**Introduction to Digital Certificates and Trust**

**Week-5, Day-4**

**Team-C**

* 28805- Deep Raghuwanshi
* 28806- Kundan Keshav Gawande
* 28807- Prajwal Thangan
* 28809- Devashish Lokhande
* 29080- V.S Goptherya
* 29084- Thulasi Ram

**Content:**

1. Introduction to Digital Certificates and Trust - Deep

2. Attack Simulation (Conceptual)- Deep

3.Observe and Identify- Kundan, Thulasi

4.Basic IR plan- Prajwal

5. Mitigation- V.S

6. Simple Policy- Devashish

7. Concise RCA- Devashish

**28805- Deep Raghuwanshi**

**Introduction to Digital Certificates and Trust**

**🔐 A. What Are Digital Certificates?**

A digital certificate is an electronic document that uses cryptography to verify the identity of a device, server, or user, similar to a physical passport or driver's license.

**Purpose:**

Digital certificates are used to verify the identity of entities in online interactions, ensuring that users, devices, and websites are who they claim to be.

**How it works:**

They rely on Public Key Infrastructure (PKI) and cryptography to bind a public key to a specific entity, like a website, user, or server.

**Issuance:**

Certificates are issued by trusted Certificate Authorities (CAs) that verify the identity of the certificate holder.

**Contents:**

A digital certificate includes details like the certificate holder's information, the issuing CA's information, the public key, and the certificate's validity period.

**Examples:**

* + Websites use digital certificates (like TLS/SSL certificates) to establish secure, encrypted connections (HTTPS) with browsers.
  + Digital signatures, used for email and documents, rely on digital certificates to verify the sender's identity.
  + Credit and debit cards use chip-embedded digital certificates for secure transactions.

**Key Benefits:**

* **Authentication:** Confirms the identity of the sender or device.
* **Encryption:** Enables secure communication by encrypting data during transmission.
* **Integrity:** Ensures data hasn't been tampered with during transfer.
* **Non-repudiation:** Prevents the sender from denying they sent a message or transaction.

**B. HTTPS vs HTTP:**

| Feature | HTTP | HTTPS |
| --- | --- | --- |
| Encryption | ❌ No | ✅ Yes (via SSL/TLS) |
| Data Integrity | ❌ Susceptible to tampering | ✅ Ensured |
| Authentication | ❌ None | ✅ Via digital certificate |
| Browser Indicator | "Not Secure" | 🔒 Lock icon |

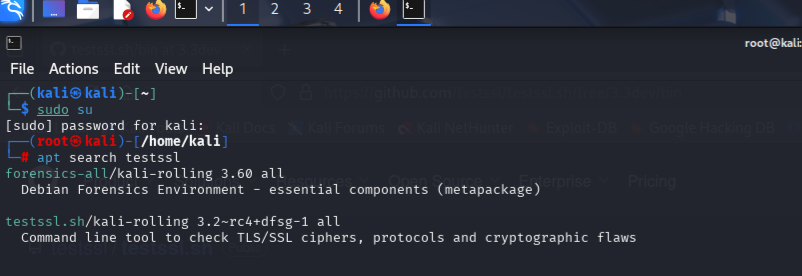
**C. Certificate Anatomy:**

* Issued To (Domain)
* Issued By (CA e.g., DigiCert, Let's Encrypt)
* Valid From / To
* Public Key
* Signature Algorithm
* Chain of Trust (Intermediate → Root CA)

**How to do SSL configuration of any Website**

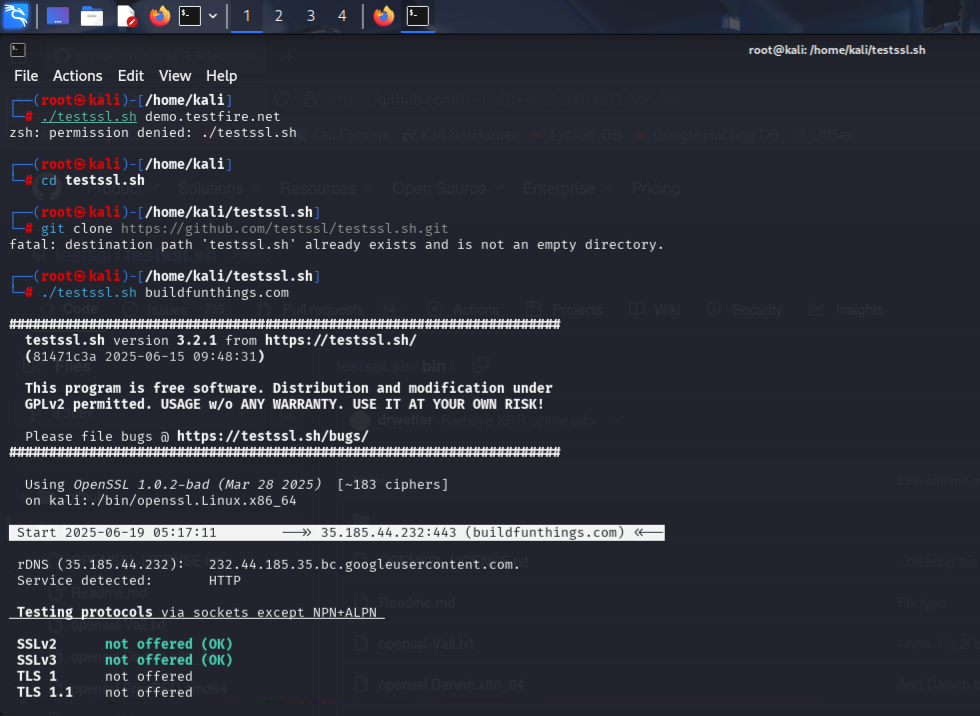
1. First Open kali Linux

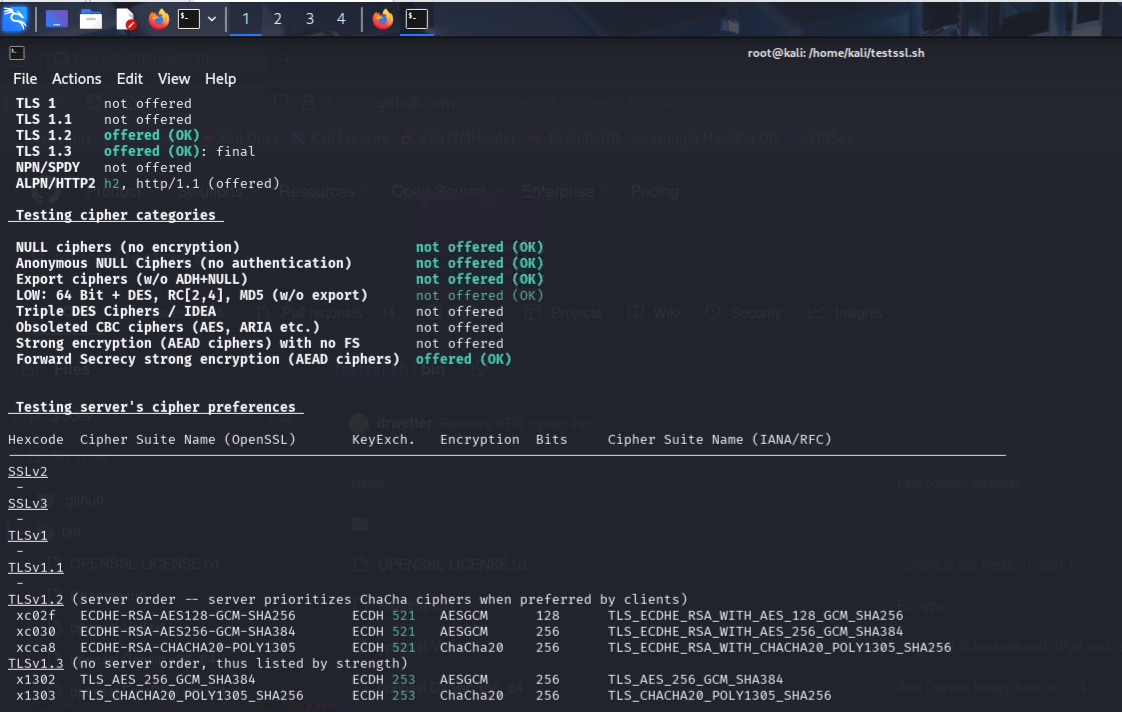
2. Then type apt kali search testssl

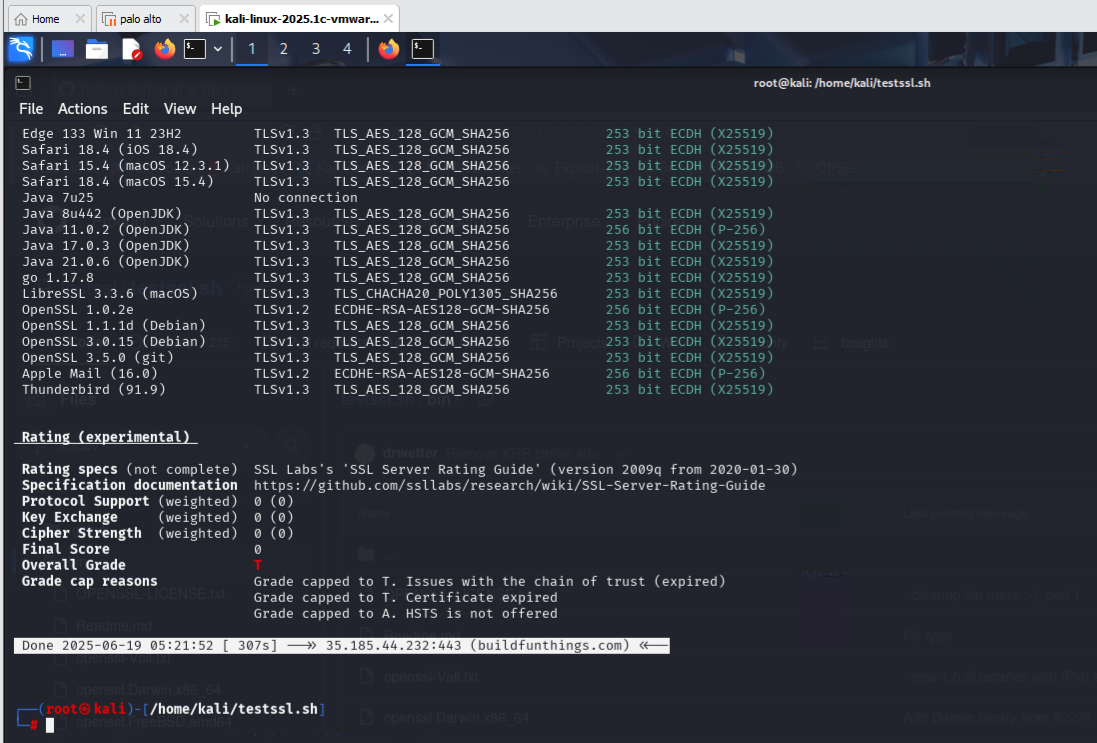


3. git clone from github of testssl.sh

4. then ./testssl.sh buildfunthings.com

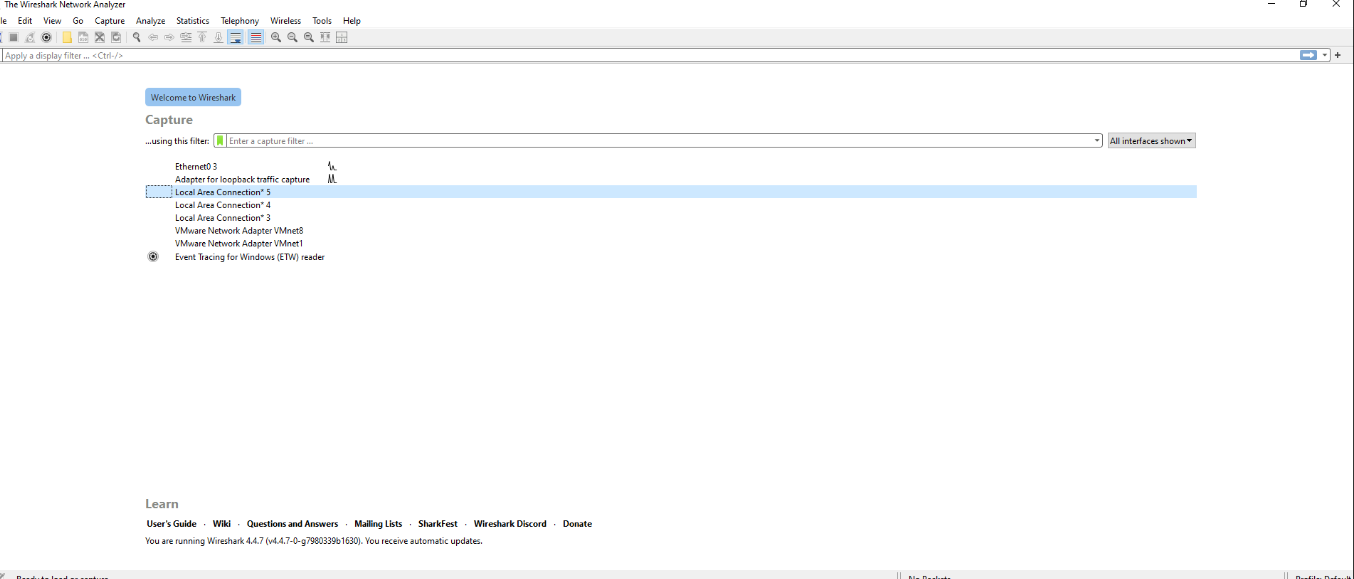




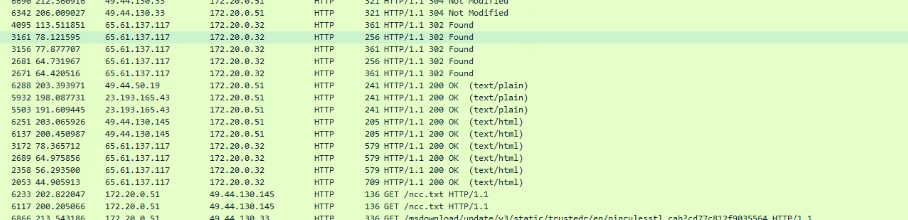


**28806- Kundan Keshav Gawande**

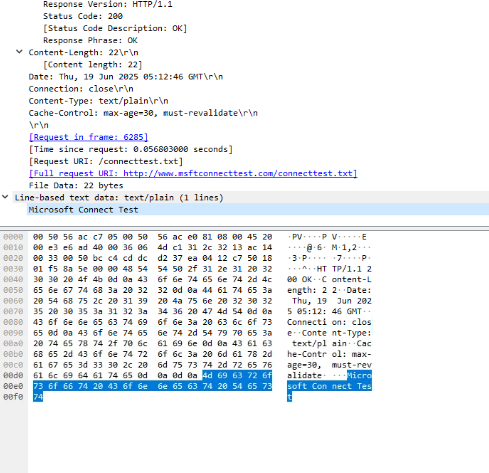
* **Monitoring ‘HTTP’ and ‘HTTPS’ Traffic Using Wireshark**
* **HTTP**
* Open Wireshark, Select Ethernet 0

****

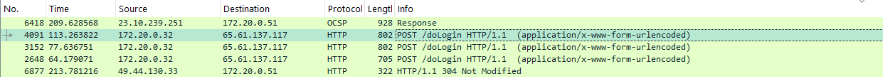
* **Step No.2:-**



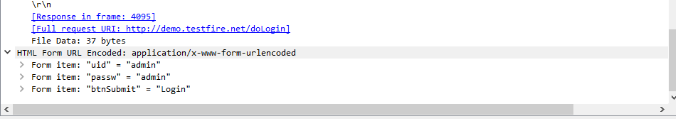
* **Step No.3:-**



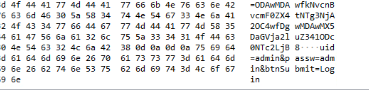
* **Step No.4:-**

****

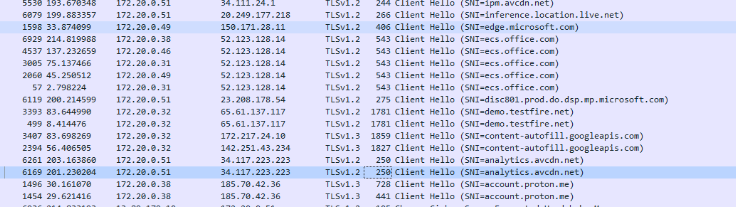
* **Step No.5:-**

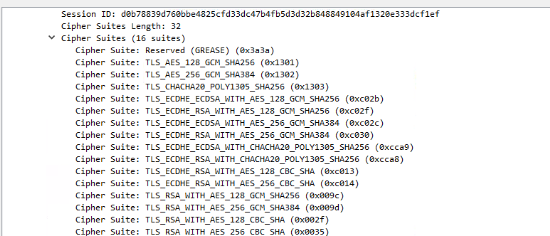
****

* **Step No.6:-**

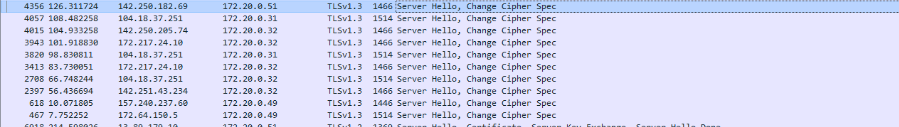
****

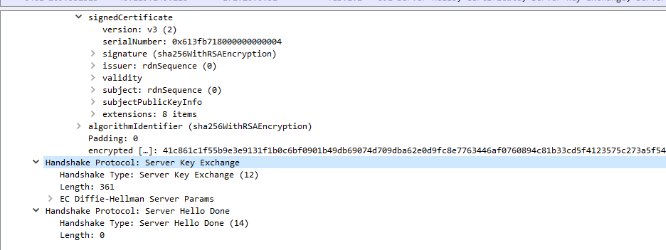
* **HTTPS**
* **Client Hello**

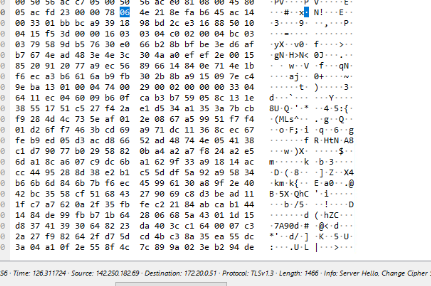
****

****

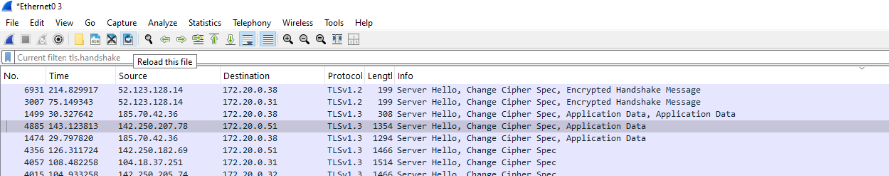
* **Server Hello**

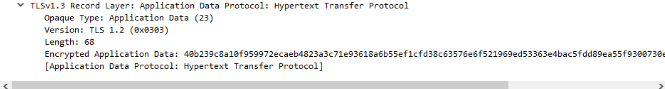
****

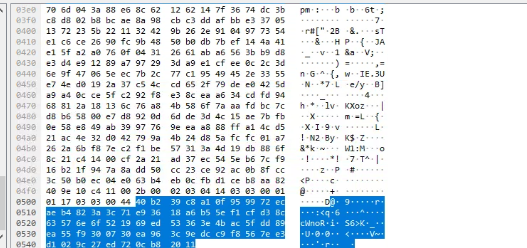
****

****

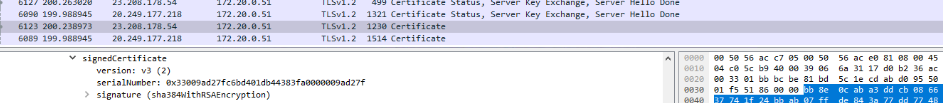
* **Application Data:-**

****

****

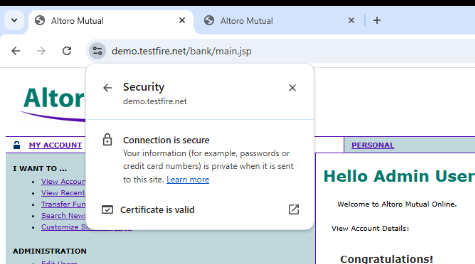
****

* **Certificate :-**

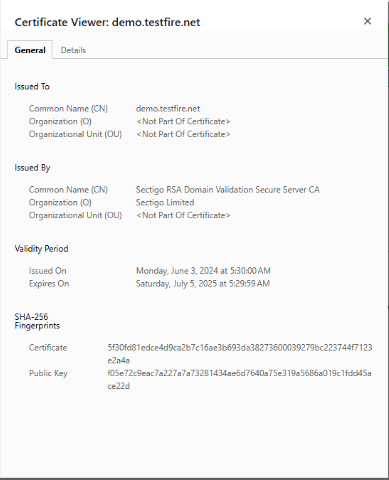
****

****

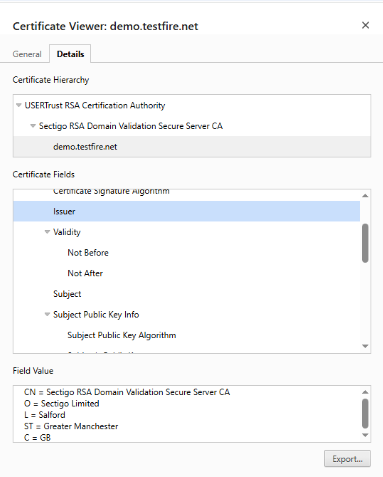
* **Certificate Details**
* **Secure Site:-**

****

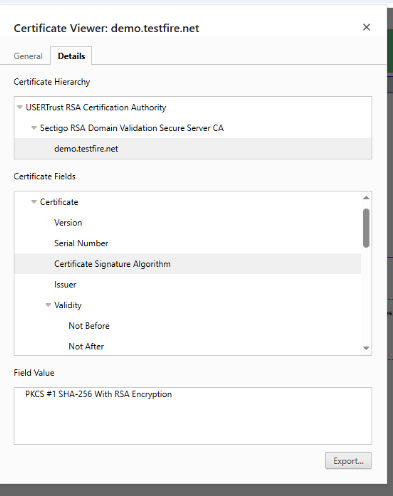
* **General Details:-**

****

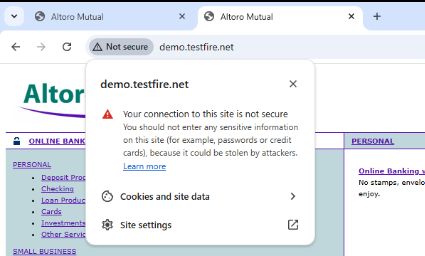
* **Issuer Details:-**

****

* **Certificate Signature Algo:-**

****

* **Not Secure:-** [**Http://demo.testfire.net**](Http://demo.testfire.net)

****

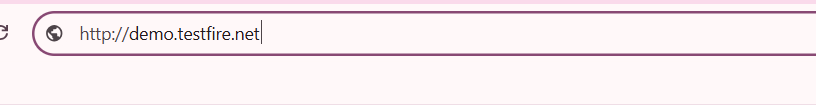
* **Vulnerabilities for ‘HTTP’ VS ‘HTTPS’**
* **For HTTP:- Unencrypted Credentials in HTTP**
* **Vulnerability: Man-in-the-middle attackers can steal or get access to your credentials.**
* **Sensitive data (username/password) is transmitted in plain text**
* **Anyone on the same network (e.g., Wi-Fi) can capture it using Wireshark**
* **Secure Behavior: HTTPS Encrypts the Data**
* **HTTPS prevents packet sniffers from seeing any credentials or sensitive content**
* **Wireshark only shows "Encrypted Application Data"**
* **Attackers can't see credentials because they're encrypted with TLS**
* **Vulnerability: Invalid / Self-Signed Certificates**
* **Some HTTPS sites use self-signed or expired certificates**
* **These sites trigger browser warnings like: "Your connection is not private"**
* **Invalid certificates mean the identity of the server cannot be trusted — attackers can impersonate legitimate sites (MITM attack)**.
* **Vulnerability: Mixed Content (HTTPS site loading HTTP resources)**
* **A secure page is weakened because it loads insecure content (like images or scripts from HTTP sources). These parts can be modified by attackers even though the main page is HTTPS.**

|  |  |  |  |
| --- | --- | --- | --- |
| Vulnerability | Protocol | Demonstration Tool | Risk |
| Plain-text credentials | HTTP | Wireshark | Credentials easily stolen |
| Encrypted data | HTTPS | Wireshark | Protected (not a vulnerability) |
| Self-signed/expired cert | HTTPS | Browser | Cannot trust identity of site |
| Mixed content | HTTPS | Browser | Secure site loading insecure data |

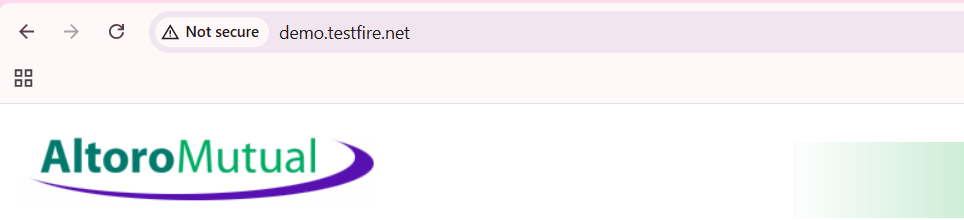
**29084- Thulasi Ram**

**Identify and Observe:**

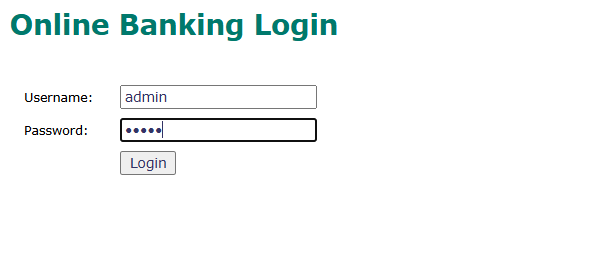
HTTP Site

****

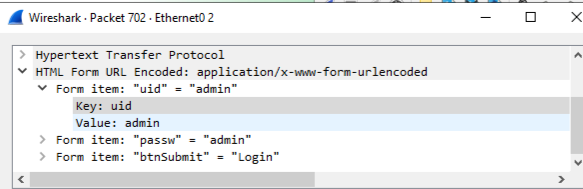
Here It Showing not Secure

****

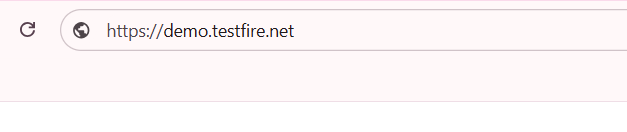
It is not secure to login in HTTP site

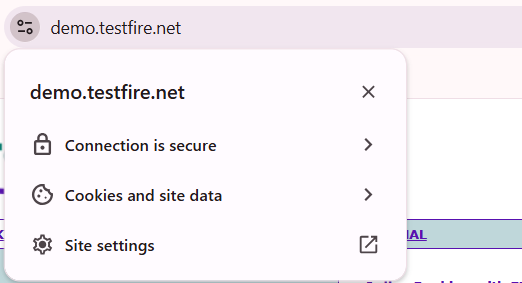
****

**Capturing HTTP traffic by Wireshark-**

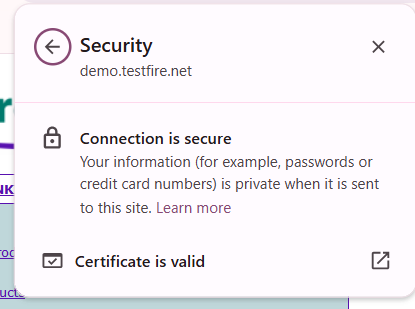
****

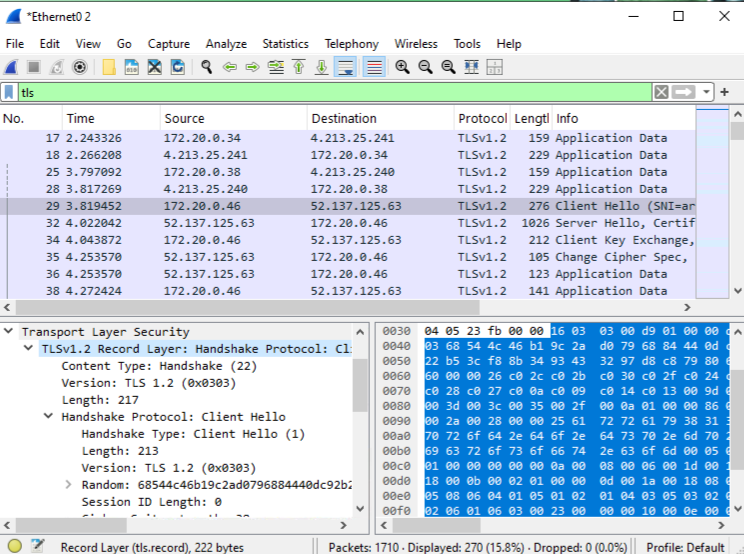
**HTTPS Site**

****

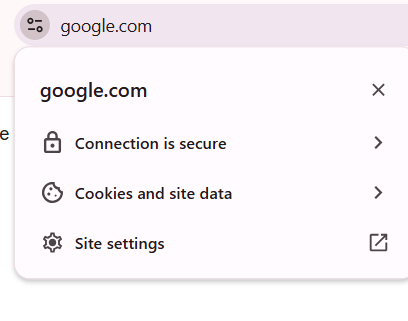
****

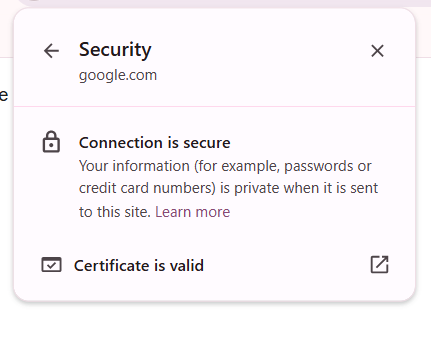
**Connection is secure**

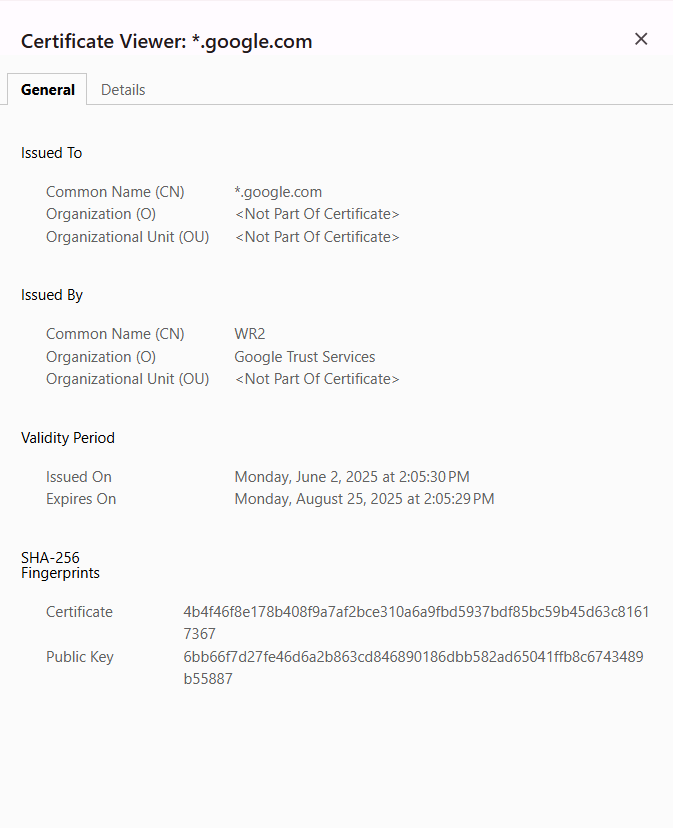
****

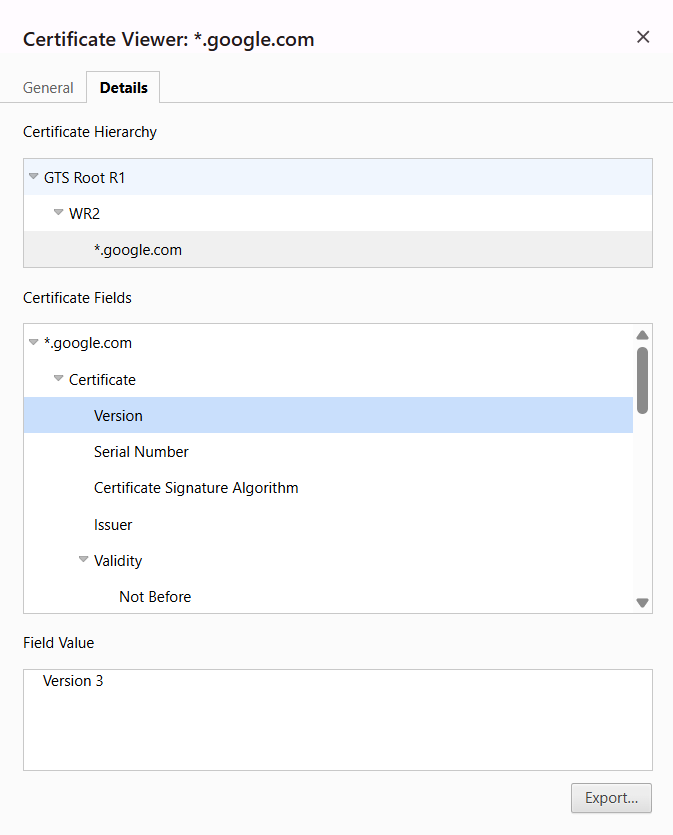
****

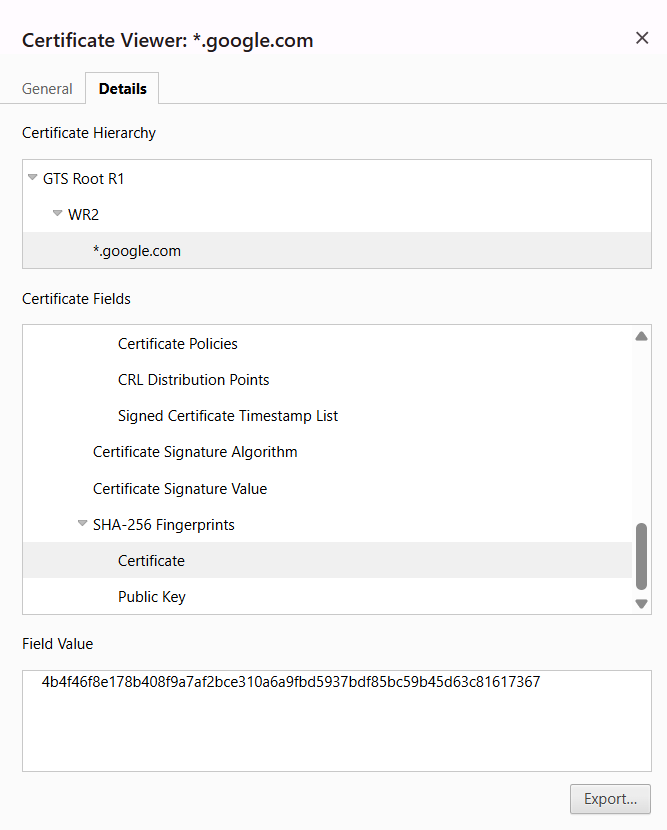
**Certificate details for a trusted HTTPS site-**

****

****

****

****



**28807- Prajwal Thangan**

Incident Response (IR) Plan for Project: Introduction to Digital Certificates and Trust

**1. Introduction**

This Incident Response (IR) plan is part of the project titled 'Introduction to Digital Certificates and Trust'. The project explores the importance of digital certificates in secure communication over the internet and highlights the differences between HTTP and HTTPS protocols. This IR plan outlines the steps to understand, identify, and respond to communication vulnerabilities associated with HTTP, and emphasizes the implementation of HTTPS to mitigate risks.

**2. Objective of the IR Plan**

To provide a structured response strategy to the inherent vulnerabilities of HTTP (unencrypted) communication and to reinforce the adoption of HTTPS using SSL/TLS and valid digital certificates.

**3. Incident Response Plan Phases**

**3.1 Preparation**

- Educate users on the risks of using HTTP.  
- Configure browsers and systems to flag HTTP connections.  
- Maintain an inventory of all web applications and their SSL status.

**3.2 Identification**

- Regularly inspect websites for HTTPS implementation.  
- Use browser indicators (lock icon or 'Not Secure') to identify HTTP sites.  
- Inspect SSL certificate details including issuer, validity, and trust chain.

**3.3 Containment**

- Disable access to HTTP-only web pages handling sensitive data.  
- Configure firewall rules to block outbound HTTP traffic if needed.

**3.4 Eradication**

- Migrate all HTTP sites to HTTPS.  
- Remove outdated or invalid digital certificates.  
- Ensure SSL/TLS configurations meet current security standards.

**3.5 Recovery**

- Validate the proper functioning of HTTPS after implementation.  
- Conduct security tests using tools like SSL Labs to evaluate certificate strength.  
- Ensure end-users are informed about the secure status of websites.

**3.6 Lessons Learned**

- Review the incident response effectiveness.  
- Update policies and configurations to enforce HTTPS usage.  
- Document the process and improvements for future reference.

**4. Practical Instructions**

- Visit an HTTPS site, click the lock icon, and view certificate details (issuer, expiry, trust chain).  
- Visit an HTTP site and observe the 'Not Secure' warning.  
- Use tools like OpenSSL, SSL Labs, or browser developer tools to inspect certificates.  
- Implement HTTPS in test environments using free certificates (e.g., Let’s Encrypt).

**Real-World Example: Incident Response to HTTP Vulnerability:**

A notable example of incident response to HTTP-related vulnerabilities comes from the early days of e-commerce, when major websites began transitioning from HTTP to HTTPS in response to increasing cyber threats and user concerns.

**Case Study: Google's Migration to HTTPS**

In 2014, Google began encouraging websites to move to HTTPS by announcing that HTTPS would be used as a ranking signal in search results. This came after several incidents where user data was intercepted on HTTP connections, especially on public Wi-Fi.

Google's Incident Response Plan (IRP) involved several key actions:  
- Preparation: Developed internal security standards and HTTPS adoption strategy.  
- Identification: Identified user data exposure risks over HTTP from multiple incident reports.  
- Containment: Flagged HTTP sites as 'Not Secure' in Chrome browser and restricted sensitive operations.  
- Eradication: Migrated all major services (e.g., Gmail, Search) to HTTPS.  
- Recovery: Implemented HSTS (HTTP Strict Transport Security) to enforce HTTPS.  
- Lessons Learned: Promoted global HTTPS adoption through tools, documentation, and search engine policies.

This real-world example highlights the effectiveness of an IRP in addressing the risks posed by unencrypted communication and accelerating a secure web ecosystem.

**5. Conclusion**

Understanding the security differences between HTTP and HTTPS is essential. This IR plan enables informed observation, response, and mitigation strategies to secure communication using digital certificates and enforce HTTPS protocols.

**29080- V.S Goptherya**

**Mitigation Policy : Block HTTP and Allow Only HTTPS**

**📜 Policy Statement**

All web applications must block unencrypted HTTP traffic and allow only secure HTTPS communication to protect data in transit.

**Now Let’s do Practical Implementation Using iptables**

**🡪 Step-by-step (block incoming & outgoing port 80)**

* Block outgoing HTTP requests:

**sudo iptables -A INPUT -p tcp --dport 80 -j REJECT**

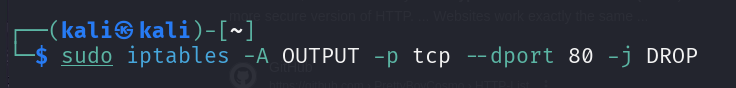
**sudo iptables -A OUTPUT -p tcp --dport 80 -j REJECT**

* You can also choose **DROP** instead of **REJECT** if you want it to silently fail:

**sudo iptables -A INPUT -p tcp --dport 80 -j DROP**

**sudo iptables -A OUTPUT -p tcp --dport 80 -j DROP**

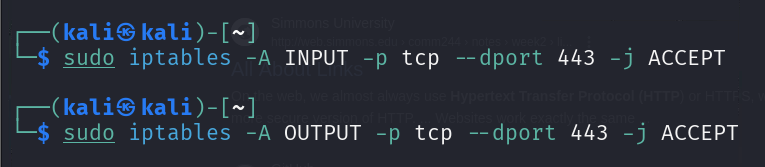




* **Allow HTTPS Outgoing**

Make sure you allow HTTPS to still work:

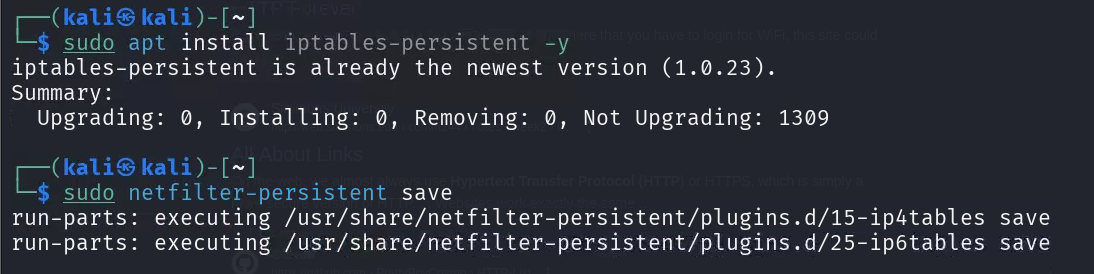
* **sudo iptables -A INPUT -p tcp --dport 443 -j ACCEPT**
* **sudo iptables -A OUTPUT -p tcp --dport 443 -j ACCEPT**



* **Make Rules Persistent**

To make your iptables rules survive reboot:

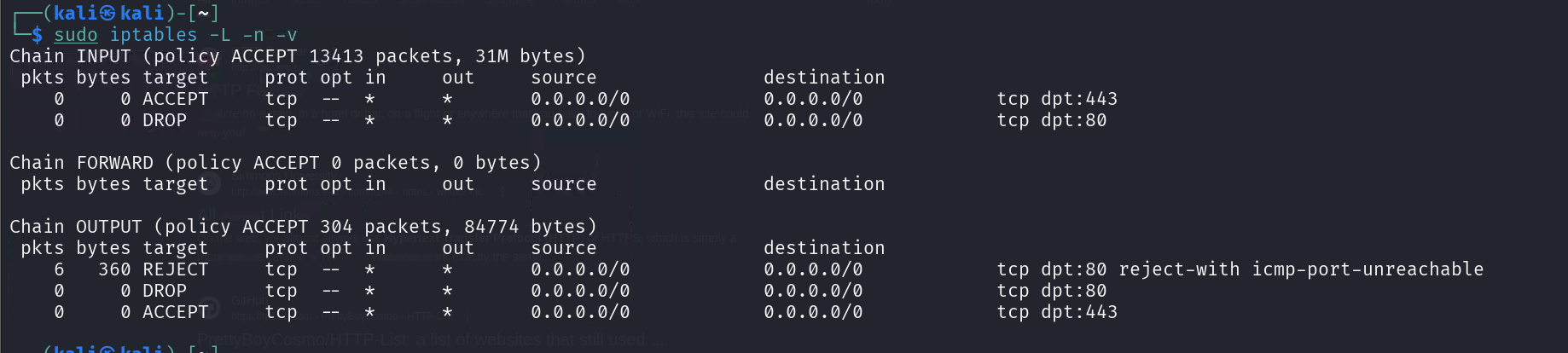
* **sudo apt install iptables-persistent**
* **sudo netfilter-persistent save**



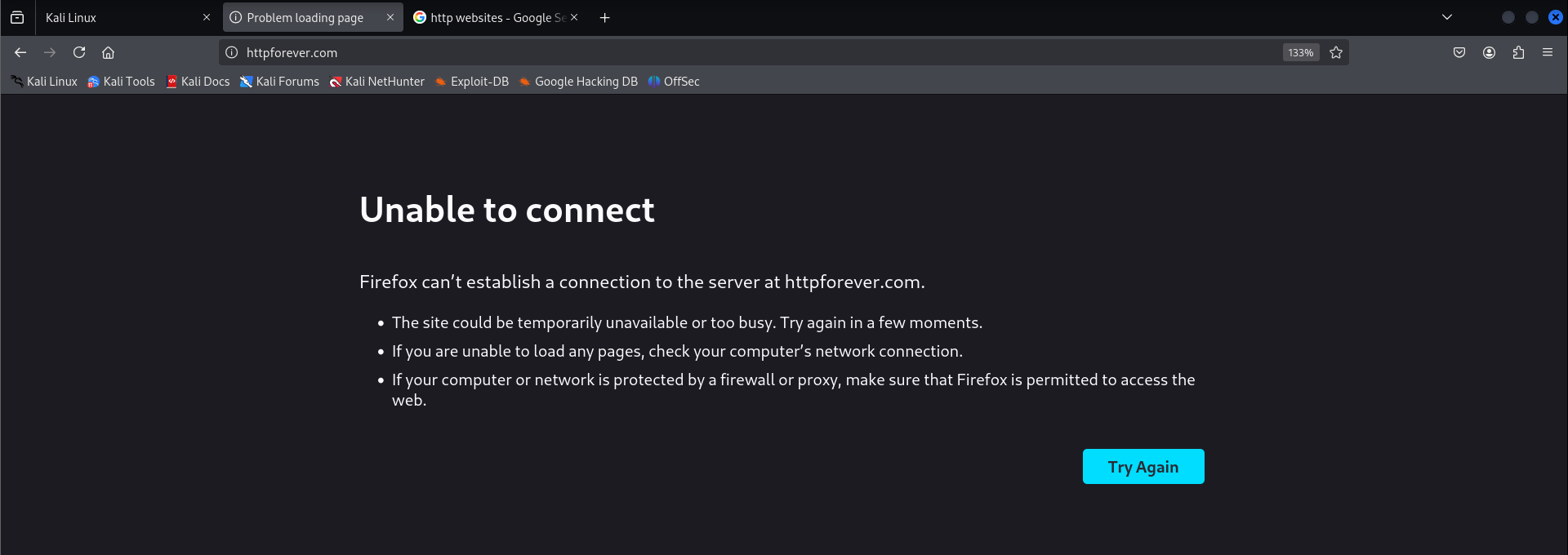
* **sudo iptables -L -n –v**

is used to **display the current firewall rules** configured using iptables on a Linux system. Here's a breakdown of what each part does:

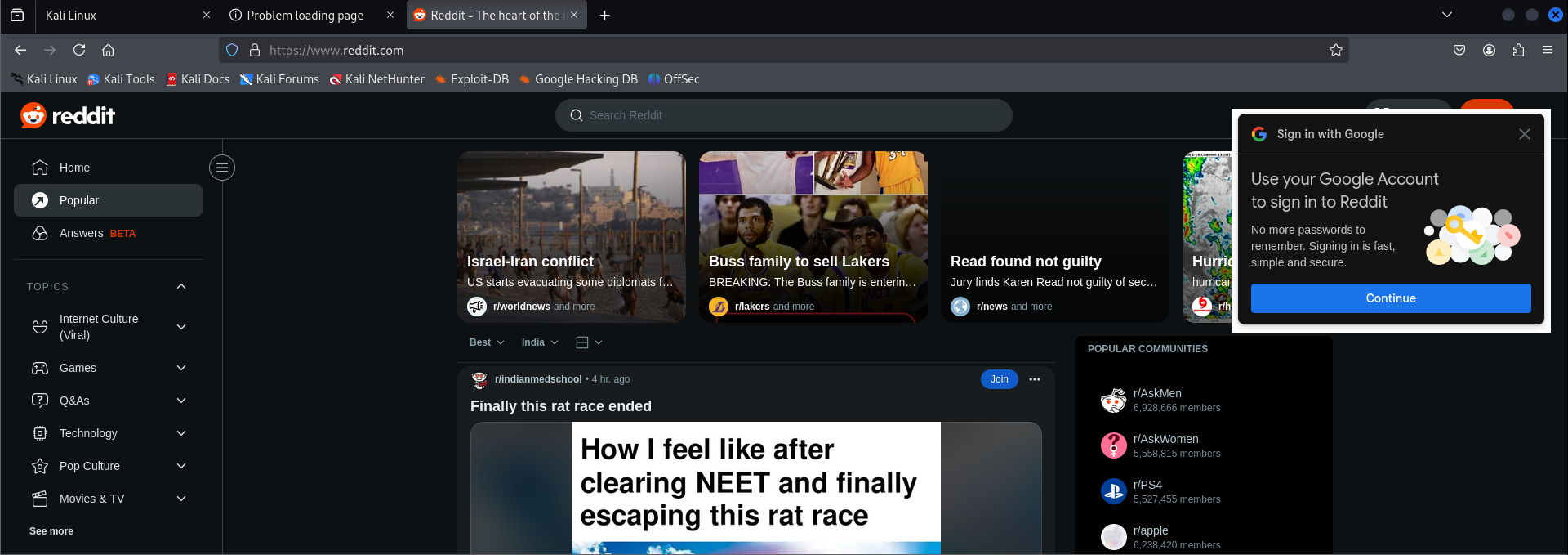
* **sudo**: Runs the command with root (administrator) privileges.
* **iptables**: The Linux command-line firewall tool.
* **-L**: Lists all current firewall rules.
* **-n**: Shows IPs and ports in numeric form (no DNS/service name resolution).
* **-v**: Displays detailed info like packet counts and interfaces.



* **Lets check whether it’s working or not.**
* Try to open any **http** website. (http://httpforever.com)



* Now try opening **https** website. (https://www.reddit.com)



**🡪 Summary**

|  |  |
| --- | --- |
| **Rule Purpose** | **Command** |
| Block **incoming HTTP** | sudo iptables -A INPUT -p tcp --dport 80 -j DROP |
| Block **outgoing HTTP** | sudo iptables -A OUTPUT -p tcp --dport 80 -j DROP |
| Allow **outgoing HTTPS** | sudo iptables -A OUTPUT -p tcp --dport 443 -j ACCEPT |

**28809- Devashish Lokhande**

**Simple Policy**

**Introduction**

This report focuses on a critical security rule: "Any website that handles private data must use HTTPS with a valid certificate." This rule helps ensure that data transmitted between users and websites is encrypted, preventing unauthorized access, tampering, or eavesdropping.

**What is HTTPS and SSL/TLS?**

HTTPS (Hypertext Transfer Protocol Secure) is the secure version of HTTP. It uses SSL/TLS (Secure Sockets Layer / Transport Layer Security) protocols to encrypt data during transmission. A digital certificate (SSL certificate) is issued by a trusted Certificate Authority (CA) and proves that the website is authentic and secure.

**Risk of Not Using HTTPS**

When a website uses HTTP, data is transferred in plain text. This makes it vulnerable to:  
- Man-in-the-Middle (MITM) attacks  
- Credential theft  
- Data leakage  
  
For example, login credentials sent via HTTP can be captured by anyone monitoring the network.

**Real Life Example**

In 2015, a major mobile app transmitted user data over HTTP instead of HTTPS, and it led to leaked login credentials.

You can search and include short case studies like: Yahoo HTTP login leaks Any old version of Facebook or Instagram before HTTPS enforcement

**Policy Recommendation**

All websites, especially those handling sensitive information (logins, payments, personal details), must implement HTTPS using a valid SSL/TLS certificate. This policy helps ensure data confidentiality, integrity, and trustworthiness.

**How To Get SSL/TLS Certificate**

Let’s Encrypt (https://letsencrypt.org/): A popular, free, automated, and open certificate authority. Ideal for personal blogs, small business websites, and non-commercial projects. Many web hosting providers support automatic Let's Encrypt integration.

Paid Certificates: Offered by trusted Certificate Authorities (CAs) such as:

DigiCert, GoDaddy, Comodo/Sectigo, GlobalSign Prices vary based on type (e.g., DV, OV, EV) and features like warranty or wildcard support.

Types of Certificates:

DV (Domain Validation): Basic security, quick issuance.

OV (Organization Validation): Includes organization identity verification.

EV (Extended Validation): Highest level of trust, shows organization name in browser.

**Conclusion**

The use of HTTPS is a basic yet essential step in securing web applications and protecting user privacy. This report demonstrated the importance of HTTPS, tools to analyze it, and risks of ignoring this rule. Always look for the lock icon 🔒 before entering sensitive data online.

**Tools used**

**1.** [**https://testssl.sh/**](https://testssl.sh/)

**2.** [**https://github.com/testssl/testssl.sh**](https://github.com/testssl/testssl.sh)

**3.** [**https://board.flexibleir.com/b/2djkqueZ7KWptJWJQ/introduction-to-digital-certificate-and-trust**](https://board.flexibleir.com/b/2djkqueZ7KWptJWJQ/introduction-to-digital-certificate-and-trust)

**4.** [**https://demo.testfire.net/**](https://demo.testfire.net/)