Buffer overflows

Buffer overflows from 10,000 ft

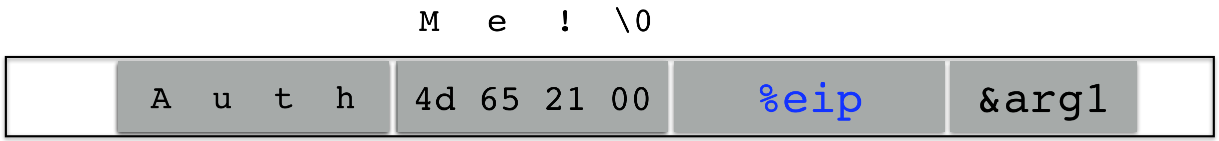
- **Buffer** =
  - Contiguous memory associated with a variable or field
  - Common in C
    - All strings are (NUL-terminated) arrays of `char`'s
- **Overflow** =
  - Put more into the buffer than it can hold
- **Where does the overflowing data go?**
  - Well, now that you are an expert in memory layouts…

Benign outcome

```
void func(char *arg1)
{
    char buffer[4];
    strcpy(buffer, arg1);
    ...
}

int main()
{
    char *mystr = “AuthMe!”;
    func(mystr);
    ...
}
```

Upon return, sets `%ebp` to `0x0021654d`



M e ! \0

A u t h	4d 65 21 00	%eip	&arg1

buffer

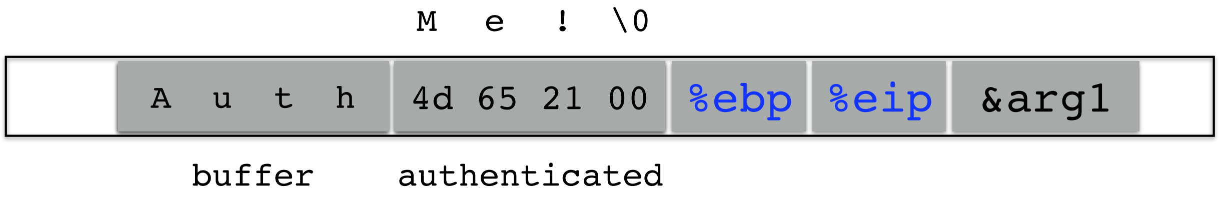
SEGFAULT (0x00216551) (during subsequent access)

Security-relevant outcome

```
void func(char *arg1)
{
    int authenticated = 0;
    char buffer[4];
    strcpy(buffer, arg1);
    if(authenticated) { ...
}

int main()
{
    char *mystr = “AuthMe!”;
    func(mystr);
    ...
}
```

Code still runs; user now ‘authenticated’



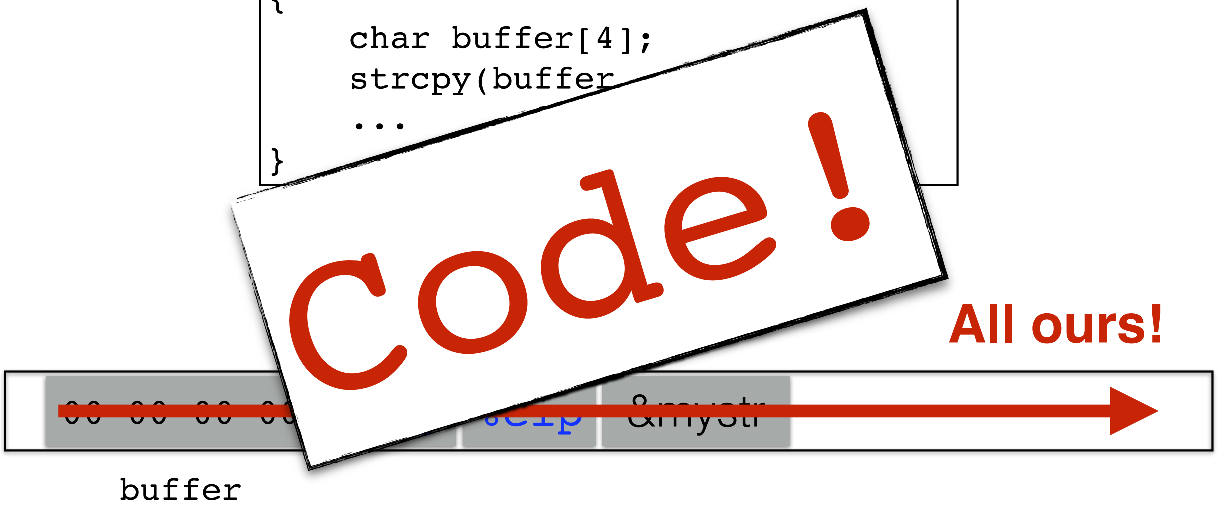
M e ! \0

A u t h | 4d 65 21 00 | %ebp | %eip | &arg1

buffer authenticated

Could it be worse?

```
void func(char *arg1)
{
    char buffer[4];
    strcpy(buffer
    ...
}
```



`strcpy` will let you write as much as you want (til a ‘\0’)

What could you write to memory to wreak havoc?

Aside: User-supplied strings

- These examples provide their own strings
- In reality **strings** come **from *users*** in myriad aways
  - **Text** input
  - **Packets**
  - **Environment variables**
  - **File** input…
- **Validating assumptions** about **user input** is extremely **important**
  - We will discuss it later, and throughout the course