how can we generate the traffic and extract the relevant features

**ChatGPT said:**

ChatGPT

To generate TLS traffic and extract relevant features for training a machine learning model, you can follow these steps:

**1. Traffic Generation**

Generating compliant and non-compliant TLS traffic requires simulating various network conditions, using real applications, and deliberately configuring some scenarios to violate compliance rules.

**Tools for Generating Traffic**

1. **Real Applications**
   * Use real-world applications like browsers (Chrome, Firefox) and servers (Apache, Nginx) for compliant traffic.
   * Misconfigure servers or use old libraries to generate non-compliant traffic (e.g., weak cipher suites, outdated TLS versions).
2. **Traffic Simulation Tools**
   * **Scapy**
     + A Python-based tool for crafting and sending custom packets.
     + Example: Create TLS handshakes and manipulate fields like cipher suites, versions, or certificate chains.
   * **OpenSSL**
     + Use OpenSSL commands to generate compliant and non-compliant TLS traffic:
       - Compliant: openssl s\_client -connect server:443 -tls1\_2
       - Non-compliant: Force old protocols or weak ciphers: openssl s\_client -connect server:443 -cipher DES-CBC3-SHA
   * **TRex Traffic Generator**
     + Cisco's high-performance tool can simulate bulk compliant and non-compliant traffic at scale.
   * **LOIC/HOIC** (Low Orbit Ion Cannon/High Orbit Ion Cannon)
     + While often associated with stress testing or DDoS simulations, they can create non-compliant TLS traffic patterns.
   * **BRO/Zeek Framework**
     + Monitor and analyze both synthetic and real traffic flows.
3. **Custom Servers**
   * Deploy two environments with:
     + Compliant Servers: Modern servers like Nginx or Apache with up-to-date TLS configurations.
     + Non-Compliant Servers: Misconfigured servers (e.g., self-signed certificates, deprecated ciphers).

**2. Capturing the Traffic**

Use packet capture tools to collect data during the generation process.

* **Wireshark/tshark**
  + Capture live traffic and export data in .pcap format for later analysis.
  + Example: tshark -i eth0 -w traffic.pcap
* **tcpdump**
  + A lightweight alternative to capture packets:
    - Example: tcpdump -i eth0 port 443 -w traffic.pcap

with nginx do i have to deploy it?

**ChatGPT said:**

ChatGPT

You don’t need to fully deploy an Nginx server for public access if you’re only testing traffic generation in a controlled environment. Instead, you can run it locally on a machine or within a virtualized environment. Here's how you can set it up without deploying it publicly:

**1. Install and Configure Nginx Locally**

You can install and run Nginx on your local machine or a virtual machine.

1. **Install Nginx**:
   * On Ubuntu:

bash

Copy code

sudo apt update

sudo apt install nginx

* + On CentOS/RHEL:

bash

Copy code

sudo yum install nginx

1. **Create a Self-Signed Certificate for Local Use**:
   * Generate a certificate and private key:

bash

Copy code

openssl req -x509 -nodes -days 365 -newkey rsa:2048 \

-keyout /etc/ssl/private/nginx-selfsigned.key \

-out /etc/ssl/certs/nginx-selfsigned.crt

1. **Edit the Nginx Configuration File**:
   * Open the Nginx configuration file:

bash

Copy code

sudo nano /etc/nginx/sites-available/default

* + Update the configuration to enable SSL/TLS locally:

nginx

Copy code

server {

listen 443 ssl;

server\_name localhost;

ssl\_protocols TLSv1.2 TLSv1.3;

ssl\_ciphers 'ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384';

ssl\_prefer\_server\_ciphers on;

ssl\_certificate /etc/ssl/certs/nginx-selfsigned.crt;

ssl\_certificate\_key /etc/ssl/private/nginx-selfsigned.key;

location / {

root /var/www/html;

index index.html;

}

}

1. **Start Nginx**:
   * Test the configuration:

bash

Copy code

sudo nginx -t

* + Start or reload Nginx:

bash

Copy code

sudo systemctl start nginx

sudo systemctl reload nginx

1. **Access the Server Locally**:
   * Use https://localhost in your browser or testing tools.
   * This setup ensures the server runs locally without exposing it to the internet.

**2. Run Nginx in a Docker Container**

If you don’t want to modify your local system, you can use Docker to run Nginx in an isolated environment.

1. **Install Docker**: Follow instructions for your OS from Docker's official site.
2. **Set Up an Nginx Container with SSL**:
   * Create a directory for Nginx configuration:

bash

Copy code

mkdir nginx-test

cd nginx-test

* + Create an nginx.conf file:

nginx

Copy code

server {

listen 443 ssl;

server\_name localhost;

ssl\_protocols TLSv1.2 TLSv1.3;

ssl\_ciphers 'ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384';

ssl\_prefer\_server\_ciphers on;

ssl\_certificate /etc/nginx/selfsigned.crt;

ssl\_certificate\_key /etc/nginx/selfsigned.key;

location / {

root /usr/share/nginx/html;

index index.html;

}

}

* + Create SSL certificates in the same directory:

bash

Copy code

openssl req -x509 -nodes -days 365 -newkey rsa:2048 \

-keyout selfsigned.key -out selfsigned.crt

* + Run the container:

bash

Copy code

docker run --name nginx-test -v $(pwd):/etc/nginx/conf.d:ro \

-v $(pwd):/etc/nginx:ro -p 443:443 -d nginx

1. **Test Locally**:
   * Access the containerized server at https://localhost.

knowing i have an ubuntu and a kali machine, how can i set up servers on them with different tls complaince metrics. give me the detailed implementation

**ChatGPT said:**

ChatGPT

You can set up Ubuntu and Kali machines to act as servers and clients to generate compliant and non-compliant TLS traffic. Here's a step-by-step guide:

**Step 1: Setting Up the Machines**

Ensure both Ubuntu and Kali machines are up and running with proper networking:

* **Ubuntu**: Used as the main TLS server.
* **Kali**: Used as a client for generating and testing traffic.

**Networking Configuration**

* **Bridged Network**: Allows both machines to communicate as if on the same physical LAN.
* Verify connectivity between the machines:

bash

Copy code

ping <other-machine-IP>

**Step 2: Install Web Servers on Ubuntu**

You’ll set up an Nginx server on Ubuntu and configure it for compliant and non-compliant TLS traffic.

**Install Nginx**

bash

Copy code

sudo apt update

sudo apt install nginx openssl

**Configure TLS for Compliant Traffic**

1. **Generate a Valid Certificate**

bash

Copy code

sudo mkdir -p /etc/nginx/ssl

sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 \

-keyout /etc/nginx/ssl/nginx.key \

-out /etc/nginx/ssl/nginx.crt \

-subj "/CN=secure.example.com"

1. **Edit the Nginx Configuration** Open /etc/nginx/sites-available/default:

bash

Copy code

sudo nano /etc/nginx/sites-available/default

Add the following:

nginx

Copy code

server {

listen 443 ssl;

server\_name secure.example.com;

ssl\_certificate /etc/nginx/ssl/nginx.crt;

ssl\_certificate\_key /etc/nginx/ssl/nginx.key;

# Compliant TLS settings

ssl\_protocols TLSv1.2 TLSv1.3;

ssl\_ciphers HIGH:!aNULL:!MD5;

ssl\_prefer\_server\_ciphers on;

location / {

root /var/www/html;

index index.html;

}

}

1. **Restart Nginx**

bash

Copy code

sudo systemctl restart nginx

**Configure TLS for Non-Compliant Traffic**

Create a separate Nginx configuration file for non-compliant traffic.

1. **Edit Configuration**

bash

Copy code

sudo nano /etc/nginx/sites-available/non\_compliant

Add the following:

nginx

Copy code

server {

listen 444 ssl;

server\_name insecure.example.com;

ssl\_certificate /etc/nginx/ssl/nginx.key;

ssl\_certificate\_key /etc/nginx/ssl/nginx.key;

# Non-compliant TLS settings

ssl\_protocols SSLv3 TLSv1.0;

ssl\_ciphers DES-CBC3-SHA:RC4-SHA;

location / {

root /var/www/html;

index index.html;

}

}

1. **Enable the Configuration** Link it to Nginx sites-enabled:

bash

Copy code

sudo ln -s /etc/nginx/sites-available/non\_compliant /etc/nginx/sites-enabled/

sudo systemctl restart nginx

1. **Test the Server**
   * Compliant traffic: https://<ubuntu-ip>:443
   * Non-compliant traffic: https://<ubuntu-ip>:444

sudo mkdir -p /etc/nginx/ssl sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 \ -keyout /etc/nginx/ssl/nginx.key \ -out /etc/nginx/ssl/nginx.crt \ -subj "/CN=secure.example.com" explain this

**ChatGPT said:**

ChatGPT

This sequence of commands sets up a directory for storing SSL/TLS keys and certificates, then generates a self-signed SSL/TLS certificate using OpenSSL. Here's a detailed breakdown:

**1. Create the Directory for SSL Certificates**

bash

Copy code

sudo mkdir -p /etc/nginx/ssl

* **sudo**: Executes the command with superuser privileges, required for modifying system directories.
* **mkdir -p**:
  + **mkdir**: Creates a new directory.
  + **-p**: Ensures that parent directories are created if they don't already exist. This prevents errors if the /etc/nginx/ssl path doesn't exist.
* **/etc/nginx/ssl**: The target directory where the SSL certificate and key will be stored. This is a common location for SSL/TLS files for the Nginx web server.

**2. Generate a Self-Signed Certificate**

bash

Copy code

sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 \

-keyout /etc/nginx/ssl/nginx.key \

-out /etc/nginx/ssl/nginx.crt \

-subj "/CN=secure.example.com"

**Command Breakdown**

* **sudo**: Runs the command with superuser privileges.
* **openssl**: A widely used command-line tool for creating and managing SSL/TLS keys and certificates.
* **req**: Starts a process to create and handle a Certificate Signing Request (CSR). The CSR contains the information used to generate the certificate.
* **-x509**:
  + Specifies that the certificate will be self-signed.
  + X.509 is a standard for public key certificates, used in SSL/TLS.
* **-nodes**:
  + Tells OpenSSL not to encrypt the private key with a passphrase.
  + Without this, Nginx would require a passphrase every time it starts, which can be inconvenient.
* **-days 365**:
  + Sets the certificate's validity period to 365 days (1 year).
  + After this period, the certificate will expire and need renewal.
* **-newkey rsa:2048**:
  + Generates a new private key and certificate request simultaneously.
  + **rsa:2048**: Specifies the RSA algorithm with a 2048-bit key size (sufficient for secure communication).
* **-keyout /etc/nginx/ssl/nginx.key**:
  + Specifies the output file for the private key.
  + This private key is stored securely at /etc/nginx/ssl/nginx.key.
* **-out /etc/nginx/ssl/nginx.crt**:
  + Specifies the output file for the generated self-signed certificate.
  + This certificate will be saved at /etc/nginx/ssl/nginx.crt.
* **-subj "/CN=secure.example.com"**:
  + Specifies the "Subject" field for the certificate.
  + **/CN=secure.example.com**:
    - CN stands for "Common Name."
    - This is typically the hostname or domain name the certificate is issued for.
    - In this example, the certificate is for secure.example.com.

**Result**

After running the command:

1. A private key (nginx.key) is saved in /etc/nginx/ssl/.
2. A self-signed certificate (nginx.crt) is also saved in /etc/nginx/ssl/.
3. The certificate can now be used by Nginx to establish secure HTTPS connections.

sudo apt install openssl=1.0.2 if i do this, will it affect the complaint server that i did before?

**ChatGPT said:**

ChatGPT

Yes, installing or downgrading OpenSSL to version 1.0.2 could potentially affect your compliant server configuration, depending on how your Nginx installation is linked to OpenSSL.

**Why It Might Affect Your Compliant Server**

1. **Nginx-OpenSSL Dependency**:
   * Nginx dynamically links to the system-installed OpenSSL library. If you downgrade OpenSSL, Nginx will use the older version for all SSL/TLS operations, including your compliant server.
   * OpenSSL 1.0.2 does not fully support TLS 1.3, so your compliant server configured with ssl\_protocols TLSv1.2 TLSv1.3 might fail.
2. **Cipher Support**:
   * OpenSSL 1.0.2 has fewer secure ciphers compared to newer versions. Your compliant server might not meet modern TLS standards.

**Options to Avoid Impacting the Compliant Server**

**Option 1: Compile a Separate Nginx with OpenSSL 1.0.2**

You can compile a second Nginx instance specifically linked to OpenSSL 1.0.2, leaving your system-installed Nginx untouched:

1. Download and compile OpenSSL 1.0.2.
2. Compile Nginx with the OpenSSL 1.0.2 library.
3. Use this custom Nginx binary exclusively for the non-compliant server.

**Option 2: Use Containers for Non-Compliant Traffic**

Run a container with older versions of OpenSSL and Nginx for non-compliant traffic while keeping your compliant server unaffected. See the **Docker Option** in the previous response.

**Option 3: Use a Separate Virtual Machine**

Set up a virtual machine for the non-compliant server. This avoids changes to the host system, isolating the non-compliant setup.

**What Happens If You Install OpenSSL 1.0.2?**

1. **System-Wide Downgrade**: All applications (including Nginx) will use the downgraded version.
2. **TLS 1.3 Incompatibility**: Your compliant server will not support TLS 1.3.
3. **Potential Security Risks**: OpenSSL 1.0.2 is deprecated and no longer receives security updates.

when i wanted to test the https://<ubuntu-ip>:443 in the kali machine on firefox i got an unable to connect

**ChatGPT said:**

ChatGPT

If you're unable to connect to your HTTPS server from the Kali machine via Firefox, it suggests an issue with either the server's configuration, network connectivity, or the browser's trust settings. Follow these steps to diagnose and resolve the issue:

**1. Verify the Nginx Service**

Ensure Nginx is running on the Ubuntu server:

bash

Copy code

sudo systemctl status nginx

* If it's not running, start it:

bash

Copy code

sudo systemctl start nginx

* If it fails to start, troubleshoot the configuration with:

bash

Copy code

sudo nginx -t

**2. Check Network Connectivity**

Confirm that the Kali machine can reach the Ubuntu server:

1. **Ping the Ubuntu Server from Kali**:

bash

Copy code

ping <ubuntu-ip>

* + If the ping fails, verify the network settings on both machines.
  + Ensure both are on the same subnet or properly routed.

1. **Check Port Accessibility**: Use telnet or nc from Kali to check if port 443 is open:

bash

Copy code

nc -zv <ubuntu-ip> 443

* + If the port is closed, verify your Nginx configuration and firewall settings.

**3. Verify the Firewall Configuration**

On the Ubuntu server:

1. **Check UFW Status**:

bash

Copy code

sudo ufw status

1. **Allow HTTPS Traffic**: If HTTPS (port 443) is not allowed, enable it:

bash

Copy code

sudo ufw allow 'Nginx Full'

sudo ufw reload

**4. Confirm the Nginx Configuration**

Check if Nginx is listening on port 443:

bash

Copy code

sudo netstat -tuln | grep 443

* If no process is listening on port 443, inspect the configuration in /etc/nginx/sites-enabled/default or your specific site configuration file.

**5. Check the Browser Settings**

Firefox may block access if the certificate is not trusted:

1. Open Firefox on the Kali machine and attempt to access the server:

arduino

Copy code

https://<ubuntu-ip>:443

1. If you see a **security warning**:
   * Click **Advanced** and then **Accept the Risk and Continue** to bypass the warning for self-signed certificates.

**6. Test with cURL**

To rule out browser-specific issues, test the HTTPS connection using curl from the Kali machine:

bash

Copy code

curl -k https://<ubuntu-ip>:443

* **Expected Output**: The HTML content of the served page.
* If it works in curl but not in Firefox, the issue lies with Firefox's handling of the certificate.

**7. Additional Debugging**

* **Nginx Logs**: Check the Nginx access and error logs for incoming connection attempts or errors:

bash

Copy code

sudo tail -f /var/log/nginx/access.log /var/log/nginx/error.log

* **Browser Logs**: Inspect Firefox's network activity using Developer Tools (F12 > Network).

openssl s\_client -connect 192.168.1.19:443 -cipher ECDHE-RSA-AES128-GCM-SHA256 im not able to establish a connection using this

**ChatGPT said:**

ChatGPT

If you're unable to establish a connection using the openssl s\_client command, there are several possible reasons. Let's troubleshoot step-by-step:

**Step 1: Check Server Status**

Ensure your Nginx server is running and listening on port 443.

1. **Verify Nginx is Active:**

bash

Copy code

sudo systemctl status nginx

If Nginx is not running, start it:

bash

Copy code

sudo systemctl start nginx

1. **Check Listening Ports:** Verify that Nginx is listening on port 443:

bash

Copy code

sudo netstat -tuln | grep 443

You should see an entry like:

plaintext

Copy code

tcp 0 0 0.0.0.0:443 0.0.0.0:\* LISTEN

If port 443 is not listed, check your Nginx configuration for the listen 443 ssl; directive.

**Step 2: Verify Connectivity**

Test basic connectivity between the client and server:

1. **Ping the Server:**

bash

Copy code

ping 192.168.1.19

If the server is unreachable, verify the network configuration.

1. **Check Firewall Rules:** Ensure the server allows incoming traffic on port 443:

bash

Copy code

sudo ufw status

If port 443 is not open, allow HTTPS traffic:

bash

Copy code

sudo ufw allow 443/tcp

sudo ufw reload

1. **Test Port Access:** From the client machine, check if port 443 is reachable:

bash

Copy code

telnet 192.168.1.19 443

or

bash

Copy code

nc -zv 192.168.1.19 443

If these fail, either the server is not listening on port 443, or a firewall is blocking the connection.

**Step 3: Verify Nginx Configuration**

Check your Nginx configuration for errors or missing SSL settings.

1. Test the configuration syntax:

bash

Copy code

sudo nginx -t

Fix any errors reported in the output.

1. Check your SSL server block in /etc/nginx/sites-enabled/default (or the relevant configuration file):

nginx

Copy code

server {

listen 443 ssl;

server\_name secure.example.com;

ssl\_certificate /etc/nginx/ssl/nginx.crt;

ssl\_certificate\_key /etc/nginx/ssl/nginx.key;

ssl\_protocols TLSv1.2 TLSv1.3;

ssl\_ciphers 'ECDHE-RSA-AES128-GCM-SHA256';

ssl\_prefer\_server\_ciphers on;

location / {

root /var/www/html;

index index.html;

}

}

Ensure the ssl\_certificate and ssl\_certificate\_key paths point to valid files.

**Step 4: Verify SSL Certificate**

Ensure the SSL certificate and private key are valid.

1. Check if the certificate and key files exist:

bash

Copy code

ls -l /etc/nginx/ssl/nginx.crt /etc/nginx/ssl/nginx.key

1. Verify the certificate and key:

bash

Copy code

openssl x509 -in /etc/nginx/ssl/nginx.crt -noout -text

openssl rsa -in /etc/nginx/ssl/nginx.key -check

Look for errors such as mismatched certificate and key.

**Step 5: Test with OpenSSL Locally**

Run the openssl s\_client command on the server itself to test connectivity:

bash

Copy code

openssl s\_client -connect localhost:443 -cipher ECDHE-RSA-AES128-GCM-SHA256

* If this works locally but not from the client machine, the issue is likely network-related.
* If it fails locally, the issue is with the Nginx configuration or SSL setup.

**Step 6: Check Logs**

Review Nginx error logs for more information about the failed connection:

bash

Copy code

sudo tail -f /var/log/nginx/error.log

Look for errors related to SSL, such as:

* no shared cipher
* ssl handshake failure

**Step 7: Use Verbose Mode for Debugging**

Enable verbose output in the openssl command to gather more information:

bash

Copy code

openssl s\_client -connect 192.168.1.19:443 -cipher ECDHE-RSA-AES128-GCM-SHA256 -debug

**Step 8: Simplify the Test**

Try removing specific ciphers from the test to verify if the issue is with cipher compatibility:

* **Test Without Cipher:**

bash

Copy code

openssl s\_client -connect 192.168.1.19:443 -tls1\_2

* **Test TLS 1.3:**

bash

Copy code

openssl s\_client -connect 192.168.1.19:443 -tls1\_3

If these succeed, the issue lies with the specific cipher.

how to get the ecdsa certifacte, key an crs on nginx ubuntu

**ChatGPT said:**

ChatGPT

To generate an ECDSA certificate, key, and CSR (Certificate Signing Request) on an Nginx server running Ubuntu, follow these steps:

**1. Install OpenSSL (if not already installed)**

Run the following command to ensure OpenSSL is installed:

bash

Copy code

sudo apt update

sudo apt install openssl -y

**2. Generate an ECDSA Private Key**

Use the openssl command to generate an ECDSA private key. For example, to use the prime256v1 curve (a commonly used curve):

bash

Copy code

openssl ecparam -genkey -name prime256v1 -out ecdsa.key

This generates a private key file named ecdsa.key.

**3. Create a Certificate Signing Request (CSR)**

Generate a CSR based on the private key. Replace yourdomain.com with your actual domain:

bash

Copy code

openssl req -new -key ecdsa.key -out ecdsa.csr

You’ll be prompted to provide details for the certificate. Example prompts:

* **Country Name (2 letter code)**: US
* **State or Province Name**: California
* **Locality Name**: San Francisco
* **Organization Name**: Your Company
* **Organizational Unit Name**: IT
* **Common Name**: yourdomain.com
* **Email Address**: admin@yourdomain.com

Leave the optional fields blank or provide necessary details.

**4. Submit the CSR to a Certificate Authority (CA)**

Send the ecdsa.csr file to a trusted Certificate Authority (CA) like Let’s Encrypt, DigiCert, or GlobalSign. They will issue a certificate (ecdsa.crt) for your domain.

Alternatively, for testing or internal use, you can self-sign the certificate:

bash

Copy code

openssl x509 -req -days 365 -in ecdsa.csr -signkey ecdsa.key -out ecdsa.crt

**5. Configure Nginx to Use the ECDSA Certificate**

Update your Nginx configuration file (e.g., /etc/nginx/sites-available/yourdomain.com):

nginx

Copy code

server {

listen 443 ssl;

server\_name yourdomain.com;

ssl\_certificate /path/to/ecdsa.crt;

ssl\_certificate\_key /path/to/ecdsa.key;

# Optional: Specify ECDSA-only settings

ssl\_protocols TLSv1.2 TLSv1.3;

ssl\_ecdh\_curve prime256v1;

}

Replace /path/to/ with the actual paths to your ecdsa.crt and ecdsa.key files.

**6. Test the Nginx Configuration**

Verify the configuration syntax:

bash

Copy code

sudo nginx -t

If there are no errors, reload Nginx:

bash

Copy code

sudo systemctl reload nginx

**7. Verify the ECDSA Certificate**

Use OpenSSL to check the certificate:

bash

Copy code

openssl x509 -in /path/to/ecdsa.crt -text -noout

You can also test the HTTPS connection with curl:

bash

Copy code

curl -v https://yourdomain.com

give me the detailed implementation for extracting and labeling the data

**ChatGPT said:**

ChatGPT

Below is a detailed implementation for **extracting and labeling data** from a PCAPNG file using Python. This implementation includes TLS feature extraction and labeling the traffic for compliance.

**Step 1: Install Required Libraries**

Ensure you have the necessary libraries installed:

bash

Copy code

pip install pyshark pandas

**Step 2: Extract Features from PCAPNG**

Here is the code for parsing the PCAPNG file and extracting TLS-specific features:

python

Copy code

import pyshark

import pandas as pd

# Load the PCAPNG file

pcap\_file = '/mnt/data/trafic.pcapng'

capture = pyshark.FileCapture(pcap\_file, display\_filter='tls') # Apply TLS display filter

# Define a function to extract relevant TLS features from each packet

def extract\_tls\_features(packet):

try:

features = {

'timestamp': packet.sniff\_time, # Timestamp of the packet

'tls\_version': getattr(packet.tls, 'handshake\_version', None), # TLS version

'cipher\_suite': getattr(packet.tls, 'handshake\_ciphersuite', None), # Cipher suite

'alert\_level': getattr(packet.tls, 'alert\_level', None), # Alert level (if present)

'packet\_length': int(packet.length), # Packet length

'session\_resumed': getattr(packet.tls, 'handshake\_session\_resumed', None), # Resumed session flag

'server\_name': getattr(packet.tls, 'handshake\_extensions\_server\_name', None), # Server Name Indication (SNI)

}

return features

except AttributeError:

return None

# Iterate over packets and extract features

data = []

for packet in capture:

features = extract\_tls\_features(packet)

if features:

data.append(features)

# Convert the extracted features into a DataFrame

df = pd.DataFrame(data)

print(df.head()) # Preview the extracted data

**Step 3: Label the Data**

The compliance labeling depends on the rules you define. For example:

* Traffic is **compliant** if it uses **TLS 1.3** and secure cipher suites.
* Traffic is **non-compliant** if it uses older versions (e.g., TLS 1.0) or insecure cipher suites.

Below is the labeling implementation:

python

Copy code

# Define compliance rules for labeling

def label\_compliance(row):

compliant\_tls\_versions = ['TLS 1.3'] # Define compliant TLS versions

secure\_cipher\_suites = ['TLS\_AES\_256\_GCM\_SHA384', 'TLS\_AES\_128\_GCM\_SHA256'] # Define secure cipher suites

if row['tls\_version'] in compliant\_tls\_versions and row['cipher\_suite'] in secure\_cipher\_suites:

return 1 # Compliant

else:

return 0 # Non-compliant

# Apply labeling function to the DataFrame

df['compliance'] = df.apply(label\_compliance, axis=1)

# Display the labeled data

print(df[['tls\_version', 'cipher\_suite', 'compliance']])

**Step 4: Save the Labeled Data**

Save the labeled data for further processing or model training:

python

Copy code

# Save the labeled data to a CSV file

output\_file = '/mnt/data/tls\_traffic\_labeled.csv'

df.to\_csv(output\_file, index=False)

print(f"Labeled data saved to {output\_file}")

**Step 5: Additional Enhancements**

* **Handle Missing Data:** Drop rows or impute missing values in features like cipher\_suite or tls\_version.
* **Add Flow Statistics:** Use tools like tshark to compute flow-level statistics (e.g., packet count, flow duration).

python

Copy code

# Example of dropping missing values

df = df.dropna(subset=['tls\_version', 'cipher\_suite'])

(I had to analyze the whole packet and dissect it in detail to be able to extract data but this helped as a starting point)