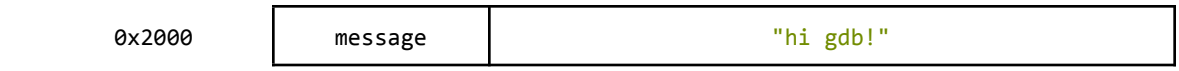
CS 3410

Lab 7 Worksheet: All about Buffer Overflow!

1. Let's analyze a stack!

How is a string represented in a real stack? Given the following message and GDB commands, write the terminal output. Do NOT use GDB for this; try to think on your own. You may find these commands useful for A7.

```
char* message = "hi gdb!";
```




```
(gdb) x/1s message
```

```
(gdb) print 0x2016 - (unsigned long long)&message
```

2. Buffer Overflow overflows my mind

The variable secret sits immediately after buf on the stack. What property must an input string have in order to change the value of secret?

Hint: Think about how many characters are needed to fill all of buf and then spill into the space where secret is stored.

```c
int main() {
    char buf[8];
    int secret = 1234;      // stored just after buf on the stack

    printf("secret is %d\n", secret);
    printf("Enter input: ");
    fgets(buf, 100, stdin);    // read up to 99 characters + null terminator
    printf("You entered: %s\n", buf);
    printf("secret is now %d\n", secret);
    return 0;
}
```

CS 3410

Lab 7 Worksheet: All about Buffer Overflow!

3. Conceptual Stack Smashing

Discuss with your lab group how a stack-based buffer overflow can allow arbitrary code execution.
Specifically, there are three parts to your buffer overflow exploit: filler bytes, a new return address, and shellcode. Define these three terms in your own words and explain how they all fit together to form an exploit payload.

4. Super interesting GDB commands!

Part 4A. Breakpoints

```
(gdb) break main
Breakpoint 1 at 0x555555555169: file main.c, line 5.
(gdb) continue
Continuing.

Breakpoint 1, main () at main.c:5
5       printf("Hello world!\n");
```

1. What line did GDB stop at after running the program?
2. How can you set a breakpoint at line 12 of the same file without restarting GDB?
3. Why might you set a breakpoint at the start of main?

Part 4B. Which GDB command?

You're debugging a program with this info frame output. vulnerable_function has char buf[64];

```
Stack level 0, frame at 0x7fffffffe1f0:
 pc = 0x4012f2 in vulnerable_function (vuln.c:42); saved pc = 0x7fffffffe258
 Saved registers:
  ra at 0x7fffffffe258, fp at 0x7fffffffe250, pc at 0x7fffffffe258
```

1. How do you find the memory address of buf in GDB?
2. Say you run that command and get buf is at address 0x7fffffffe200. Given the info frame output above, how many bytes do you need to write to overwrite the return address completely?