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.

Part 1: Open Wireshark and load the PCAP file.

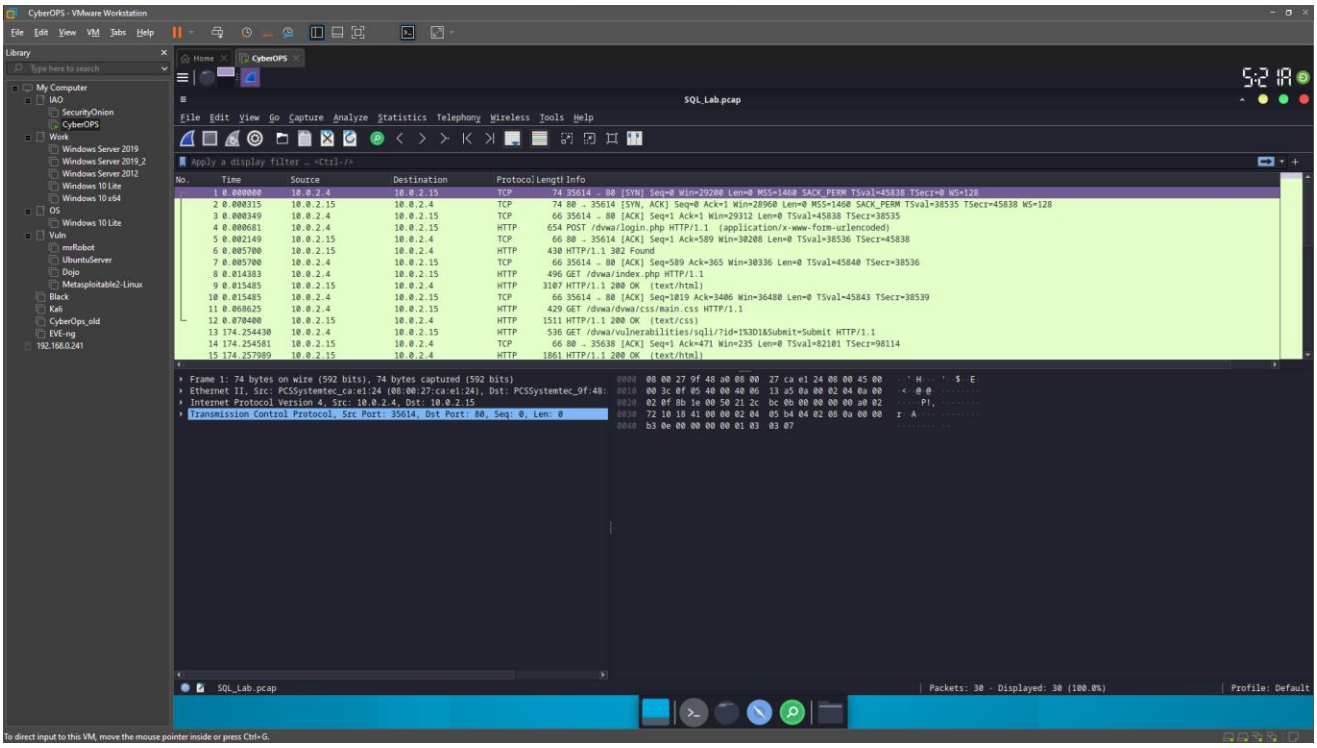
The Wireshark application can be opened using a variety of methods on a Linux workstation.

a. Start the CyberOps Workstation VM.

b. Click **Applications > CyberOPS > Wireshark** on the desktop and browse to the Wireshark application.

c. In the Wireshark application, click **Open** in the middle of the application under Files.

d. 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.



e. 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.

No.	Time	Source	Destination	Protocol	Length	Info
7	0.005700	10.0.2.4	10.0.2.15	TCP	66	35614→80 [ACK] Seq=589 Ack=365 Win=30336 Len=0 TSval=45840 TSe
8	0.014383	10.0.2.4	10.0.2.15	HTTP	496	GET /dvwa/index.php HTTP/1.1
9	0.015485	10.0.2.15	10.0.2.4	HTTP	3107	HTTP/1.1 200 OK (text/html)
10	0.015485	10.0.2.4	10.0.2.15	TCP	66	35614→80 [ACK] Seq=1019 Ack=3406 Win=36480 Len=0 TSval=45843 TS
11	0.068625	10.0.2.4	10.0.2.15	HTTP	429	GET /dvwa/dvwa/css/main.css HTTP/1.1
12	0.070400	10.0.2.15	10.0.2.4	HTTP	1511	HTTP/1.1 200 OK (text/css)
13	174.254430	10.0.2.4	10.0.2.15	HTTP	536	GET /dvwa/vulnerabilities/sqli/?id=1%3D1&Submit=Submit HTTP/1.1
14	174.254581	10.0.2.15	10.0.2.4	TCP	66	80→35638 [ACK] Seq=1 Ack=471 Win=235 Len=0 TSval=82101 TSecr=9
15	174.257989	10.0.2.15	10.0.2.4	HTTP	1861	HTTP/1.1 200 OK (text/html)
16	220.490531	10.0.2.4	10.0.2.15	HTTP	577	GET /dvwa/vulnerabilities/sqli/?id=1%27+or+%270%27%3D%270+&Subr
17	220.490637	10.0.2.15	10.0.2.4	TCP	66	80→35640 [ACK] Seq=1 Ack=512 Win=235 Len=0 TSval=93660 TSecr=1
18	220.493085	10.0.2.15	10.0.2.4	HTTP	1918	HTTP/1.1 200 OK (text/html)
19	277.727722	10.0.2.4	10.0.2.15	HTTP	630	GET /dvwa/vulnerabilities/sqli/?id=1%27+or+1%3D1+union+select+
20	277.727871	10.0.2.15	10.0.2.4	TCP	66	80→35642 [ACK] Seq=1 Ack=565 Win=236 Len=0 TSval=107970 TSecr=
21	277.732200	10.0.2.15	10.0.2.4	HTTP	1955	HTTP/1.1 200 OK (text/html)
22	313.710129	10.0.2.4	10.0.2.15	HTTP	659	GET /dvwa/vulnerabilities/sqli/?id=1%27+or+1%3D1+union+select+
23	313.710277	10.0.2.15	10.0.2.4	TCP	66	80→35644 [ACK] Seq=1 Ack=594 Win=236 Len=0 TSval=116966 TSecr=
24	313.712414	10.0.2.15	10.0.2.4	HTTP	1954	HTTP/1.1 200 OK (text/html)
25	383.277032	10.0.2.4	10.0.2.15	HTTP	680	GET /dvwa/vulnerabilities/sqli/?id=1%27+or+1%3D1+union+select+
26	383.277811	10.0.2.15	10.0.2.4	TCP	66	80→35666 [ACK] Seq=1 Ack=615 Win=236 Len=0 TSval=134358 TSecr=
27	383.284289	10.0.2.15	10.0.2.4	HTTP	4068	HTTP/1.1 200 OK (text/html)
28	441.804070	10.0.2.4	10.0.2.15	HTTP	685	GET /dvwa/vulnerabilities/sqli/?id=1%27+or+1%3D1+union+select+
29	441.804427	10.0.2.15	10.0.2.4	TCP	66	80→35668 [ACK] Seq=1 Ack=620 Win=236 Len=0 TSval=148990 TSecr=
30	441.807206	10.0.2.15	10.0.2.4	HTTP	2091	HTTP/1.1 200 OK (text/html)

Question:

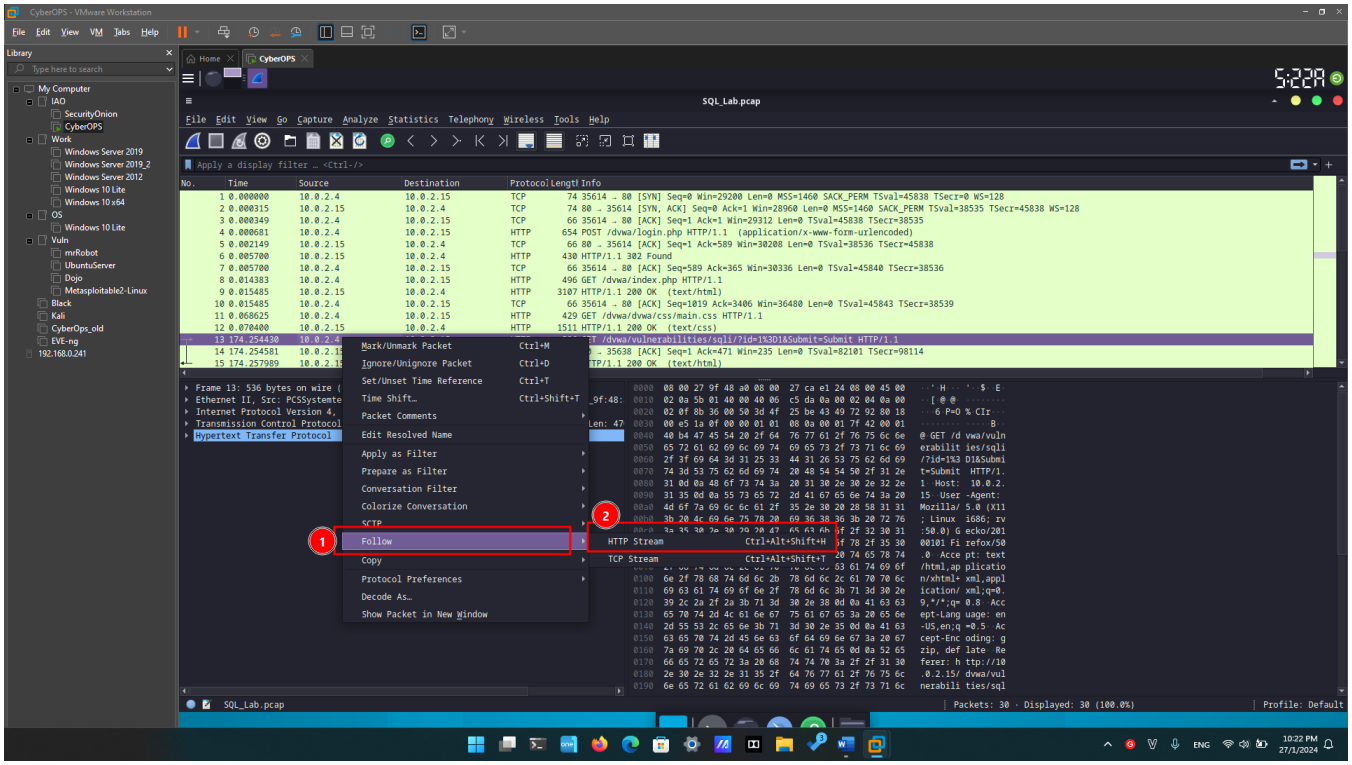
What are the two IP addresses involved in this SQL injection attack based on the information displayed?

10.0.2.4 and 10.0.2.15

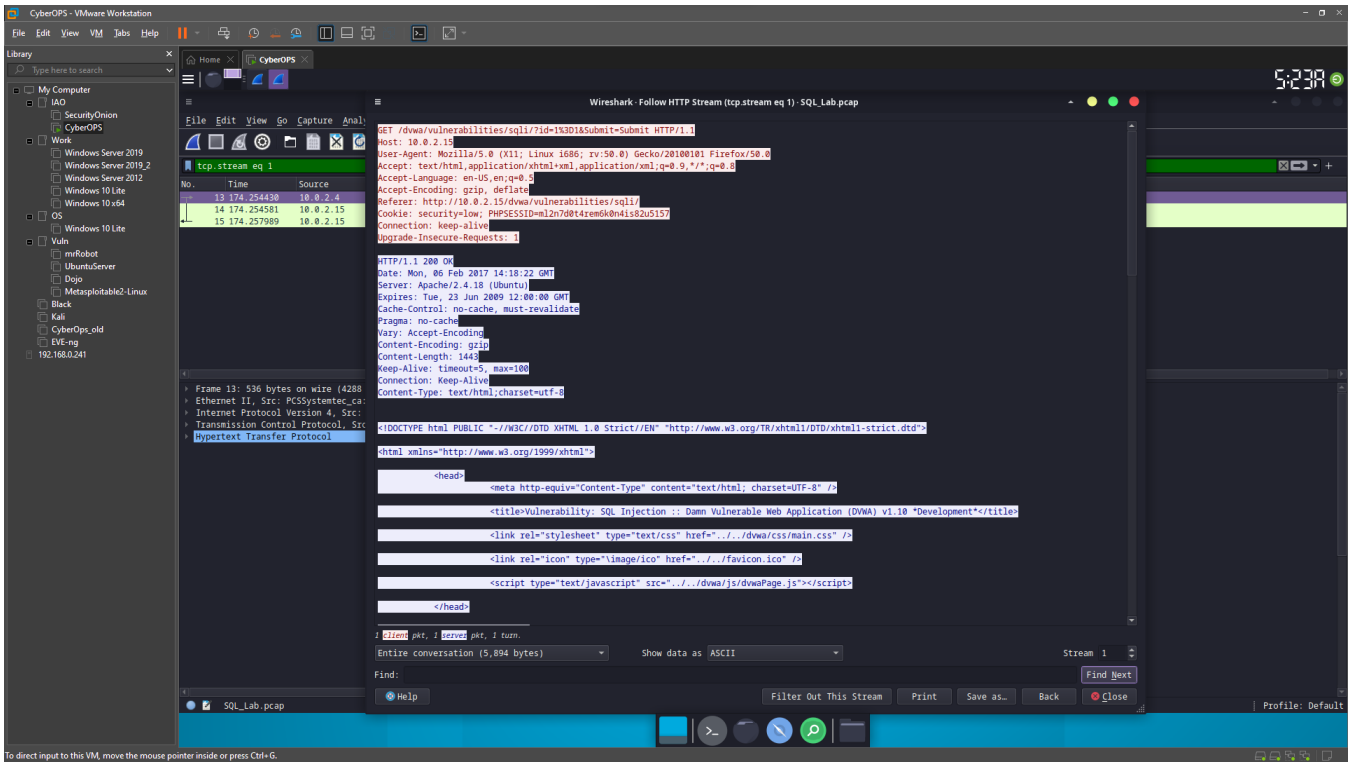
Part 2: View the SQL Injection Attack.

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

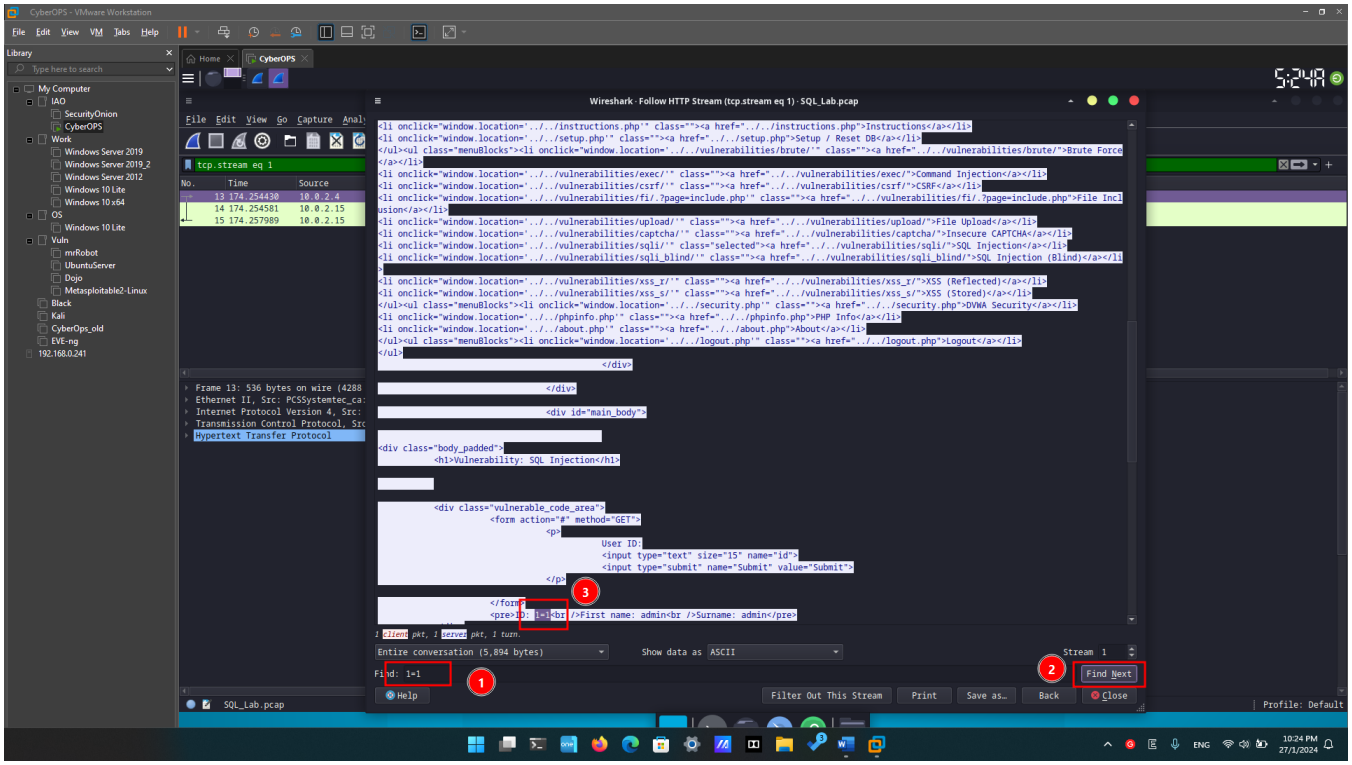
a. Within the Wireshark capture, right-click line 13 and select **Follow** > **HTTP Stream**. Line 13 was chosen because it is a GET HTTP request. This will be very helpful in following the data stream as the application layers sees it and leads up to the query testing for the SQL injection.



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.

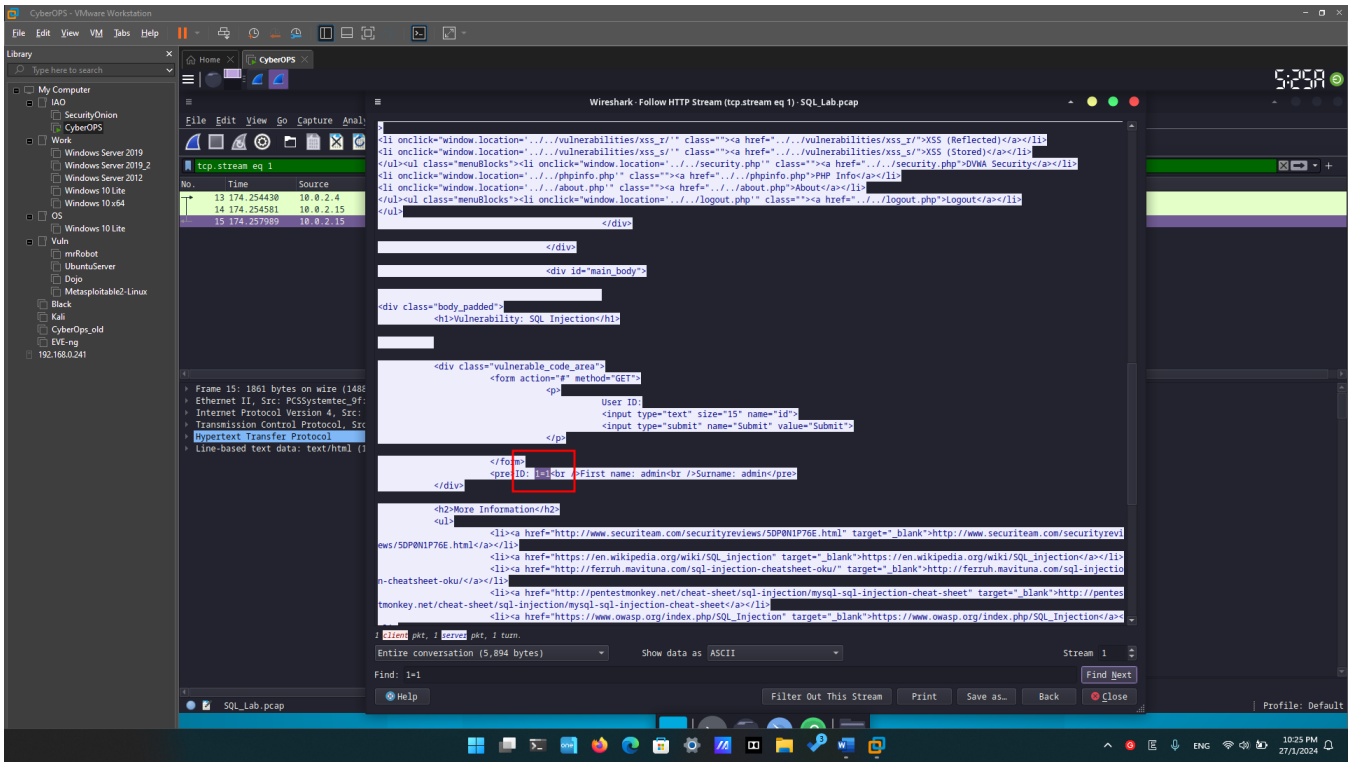


b. In the **Find** field, enter **1=1**. Click **Find Next**.



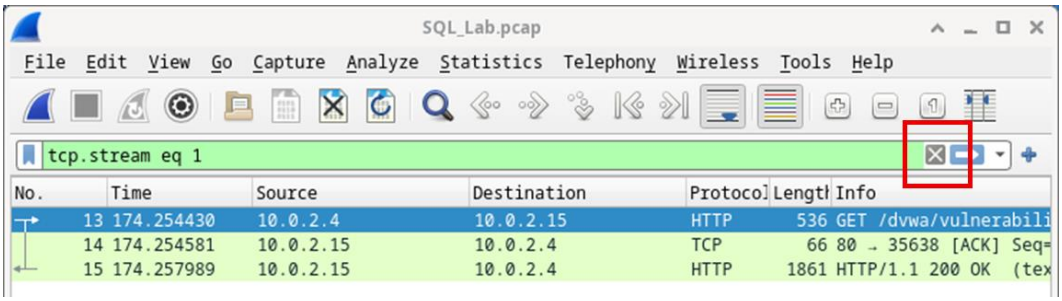
c. The attacker has entered a query (1=1) into a UserID search box on the target 10.0.2.15 to see if the application is vulnerable to SQL injection. Instead of the application responding with a login failure message, it responded with a record from a database. The attacker has verified they can input 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.



d. Close the Follow HTTP Stream window.

e. Click **Clear display filter** to display the entire Wireshark conversation.

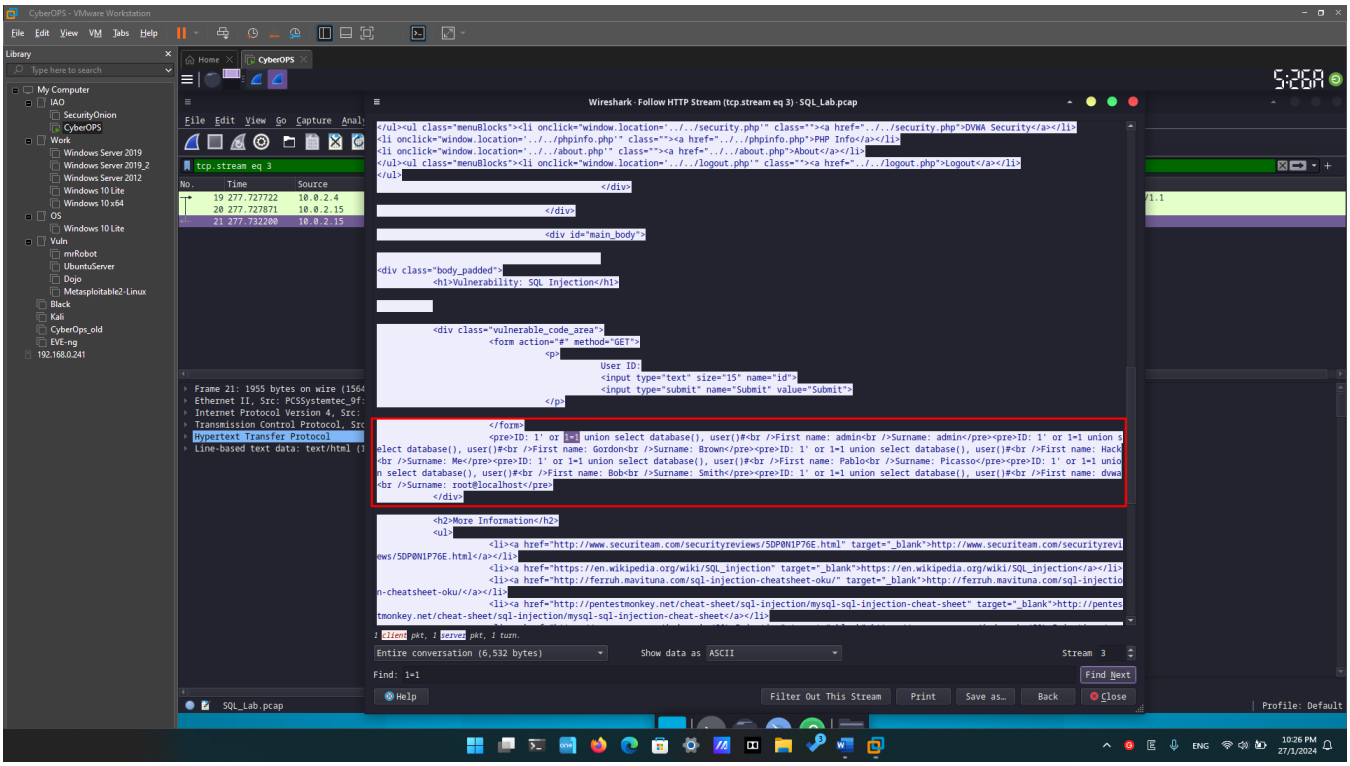


Part 3: The SQL Injection Attack continues...

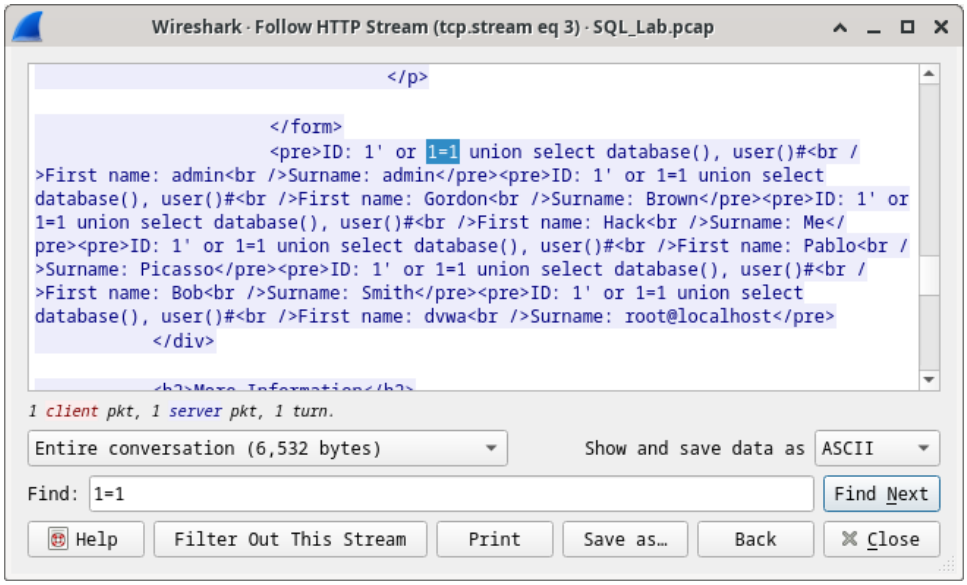
In this step, you will be viewing the continuation of an attack.

a. Within the Wireshark capture, right-click line 19, and click **Follow** > **HTTP Stream**.

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



c. 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.

d. Close the Follow HTTP Stream window.

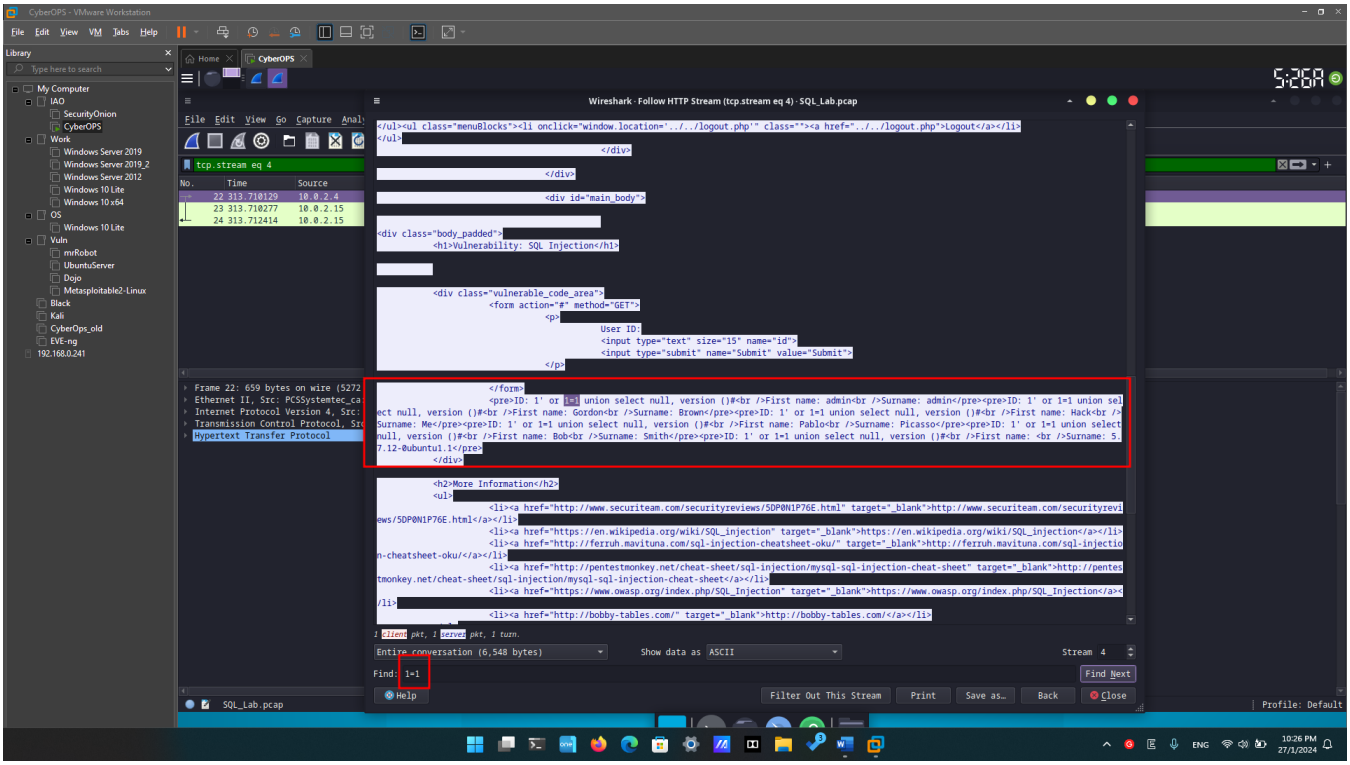
e. Click **Clear display filter** to display the entire Wireshark conversation.

Part 4: The SQL Injection Attack provides system information.

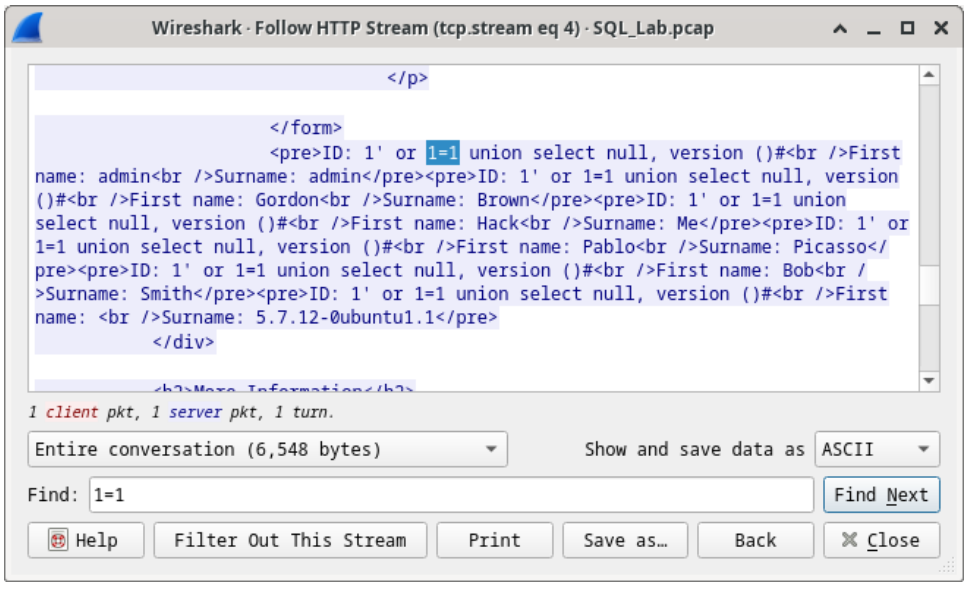
The attacker continues and starts targeting more specific information.

a. 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.

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



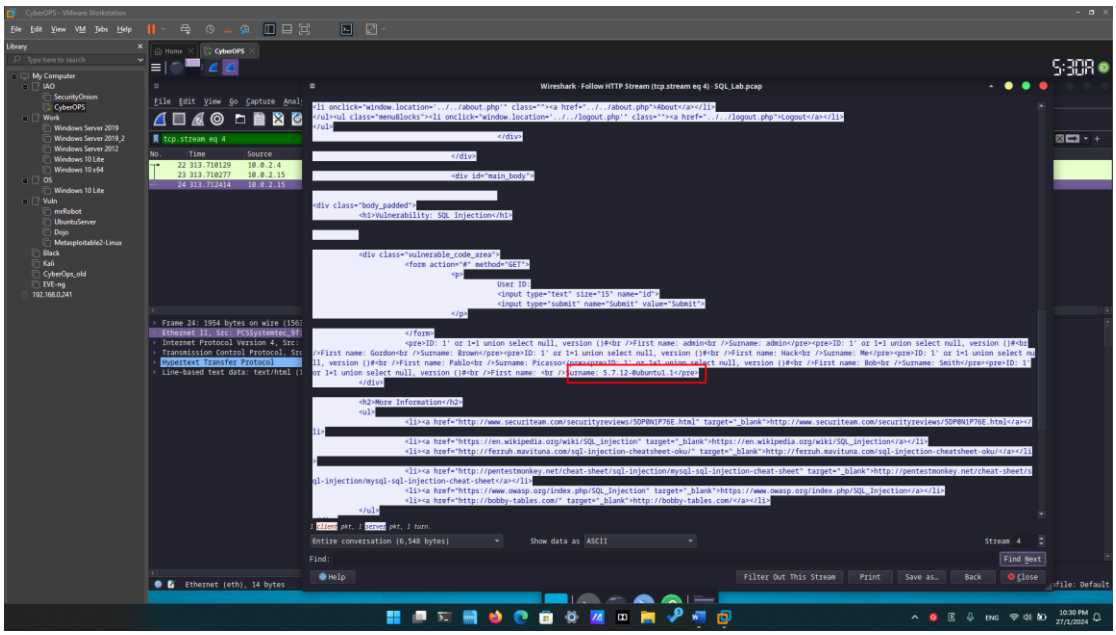
c. The attacker has entered a query (1' or 1=1 union select null, version ()#) into a UserID search box on the target 10.0.2.15 to locate the version identifier. Notice how the version identifier is at the end of the output right before the </pre>. closing HTML code.



Question:

What is the version?

5.7.12-0ubuntu1.1



d. Close the Follow HTTP Stream window.

e. Click **Clear display filter** to display the entire Wireshark conversation.

Part 5: The SQL Injection Attack and Table Information.

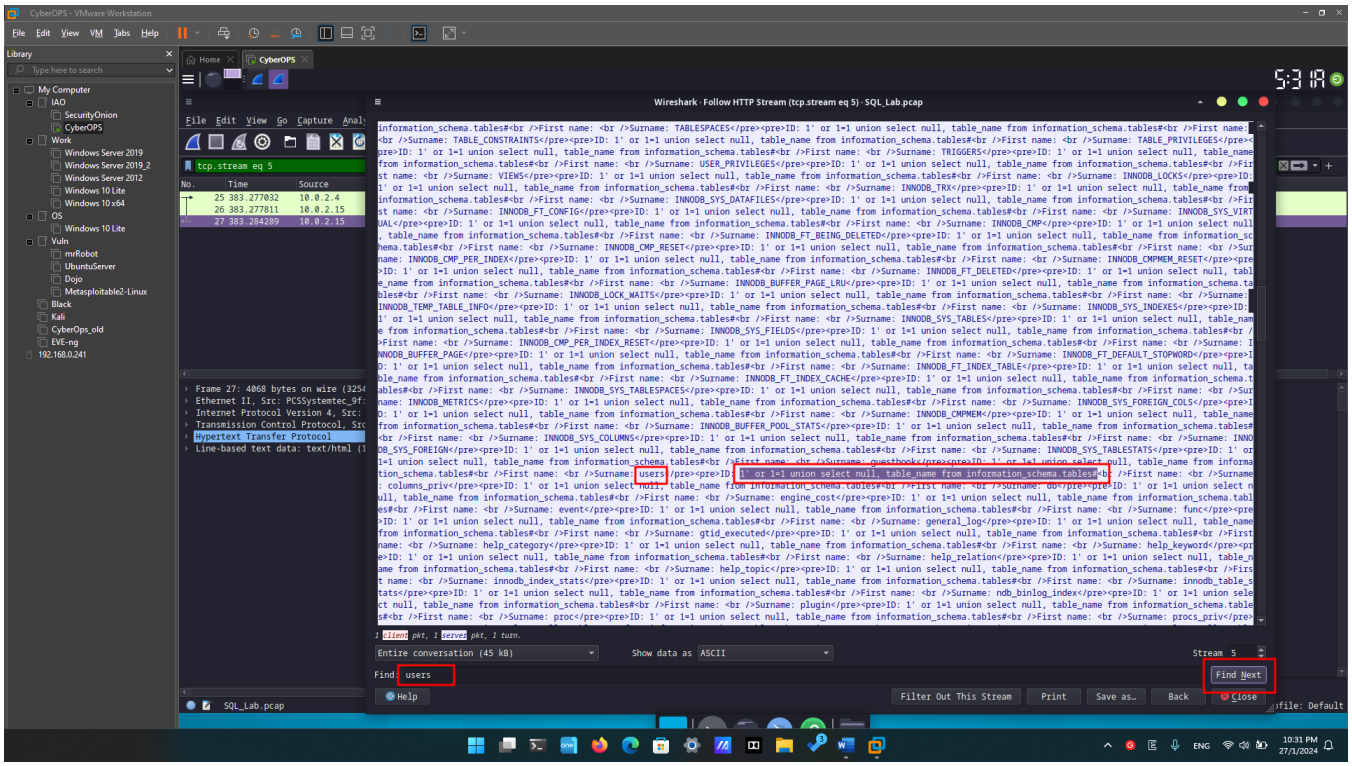
The attacker knows that there is a large number of SQL tables that are full of information. The attacker attempts to find them.

a. 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.

b. In the **Find** field, enter **users**. Click **Find Next**.

c. 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.



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).

d. Close the Follow HTTP Stream window.

e. Click **Clear display filter** to display the entire Wireshark conversation.

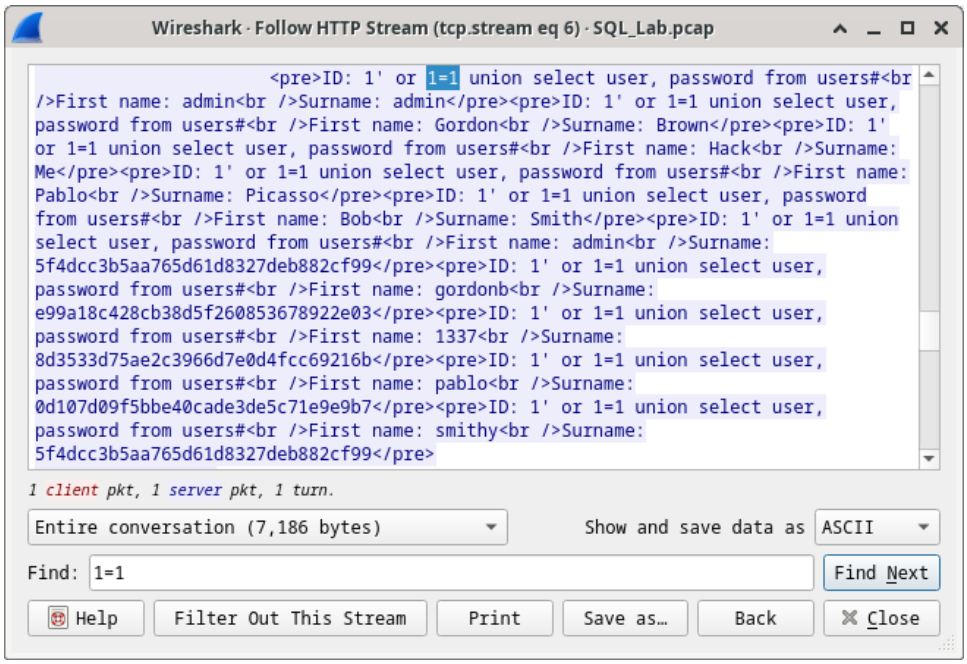
Part 6: The SQL Injection Attack Concludes.

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

a. 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.

b. 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?

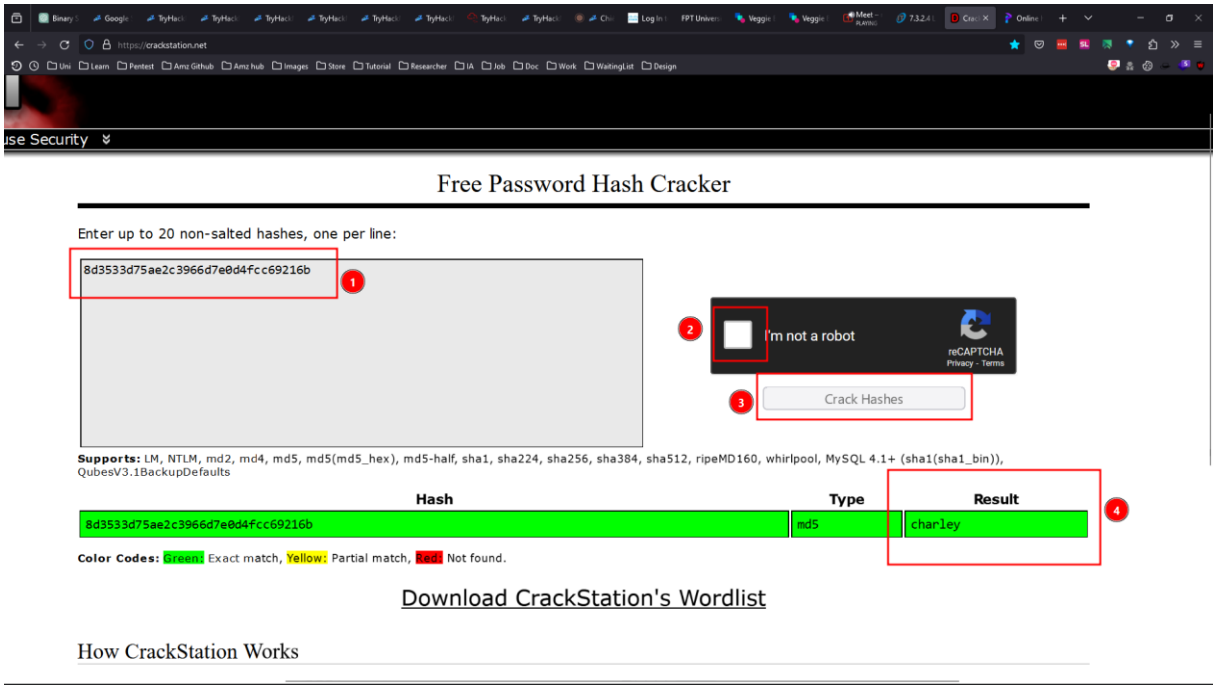
1337

c. 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



d. Close the Follow HTTP Stream window. Close any open windows.

Reflection Questions

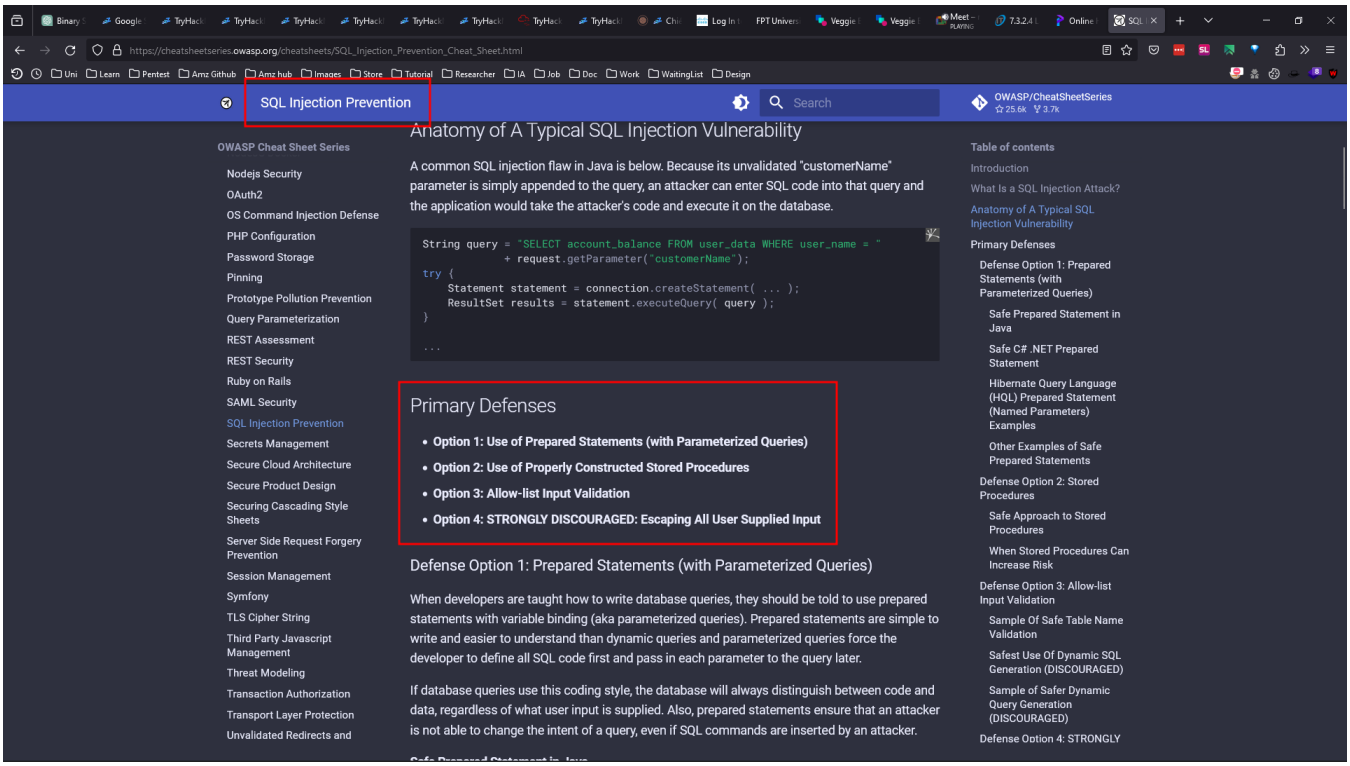
1. What is the risk of having platforms use the SQL language?

   It can be attack with SQL Injection.

2. 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



End of document