The details of proxy implementation

Masoomeh Rudafshani

1 Implementation of Proxy

This report first explains the architecture of proxy used in LeakSpot. Then, it describes how the proxy handles SSL-related errors.

1.1 Proxy Architecture

To modify the JavaScript code of a page, the proxy intercepts all the messages between the client and server. To make it possible for the proxy to work for all different web pages, it handles both HTTP and HTTPS connections:

- HTTP connections: In the case of an HTTP connection, the proxy simply forwards the request from the client to the server, modifies the replies from the server, and forwards the modified replies to the client.
- HTTPS connections: The HTTPS proxy just relays the data without knowing anything about its content. In this case, to be able to intercept data, the MITM proxy sits in the connection between the client and the server. It replies to the initial request by the client, so the client assumes it is the server, while acting as the client for the server. As shown in Figure 1.1, the HTTPS proxy forwards data between the client and MITMP proxy without knowing anything about the content of the data by creating a secure channel (pipe). It cannot view or manipulate the data since the data is encrypted. Please note that such a secure channel is not needed for HTTP connections.

1.2 Dealing with SSL-Related Errors

Whenever there is a man-in-the-middle attack, the SSL protocol detects and lets the browser know that there is a problem and, so, it gives a warning. It is possible to develop the proxy in a way so that the browser is not notified, as discussed by Marlinspike [**?**]; however, going into these details is beyond the scope of this work. To avoid the warnings generated because of man-in-the-middle attacks, the different warning messages are handled as follows.

- **Certificate authority is not trusted by the browser**: The certificate used by the MITM proxy, when acting as a server, is self-signed, so it is not trusted by the client, i.e., browser. The browser allows the communication to proceed by giving the user a notification. To avoid this warning, the certificate authority needs to be known to the browser.
- " **This is not probably the site you are looking for**": This warning is generated when the domain name in the common name field of the certificate differs from the domain requested by the client. This error is especially problematic when the client needs to connect to a page containing several domains. When connecting to a single site through the browser, it is possible to manually accept the SSL certificate and move forward; however, in cases where it is necessary to connect to different domains to fully load a page, using a single-domain certificate results in the page not loading fully. To

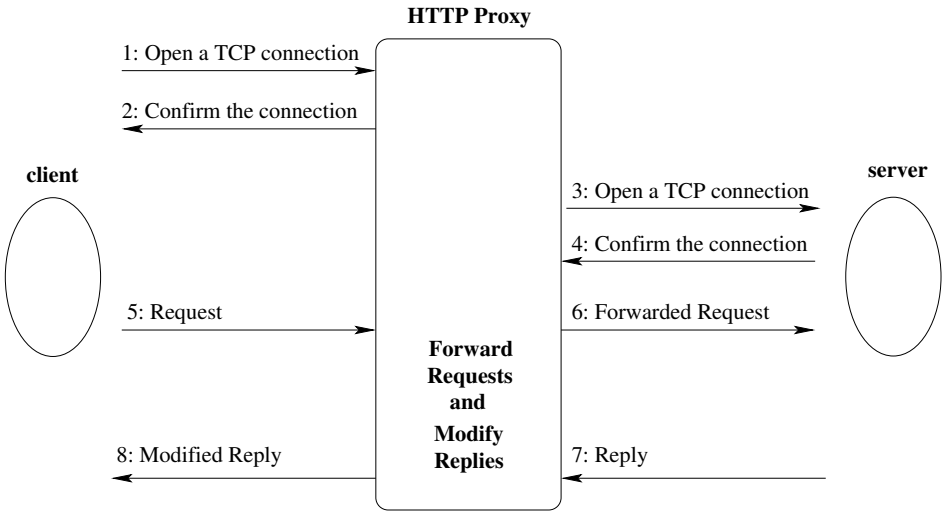

Figure 1.1: Handling HTTP connections by the proxy.

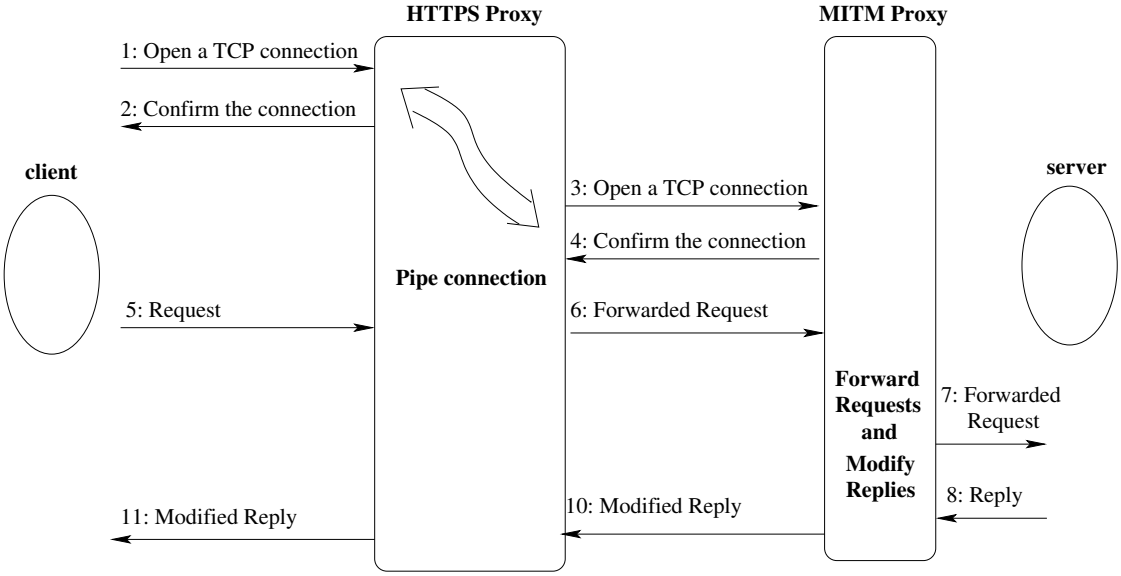

Figure 1.2: Handling HTTPS connections by the proxy

avoid this problem, multi-domain certificats are used. Creating multi-domain certificates for a page requires knowledge of the different domains in the page. This data can be extracted by accessing the page through the proxy, and recording all the different domains. This process is done just once for every page. Implementing the proxy so that the certificates are generated on the fly for every domain could be a subject of future work.

1.3 Commands used to deal with SSL-Related Errors

First, the rootCA certificate is created and added to the list of trusted certificate authorities by the browser. Then, a request for a key is created using a configuration file, i.e., multi-domain.conf in the following, which specifies several domains. Then the generated certificate is signed using the rootCA certificate authority. The following commands are used to create the certificate authority and the multi-domain certificates:

- Create a private key

  ```
  openssl genrsa -out rootCA.key 2048
  ```

- Self-sign the key:

  ```
  openssl req -x509 -new -nodes -key rootCA.key -days 1024 -out \
  rootCA.pem
  ```

  This will create an SSL certificate called rootCA.pem, signed by itself, valid for 1024 days.

- Install `rootCA` on your browser.

The certificate that the fake server is using is created using the following steps:

- Create CSR for the server

  ```
  openssl genrsa -out device.key 2048
  ```

- Sign CSR with root CA key

  ```
  openssl req -new -key device.key -out device.csr
  ```

To enter several keys, I created the configuration file and entered several keys.

- Create CSR for device

  ```
  openssl req -new -out san_domain_com.csr -key \
  san_domain_com.key -config multidomain.conf
  ```

- Sign CSR with root CA key

  ```
  openssl x509 -req -in san_domain_com.csr -CA rootCA.pem \
  -CAkey rootCA.key -CAcreateserial -out san_domain_com.crt \
  -extensions v3_req -days 500 -extfile multidomain.conf
  ```