VORMUND

Team 6

Seth Cook
Margarita Lee Li
Samuel Marshall
Adam Rice
Mitchell Stendal
John Vall

Problem Statement

Many people have poor password practices. This can be manifested through the use of similar passwords for multiple accounts, not using secure enough passwords to make memorization easier, and storing them in insecure locations. This practice can have consequences of varying severity, from harmless pranks by friends to devastating thefts of financial information or assets.

Background Information

Passwords are undoubtedly the dominant method of authentication on the Internet. Unfortunately, they are also prone to serious vulnerabilities. While they are commonly used, they are often the only barrier between the user and the account. There are many websites that do not employ secondary checks when logging in, such as security questions (e.g., “what was the name of your first pet?”), text message code verification, and even biometrics. This would leave an attacker just a password away from potentially destroying your work or stealing thousands of dollars from a bank account.

It has been established that passwords are extremely important, but despite the risks, many users continue to have poor password practices. One common habit involves users storing their passwords in plain text or in a spreadsheet, often labeled “Usernames & Passwords” for convenient access. These documents are goldmines for attackers as they allow extremely easy access once a backdoor or trojan is set in place.

The need for a secure password storage system has been addressed: users need to be encouraged to create long, secure passwords, but they may not be willing to remember them, thus want to save them for reference later. We will create a cross-platform, easy-to-use desktop application for any computer user hoping to take advantage of secure password storing techniques.

Environments and System Models

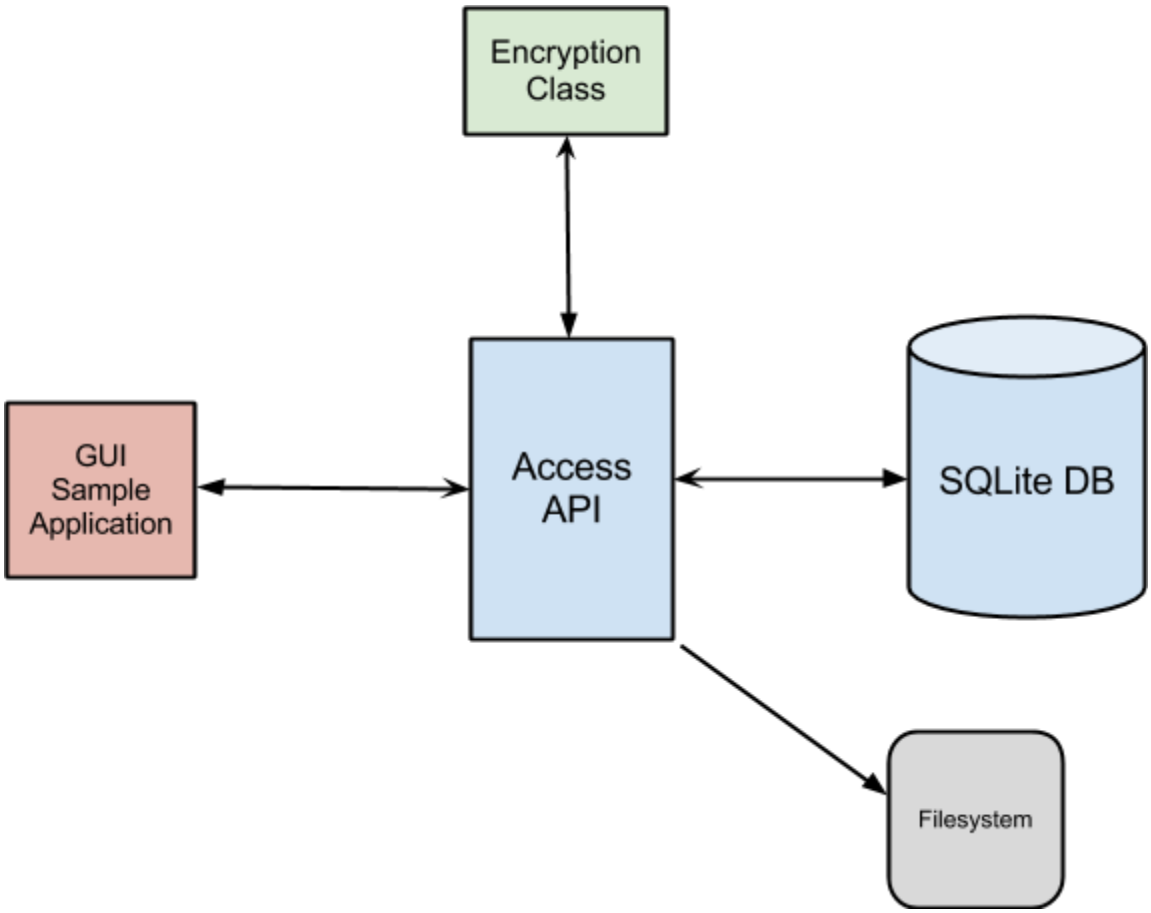
To tackle this problem, we chose the Java SDK because it will allow us to develop a cross-platform application to reach a maximum-size target audience as possible. Our solution stack comes with four separate parts:

- **SQLite database:** our database will hold basic user information and metadata about the records stored, like usernames/passwords, bank account information, etc.; these records need not be encrypted. They will also contain information about where the records are stored on the filesystem. We chose SQLite because it is lightweight, easy to install, and is widely used and trusted around the world.
- **Access API:** our API will allow developers to store and retrieve records with one Java class. Methods will be included for creating, updating, and deleting records as well as

changing master passwords and even user preferences. This API will deal with the hashing and encryption methods, leaving the consumer to not worry about handling it themselves.

- **Encryption scheme and methods:** we intend to create our own static class with wrapper methods for encryption schemes provided by the Java SDK. This will allow our API and others to use this as a helpful utility class.
- **GUI Demonstration:** we will create a graphical program using Java's own Swing and AWT libraries that consumes and demonstrates our API. This will be bundled with the .jar archive and will serve as the standard application we intend our users to use.

A graphic demonstrating the relationship between this four:



Functional Requirements

- **Relationship between database and access API**
    - Methods must be able to take input from consumer, and produce valid SQL queries to make desired changes in database, particularly creating, updating, reading, and deleting user records.
    - API must be able to consume data dump from database and parse it into an easy-to-use fashion for consumer. This might involve creating another internal class to serialize this data.
- **Encryption schemes**
    - Encryption class must be able to provide a two-way encryption scheme guarded by one master password.
        - User records (e.g., passwords, bank info) must be encrypted on file system to prevent attackers on compromised systems from making use of this data
        - These must be recoverable for our system to be usable
    - There is some sort of a master password to access this entire system.
- **GUI will demonstrate all features of API**
    - Lets user to authenticate him/herself and change settings
    - Provides interface to manage username/password records and bank accounts
    - Allows user to restart database service instance in case of failure

Non-Functional Requirements

- **Reliability**
    - Users should not experience problems due to database connectivity, API flaws
- **Usability**
    - Installation should be as easy as running a script or double-clicking an executable. There shouldn't be any complex configuration steps, like manually editing the PATH. Uninstalling our application should be just as easy.
    - When using our application the first time, the user should not have to guess or spend more than a few seconds finding a feature
    - The UI should be fluid, no tedious multiple windows or switching back and forth
    - User should be able to click immediately on a record to copy to clipboard
- **Security**
    - Should not leave attackers finding an easily exploitable storage system like the default password managers of Firefox and Chrome
    - Users must acknowledge that there is no absolutely secure system, and that if an attacker has knowledge of their master password, then the protection our system offers becomes useless

Use Cases

Before doing any of the following bullet points below, the user must:

1. Launch the .jar application and make sure the SQLite service is running
    a. Can launch the script if unsure and it will take care of both of these
2. Provide master password for authentication

- **Storing a new username/password**
    a. Click on the username/passwords panel
    b. Click “Create New”
    c. Modal/popup appears asking for this information
    d. Clicks “Ok”, record stored, view updated
- **Updating/deleting a record**
    a. Click on username/passwords panel
    b. Click on individual record by scrolling through or searching for it
    c. Can edit data in-line or modal/popup will appear
    d. Can apply/cancel changes
- **“Self destruct” (clear all records)**
    a. Click on *Settings* on main panel
    b. Click on red *Delete All Data* button
    c. Confirm
    d. Confirm again
    e. Records are removed, view updated and cleared
- **Changing a master password**
    a. Click on *Settings* on main panel
    b. Click on *Change Master Password*, view updated
    c. Provide old password, then new one, then confirm new one
    d. Click *Ok*
    e. Master password updated