Information Security Homework #1

Question I. Please find the vulnerabilities in the following programs and explain their potential security risks (15 points)

1. ***Answer***

- *Vulnerabilities:* **An array access out-of-bounds vulnerability exists.** The character array **`login[]`**[1] defined in the **`main()`** function has a space length of 512, and the **`fgets()`** function is limited to 512 input characters, which means that **`login[]`** can be up to 512 characters long (including the "\0" character at the end). The character array **`user[]`** defined in the **`verify()`** function has a length of 256, but when the length of the **`login[]`** array is greater than 256, the following core assignment code will have **`i>=256`**, resulting in out-of-bounds access to the user array. Presenting "passwords" here in **plaintext** also carries a high risk of being attacked.

```
for (i = 0; name[i] != '\0'; ++i) {
    user[i] = tolower(name[i]);
}
user[i] = '\0';
```

- *Potential Security Risks:* There are several risks associated with this vulnerability: **(1) Data Corruption**. Data may be inadvertently modified in areas of memory (memory locations above the user array) that should not be modified, during the copying, resulting in data errors; **(2) Program Crash**. The operating system may detect a memory access violation and terminate the program, resulting in a premature termination; **(3) Security Vulnerability**. Hackers may be able to exploit an access violation to execute arbitrary code or read sensitive information to complete a hacking attack.
- *Other Bugs:* The core logic of this code is that when entering "xyzzy", it can be matched and the **`reveal_secret()`** function is executed. However, after testing, we found that even if we type "xyzzy", **`strcmp(user,"xyzzy")` still has a value of 1**, which means that the **`login`** character array obtained from typing "xyzzy" is not equal to that of "xyzzy". The core reason for this is that the **`fgets()`** function itself has a shortcoming that injects the newline character "**\n**" into the character array, causing the **`login`** character array to be "xyzzy\n" after typing "xyzzy" and returns. The usual workaround is to remove the newline character after **`fgets()`**, e.g., using

```
login[strcspn(login, "\n")] = '\0';
```

2. ***Answer***

- *Vulnerabilities:* **An array access out-of-bounds vulnerability exists.** In the **`loop()`** function, the array **`a[]`** is defined directly by assignment, and its length is **fixed at 10**. However, in the for loop code, since the restriction on **`i`** is **`i<11`**, **`i`** may reach 10, and **`a[10]`** will obviously cause access to the array a to go out of bounds!

[1] To present the elements in the code snippet more visibly, code content such as variables and functions are font-emphasized.

- *Potential Security Risks:* First of all, since **a[10]** corresponds to the variable **above the tenth element** of the array **a[]**, i.e., variable **i**, the assignment of **a[10]=0** means that **i** will go back to 0 and start a new loop, which ultimately results in a **dead loop** that keeps outputting "Hello World". Second, because of the array out-of-bounds access vulnerability, the potential problem in the previous problem still exists here.

3. ***Answer***

- *Vulnerabilities:* **An array access out-of-bounds vulnerability exists, too.** Similar to question 1.1, since the definition of the length of the **str[]** character array and the length limit of the **fgets()** function are both 256, the length of the **str[]** array could be up to 256, which is much larger than the length of the **buffer[]** character array defined in the **func()** function. Because the **strcpy()** function does not have an array out-of-bounds detection mechanism, if the length of the **str[]** array is greater than 12, it will be out of bounds when it is assigned.
- *Potential Security Risks:* All three problems mentioned above are also present here. Specifically, out-of-bounds accesses to **buffer[]** can cause the value of the variable **canary** to be rewritten to bypass the **canary==secret** check. In addition, since the **getRandomNumber()** function generates a pseudo-random number, an attacker can try repeatedly to guess the value of the secret and break it.
- *Note:* However, the **strcmp()** function should now have some protection, as shown in Figure 1. Even though gcc compiles with array bounds protection turned off, when the **str[]** array is longer than 12, the program still terminates and does not continue to run.

Question II. Read the following code and answer: Except for " Ft369BfiA", what can be entered to make the program output "Welcome!\n"? (10 points)

Tip: you can enter multiple times

Answer: For the program to output "Welcome!\n" it needs to make the char array **inputs[]** and **true_password[]** equal. The only way to make them equal without inputting "Ft369BfiA" would be to modify **true_password[]**. Since this code contains the **gets()** function, it is possible to alter the value of **true_password[]** using a vulnerability injection. The stack memory map shown in Figure 2. can be obtained after analysis. As long as more than 32 characters are entered into the **gets()** function, the **true_password[]** string can be modified entirely. Based on this idea, there can be two ways to achieve the goal:

- *Input twice:* The first time, the value in **true_password[]** is modified by **gets()** injection, and the second time, the same **inputs[]** are inputted to realize the output of "Welcome!\n", which is shown in Figure 3. Specifically, the first time, the modification of **true_password[]** is accomplished by typing a string containing **34 A**, and the second time, the match is accomplished by typing the modified value "**AA**" directly.

- *Input once:* Modify **`true_password[]`** while making the value in the **`inputs[]`** equal to it. In this case, the **`inputs[]`** need to be explicitly "**`\0`**" terminated, so the following pattern of strings will accomplish this goal:
  "XXXX`(<=16)`**`[NULL]`**`(*N length be 32)`**XXXX**`(Same as the front)`**"**
  It should be noted that the **input method** of "**`[NULL]`**" is related to the operating system, but you cannot input "**`\0`**" or "**`[NULL]`**" directly. Figure 4 presents one solution.

Question III.Read the following code and draw the stack structure of the program when the 4th line of code is executed. (10 points)

Answer: The stack structure of the program is shown below. To show it clearly, I split all of the things into **three** columns:

- the left column represents the functions;
- the middle column represents values in the stack;
- the right column provides explanations for the corresponding values.

functions	**stack frame**	**explaination**
`main()`	SFP	SFP of `main()`.
	`score`	Variable in the `main()` function.
	88	Argument #2 of `GetScores()`.
	82	Argument #1 of `GetScores()`.
	RIP	RIP of `GetScores()`.
`GetScores()`	SFP0	SFP of `GetScores()`.
	170	`total_score`
	`avg_score`	Variable in the `GetScores()` function.
	88	Argument #2 of `GetAvgScores()`.
	82	Argument #1 of `GetAvgScores()`.
	RIP	RIP of `GetAvgScores()`.
`GetAvgScores()`	SFP1	SFP of `GetAvgScores()`.
	85	`avg_score`

Appendix Figures:

Figure 1. Schematic diagram of the operation of question 1.3[2].

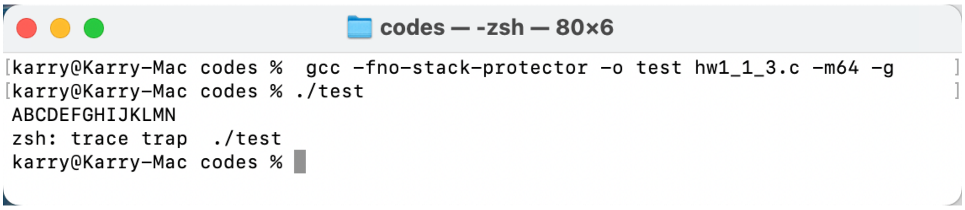


Figure 2. The stack memory map of question II.

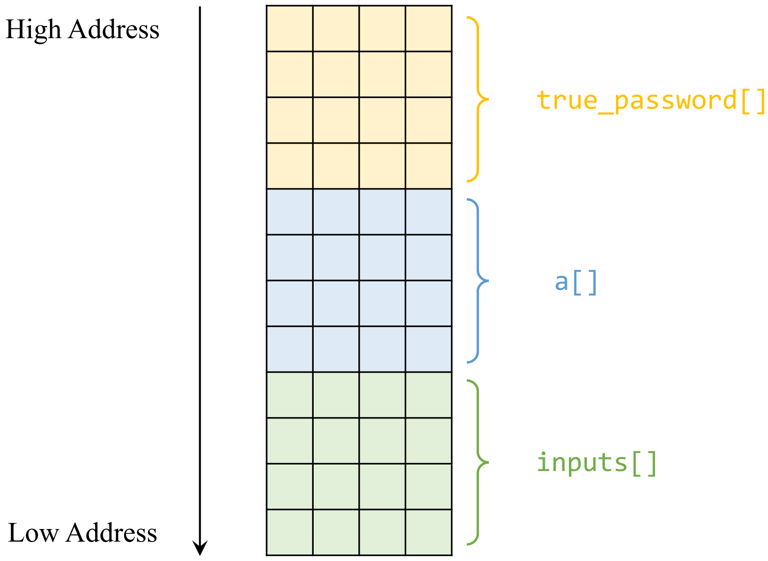


Figure 3. "*Input twice*" solution for question II[3].

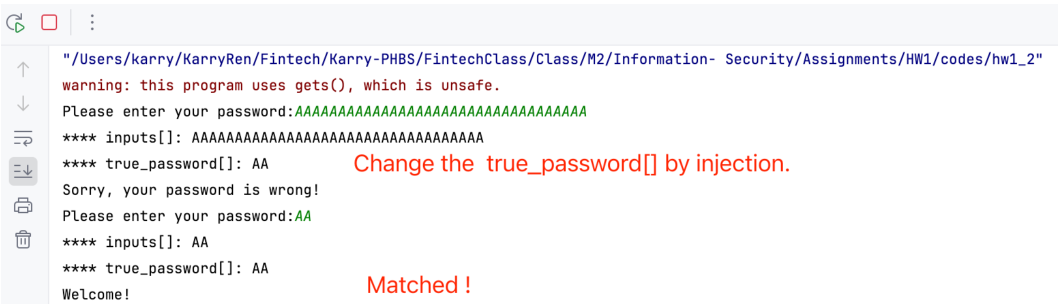
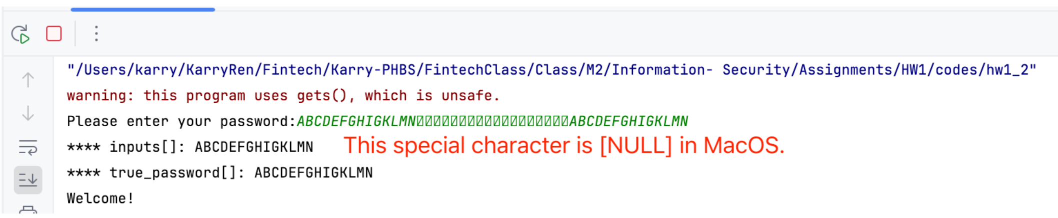


Figure 4. "*Input once*" solution for question II.



[2] All of the runnable codes are released in https://github.com/KarryRen/Information-Security-Project/tree/main/HW_Code/codes_hw1
[3] To make it easier to see, "**** XXXX" is printed as a comment. Figure 4 is the same.