HTTP vs HTTPS: Wireshark & Simulation-Based Comparison

# Overview

This project explores and compares HTTP and HTTPS protocols using real-world tools like Wireshark, Cisco Packet Tracer, and a local server demo. We highlight the security risks of HTTP and how HTTPS protects sensitive data like usernames and passwords during transmission.

# Table of Contents

* 1. Theory: HTTP vs HTTPS
* 2. TLS, SSL, and Encryption Deep Dive
* 3. How TLS Encryption Works
* 4. MTU (Maximum Transmission Unit) Behavior
* 5. Real-World Security Risks (When Not Using HTTPS)
* 6. Wireshark Demo – HTTP
* 7. Wireshark Demo – HTTPS
* 8. Cisco Packet Tracer Simulation
* 9. Local HTTPS Server Demo
* 10. Conclusion
* 11. References

Ahmed Moataz 22-101092

Ismail Taher 21-

# 1. Theory: HTTP vs HTTPS

|  |  |  |
| --- | --- | --- |
| Feature | HTTP | HTTPS |
| Protocol | HyperText Transfer Protocol | HyperText Transfer Protocol Secure (HTTP over TLS/SSL) |
| Port | Port 80 by default | Port 443 by default |
| Encryption | No encryption — data is sent in plaintext | Uses TLS (formerly SSL) to encrypt data end-to-end |
| Authentication | No verification of server identity | Uses digital certificates to verify server authenticity |
| Data Visibility | Anyone intercepting traffic can read it (MITM risk) | Data is encrypted; third parties can’t see the content |
| Performance | Slightly faster due to no encryption overhead | Slightly slower due to encryption, but often negligible with HTTP/2 |
| Security | Vulnerable to eavesdropping, tampering, and impersonation | Protects against MITM, tampering, and data theft |
| SEO Ranking | Not favored by search engines | Google and others give ranking boosts to HTTPS sites |
| Browser Indicator | “Not secure” warning in modern browsers | Shows padlock icon and sometimes “Secure” label |
| Use Cases | OK for public, non-sensitive sites (e.g., blogs, static content) | Required for login forms, e-commerce, banking, any sensitive data |
| Certificate Needed | None | Needs SSL/TLS certificate (can be free via Let’s Encrypt or self-certified using OpenSSL) |
| Content Integrity | Data can be modified in transit without detection | TLS ensures integrity — tampering is detectable |
| Compliance | Often non-compliant with data protection standards (e.g., GDPR, PCI) | HTTPS is required for compliance in many regulatory frameworks |

# 2. TLS, SSL, and Encryption Deep Dive

## SSL vs TLS

TLS (Transport Layer Security) is the modern protocol that powers HTTPS today. It evolved from SSL (Secure Sockets Layer), which is now considered obsolete and insecure.

|  |  |  |
| --- | --- | --- |
| Feature | SSL | TLS |
| First Released | 1995 | 1999 (TLS 1.0) |
| Latest Version | SSL 3.0 (1996) | TLS 1.3 (2018) |
| Secure? | No — deprecated | Yes — modern and safe |
| Status | Deprecated | Actively maintained |

Fun Fact: Most browsers and APIs completely disabled SSL/TLS 1.0–1.1 after 2020 due to serious vulnerabilities like POODLE and BEAST.

## TLS Versions

* ***TLS 1.0 / 1.1:*** Deprecated and insecure
* ***TLS 1.2:*** Still widely used; default for most apps today
* ***TLS 1.3:***
  + Released in ***2018***
  + Faster handshake (1 round trip)
  + Removes insecure algorithms (e.g., RSA key exchange)
  + Reduces attack surface for MITM or downgrade attacks

# 3. How TLS Encryption Works

## Step 1: Handshake Phase

* Client sends a ***ClientHello*** message with supported TLS versions and cipher suites
* Server replies with a ***ServerHello***, selects the version/cipher, and sends its certificate
* Key exchange happens (ECDHE/DHE) to establish a shared secret
* A session key is derived for encryption

Asymmetric encryption is only used during this phase. Afterward, the connection switches to symmetric encryption using the session key.

## Step 2: Symmetric Encryption Phase

* All data is encrypted using fast symmetric encryption (e.g., AES-GCM)
* Ensures:
* ***Confidentiality*** (no one can read the data)
* ***Integrity*** (tampering is detectable)
* ***Authentication*** (client knows it’s the right server)

# 4. MTU (Maximum Transmission Unit) Behavior

## What is MTU?

* MTU is the largest packet size (in bytes) that a network link can transmit in one piece.
* Typical Ethernet MTU = 1500 bytes

## MTU Effects on HTTP vs HTTPS

* HTTPS packets include extra TLS overhead (e.g., encryption, record layer)
* Large TLS-encrypted data may exceed the MTU, causing:
* ***IP fragmentation*** (split into smaller packets)
* Or ***TCP segmentation*** if ***MTU is respected at application level***
* Or ***ICMP errors*** if ***fragmentation is blocked*** (Don't Fragment bit set)

## Real-world Impact

* Packet fragmentation can increase latency and decrease performance
* In some misconfigured networks, it can even break TLS handshakes
* TLS 1.3 helps mitigate this by using shorter handshake messages

Our custom Python HTTPS server includes code to simulate this! We use Content-Length, send large payloads, and analyze fragmentation in Wireshark.

# 5. Real-World Security Risks (When Not Using HTTPS)

## Without HTTPS

* ***MITM Attacks:*** Attackers can intercept, read, or modify content
* ***Credential Theft:*** Login forms over HTTP expose usernames/passwords in plaintext
* ***Session Hijacking:*** Cookies can be stolen to impersonate users
* ***Content Injection:*** ISPs or attackers can inject ads or malware into HTTP pages

## Known Incidents

* ***Heartbleed (2014):*** Bug in OpenSSL let attackers read server memory (TLS handshake secrets!)
* ***Firesheep (2010):*** Browser extension that hijacked HTTP sessions on public Wi-Fi
* ***NSA/PRISM leaks:*** Mass surveillance exploited unencrypted web traffic

# 6. Wireshark Demo – HTTP

## Objective

Demonstrate how HTTP POST data (like usernames and passwords) can be intercepted and read using Wireshark.

## Tools Used

* Python HTTP server
* HTML login form
* Wireshark

## Details

Submitted:

* username=hello123
* password=thisisapassword

## Screenshot

A computer screen shot of a black screen

AI-generated content may be incorrect.

## Analysis

***HTTP*** *transmits data in* ***plaintext****. Any attacker* ***with access to the network (or a packet capture)*** *can* ***easily read sensitive information*** *like* ***login credentials.***

# 7. Wireshark Demo – HTTPS

## Objective

Show that HTTPS encrypts traffic, preventing attackers from reading POST data.

## Tools Used

* Python HTTPS server with self-signed certificate
* Wireshark

## What You See in Wireshark

* TLS Handshake
* Encrypted Application Data instead of visible POST content

## A screenshot of a computer AI-generated content may be incorrect.Screenshot

## Analysis

*The* ***same*** *login form is used, but the data is* ***encrypted****. Wireshark* ***cannot show the username or password****, proving* ***HTTPS prevents sniffing attacks.***

# 8. Cisco Packet Tracer Simulation

## Objective

Simulate how HTTP and HTTPS behave in a network environment.

## 

## Setup

* Devices: PC, Switch, Web Server, DNS Server
* Protocols: HTTP and symbolic HTTPS
* Ports: 80 (HTTP), 443 (HTTPS)

## 📸 Screenshots

* A screenshot of a computer

  AI-generated content may be incorrect.HTTP server:
* A computer network diagram with text

  AI-generated content may be incorrect.HTTPS setup:

## Notes

*Cisco Packet Tracer* ***doesn't*** *simulate real TLS encryption, but it allows you to* ***show protocol differences*** *and* ***port usage visually.***

# 9. Local HTTPS Server Demo

## Objective

Run a secure HTTPS server locally using Python and a self-signed SSL certificate to demonstrate encrypted traffic.

## Steps

1. Generate certificate:

openssl req -new -x509 -keyout cert.key -out cert.pem -days 365 -nodes

1. Run Python script:

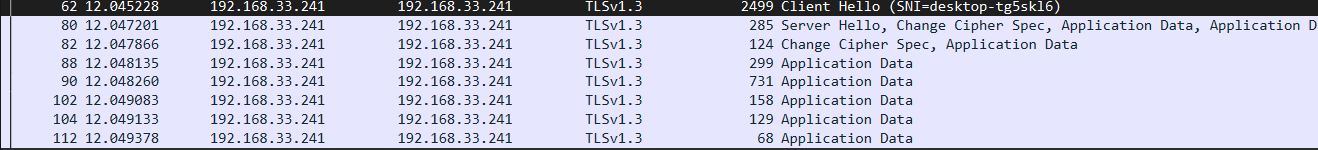
python secure\_server.py

1. Open in browser:

https://localhost:4443

A screenshot of a computer error

AI-generated content may be incorrect.Self-certificate warning:

Wireshark TLS traffic:

## Outcome

*Data submitted via HTTPS is not readable in Wireshark. The form and credentials are protected by encryption—even with a self-signed certificate.*

# 10. Conclusion

* HTTP is insecure: Anyone can intercept and read data.
* HTTPS is essential: It encrypts data and validates server identity.
* Wireshark clearly shows the difference: one is open, the other locked down.
* Even in basic demos, the security benefits of HTTPS are clear and measurable.
* Always use HTTPS for any form-based or sensitive interactions.

# 11. References

* 🔗 Wireshark User Guide
* 🔗 Python HTTP Server Docs
* 🔗 TLS 1.3 – RFC 8446
* 🔗 HTTP 1.1 – RFC 2616
* 🔗 Cisco Packet Tracer