Network Traffic Analysis with Wireshark: A Hands-On Exploration

November 30, 2025

Overview

This project demonstrates hands-on network traffic analysis using Wireshark. By generating traffic with commands like `ping`, `nslookup`, `dig`, `curl`, `openssl`, and `traceroute`, I captured and filtered DNS, HTTP, HTTPS/TLS, ICMP, TCP, and QUIC packets. The screenshots highlight how different protocols behave, how insecure traffic can be identified, and how secure communication is verified

Goals

1. **Hands-on practice with Wireshark**: Capture and analyze real network traffic generated by common Linux commands.
2. **Protocol exploration**: Investigate DNS, HTTP, HTTPS/TLS, ICMP, TCP, NTP, and QUIC to understand how different layers of communication work.
3. **Filter mastery**: Apply targeted Wireshark filters (IP, port, flags, frame size, string search) to isolate meaningful traffic.
4. **Security awareness**: Highlight insecure traffic (HTTP, plaintext POST requests) versus secure traffic (TLS, QUIC) and explain the risks.

5. **Forensic correlation**: Connect terminal actions (ping, nslookup, dig, curl, traceroute, openssl) with the exact packets they generate.

Set up and Methodology

Environment: Ubuntu virtual machine running in VirtualBox (CyberLab).

Tools Used: Wireshark for packet capture and analysis, Linux terminal for traffic generation.

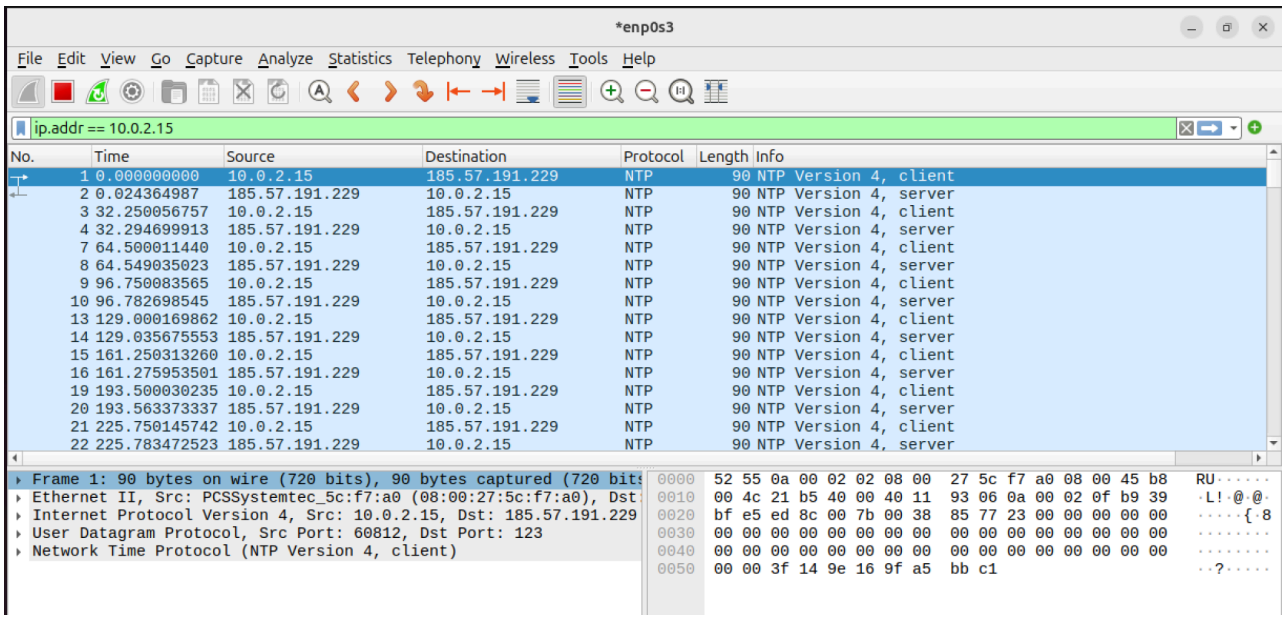
Traffic Generation: Commands like `ping`, `nslookup`, `dig`, `curl`, `openssl`, and `traceroute` were used to create real network traffic.

Filtering Approach: Applied targeted Wireshark filters (IP, port, flags, frame length, string search, application-layer methods) to isolate specific protocols and behaviors.

Documentation: Each screenshot was paired with a caption explaining the command, filter, key findings, and security value.

Captured Traffic & Protocol Analysis

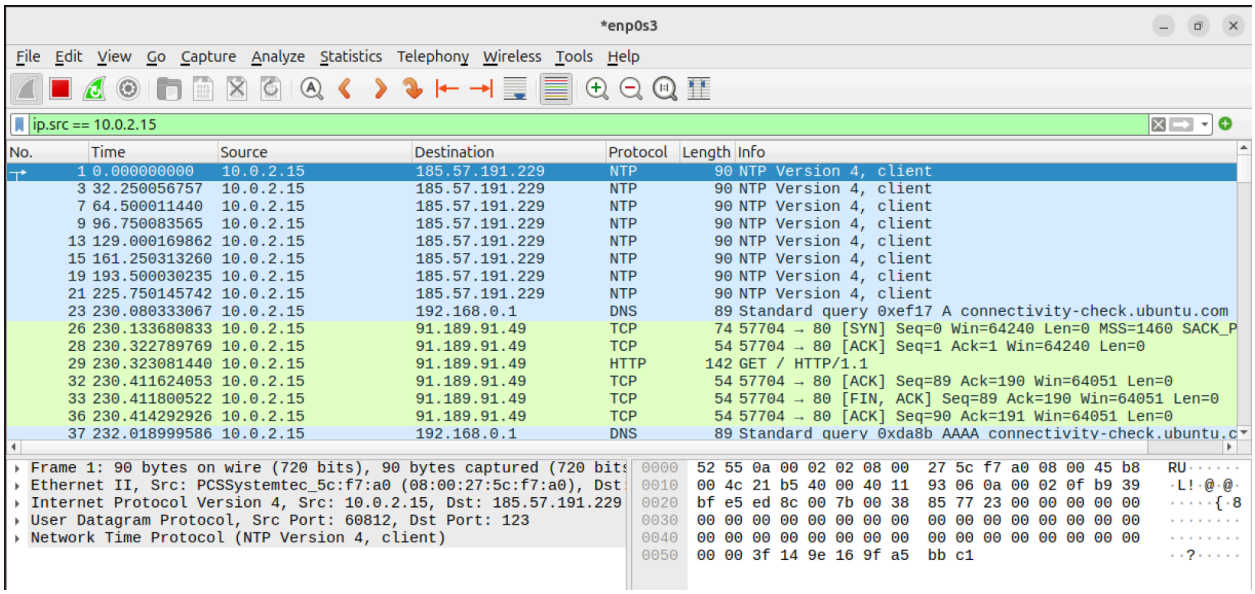
1. Isolation and Identification of Host Traffic (NTP)



Filter Used: ip.addr == 10.0.2.15

- **What it shows:** All network communication for the virtual machine (10.0.2.15).
- **Key Finding:** The machine is primarily using **Network Time Protocol (NTP)** to constantly update its system time with an external server.
- **Security Value:** The presence of NTP traffic **confirms time synchronization is functional**, which is **critical for the forensic integrity** of security logs and the accurate validation of digital certificates used in TLS traffic.

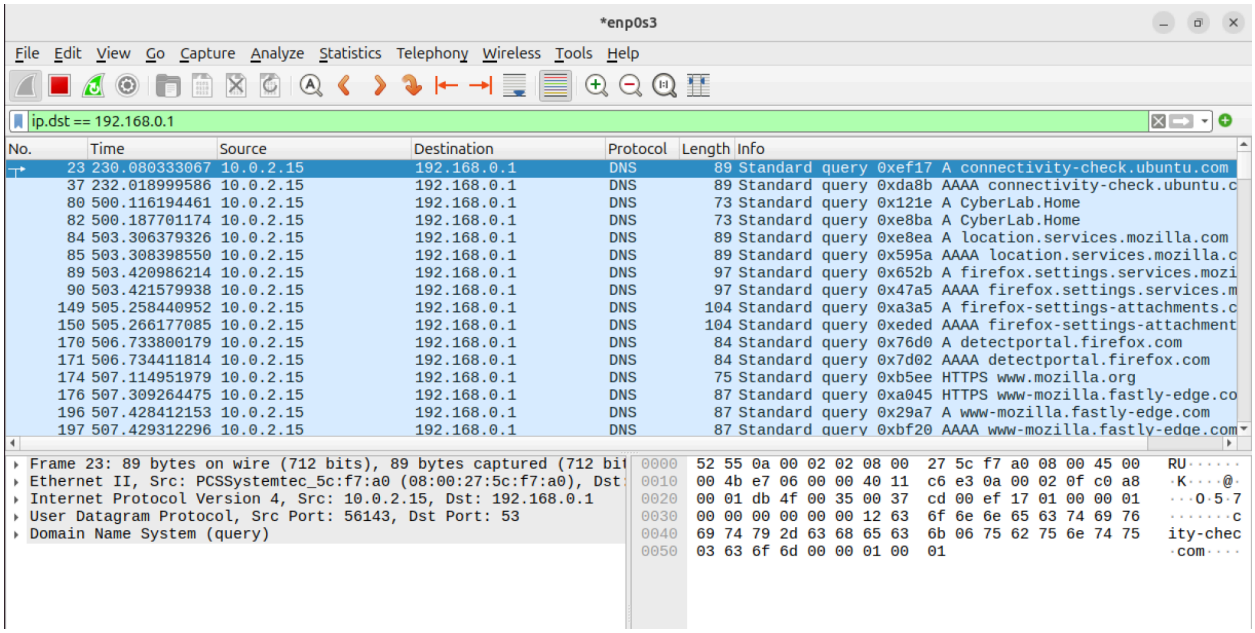
2. Multi-Protocol Communication and Connectivity Check



Filter Used: ip.src == 10.0.2.15 (Outgoing traffic only)

- **What it shows:** This capture creates a chronological **timeline of every network action** initiated by the host machine (10.0.2.15). This includes background tasks like NTP synchronization, DNS queries, and spontaneous TCP attempts.
- **Key Findings:** The capture contains three essential protocols:
    1. **NTP:** Continued time synchronisation (as seen previously).
    2. **DNS (Port 53):** A request for connectivity-check.ubuntu.com to resolve an IP address.
    3. **HTTP (Port 80):** A plain-text **GET request** confirming internet access.
    4. In simple terms I now have the complete list of **every website my computer attempted to contact**, which helps me establish a baseline of normal browsing and background checks.
- **Security Value:** Identifying DNS and HTTP traffic flows allows a security analyst to **map application behaviour** and identify potential unencrypted (plaintext) data transfers. The use of the specific source filter (ip.src) demonstrates proficiency in **tracking outbound attack traffic or data exfiltration.**
- This process allows **me** to conduct **threat tracking**. If I see my machine asking for the address of a **hacker's command-and-control server** here, it gives me an immediate, confirmed **red flag** that my machine is infected.

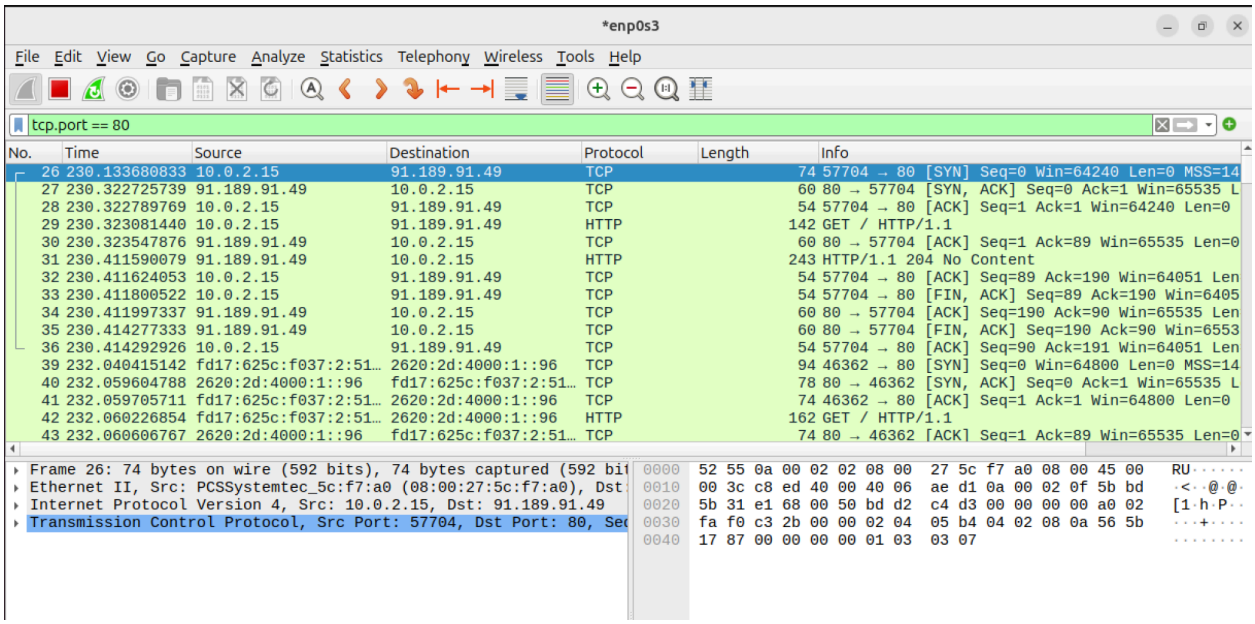
3. Domain Name System (DNS) Query Analysis



Filter Used: ip.dst == 192.168.0.1 (Traffic destined for the local router/DNS server)

- **What it shows:** A detailed sequence of **Domain Name System (DNS)** queries being sent from the VM to the local DNS resolver (192.168.0.1).
- **Key Findings:** The capture reveals the exact hostnames the machine is attempting to look up (e.g., A CyberLab.Home, mozilla.com, fastly-edge.com). This traffic is essentially a "**web history**" log of the host's activity.
- **Security Value:** Monitoring DNS traffic is fundamental to **threat hunting**. Analysts can use this information to detect requests for **Command and Control (C2) servers** or known malicious domains, even if the final connection is encrypted.

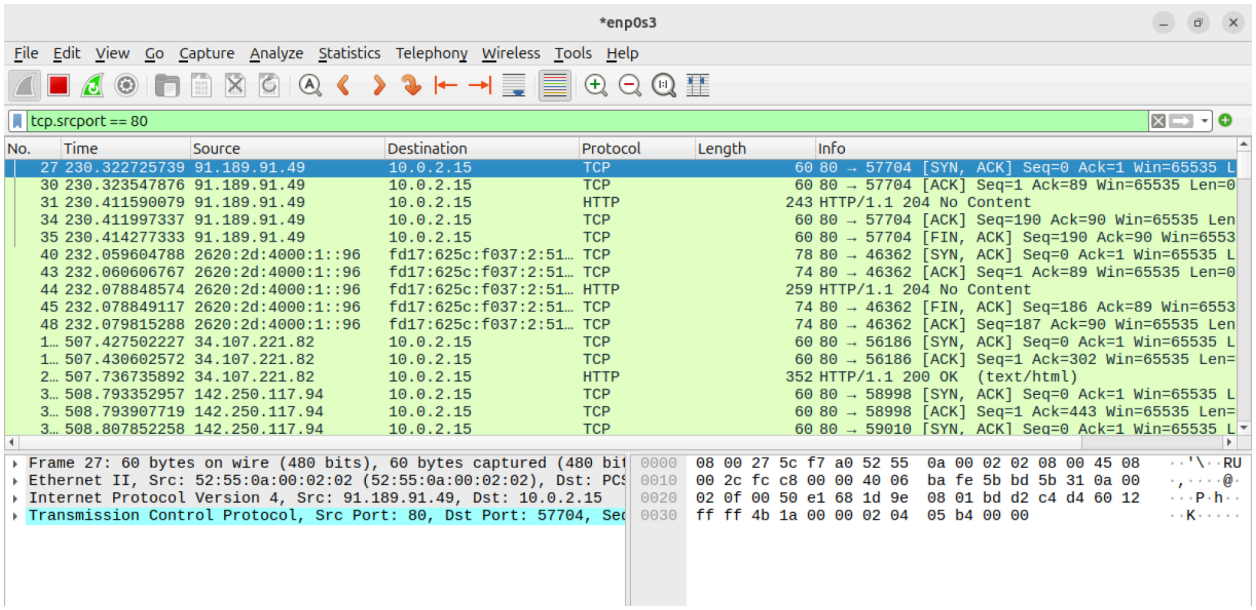
4. Analysis of Insecure HTTP Communication



Filter Used: tcp.port == 80 (Standard port for unencrypted web traffic)

- **What it shows:** This trace isolates all unencrypted web communication (HTTP). It clearly shows the entire process of connecting:
    1. **Agreement:** The three-step handshake needed to start the conversation.
    2. **The Message:** The actual request (GET) my computer sent to the server.
- **Key Findings:** By using TCP Port 80, the traffic is confirmed to be **unencrypted (plaintext)**. This means all transmitted data, including URLs, headers, and any submitted forms, are visible to an intercepting attacker.
- **Security Value:** This clearly demonstrates the **risk of HTTP** and the necessity of enforcing **HTTPS (TLS/SSL)**for confidentiality. The packet details pane confirms the use of the reliable, connection-oriented **Transmission Control Protocol (TCP)**.
- **Forensics Insight:** The filter successfully isolates all traffic related to the HTTP connection, showing the complete session flow from establishment to data transfer.

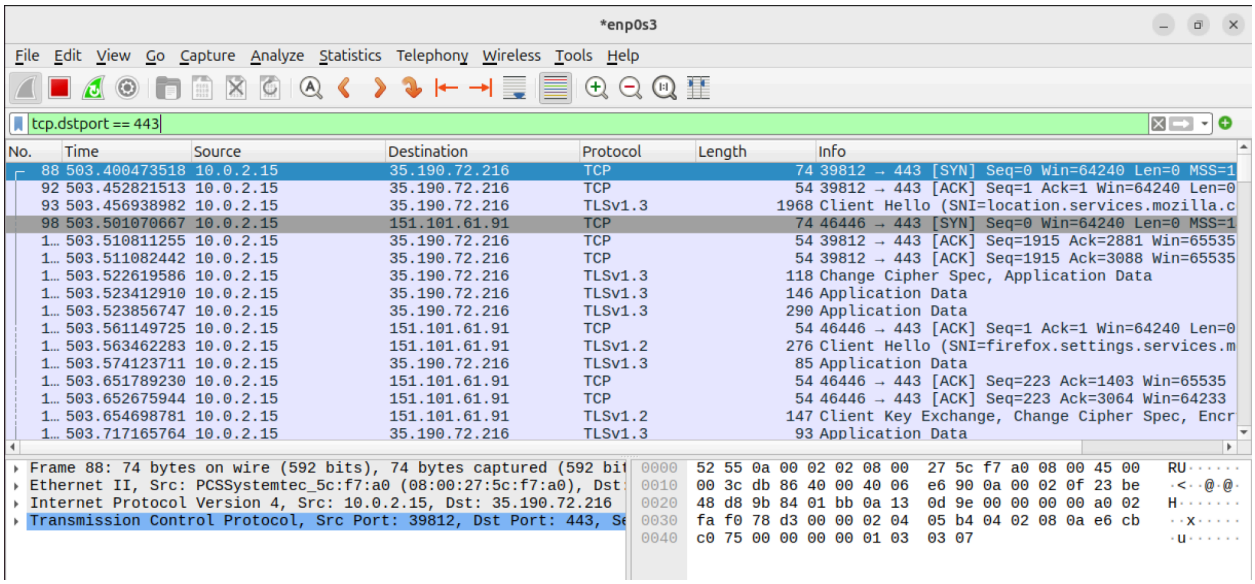
5. HTTP Server Response and IPv6 Observation



Filter Used: tcp.srcport == 80 (Focusing on traffic coming *from* the HTTP server)

- **What it shows:** This trace isolates the **entire communication sequence** of a basic, unencrypted web transaction (HTTP). It reveals the two essential parts:
    - **Agreement (TCP Handshake):** The three-step greeting (SYN, SYN/ACK, ACK) that must occur to establish the connection before any data is exchanged.
    - **The Message (HTTP GET):** The actual request sent by the machine, asking the server for a webpage.
- **Key Findings:** The filter demonstrates the ability to analyze the **full lifecycle of a web connection**, from the initial agreement to the final message (GET) and the graceful closure (FIN/ACK). This confirms the connection was successful and ended cleanly.
- **Security Value:** This highlights a common vulnerability: since the communication is **unencrypted (HTTP) over Port 80**, the **entire conversation—from the connection setup to the final request—is** exposed in plaintext. This proves the necessity of enforcing HTTPS to prevent data exposure.
- **Forensics Insight:** The filter successfully isolates the server's response stream, confirming the data delivery and proper session teardown.

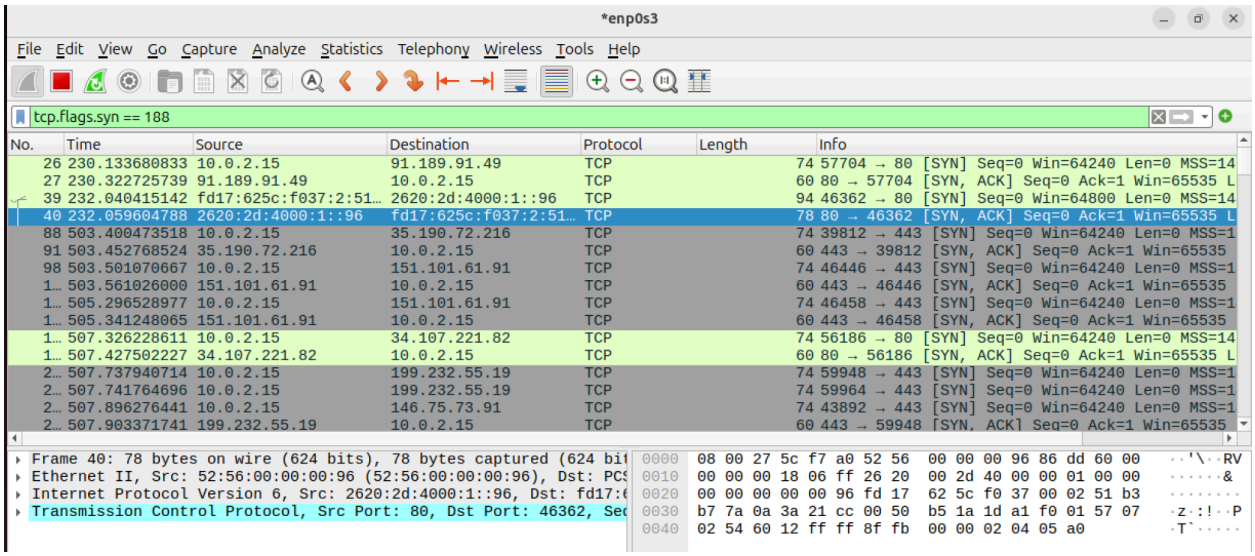
6. Encrypted Communication (TLS/HTTPS) Handshake



Filter Used: tcp.dstport == 443 (Standard port for secure web traffic)

- **What it shows:** The filter isolates the sequence of security messages required to establish a secure tunnel over HTTPS. This trace demonstrates the full negotiation process:
    - **Connection Agreement:** The initial TCP handshake (SYN, SYN/ACK, ACK) to start the connection.
    - **Encryption Negotiation:** The TLS messages (Client Hello, Server Hello) where the two parties agree on the specific encryption standard (TLSv1.3) and shared keys.
    - **Secure Tunnel:** The final packets are labeled **Application Data**, confirming the session is now successfully encrypted and protected.
- **Key Findings:** The TLS protocol script is successfully read and dissected, demonstrating the capability to confirm the initiation of an encrypted connection. This expertise is vital for diagnosing issues such as certificate errors or weak cipher configurations.
- **Security Value:** Unlike the plaintext HTTP shown earlier, the data in the subsequent **Application Data** packets (not shown here, but implied) is **scrambled and unreadable**. This demonstrates the successful implementation of the **Confidentiality** principle, preventing external attackers from reading the content.

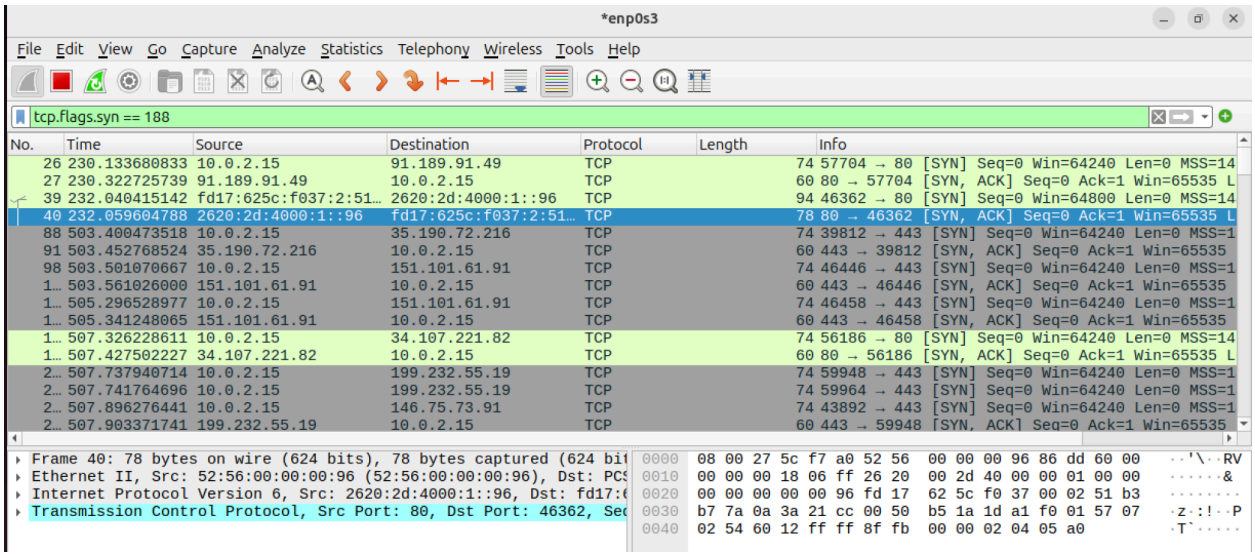
7. HTTP Content Filtering (GET Requests)



Filter Used: http.request.method == "GET" (Filtering based on the application layer content)

- **What it shows:** Advanced filtering used to isolate only the packets where the client is explicitly **requesting data** (HTTP GET method). This successfully strips away all other noise (e.g., TCP handshakes, server responses).
- **Key Findings:** The system is running several connectivity checks, asking for files such as /canonical.html and /success.txt. These requests show that the host is still using insecure HTTP connections on both IPv4 and IPv6, meaning the traffic is visible in plain text instead of being protected by encryption.
- **Security Value:** Filtering by the GET method is crucial in **incident response**. It immediately highlights what data or resources a compromised machine is attempting to retrieve, helping an analyst quickly understand the scope of the attack or system activity.

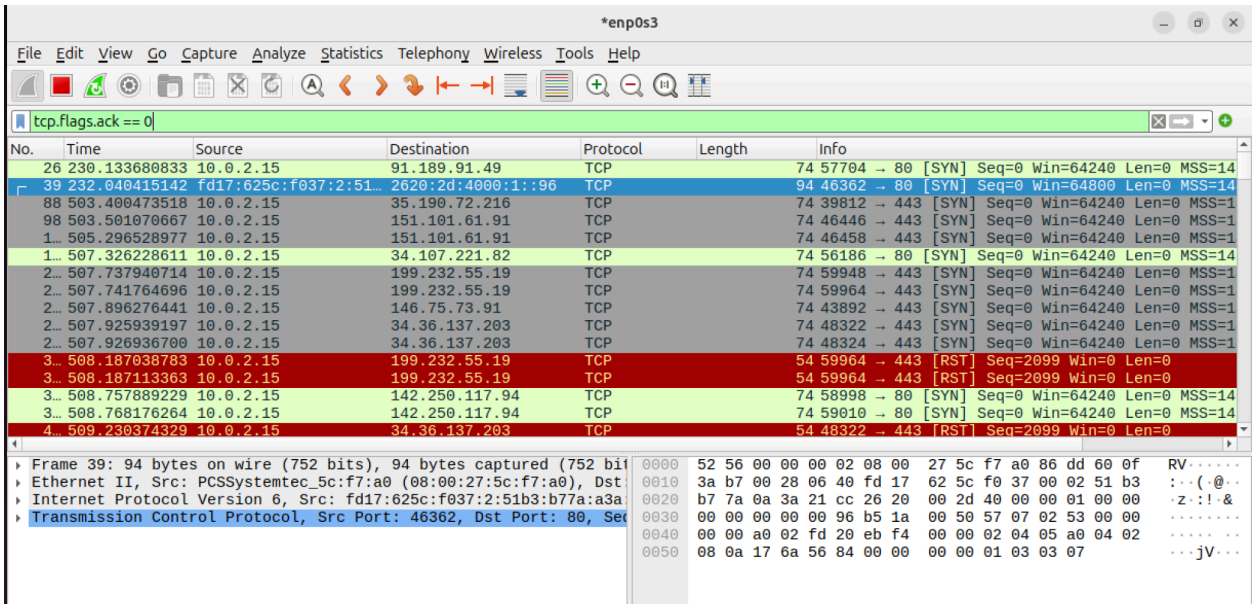
8. Advanced TCP Flag Analysis (Complex Value 188)



Filter Used: tcp.flags.syn==188 (Filtering based on complex TCP flag combinations)

- **What it shows:** This filter isolates packets where the SYN flag is set alongside other control flags (e.g., ACK, PSH, URG). It's a more advanced view of connection behavior, beyond the standard handshake.
- **Key Findings:** The system is starting connections with several control flags active at the same time, such as SYN and ACK. This pattern can point to things like repeated attempts to reconnect mid-session, unusually forceful connection setups, or traffic that doesn't follow the standard TCP rules.
- **Security Value:** Analyzing unusual flag combinations helps detect anomalies, stealth scans, or misconfigured applications. It's especially useful in threat hunting or forensic analysis where attackers may manipulate TCP flags to evade detection.

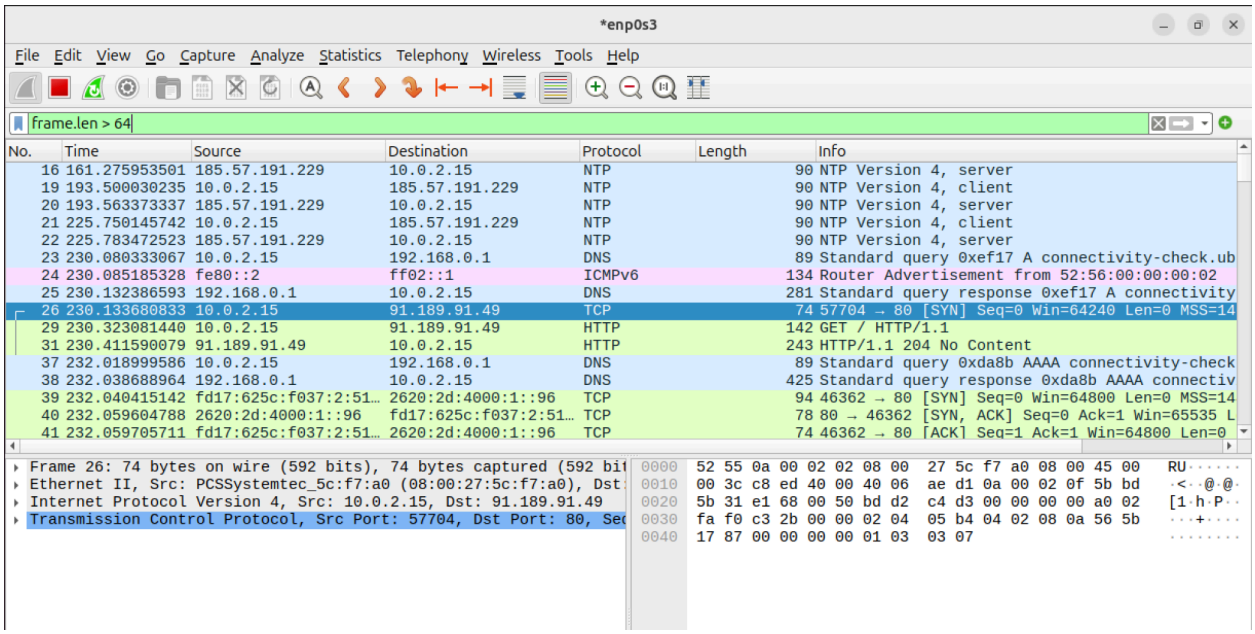
9. Detection of TCP Connection Resets (RST)



Filter Used: tcp.flags.ack == 0 (Used to find initial packets like SYN, or sometimes, RSTs sent without prior ACKs)

- **What it shows:** The packets highlighted in **red** are **TCP Reset (RST) packets** sent from the destination server (e.g., 199.232.55.19).
- **Key Findings:** The RST packets indicate an **immediate, non-graceful termination** of a connection. In this context, they are likely being sent in response to the initial SYN packets (seen in the grey lines), confirming the targeted ports are **closed** or the connections are being **rejected**.
- **Security Value:** Identifying a high volume of RST packets is a critical indicator in **security analysis**, often confirming that a **port scan** is underway or that a system's **firewall** is explicitly dropping the traffic.

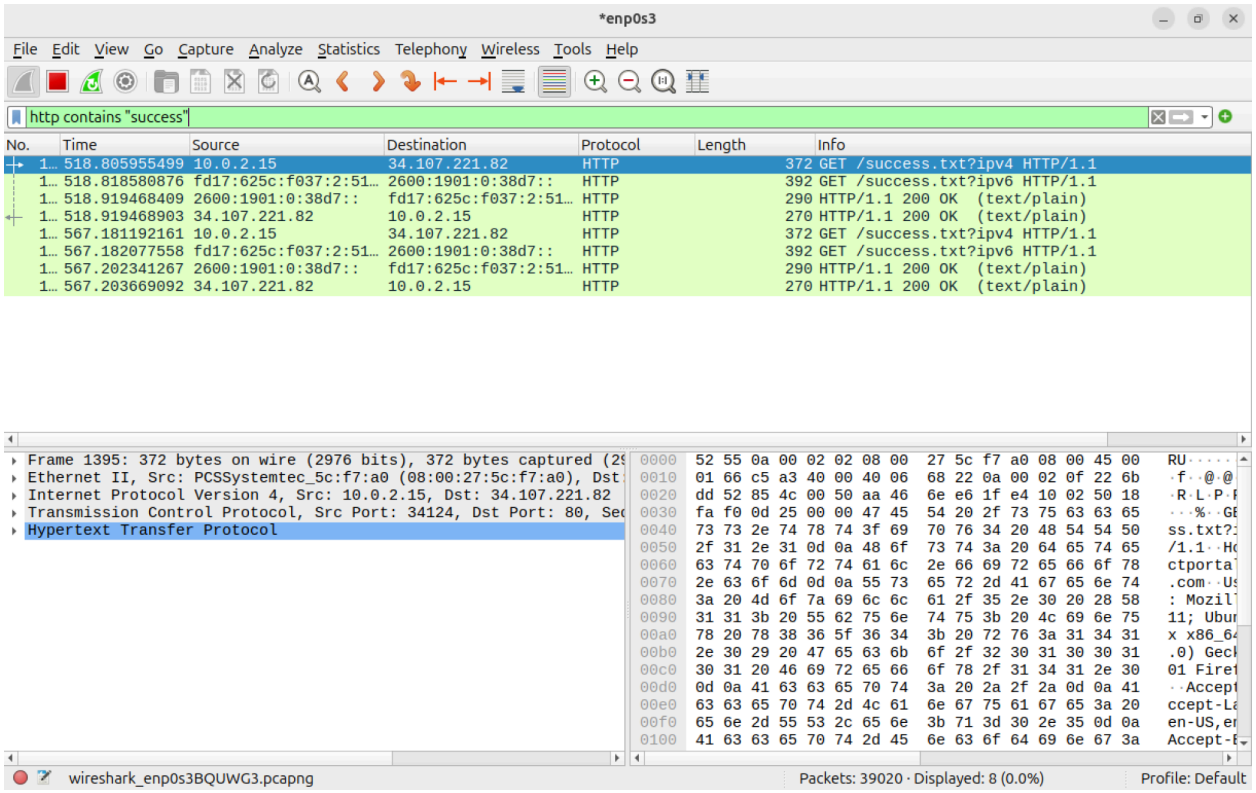
10. Advanced Filtering by Packet Size (Frame.len≤64)



Filter Used: frame.len <= 64 (Filtering for all packets smaller than or equal to 64 bytes)

- **What it shows:** This advanced filter isolates **small network packets**, demonstrating the ability to filter traffic based on low-level characteristics, a crucial skill in finding network anomalies.
- **Key Findings:** The filtered results show small bits of network activity, like time updates (NTP), website lookups (DNS), and router announcements (ICMPv6). By removing the larger data transfers, the filter makes it easier to focus on these quick signaling exchanges that keep the network running in the background.
- **Security Value:** Filtering by frame size is a technique used in **malware analysis** and **threat hunting**. Specific types of command-and-control (C2) communication or **steganography** traffic often rely on abnormally small or specific packet sizes to evade detection.

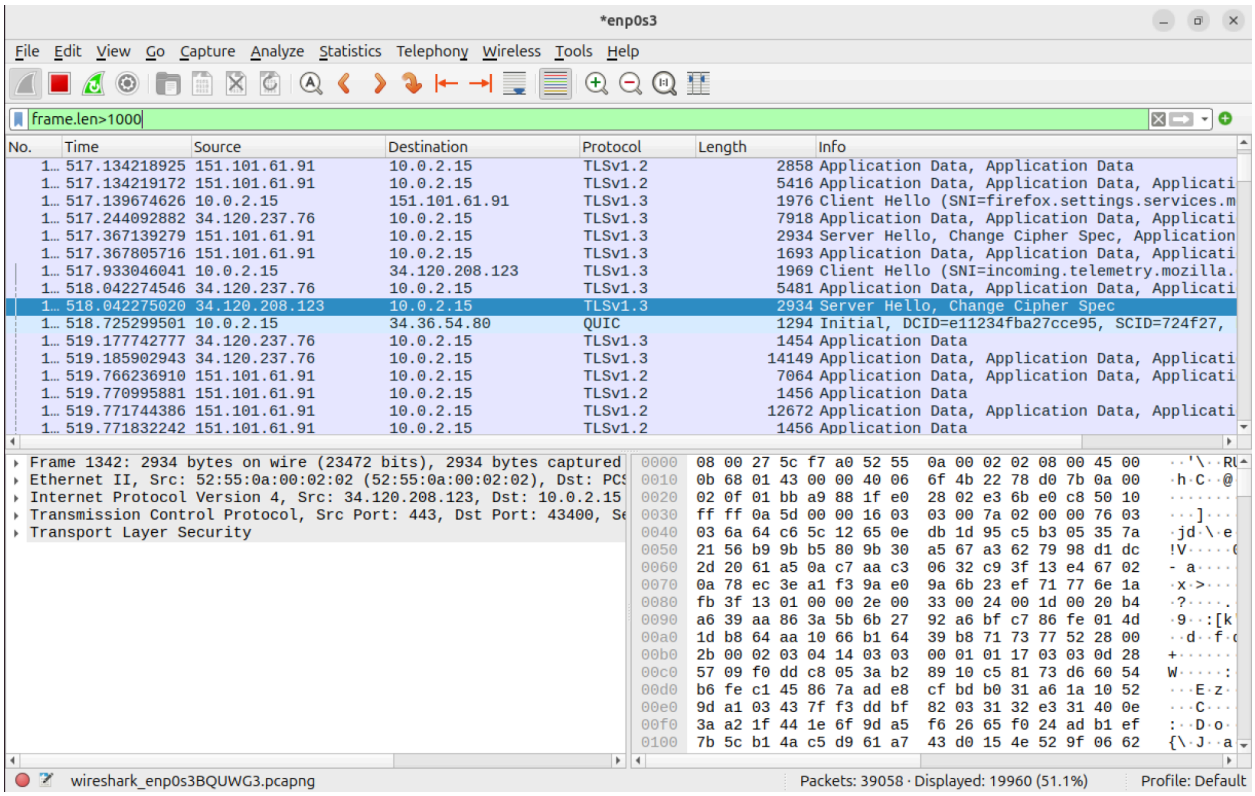
11. Application Layer String Filtering (Forensic Search)



Filter Used: http.contains "success" (Searching for a specific text string inside the packet data)

- **What it shows:** This filter isolates all HTTP traffic where the packet payload contains the string "**success**". This is a powerful demonstration of looking *inside* the data being transmitted.
- **Key Findings:** The capture shows multiple HTTP GET requests for /success.txt over both **IPv4 and IPv6**. The filter successfully highlighted the application's attempt to confirm connectivity or report status.
- **Security Value:** Filtering by string content is crucial for **data leakage prevention** and **incident response**. It allows an analyst to search for keywords like usernames, passwords, credit card numbers, or proprietary terms, which is the most direct way to identify **data exfiltration** over unencrypted channels.

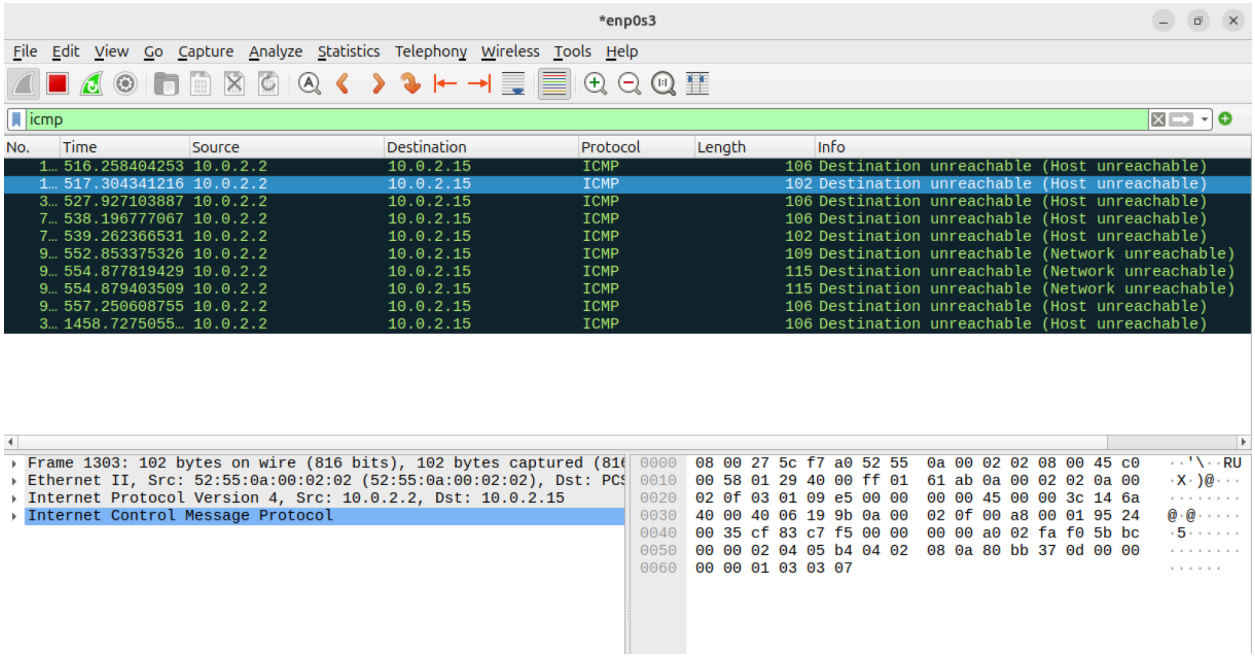
12. Advanced Filtering by Packet Size (Frame.len>1000)



Filter Used: frame.len > 1000 (Filtering for all packets **larger** than 1000 bytes)

- **What it shows:** This advanced filter isolates **large data-carrying packets**, demonstrating the ability to focus on application layer traffic and data transfers rather than signaling.
- **Key Findings:** The filtered results highlight the main session data, made up of large encrypted packets from TLS versions 1.2 and 1.3. It also shows QUIC traffic — a modern, secure protocol used by services like Google and Chrome — demonstrating awareness of how newer technologies are shaping secure communication online.
- **Security Value:** Analyzing large packets helps identify potential **data exfiltration** of large files or bulk data. The simultaneous presence of encrypted TLS and QUIC confirms the machine is adhering to modern security standards for data confidentiality.

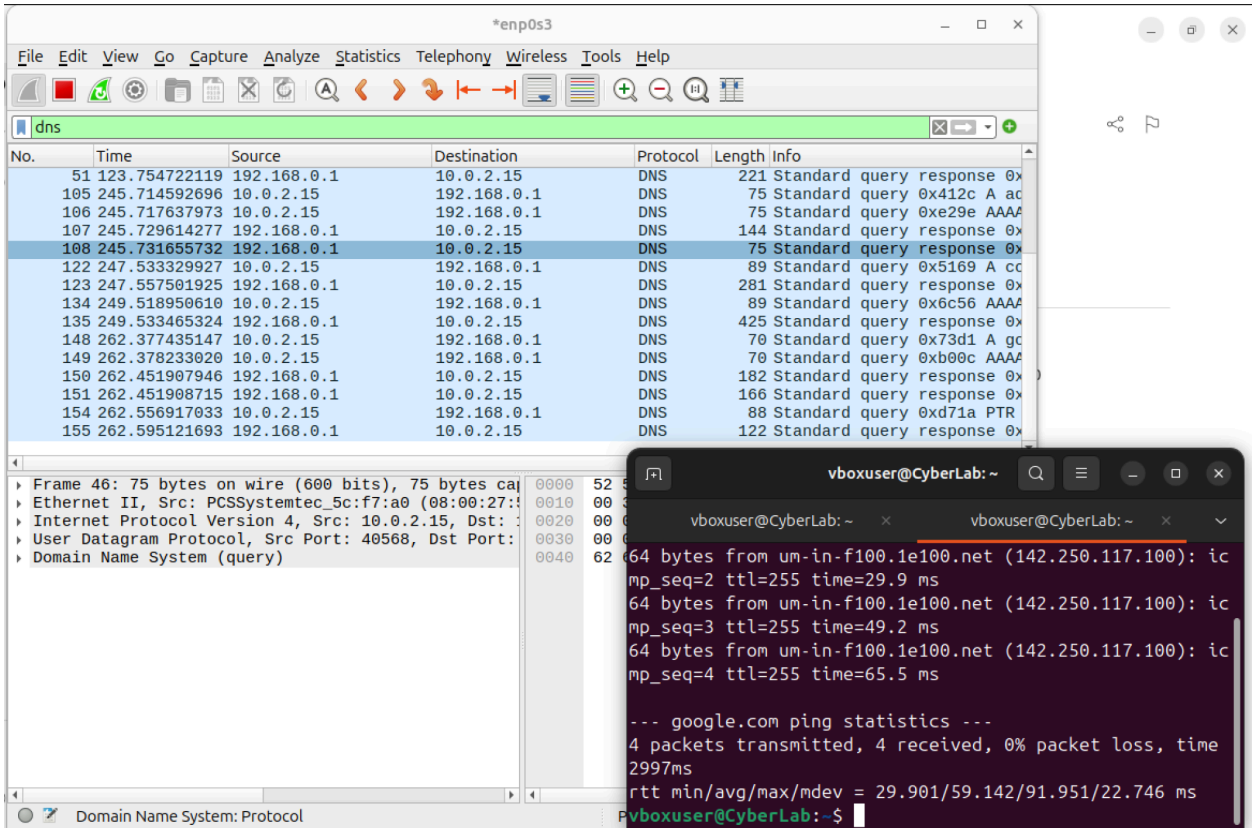
13. Network Diagnostic and Error Reporting (ICMP)



Filter Used: icmp (Filtering for all Internet Control Message Protocol packets)

- **What it shows:** The capture contains several **ICMP error messages** (Protocol 1), which are the standard mechanism used by network devices to send error reports and operational information.
- **Key Findings:** The messages are all "Destination unreachable" errors, specifically "Host unreachable" or "Network unreachable." This indicates that the source host (10.0.2.2) **could not establish a path** to the destination host (10.0.2.15).
- **Highlighting Significance:** The packets are highlighted in **dark blue/black**. This is Wireshark's default coloring for packets that indicate **network problems, errors, or exceptional conditions**, immediately alerting the analyst to an issue preventing normal data flow.
- **Security Value:** ICMP is essential for **troubleshooting** and is often used by attackers for **network discovery (pingscanning)** or creating cover channels (ICMP tunneling). Understanding these messages is critical for firewall configuration and defense.

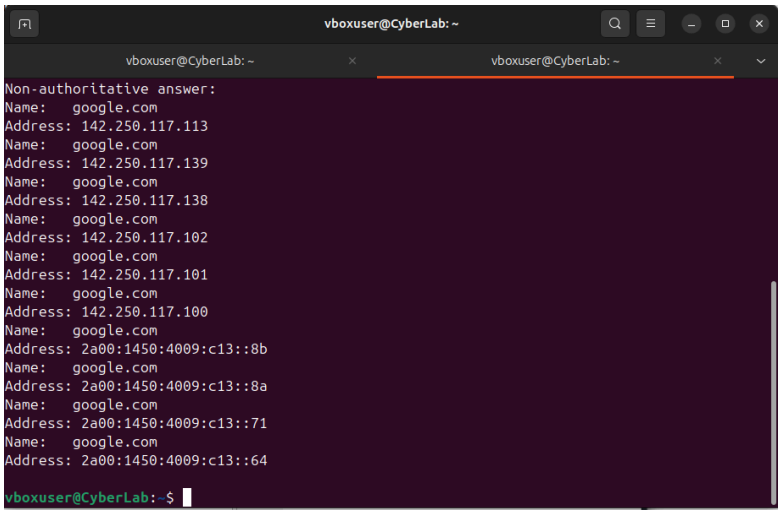
14. Linking Command Line Action to DNS Resolution (Ping Analysis)

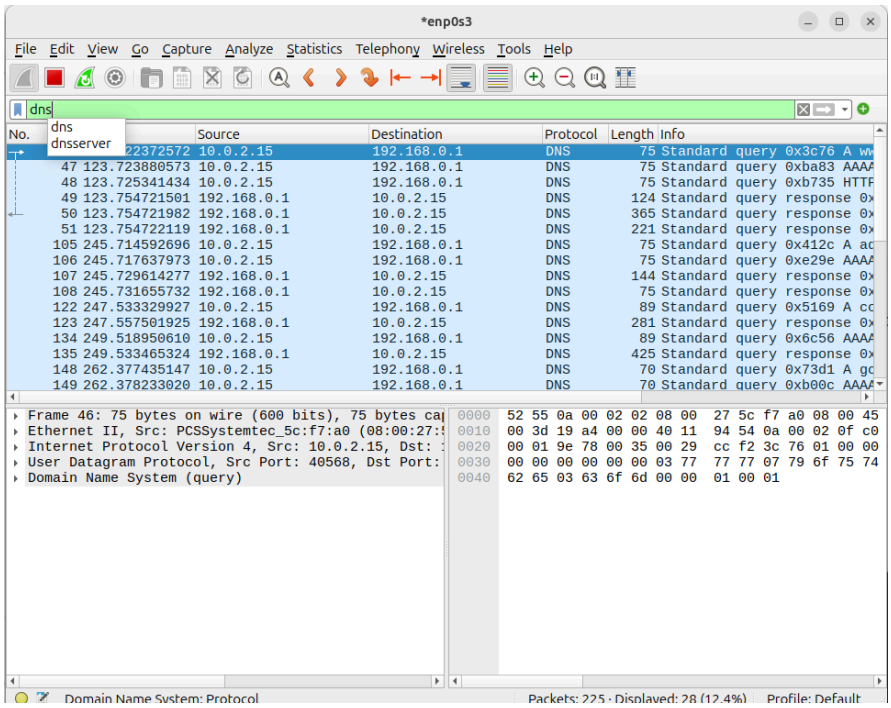


Filter Used: dns (Focusing on Domain Name System traffic)

- **Action Demonstrated:** The Ubuntu terminal (visible in the bottom right) shows the execution of ping -c 4 google.com. The subsequent Wireshark capture demonstrates the network traffic generated by this command.
- **Key Findings:** The capture successfully isolates the **DNS queries** and **responses** (lines 107 and 108) sent to the local DNS server (192.168.0.1) to **resolve the hostname google.com** into an IP address before the ping can start.
- **Forensic Value:** This perfectly illustrates the initial phase of any connection and proves the analyst can **correlate user or system actions** with the precise network packets they generate. The blue highlight confirms the selection of a DNS response packet.
- **Protocol Detail:** The DNS communication uses the **User Datagram Protocol (UDP)** on Port 53

15. Deep DNS Resolution Analysis (nslookup)





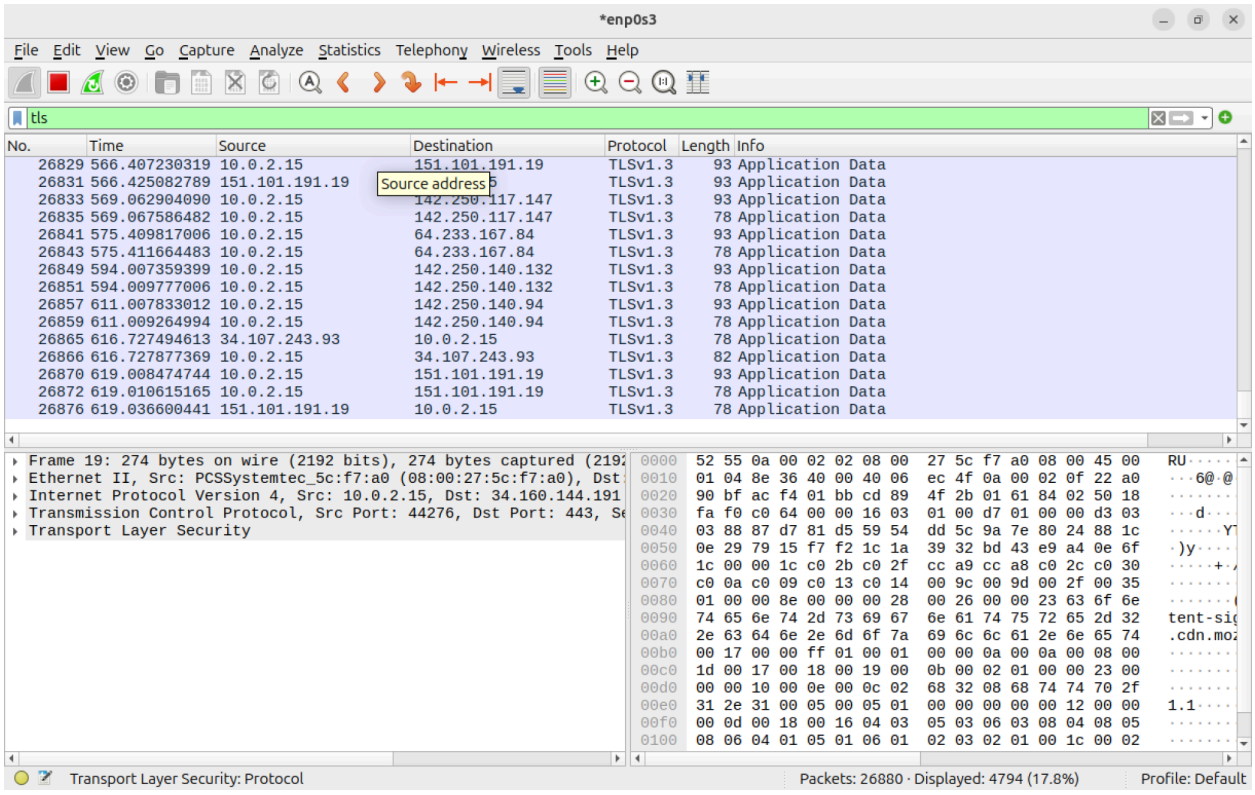
Filter Used: dns (Showing the Domain Name System packets)

- **Action Demonstrated:** The use of the nslookup google.com command in the Ubuntu terminal (output shown in the accompanying screenshot) forces a deep DNS lookup.
- **Key Findings:** Unlike a simple ping, the nslookup command reveals the server's **non-authoritative answer**, which includes a full set of IP addresses for **Google.com**, covering both multiple IPv4 addresses (142.250.x.x) and multiple **IPv6 addresses** (2a00:1450:x:x). The Wireshark view confirms the multiple DNS responses carrying this data.

- **Forensic Value:** This demonstrates the skill of using **advanced diagnostic tools** (like nslookup) and immediately translating the diagnostic traffic into network packets. It highlights the analyst's ability to map a target's full network footprint (both IPv4 and IPv6).

16. Advanced Certificate Verification and TLS Application Data

```
vboxuser@CyberLab:~$ openssl s_client -connect google.com:443
CONNECTED(00000003)
depth=2 C = US, O = Google Trust Services LLC, CN = GTS Root R1
verify return:1
depth=1 C = US, O = Google Trust Services, CN = WR2
verify return:1
depth=0 CN = *.google.com
verify return:1
---
Certificate chain
 0 s:CN = *.google.com
   i:C = US, O = Google Trust Services, CN = WR2
   a:PKEY: id-ecPublicKey, 256 (bit); sigalg: RSA-SHA256
   v:NotBefore: Oct 27 08:33:43 2025 GMT; NotAfter: Jan 19 08:33:42 2026 GMT
 1 s:C = US, O = Google Trust Services, CN = WR2
   i:C = US, O = Google Trust Services LLC, CN = GTS Root R1
   a:PKEY: rsaEncryption, 2048 (bit); sigalg: RSA-SHA256
   v:NotBefore: Dec 13 09:00:00 2023 GMT; NotAfter: Feb 20 14:00:00 2029 GMT
 2 s:C = US, O = Google Trust Services LLC, CN = GTS Root R1
   i:C = BE, O = GlobalSign nv-sa, OU = Root CA, CN = GlobalSign Root CA
   a:PKEY: rsaEncryption, 4096 (bit); sigalg: RSA-SHA256
   v:NotBefore: Jun 19 00:00:42 2020 GMT; NotAfter: Jan 28 00:00:42 2028 GMT
---
Server certificate
-----BEGIN CERTIFICATE-----
MIIOSDCCDTCgAwIBAgIRAJ2e0OHMDsyECgQuPC+e1XEwDQYJKoZIhvcNAQELBQAw
OzELMAkGA1UEBhMCVVMxHjAcBgNVBAoTFUdvb2dsZSBUcnVzdCBTZXJ2aWNlczEM
MAoGA1UEAxMDV1IyMB4XDTI1MTAyNzA4MzM0M1oXDTI2MDExOTA4MzM0MlowFzEV
MBMGA1UEAwwMKi5nb29nbGUuY29tMFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAE
8eDmUd1KMtjPw+FE4Zaje6Ck/RhUZ10Vh6lM06IP6t2ESNw7o4fyPdATskD20rhF
c3GNxyYaXqq6SFY48PPVKQCDDQwggwwMA4GA1UdDwEB/wQEAwIHgDATBgNVHSUE
```

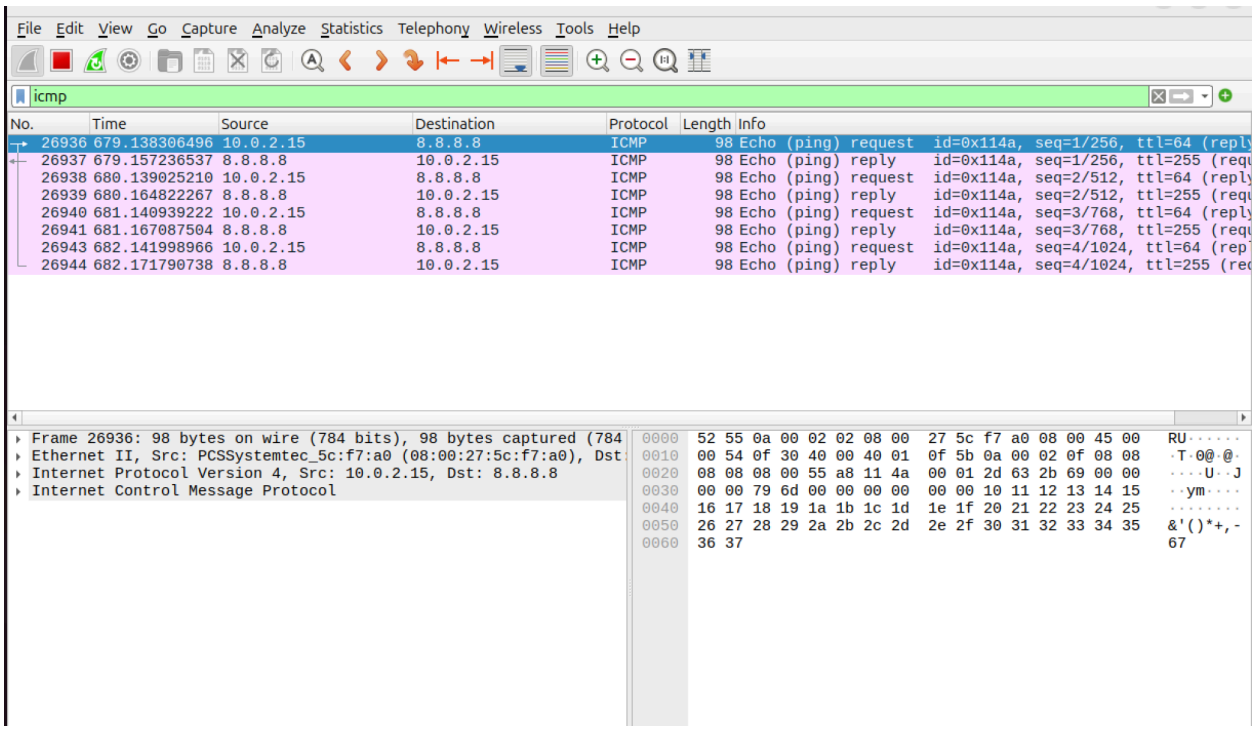


Terminal Command: openssl s_client -connect google.com:443 **Wireshark Filter Used:** tls (Focusing on Transport Layer Security)

- **Action Demonstrated:** The use of the advanced **openssl s_client** utility forces a secure connection to the target server on Port 443 and retrieves the full **TLS Certificate Chain** (shown in the terminal screenshot), verifying the server's identity.
- **Key Findings:** The Wireshark capture shows TLSv1.3 packets being exchanged, confirming that the client and server completed a secure handshake and are now communicating with encrypted data.
- **Certificate Analysis:** he terminal output verifies the certificate chain, confirms the server identity (*.google.com), and checks validity dates — all essential steps to prevent attacks like man-in-the-middle.
- **Security Value:** This demonstrates strong competence in ensuring both data confidentiality and server trust, which are core principles of modern web security.

17. Fundamental Connectivity Check (Ping/ICMP Success)

```
vboxuser@CyberLab:~$ ping -c 4 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=255 time=19.0 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=255 time=25.8 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=255 time=26.2 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=255 time=29.8 ms

--- 8.8.8.8 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3002ms
rtt min/avg/max/mdev = 19.006/25.205/29.818/3.906 ms
vboxuser@CyberLab:~$
```

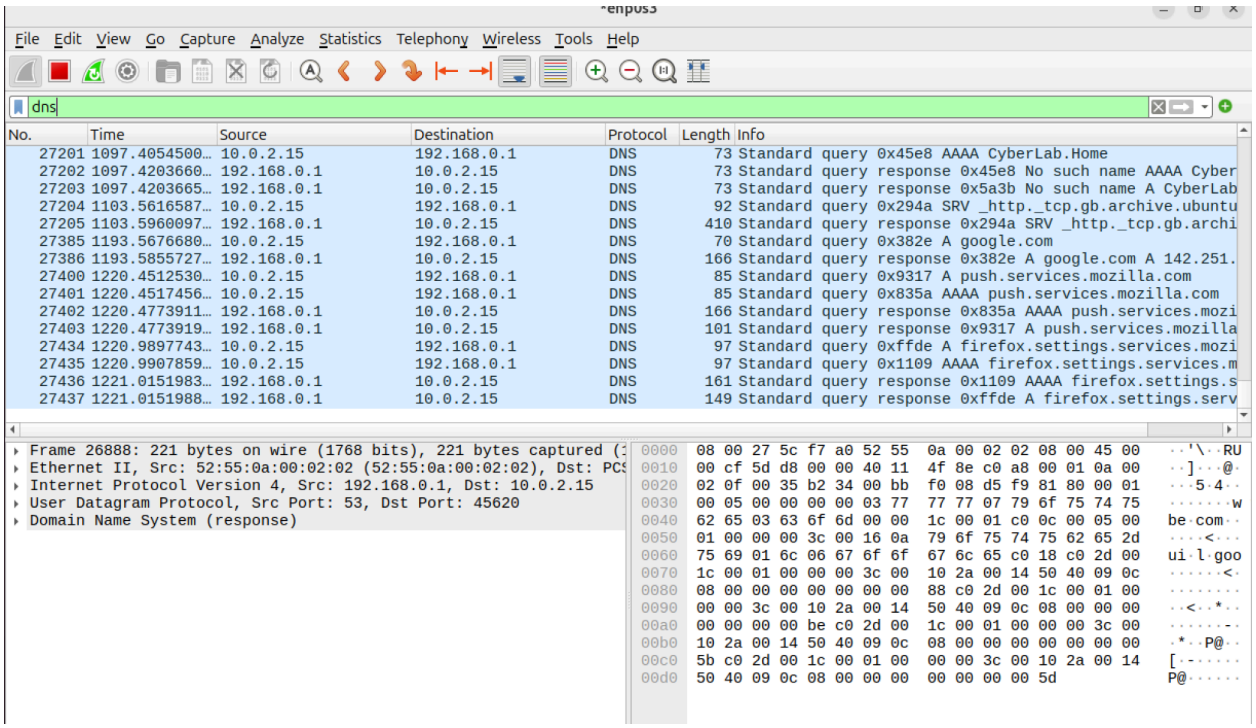


Terminal Command: ping -c 4 8.8.8.8 Wireshark Filter Used: icmp (Filtering for Internet Control Message Protocol)

- **Action Demonstrated:** The ping command successfully tested reachability to the destination server (8.8.8.8), as confirmed by the 0% packet loss in the terminal output.
- **Key Findings:** The capture isolates the fundamental network exchange: **four ICMP Echo Requests** (sent by the host, shown in light blue) immediately followed by **four ICMP Echo Replies** (returned by the server, shown in light pink/purple). The alternating sequence confirms successful, bi-directional connectivity.
- **Security Value:** This demonstrates the ability to use and analyze the most basic diagnostic tool, which is foundational for all **network discovery and confirming that necessary firewall rules (e.g., allowing ICMP) are correctly implemented.**

18. Expert DNS Reconnaissance (dig)

```
vboxuser@CyberLab:~$ dig google.com

; <<>> DiG 9.18.39-0ubuntu0.24.04.2-Ubuntu <<>> google.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 22046
;; flags: qr rd ra; QUERY: 1, ANSWER: 6, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 65494
;; QUESTION SECTION:
;google.com.                    IN      A

;; ANSWER SECTION:
google.com.             168     IN      A       142.251.30.138
google.com.             168     IN      A       142.251.30.102
google.com.             168     IN      A       142.251.30.139
google.com.             168     IN      A       142.251.30.101
google.com.             168     IN      A       142.251.30.113
google.com.             168     IN      A       142.251.30.100

;; Query time: 17 msec
;; SERVER: 127.0.0.53#53(127.0.0.53) (UDP)
;; WHEN: Sat Nov 29 21:27:11 UTC 2025
;; MSG SIZE  rcvd: 135

vboxuser@CyberLab:~$
```

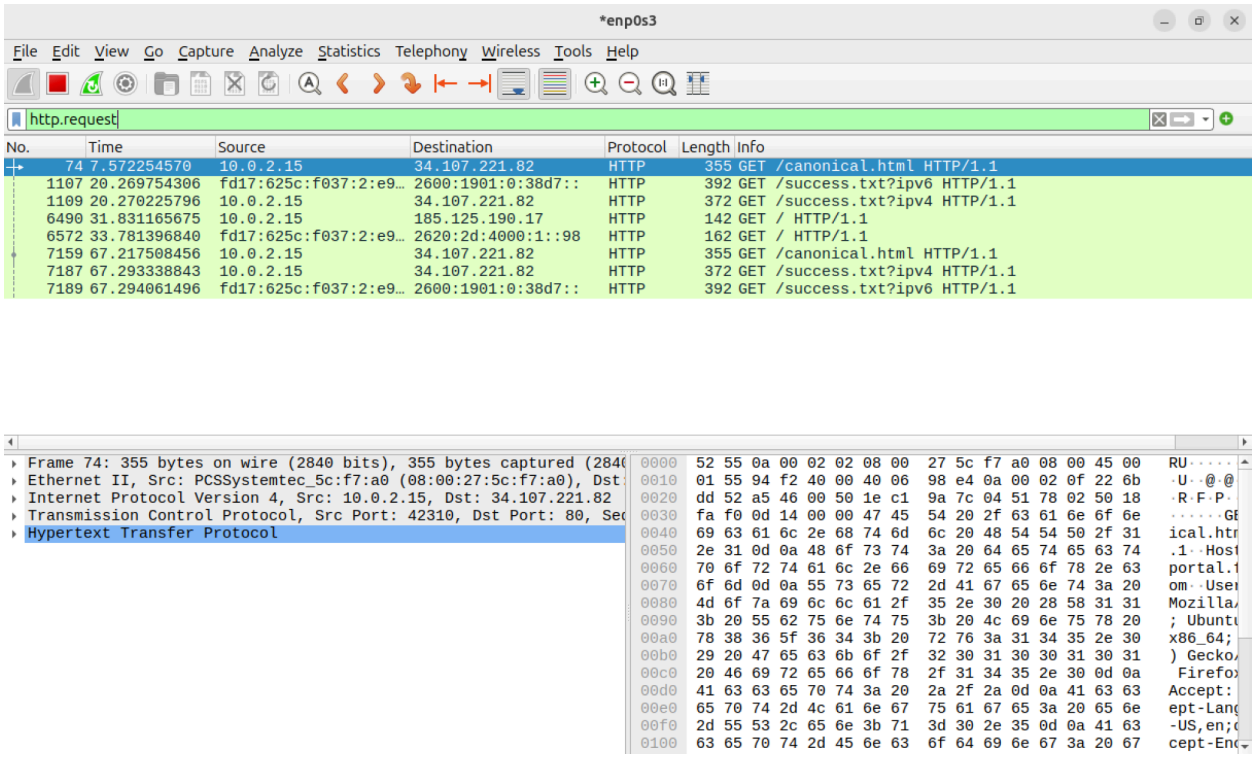


Terminal Command: dig google.com Wireshark Filter Used: dns (Filtering for Domain Name System traffic)

- **Action Demonstrated:** The execution of the dig (Domain Information Groper) command is the most advanced method for querying DNS servers. It is the preferred tool for administrators and security analysts over simpler tools like nslookup.
- **Key Findings**: The terminal output (and corresponding packets in Wireshark) provides extensive detail, including the full HEADER and ANSWER SECTION (multiple A records for google.com). It confirms the query status is NOERROR and provides key information like the server's ID and Query Time.
- **Security Value:** This demonstrates expert-level reconnaissance skills. Dig is essential for penetration testing and forensic analysis, allowing analysts to manually verify DNS records, check for misconfigurations, and map a target's infrastructure precisely.

19. Application-Layer Testing and HTTP Redirection (curl)

```
vboxuser@CyberLab:~$ curl http://youtube.com
```

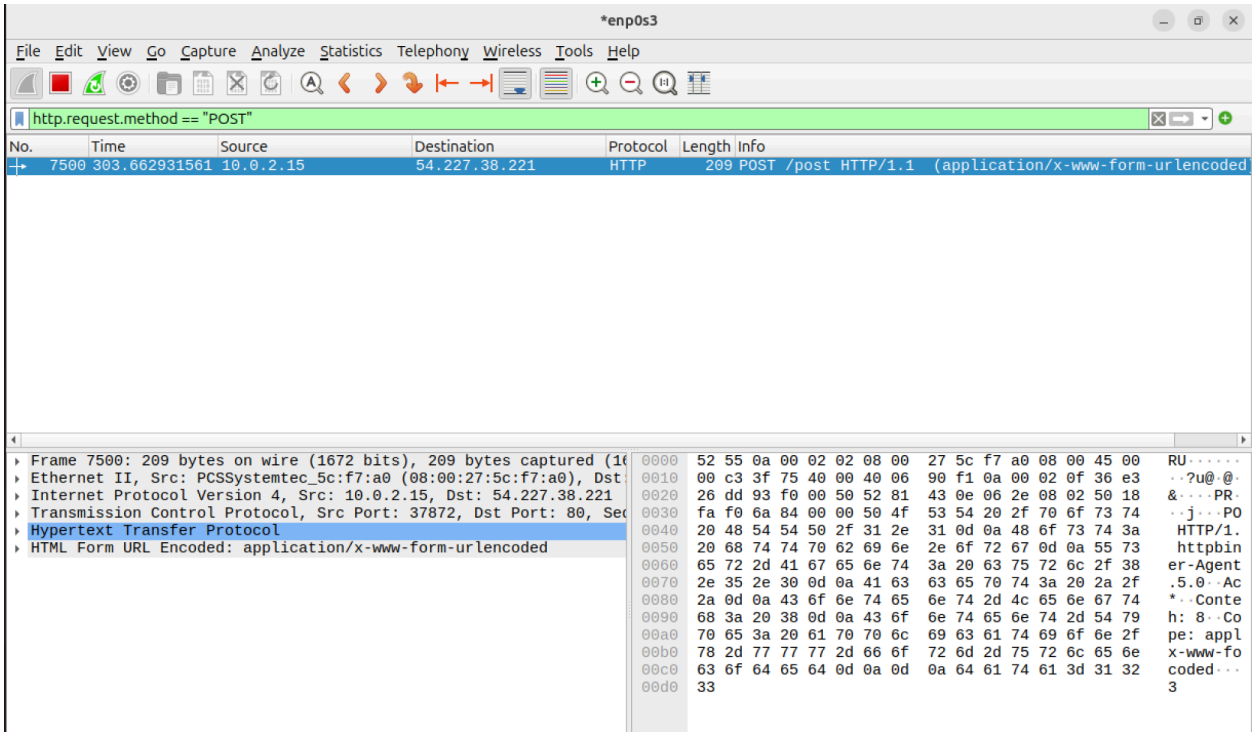


Terminal Command: curl http://youtube.com Wireshark Filter Used: http.request (Filtering for all client HTTP requests)

- **Action Demonstrated:** The **curl** command is used to directly test the network service at the application layer (HTTP), forcing an immediate GET request for the root path of youtube.com.
- **Key Findings:** The capture successfully isolates the **HTTP GET requests** sent by the host. These packets epresent the raw, unencrypted attempt to connect. The server's response (not fully visible here, but implied by the curl output) would be a **HTTP 301 or 302 redirect**, forcing the client to switch to the secure **HTTPS** version.
- **Security Value:** This demonstrates the ability to force a specific application-layer event and capture its response. This technique is crucial for **testing web server configurations** and verifying that insecure HTTP requests are correctly redirected to secure HTTPS.

20. Application-Layer Data Exfiltration Simulation (POST Request)

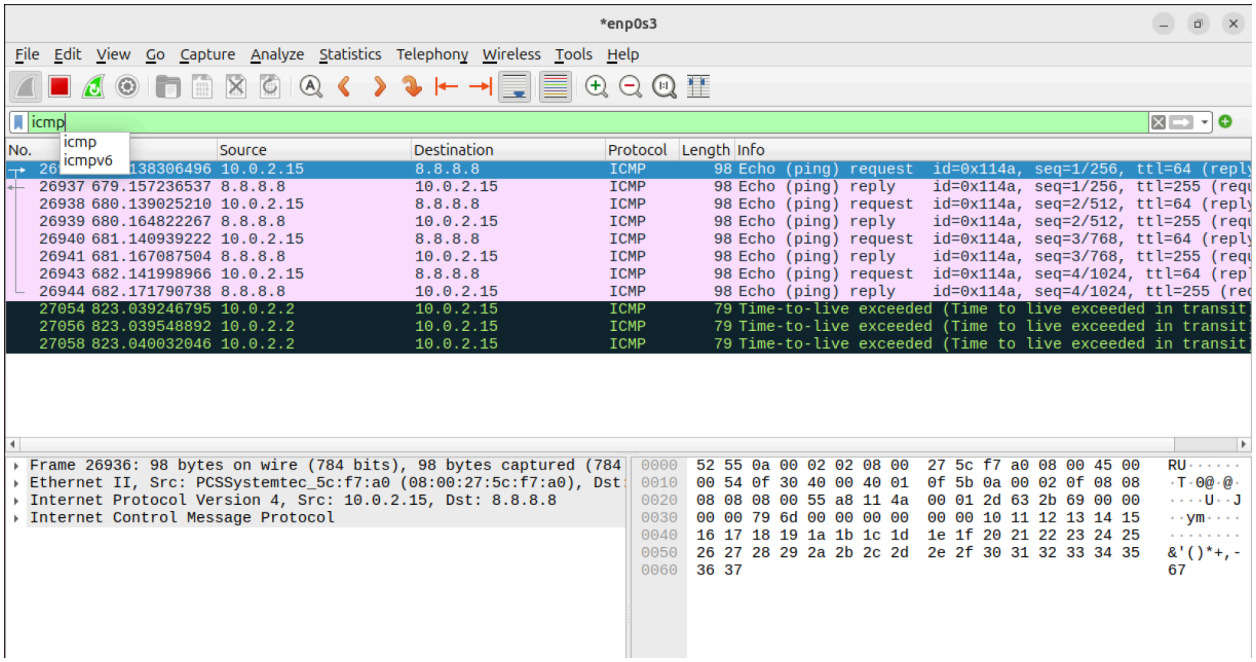
```
vboxuser@CyberLab:~$ curl -X POST httpbin.org/post -d "data=123"
{
  "args": {},
  "data": "",
  "files": {},
  "form": {
    "data": "123"
  },
  "headers": {
    "Accept": "*/*",
    "Content-Length": "8",
    "Content-Type": "application/x-www-form-urlencoded",
    "Host": "httpbin.org",
    "User-Agent": "curl/8.5.0",
    "X-Amzn-Trace-Id": "Root=1-692b869d-1d81181f495466dc38a3666d"
  },
  "json": null,
  "origin": "151.227.124.200",
  "url": "http://httpbin.org/post"
}
vboxuser@CyberLab:~$
```



Terminal Command: curl -X POST httpbin.org/post -d "data=123" **Wireshark Filter Used:** http.request.method == "POST" (Filtering for HTTP POST requests)

- **Action Demonstrated:** The use of the curl command with the -X POST flag simulates a client submitting data to a web server. This is the mechanism used by login forms and data submission pages.
- **Key Findings:** The capture successfully isolates the **HTTP POST request**. The analysis pane shows the HTTP packet containing the unencrypted payload: data=123 (visible under HTML Form URL Encoded in the packet details).
- **Security Value:** This demonstrates the ability to simulate and analyze a core mechanism of **data exfiltration**. Since the traffic is HTTP (unencrypted), it proves that an analyst can intercept and view sensitive data being transmitted, highlighting the absolute necessity of enforcing HTTPS for all data submissions.

21. Filtering based on Internet Control Message Protocol




```
vboxuser@CyberLab:~$ traceroute google.com
Command 'traceroute' not found, but can be installed with:
sudo apt install inetutils-traceroute  # version 2:2.4-3ubuntu1, or
sudo apt install traceroute            # version 1:2.1.5-1
vboxuser@CyberLab:~$ sudo apt install inetutils-traceroute # version 2:2.4-3ubuntu1
[sudo] password for vboxuser:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following package was automatically installed and is no longer required:
  libllvm19
Use 'sudo apt autoremove' to remove it.
The following NEW packages will be installed:
  inetutils-traceroute
0 upgraded, 1 newly installed, 0 to remove and 1 not upgraded.
Need to get 42.2 kB of archives.
After this operation, 113 kB of additional disk space will be used.
Get:1 http://gb.archive.ubuntu.com/ubuntu noble/universe amd64 inetutils-traceroute amd64 2:2.5-3ubuntu4 [42.2 kB]
Fetched 42.2 kB in 0s (242 kB/s)
Selecting previously unselected package inetutils-traceroute.
(Reading database ... 151783 files and directories currently installed.)
Preparing to unpack .../inetutils-traceroute_2%3a2.5-3ubuntu4_amd64.deb ...
Unpacking inetutils-traceroute (2:2.5-3ubuntu4) ...
Setting up inetutils-traceroute (2:2.5-3ubuntu4) ...
update-alternatives: using /usr/bin/inetutils-traceroute to provide /usr/bin/traceroute (traceroute) in auto mode
Processing triggers for man-db (2.12.0-4build2) ...
vboxuser@CyberLab:~$ traceroute google.com
traceroute to google.com (142.251.30.113), 64 hops max
  1   10.0.2.2  0.006ms  0.004ms  0.003ms
```

- **What it shows:** The filter isolates ICMP packets, including echo requests and replies generated by ping and traceroute commands. One highlighted example is an ICMP Echo Request sent from 10.0.2.15 to 8.8.8.8.
- **Key Findings:** TThe selected packet (Frame 23096) represents a standard ping request carrying 98 bytes of data.

    - The protocol stack is clearly visible in the middle pane — Ethernet, IP, and ICMP — confirming the packet's structure and destination.
    - Later packets display "Time-to-live exceeded" messages, which confirm that traceroute is functioning as expected.
- **Security Value:** Inspecting ICMP traffic helps verify the integrity and intent of diagnostic activity. Since ICMP is often used for reconnaissance, understanding its structure is crucial for detecting scans, spoofing attempts, or unusual routing behavior that could signal malicious activity.

What I learned

Protocol behavior: How DNS, HTTP, HTTPS/TLS, ICMP, TCP, NTP, and QUIC traffic looks at the packet level.

Filter mastery: Using Wireshark filters (IP, port, flags, frame size, string search, application-layer methods) to isolate specific traffic and strip away noise.

Security insights: The difference between insecure plaintext traffic (HTTP, POST data) and secure encrypted traffic (TLS, QUIC), plus the risks of misconfigured IPv6 services.

Forensic correlation: Linking terminal commands (`ping`, `nslookup`, `dig`, `curl`, `openssl`, `traceroute`) directly to the packets they generate, proving cause-and-effect.

Diagnostic skills: Reading error messages (ICMP unreachable, TCP resets) and understanding what they reveal about connectivity, routing, and firewall behavior.

Professional documentation: Turning raw packet captures into beginner-friendly explanations with captions, showing both technical depth and communication ability.

Conclusion

This Wireshark project provided hands-on experience in capturing, filtering, and analyzing real network traffic across protocols such as DNS, HTTP, HTTPS/TLS, ICMP, TCP, NTP, and QUIC. By linking terminal commands directly to packet behavior, I demonstrated both technical depth and the ability to communicate findings clearly. The project highlights key cybersecurity skills including protocol analysis, forensic investigation, and secure communication validation — all of which strengthen my foundation for a career in cybersecurity and prepare me to tackle real-world challenges in network defense and threat hunting.