

Project Report: SSL/TLS Analyzer

Repository:

https://github.com/mayank24000/University_Assignments/tree/main/Crypto_Assignments/ssl_tls_analyzer

1. Introduction

The **ssl_tls_analyzer** is a Python-based command-line tool designed to inspect the SSL/TLS certificates of remote servers. It demonstrates how secure connections are established and allows users to view the cryptographic details of a server's digital certificate, including the **Issuer**, **Subject**, and **Validity Period**.

2. Prerequisites & Dependencies

To run this code, the following standard Python libraries are utilized. No external pip installations are strictly required as these are built-in:

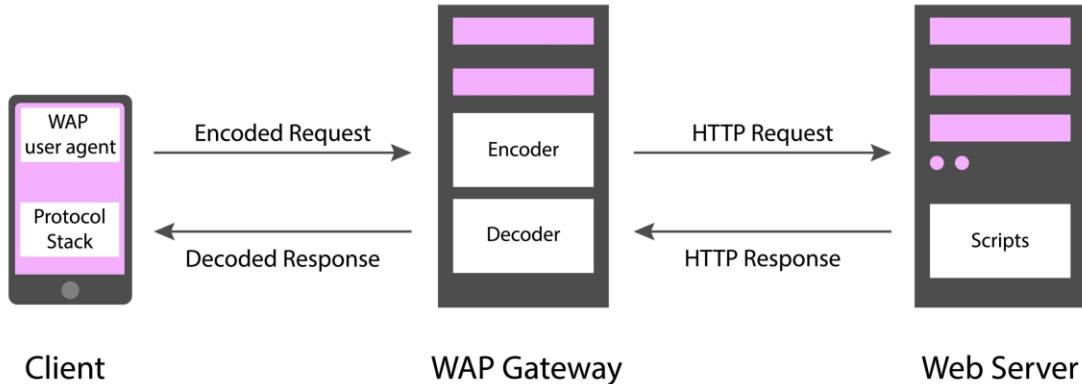
- **socket**: Used for low-level network communication (TCP/IP).
 - **ssl**: Python's standard library for handling SSL/TLS protocols.
 - **sys**: Used to handle command-line arguments.
 - **pprint**: Used for "pretty printing" the dictionary output of certificate details.
-

3. Code Logic & Workflow

The script (ssl_analyzer.py) follows a specific sequence to retrieve data:

1. **Argument Parsing**: The script accepts a hostname and an optional port (defaulting to **443** for HTTPS) via command-line arguments.
2. **Socket Creation**: A standard TCP socket is created using `socket.socket(socket.AF_INET, socket.SOCK_STREAM)`.
3. **SSL Wrapping**: The script creates a default SSL context using `ssl.create_default_context()`. It then wraps the standard TCP socket with this SSL context to create a secure connection.
4. **Server Connection**: The script connects to the target host.
5. **Handshake**: The `getpeercert()` method is called on the secure socket to retrieve the binary certificate data from the server.
6. **Data Extraction**: The binary data is converted into a human-readable format (dictionary) and printed to the console.

Wireless Application Protocol



4. Source Code Implementation

The following is the core code extracted from the repository:

Python

```
import socket  
import ssl  
import sys  
import pprint  
  
def get_certificate_info(hostname, port=443):  
    # Create a socket  
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)  
  
    # Wrap the socket with SSL  
    context = ssl.create_default_context()  
    ssl_sock = context.wrap_socket(sock, server_hostname=hostname)
```

```
try:
    # Connect to the server
    ssl_sock.connect((hostname, port))

    # Get the certificate
    cert = ssl_sock.getpeercert()

    # Close the connection
    ssl_sock.close()

    return cert
except Exception as e:
    return f"Error: {e}"

if __name__ == "__main__":
    if len(sys.argv) < 2:
        print("Usage: python ssl_analyzer.py <hostname> [port]")
        sys.exit(1)

    hostname = sys.argv[1]
    port = int(sys.argv[2]) if len(sys.argv) > 2 else 443

    print(f"Connecting to {hostname} on port {port}...")
    cert_info = get_certificate_info(hostname, port)

    print("\n--- Certificate Details ---")
    pprint.pprint(cert_info)
```

5. Key Code Components Explained

A. `ssl.create_default_context()`

This function creates an SSL context with default security settings. It ensures that the client (the script) validates the server's certificate against trusted Certificate Authorities (CAs), preventing **Man-in-the-Middle (MITM)** attacks.

B. `wrap_socket(sock, server_hostname=hostname)`

This is the critical step where the plain text socket is upgraded to an encrypted SSL/TLS socket. The `server_hostname` argument is crucial for **SNI (Server Name Indication)**, allowing the server to present the correct certificate if it hosts multiple websites on the same IP address.

C. `getpeercert()`

This method returns a dictionary containing the X.509 certificate details.

The output typically includes:

- **subject:** The entity the certificate belongs to (Common Name, Organization, etc.).
 - **issuer:** The Certificate Authority that signed the certificate.
 - **notBefore / notAfter:** The validity period of the certificate.
 - **serialNumber:** The unique ID of the certificate.
-

6. How to Run

1. Ensure **Python 3** is installed.
2. Save the code as `ssl_analyzer.py`.
3. Run the script via terminal:

Bash

```
python ssl_analyzer.py www.google.com
```

7. Conclusion

This repository serves as an effective educational tool for understanding the SSL/TLS handshake process. By bypassing high-level HTTP libraries (like `requests`) and using raw sockets, it provides a granular view of the certificate exchange that occurs before any actual data is transmitted securely.