Lab - Attacking a mySQL Database

# Objectives

In this lab, you will view a PCAP file from a previous attack against a SQL database.

Part 1: Open Wireshark and load the PCAP file.

Part 2: View the SQL Injection Attack.

Part 3: The SQL Injection Attack continues…

Part 4: The SQL Injection Attack provides system information.

Part 5: The SQL Injection Attack and Table Information

Part 6: The SQL Injection Attack Concludes.

# Background / Scenario

SQL injection attacks allow malicious hackers to type SQL statements in a web site and receive a response from the database. This allows attackers to tamper with current data in the database, spoof identities, and miscellaneous mischief.

A PCAP file has been created for you to view a previous attack against a SQL database. In this lab, you will view the SQL database attacks and answer the questions.

# Required Resources

* CyberOps Workstation virtual machine

# Instructions

You will use Wireshark, a common network packet analyzer, to analyze network traffic. After starting Wireshark, you will open a previously saved network capture and view a step by step SQL injection attack against a SQL database.

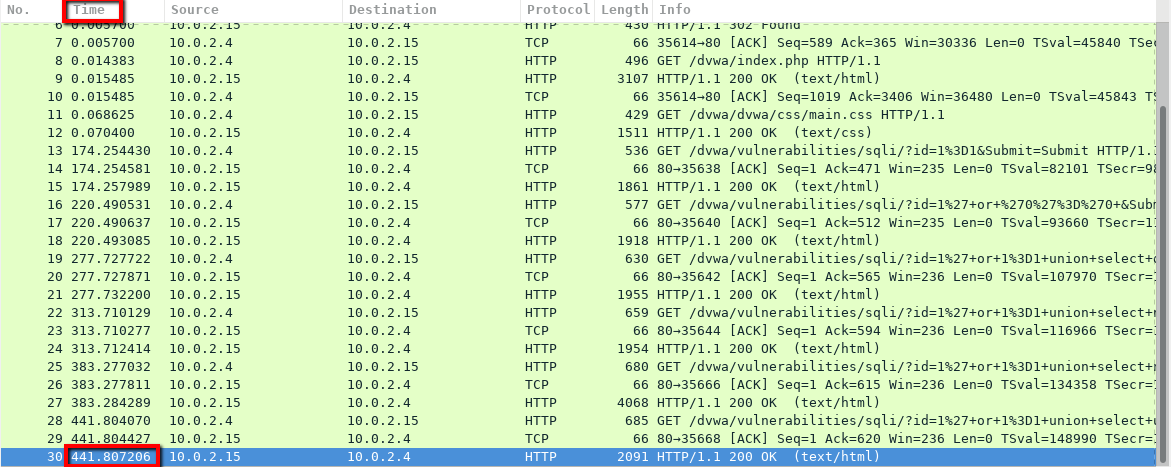
## Open Wireshark and load the PCAP file.

The Wireshark application can be opened using a variety of methods on a Linux workstation.

* + 1. Start the CyberOps Workstation VM.
    2. Click **Applications > CyberOPS >** **Wireshark** on the desktop and browse to the Wireshark application.
    3. In the Wireshark application, click **Open** in the middle of the application under Files.
    4. Browse through the **/home/analyst/** directory and search for **lab.support.files**. In the **lab.support.files** directory and open the **SQL\_Lab.pcap** file.



* + 1. The PCAP file opens within Wireshark and displays the captured network traffic. This capture file extends over an 8-minute (441 second) period, the duration of this SQL injection attack.



### Question:

What are the two IP addresses involved in this SQL injection attack based on the information displayed?

10.0.2.4 and 10.0.2.15

## View the SQL Injection Attack.

In this step, you will be viewing the beginning of an attack.

* + 1. Within the Wireshark capture, right-click line 13 and select **Follow** > **HTTP Stream**. Line 13 was chosen because it is a GET HTTP request. This will be very helpful in following the data stream as the application layers sees it and leads up to the query testing for the SQL injection.

A screenshot of a computer

Description automatically generated

The source traffic is shown in red. The source has sent a GET request to host 10.0.2.15. In blue, the destination device is responding back to the source.

A screenshot of a computer

Description automatically generated

* + 1. In the **Find** field, enter **1=1**. Click **Find Next**.

A screenshot of a computer

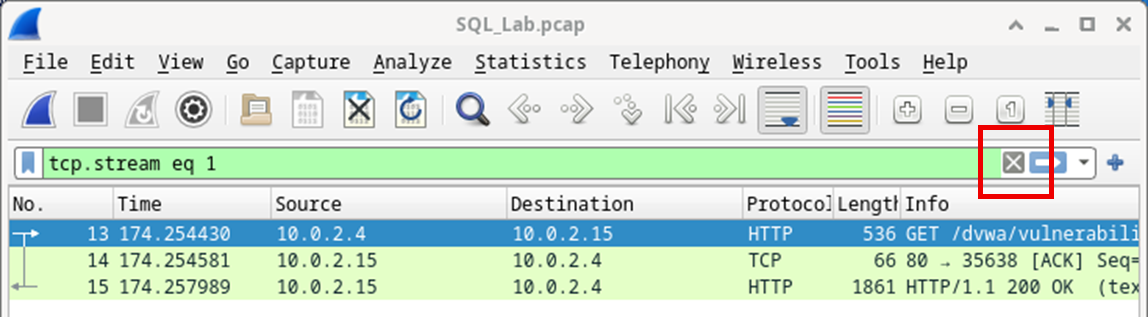
Description automatically generated

* + 1. The attacker has entered a query (1=1) into a UserID search box on the target 10.0.2.15 to see if the application is vulnerable to SQL injection. Instead of the application responding with a login failure message, it responded with a record from a database. The attacker has verified they can input an SQL command and the database will respond. The search string 1=1 creates an SQL statement that will be always true. In the example, it does not matter what is entered into the field, it will always be true.

A screenshot of a computer

Description automatically generated

* + 1. Close the Follow HTTP Stream window.
    2. Click **Clear display filter** to display the entire Wireshark conversation.



## The SQL Injection Attack continues...

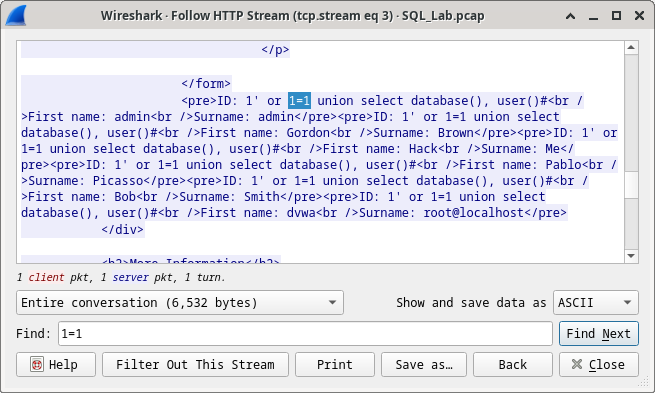
In this step, you will be viewing the continuation of an attack.

* + 1. Within the Wireshark capture, right-click line 19, and click **Follow** > **HTTP Stream**.
    2. In the **Find** field, enter **1=1**. Click **Find Next**.

A screenshot of a computer

Description automatically generated

* + 1. The attacker has entered a query (1’ or 1=1 union select database(), user()#) into a UserID search box on the target 10.0.2.15. Instead of the application responding with a login failure message, it responded with the following information:



The database name is **dvwa** and the database user is **root@localhost**. There are also multiple user accounts being displayed.

* + 1. Close the Follow HTTP Stream window.
    2. Click **Clear display filter** to display the entire Wireshark conversation.

## The SQL Injection Attack provides system information.

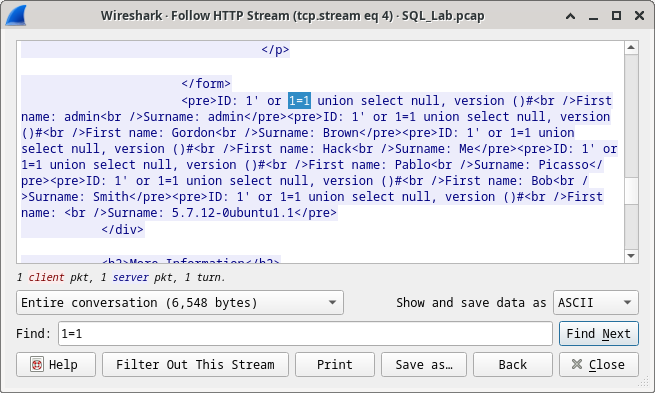
The attacker continues and starts targeting more specific information.

* + 1. Within the Wireshark capture, right-click line 22 and select **Follow** > **HTTP Stream**. In red, the source traffic is shown and is sending the GET request to host 10.0.2.15. In blue, the destination device is responding back to the source.
    2. In the **Find** field, enter **1=1**. Click **Find Next**.

A screenshot of a computer

Description automatically generated

* + 1. The attacker has entered a query (1’ or 1=1 union select null, version ()#) into a UserID search box on the target 10.0.2.15 to locate the version identifier. Notice how the version identifier is at the end of the output right before the </pre>.</div> closing HTML code.



### Question:

What is the version?

5.7.12-0ubuntu1.1

A screenshot of a computer

Description automatically generated

* + 1. Close the Follow HTTP Stream window.
    2. Click **Clear display filter** to display the entire Wireshark conversation.

## The SQL Injection Attack and Table Information.

The attacker knows that there is a large number of SQL tables that are full of information. The attacker attempts to find them.

* + 1. Within the Wireshark capture, right-click on line 25 and select **Follow** > **HTTP Stream**. The source is shown in red. It has sent a GET request to host 10.0.2.15. In blue, the destination device is responding back to the source.
    2. In the **Find** field, enter **users**. Click **Find Next**.
    3. The attacker has entered a query (1’or 1=1 union select null, table\_name from information\_schema.tables#) into a UserID search box on the target 10.0.2.15 to view all the tables in the database. This provides a huge output of many tables, as the attacker specified “null” without any further specifications.

A screen shot of a computer

Description automatically generated

### Question:

What would the modified command of (**1' OR 1=1 UNION SELECT null, column\_name FROM INFORMATION\_SCHEMA.columns WHERE table\_name='users'**) do for the attacker?

Show all column name of table ***users*** as the result (Surname).

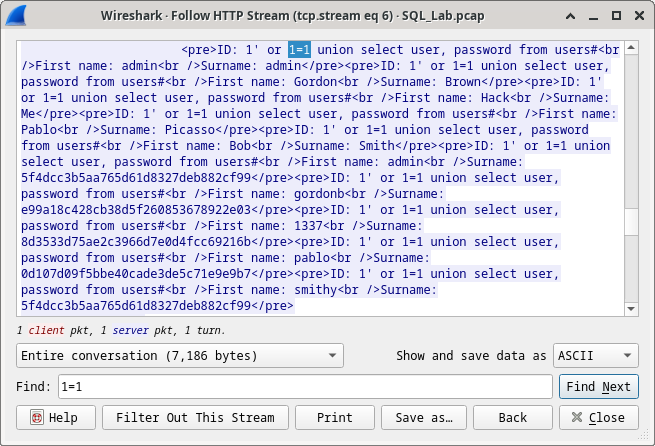
* + 1. Close the Follow HTTP Stream window.
    2. Click **Clear display filter** to display the entire Wireshark conversation.

## The SQL Injection Attack Concludes.

The attack ends with the best prize of all; password hashes.

* + 1. Within the Wireshark capture, right-click line 28 and select **Follow** > **HTTP Stream**. The source is shown in red. It has sent a GET request to host 10.0.2.15. In blue, the destination device is responding back to the source.
    2. Click **Find** and type in **1=1**. Search for this entry. When the text is located, click **Cancel** in the Find text search box.

The attacker has entered a query (1’or 1=1 union select user, password from users#) into a UserID search box on the target 10.0.2.15 to pull usernames and password hashes!



### Question:

Which user has the password hash of 8d3533d75ae2c3966d7e0d4fcc69216b?

1337

* + 1. Using a website such as <https://crackstation.net/>, copy the password hash into the password hash cracker and get cracking.

### Question:

What is the plain-text password?

charley

A screenshot of a computer

Description automatically generated

* + 1. Close the Follow HTTP Stream window. Close any open windows.

# Reflection Questions

* 1. What is the risk of having platforms use the SQL language?

It can be attack with SQL Injection.

* 1. Browse the internet and perform a search on “prevent SQL injection attacks”. What are 2 methods or steps that can be taken to prevent SQL injection attacks?

Use of Prepared Statements (with Parameterized Queries)

Use of Properly Constructed Stored Procedures

A screenshot of a computer

Description automatically generated

End of document