University of South Florida

CIS 4930 Hands-on Hardware Security

Report

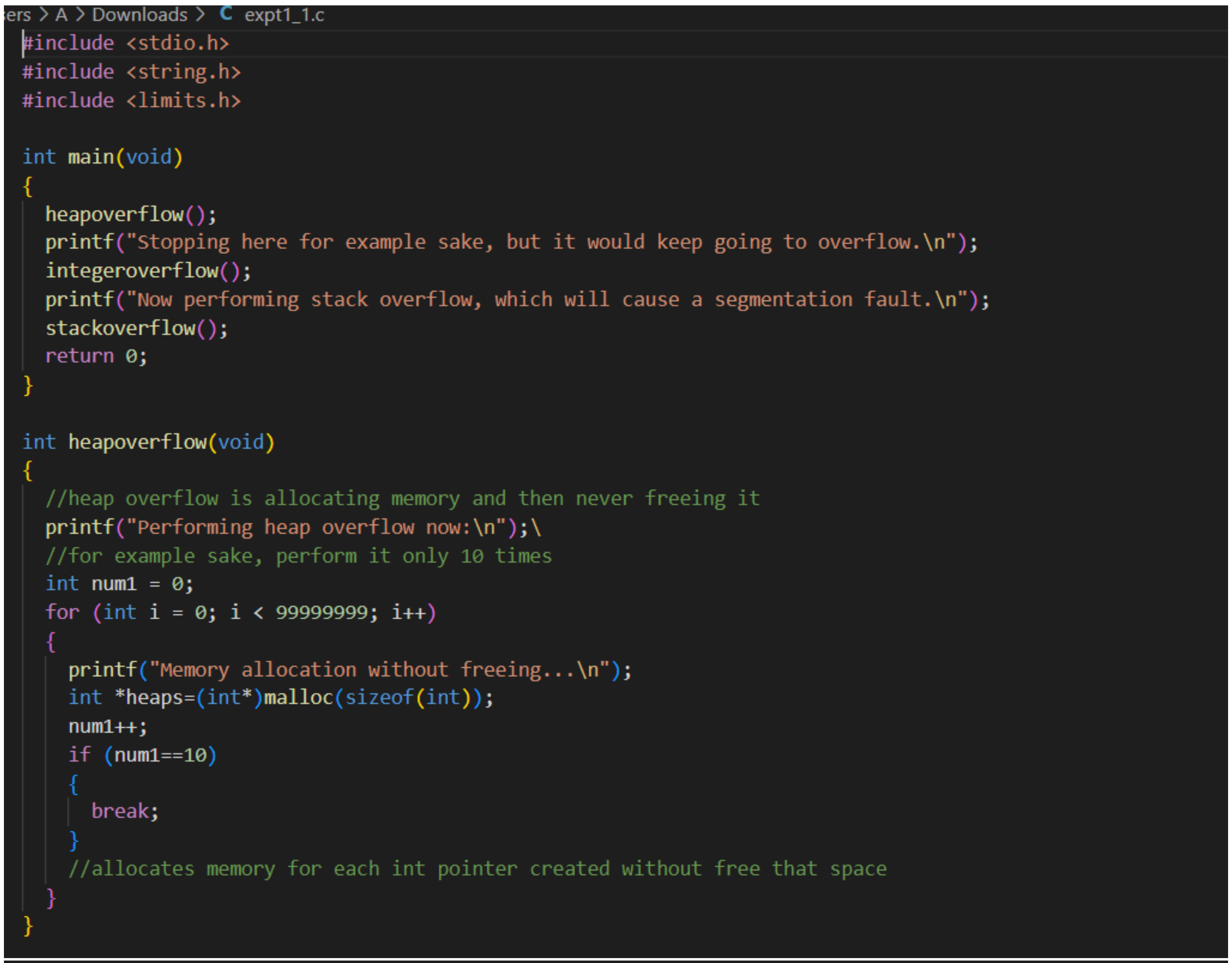
Semester: Summer 2023		
Assignment / Experiment	*Assignment Number:*	02
	Experiment Number:	01
	Date:	< date of experiment > 05/24/2023
Course	*Section:*	002
	Instructors:	Drs. Karam, Katkoori, and Zheng
Report	*Due Date:*	05/31/2023
	Submission Date:	05/28/2023
Team	*Member #1 name:*	Bryce Mccarty
	Member #2 name:	Geonhee Choi

After completing your report (.pdf) and code (.zip), submit them to Canvas. Make sure all the code files are in the same zip file but do not zip the report.

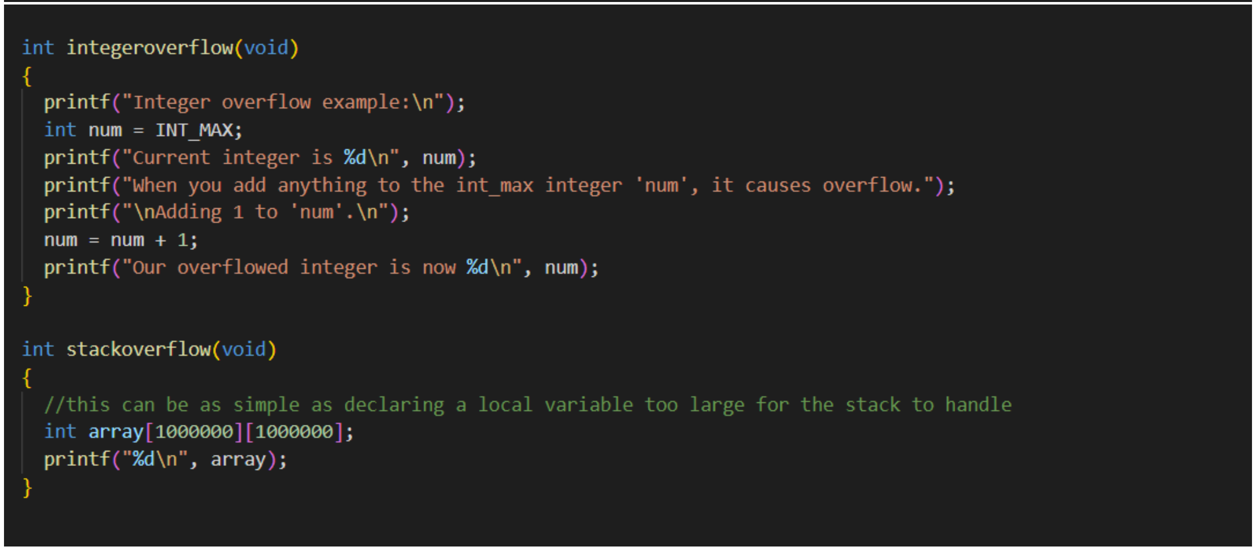
NOTE: please name and organize your code accordingly. Use easy to follow directory structure by parts and use consistent naming convention. **Include readme file.**

PART I: Code and test simple examples

1) [9pts: 3pts each] <u>Submit the code</u> containing the three examples as functions inside one C code.

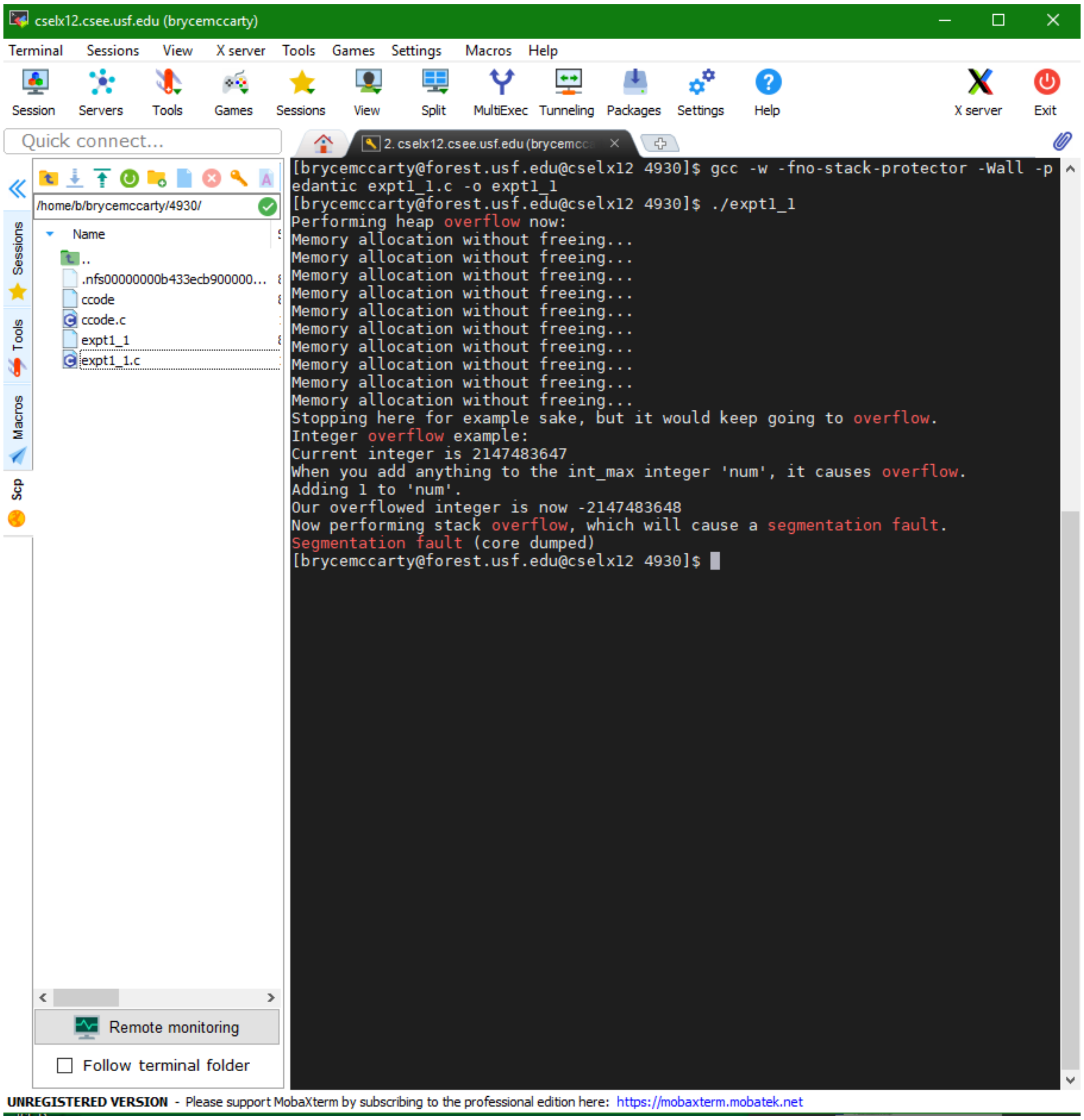


```c
#include <stdio.h>
#include <string.h>
#include <limits.h>

int main(void)
{
    heapoverflow();
    printf("Stopping here for example sake, but it would keep going to overflow.\n");
    integeroverflow();
    printf("Now performing stack overflow, which will cause a segmentation fault.\n");
    stackoverflow();
    return 0;
}

int heapoverflow(void)
{
    //heap overflow is allocating memory and then never freeing it
    printf("Performing heap overflow now:\n");\
    //for example sake, perform it only 10 times
    int num1 = 0;
    for (int i = 0; i < 99999999; i++)
    {
        printf("Memory allocation without freeing...\n");
        int *heaps=(int*)malloc(sizeof(int));
        num1++;
        if (num1==10)
        {
            break;
        }
        //allocates memory for each int pointer created without free that space
    }
}
```



```c
int integeroverflow(void)
{
    printf("Integer overflow example:\n");
    int num = INT_MAX;
    printf("Current integer is %d\n", num);
    printf("When you add anything to the int_max integer 'num', it causes overflow.");
    printf("\nAdding 1 to 'num'.\n");
    num = num + 1;
    printf("Our overflowed integer is now %d\n", num);
}

int stackoverflow(void)
{
    //this can be as simple as declaring a local variable too large for the stack to handle
    int array[1000000][1000000];
    printf("%d\n", array);
}
```

2) [6pts: 2pts each] Take screenshots of the output of the three examples.


cselx12.csee.usf.edu (brycemccarty)
Terminal Sessions View X server Tools Games Settings Macros Help
Session Servers Tools Games Sessions View Split MultiExec Tunneling Packages Settings Help X server Exit
Quick connect...
2. cselx12.csee.usf.edu (brycemcc
/home/b/brycemccarty/4930/
Name
..
.nfs00000000b433ecb900000...
ccode
ccode.c
expt1_1
expt1_1.c
Remote monitoring
Follow terminal folder
Sessions Tools Macros Scp

```
[brycemccarty@forest.usf.edu@cselx12 4930]$ gcc -w -fno-stack-protector -Wall -p
edantic expt1_1.c -o expt1_1
[brycemccarty@forest.usf.edu@cselx12 4930]$ ./expt1_1
Performing heap overflow now:
Memory allocation without freeing...
Memory allocation without freeing...
Memory allocation without freeing...
Memory allocation without freeing...
Memory allocation without freeing...
Memory allocation without freeing...
Memory allocation without freeing...
Memory allocation without freeing...
Memory allocation without freeing...
Memory allocation without freeing...
Stopping here for example sake, but it would keep going to overflow.
Integer overflow example:
Current integer is 2147483647
When you add anything to the int_max integer 'num', it causes overflow.
Adding 1 to 'num'.
Our overflowed integer is now -2147483648
Now performing stack overflow, which will cause a segmentation fault.
Segmentation fault (core dumped)
[brycemccarty@forest.usf.edu@cselx12 4930]$
```

UNREGISTERED VERSION - Please support MobaXterm by subscribing to the professional edition here: https://mobaxterm.mobatek.net

3) [9pts: 3pts each] Suggest ways (at least 1 per each example) to improve it against this attack implementation.

Heap Overflow:
To prevent the heap overflow, we can do bound checking as it is always important to perform boundary checks for buffers and ensure that no more data is written to a buffer than it can handle.

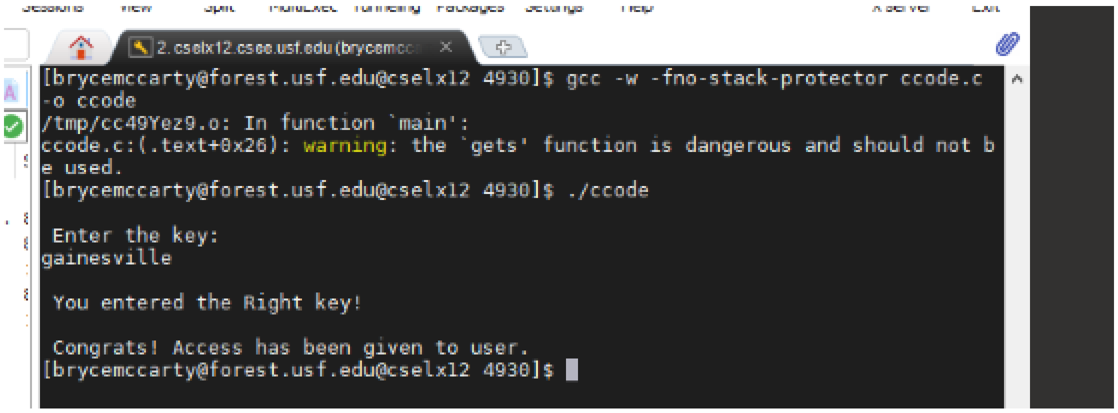
Integer Overflow:
To prevent the integer overflow, we can try safe mathematical operations because we utilized safe math libraries or functions that perform operations in a way that prevents overflow. This is done often by checking the result and throwing an error or exception if an overflow condition is detected.

Stack Overflow:
To prevent stack overflow, we can use a stack canary as a security mechanism used to detect a stack overflow before it can cause an execution flow disruption. It involves placing a small integer, the value of which is randomly chosen at the program start, in memory just before the stack return pointer.

PART II: Buffer Overflow Simple Challenge

1) [5pts] What is output when you enter correct key (key = "Gainesville")? Also, take a screenshot of the output.

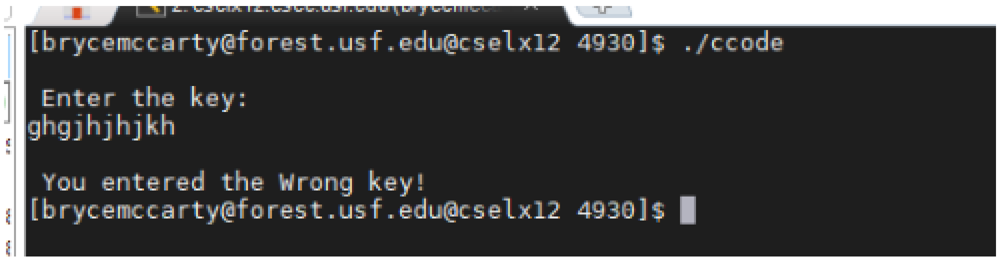
"You entered the Right key!
Congrats! Access has been given to user."



```
[brycemccarty@forest.usf.edu@cselx12 4930]$ gcc -w -fno-stack-protector ccode.c
-o ccode
/tmp/cc49Yez9.o: In function `main':
ccode.c:(.text+0x26): warning: the `gets' function is dangerous and should not b
e used.
[brycemccarty@forest.usf.edu@cselx12 4930]$ ./ccode

 Enter the key:
gainesville

 You entered the Right key!

 Congrats! Access has been given to user.
[brycemccarty@forest.usf.edu@cselx12 4930]$
```

2) [5pts] What is output when you enter a wrong key between 10-12 characters (e.g., key = "ghgjhjhjkh")? Also, take a screenshot of the output.

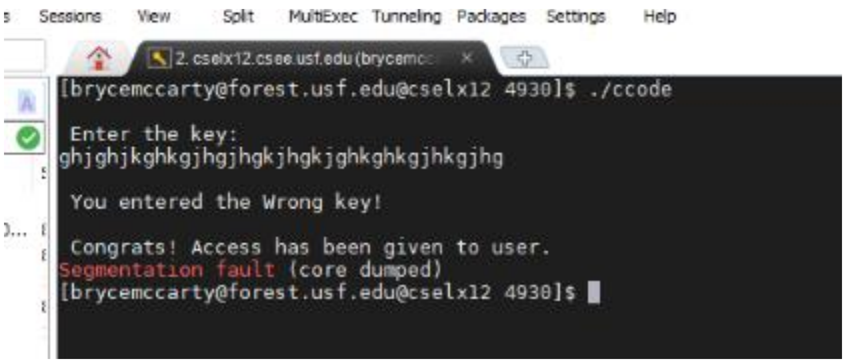
"You entered the Wrong key!"



```
[brycemccarty@forest.usf.edu@cselx12 4930]$ ./ccode

 Enter the key:
ghgjhjhjkh

 You entered the Wrong key!
[brycemccarty@forest.usf.edu@cselx12 4930]$
```

3) [5pts] What is output when you enter a wrong key longer than 12 characters (e.g., key = “ghgjhkjhkjkhkhkjhkjlkjlkjkjlkjlkjlkjkljkjlkjk”)? Also, take a screenshot of the output.

Entered wrong key, but access is then given and then a segmentation fault occurs.



```
[brycemccarty@forest.usf.edu@cselx12 4930]$ ./ccode

 Enter the key:
ghjghjkghkgjhgjhgkjhgkjghkghkgjhkgjhg

 You entered the Wrong key!

 Congrats! Access has been given to user.
Segmentation fault (core dumped)
[brycemccarty@forest.usf.edu@cselx12 4930]$
```

4) [10pts] Write your conclusions. (at least 5 sentences)

This code does check the right password with the strcmp function using the parameter(buffer), but it is not 100% secure. This is because the length of the buffer is only 10 characters and the get function may need to be replaced as fgets or formatted (scanf(“%s”, buffer)). The strcmp function may require additional formatting for security. Furthermore, this code allows the user to access even if they put the wrong password with printf("\nCongrats! Access has been granted to the user.\n"); This is obvsiously not something that should be happening and is a security concern for buffer overflow attacks, since it only happens when many characters are entered.

5) [10pts] Answer this question: Why do all these issues (buffer overflow attacks from part II) happen?

The get function can cause the buffer overflow attack since it does not include a boundary check. And strcmp does not have boundary checking as well. This code has a vulnerability of buffer overflow attacks due to these functions.

6) [10pts] Answer this question: How can you correct them (buffer overflow attacks from part II)?

Using scanf with scanf(“%s”,buffer); to be safe instead of get function. And strcmp needs to check the length of the string before it performs the login.

PART III: Buffer Overflow Malicious Code Injection

SKIP PART III

PART IV: Optional Follow-Up: Advanced Prevention Techniques

1) [10pts Extra Credit] Use "Valgrind" tool to prevent and detect buffer overflow attack by proper memory auditing. Use the user manual in the reference section (page 9). Attach screenshot(s) and submit your code with your solution.