Javulna

Table of Contents

Introduction

Javulna is an intentionally vulnerable Java application. It is created for educational purposes. It is intended mainly for Java developers.
Javulna is a movie-related application, where you can log in and out, read information about movies, buy movie-related objects, send messages to other users of the application, etc. The functionalities are far from complete or coherent, they just serve the purpose of demonstrating specific vulnerabilities.
This document contains exercises which can be done with Javulna to understand how to exploit and how to fix specific vulnerabilities.

Building the application

Javulna is a standard Spring Boot application, built with Maven.

You can build the project with:

```
mvn clean install
```

Than you can run it with

```
java -jar target/javulna-1.0-SNAPSHOT.jar
```

This will start an embedded Tomcat, and run the app. If you want to change the port of the embedded Tomcat to 8089 (default is 8080):

```
java -jar target/javulna-1.0-SNAPSHOT.jar --server.port=8089
```

If you want to debug it:

```
java -Xdebug
-Xrunjdwp:server=y,transport=dt_socket_address=5005,suspend=n -jar
target/javulna-1.0-SNAPSHOT.jar
```

Alternatively you can run (and debug) the project from your preferred IDE by simply running the Application.java class.

Accessing the API via Postman

Javulna in itself does not contain any user interface (except a default login page and an empty index.html). It is a RESTfull application accepting http requests and responding JSON strings. In the doc folder you can find a Postman collection export. We suggest you to install Postman on your device and import this collection, since it helps you a lot with starting the exercises.

Exercises

Exercise 1 – Find users of the app and their passwords

Short Description
The list of the movies of the application is accessible by all users (including anonymous users too). Find a vulnerability in this service and exploit it, so that you can see all users of the application and their passwords!

Service endpoint
On the /rest/users endpoint you can list movies of the database. This endpoint is accessible to anonymous (not logged in) users too.
Request Method: `GET`
URL: `/rest/movie?title=<title>&description=<desc>&genre=<genre>&id=<id>` (none of the request parameters are mandatory)
Response: a JSON containg movies which fulfill the search conditions

Postman request
With Postman check the List Movies request in the Javulna collection to see how it works!

Detailed description
The service behind this endpoint is vulnerable to one of the most classic exploit of programming. Find the vulnerability, and exploit it so that you can get users and their passwords from the database! (Hint: The table containing the users' data is called APPUSER.)
When you are done, check the source code (MovieService.findMovie) and fix it.
Discuss what could be the developers motivation creating this code!

Exercise 2 - log in to the application

Short Description
Using the usernames and passwords discovered in the previous exercise log in to the application. There is no hacking involved here, this step is only necessary so that you can continue with the next exercises.

Service endpoint
On the /rest/users endpoint you can list movies of the database. This endpoint is accessible to anonymous (not logged in) users too.
Request Method: `POST`
URL: `/login`
Request body: username, password fields
Response: a JSON containg the name of the logged in user and a cookie which can be used for subsequent authentication

Postman request
Use the login request in the Javulna collection (Postman will automatically submit the cookie with the following requests)

Exercise 3 – change another user's password

Short Description
The application contains a password change functionality. Abuse it to change another user's password!

Service endpoint

Request Method: `GET`
URL: `/rest/user/password?user=Yoda&oldPassword=<old_password>&newPassword=<new_password>/`
Response: Ok or Not ok

Postman request
Change password

Detailed description
The change password service first creates a password-change xml to call a remote password change service with it (in reality the remote service does nothing remotely, just parses the xml and changes the password locally).
Find a vulnerability within this service!
This is how the password service creates the xml file:

```
private String createXml(String name, String newPassword) {
   try {
      String xmlString =
IOUtils.toString(getClass().getClassLoader().getResourceAsStream("xml/PasswordChange.xml"),
"UTF-8");
      xmlString = xmlString.replaceAll("PWD_TO_REPLACE", newPassword);
      xmlString = xmlString.replaceAll("USERNAME_TO_REPLACE", name);
      return xmlString;
   } catch (IOException ex) {
      throw new RuntimeException(ex);
   }
}
```

The PasswordChange.xml looks like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<PasswrodChange>
   <pwd>PWD_TO_REPLACE</pwd>
   <userName>USERNAME_TO_REPLACE</userName>
</PasswrodChange>
```

After the exploit fix the vulnerability within the code.

Exercise 4 – Buy cheaper

Short Description

You can buy movie-related objects with the application. Each object have a name, a description and a price. Try to by something for cheaper than the original price!

Service endpoint

Request Method: `PUT`
URL: `/rest/order`
Body: a JSON string containing the order

```
{
     "orderItems": [{
          "movieObjectId": "<id>",
          "nrOfItemsOrdered": <Number of items>
     },
     {
          "movieObjectId": "<id>",
          "nrOfItemsOrdered": <Number of items>
     }
     ]
}
```

Response: a JSON containing the details of the order and the final price.

Postman request
Use the “Buy movie objects” request to place an order and the “List buyable movie objects” request to see what you can buy!

Detailed description
Find a way to buy something for a cheaper price than intended!
After you found the vulerability, fix the code!

Exercise 5 –

Short Description

Service endpoint

Request Method: `GET`
URL: `/rest/`
Response:

Postman request

Detailed description

Exercise 6 –

Short Description

Service endpoint

Request Method: `GET`

URL: `/rest/`
Response:

Postman request

Detailed description