Technische Hochschule Rosenheim
Fakultät für Informatik
Prof. Dr. Reiner Hüttl

IT-Security

Exercise 6

In this exercise, we make some consideration about storing passwords. We implement some password hashing algorithms and test them in a Junit Test.

Task 1: Questions about saving passwords

Answer the following questions:

1. What is the problem with storing a password in plaintext?

2. What problems are there with the encrypted storage of the password?

3. What are the advantages of storing the password using a hash function?
   Which attacks make these procedures insecure?

4. How do Hash Look Up Tables and Dictionary Attacks work?

5. How do rainbowtables work?

6. How do "salted" passwords work?
   Why do they provide additional security?

7. How can passwords stored with a hash and a salt be further improved?

Information can be found at https://happycoding.io/tutorials/java-server/secure-password-storage

Task 2: Implementation of hashing algorithms with Java JCE

Implement the hashing algorithms ***SHA512*** and ***PBKDF2*** in the `PasswordHashing` class with the Java JCE and verify them with the given Junit test TestPasswordHashing.

Documentation about the necessary classes you can find here

- https://docs.oracle.com/en/java/javase/17/docs/api//java.base/java/security/MessageDigest.html
- https://docs.oracle.com/en/java/javase/17/docs/api//java.base/javax/crypto/spec/PBEKeySpec.html
- https://docs.oracle.com/en/java/javase/17/docs/api/java.base/javax/crypto/SecretKeyFactory.html

Hints:

- Make a Maven Project in your IDE and us the given `pom.XML`.
  You need dependencies to `springframework`, `bouncycastle` and `junit`.
- Combine the hash procedure with a salt

Task 3: Implementation of hashing algorithms with Spring Security

Now we implement the algorithms ***Bcrypt***, ***Scrypt*** and ***Argon2*** using the library Spring Security
https://docs.spring.io/spring-security/site/docs/current/api/index.html
https://spring.io/
https://docs.spring.io/spring-security/reference/index.html

Use the classes `BCryptPasswordEncoder, Argon2PasswordEncoder, SCryptPasswordEncoder` and verify your implementation with the Junit test `TestPasswordHashing.`

When running the tests pay attention to the execution times of each algorithm. Vary the parameters of the algorithms and observe the changes in the execution time.

Remark: Spring Boot also supports PBKDF2. In this exercise, however, the implementation from the Java standard library is used to show that it is possible without additional dependencies.

Task 4: Futureproofing the application

Hashing Algorithm can get outdated, or the parameters must be modified to resist attackers in the future.

Consider a procedure to change the hash algorithm in a running application and discuss your solution with the others.