

Homework 1

Question 1:

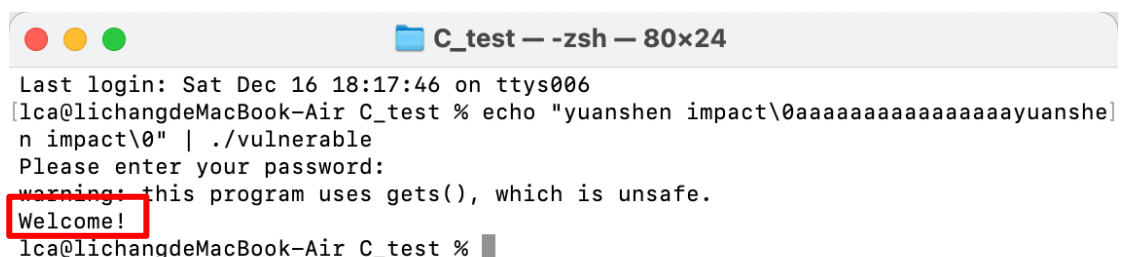
1. There is a risk of array out of bounds in this code. The length of the login array is 512, so users can input up to 512 characters into this array. This array will be passed as a parameter to the verify function, which will copy the value of the login array to the user array. The length of the user array is only 256. Therefore, if the length of the array input by the user is greater than 256, it will overwrite other locations in memory (memory locations above the user array) during the copying process, leading to runtime errors.
2. There is a risk of array out of bounds in this code. Due to the length of array **a** being only 10, only 10 int of space were allocated in memory during creation. But in the following loop, you may access the 11th element of array **a**, which is the variable stored above the 10th element, that is, the number **i**. Therefore, when the loop ends, **i** will be reset to 0 and the loop will continue. So this program will run forever.
3. There is a risk of array out of bounds in this code, too. The length of the array **str** is 256, while the length of the buffer is only 12. The function **strcpy** does not check for array out of bounds, so if the length of the **str** input by the user is greater than 12, it will overwrite the value of the variable **canary**. This will result in different values for variables **canary** and **secret**, leading to program errors. But I think this itself is a protection mechanism that can ensure that it can only run normally when entering 12 characters or less.

Question 2:

We can exploit the vulnerability in the **gets** function to make the inputs and **true_password** variables the same value. Because both variables are arrays of size 16, and the **strcmp** function below compares these two strings until it encounters '\0'. So, we can enter the following text:

"yuanshen impact\0aaaaaaaaaaaaaaaayuanshen impact\0"

The first 16 characters and last 16 characters of this string are the same, with useless 16 characters filled in between. So that the inputs will be overwritten by "yuanshen impact\0", and the **true_password** will also be overwritten by "yuanshen impact\0". So the **strcmp** function will return 1, and we will get output "Welcome!\n". The following picture shows the result of inputting this text.



```

Last login: Sat Dec 16 18:17:46 on ttys006
[ica@lichangdeMacBook-Air C_test % echo "yuanshen impact\0aaaaaaaaaaaaaaaayuanshen impact\0" | ./vulnerable
Please enter your password:
warning: this program uses gets(), which is unsafe.
Welcome!
ica@lichangdeMacBook-Air C_test %

```

Alternatively, it can be done by inputting multiple times. For example, we can input the string "aayuanshen impact" firstly, and

then input the string “yuanshen impact” secondly, we will get the same result, as shown in the following. The principle of program operation is the same as when only one input is required, so it will not be elaborated here.

```

Please enter your password:
warning: this program uses gets(), which is unsafe.
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaayuanshen impact
sorry
Please enter your password:
yuanshen impact
Welcome!
Program ended with exit code: 0

```

Question 3:

The left column represents the functions, the middle column represents values in the stack, while the right column provides explanations for the corresponding values.

	function	stack frame	explanation
2	main	score	variable in the main function
3		88	paramter2 of GetScores
4		82	paramter1 of GetScores
5		RIP	eip of main
6	GetScores	SFP	ebp of main
7		170	total_score
8		avg_score	variable in the GetScores function
9		88	paramter2 of GetAvgScore
0		82	paramter1 of GetAvgScore
1		RIP	eip of GetScores
2	GetAvgScore	SFP	ebp of GetScores
3		85	avg_score
4			