Buffer overflow

Systems Software

A routine that reads data into a buffer

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

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

What happens when we run the program?

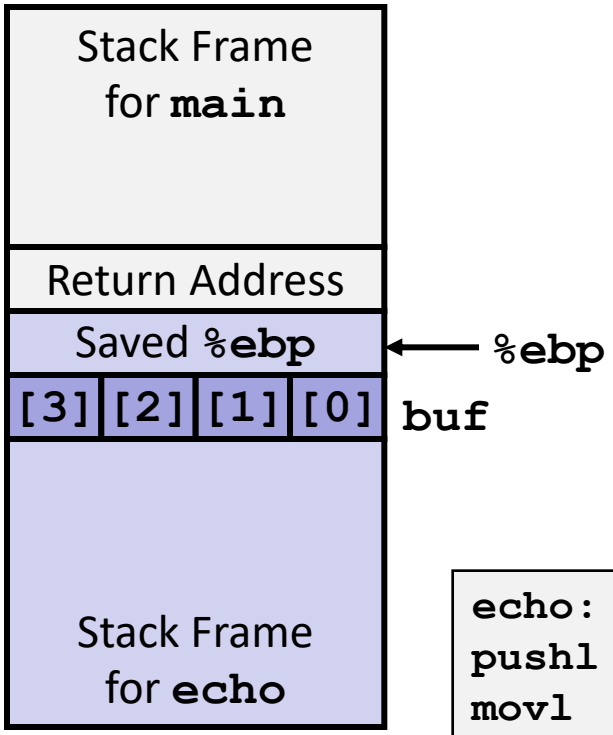
```
unix>./bufdemo
Type a string:123
123
```

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

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

Code disassembly

Before call to gets

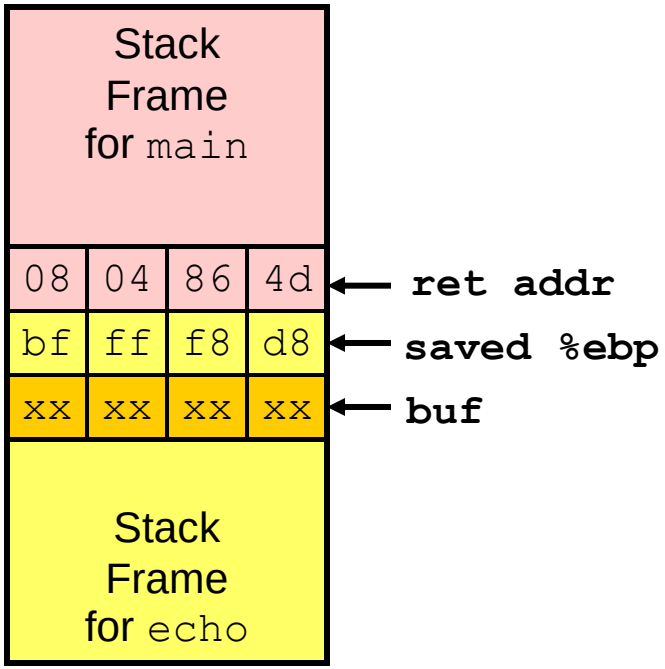



```
/* Echo Line */
void echo()
{
    char buf[4];  /* Way too small! */
    gets(buf);
    puts(buf);
}
```

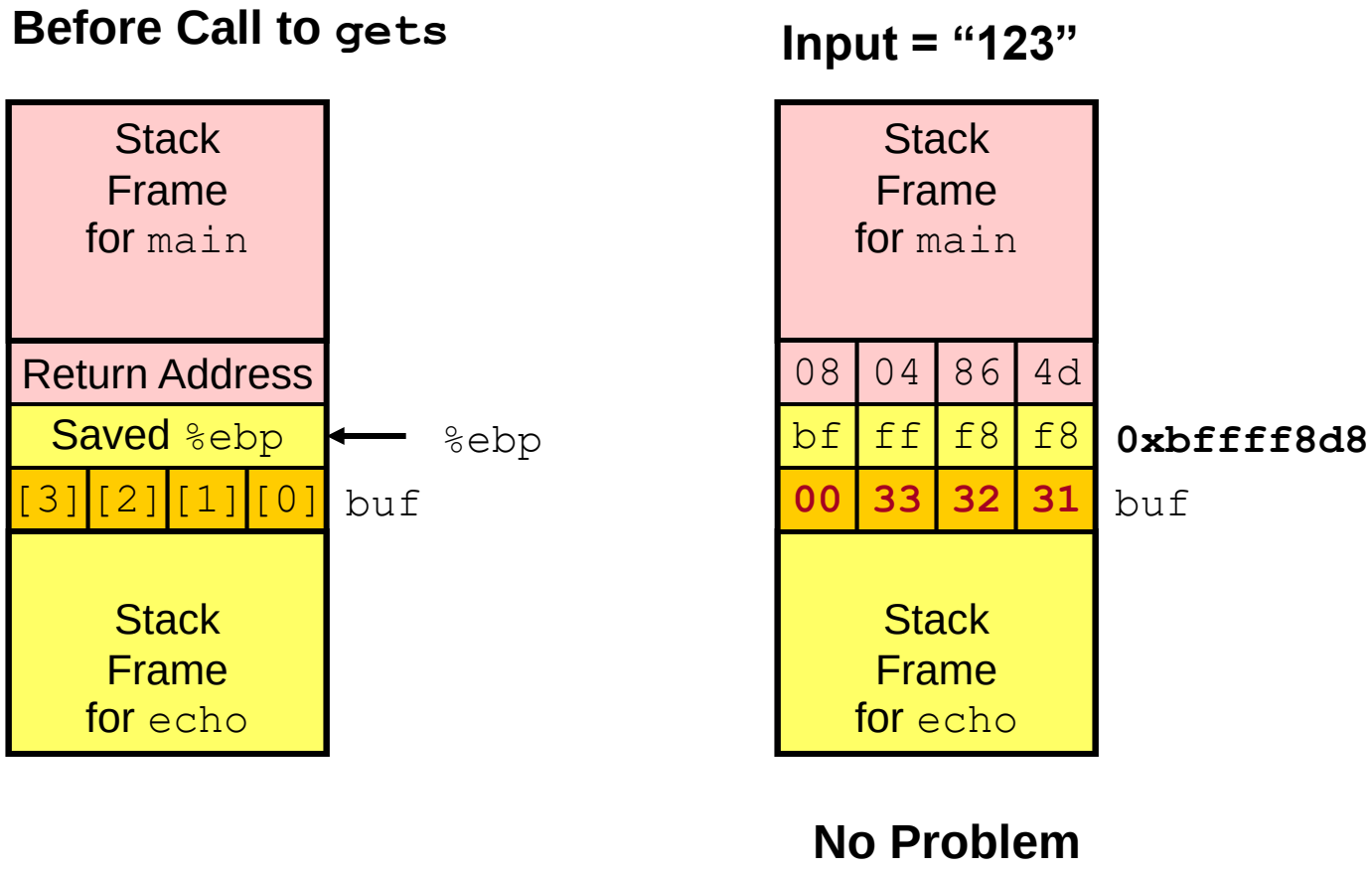
```
echo:
pushl %ebp                  # Save %ebp on stack
movl  %esp, %ebp
subl  $20, %esp             # Allocate stack space
pushl %ebx                  # Save %ebx
addl  $-12, %esp            # Allocate more space
leal  -4(%ebp),%ebx         # Compute buf as %ebp-4
pushl %ebx                  # Push buf on stack
call  gets                  # Call gets
. . .
```

Stack layout for echo

```
unix> gdb bufdemo
(gdb) break echo
Breakpoint 1 at 0x8048583
(gdb) run
Breakpoint 1, 0x8048583 in echo ()
(gdb) print /x *(unsigned *)$ebp
$1 = 0xbffff8d8
(gdb) print /x *((unsigned *)$ebp + 1)
$3 = 0x804864d
```



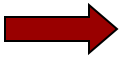
What happens when the string fits into buf[ ]?

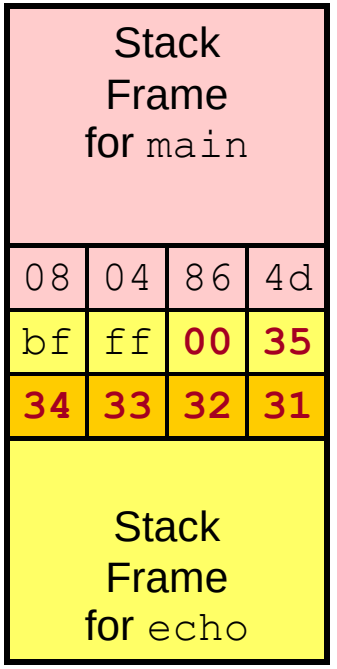


What happens if we enter a string that's too long?

```
8048592:  push   %ebx
8048593:  call   80483e4 <_init+0x50>  # gets
8048598:  mov    0xffffffe8(%ebp),%ebx
804859b:  mov    %ebp,%esp
804859d:  pop    %ebp     # %ebp gets set to invalid value
804859e:  ret
```

- **Say we enter the string "12345"**
  - This clearly overflows the buffer
  - Where do the extra bytes end up?
- **Overwrite the saved %ebp on the stack!**
  - What will this do to the program?

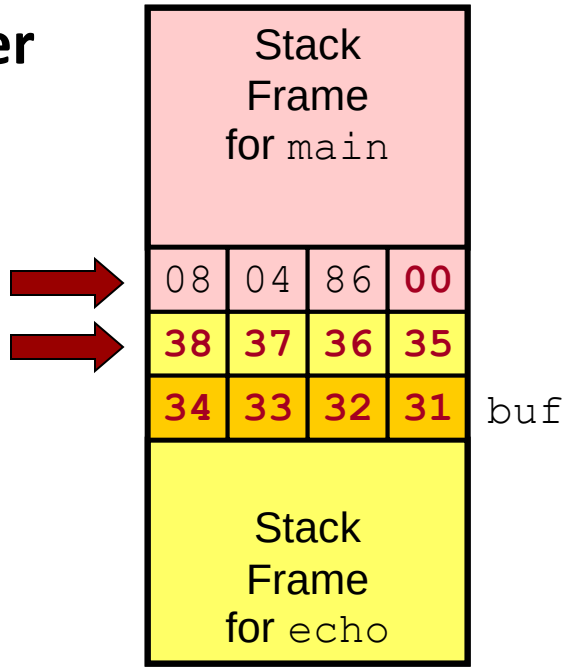




What happens when we enter a string that's too long?

```
8048592:  push   %ebx
8048593:  call   80483e4 <_init+0x50>  # gets
8048598:  mov    0xfffffff8(%ebp),%ebx
804859b:  mov    %ebp,%esp
804859d:  pop    %ebp     # %ebp gets set to invalid value
804859e:  ret
```

- **What happens if we enter an even longer string?**
  - Both saved %ebp and return value are corrupted!
  - Where do the extra bytes end up?
- **What does this do to the program's execution?**



Security hole

- **Buffer overflow overwrites other memory used by the program**
  - For example, the return address on the stack
- **Malicious use of buffer overflow**
  - Attacker feeds program data that overflow the buffer
  - For example, if we can overwrite portions of the stack, we can cause the program to **jump to an address of our choice!**

http://blogs.wsj.com/digits/2014/04/03/bounty-hunter-earns-record-payout-from-facebook/

By REED ALBERGOTTI
Digits
The Wall Street Journal
April 3, 2014

Reginaldo Silva was poring over computer code in November when the one-time software engineer found what he thought was a security loophole on Facebook's servers. The discovery led to the largest "bug bounty" ever paid by the company, and a job for Silva as an engineer at Facebook.

Silva earned $33,500 for notifying Facebook of the flaw, which he said could have allowed a hacker to enter Facebook's servers and execute code.
In a worst-case scenario, the breach could have allowed the hacker to access Facebook accounts or even spread a computer virus to members. A Facebook spokesman said any manipulation of its servers would have been quickly identified and stopped by the company.

Facebook employs hundreds of engineers who ferret out loopholes and bugs, but like many companies offers rewards to "white hat" hackers who find undetected chinks in the digital armor.

"They've found things we wouldn't have found," says Alex Rice, head of product security at Facebook. "The bounty program has by far been the best tool we have for identifying bugs that make it out into the wild."

[...]