CyberSecurity Lab4

Soumen Kumar
B22ES006

Task 1:
In this Task, you will capture network traffic to explore how credentials are transmitted over different protocols. You will log in to a local FTP server (10.40.0.103) and an HTTP(http://10.40.0.103) server using the credentials: username test1 and password test1. Using Wireshark, you'll observe how these protocols transmit information in plaintext, making them vulnerable to interception. In contrast, you'll compare this with an HTTPS connection, where encryption protects the login details during transmission. This exercise highlights the importance of using secure protocols like HTTPS and FTPS over their insecure Counterparts.

Solution:

Using FTP Protocol

ftp

No.	Time	Source	Destination	Protocol	Length	Info
876	16.778548	10.40.0.103	10.23.24.150	FTP	74	Response: 220 (vsFTPd 3.0.5)
877	16.780954	10.23.24.150	10.40.0.103	FTP	68	Request: OPTS UTF8 ON
879	16.781608	10.40.0.103	10.23.24.150	FTP	80	Response: 200 Always in UTF8 mode.
967	24.178069	10.23.24.150	10.40.0.103	FTP	66	Request: USER test1
969	24.179105	10.40.0.103	10.23.24.150	FTP	88	Response: 331 Please specify the password.
1202	30.129403	10.23.24.150	10.40.0.103	FTP	66	Request: PASS test1
1209	30.167296	10.40.0.103	10.23.24.150	FTP	77	Response: 230 Login successful.
1454	45.907976	10.23.24.150	10.40.0.103	FTP	60	Request: QUIT
1455	45.908863	10.40.0.103	10.23.24.150	FTP	68	Response: 221 Goodbye.

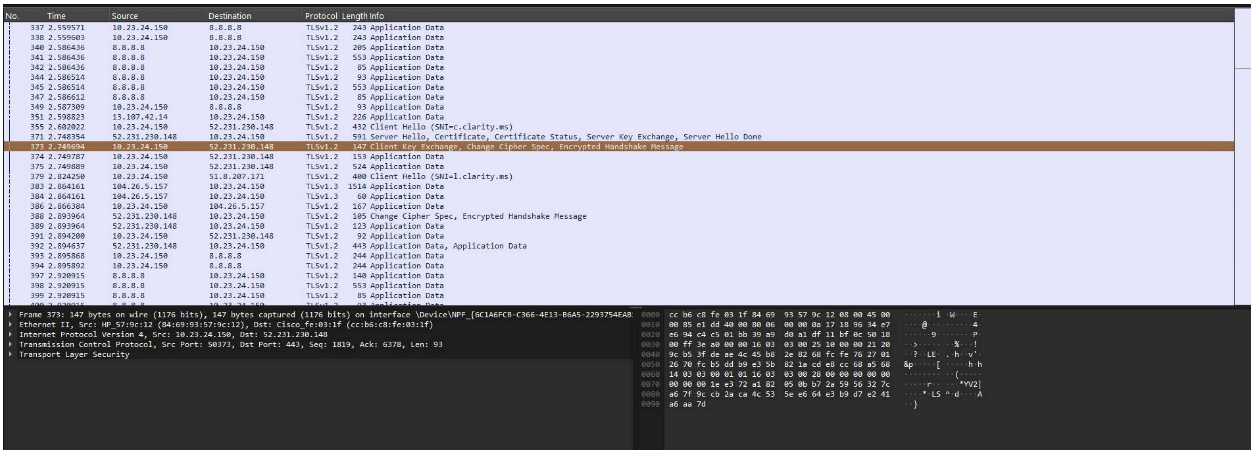
In this,if we look at **USER** and **PASS** commands in the captured packets. We can see the username and password transmitted in **plaintext**, making them easily readable

Using HTTP Protocol

No.	Time	Source	Destination	Protocol	Length	Info
277	5.756119	10.23.24.150	23.223.243.40	HTTP	165	GET /connecttest.txt HTTP/1.1
281	5.816804	23.223.243.40	10.23.24.150	HTTP	241	HTTP/1.1 200 OK (text/plain)
403	10.455242	10.23.24.150	10.40.0.103	HTTP	577	GET /login_process.html?username=test1&password=test1 HTTP/1.1
408	10.457248	10.40.0.103	10.23.24.150	HTTP	766	HTTP/1.1 200 OK (text/html)
410	10.470710	10.23.24.150	10.40.0.103	HTTP	579	GET /success.html HTTP/1.1
418	10.472446	10.40.0.103	10.23.24.150	HTTP	836	HTTP/1.1 200 OK (text/html)

In this,under the **HTTP request body**, we can see the username and password in **plaintext**, making them vulnerable to interception

Using HTTPS Protocol



Unlike HTTP and FTP, the login credentials here are **encrypted**,we can see encrypted TLS packets

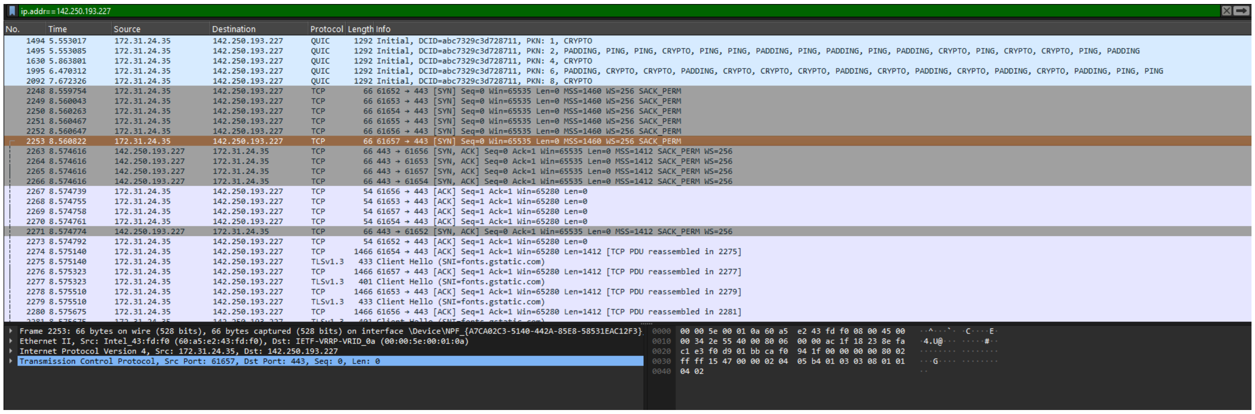
Task 2:
What is TCP handshake delay in Wireshark, and how does it impact network performance by increasing latency and slowing down communication? How can TCP handshake delay be measured using SYN, SYN-ACK, and ACK packets to identify network congestion or server-side issues? What methods can be used to analyze application-level delays, such as HTTP request-response time or DNS query-response time, to diagnose performance bottlenecks and optimize data transmission?

Solution:

The **TCP handshake delay** refers to the time it takes for a client and server to establish a connection. This process follows three steps:

1. **SYN (Client → Server):** The client sends a request to start a connection.
2. **SYN-ACK (Server → Client):** The server acknowledges the request.
3. **ACK (Client → Server):** The client confirms, and the connection is established.

If this handshake takes too long, it increases **network latency**, slowing down communication. High delays can be caused by **network congestion, slow server response, or inefficient routing**.

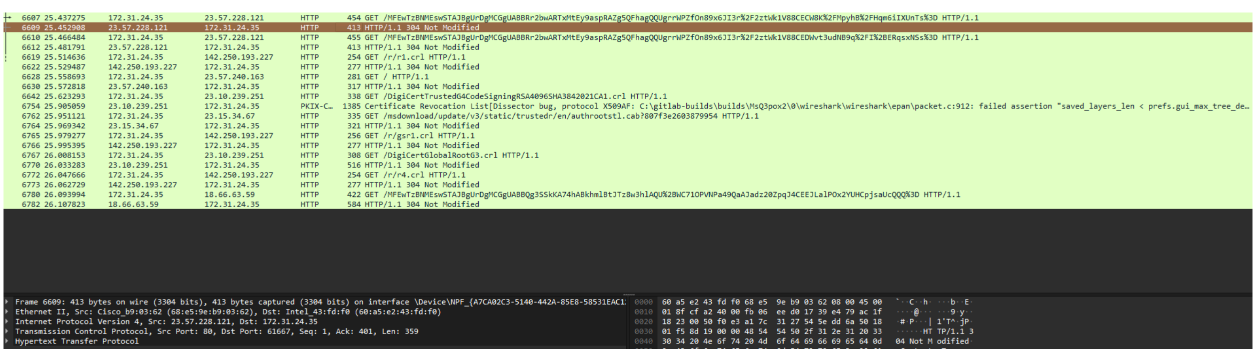


From above :Handshake Delay can be calculated by subtracting the timestamp of ACK request and SYN request:

Delay=TimeStamp(ACK)-TimeStamp(SYN)
In my case of a google search,this delay was around 13 ms

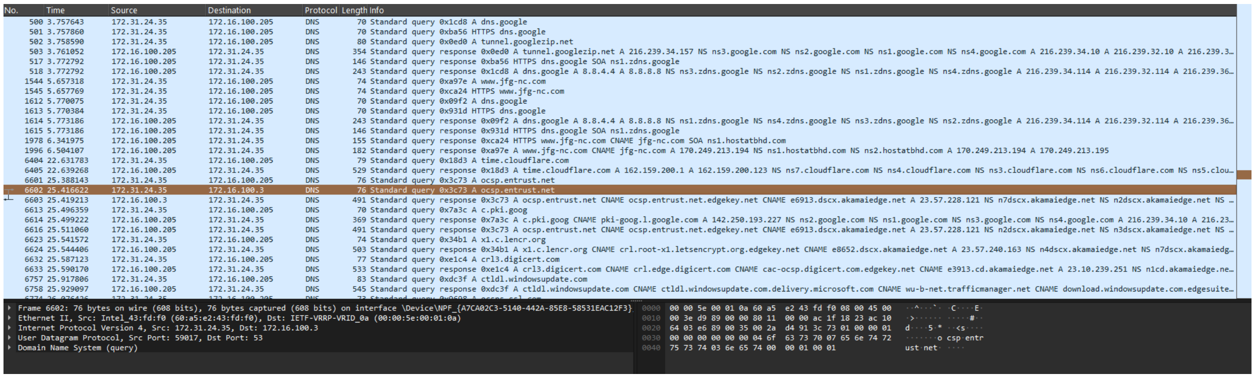
Ways to Analyze Application-Level Delays

Analyzing application-level delays involves measuring the time taken for various network interactions, such as HTTP requests, DNS queries, and database responses. For web applications, HTTP request-response time is a key metric; this can be analyzed by filtering for **HTTP traffic**, identifying **GET or POST requests**, and measuring the delay until the **HTTP 200 OK response**. Similarly, **DNS query-response time** can be evaluated by tracking the time difference between a **DNS request and its corresponding response**, which helps identify slow DNS resolution issues. Other methods include using **network performance monitoring tools** like Ping, Traceroute, or browser developer tools to inspect **resource loading times** and detect bottlenecks. By identifying these delays, developers can optimize performance through **caching, compression, load balancing, or server-side optimizations**, ensuring faster and more efficient data transmission



HTTP Delay= TS(HTTP OK)-TS(GET/POST Req)

For above it comes around 15.6ms



DNS Delay=TS(response)-TS(query)

For above it comes around 2ms

Diagnosing and Optimizing Performance

If network performance is slow, the following improvements can help:

1. Reducing TCP Handshake Delay

- Use **TCP Fast Open** to speed up handshakes.
- Reduce **packet loss** by ensuring a stable network.

2. Improving HTTP Performance

- Use a **Content Delivery Network (CDN)** to cache content closer to users.
- Enable **server-side caching and compression** to reduce load times.

3. Speeding Up DNS Resolution

- Switch to **faster DNS servers** like **Google DNS (8.8.8.8)**.
- Enable **DNS caching** to store previously resolved addresses.

Task3:

What filters can you apply in Wireshark to separate attack traffic from legitimate traffic, focusing on patterns like high SYN packets? How can you determine the type of DDoS attack (e.g., SYN flood, UDP flood, ICMP flood)? Lastly, how can you identify the attack source and analyze its impact on the network, looking for service degradation?

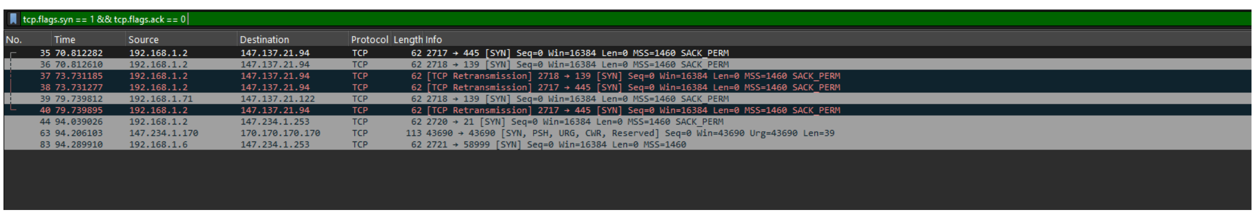
Solution:

In a DDoS attack, attackers overwhelm a network with excessive traffic, making services slow or unavailable for legitimate users. Using Wireshark, we can separate attack traffic from normal traffic by looking for unusual patterns, such as a high number of SYN packets (indicating a SYN flood) or excessive UDP and ICMP requests (suggesting a UDP or ICMP flood).

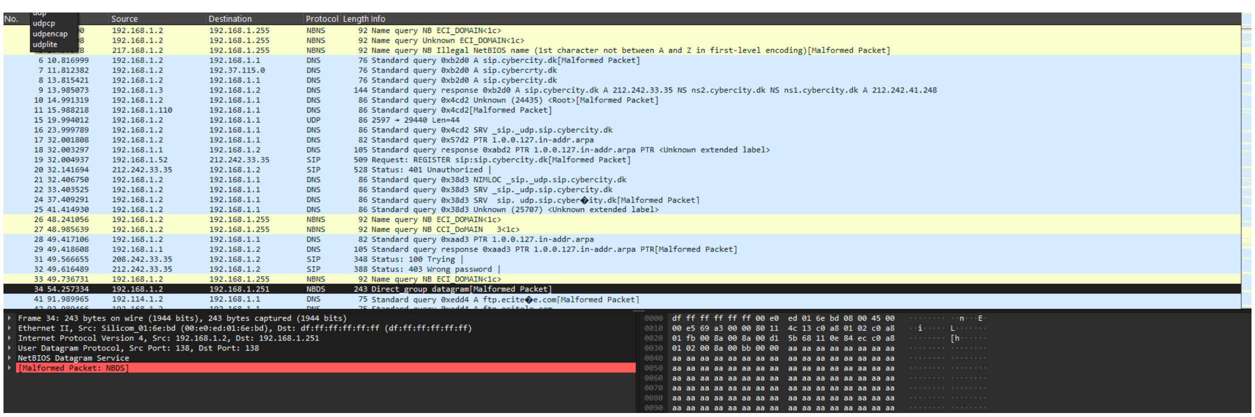
To detect a SYN flood, we filter packets where only the SYN flag is set, meaning the attacker is sending connection requests without completing the handshake. A UDP flood can be identified by tracking large volumes of UDP traffic sent to random ports, while an ICMP flood is evident when there are too many ping requests with very few responses.

Once suspicious traffic is found, we can analyze the source IP addresses to check if the attack is coming from a single system or a botnet. Sorting packets by source IP helps identify which addresses are sending excessive traffic. Finally, we assess the impact on the network by looking at packet loss, delayed responses, and failed connections, which indicate that the attack is degrading service quality.

By carefully filtering and analyzing captured traffic, we can not only detect a DDoS attack but also determine its type, source, and impact, helping to mitigate future threat



This shows all **SYN packets** without an ACK response, which could indicate an attack.



This shows UDP Packets being sent from few IP to random ports,indicating UDP Flood Attack

A **ping flood** sends a massive number of ICMP Echo Requests (ping requests) to overload the network.

To check for this,we can use this command:`icmp`

Things to keep in mind

- Look for **a high number of ICMP Echo Requests** (Type 8) without corresponding Echo Replies (Type 0).
- If most traffic comes from a single source, it may be a **direct attack**; if from multiple sources, it suggests a **botnet-based DDoS**.

To find the **IP addresses involved in the attack**:

- We can sort packets by **Source IP** and check which IPs are sending an excessive number of requests.

Using the filter:
`ip.src == x.x.x.x`

- (Replace `x.x.x.x` with a suspicious IP) to isolate and analyze its behavior.
- Cross-check against known **DDoS IP databases** to confirm if the IP is part of a botnet.

To determine how the attack affects network performance:

We can check **packet loss** using the filter:
`Tcp.analysis.lost_segment`

We can look for **high latency** in TCP handshakes with:
`tcp.analysis.initial_rtt`

- If **legitimate user connections** (e.g., HTTP requests) are getting delayed or dropped, this indicates **service degradation** due to the attack.