Network and Systems Security II – Winter 2025 (Assignment 3)

Sambuddho Chakravarty

March 25, 2025

Implementing a Secure Reverse Proxy for an HTTPS File Server

Due date: April 15, 2025 (2359hrs. HARD DEADLINE, NO EXTENSIONS!)

Total points: 87.5

In this assignment, you will implement a reverse proxy that interfaces with an HTTPS server (Nginx or Apache) running on port 443, acting as a file-sharing service. Clients will connect to the reverse proxy over a TLS-encrypted connection using OpenSSL or LibreSSL. The reverse proxy will authenticate clients using the Pluggable Authentication Modules (PAM) library and, upon successful authentication, provide a command-line interface with a `HTTPS_SERVER>` prompt. At this prompt, clients can issue commands such as `ls` (list files), `get` (download files), `put` (upload files), and `exit` (terminate the session). The reverse proxy handles these commands by sending them to the back end server for showing the files in the web server, uploading new ones/downloading existing ones or by terminating the connection to the client as well as the web server. The reverse proxy must support multiple concurrent users, each authenticated against valid usernames on the host system, and facilitate file operations on the backend HTTPS server.

Overview of tasks:

- **Deploy an HTTPS Server**: Configure Nginx or Apache as a file-sharing server on port 443. Rely on a directory with various files in it with appropriate read and write permissions.
- **Implement a Reverse Proxy**: Create a reverse proxy that encrypts client connections with TLS and forwards requests to the HTTPS server, and relays back responses to the users. It also must do the following:
  - **Authenticate Users**: Use PAM to authenticate clients with system usernames and passwords.
  - **Command-Line Interface**: Provide a secure, interactive shell for file operations.
  - **Support Concurrency**: Handles multiple simultaneous users.

Background

Reverse proxies are critical in network security, acting as intermediaries between clients and servers to enhance security, load balancing, and access control. In this assignment, you will build a reverse proxy that secures client connections with TLS, authenticates users via PAM, and interacts with an HTTPS server to manage file sharing. This setup mimics real-world scenarios where secure access to resources is mediated by a trusted proxy, combining TLS encryption, authentication, and command-based interaction.

Assignment Tasks

This assignment consists of two main components: deploying the HTTPS server and implementing the reverse proxy with a client interface.

Deploying the HTTPS Server

You will configure an HTTPS server to serve as the backend file-sharing service.

1. Setup Environment:
    - Linux/FreeBSD VM.
    - Install Nginx or Apache. The server must use the TLS certificates through appropriate configuration changes.
2. Create a dummy CA and generate self-signed server certificates to sign the server certificate requests. The dummy CA's certificates must be available with the client(s) as the server in their respective certificate directories (`OpenSSL` has its appropriate directory).
3. Run the server with appropriate certificates, listening on localhost:443.
4. Create a file directory (e.g., `/var/www/files`) with sample files. This directory is supposed to act as the file share directory.

Implementing the Reverse Proxy and Client

You will write a reverse proxy and client program in C using OpenSSL and PAM.

1. **Reverse Proxy Setup**:
    - Implement a server that:
        - Spawns a thread/process per client for concurrency.
        - Listens on a port (e.g., 8443) with TLS using OpenSSL. The client programs connect to this port, and it should have its own (*i.e.* reverse proxy's) self-signed certificate using the same dummy CA.
        - Authenticates clients with PAM library using system usernames/passwords. The clients must have valid login user credentials on the VM hosting the reverse proxy and the HTTPS server.
        - Forwards commands to the HTTPS server via TLS.

– Provides a `HTTPS_SERVER>` prompt after authentication that accepts commands from users and interprets and acts on them connecting to the backend HTTP server using TLS. There are four commands, *viz.* `ls` (for listing files on the server), `put` (for uploading files on the server), `get` (for downloading files from the server), and `exit` to terminate the connection between the client and the reverse proxy, and between the reverse proxy and the backend server.

REMEMBER: **There are two different authentication steps involved. One where the TLS/TCP client authenticates the TLS/TCP reverse proxy server during the TLS handshake (since the latter's certificate can be authenticated) and secondly when authenticating the user using PAM.**

2. **Client Program**:
   - Connects to the reverse proxy over TLS.
   - Sends username and password (unhashed) at a login prompt.
   - Allows commands: `ls`, `get`, `put`, `exit`. Any other command must be rejected as an `unknown` command.
3. **Test basic functionality:**
   - User should be able to log in and run the various commands that he/she is allowed to. The system must allow multiple users through different connections.
4. **Security Analysis**:
   - Test/demonstrate how the system handles invalid user credentials.
   - Test/demonstrate how the system handles self-signed certificates correctly. If the server presents an invalid or unknown certificate—such as one with an incorrect signature, expired date, or unrecognized issuer—the system must reject the connection. No workaround should allow it to proceed, ensuring security by preventing unverified endpoints from being trusted.

What to Submit

1. The source codes for all the components.
2. `Makefile` which is to be used for compiling the individual components.
3. `Readme` file describing how the system work and what assumptions you have made. You must highlight atleast two corner cases you are taking care of and must demonstrate them to the TAs at the time of grading.

Grading scheme

1. The program compiles correctly – 10 points.
2. Use of appropriate PAM library routines to authenticate users with their correct explanation – 20 points if authentication works correctly and the usage/description of the library is correct. 10 points if there are some issues with authentication. **Zero** if the student cannot explain which PAM functions were used and what were the arguments used).

3. Wireshark capture demonstrating the correct client to reverse proxy TLS handshake to show if the handshake was successful. 20 points for correct handshake. 7.5 points for handshake errors.

4. Description and correct usage of the OpenSSL/LibreSSL libraries for correctly connecting to the reverse proxy and between the reverse proxy and the backend HTTP(S) server. 20 points for correct description. 7.5 points if neither the program runs correctly nor the description/usage is correct. **Zero** points for correct working but lack of correct understanding of the functions and arguments used.

5. Correctly handling at least 4 corner cases (2 for TLS handshake and connection establishment and 2 for user authentication using PAM) that must be described in the `Readme` – 10 points.

6. `Readme` describing the functionality of the system with the assumptions you made, the test cases and arguments to use to run and test the code, the users on the reverse proxy host and their login credentials that are required to test, *etc.* – 7.5 points for providing all the necessary details in the `Readme`. 3.5 points only partial details presented.