Buffer Overflow (4)



Input = "12345"

0xbffff8d8

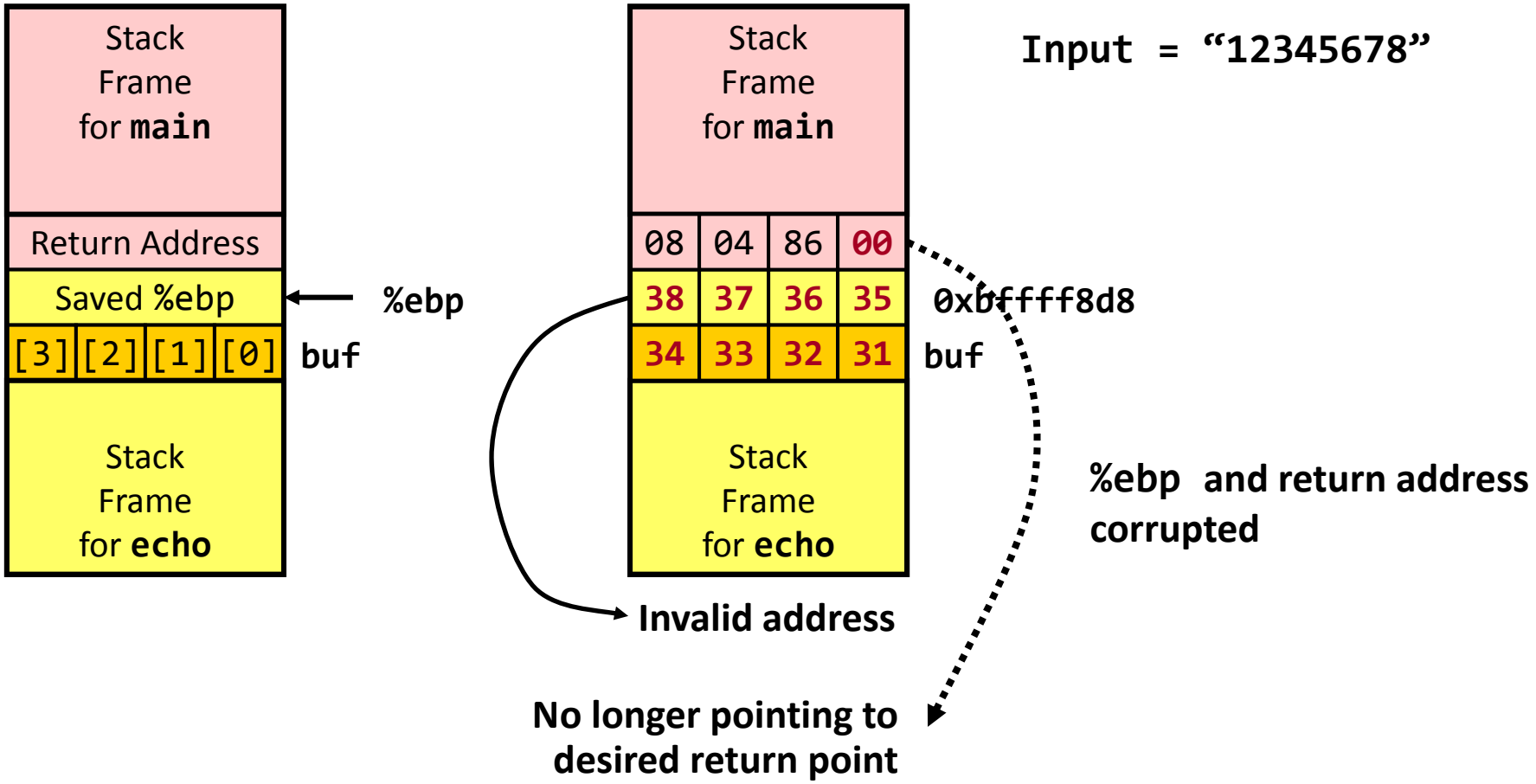
Saved value of %ebp set to 0xbfff0035

Bad news when later attempt to restore %ebp

<echo code>

```
 . . .
8048600:   call   80482c4        # gets
8048605:   leal   0xfffffffc(%ebp),%eax
8048608:   movl   %eax,(%esp)
804860b:   call   80482d4        # puts
8048610:   leave                 # movl %ebp, %esp; popl %ebp
8048611:   ret
```

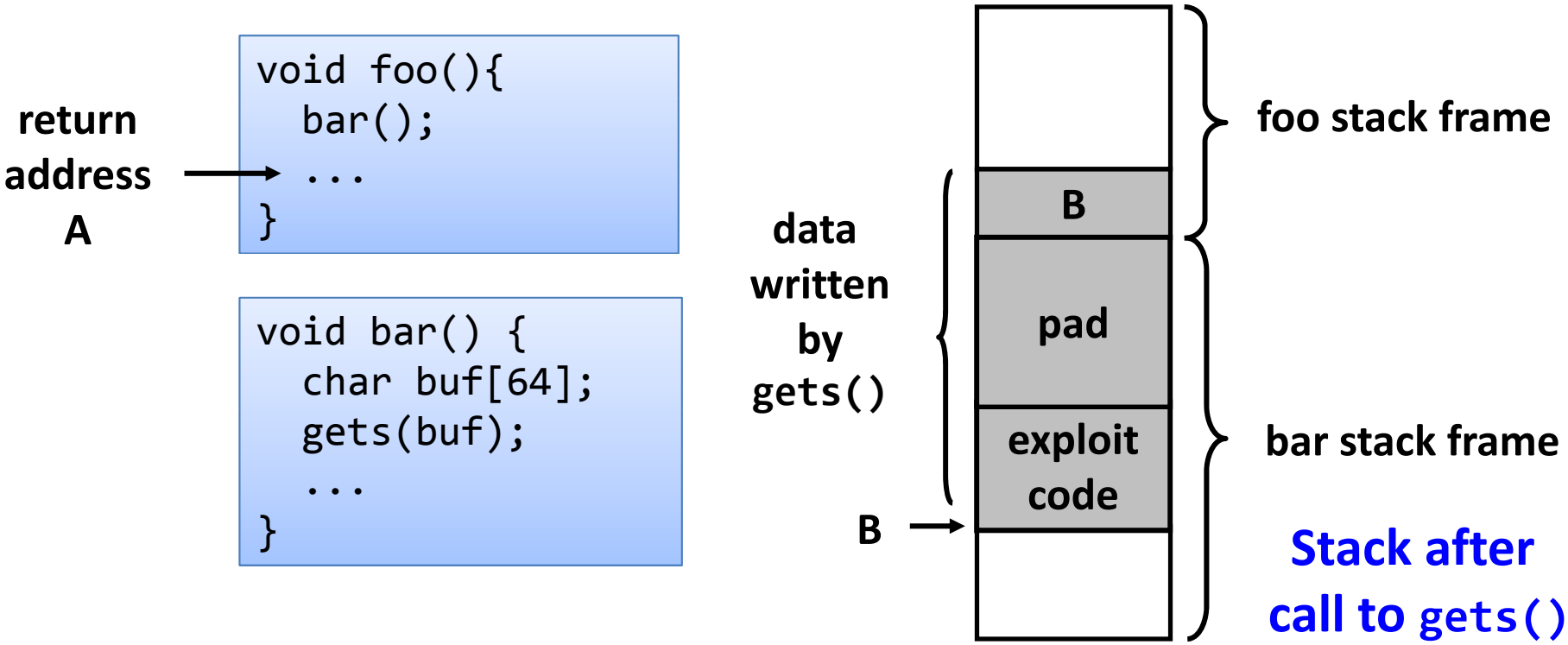
Buffer Overflow (5)




```
8048648:   call 804857c <echo>
804864d:   mov  $0,%eax          # Return Point
```

Buffer Overflow Attack (1)

- **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.



Buffer Overflow Attack (2)

- **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 kildong@skku.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.

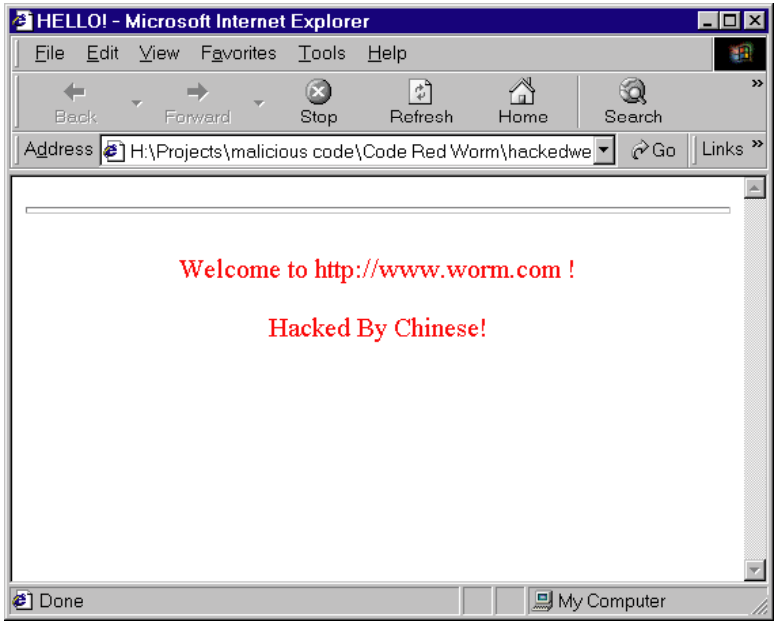
Code Red Worm (1)

- **History**
  - June 18, 2001. Microsoft announces buffer overflow vulnerability in IIS Internet server
  - July 19, 2001. Over 250,000 machines infected by new virus in 9 hours.
  - White house must change its IP address. Pentagon shut down public WWW servers for day.
- **Received strings of form**

```
GET
   /default.ida?NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN....NNNN
   NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN%u9090%u6858%ucbd3%u780
   1%u9090%u6858%ucbd3%u7801%u9090%u6858%ucbd3%u7801%u9090%u909
   0%u8190%u00c3%u0003%u8b00%u531b%u53ff%u0078%u0000%u00=a
HTTP/1.0" 400 325 "-" "-"
```

Code Red Worm (2)

- **Code Red exploit code**
  - Starts 100 threads running
  - Spread self
    - Generate random IP addresses & send attack string
    - Between 1st & 19th of month
  - Denial of service attack to www.whitehouse.gov
    - Send 98,304 packets; sleep for 4-1/2 hours; repeat
    - Between 21st & 27th of month
  - Deface server's home page
    - After waiting 2 hours

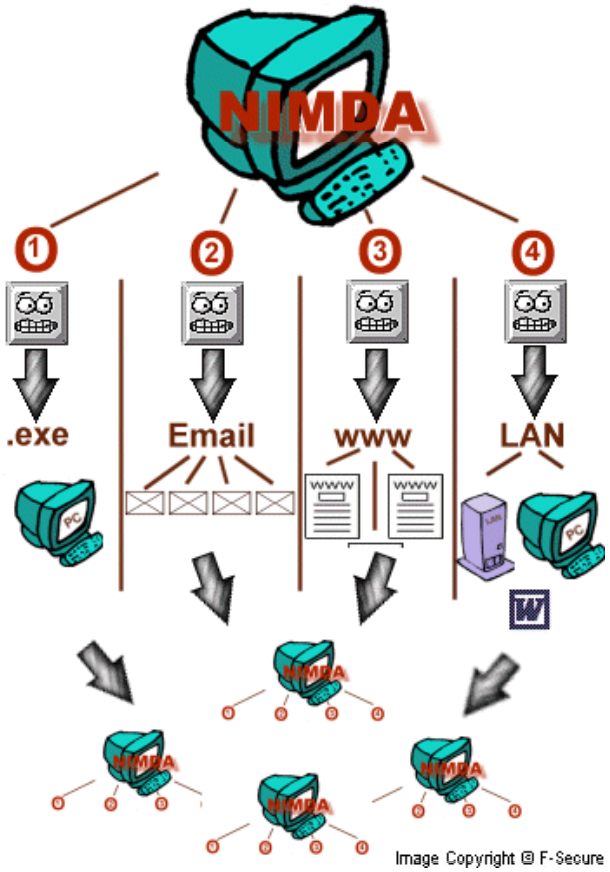


Code Red Worm (3)

- **Code Red effects**
  - Later version even more malicious
    - Code Red II
    - As of April 2002, over 18,000 machines infected
    - Still spreading
  - Paved way for NIMDA
    - Variety of propagation methods
    - One was to exploit vulnerabilities left behind by Code Red II

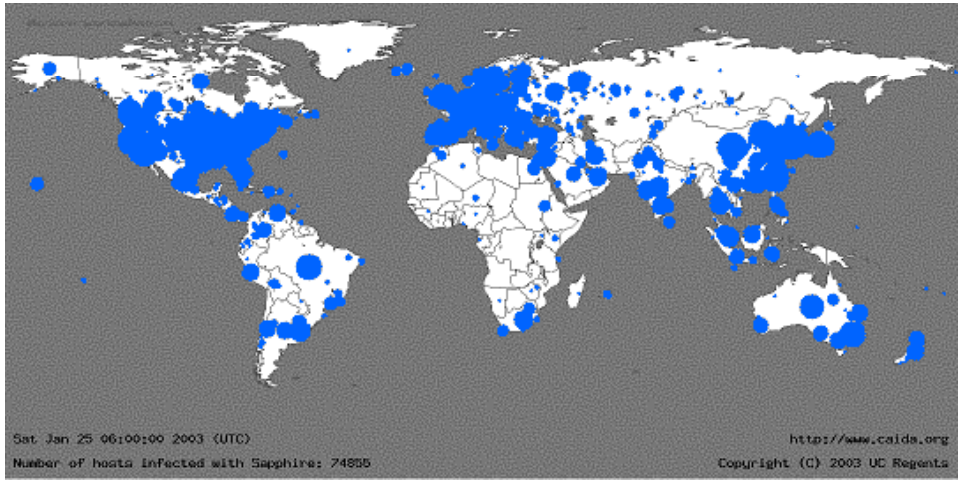
Nimda Worm

- **Nimda (2001)**
  - Five different infection methods:
    - Via e-mail
    - Via open network shares
    - Via browsing of compromised web sites
    - Exploitation of various Microsoft IIS 4.0/5.0 directory traversal vulnerabilities
    - Via back doors left behind by the "Code Red II" and "Sadmind/IIS" worms
  - One of the most widespread virus/worm



SQL Slammer Worm

- **SQL slammer (2003)**
  - Exploited two buffer overflow bugs in Microsoft's SQL Server and Desktop Engine.
  - Infected 75,000 victims within 10 minutes
  - Generate random IP addresses and send itself out to those addresses, slowing down Internet traffic dramatically.
  - 1/25 nationwide Internet shutdown in South Korea



30 minutes after release

Avoiding Buffer Overflow

- **Use library routines that limit string lengths**
  - **`fgets()`** instead of **`gets()`**
  - **`strncpy()`** instead of **`strcpy()`**
  - Don't use **`scanf()`** with **`%s`** conversion specification
    - Use **`fgets()`** to read the string
    - Or use **`%ns`** where **`n`** is a suitable integer

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

System-Level Protections

- **Randomized stack offsets**
  - At start of program, allocate random amount of space on stack
  - Makes it difficult for hacker to predict beginning of inserted code

- **Executable space protection**
  - Mark certain areas of memory as non-executable
  - Hardware assistance:
    - Intel NX (No eXecute) bit
    - AMD XD (eXecute Disable) bit

Summary

- **Memory layout**
  - OS/machine dependent (including kernel version)
  - Basic partitioning:
    - stack, data, text, heap, DLL found in most machines
- **Avoiding buffer overflow vulnerability**
  - Important to use library routines that limit string lengths
- **Working with strange code**
  - Important to analyze nonstandard cases
    - e.g., what happens when stack corrupted due to buffer overflow
  - Helps to step through with GDB