MIECT: Security 2019-20

Practical Exercise:
SQL injection attack

December 18, 2019 Due date: no date

Changelog

- v1.0 - Initial Version.

1 Introduction

SQL injection attacks represent a serious threat to any database-driven site. The methods behind an attack are easy to learn and the damage caused can range from considerable to complete system compromise. Despite these risks, an incredible number of systems on the Internet are susceptible to this form of attack.

Not only is it a threat easily instigated, it is also a threat that, with a little common-sense and forethought, can easily be prevented.

It is always good practice to clean and validate all input data, especially data that will used in OS command, scripts, and database queries, even if the threat of SQL injection has been prevented in some other manner.

For complete security bind your parameters to the SQL query and do not create string based SQL queries with user (or other external) data.

2 SQL injection attacks

This guide guide will be based on the WebGoat application, which can be downloaded from `https://github.com/WebGoat/WebGoat/releases`; get the latest version (currently v8.0.0.M26) of the server JAR file.

After WebGoat is installed, run it with the `java` command. If you are using a version of Java higher or equal than 9 add `--add-modules java.xml.bind` to the `java` arguments. Access the WebGoat server (`http://localhost:8080/WebGoat`) with a browser, create an account and proceed to the set of attacks denominated "(A1) Injections", which focus on SQL injection issues.

It will be useful to install the "Tamper Data" Firefox extension. This will allow for easily intercept HTTP POST requests, and modify them in order to add injection statements into POST parameters.

2.1 Numeric SQL injection

For this lesson, use the "Tamper Data" extension and intercept the HTTP POST when you select a new weather station. Consider how the code is validating the information you send, and create a valid SQL query which returns all results. In this case, the developer could assume that because the station identifier is hard coded in a Select element, no injection could be done. However, as you will see, requests can be intercepted and tampered with.

2.2 String SQL injection

In this lesson, the objective is similar to the previous. One difference is that quotes must be considered. Therefore, devise a query which takes in consideration the existing quotes and returns all values.

Because data can be entered directly, there is no need to use the “Tamper Data” extension. A simple, commonly used, yet ineffective solution would be to validate the information using Javascript.

2.3 Stage 1: string SQL injection

This attack considers the case where an SQL injection is used to bypass authentication. In this specific case, the validation begin done is very simple. However, the password field is limited to just a few bytes. Therefore, it may be required to use the “Tamper Data” extension.

Choose the user Neville Bartholomew and submit any password. In the “Tamper Data” extension replace the password with an specially crafted SQL injection.

2.4 Stage 2: parameterized query

The objective of this lesson is to prevent the previous attacks by means of Parameterized queries. For this purpose, you must modify file `/var/lib/tomcat7/webapps/WebHoat/WEB-INF/classes/org/owasp/webgoat/lessons/`

`SQLInjection/Login.java` in order to replace line 125 with a prepared statement. In Java this is achieved by means of using the `PreparedStatement` class to prepare the query, setting the parameters, and then issuing the executeQuery() method.

You can rebuild the `war` file by issuing:

```
cd /var/lib/tomcat7/webapps/WebGoat
jar -cvf ../WebGoat.war *
```

2.5 Stage 3: numeric SQL injection

This lesson shows that putting focus in the authentication, and relaxing the security guidelines for authenticated users is not a good solution. The objective is to login as Larry Stooge (password is larry), and the find a way to view his boss profile page. In this case, the “Tamper Data” extension will be useful, and the flaw is in the View Profile action. The problem is that the ViewProfile page only views one record. Therefore, the typical injection string “0 or 1=” will not work.

Just think, in a database, how to differentiate the boss from the remaining workers? Probably it will have an higher salary...

Stage 4: Parameterized Query is similar to Stage 2.

2.6 Modify/add data with SQL injection

Until now the attacks focused in accessing otherwise private information. This attack focus in actually changing the database. The user is asked to provide a username to searched in the `salary` table. However, SQL allows multiple statements in the same query, meaning that an UPDATE or DELETE statement can be injected. Each query must be separated by a `;` character.

The first lesson (Modify Data) focus in modifying an existing entry. With this, you can set the value of any salary. As an example, increase jsmith’s salary to 90000 dollars. Afterwards, verify that jsmith is receiving a much larger salary.

The second lesson (Add Data) requires you to add or remote information from the table. It is similar to the previous case, but will make use of a different SQL statement. You can delete or insert new records. The lesson expects you to insert a new entry to the table. After the attack, verify that the new user was added.

2.7 Blind SQL injection

Blind SQL injection focus in inserting queries in order to obtain information from other fields. The objective is not to change any data.

In the first lesson there is a table named pins with two relevant fields pin and cc_number. We aim to get information from a different table (pins). This is possible to achieve because SQL also supports nested SQL queries, in the format:

```
SELECT * FROM accounts where ACCOUNT='string' AND ((SELECT pin FROM .....) > 1000);
```

In the first lesson, if we know that a given account exists, we can use the AND statement to inject any other SELECT over any other table in the database. The objective is to find the pin code of the account with cc_number=1111222233334444. Check different values, and interpret the resulting message. If the application shows that the account exists, you are guessing correctly. When you find the pin code, just provide that number as the account number.

The second lesson is similar to the first, but operates over Strings, and the objective is to find a password. Instead of comparing if a pin code is larger or smaller than a given amount, you can use the SUBSTRING or LIKE statements to guess the password. These statements can be used as follows:

```
... AND( (SELECT ... ) LIKE 'A%')
... AND( SUBSTRING( SELECT ...),1,1) < 'b')
```

After you find the password, search for it as the account number.