CSCI 476 Computer Security

Exercise for Chapter 3.

(a) We have the following piece of C code:

```
int main(int argc, char *argv[]) {
    char success = 0;
    char password[8];
    strcpy(password, argv[1]);
    if (strcmp(password, "CSCI476") == 0)
      success = 1;
    if (success) {
       *login();
    }
}
```

In the above code, `*login()` is a pointer to the function `login()`. In C, one can declare pointers to functions which means that the call to the function is a memory address that indicates where the executable code of the function lies.

(1) Is this code vulnerable to a buffer-overflow attack with reference to the variables `password[]` and `success`? If yes, describe how an attacker can achieve this and give an ideal ordering of the memory cells (assume that the memory addresses increase from left to right) that correspond to the variables `password[]` and `success` of the code so that this attack can be avoided.

(2) To fix the problem, a security expert suggests to remove the variable `success` and simply use the comparison for login. Does this fix the vulnerability? What kind of new buffer overflow attack can be achieved in a multiuser system where the `login()` function is shared by a lot of users (both malicious and nonmalicious) and many users can try to login at the same time? For this question, assume that the pointer is on the stack.

(3) What is the existing vulnerability when `login()` is not a pointer to the function code but terminates with a `return()` command? Note that the function `strcpy` does not check an array's length.