A Transparent SSL-Proxy Server

CS745: Principles of Data and System Security
Course Project Report

Arijeet De (23M0742)
A Asish (23M0759)
Tirthesh Jain (23M0758)
Chiluveru Pranav (23M0787)

Department of Computer Science and Engineering
Indian Institute of Technology, Bombay
Mumbai 400 076

Spring 2024

Contents

Chapter 1

Introduction

SSL or TLS is an application layer protocol for communication security. It allows two communicating end-points to protect their communication from a third party. These two communicating parties set up an encrypted tunnel between them. The encryption/decryption of this communication tunnel is done using a symmetric key, which is negotiated using the SSL/TLS protocol. SSL/TLS protocol requires the exchange of digital certificates to ensure the authenticity of communicating parties to each other. Usually, a PKI is involved in distributing digital certificates. PKI can be skipped if the two parties know each other's digital certificates by direct introduction.

Chapter 2

Experimental Setup

Experimental setup has been done as described in the problem statement. We have two servers : vm-server-A and vm-server-B. We also have a client called vm-browser-1. There is also vm-proxy that acts as an access point for vm-server-A, i.e, vm-server-A can be only accessed from vm-proxy. We have used nginx to host our application, so in vm-proxy we have modified the nginx.conf file in the path /etc/nginx to enable forwarding all accesses to vm-proxy to vm-server-A.

Chapter 3

Methodology

1 Task 1: XSS

XSS (Cross-Site Scripting) is a prevalent web attack where malicious scripts are injected into web pages viewed by other users. Exploiting vulnerabilities in web applications, attackers can execute unauthorized actions on behalf of users, such as stealing sensitive data, hijacking sessions, or spreading malware. XSS poses serious risks, including data breaches and financial losses. Preventive measures like input validation and content security policies are essential to mitigate these threats and safeguard web applications and users.

1.1 Setup:

The simulation involved three VMs:

- vm_server_A: Hosted the webpage where XSS attack is possible.
- vm_browser: Accessed the webpage hosted on vm_server_A to initiate the attack.
- vm_proxy: Acted as a proxy server to intercept and manipulate the traffic between the browser and server A, facilitating packet capturing.

1.2 Attack Scenario:

In the simulated scenario, the following steps were executed:

- The user accessed the webpage hosted on vm_server_A using the browser (vm_browser).

- The webpage hosted on vm_server_A contained a vulnerability that allowed injection of malicious scripts (view *task*1.*py*).
- Upon loading the webpage, the malicious script executed within the user's browser, leading to the XSS attack.
- All traffic between the browser and server A was routed through vm_proxy, enabling packet capturing for analysis.

1.3 Mitigation:

To mitigate XSS attacks, input sanitization techniques is implemented on the server-side (*task*1*M.py* implements it). This involves filtering and validating user inputs to remove or neutralize potentially harmful characters or scripts.

2 Task 2: CSRF

Cross-Site Request Forgery (CSRF) is a type of attack that exploits the trust between a user's browser and a web application. In a CSRF attack, an attacker tricks a user into unknowingly executing unwanted actions on a web application where the user is authenticated.

The attacker can perform various malicious actions such as unauthorized data modification or deletion, initiating financial transactions, accessing to sensitive information, hijacking account and many more.

2.1 Setup:

The simulation involved four VMs:

- vm_server_A: Hosted the legitimate web application.
- vm_server_B: Hosted the malicious webpage containing the CSRF payload.
- vm_proxy: Acted as a proxy server to intercept and manipulate the traffic between the browser and servers A and B.
- vm_browser: Used to access both the legitimate and malicious webpages.

2.2 Attack Scenario:

In the simulated scenario, the following steps were executed:

- The user accessed the malicious webpage hosted on vm_server_B using the browser (vm_browser).
- Upon loading the malicious webpage, an automatic request was triggered from the browser to vm_server_A, containing a vulnerable endpoint.
- All traffic between the browser and servers A and B was routed through vm_proxy, allowing packet capturing of requests and responses

2.3 Mitigation:

To mitigate CSRF attacks, we have implemented CSRF tokens. By adding unique tokens to each user session to validate the authenticity of requests we can easily avoid CSRF attacks.

3 Task 4

Create self-signed digital certificates on vm-server-A and vm-server-B. Generated self-signed certificate and private key using OpenSSL

```
openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout server.key
-out server.crt
```

When prompted, fill out the required information for the certificate like country, organization, email address etc.

Move the generated certificate and private key to a secure location on your server.

```
sudo mkdir -p /etc/nginx/ssl/
sudo cp server.crt server.key /etc/nginx/ssl/
```

Open your Nginx configuration file (/etc/nginx/nginx.conf) and add the SSL configuration to Nginx server.

```
ssl_certificate /etc/nginx/ssl/server.crt;
ssl_certificate_key /etc/nginx/ssl/server.key;
```

Used Wireshark to capture traffic and saved as traffic-task-4.pcap

Verified that traffic intercepted on vm-proxy is no more plaintext and is encrypted.

Chapter 4

Research Project

Additionally We have also done a research project.
GitHub Link: Click Here

Implemented 2 Research Papers and 1 Article

- Design of Hybrid Cryptography System
- An Extended Version of the Polybius Cipher
- SHA-256 Cryptographic Hash Algorithm

The implementation is shown using a simple Chat Application

- For end-to-end encryption hybrid cryptography and extended polybius cipher are used.
- For end-to-end integrity check SHA-256 is used.
- We also showed demo of Man-In-The-Middle attack.