COMPUTER SECURITY, 2023/2024

SECOND LAB EXERCISE: AUTHENTICATION USING PASSWORDS

02. 04. 2022

INTRODUCTION

Passwords are the most common and simplest method of authentication - both for implementation and for use. However, passwords also have a lot of vulnerabilities that threats can exploit. As part of this laboratory exercise, you should write a simple login application that uses passwords, taking into account threats and vulnerabilities.

FUNCTIONAL REQUIREMENTS

As part of the exercise, it is necessary to implement two tools, one that enables the management of passwords and usernames and is intended for administrators, and the other that serves to log in users. The username management tool (let it be called `usermgmt`) should have the following funcionalities:

1. Adding a new username (operation *add*).
2. Changing the password of an existing username (operation *passwd*).
3. Forcing a password change when user logs in (operation *forcepass*).
4. Removing an existing username (operation *del*).

The second tool is for logging into the system (let it be called `login`). This tool should have the following functionalities:

1. Entering the username and password, where the password must not be visible during the entry.
2. Forcing password change after a successful login - if requested by an administrator.
3. In the case of successful login, some command has to be started (e.g. bash)

The tools are used from the command line, and the data necessary for tools to operate correctly should be written to a file whose format is up to you. The tool used for logging in should print whether the login was successful or not before finishing.

Interaction with the tools may look like this:

```
$ ./usermgmt add sgros
Password:
Repeat Password:
User add failed. Password mismatch.

$ ./usermgmt add sgros
Password:
Repeat Password:
User sgros successfully added.

$ ./usermgmt passwd sgros
Password:
Repeat Password:
```

```
Password change failed. Password mismatch.

$ ./usermgmt passwd sgros
Password:
Repeat Password:
Password change successful.

$ ./usermgmt forcepass sgros
User will be requested to change password on next login.

$ ./usermgmt del sgros
User successfully removed.

$ ./login sgros
Password:
bash$

$ ./login sgros
Password:
New password:
Repeat new password:
bash$

$ ./login sgros
Password:
Username or password incorrect.
Password:
Username or password incorrect.
Password:
Username or password incorrect.
$
```

Note that the tool does not provide information on what exactly is the problem when logging in, a non-existent username or a wrong password. Think about why that is the case.

For the sake of simplicity, we can assume that the address and password will consist of a maximum of 256 characters and that all characters will be printable ASCII characters (ASCII codes from 33 to 126 inclusive), so it is not necessary to support UNICODE characters.

SECURITY REQUIREMENTS

In order to correctly predict and implement protection mechanisms assume the following threat model:

> The login command is launched in such a way that only legitimate users will use it, i.e. attacker cannot access it. All legitimate users are absolutely trusted not to abuse their authority, and the usermgmt command can only be run by an administrator anyway. There is a possibility for file with passwords to be stolen, i.e. fall into the hands of attackers, where attackers have very advanced methods of guessing passwords and access to significant amounts of computer resources.

Password threats and vulnerabilities are outlined in the lecture, followed by mechanisms used to remove or mitigate vulnerabilities. Use that material to help you design proper protection mechanisms.

TASKS

1. Study the lecture materials.
2. Design and implement password management and login tools that meet the functional requirements described above and contain the necessary protection mechanisms.
3. Create a tool using a programming language C/C++/Java/Python.

For those who want more:

1. Implement OTP in addition to password authentication using HOTP algorithm.
2. Write an additional tool that will try to guess passwords.
3. Try using one of the password guessing tools mentioned in the lecture.

IMPLEMENTATION

You can solve the laboratory exercise using a programming language C/C++/Java/Python. We recommend higher-level programming languages, such as Java or Python. The solution must be runnable using standard tools and compilers on a Linux operating system.

SUBMISSION DEADLINES

It is necessary to submit an archive containing:

- The source code of your solution.
- Compile and run instructions, ideally in the form of *a shell* script that will compile your solution on startup and then run it to demonstrate all functionality.
- **A text file containing a description of your system and for each protection described in the lectures, state whether you have implemented it or not and justify your decision.**

The deadline for submitting the laboratory exercise is **April 14th, 2024. at 23:59** .

If, for any reason, students do not manage to solve the laboratory exercise by the given deadline, they can still hand it in until **05.05.2023. at 23:59**. Correct solutions sent by that deadline do not earn points but enable students to meet the minimum requirements and pass the course.

In case of problems or doubts during the preparation of the exercise, please contact the teaching staff in a timely manner via the course mailing list srs@fer.hr (exclusively using the fer.hr email address) .

Important: It is allowed and desirable to discuss possible approaches to solving the exercise between students. However, students must do the laboratory exercise itself. The teaching staff will check the similarity of submitted solutions, and we will report behavior that is not in accordance with the FER Student Code of Conduct to the Committee for Disciplinary Responsibility of Students and determine additional sanctions within the subject.