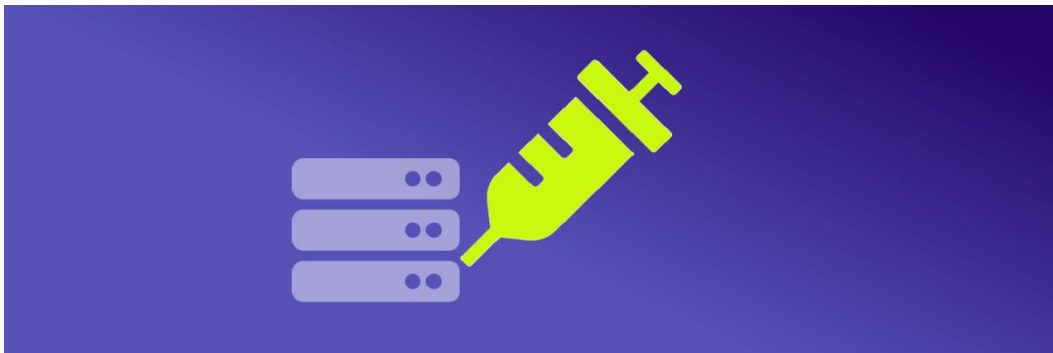


18 MARS 2019



# TP REMOTE 1

SYNTHESE

TRISTAN GUERIN

ENSIBS

Cybersécurité du Logiciel A2

Description	3
Base de données	3
Application	4
Scan d'application	6
Unsafe	6
Safe	7
Page <i>Unsafe</i>	8
Exemples d'injection	9
Page <i>Safe</i>	11
Injections inefficaces	12
Sources	13

## Description

Le but de ce TP est de créer une application web se connectant à une base de données et permettant ainsi d'afficher les informations contenues dans une table de la base de données.

Cela va nous permettre de tester quelques injections SQL possibles sur cette application et des solutions pour y remédier.

## Base de données

On crée tout d'abord une base de données nommée selon notre nom de famille. (Ici base de données 'GUERIN' dans le SGBD 'MySQL')

On peut ensuite créer une table 'user' comportant cinq champs différents ('id', 'name', 'password', 'salary' et 'age').

On insère ensuite quelques valeurs à l'intérieur.

On crée aussi un utilisateur 'GUERIN' ayant accès qu'à cette table (il ne possède pas de mot de passe).

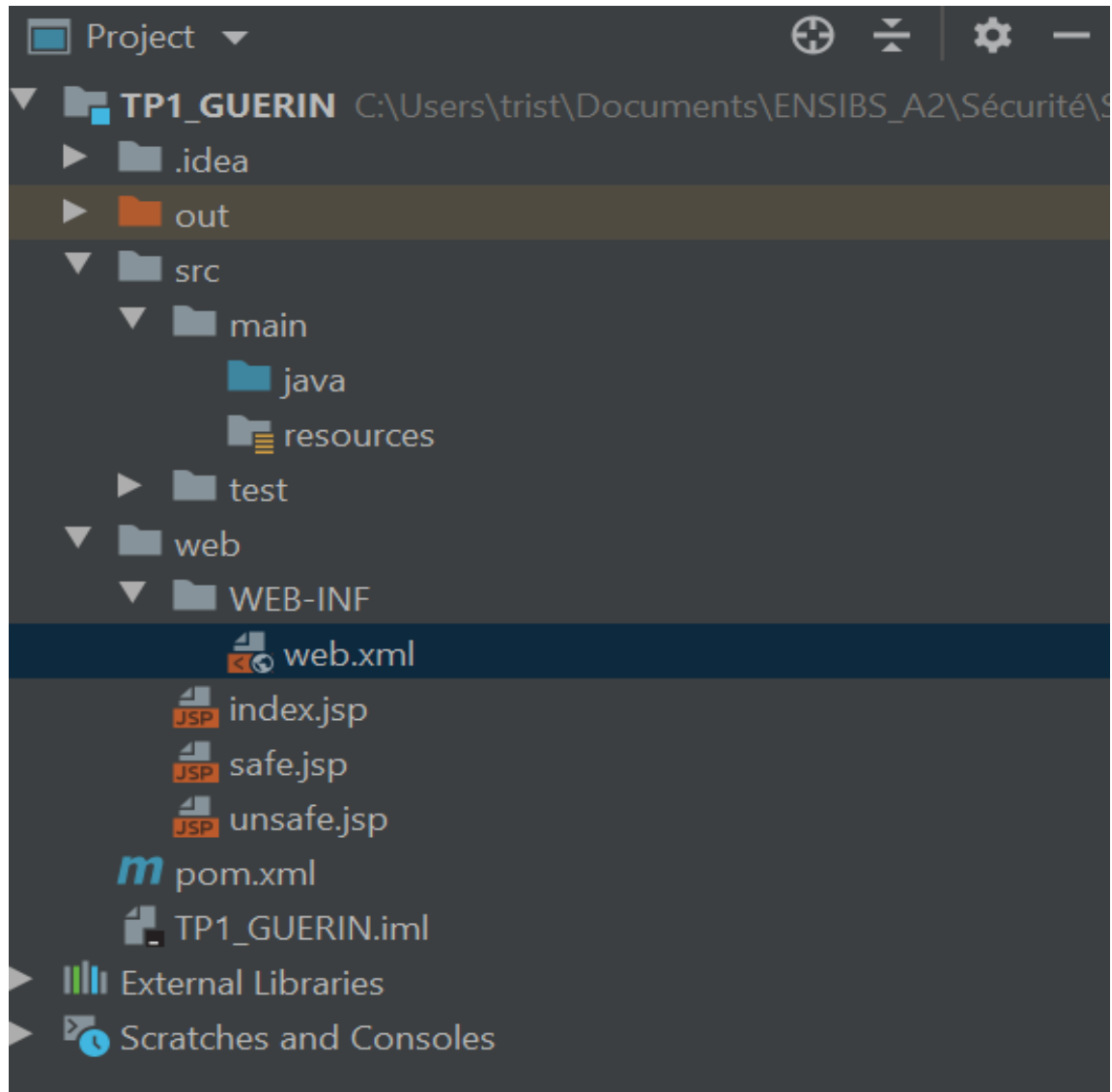
(Toutes les commandes nécessaires sont dans le fichier 'scriptSQL.sql' ci-joint à ce rapport)

```
mysql> select * from GUERIN.user;
+----+-----+-----+-----+-----+
| id | name  | password | salary | age |
+----+-----+-----+-----+-----+
| 1  | user1 | test     | 2500   | 21  |
| 2  | user2 | test2    | 2400   | 27  |
| 3  | user3 | test3    | 3900   | 18  |
+----+-----+-----+-----+-----+
3 rows in set (0.00 sec)

mysql> show columns from GUERIN.user;
+-----+-----+-----+-----+-----+-----+
| Field | Type          | Null | Key | Default | Extra          |
+-----+-----+-----+-----+-----+-----+
| id    | int(11)       | NO   | PRI | NULL    | auto_increment |
| name  | varchar(20)   | YES  |     | NULL    |                |
| password | varchar(30)  | YES  |     | NULL    |                |
| salary | bigint(20)    | NO   |     | NULL    |                |
| age   | int(10) unsigned | NO   |     | NULL    |                |
+-----+-----+-----+-----+-----+-----+
5 rows in set (0.02 sec)
```

## Application

L'application est développée en Java/JSP et utilise 'Tomcat 9.0.30'.



*Structure du projet sous IntelliJ*

Elle se compose alors notamment de trois fichiers principaux : *index.jsp*, *unsafe.jsp* et *safe.jsp*.

Lors du lancement du serveur *Tomcat*, le navigateur ouvre directement la page *index.jsp* à l'adresse suivante :

[http://localhost:8080/TP1\\_GUERIN\\_war\\_exploded/](http://localhost:8080/TP1_GUERIN_war_exploded/)

App Web

## Connection status

Successfully connected to mysql database

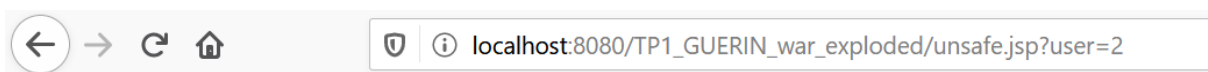
Site	Unsafe link	Safe link
user1	<a href="#">Unsafe details page</a>	<a href="#">Safe details page</a>
user2	<a href="#">Unsafe details page</a>	<a href="#">Safe details page</a>
user3	<a href="#">Unsafe details page</a>	<a href="#">Safe details page</a>

Cette page nous affiche alors la liste des utilisateurs dans la table 'user'.

Pour chaque utilisateur, il existe deux liens :

Un lien sensible à l'injection SQL

Un lien non sensible



Details :

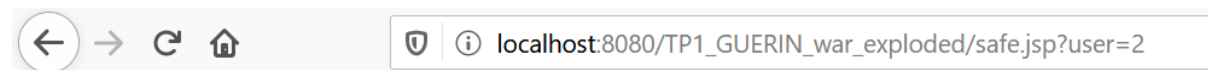
ID :2

Name :user2

Password :test2

Salary :2400

Age :27



Details :

ID :2

Name :user2

Password :test2

Salary :2400

Age :27

On voit ici qu'en cliquant sur les deux liens concernant un utilisateur, les détails de ce dernier s'affiche.

On va alors pouvoir scanner les deux pages avec un *OWASP* comme *ZAP* afin de voir les vulnérabilités sur ces pages.

# Scan d'application

## Unsafe

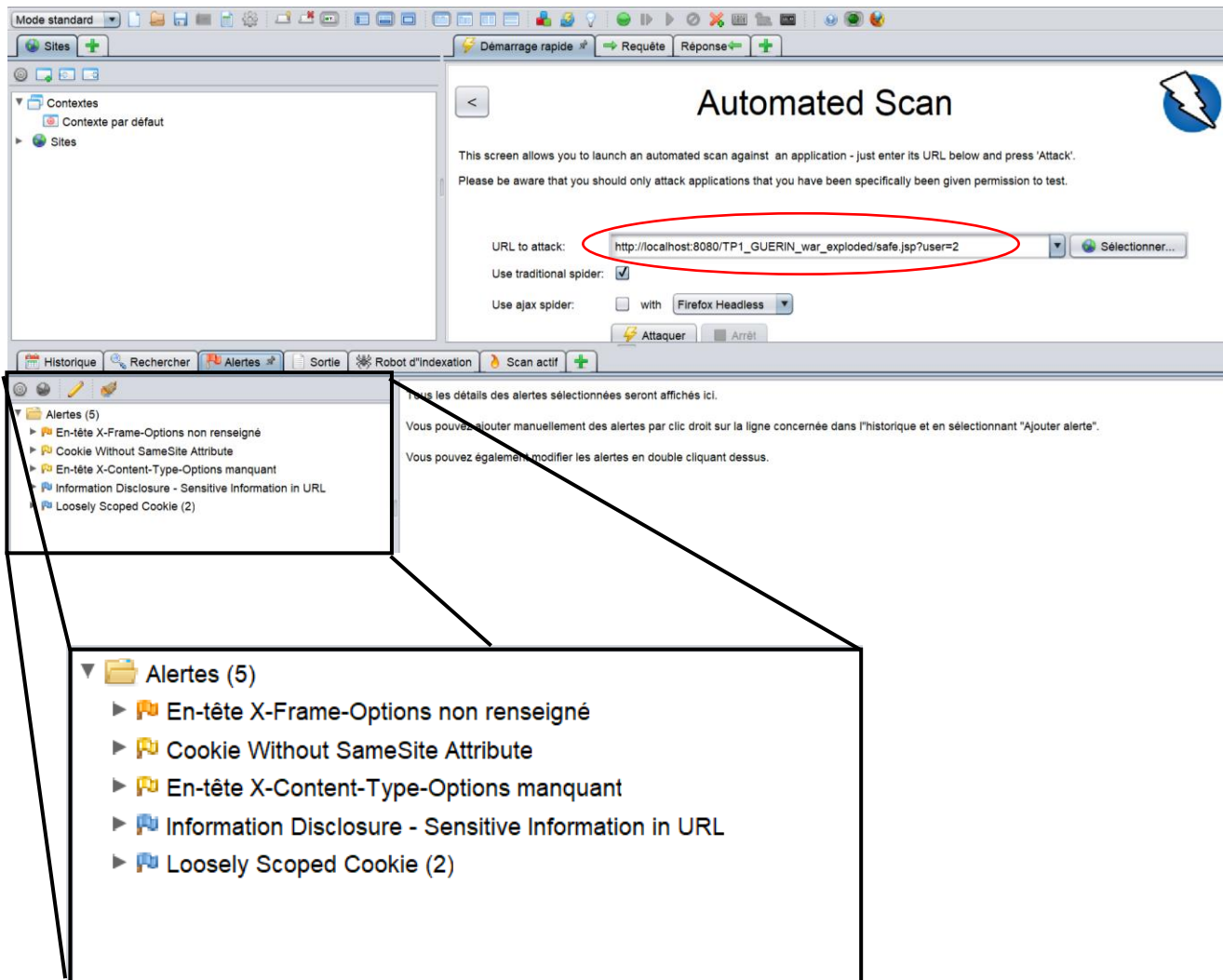
The screenshot displays the ZAP web interface. The 'Automated Scan' section is active, showing the URL to attack as `http://localhost:8080/TP1_GUERIN_war_exploded/unsafe.jsp?user=2`, which is circled in red. The 'Use traditional spider' checkbox is checked. Below this, the 'Attaquer' button is visible. The 'Alertes' pane on the left shows a list of alerts, with 'Injection SQL' selected. A detailed view of the 'Injection SQL' alert is shown in a callout box, containing the following information:

**Injection SQL**  
URL: `http://localhost:8080/TP1_GUERIN_war_exploded/unsafe.jsp?user=4-2`  
Risque: High  
Confiance: Medium  
Paramètre: user  
Attaquer: 4-2  
Preuve :  
Id CWE : 89  
Id WASC : 19  
Source: Actif (40018 - Injection SQL)  
Description:  
Une injection SQL peut être possible.

On effectue ici avec ZAP un scan automatique sur la page [http://localhost:8080/TP1\\_GUERIN\\_war\\_exploded/unsafe.jsp](http://localhost:8080/TP1_GUERIN_war_exploded/unsafe.jsp).

Le résultat du scan nous montre bien qu'une injection SQL est possible, en nous donnant comme exemple de mettre 4-2 comme paramètre d'un id utilisateur, alors que seul un entier devrait être ici possible. On essaiera plus tard dans ce rapport différentes injections sur cette page.

Safe



On effectue ensuite avec ZAP un scan automatique sur la page [http://localhost:8080/TP1\\_GUERIN\\_war\\_exploded/safe.jsp](http://localhost:8080/TP1_GUERIN_war_exploded/safe.jsp).

On remarque alors que sur cette page aucune injection SQL n'est possible, les seuls alertes de ZAP sont des en-têtes non renseignés/manquants.

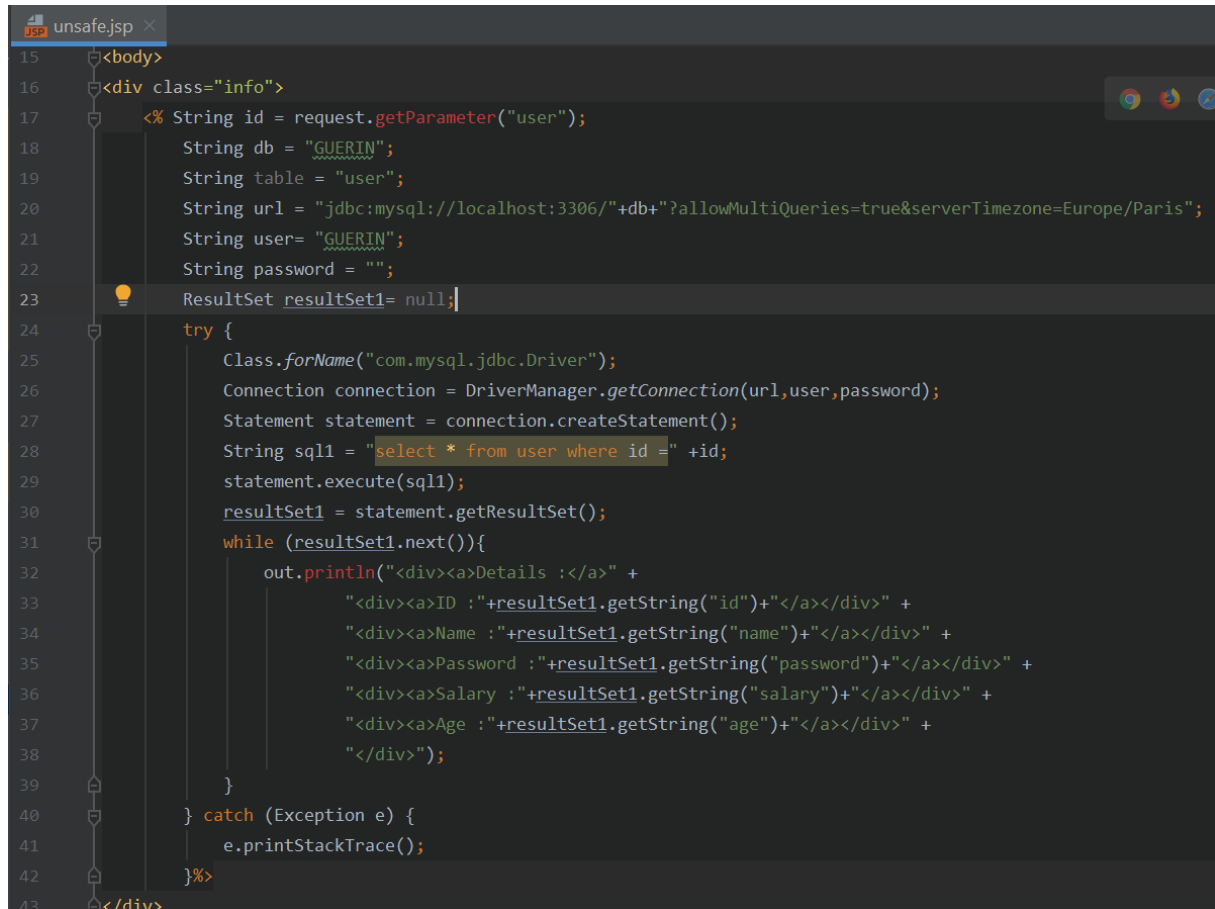
```
java.lang.NumberFormatException: For input string: "c:/Windows/system.ini"
at java.base/java.lang.NumberFormatException.forInputString(NumberFormatException.java:68)
at java.base/java.lang.Integer.parseInt(Integer.java:658)
at java.base/java.lang.Integer.parseInt(Integer.java:776)
at org.apache.jsp.safe_jsp._jspService(safe_jsp.java:149)
at org.apache.jasper.runtime.HttpJspBase.service(HttpJspBase.java:70)
at javax.servlet.http.HttpServlet.service(HttpServlet.java:741)
at org.apache.jasper.servlet.JspServletWrapper.service(JspServletWrapper.java:477)
at org.apache.jasper.servlet.JspServlet.serviceJspFile(JspServlet.java:385)
at org.apache.jasper.servlet.JspServlet.service(JspServlet.java:329)
at javax.servlet.http.HttpServlet.service(HttpServlet.java:741)
at org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:231)
at org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:166)
at org.apache.tomcat.websocket.server.WsFilter.doFilter(WsFilter.java:53)
at org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:193)
at org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:166)
```

On remarque d'ailleurs dans la console du serveur Tomcat qu'il y a eu pour chaque test d'injection de ZAP une erreur levée car l'application attendait un entier comme paramètre de la page.

Les rapports des deux scans sont ci-joints à cette synthèse.

## Page Unsafe

On va voir ici comment est codé la page *unsafe.jsp* et pourquoi elle est sensible aux injections SQL.



```
15 <body>
16 <div class="info">
17   <% String id = request.getParameter("user");
18       String db = "GUERIN";
19       String table = "user";
20       String url = "jdbc:mysql://localhost:3306/"+db+"?allowMultiQueries=true&serverTimezone=Europe/Paris";
21       String user= "GUERIN";
22       String password = "";
23       ResultSet resultSet1= null;
24       try {
25         Class.forName("com.mysql.jdbc.Driver");
26         Connection connection = DriverManager.getConnection(url,user,password);
27         Statement statement = connection.createStatement();
28         String sql1 = "select * from user where id =" +id;
29         statement.execute(sql1);
30         resultSet1 = statement.getResultSet();
31         while (resultSet1.next()){
32           out.println("<div><a>Details :</a>" +
33             "<div><a>ID :"+resultSet1.getString("id")+"</a></div>" +
34             "<div><a>Name :"+resultSet1.getString("name")+"</a></div>" +
35             "<div><a>Password :"+resultSet1.getString("password")+"</a></div>" +
36             "<div><a>Salary :"+resultSet1.getString("salary")+"</a></div>" +
37             "<div><a>Age :"+resultSet1.getString("age")+"</a></div>" +
38             "</div>");
39         }
40       } catch (Exception e) {
41         e.printStackTrace();
42       }%>
43 </div>
```

Ici on comprend que la page effectue une connexion à la base de données 'GUERIN' en tant que 'GUERIN'.

Ce qui est indiqué après *unsafe.jsp?user=* est récupéré dans une variable *id*.

On construit la requête suivante : 'select \* from user where id=*id*' qui va être exécutée par la base de données. Aucune vérification plus précise n'est effectuée, et la fonction *execute* de la classe *Statement* permet même d'exécuter plusieurs requêtes à la fois.



## Exemples d'injection

On va pouvoir afficher tous les utilisateurs de la table 'user' en rajoutant `%20or%201--` à l'URL de départ :

[http://localhost:8080/TP1\\_GUERIN\\_war\\_exploded/unsafe.jsp?user=2%20or%201--](http://localhost:8080/TP1_GUERIN_war_exploded/unsafe.jsp?user=2%20or%201--)

Cela va alors permettre de créer la requête suivante : *select \* from user where id=2 or 1--* ;



Les détails de chaque utilisateur ont donc été affichés sur la page.

Cette injection est une des plus simple et ne permet que de récupérer des informations, mais certaines injections peuvent avoir des incidences plus graves.

En rajoutant

`;%20insert%20into%20user(name,password,salary,age)%20values%20(%27user4%27,%27test4%27,5600,78)` à l'URL de départ

([http://localhost:8080/TP1\\_GUERIN\\_war\\_exploded/unsafe.jsp?user=2;%20insert%20into%20user\(name,password,salary,age\)%20values%20\(%27user4%27,%27test4%27,5600,78\)\)](http://localhost:8080/TP1_GUERIN_war_exploded/unsafe.jsp?user=2;%20insert%20into%20user(name,password,salary,age)%20values%20(%27user4%27,%27test4%27,5600,78)))), cela va construire la requête suivante :

```
select * from user where id=2 ; insert into user(name,password,salary,age)
values('user4','test4',5600,78) ;
```

Cela va alors dans un premier temps récupérer les informations concernant l'utilisateur d'id 2 et ensuite insérer un nouveau tuple dans la table 'user'.

← → ↺ 🏠 localhost:8080/TP1\_GUERIN\_war\_exploded/unsafe.jsp?user=2; insert into user(name,p

Details :  
ID :2  
Name :user2  
Password :test2  
Salary :2400  
Age :27

On peut vérifier alors dans la table qu'une nouvelle ligne a bien été ajoutée :

```
mysql> show tables from GUERIN;  
+-----+  
| Tables_in_guerin |  
+-----+  
| user              |  
+-----+  
1 row in set (0.01 sec)
```

```
mysql> select * from GUERIN.user;  
+----+-----+-----+-----+-----+  
| id | name  | password | salary | age |  
+----+-----+-----+-----+-----+  
| 1  | user1 | test     | 2500   | 21  |  
| 2  | user2 | test2    | 2400   | 27  |  
| 3  | user3 | test3    | 3900   | 18  |  
| 5  | user4 | test4    | 5600   | 78  |  
+----+-----+-----+-----+-----+  
4 rows in set (0.00 sec)
```

On peut tester une dernière injection en rajoutant `;%20drop%20table%20user` à l'URL de départ ([http://localhost:8080/TP1\\_GUERIN\\_war\\_exploded/unsafe.jsp?user=2;%20drop%20table%20user](http://localhost:8080/TP1_GUERIN_war_exploded/unsafe.jsp?user=2;%20drop%20table%20user)).

Cela va construire la requête suivante :

```
select * from user where id=2 ; drop table user ;
```

Cela va alors dans un premier temps récupérer les informations concernant l'utilisateur d'*id* 2 et ensuite supprimer la table 'user' de la base de données 'GUERIN'.

← → ↺ 🏠 localhost:8080/TP1\_GUERIN\_war\_exploded/unsafe.jsp?user=2; drop table user

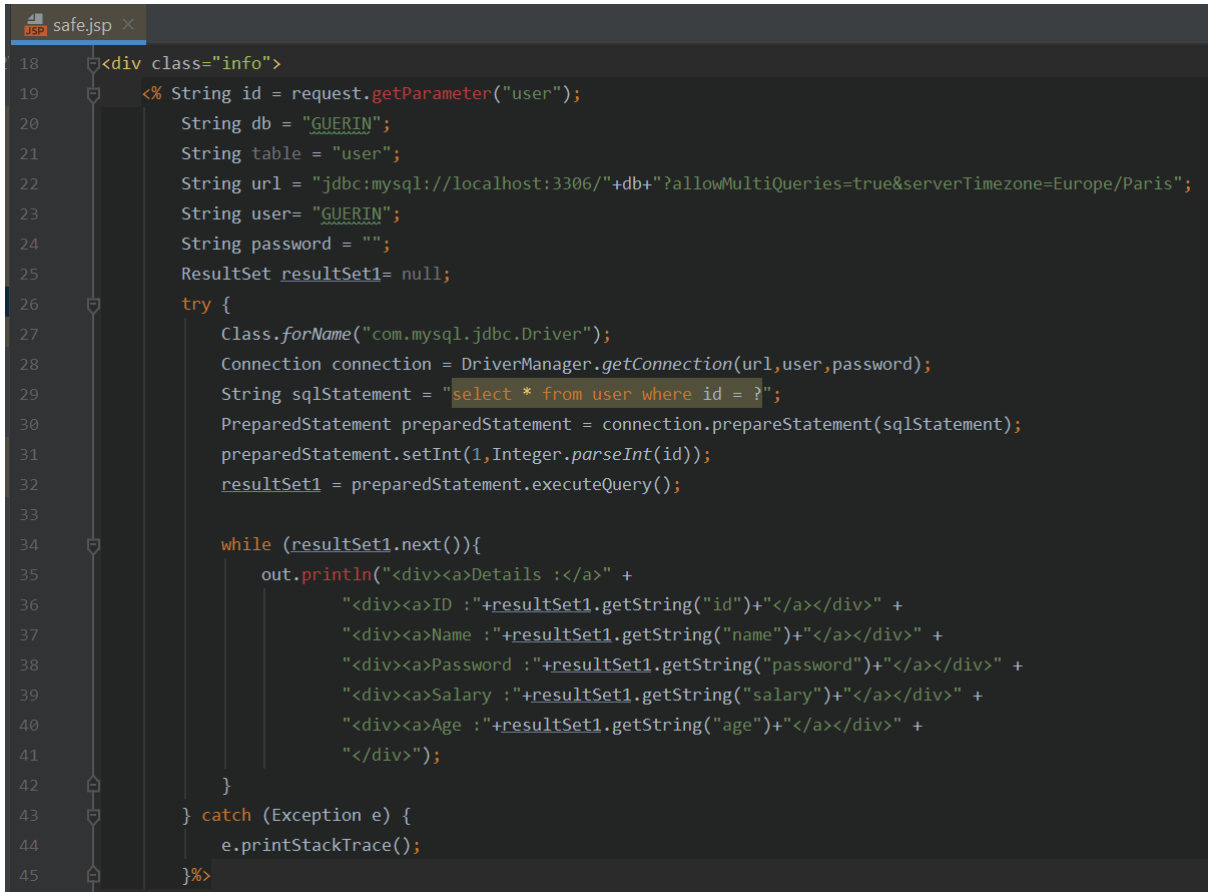
Details :  
ID :2  
Name :user2  
Password :test2  
Salary :2400  
Age :27

```
mysql> show tables from GUERIN;  
Empty set (0.00 sec)
```

On voit bien que la table a été supprimée de la base de données.

## Page Safe

On va voir ici comment est codé la page *safe.jsp* et pourquoi elle n'est pas sensible aux injections SQL.



```
18 <div class="info">
19 <% String id = request.getParameter("user");
20     String db = "GUERIN";
21     String table = "user";
22     String url = "jdbc:mysql://localhost:3306/"+db+"?allowMultiQueries=true&serverTimezone=Europe/Paris";
23     String user= "GUERIN";
24     String password = "";
25     ResultSet resultSet1= null;
26     try {
27         Class.forName("com.mysql.jdbc.Driver");
28         Connection connection = DriverManager.getConnection(url,user,password);
29         String sqlStatement = "select * from user where id = ?";
30         PreparedStatement preparedStatement = connection.prepareStatement(sqlStatement);
31         preparedStatement.setInt(1,Integer.parseInt(id));
32         resultSet1 = preparedStatement.executeQuery();
33
34         while (resultSet1.next()){
35             out.println("<div><a>Details :</a>" +
36                 "<div><a>ID : "+resultSet1.getString("id")+"</a></div>" +
37                 "<div><a>Name : "+resultSet1.getString("name")+"</a></div>" +
38                 "<div><a>Password : "+resultSet1.getString("password")+"</a></div>" +
39                 "<div><a>Salary : "+resultSet1.getString("salary")+"</a></div>" +
40                 "<div><a>Age : "+resultSet1.getString("age")+"</a></div>" +
41                 "</div>");
42         }
43     } catch (Exception e) {
44         e.printStackTrace();
45     }%>
```

Ici on comprend que la page effectue une connexion à la base de données 'GUERIN' en tant que 'GUERIN'.

Ce qui est indiqué après *safe.jsp?user=* est récupéré dans une variable *id*.

On construit la requête suivante : 'select \* from user where id= ?'.

On utilise ici les *PreparedStatement* à la place de simple *Statement* afin de pouvoir préciser le type attendu pour chaque paramètre à donner à la requête.

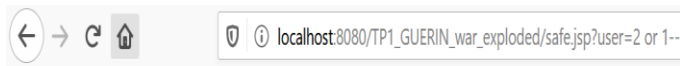
Ici on précise via l'instruction *preparedStatement.setInt(1,Integer.parseInt(id))* ; que le premier paramètre nécessaire à la requête doit être un entier.

La méthode *executeQuery* de la classe *Statement* permet aussi d'empêcher de faire plusieurs requêtes d'un seul coup, et ne permet que de faire des requêtes de type SELECT.

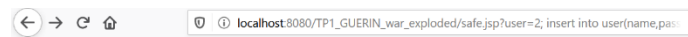
## Injectons inefficaces

Ainsi, comme le scan automatique l'a précédemment indiqué, aucune injection SQL n'est possible sur cette page.

Si l'on teste les injections effectuées sur la page *unsafe.jsp* :



```
java.lang.NumberFormatException: For input string: "2 or 1--"
    at java.base/java.lang.NumberFormatException.forInputString(NumberFormatException.java:68)
    at java.base/java.lang.Integer.parseInt(Integer.java:658)
    at java.base/java.lang.Integer.parseInt(Integer.java:776)
    at org.apache.jsp.safe_jsp._jspService(safe_jsp.java:149)
    at org.apache.jasper.runtime.HttpJspBase.service(HttpJspBase.java:70)
    at javax.servlet.http.HttpServlet.service(HttpServlet.java:741)
    at org.apache.jasper.servlet.JspServletWrapper.service(JspServletWrapper.java:477)
    at org.apache.jasper.servlet.JspServlet.serviceJspFile(JspServlet.java:385)
    at org.apache.jasper.servlet.JspServlet.service(JspServlet.java:329)
    at javax.servlet.http.HttpServlet.service(HttpServlet.java:741)
    at org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:231)
    at org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:166)
```



```
java.lang.NumberFormatException: For input string: "2; insert into user(name,password,salary,age) values ('user4','test4',5600,78)"
    at java.base/java.lang.NumberFormatException.forInputString(NumberFormatException.java:68)
    at java.base/java.lang.Integer.parseInt(Integer.java:658)
    at java.base/java.lang.Integer.parseInt(Integer.java:776)
    at org.apache.jsp.safe_jsp._jspService(safe_jsp.java:149)
    at org.apache.jasper.runtime.HttpJspBase.service(HttpJspBase.java:70)
    at javax.servlet.http.HttpServlet.service(HttpServlet.java:741)
    at org.apache.jasper.servlet.JspServletWrapper.service(JspServletWrapper.java:477)
    at org.apache.jasper.servlet.JspServlet.serviceJspFile(JspServlet.java:385)
    at org.apache.jasper.servlet.JspServlet.service(JspServlet.java:329)
```



```
java.lang.NumberFormatException: For input string: "2; drop table user"
    at java.base/java.lang.NumberFormatException.forInputString(NumberFormatException.java:68)
    at java.base/java.lang.Integer.parseInt(Integer.java:658)
    at java.base/java.lang.Integer.parseInt(Integer.java:776)
    at org.apache.jsp.safe_jsp._jspService(safe_jsp.java:149)
    at org.apache.jasper.runtime.HttpJspBase.service(HttpJspBase.java:70)
    at javax.servlet.http.HttpServlet.service(HttpServlet.java:741)
    at org.apache.jasper.servlet.JspServletWrapper.service(JspServletWrapper.java:477)
    at org.apache.jasper.servlet.JspServlet.serviceJspFile(JspServlet.java:385)
    at org.apache.jasper.servlet.JspServlet.service(JspServlet.java:329)
    at javax.servlet.http.HttpServlet.service(HttpServlet.java:741)
    at org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:231)
    at org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:166)
```

On voit bien qu'à chaque lancement, aucune information n'est affichée et le serveur *Tomcat* lève une exception car il n'obtient pas un argument de type entier.

```
mysql> show tables from GUERIN;
+-----+
| Tables_in_guerin |
+-----+
| user              |
+-----+
1 row in set (0.00 sec)

mysql> select * from GUERIN.user;
+----+-----+-----+-----+-----+
| id | name  | password | salary | age |
+----+-----+-----+-----+-----+
| 1  | user1 | test     | 2500   | 21  |
| 2  | user2 | test2    | 2400   | 27  |
| 3  | user3 | test3    | 3900   | 18  |
+----+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

On peut vérifier que la base de données 'GUERIN' et sa table 'user' n'ont subi aucune modification/suppression.

## Sources

[https://miro.medium.com/max/2844/1\\*qWAFJ0WnyExJw37sQcR3xQ.png](https://miro.medium.com/max/2844/1*qWAFJ0WnyExJw37sQcR3xQ.png)

<http://www.unixwiz.net/techtips/sql-injection.html>

[https://www.w3schools.com/sql/sql\\_injection.asp](https://www.w3schools.com/sql/sql_injection.asp)

<https://www.javacodegeeks.com/2012/11/sql-injection-in-java-application.html>

<https://mkyong.com/intellij/intellij-idea-run-debug-web-application-on-tomcat/>

<http://objis.com/tutoriel-java-n12-acces-base-de-donnees-mysql/>

<https://www.consultingit.fr/en/owasp-zap-et-son-mode-attaque-en-test-des-applications-web-de-ce-scanner-de-vulnerabilites-pour-le-pentesting-et-les-audits-tests-d-intrusions>

<https://docs.oracle.com/javase/7/docs/api/java/sql/Statement.html>

<https://docs.oracle.com/javase/7/docs/api/java/sql/PreparedStatement.html>

<https://stackoverflow.com/questions/10797794/multiple-queries-executed-in-java-in-single-statement>