ECE 188- Secure Computing Systems - UCLA

Computer Assignment 2

Winter 2023

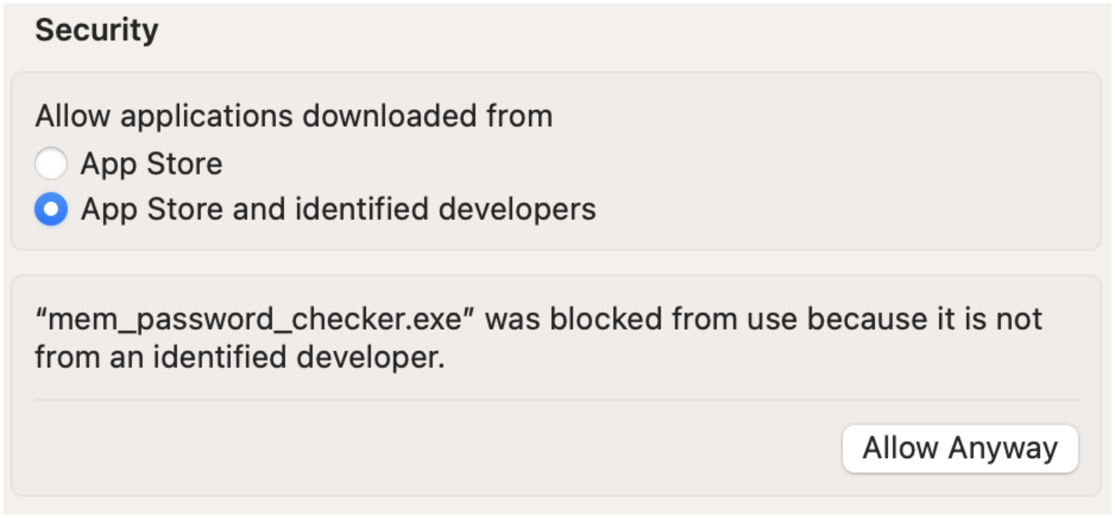
Deadline: Feb 20, 2023, 11:55 PM
(Upload it to Gradescope.)

Project Description

This assignment includes two parts, and each of these two parts uses a different attack method to obtain the password. The purpose of this assignment is to give you some hands on experience exploiting side channels (digital timing side channels, specifically).

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 because Apple cannot check it for malicious software. 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.

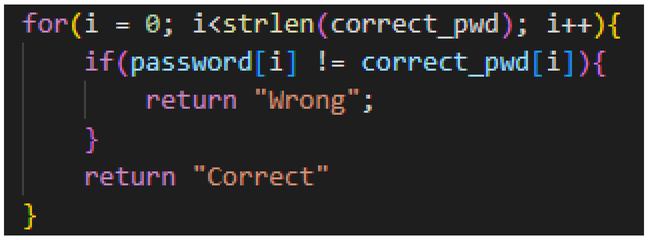

Security

Allow applications downloaded from
App Store [✓ App Store and identified developers]

"mem_password_checker.exe" was blocked from use because it is not from an identified developer.

Password Checker Implementation:

The password checker uses the following password check method (not exactly the same): It will return immediately when it finds a mismatch, and it does not check for equal length at first.


```
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 memHack folder, there are three files. *mem_password_checker.exe* is the target program. We want to get the correct password from it. *mem_ctl.exe* is a memory control program which we can use to set a range of memory addresses that cannot be accessed (read or write). You do not need to run these two programs manually, in mem_attack_script.py we provide methods to let you execute them during python program execution.

mem_password_checker will use 1MB 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 1MB 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 (inlcuded), ending with end_index (inlcuded). 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 calingl *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.

Index	0	1	2	3	4	5	6	7	8	9	…rest memory
accessibility	🟩	🟩	🟩	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟩
value	g	u	e	s	s	p	w	d	\0		random value

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

1. The correct password is “goodluck“:
   Step 1: check 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 letter ‘u’. Correct, go next.
   Step 3: check letter ‘e’. Correct, go next.
   Step 4: check letter ‘s’. Accessing a not accessible memory, return SEG ERROR.

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

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 to use this side channel to iteratively query the password checker with another guess?

You should submit your final code on Gradescope. Same as CA1, 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.

Part 2: Side Channel (Time Based)

You might have noticed that in some operating systems (like Linux), when you enter the wrong password, the system waits a while before alerting you. The purpose of this behavior is to prevent attackers from getting the password based on the response time.

In this part, we are exploring what would happen if this feature doesn'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 depdens on how many characters are correct (e.g., a guess with 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 add 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.) has 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 amount of 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, mode.

You should submit your final code on Gradescope. Same as CA1, 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.

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 password.
4. Also, for part 1, if array out of bounds and SEG ERROR happens at same time, the program will return SEG ERROR and terminate. Therefore, arrays out of bounds will not happen.
5. The time limit for both parts is 30 minutes, so please make sure that your solution can finish in 30 minutes. Otherwise, gradescope will terminate your program with out grading. (We already tested all cases, all of them can be solved in 30 minutes.)

What to Submit:

1. mem_attack_script.py for part1
2. time_attack_script.py for part2

Also, please add explanation at the end of the python files to explain your solution (you can add comment to each line and use a short description at the end). Submissions without enough explanation will only lose some points.