**Buffer overflows (a security interlude)**

Address space layout + C's lack of bounds-checking

HUGE PROBLEM

---

**Buffer Overflow Disassembly**

```
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
```

---

**String Library Code**

C standard library function `gets()`

```c
/* 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;
}
```

---

**Buffer Overflow Stack Example**

**Before call to `gets()`**

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

---

**Buffer Overflow Stack Example**

**Return Address**

```
00 00 00 00
00 40 06 06
```

---

**Buffer Overflow Stack Example**

**20 bytes unused**

```
4006e6: 48 83 c4 08 add $0x8,%rsp
4006e6: 48 83 c4 08 add $0x8,%rsp
4006fa: c3 retq
```

---

**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 Stack Example #2

After call to gets

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

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

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

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

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

echo: 
    subq $24, %rsp
    movq %rsp, %rdi
    call 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

buf ← %rsp

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

Overflowed buffer and corrupted return pointer

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

Malicious Use of Buffer Overflow

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)