Distributed Systems – Assignment 4

Cédric Bürke
ETH ID 08-918-120
cbuerke@student.ethz.ch

Kevin Kipfer
ETH ID 09-929-993
kkipfer@student.ethz.ch

Marc Gähwiler
ETH ID 10-927-796
gamarc@student.ethz.ch

ABSTRACT

For the final project we decided to tackle the task to design and develop a completely open sourced password managing solution. Because the first part of this course mainly focused on Android development, we decided to begin our journey by developing a simple backend server application and then concentrating on implementing a basic Android application, that allows to use the server we previously created.

All in all we are content with ourselves, as we think that we succeeded in developing a basic proof-of-concept application, that is already usable. Still, we want to keep the project alive and therefore mentioned a few ideas how the project can be improved in the future.

1. PROBLEM STATEMENT AND REQUIREMENTS

Everybody that uses a few different sites that use basic username/password authentication knows the problem: even though it is clear, that you should not use the same password on two different websites, almost everybody is too lazy or forgetful to use a different password on each website he uses.

To counter this problem and make it easier for everybody to use an unique password for every site, so called password managers exist. In the following paragraphs we list some examples of existing password managers and why we do not consider them to be a satisfying solution.

1.1 KeePass/KeePassX

KeePass is a free open source password manager, which helps you to manage your passwords in a secure way. You can put all your passwords in one database, which is locked with one master key or a key file. So you only have to remember one single master password or select the key file to unlock the whole database. The databases are encrypted using the best and most secure encryption algorithms currently known (AES and Twofish). [3]

1.1.1 Pros

- Free
- Open-Source
- Using tested and known encryption algorithms
- Available on a huge number of platforms
- Actively developed

1.1.2 Cons

- Two different main source trees (KeePass 2 and KeePass X)
- Passwords are stored in a file, thus to keep the password synchronized over more than one device, the password file itself has to be synchronized, which can be quite difficult
- Browser integration is complicated

1.2 LastPass

LastPass Password Manager is a freemium password management service developed by LastPass. It is available as a plugin for Internet Explorer, Mozilla Firefox, Google Chrome, Opera, and Safari. There is also a LastPass Password Manager bookmarklet for other browsers.[4] [5]

1.2.1 Pros

- Web application is free
- As far as we know it is using tested and known encryption algorithms
- Web application with plugins for all mayor browsers
- Android and iPhone application
- All passwords are distributed to all clients

1.2.2 Cons

- Mobile applications are not available for free users
- Closed source
- For profit company that could potentially have access to all of your passwords and therefore all your accounts
- US company

1.3 Other solutions

While we researched this topic we came across a few other possible solutions, but they all fell into one of the two categories we mentioned above: either they were offered by for profit companies or the were free or even open source, but there was no possibility to distribute the passwords to all possible clients (like a desktop computer, a laptop, an office computer, a smartphone and a tablet) a user owns.

2. REQUIREMENTS

Because of the reasons mentioned in the last section, we decided to develop a service that satisfies a larger number of our needs which include but are not limited to:

- Open source
- Using well-known and -tested cryptographic algorithms
- Available at least as a web application, a browser plugin and mobile applications for Android and iOS
- Possibility to distribute the stored passwords to all (or at least a majority) of all devices a user uses
- Not controlled by a company

To achieve this goal we decided to begin with developing a simple Android application, that offers a user to synchronize a list of passwords with a server. It should be possible for every user to host his own password database server if he has some basic system administrator knowledge to host a server and run a basic python application on it.

3. ARCHITECTURE

As we knew that it is not really viable to implement all our planned features, we decided to stay simple and just implement the most important features. This includes basic user management (login by a username and a password), password encryption and storage (encrypted with the master password).

3.1 Backend

For developing the backend we used

- Python [6]
- Flask [8] a python web microframework
- SQLAlchemy [7] a python ORM and the flask version Flask-SQLAlchemy [2]
- Flask-Restful [1] to provide a simple REST interface to the Android application

We currently provide the following REST endpoints to the Android application to communicate with the backend:

- TODO: add possbile requests from the readme

Basically at the moment the web application is just a simple wrapper around a SQLite database. Still, this was enough for us to develop the application.

3.2 Communication

TODO: Some source code and a basic description about AsyncTask and how HTTP requests were done

3.3 Interface

TODO: Describe the interface, maybe a state diagram and of course a few screenshots

4. IMPLEMENTATION

4.1 TODO: More implementation details

4.2 Cryptography

As we all know that cryptography is hard and a pain, we used the standard Java javax.crypto cryptography library to implement hashing, encryption and decryption.

We decided to use the well known AES symetric encryption cipher in CBC mode and used PKCS5 as the padding scheme:

```
public Encryption(Server server) {
        ...
        cipher = Cipher.getInstance("AES/CBC/
            PKCS5Padding");
        ...
    }

    ...

    public byte[] encrypt(String toEncrypt) {
        ...
        // Generate random IV

    ...
        cipher.init(Cipher.ENCRYPT_MODE, key);
        byte[] encryptedString = cipher.doFinal(
            toEncrypt.getBytes("UTF-8"));
        ...
    }
    ...
}
```

Listing 1: SensorWrapper class

At the moment we simply hash the password using the hashing algorithm SHA256 and transfer the hashed password to the server via the REST API. At the same time we use the same hash to de- and encrypt the users usernames and passwords, which of course is a fatal mistake. But for our purposes this suffices, as we do not want anyone to use our system in production yet.

It is important to mention, that we decided to encode all passwords with Base64 to make it easier to debug the connection because of the lack of any binary transfers.

In later implementations, it might be useful to use the bcrypt or scrypt libraries for a much better security.

4.3 Problems

4.3.1 Callback stuff

4.3.2 Long clicks

4.3.3 Inefficiency

Every time we login to the server we update our local database by dropping the current password table and adding all the passwords we get from the server. It's obvious that we reinsert most of the entries. This concept works fine as long as there are just a few password. But consider the case we have some thousand ones we might get an unacceptable latency. It would take much to much time especially for displaying the entries.

One possible solution would be to send just the information that is needed. But this approach would make the whole application design much more complicated especially for implementing the server. It would have to keep track of which client got which information by now.

4.4 Temporary Loss of Consistency

The Basic Strategy for changing our database works in a best effort like manner. We update our local database every time when we're adding, deleting or editing a password. On the other hand the server database will only be updated if it's possible to get a connection. If that's not the case we don't do much more at moment. Consider the case someone makes any changes while he's offline. The Server will never get updated and hence the other affiliated devices will never learn about the modifications. It's even worse because every time the application restarts the local database gets overwritten and all changes will be lost. As the only good side effect we get that the consistency is reestablished.

To solve this problem we would have to add another table that stores informations about all the modifications that couldn't be sent to the server for any reasons. When a connection succeeded it should exchange as much data as possible. In the optimal case both databases get consistent. Otherwise keep the not yet sent passwords stored.

5. FURTHER PLANS

We intended to build some extra features. For example, we wanted to give the client the possibility to save some comments for each Password. Those could be shared among the different devices or just stored locally. we also planed to give the clients the possibility not just to register for given Accounts but to create new ones as well. This would be handy if someone wants to share the service with different peoples he didn't even met yet. With our current solution we would have to create for example a website to get the same functionality. Even though that's much less handy.

A small gimmick we wanted to include is a password and user name generator. We have even written most of the code but we decided to focus on the main features.

Something else fairly simple to implement is to provide a button that gives the user the possibility to decide whether it should update the passwords or not.

5.1 More Features

It might be interesting to give the users the possibility to merge the informations of several servers. The simplest approach is to just store all the date on just one server. An other idea would be to let the passwords where they're stored but to create for the user an illusion of storing everything on just one server. It's even possible to store the passwords on several servers to guarantee that no information is lost. The User wouldn't have to remember any longer where what Password is stored.

There are much more Features one might implement, for example some kind of hierarchy between the passwords or to give permissions to other users to access just some specified passwords. But most of them are kind of tricky to implement.

6. CONCLUSION

7. REFERENCES

[1] Flask-RESTful - Flask-RESTful 0.2.1 documentation. `http://flask-restful.readthedocs.org/en/latest/`. Accessed on 19 Dec 2013.

[2] Flask-SQLAlchemy - Flask-SQLAlchemy 0.16 documentation. `http://pythonhosted.org/Flask-SQLAlchemy/`. Accessed on 19 Dec 2013.

[3] KeePass Password Safe. `http://keepass.info/`. Accessed on 19 Dec 2013.

[4] LastPass | The Last Password You Have to Remember. `http://lastpass.com/`. Accessed on 19 Dec 2013.

[5] LastPass Password Manager - Wikipedia, the free encyclopedia. `https://en.wikipedia.org/wiki/LastPass`. Accessed on 19 Dec 2013.

[6] Python Programming Language - Official Website. `http://python.org/`. Accessed on 19 Dec 2013.

[7] SQLAlchemy - The Database Toolkit for Python. `http://www.sqlalchemy.org/`. Accessed on 19 Dec 2013.

[8] Welcome | Flask (A Python Microframework). `http://flask.pocoo.org/`. Accessed on 19 Dec 2013.