Buffer overflows (a security interlude)

Address space layout

the stack discipline

+ C's lack of bounds-checking

HUGE PROBLEM

String Library Code

C standard library function **gets()**

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

pointer to start of an array

same as:
`*p = c;`
`p = p + 1;`

What could go wrong in this code?

Same problem in many functions:

`strcpy`: Copies string of arbitrary length

`scanf`, `fscanf`, `sscanf`, when given **`%s`** conversion specification

Buffer Overflow Disassembly

echo code

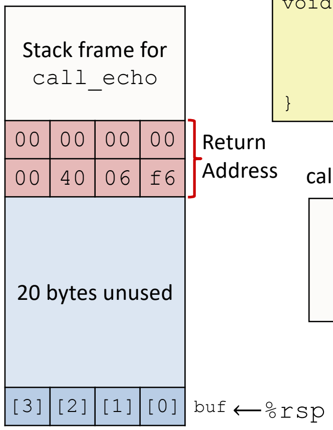
```
 00000000004006cf <echo>:
  4006cf:  48 83 ec 18              sub    $24,%rsp
  4006d3:  48 89 e7                 mov    %rsp,%rdi
  4006d6:  e8 a5 ff ff ff           callq  400680 <gets>
  4006db:  48 89 e7                 mov    %rsp,%rdi
  4006de:  e8 3d fe ff ff           callq  400520 <puts@plt>
  4006e3:  48 83 c4 18              add    $24,%rsp
  4006e7:  c3                       retq
```

caller code

```
 4006e8:   48 83 ec 08              sub    $0x8,%rsp
  4006ec:  b8 00 00 00 00           mov    $0x0,%eax
  4006f1:  e8 d9 ff ff ff           callq  4006cf <echo>
  4006f6:  48 83 c4 08              add    $0x8,%rsp
  4006fa:  c3                       retq
```

Buffer Overflow Stack Example

Before call to gets



```
void echo() {
    char buf[4];
    gets(buf);
    . . .
}
```

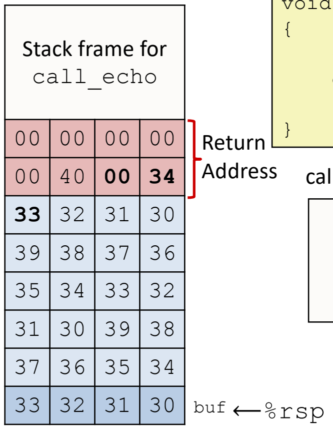
```
echo:
  subq  $24, %rsp
  movq  %rsp, %rdi
  call  gets
  . . .
```

call_echo:

```
  . . .
  4006f1:  callq  4006cf <echo>
  4006f6:  add    $0x8,%rsp
  . . .
```

Buffer Overflow Stack Example #2

After call to gets



Stack frame for `call_echo`			
00	00	00	00
00	40	**00**	**34**
33	32	31	30
39	38	37	36
35	34	33	32
31	30	39	38
37	36	35	34
33	32	31	30

```
void echo()
{
    char buf[4];
    gets(buf);
    . . .
}
```

```
echo:
  subq  $24, %rsp
  movq  %rsp, %rdi
  call  gets
  . . .
```

call_echo:

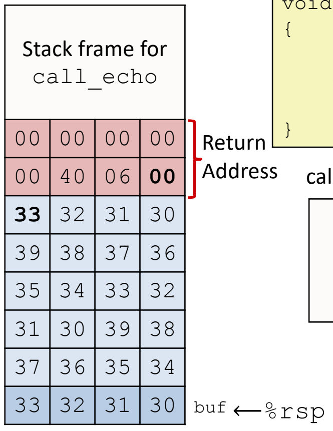
```
  . . .
  4006f1:  callq   4006cf <echo>
  4006f6:  add     $0x8,%rsp
  . . .
```

```
unix> ./bufdemo
Type a string: 0123456789012345678901234
Segmentation Fault
```

Overflowed buffer and corrupted return pointer

Buffer Overflow Stack Example #3

After call to gets



Stack frame for `call_echo`			
00	00	00	00
00	40	06	**00**
33	32	31	30
39	38	37	36
35	34	33	32
31	30	39	38
37	36	35	34
33	32	31	30

```
void echo()
{
    char buf[4];
    gets(buf);
    . . .
}
```

```
echo:
  subq  $24, %rsp
  movq  %rsp, %rdi
  call  gets
  . . .
```

call_echo:

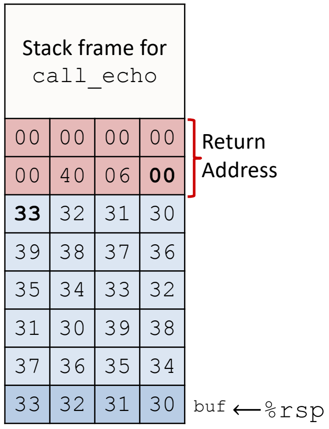
```
  . . .
  4006f1:  callq   4006cf <echo>
  4006f6:  add     $0x8,%rsp
  . . .
```

```
unix> ./bufdemo-nsp
Type a string: 012345678901234567890123
012345678901234567890123
```

Overflowed buffer, corrupted return pointer, but program seems to work!

Buffer Overflow Stack Example #3

After call to gets



Stack frame for `call_echo`			
00	00	00	00
00	40	06	**00**
33	32	31	30
39	38	37	36
35	34	33	32
31	30	39	38
37	36	35	34
33	32	31	30

Some other place in .text

```
  . . .
  400600:  mov     %rsp,%rbp
  400603:  mov     %rax,%rdx
  400606:  shr     $0x3f,%rdx
  40060a:  add     %rdx,%rax
  40060d:  sar     %rax
  400610:  jne     400614
  400612:  pop     %rbp
  400613:  retq
```

"Returns" to unrelated code
Lots of things happen, without modifying critical state
Eventually executes `retq` back to `main`

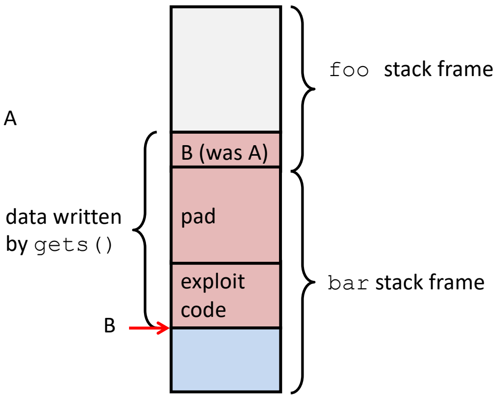
Malicious Use of Buffer Overflow

```
void foo(){
  bar();
  ...
}
```

← return address A

```
int bar()  {
  char buf[64];
  gets(buf);
  ...
  return ...;
}
```

Stack after call to `gets()`



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