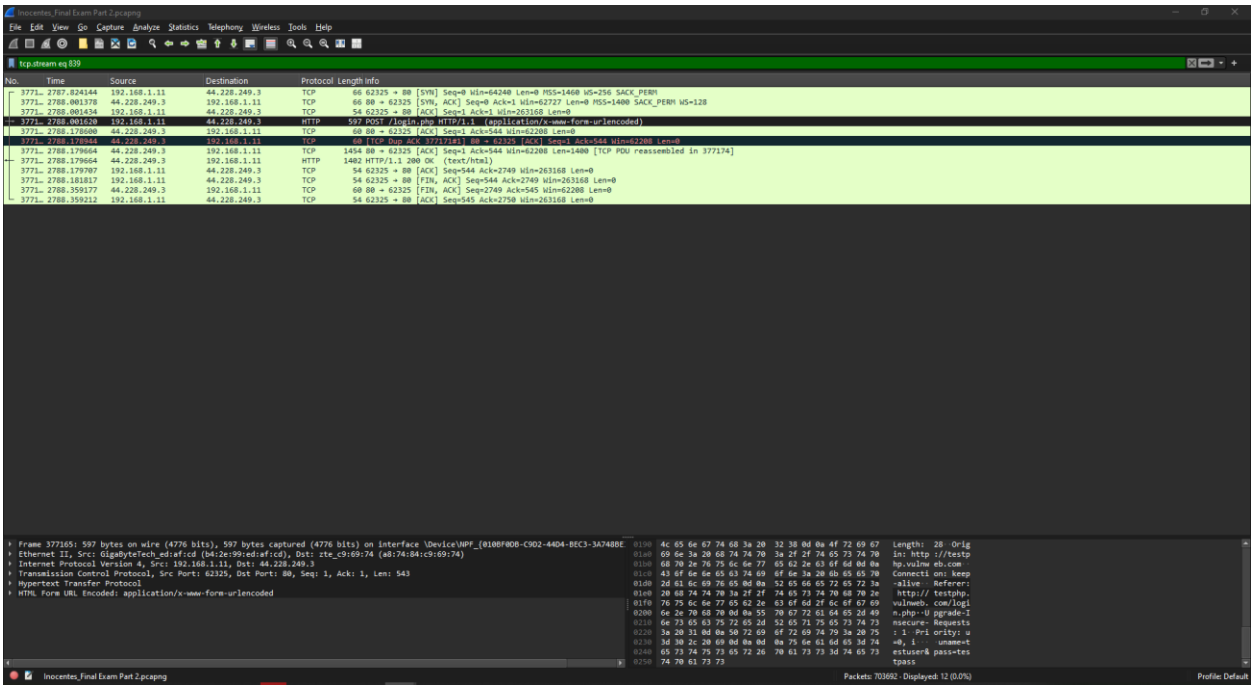
Inocentes, Raebv Lielmo V.

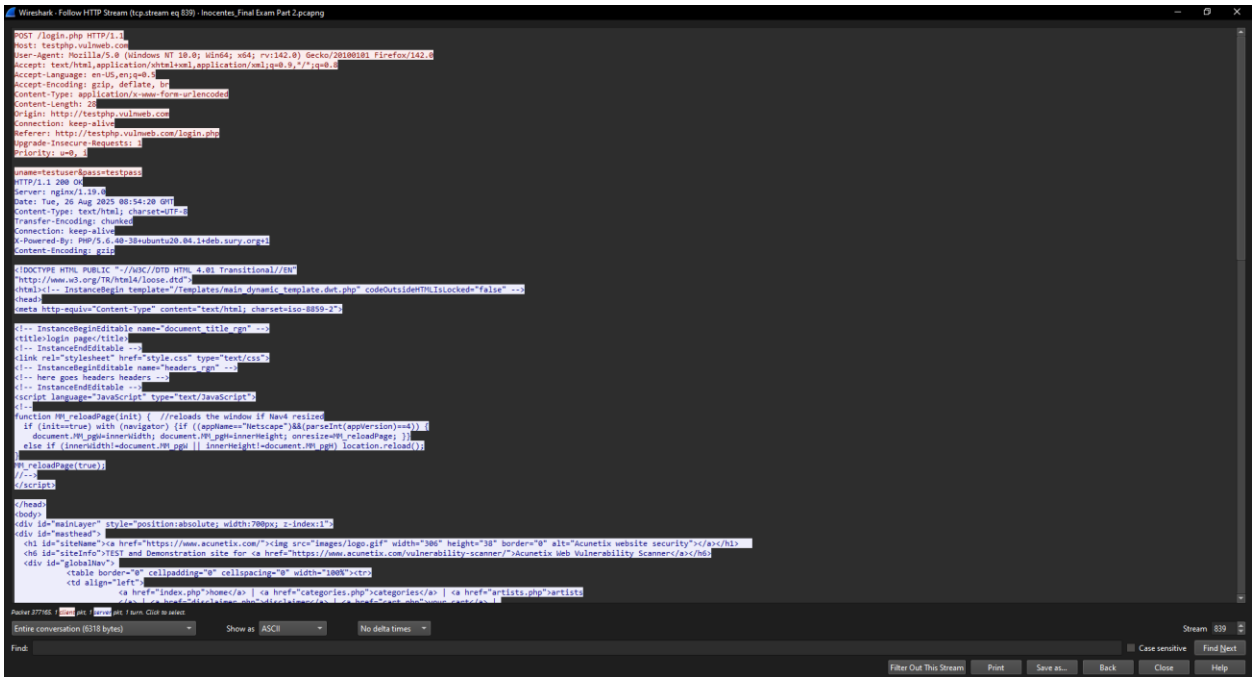
BSIT 3-2 August 26, 2025

Website Vulnerability Detection using Wireshark

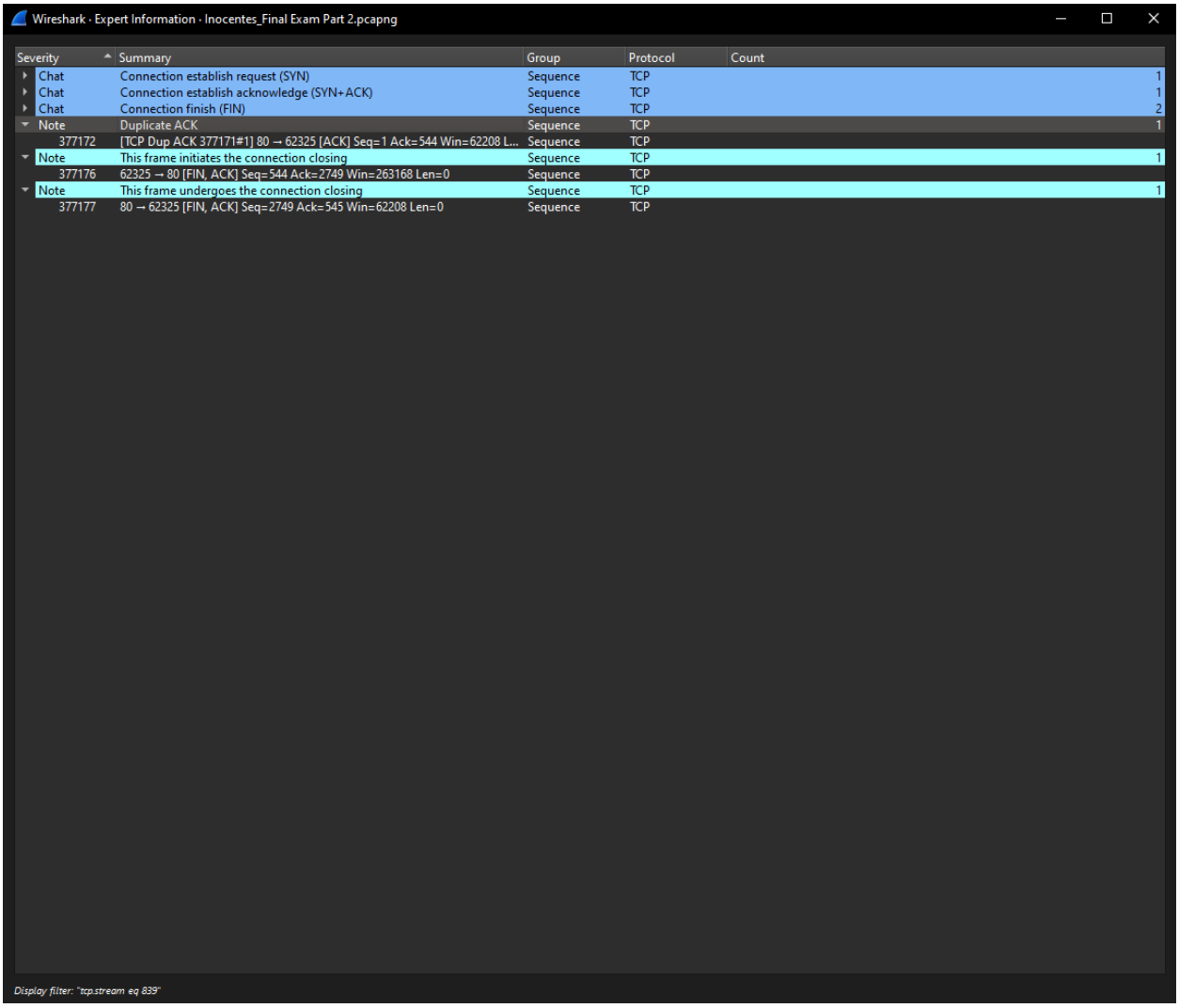
Screenshots of Wireshark captures:

TCP Stream:





Expert Information:



Guide Questions (Answer in your report):

1. **What is the difference between HTTP and HTTPS in terms of packet security?**

-HTTPS is more secure than HTTP in terms of packet security. With HTTP, the data you send, like passwords or messages, can be read easily if someone manages to intercept it since it's not protected just like the Wireshark Activity. While with HTTPS, the "S" stands for secure that's why your data is encrypted which means it's turned into a secret code while traveling. Even if hackers get the packets, they won't understand it. Basically, HTTP is like sending a postcard that anyone can read, while HTTPS is like sending a locked box that only the receiver can open. That's why most websites today use HTTPS, especially ones where you log in or make payments. It just keeps your data way safer.

2. **Were you able to see any sensitive information (e.g., username/password, cookies) in plain text? Provide evidence.**

-Yes, I saw sensitive information visible in plain text by applying the filter "**http.request.method == "POST""** and by following the TCP stream in Wireshark, I was able to see the login request:

```
POST /login.php HTTP/1.1
Host: testphp.vulnweb.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:142.0) Gecko/20100101 Firefox/142.0
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, br
Content-Type: application/x-www-form-urlencoded
Content-Length: 28
Origin: http://testphp.vulnweb.com
Connection: keep-alive
Referer: http://testphp.vulnweb.com/login.php
Upgrade-Insecure-Requests: 1
Priority: u=0, i

uname=testuser&pass=testpass
```

This shows that the request containing both the username and password were transmitted without encryption.

3. **What vulnerabilities can you conclude from the captured packets?**

   -I think there are many present vulnerabilities in this site. One of the obvious vulnerabilities is that the login credentials are being sent in plaintext, making them very easy for an attacker to intercept. Since the site uses regular HTTP instead of HTTPS, none of the traffic is encrypted which means not just usernames and passwords but also cookies and session data can be stolen by anyone sniffing the network.

   Another vulnerability that I noticed is that the server versions are leaking in response.

```
HTTP/1.1 200 OK
Server: nginx/1.19.0
Date: Tue, 26 Aug 2025 08:08:03 GMT
Content-Type: text/html; charset=UTF-8
Transfer-Encoding: chunked
Connection: keep-alive
X-Powered-By: PHP/5.6.40-38+ubuntu20.04.1+deb.sury.org+1
```

The attacker can take advantage of this by looking at the possible vulnerabilities that these versions have and thus, exploiting them if any of them exist.

Looking at the request again, it looks like the credentials sent are passed directly to the backend without much protection. This could open the site up to SQL injection or command injection, which are common issues in PHP web apps.

4. **If this were a real penetration test, what security recommendations would you give to the website owner?**

   If this were a real-world penetration test, my first recommendation would be to enable HTTPS/TLS on the website so that all traffic between the client and server is encrypted. This would prevent attackers from being able to read sensitive information in transit. Additionally, the website should make sure that authentication is handled securely by hashing the passwords on the server side instead of storing or transmitting them in plain text. Cookies should also be protected by adding the Secure and HttpOnly flags to reduce the chance of session hijacking. It would also be a good idea to implement proper security headers such as Strict-Transport-Security and X-Frame-Options to defend against other types of web attacks. Finally, the input fields should be validated and sanitized properly to protect against SQL injection or other injection-based attacks. The server should be configured to hide or minimize version details in HTTP response headers to reduce the attack surface of the attackers. This can be done by disabling or modifying header output in the server configuration