Computer Assignment 2

Winter 2024

Deadline: Feb 11, 2023, 11:59 PM

(Upload it to Gradescope.)

Project Description

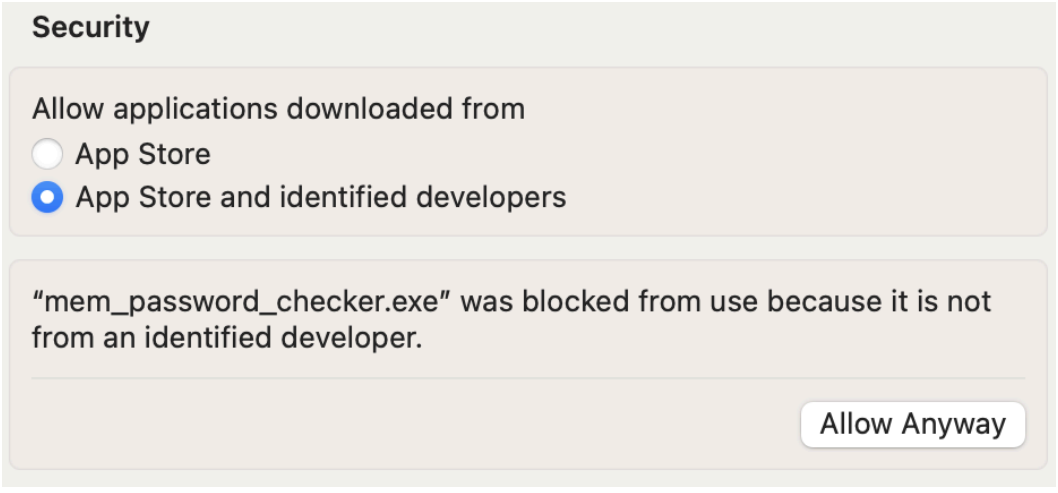
This assignment focuses on understanding the impact of side channels on finding secrets (i.e., a password in this assignment). We will focus on timing side channels which can be either measured directly (using timers) or indirectly using other modalities such as power, EM, etc.

Your assignment includes two parts, and each of these two parts uses a different attack method to obtain the password. You should submit ***a different set of files for each part.*** We have set an autograder on Gradescope for each part. You can test your code as many times as you want.

Setup:

There is one zip file you need to download: Side Channel Zip file. Make sure to download the correct version based on your operating system. After unzipping it, you will find 2 folders: *memHack* and *timeHack* which are used in parts one and two.

For Mac users, when you run the mem_attack_script.py, the OS will block the execution of mem_ctl.exe and mem_password_checker.exe due to security reasons. To solve this problem, when you see that dialog box, go to system settings->privacy&security, scroll down to the bottom, and click “Allow Anyway” for both mem_ctl.exe and mem_password_checker.exe. Then you can run it normally.



Password Checker Implementation:

The password checker uses the following password check method: It will return immediately when it finds a mismatch, and it does not check for equal length at first. We will use this behavior to launch a side-channel attack as described in the following.

```
for(i = 0; i<strlen(correct_pwd); i++){
    if(password[i] != correct_pwd[i]){
        return "Wrong";
    }
    return "Correct"
}
```

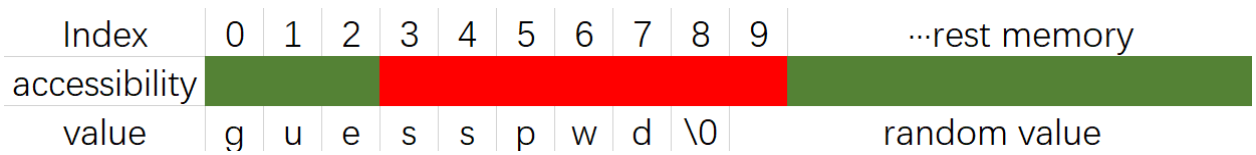
Part 1: Side Channel (Memory Based)

In the memHack folder, you will find three files. The "mem_password_checker.exe" is the target binary file. Our goal is to obtain the correct password from this program. Another file, "mem_ctl.exe," is a memory control program that allows us to restrict access (reading or writing) to a specific range of memory addresses (details later). You don't have to manually run these two programs because we have provided methods in the "mem_attack_script.py" file to execute them for you.

mem_password_checker will use 1 MB of memory to store the password user input, and it will always store the password from the beginning of the memory space (index 0). It has 3 possible results: “Wrong”, “Correct”, and “SEG ERROR”.

mem_ctl can control which part of that 1 MB memory can not be accessed. You can use *setProtectMem(self, start_index, end_index)* method in the provided Python script (mem_attack_script.py) to set a range of memory that can not be accessed starting from start_index (included), ending with end_index (included). By setting that address range, if *mem_passwrod_checker* accesses any address within that range, *mem_password_checker* will be terminated and return a SEG ERROR.

For example, after you set the range to: [3, 9] while calling *mem_passwrod_checker* to check a random password, for example: “guesspwd”, the memory structure should look like as follows: Green color means can be accessed, and red color means can not be accessed.



We use 3 examples to demonstrate the behavior of mem_passwork_checker in this case.

1. The correct password is “goodluck“:
   Step 1: Check the letter ‘g’. Correct, go next.

   Step 2: check letter ’u’. Incorrect, return Wrong.

2. The correct password is “guesscorrect”:
   Step 1: Check letter ’g’. Correct, go next.
   Step 2: Check the letter ‘u’. Correct, go next.
   Step 3: Check the letter ‘e’. Correct, go next.
   Step 4: Check the letter ‘s’. Accessing a not accessible memory, return SEG ERROR.

3. The correct password is “gue”
   Step 1: Check the letter ’g’. Correct, go next.
   Step 2: Check the letter ‘u’. Correct, go next.
   Step 3: Check the letter ‘e’. Correct, the password check passed. Return Correct, the program stops.

The goal in this part is to use this side channel information to find the correct password. You should use *mem_attack_script.py* to write your code that can find the password.

Hints:

1. Think about when we should let the password checker return a SEG ERROR.
2. If we got a SEG ERROR, then what does it mean?
3. Think about how you can use this side channel to repeatedly ask the password checker for another guess.

What to submit:

You should submit your final code on Gradescope. You can try your code on Gradescope as many times as you want before the deadline. We will take the latest submission as your final solution.

You don’t need to submit any report. Please make sure that your code is well-commented.

Part 2: Side Channel (Time Based)

You may have observed that certain operating systems, such as Linux, employ a delay before notifying users about an incorrect password entry. This delay serves the purpose of deterring attackers from obtaining the password by analyzing response times.

In this part, we are exploring what would happen if this feature didn't exist. The idea is to use memory access delay as a side-channel information. The principle of this attack is simple. If the password checker returns false immediately when it gets a character mismatch, then the time it takes to return false depends on how many characters are correct (e.g., a guess with the first three correct characters takes longer to be rejected compared to a completely wrong guess since more memory lines need to be accessed).

To mimic the behavior of a real system, we use a delay function to simulate the real delay when accessing a memory line. To make things more complicated, we also added some noise to this delay simulation. The noise is added to simulate the behavior of a real system where background activities (e.g., interrupts, cache behavior, etc.) have a noticeable effect on the memory delay.

The goal of this part is to use this memory timing side channel to find the correct password. Your code, however, should be able to handle random noise created by the system. You should write your solution in *mem_attack_script.py*.

Hints:

1. You need some form of averaging to remove the noise. However, be careful on short messages, where the delay might be much bigger than the delay itself.
2. Instead of mean, are there other statistics values that can be used? For example, median and mode.

What to submit:

You should submit your final code on Gradescope. Same as part 1, you can try your code on Gradescope as many times as you want before the deadline. We will take the latest submission as your final solution.

You don't need to submit a report for this part, either. Just submit your well-commented code.

Notes:

1. The max length of passwords in part 1 is 20, and in part 2 is 11.
2. All ASCII characters from 33(!) to 126(~) can be used in password.
3. In part 2, you do not need to worry about arrays out of bounds, because we will add long enough spaces after your input. "Space" will never be used in a password.

4. Also, for part 1, if the array is out of bounds and SEG ERROR happens at the same time, the program will return SEG ERROR and terminate. Therefore, arrays out of bounds will not happen.

5. The time-out limit on the server for both parts is 30 minutes, so please make sure that your solution can finish in 30 minutes. Otherwise, Gradescope will terminate your program without grading. (We already tested all cases, and all of them can be solved in 30 minutes.)
6. Use Campuswire to ask questions. Note the due date too!

Good luck!