CAA 2022

Lab #2

02-05-2022

1 Introduction

The goal of this lab is to implement a **multi-user password manager** that respects some security requirements.

- Please provide also your **code** which will count as a **lab grade**.
- Provide a small **guideline** on how to use your program.
- You do not need to do a GUI for your program. A command-line version is sufficient.
- You do not need to use any complicated database system. Simple text files are enough for this lab.
- You do not need to implement networking. Having everything local is enough.
- The programming language is free although Rust or C/C++ is recommended as it allows to erase data from memory (which is hard to do in Java).
- Please provide a **report** describing how you modelled your security. In particular explain how you secured your program to meet the required security properties shown below. The report grade will be a **course grade**. Write also in your report the **impact of the chosen programming language**.
- **Bonus points** will be given for any cool bonus functionality.

2 Password Managers

Password managers are software used to manage the passwords of different websites/programs. They are unlocked using a **master password** which is the only password the user has to remember. He will use this password to **login** into the software.

Our password manager can be in two different **states**:

1. **Not running**: the state in which the password manager is before log-in.
2. **Unlocked**: once the user logged into the password manager and entered his master password, the password manager is in the unlocked state for that user. To recover passwords, the user does not have to type his master password anymore.

In each state, we have different security requirements.

3 Not Running State

- One should **not** be able to recover any password (including the master password) without knowing the master password in this state.
- **Bruteforcing** the master password should be difficult even if some passwords of the database are known. Being able to bruteforce trivial passwords is ok (123456 is trivial. HouseWithHorse is not).

4 Unlocked State

In this state, the user can freely ask for passwords corresponding to websites.

- It should **not** be possible to extract the **master password** from the memory.
- **Unaccessed passwords** should not be in clear in the memory.

5 Password Sharing

Finally, a user should be able to **share** his password with another user of the software. For this, he simply has to type (or select) an other username and a label and the password should be added (in a secure way) in the other user's account. Note that while sharing, the password should **remain secure** and not leak or stay in clear somewhere.

6 Your Implementation

Your implementation should include the following functionalities:

- A way to **recover** the password of a website. Displaying the password in the terminal is fine. Putting it directly in the clipboard is a plus.
- A way to **add** a new password in the database.
- A way to **change** the master password.
- A way to **share** a password with another user.

7 Your Report

Your report should explain **in details** how you will model the **security** of this software and how you perform the various operations. In particular, algorithm names, key sizes, modes of operations, . . . have to be given and be consistent with each other.