Telecom Nancy

Université de Lorraine

RSA

Proxy HTTP

Auteurs :
Vincent Albert

6 avril 2016

Table des matières

1 Présentation du sujet

1.1 L'objectif

L'objectif du projet est de réaliser un proxy transparent relayant les requêtes du navigateur au serveur désiré. Le proxy doit être capable de gérer des requêtes HTTP et HTTPS en IPv4 et en IPv6.

1.2 Les outils utilisés

Le projet a été codé en langage C sous systèmes de type Unix. Afin de faciliter la gestion des versions et le partage des fichiers, le gestionnaire de version git a été utilisé. Le projet se trouve sur le dépôt Github suivant : `https://github.com/Neressea/RSA`

2 Analyse du fonctionnement d'un proxy

2.1 Analyse d'échanges TCP et HTTP sans proxy

Afin de ne pas crouler sous les données et de pouvoir analyser agréablement l'échange, nous avons utilisé le filtre : "(ip.src==192.168.1.76 or ip.dst==192.168.1.76) and (ip.src==193.54.21.201 or ip.dst==193.54.21.201) and tcp" afin de ne récupérer que les données TCP échangées entre le client et le serveur (adresse de telecomnancy.eu). HTTP est une surcouche à TCP, les paquets sont donc aussi récupérés.

La requête engendre tout d'abord un three-way handshake (SYN, SYN-ACK, ACK) afin d'initialiser la connexion TCP. Après ces 3 paquets TCP, le premier paquet HTTP est envoyé par le navigateur.

Ce paquet, en plus des données d'un paquet TCP, contient aussi l'en-tête HTTP de la requête. On trouve notamment les informations concernant le type de la requête (GET), sur le navigateur, sur les cookies et enfin sur la page demandée.

Ensuite, de multiples échanges TCP se font à nouveau. Il s'agit de la réponse à la requête en fragments, comme on peut le déduire depuis l'annotation "TCP segment of a reassembled PDU". Ceci nous indique que le paquet contient des données d'un protocole étant une surcouche à TCP (HTTP dans notre cas). Il y a aussi les paquets des ACK envoyés par le navigateur.

Ensuite tous ces segments sont recomposés en un seul paquet HTTP. Ce paquet indique en effet qu'il résulte de la recomposition de 8 autres segments TCP. Ce segment contient à nouveau les données TCP et l'en-tête de la réponse HTTP qui nous indique que la requête a bien été exécutée (200 OK) mais aussi la taille des données de la réponse. Car en effet, le paquet contient en plus les données HTML correspondant à la réponse de la requête.

On remarque que malgré le fait que la requête n'ait été exécutée qu'une seule fois, plusieurs GET sont effectués. Ceci est dû à la réponse HTML, qui importe des fichiers depuis d'autres emplacements du serveur (css, javascript, ...). Le navigateur va donc effectuer des requêtes pour les récupérer afin de pouvoir afficher la page normalement.

2.2 Analyse d'échanges TCP et HTTP avec proxy

Afin d'analyser les échanges avec un proxy, nous avons modifié les paramètres de notre navigateur pour nous connecter au proxy gratuit d'adresse 162.223.88.243. Nous avons aussi modifié le filtre wireshark, étant donné que nous ne contactons plus directement le serveur de destination : "(ip.src==192.168.1.76 or ip.src==162.223.88.243) and (ip.dest==192.168.1.76 or ip.dst==162.223.88.243) and tcp"

La connexion TCP est à nouveau débutée par un three-way handshake, mais cette fois-ci entre le client et le proxy. Ensuite, le navigateur envoie la requête HTTP. N.B. : On remarque la mauvaise qualité de la connexion avec les duplications de paquets.

Ensuite, l'échange de paquets TCP contenant des fragments de la réponse HTTP a de nouveau lieu entre le client et le proxy. Le paquet HTTP est reconstitué avec son en-tête et ses données HTML.

L'échange entre le client le proxy est donc semblable à l'échange entre le client et le proxy. Le serveur proxy reçoit donc la requête vers le serveur, l'exécute, reçoit les données et les envoie de la même manière qu'il les a reçues.

2.3 Algorithme général d'un proxy

```
Data: serverSocket, clientSocket, destSocket, rcvBuff, sendBuff, servAddr
Result: Réception d'une requête d'un client, envoi au serveur et transfert de la réponse au client
while true do
    Attente de la connexion TCP d'un client;
    while données à lire sur clientSocket do
        rcvBuff += lire(socketClient) ; //On récupère la requête du client
    end
    servAddr = gethostbyname(rcvBuff.hostname) ; //On récupère l'adresse du serveur
    destSocket = socket(servAddr) ; //On crée la socket sur le serveur de destination
    send(rcvBuff, destSocket) ; //On envoie la requête au serveur
    accept(destSocket) ; //On attend la réponse du serveur
    while données à lire sur destSocket do
        if taille des données == 0 then
            //On ferme la connexion
            close(serverSocket);
            close(clientSocket) ;
        else
            sendBuff += lire(destSocket) ; // On lit un morceau de la réponse send(sendBuff,
            clientSocket) ; //On transfère la donnée
        end
    end
end
```

Algorithm 1: Algorithme d'un proxy

3 Développement d'un proxy

3.1 Version standard du proxy

3.2 Version multi-utilisateur

3.3 Version HTTPS

Références et autres sources d'inspiration

`http://blog.hikoweb.net/index.php?post/2011/11/06/Exemple-de-rapport-en-LaTeX`
Source pour le template LaTeXayant servi à la rédaction de ce rapport.

`http://livre.g6.asso.fr/`
Site expliquant en détail le fonctionnement de l'IPv6 et l'utilisation de getaddrinfo