



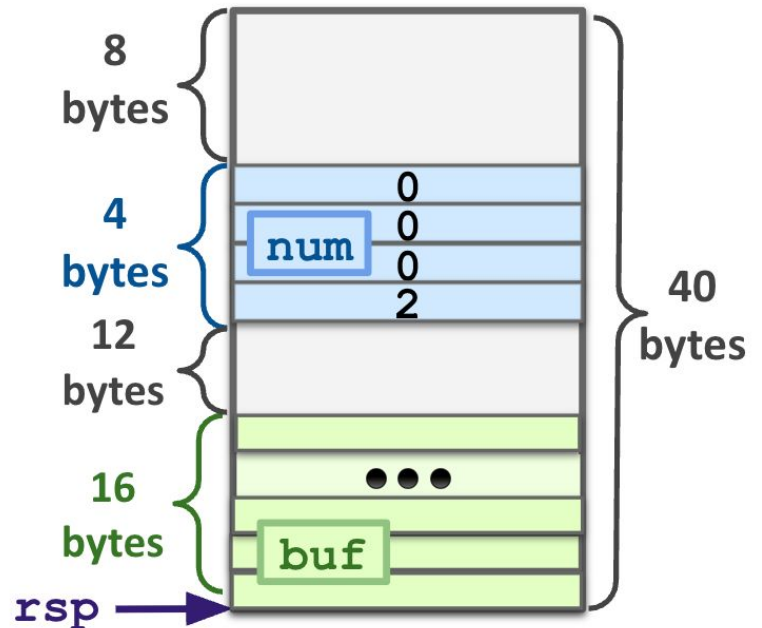
Solution to greet() stack exercise – see the lab slides for more on how the stack diagram was constructed from the C code & assembly!

```
sub    $0x28,%rsp
movl   $0x2,0x1c(%rsp)
mov     $0x402008,%edi
mov     $0x0,%eax
callq   0x401050 <printf@plt>
mov     %rsp,%rax
mov     %rax,%rdi
callq   0x401060 <gets@plt>
```

```
# define BUF_LEN 16
int greet() {
    int num = 2;
    char buf[BUF_LEN];

    printf("What is your name?");
    if (gets(buf)){ ...
    // eventually, return num;
```

Figure's scale is cooked

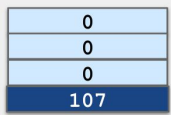


4-panel comic strip on how to override num with buffer overflow!

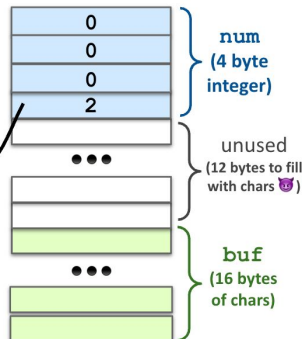
1

Here, we'll accomplish the impossible: tricking `greet()` to return 107! 🍷

In other words, in memory, can we can override the lowest byte of `num` to be 107?



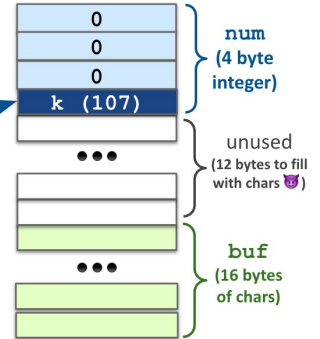
Note that `gets()` will start writing input characters in `buf`, overflowing up to `num`.



2

Here, we'll accomplish the impossible: tricking `greet()` to return 107! 🍷

Yes! Note that the ASCII character with value 107 is `[k]`, so in the bytes that belong to `num`, we have to place a `[k]`. Due to `num` being an `int` data type, the `k` will be interpreted / casted as 107.

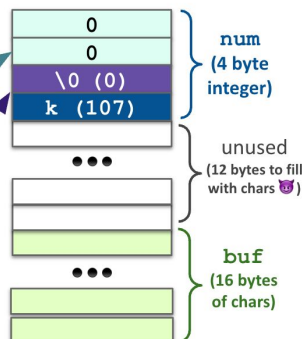


3

Here, we'll accomplish the impossible: tricking `greet()` to return 107! 🍷

Note higher bytes stay 0 as `num` was initialized to 2. To make `num` even larger, you can override them also :)

Also, `gets()` follows the last character with a `\0`, which keeps ASCII value 0!



4

Here, we'll accomplish the impossible: tricking `greet()` to return 107! 🍷

The `buffer + unused` memory is 28 bytes, so our input should be a string of 29 chars total, ending in `[k]` :)

🍷 Example: `aaaaaa...aak`  
28 copies of `a`

