Master MIASHS
Université Grenoble Alpes
Jérôme David
2017-2018

Authentication in Java web apps

There are many (standards) mechanism for user authentication and access right management in Java EE (see Chapters 53 &54 of Java tutorial for details, since JavaEE 8, it seems to be easier to use).

In this course, we will not use the Java EE built-in authentication mechanisms but we will implements a basic one that use of course a minimal security (password hashes).

When a registered user want to login on a web app, he/she will use a login and a password.
For security reasons, a password should never be stored as plain text in the application data.
Basic recommendations are:

- use HTTPS,
- store salted hashes of passwords,
- blacklist IPs after a fixed number of failed attempts,
- enforce some policies on user password (at least a minimal length).

For the project, we will see how to implement salted hash of password.

Cryptographic hash functions allows to map string (i.e. passwords) into a fixed sized hash value from which it is difficult to reconstruct the original password. The problem is that if two users have the same password, then the hash values are the same. This makes the protection sensible to brute force attacks. In order to avoid this problem, we usually used a salt. A salt is a randomly generated string that is concatenated to the password before hashing it.

Principle of hashing password

The process of hashing passwords is the following:

- When the user creates/update his account, he provides a password. This password is send from the client (browser) to the server in a secure way (through HTTPS).
- On the server side, we receive this password. We generate a random string. We concatenate the password and the salt (i.e. a random string of the size of the hash function), and we generate the hash. On the user record, we store both the hash and the salt.
- When a user logs in the app, we receive a password and an username (mail address). We retrieve the salt and hash of the given username. We concatenate the provided password and the salt. We hash them and we compare the result with the stored hash. If there are the same, it is safe to assume that the user provided the right password.

Implementation

To implement this functionality, we will have to implement:

- a class (entity) that represent a user with its login (email), password hash and salt. Instances of this class will be store into a database;
- a class (session named bean) and form for login and logout actions. This class will also store the connected user for the session;

- a servlet filter which will look before transmitting a request to other app component if a user is authenticated for the current session. If not, if will not allow to access the requested page and redirect the request to the login form. More information on filters at: More information is available at: https://javaee.github.io/tutorial/servlets006.html

We will also have to use functions to generate salt and for hashing string. To that extent, you will use the following class (Note: there is now in java EE 8 some utility that facilitate this):

```
import java.nio.charset.Charset;
import java.security.*;
import java.util.*;
import java.util.logging.*;

/**
 *
 * @author Jerome David <jerome.david@univ-grenoble-alpes.fr>
 */
public class SecurityUtil {

    private static final SecureRandom RANDOM = new SecureRandom();
    private static final Charset UTF8= Charset.forName("UTF-8");

    public static String generateSalt() {
        byte[] salt = new byte[64];
        RANDOM.nextBytes(salt);
        return new String(Base64.getEncoder().encode(salt),UTF8);
    }

    public static String generateHash(String password, String salt) {
        byte[] passwordB = password.getBytes(UTF8);
        byte[] saltB = Base64.getDecoder().decode(salt.getBytes(UTF8));
        try {
            MessageDigest md = MessageDigest.getInstance("SHA-512");
                             byte[]  saltyPassword  =  Arrays.copyOf(passwordB,
passwordB.length+saltB.length);
                   System.arraycopy(saltB,  0,  saltyPassword,passwordB.length  ,
saltB.length);
            byte[] hashB = md.digest(saltyPassword);
            return new String(Base64.getEncoder().encode(hashB),UTF8);
        } catch (NoSuchAlgorithmException ex) {
                Logger.getLogger(SecurityUtil.class.getName()).log(Level.SEVERE,
null, ex);
            throw new RuntimeException(ex);
        }
    }
}
```

Further reading

If you are interested by such auth mechanisms, I suggest you to have a look to https://shiro.apache.org/ which is a complete library for java web apps security.