CS 253: Network Security - Project 3: TLS Interception and Man-in-the-Middle Attacks

Professor: Yuan Xiao **Term:** Fall 2025

1. Lab Overview

This hands-on project will guide you through establishing a victim website with TLS and then performing a Man-in-the-Middle (MitM) attack to intercept HTTPS traffic. You'll work in an isolated environment using virtual machines or containers. Do NOT try this on your native OS, as it may corrupt your whole network stack.

Note that this project is designed to be challenging. You are not supposed to complete everything, but you should do as much as you can before deadline. This project will be graded depending on which steps you have completed. There are in total 15 points, and the exact points given to each step are listed below.

This guide is provided to you in MD format so that you could easily copy-paste the provided code and commands. You may choose theoretically any VM with a web browser and networking to complete the project, but FYI the tested environment is VMware Workstation running Ubuntu 20.04 Desktop.

Do NOT copy code of others in order to complete the assignment. It is not worth it. Any found cheating will lead to (1) ZERO score for this project, and (2) the final grade of this course to be one grade lower.

2. Submission Requirements

This project requires you to submit a zipped pack to include the below contents (depending on how much you have completed before deadline) on Gradescope:

1. **Milestone Screenshots:**
    - Browser warning for self-signed certificate
    - Successful HTTPS connection with rogue CA
    - mitmproxy intercepting login credentials
2. **Lab Report:**
    - List the steps you have completed with corresponding screenshots
3. **Code:**
    - All configuration files
    - Custom scripts
    - Certificate files (private keys redacted)

3. Initial Setup

First, create your isolated lab environment:

```
# Create project directory
mkdir tls-mitm-lab && cd tls-mitm-lab

# Update system packages
sudo apt update && sudo apt upgrade -y

# Install required packages
sudo apt install -y apache2 openssl mitmproxy python3-pip
```

4. Part 1: Deploying Victim Website with Self-Signed Certificate

Step 1.1: Create Victim Website (1 pt)

Create the web root directory and a simple login page. All **XXX** in this document represents the last three digits of your student ID, used to identify assignments from different students.

```
sudo mkdir -p /var/www/victimXXX.local/html
sudo chown -R $USER:$USER /var/www/victimXXX.local/html
sudo chmod -R 755 /var/www/victimXXX.local
```

File: `/var/www/victimXXX.local/html/index.html`

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Secure Login - Victim Local</title>
```

```
    <title>Secure Login - Victim Local</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            max-width: 500px;
            margin: 50px auto;
            padding: 20px;
            background-color: #f5f5f5;
        }
        .login-form {
            background: white;
            padding: 30px;
            border-radius: 8px;
            box-shadow: 0 2px 10px rgba(0,0,0,0.1);
        }
        .form-group {
            margin-bottom: 20px;
        }
        label {
            display: block;
            margin-bottom: 5px;
            font-weight: bold;
        }
        input[type="text"],
        input[type="password"] {
            width: 100%;
            padding: 10px;
            border: 1px solid #ddd;
            border-radius: 4px;
            box-sizing: border-box;
        }
        button {
            background-color: #4CAF50;
            color: white;
            padding: 12px 20px;
            border: none;
            border-radius: 4px;
            cursor: pointer;
            width: 100%;
            font-size: 16px;
        }
        button:hover {
            background-color: #45a049;
        }
        .alert {
            padding: 10px;
            background-color: #f44336;
            color: white;
            border-radius: 4px;
            margin-bottom: 20px;
        }
    </style>
</head>
<body>
    <div class="login-form">
        <h2>Secure User Login</h2>
        <div class="alert">
            <strong>Warning:</strong> This is a test site for security education.
        </div>

        <form id="loginForm">
            <div class="form-group">
                <label for="username">Username:</label>
                <input type="text" id="username" name="username" required>
            </div>
```

```
            <div class="form-group">
                <label for="password">Password:</label>
                <input type="password" id="password" name="password" required>
            </div>

            <button type="submit">Sign In</button>
        </form>

        <div id="result" style="margin-top: 20px;"></div>
    </div>

    <script>
        document.getElementById('loginForm').addEventListener('submit', function(e) {
            e.preventDefault();

            const username = document.getElementById('username').value;
            const password = document.getElementById('password').value;

            // Simulate form submission
            const resultDiv = document.getElementById('result');
            resultDiv.innerHTML = `
                <div style="background-color: #d4edda; color: #155724; padding: 10px; border-radius: 4px;">
                    Login attempt recorded:<br>
                    Username: ${username}<br>
                    Password: ${password}
                </div>
            `;

            // In a real scenario, this would be sent to the server
            console.log('Login attempt:', { username, password });
            fetch("/capture", {
                method: "POST",
                headers: { "Content-Type": "application/json" },
                body: JSON.stringify({ username, password })
            });
        });
    </script>
</body>
</html>
```

File: `/var/www/victimXXX.local/html/capture.php`

```
<?php
// just accept POST silently
http_response_code(200);
?>
```

Step 1.2: Generate Self-Signed Certificate (2 pt)

Create and navigate to a certificates directory:

```
mkdir ~/certs && cd ~/certs
```

Inside it,

1. Generate a 2048-bit RSA private key named **victimXXX.local.key**.
2. Generate a self-signed certificate (**victimXXX.local.crt**). The certificate must have:
    - Country, State, City, Organization (arbitrary)
    - Common Name (CN) = victimXXX.local
3. Set appropriate permissions so that only the owner can read the key.

Hint: Commands will use *openssl genrsa, openssl req -new -x509*, etc.

Step 1.3: Configure Apache for HTTPS (2 pt)

Create **/etc/apache2/sites-available/victimXXX.local.conf**

Your VirtualHost should:

1. Listen on port 443
2. Have:

```
ServerName victimXXX.local
DocumentRoot /var/www/victimXXX.local/html
```

3. Enable TLS:

```
SSLEngine on
SSLCertificateFile <path to victimXXX.local.crt>
SSLCertificateKeyFile <path to victimXXX.local.key>
```

4. Include three security headers:

```
Header always set Strict-Transport-Security "max-age=63072000; includeSubDomains; preload"
Header always set X-Content-Type-Options nosniff
Header always set X-Frame-Options DENY
```

5. Use *Alias* to handle */capture*.
6. Log both access and error logs:

```
ErrorLog ${APACHE_LOG_DIR}/victimXXX.local_error.log
CustomLog ${APACHE_LOG_DIR}/victimXXX.local_access.log combined
```

7. HTTP → HTTPS redirect
    - Create a separate port-80 VirtualHost that permanently redirects to https://victimXXX.local .

Enable the site and required modules:

```
# Enable Apache modules
sudo a2enmod ssl
sudo a2enmod headers
sudo a2enmod rewrite

# Enable the victim site
sudo a2ensite victimXXX.local.conf

# Disable default site
sudo a2dissite 000-default.conf

# Restart Apache
sudo systemctl restart apache2
```

Step 1.4: Configure Local DNS (0.5 pt)

Add victimXXX.local to your hosts file:

```
echo "127.0.0.1 victimXXX.local" | sudo tee -a /etc/hosts
```

Step 1.5: Test Victim Website (0.5 pt)

```
# Test Apache configuration
sudo apache2ctl configtest

# Check if Apache is running
sudo systemctl status apache2

# Test HTTP to HTTPS redirect
curl -I http://victimXXX.local

# Test HTTPS (without -k flag, it will fail verification - this is expected)
curl -k https://victimXXX.local
```

Open https://victimXXX.local in your browser and **document** the security warning.

5. Part 2: Man-in-the-Middle Attack

Step 2.1: Become a Rogue Certificate Authority (1 pt)

Create a directory for your rogue CA:

```
mkdir ~/rogue-ca && cd ~/rogue-ca
```

Inside it,

1. Generate a 4096-bit CA private key **rogue-ca.key**
2. Generate a root CA certificate **rogue-ca.crt**.
    - CN and O must clearly indicate malicious intent.
    - e.g., **O=Evil Corp, CN=Evil Root CA**
3. Create necessary directories and files for CA operations

Step 2.2: Install Rogue CA as Trusted (1 pt)

Install **rogue-ca.crt** into your OS trust store.

On Linux:

```
# Copy CA certificate to system trust store
sudo cp rogue-ca.crt /usr/local/share/ca-certificates/rogue-ca.crt
sudo update-ca-certificates

# Verify installation
openssl verify -CAfile /etc/ssl/certs/ca-certificates.crt rogue-ca.crt
```

On macOS:

```
# Import CA certificate to Keychain
sudo security add-trusted-cert -d -r trustRoot -k /Library/Keychains/System.keychain rogue-ca.crt
```

On Windows:

- Copy `rogue-ca.crt` to Windows
- Run `certmgr.msc`
- Import to "Trusted Root Certification Authorities"

Step 2.3: Generate Forged Certificate for Victim (2 pt)

Create a certificate signing request (CSR) for victimXXX.local. You must:

1. Generate a private key **forged-victimXXX.local.key**.
2. Generate a CSR **forged-victimXXX.local.csr** with CN = victimXXX.local.
3. Create an extension file containing:
    - subjectAltName = DNS:victimXXX.local
4. Use your **rogue CA** (not openssl default CA) to sign the CSR.
5. Verify forged certificate using:

```
openssl verify -CAfile rogue-ca.crt forged-victimXXX.local.crt
```

Step 2.4: Configure mitmproxy (2 pt)

Create a custom mitmproxy script to use our forged certificates:

File: `~/mitm-custom.py`

```
from mitmproxy import http, ctx

class CertificateForger:
    def __init__(self):
        # TODO: Place your certificate here or pass by arguements.
        pass


    def tls_clienthello(data: tls.ClientHello):
        """
        TODO: If it's victimXXX.local, skip mitmproxy's verification of
        the upstream Apache certificate and use our forged certificate
        to communicate with the client.
        You can also use other scripts or parameters to accomplish this;
        this function is not required.
        """
        pass

    def request(self, flow: http.HTTPFlow) -> None:
        """
        This method runs whenever the client sends an HTTP(S) request.

        - Log the request URL using ctx.log.info.
        - Log the request headers. Access headers with flow.request.headers
        - If the request has a body, decode and log it.
        - These logs help you inspect intercepted traffic.
        """
        # Log all requests
        ctx.log.info(f"Intercepted request to: {flow.request.pretty_url}")
        ctx.log.info(f"Headers: {flow.request.headers}")
        if flow.request.content:
            ctx.log.info(f"Body: {flow.request.content.decode()}")

    def response(self, flow: http.HTTPFlow) -> None:
        """
        This method runs whenever mitmproxy receives a response from the server.

        Requirements:
        - If the response contains HTML content:
            * Inspect the HTML for a <body> tag.
            * Inject a visible warning banner (WARNING HTML) inside the HTML body
              indicating that the traffic was intercepted.
            * Replace the modified content in the flow response.

        This mimics how attackers can tamper with server responses.

        HINTS:
        - Response status: flow.response.status_code
        - Response body: flow.response.content
        - Convert bytes to text with .decode()
        - Search for "<body>"
        - Perform .replace() on string
        - Encode back to bytes before assigning to flow.response.content

        """

        # Log all responses
```

```
        ctx.log.info(f"Intercepted response from: {flow.request.pretty_url}")
        ctx.log.info(f"Status: {flow.response.status_code}")
        # This HTML snippet must be injected right after the <body> tag.
        WARNING_HTML = """
        <!-- INTERCEPTED BY MITM PROXY -->
        <div style='background:red;color:white;
                    padding:10px;
                    text-align:center;
                    font-weight:bold;
                    font-size:18px;'>
            SECURITY WARNING: TRAFFIC INTERCEPTED
        </div>
        """
        pass  # TODO: inject banner into HTML content to show we intercepted it

# Register the addon with mitmproxy
addons = [CertificateForger()]
```

Step 2.5: Set Up Traffic Redirection (2 pt)

Configure iptables to redirect traffic:

1. Enable IP forwarding
2. Clear existing rules

```
sudo iptables -F
sudo iptables -t nat -F
```

3. Redirect HTTP traffic to mitmproxy (port 8080)
4. Redirect HTTPS traffic to mitmproxy (port 8080)
5. (Optional) Save iptables rules (method varies by distribution)

Hint: Since our traffic originates from the local machine, the rule settings need to consider loop issues. Referring to this document may be helpful.

Step 2.6: Launch mitmproxy and Test Interception (1 pt)

Start mitmproxy with different interface. Note that the following commands are for reference only. Feel free to change the startup parameter configuration (such as –certs, ssl_keyfile, etc.), but you **must use transparent** mode.

Option 1: Command-line interface

```
mitmproxy --mode transparent --showhost --set confdir=~/rogue-ca -s ~/mitm-custom.py
```

Option 2: Web interface

```
mitmweb --mode transparent --showhost --set confdir=~/rogue-ca -s ~/mitm-custom.py
```

Option 3: Simple proxy mode

```
mitmdump --mode transparent -w traffic_dump.mitm
```

Now test the interception:

1. Open a new terminal and test the interception:

```
# Test without proxy awareness
curl -v https://victim.local
```

2. Clear site setting, cached items in firefox setting in browser. Then open Firefox and configure it to not use the proxy:

```
# Launch Firefox without using system proxy
env http_proxy= https_proxy= HTTP_PROXY= HTTPS_PROXY= \
firefox --no-remote --proxy-server="direct://" https://victim.local
```

3. Perform the login on the victim site and observe the traffic in mitmproxy.

6. Verification and Testing Script

Create verification scripts to test your setup (to help you test and debug):

File: `test_mitm.py`

```
#!/usr/bin/env python3
import requests
import urllib3
import sys

# Disable SSL warnings
urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)

def test_victim_site():
    """Test direct connection to victim site"""
    try:
        response = requests.get('https://victimXXX.local', verify=False, timeout=5)
        print(f"✓ Victim site accessible: Status {response.status_code}")
        return True
    except Exception as e:
        print(f"✗ Victim site not accessible: {e}")
        return False

def test_mitm_interception():
    """Test if traffic is being intercepted"""
    try:
        # This request should go through mitmproxy
        response = requests.get('https://victimXXX.local', verify=False, timeout=5)

        if 'INTERCEPTED' in response.text:
            print("✓ Traffic interception confirmed")
            return True
        else:
            print("✗ Traffic not intercepted - mitmproxy not working")
            return False
    except Exception as e:
        print(f"✗ Interception test failed: {e}")
        return False

if __name__ == "__main__":
    print("Testing MITM Setup...")
    print("=" * 40)

    victim_ok = test_victim_site()
    mitm_ok = test_mitm_interception()

    print("=" * 40)
    if victim_ok and mitm_ok:
        print("✓ All tests passed! MITM setup is working correctly.")
        sys.exit(0)
    else:
        print("✗ Some tests failed. Check your setup.")
        sys.exit(1)
```

Make it executable and run:

```
chmod +x test_mitm.py
python3 test_mitm.py
```

7. Cleanup Script

This script is a emergency helper provided to you in case you mess up your settings and want to start over:

File: `cleanup.sh`

```
#!/bin/bash
echo "Cleaning up MITM lab..."

# Remove iptables rules
sudo iptables -t nat -F
sudo iptables -F

# Disable Apache site
sudo a2dissite victimXXX.local.conf
sudo a2ensite 000-default.conf
sudo systemctl restart apache2

# Remove from hosts file
sudo sed -i '/victimXXX.local/d' /etc/hosts

# Remove rogue CA from trust store
sudo rm -f /usr/local/share/ca-certificates/rogue-ca.crt
sudo update-ca-certificates

echo "Cleanup complete. Remember to restart your browser."
```