Challenge 4

Filip Strömbäck, Ahmed Rezine

March 6, 2024

Instructions

You have to do research on the parts you do not understand. When you solve problems, do not hesitate to:

- State your starting point. What do you know from start? Why is it important?
- State your goal. What do you need to know to reach it?
- State your assumptions. What are you taking for granted or assuming?
- Show step by step how you can go from what you have to what you want.

You will get one credit for each presentation you claim prepared. You may (at random) be selected to present any of the solutions you prepared. When you are selected for presentation:

- For each step in your solution, explain what you were thinking. How did you come up with this? What gave you the clues?
- Practice so you are ready to present more than less from memory (you can still have a note for reference, but you should not need to look at it much). **Aim for 10 minutes.**
- You are not required to have a correct solution to get credit, but **you are required to solve all parts of a problem, to make a serious attempt and to "believe" in your solution**.
- If you are not prepared you loose all credit for the seminar in question.

As audience you should think of how difficult it is to clearly present a solution. Be humble and supportive. You may put forward constructive criticism of the presented solution. Compare to your solution; I did it another way, what will be the difference? You are of course welcome to take notes.

Problem 1

Serverus Corporation runs systems where users (e.g., `user1`,...) can run their own services. Each user service (e.g., `serviceA`) runs as that limited user (e.g., `user1`) and produces an audit log file in `/var/log` (e.g., `/var/log/serviceA_user1.log`). The log file is only accessible to that user:

```
$ ls -l /var/log/serviceA_user1.log
-rw------- 1 user1 user1 28 feb 10 10:49 serviceA_user1.log
```

Each user also has other private files. You are administrator and want to hire an employee to monitor the log files for possible misuse, mis-configurations and security issues.

(a) Explain how you can create a group to let your employee view the logs (using `chgrp`, etc.)

(b) Explain how you can use ACL's to let your employee view the logs (using `setfacl`, etc.)

(c) Explain how you can use a setuid program to let your employee view the logs.

(d) Evaluate how secure each of your solution is, according to principle of least privilege and need-to-know. Consider the potential scope of negative effects when an attacker gain access to the group, an ACL entry, or the security domain of the setuid program.

(e) Consider your system have services to rotate log files and a package manager to keep services up-to-date (both may reset file owner and rights to some default state). Which solution is less intrusive and more likely to still work after logs are rotated and system updated? Do you see any conflict with security?

Note: You are encouraged to check the usage of the involved commands (e.g., `chgrp`, `chmod`, `setfacl`, `getfacl`, etc.). Just explain why they are used, not the syntax.

Problem 2

1. What is the difference between access lists and capability lists? Give an example of how each of them can be used for access control.
2. Give an advantage of each of the two access control approaches.

Problem 3

For each of the following items, state and motivate (in less than 3 sentences) if it is a representative of a capability-based approach or of a ACL-based approach to protection:

1. There is a list of personally invited guests to the Nobel prize banquet.
2. Access to some buildings in Campus Valla after 17:00 requires using a personal access card with a content that matches the list of students and employees at LiU.
3. Cars have to have visible parking tickets. The parking tickets can be obtained by using cash to pay the parking fee.
4. Individual room keys can be used to open a student dormitory or corridor.

Problem 4

The function `fseek(fd, offset, SEEK_START)` can be used to set the file-position indicator for file descriptor `fd`. In particular, such a usage of `fseek` adds `offset` bytes relative to the start of the file and assigns the resulting value to the file position indicator. With this semantics of `fseek`, answer the following questions:

(a) Your friend claims that, without sufficient care in the implementation of the file system code in the operating system (OS), a malicious user can misuse `fseek`. In particular, they can retrieve confidential files by using `fseek` to seek to some well thought-out position and then use `fread` to read from there. Do you think, your friend is correct? If your answer is *yes*, demonstrate an appropriate scenario. If your answer is *no*, provide appropriate justification.

(b) If your friend is right, which file allocation method (contiguous, linked, indexed) do you think is more likely to be vulnerable to such an attack? How do you think OS can prevent such misuse of `fseek`?

Problem 5

The function `fread(fd, buffer, count)` can be used to read `count` bytes from file descriptor `fd` to DRAM memory area indicated by `buffer`.

Similarly, `fwrite(fd, buffer, count)` can be used to write `count` bytes from DRAM memory area indicated by `buffer` to file descriptor `fd`.

(a) Your friend claims that, without sufficient protection by the operating system (OS), a malicious user can misuse `fread` and `fwrite`. In particular, they can read and write arbitrary DRAM memory. Do you think, your friend is correct? If your answer is *yes*, demonstrate an appropriate scenario. If your answer is *no*, provide appropriate justification.

(b) If your answer was *yes* for the previous question, explain how the operating system can perform sufficient protection.

Problem 6

UNIX provides a mechanism called *signals* to allow a limited form of inter-process communication. In particular, a signal can be used to send an asynchronous message to a process. The system call `kill` can be used to send a specific signal to a specific process. Consider the following instances of the system call `kill`:

- `kill(pid, SIGSTP)`: Send the signal `SIGSTP` to the process `pid`. This results in suspending the execution of the process `pid`.
- `kill(pid, SIGCONT)`: Send the signal `SIGCONT` to the process `pid`. This results in resuming the execution of the process `pid`, if it was stopped.

Depending on this interpretation of mentioned signals, answer the following questions:

(a) If the operating system implementation of `kill` allows any user to send signals to any other process a malicious or careless user could misuse the signals. Suggest simple rules the operating system implementation can follow to allow or deny signals

depending on the domain of the calling process and the domain of the receiving process.

(b) Explain how you can use capabilities such as those explained in `http://man7.org/linux/man-pages/man7/capabilities.7.html` in order to allow a process to bypass the restrictions in (a) while striving for the principle of least privilege.

Problem 7

Consider a system in a office building, with a UNIX operating system, system calls, system services, users with passwords, network with ssh access and a disk with secret files protected by UNIX file security, access control lists and setuid programs.

(a) What do you believe is the weakest point(s) to gain (unauthorized) access to the system and to the files? Motivate. (There is no unique answer).

(b) What can you do to strengthen the weakest points?

(c) Assume you gain access to the hashed but not salted password file of this system. Explain first how to perform a dictionary attack and then how you can design the password hash in order to make this kind of attack considerably slower.

(d) Insufficient array limit checks can be utilized for buffer overflow attacks. Assume your processor support paging and segmentation. Can you think of any hardware and OS additions to thwart the most common type of buffer overflow attack?

Problem 8

The 1984 Turing award lecture "Reflections on trusting trust" by Ken Thompson [1] is an eye-opener to the trust we put in software we use. The principles from said paper is used in this question.

Consider the source code of the very first compiler.

The devil modify the source code of the compiler to recognize linked list implementations, and if a linked list program is found, the compiler code add instructions to the binary version of the list program making it issue random segmentation faults.

Any list programs compiled henceforth will randomly crash. But, obviously, anyone inspecting the compiler code would recognize the addition for what it is, or at least as something very strange, worthy of further investigation.

But the devil is not done. He continue by modifying the source code of the compiler to recognize the source code of itself, and if compiler code is found, the compiler will add to it's binary version

1. instructions to recognize the compiler source code and if so add both of these additions to it's binary version.
2. instructions to recognize linked list implementations and if so add segmentation faulting instructions to the binary version of the list program.

The devil now compiles this modified compiler.

Finally, the devil remove his additions from the compiler source code and release both the source and the binary to the world. The world will, after all, need the binary compiler to compile the compiler source code.

People check the source code of the compiler and determine it to be correct. People compile the source code with the binary and determine that they get an identical binary version of the compiler. People are very happy with their new compiler software, since now they do not have to use assembler anymore.

(a) Will the last compiled version of the compiler be free of the devils additions? Explain.

(b) Will any version be free of the devils additions? Explain.

(c) What does it take to detect and remove the devils additions?

For those interested, Russ Cox asked Ken Thompson for the source code of the backdoor. They received it and wrote a quite detailed explanation of how it works. The code is surprisingly short and quite easy to follow: `https://research.swtch.com/nih`

There is also a short story about what could happen if this idea is extended to the operating system and other devices: `https://www.teamten.com/lawrence/writings/coding-machines/`

References

[1] Thompson, Ken. *Reflections on trusting trust*. Communications of the ACM 27.8 (1984): 761–763.