# Information Security Homework#1

**I.** Please find the vulnerabilities in the following programs and explain their potential security risks.

**1.**

**a. The specific logic of the code:**

The program reads a string from standard input and stores it in a char array (login).

The verify( ) function is called to convert the input string to lowercase, then compared with "xyzzy".

If the input matches "xyzzy", the reveal_secret( ) function is called to output the secret information; otherwise, the program terminates with a return value of 1.

**b. Potential risks:**

**Buffer Overflow Risk:** In the verify() function, a char user[256] array is defined, but in the main() function, the fgets() function is used to read login[512] from standard input, which means it can read more than 256 characters into the login array. If the length of the user input exceeds 256 characters, copying this string into the user array in the verify() function will result in a buffer overflow because the user array has only 256 bytes of capacity. This overflow could lead to undefined behavior, including potential code injection attacks. For example, If the return address is correctly overwritten, the attacker can bypass the verify() function's validation logic and directly call the reveal_secret() function. What's more, An attacker could exploit the buffer overflow to overwrite stack memory, thereby gaining control of the program's execution flow, which might lead to arbitrary code execution.

**Logical Vulnerability in Verification:** fgets() retains the newline character (\n) when reading a string, which may cause a mismatch when comparing it to the expected value "xyzzy" in the verify() function. As a result, even if the user input is correct, the extra newline character may prevent a successful match, which could be a logical vulnerability.

**Case Conversion Issue:** The verify() function converts each character of the user input to lowercase and then compares it to "xyzzy". This means that any case-insensitive version of "xyzzy", such as "XYZZY" or other combinations of upper and lower case, will pass the verification. This may reduce the strictness of the validation and be considered a potential security issue.

**Code error:** In the program, user[i]='\0' is used, but since i is a local variable, it has not been defined here. The variable i needs to be defined before the loop in the function.

**2.**

**a. Potential risks:**

**Array Out-of-Bounds Access:** In the loop() function, an integer array 'a' of length ten is defined as int a[] = {1,2,3,4,5,6,7,8,9,10};, meaning the valid indices are from 0 to 9. However, the for loop condition is i < 11, which causes the code to attempt to access a[10], even though the array only has ten elements. Due to the array out-of-bounds access, the program will attempt to access and modify memory beyond the bounds of the array ‘a’. This could overwrite other data or code, leading to abnormal program behavior and potentially allowing an attacker to control the execution flow of the program. An attacker may exploit this out-of-bounds vulnerability to overwrite critical data or function pointers, thereby altering the normal execution logic of the program. For example, by crafting specific data, an attacker could overwrite the return address and hijack the program’s control flow to execute malicious code. This type of vulnerability is commonly known as a buffer overflow, which can lead to arbitrary code execution or program crashes.

**Code error:** The main function must have a return value, so it should be defined as int main and return an integer value. On the other hand, the loop function is void and does not need to return anything, so the return can be removed.

**Logical Flaw:** Because the program attempts to assign the value 0 to a[10], and variables are stored consecutively in memory, the location of a[10] actually stores i. Therefore, i gets reassigned to 0, causing the program to run indefinitely and continuously print “Hello World”.

**3.**

**a. Potential risks:**

**Buffer Overflow:** In the func() function, the strcpy() function is used to copy the input string name to the buffer. The size of the buffer is only 12 bytes, and since strcpy() does not perform boundary checks, if the length of the name exceeds 12 bytes, it will lead to a buffer overflow. The buffer overflow can overwrite other local variables in func(), including the canary variable, thereby bypassing subsequent checks or even overwriting the return address, allowing the attacker to control the program's execution flow.

**Stack Canary Bypass:** The line int canary = secret; attempts to prevent buffer overflow by saving the value of the global variable secret into canary. However, the buffer overflow caused by strcpy() can overwrite the canary, allowing an attacker to bypass the security check if (canary == secret). Moreover, getRandomNumber() may generate a pseudo-random number. If the attacker can guess or infer the value of the secret, they can overwrite both the buffer and canary to make the if condition true, thereby bypassing the security check.

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

**Solution:** Since the get function can read any length of char, we can enter ‘A*48’, the first 16 ‘A’ will fill the inputs array, the next 16 bytes will cover the ‘a’ array, and the last 16 ’A’ will cover the true_password. So the inputs will equal the true_password then it will get the output “Welcome!\n”.

**III.** Read the following code and draw the stack structure of the program when the 4th line of code is executed.

| |
|---|
| SFP(main) |
| Score(=?) |
| Argument#2(88) |
| Argument#1(82) |
| RIP(GetScores in main) |
| SFP0(store the address of SFP) |
| total_score(170) |
| avg_score(=?) |
| physics_score(88) |
| math_score(82) |
| RIP(GetAvgScore in GetScores) |
| SFP1(store the address of SFP0) |
| avg_score(85) |