HackerFrogs Afterschool
Beginner Cybersecurity Skills

SQL Injection /w TryHackMe – SQL Injection

Class:	**Web App Exploitation**
Workshop Number:	**AS-WEB-04**
Document Version:	**1.0**
Special Requirements:	**Completion of previous workshop (AS-WEB-03)** **Registered account at tryhackme.com**

Table of Contents

Introduction

Welcome to HackerFrogs Afterschool! HackerFrogs Afterschool is a a series of online workshops meant to teach cybersecurity skills and concepts to beginners who may not be familiar with the field of cybersecurity. This lesson is the final lesson for web app exploitation, which is a very prominent and in-demand specialization in cybersecurity. This lesson will cover the following learning objectives:

- Learn about the SQL injection vulnerabilities in web applications
- Complete the SQL Injection room on the TryHackMe website

Workshop Guide Format Notes

When following instructions in this document, there will be instructions to type specific commands in the CLI window or in a search bar. These commands will be displayed in red, and will have a different `font` applied to them, as well as a light grey background color. For example:

```
this is a sample command text line
```

However, if the input is meant to be put into the web browser or other non-CLI program, it will be colored red, and **bolded** but will not be in a different font nor will it have a light-grey highlight. For example:

This is a sample user input text line

If there are code snippets in this document, they look similar to the format above, except that the text will be black. For example:

```
this is a sample code snippet line
```

Brief Description of Capture the Flag (CTF) games and the TryHackMe Platform

Capture the Flag (CTF) games are training exercises in the field of cybersecurity. The goal of a CTF exercise is to "capture the flag" through use of cybersecurity skills. In this context, "capture" means to gain access to a file or other resource, and "flag" refers to a secret phrase or password.

The CTF platform we will be playing to learn basic web app exploitation skills is called TryHackMe, which is a popular cybersecurity education platform, with many education modules (called rooms) which span a wide range of cybersecurity topics.

Completion of this workshop will require a registered account at TryHackMe, and if we do not already have an account there, we can register a free account at the following URL:

https://tryhackme.com/signup

Part 0 - Accessing The TryHackMe Module

Step 1

Login to the TryHackMe website.

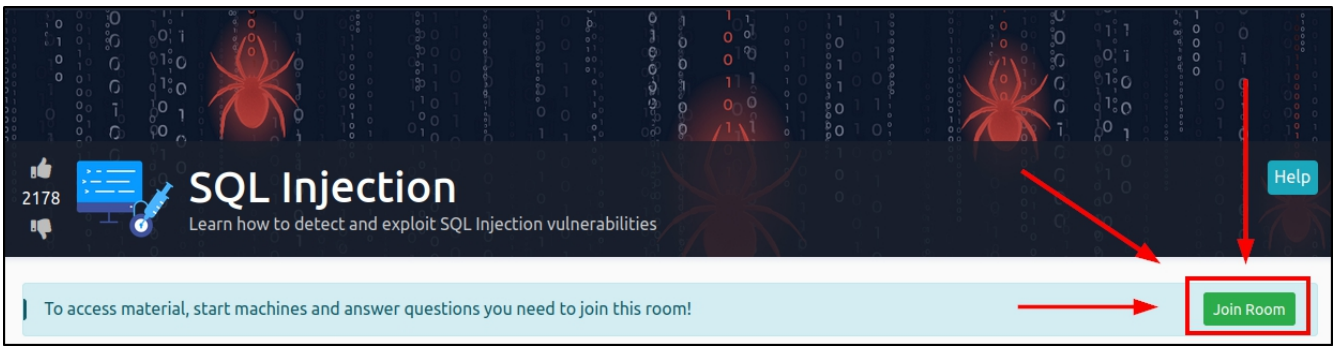
Step 2

Navigate to the following URL:

https://tryhackme.com/room/sqlinjectionlm

Step 3

Click the green **Join Room** button located inside the light-blue text box underneath the **SQL Injection** banner at the top of the webpage. See screenshot below (relevant button outlined in red):



Part 1 – Brief

SQL (Structured Query Language) injection is a type of web application exploit that targets insecure implementations of SQL queries in web applications. The exploit can be used to read other data contained in the database, obtain admin-level access to web applications, and at its highest severity, allow direct access to the underlying web server itself.

In the **Task 1** section, in the **What does SQL stand for?** field, input the following:

Structured Query Language

Part 2 – What is a Database?

Read the documentation under the **Task 2** header. Some of the key takeaways are the following:

- databases are collections of data stored in an organized system
- they are controlled by DBMS (database management system)
- they are split into two types, **relational** (the focus of this workshop) and **non-relational**
  - **relational** databases are made of tables, and tables often have common information which makes all tables in a database related to one another
  - **non-relational** databases do not use tables or columns to store their data
- relational databases are made of up tables, and each table is subdivided into columns and rows
  - columns contain data specific to the table it belongs to. For example, a table named **users** may contain columns such as **id**, **username**, and **password**
  - rows are the entries in a table with data which corresponds to the columns in the table. For example, one row in the users table might contain the data **1**, **admin**, and **password123**, which correspond to the **id**, **username**, and **password** columns, respectively

Step 1

In the Task 2 section, under the **What is the acronym for the software that controls a database?** text, enter the following:

DBMS

Step 2

Under the **What is the name of the grid-like structure which holds the data?** text, enter the following:

table

Part 3 – What is SQL?

Read over the text in the **Task 3** section. Some of the key takeaways include the following:

- SQL (structured query language) is a programming language used for querying databases
- the SELECT command retrieves data, and must be used in reference to a specific table
- the WHERE clause is used to filter out returned data according to the criteria specified
- the LIMIT clause is used to restrict the number of rows returned to a specific amount
- the != operator is used to return data that doesn't match the specified argument
- the OR operator is used to return data if one of the two comparisons on either side of it apply
- the AND operator is used to return data if both of the comparisons on either side of it apply
- the LIKE operator is used to return data if the data matches a non-specific string or number
- the UNION statement allows the combination of search results from two or more tables, provided that the number of columns returned from all tables are equal
- the INSERT statement allows for the creation of new rows in a table
- the UPDATE statement allows for the modification of existing rows in a table
- the DELETE statement allows for the deletion of rows in a table

Step 1

In the **Task 3** section, under the **What SQL statement is used to retrieve data?** text, enter the following:

SELECT

Step 2

Under the **What SQL clause can be used to retrieve data from multiple tables?** text, enter the following:

UNION

Step 3

Under the What SQL statement is used to add data? text, enter the following:

INSERT

Part 4 – What is SQL Injection?

Read the documentation in the **Task 4** section. Some of the key takeaways include the following:

- SQL injection exploits can occur when user-provided input is included in SQL queries
- SQL injection allows the attacker to manipulate the SQL queries in such a way that arbitrary data from the database can be retrieved
- SQL commands are terminated with a semicolon (**;**) symbol
- SQL injection often makes use of the double dashes (--) symbols, which are used in many SQL languages to denote a comment (comments are ignored when SQL commands are run)
- there are major types of SQL injection: **in-band**, **blind**, and **out-of-band**

In the Task 4 section, under the **What character signifies the end of an SQL query?** text, enter the following:

;

Part 5 – In-Band SQLi

Read the documentation in the **Task 5** section. Some of the key takeaways include the following:

- **in-band** SQL injection is an exploit where the results of SQL queries can be read directly from the web application. It is considered the easiest type of SQL injection to exploit
- a sub-type of **in-band** SQL injection is **error-based** SQL injection, in which error messages from the database software are printed directly to the browser screen
- another sub-type is **union-based** SQL injection, which allows for retrieval of arbitrary database information, so long as the number of columns returned from the original SQL query is known

Step 1

Click on the green **Start Machine** button at the top-right corner of the **Task 5** header. The browser window should split into two panes, with the TryHackMe webpage on the left pane, and the web-based virtual machine pane appearing on the right. The virtual machine pane should take between 1-2 minutes to initialize.

Step 2

In the **Task 5** section, read the documentation under the header labelled **Practical**, and follow the instructions there to perform a union-based SQL injection attack. The main takeaways from this section are as follows:

- SQL injection vulnerabilities can sometimes be discovered by simply inserting a single quote (') or double quote (“) into an input field and observing whether an error message is returned
- After successfully output is achieved using a union-based attack, we can use the **database** SQL function to return the name of the database used by the web app, which is the first step in further attacks
- The next step is to include the database name in another attack, where we return all the table names in the database by accessing the **information_schema.tables** object
- Then we do the same for column names in any specific table by accessing the **information_schema.columns** object
- Finally, we can retrieve all data from a specific table using a union-based attack

Step 3

Under the **What is the flag after completing level 1?** text, input the following:

THM{SQL_INJECTION_3840}

Part 6 – Blind SQLi – Authentication Bypass

Read the documentation under the **Task 6** header, stopping at the **Practical** section. Some key takeaways from this section are as follows:

- **blind SQLi** is an attack performed on a web app which returns little to no feedback on whether injected SQL queries are successful or not
- authentication bypass is a type of blind SQLi attack that can gain access to a web app that requires a login, regardless of whether or not we have a valid username / password

Step 1

Read the documentation under the **Practical** header, and follow the instructions. The key takeaways are the following:

- a common authentication bypass string for web apps using MySQL software is `' OR 1=1;--`
- note the space at the end of the SQL string

Step 2

Under the **What is the flag after completing level two? (and moving to level 3)** text, input the following:

THM{SQL_INJECTION_9581}

Part 7 – Blind SQLi – Boolean Based

Step 1

Read the documentation in the under the **Task 7** section, stopping at the **Practical** header. Some of the key takeaways are the following:

- **boolean-based** SQL injection is an attack on a SQL system that only returns output if the injected SQL query returns True in a boolean context
- we will see from the practical portion of this task, that through use of boolean-based SQL injection, we can send iterative SQL boolean statements to a SQL service to determine the names of databases, tables, and columns

Step 2

Under the **Practical** header in **Task 7**, read the documentation and follow the instructions. Some of the key takeaways from this section include the following:

- we can perform a boolean-based SQLi attack on web apps where SQL input is used to output a true or false result
- boolean-based SQLi attacks make use of union-based queries, the SQL operand **like**, and the wildcard matching symbol (**%**) to determine valid names in the system
- boolean-based SQLi involves a lot of iterative SQL queries, so it's useful to write a programming script to perform boolean-based SQLi attacks

Step 3

Under the **What is the flag after completing level three?** text, input the following:

THM{SQL_INJECTION_1093}

Part 8 – Blind SQLi – Time Based

Step 1

Read the documentation in the under the **Task 8** section, and follow the instructions. Some of the key takeaways are the following:

- **time-based** SQL injection is a type of blind SQLi attack that is very similar to boolean-based SQL injection, with the exception that time-based SQL injection can be performed on web apps that do not supply any visual feedback on whether or not a SQL query succeeded or failed
- whether a time-based blind SQL injection is successful or not is determined by how long the SQL query takes to complete
- time-based SQL injection makes use of SQL union-based queries, the SQL **sleep** method, and similar to boolean-based SQL injection, uses the **like** operator and wildcard symbols (%) to determine names in the system

Step 2

Use the SQL injection string provided to perform a time-based SQLi attack on the application:

```
referrer=admin123' UNION SELECT SLEEP(5),2 where database() like
'%';--
```

One hint is that the naming scheme is very similar to the last level, and that database was named **sqli_three**. After obtaining the database name, we can use the following string to get the table name:

```
admin123' UNION SELECT SLEEP(5),2 FROM information_schema.tables
WHERE table_schema = 'sqli_four' and table_name like '%';--
```

Once again, remember that the databases follow the same patterns for names. The next string will enumerate column names:

```
admin123' UNION SELECT SLEEP(5),2 FROM information_schema.columns
WHERE table_schema = 'sqli_four' and table_name = 'users' and
column_name like '%';--
```

After discovering the relevant column names, use the following string to discover the user password:

```
admin123' UNION SELECT SLEEP(5),2 FROM users WHERE username='admin'
and password like '%';--
```

Step 3

Under the **What is the final flag after completing level four?** text, input the following:

THM{SQL_INJECTION_MASTER}

Part 9 – Out-of-Band SQLi

Read the documentation in the under the **Task 9** section. Some of the key takeaways are the following:

- Out-of-band (OOB) SQL injection is an uncommon vulnerability, because it relies on other software or specific settings enabled on the web application in order to work
- OOB SQL injection requires two communication channels: one to launch the attack (the vulnerable SQL system), and one to retrieve the output
- Some examples of services that can be used to retrieve OOB SQL injection output are HTTP/DNS

Under the **Name a protocol beginning with D that can be used to exfiltrate data from a database** text, enter the following:

DNS

Part 10 – Remediation

In cybersecurity, remediation is the process of modifying an app or system so that it is no longer vulnerable to an exploit. Here we will discuss remediation steps for applications to make them secure against SQL injection exploits.

Read the documentation in the under the **Task 10** section. Some of the key takeaways are the following:

- **prepared statements** allow developers to take user input and define it as a parameter that is inserted into the SQL statement, then processed by the SQL system instead of accepting user input as a modification of a SQL statement that is processed by the SQL system.
- **input validation** is a method of modifying user input. It can be used with an allow-list of permitted characters, or a deny-list of restricted characters. If user input includes characters outside of an allow-list or within a deny-list, then the input is modified in some way
- **escaping user input** is a method through which user-input that contain specials characters that could be used in SQL injection attacks (**'"$**) is modified so that they are treated as regular strings and not as special SQL characters

In the **Name a method of protecting yourself from an SQL Injection exploit** field, enter the following answer:

prepared statements

Summary

In this workshop, we learned a lot about different types of SQL injection (SQLi) attacks

In-band SQLi

In-band SQLi attacks are attacks in which the output of SQL queries is returned back to the web app, which makes this type of SQL injection the easiest to exploit when compared to **blind** or **out-of-band** SQLi attacks.

A sub-type of in-band SQLi attacks is **error-based** SQLi, in which SQL error messages are returned to the app if invalid SQL statements are sent to the SQL system, which is usually a indication of a web app's vulnerability to SQLi attacks.

Another method through which in-band SQLi can be exploited is **union-based** SQL queries, which allow for the output of SQL data in web apps

Blind SQLi

Blind SQLi attacks are attacks in which the output of SQL queries is not returned to the web app, which makes these attacks more difficult to perform, mostly due to the increased number of SQL commands required to acquire relevant information from the database.

Authentication bypass is a type of blind SQLi attack which allows authenticated access to a web application without the need for a valid username / password combination.

Boolean-based blind SQLi is an attack where info from a SQL database is iteratively gained through numerous SQL commands that cycle through a character sets to determine the names of valid data in the system. This attack relies on the SQL system responding with a valid (boolean true) response when a character in a name is guessed successfully, and not responding at all when a character in a name is guessed incorrectly.

Time-based blind SQLi is an attack that is similar to boolean-based SQLi in almost every way, except that this attack is performed in web app environments where the SQL system does not return any output to the web app, so the attack must instead rely on system response times to determine whether a guessed character is valid or not.

Out-of-Band SQLi

Out-of-band (OOB) SQLi is an attack where SQL injection is exploited in a web app, but the output of SQL queries is sent to a service outside of the web app, for instance to an HTTP / DNS server which the attacker controls. Due to the specific conditions which must be met for a web app to be vulnerable to OOB SQLi, it is considered the most uncommon form of SQLi attack.

This is the last workshop in the web exploitation course for HackerFrogs AfterSchool. More web exploitation topics will be explored in other HackerFrogs programs, and additional learning materials on web exploitation can be found at the on various platforms, such as TryHackMe and Portswigger:

https://tryhackme.com
https://portswigger.net/web-security

Extra Credit

If you're looking for more SQL injection fun, there's more materials on SQL injection at the following links:

Rana Khalil's Web Security Academy SQLi Playlist

https://www.youtube.com/watch?v=1nJgupaUPEQ&list=PLuyTk2_mYISLaZC4fVqDuW_hOk0dd5rlf

TryHackMe's SQL Injection Lab Room

https://tryhackme.com/room/sqlilab

Extra Research

The following articles will help us further understand some of the concepts covered in this workshop:

Portswigger Academy's section on SQL Injection:
https://portswigger.net/web-security/sql-injection

Until next time, HackerFrogs!