HAL
open science

Étude du filtrage de flux HTTPS

Antoine Goichot

To cite this version:

Antoine Goichot. Étude du filtrage de flux HTTPS. Réseaux et télécommunications [cs.NI]. 2014. hal-01097781

HAL Id: hal-01097781

https://inria.hal.science/hal-01097781v1

Submitted on 22 Dec 2014

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Rapport de stage

Étude du filtrage de flux HTTPS

Antoine Goichot

Année 2013–2014

Stage de 2e année réalisé chez Inria

Maître de stage : Isabelle Chrisment

Encadrant universitaire : Van-Phuc Do

Déclaration sur l'honneur de non-plagiat

Je soussigné(e),

Nom, prénom : Goichot, Antoine

Élève-ingénieur(e) régulièrement inscrit(e) en 2e année à TELECOM Nancy

Numéro de carte de l'étudiant(e) : 31215052

Année universitaire : 2013–2014

Auteur(e) du document, mémoire, rapport ou code informatique intitulé :

Étude du filtrage de flux HTTPS

Par la présente, je déclare m'être informé(e) sur les différentes formes de plagiat existantes et sur les techniques et normes de citation et référence.

Je déclare en outre que le travail rendu est un travail original, issu de ma réflexion personnelle, et qu'il a été rédigé entièrement par mes soins. J'affirme n'avoir ni contrefait, ni falsifié, ni copié tout ou partie de l'œuvre d'autrui, en particulier texte ou code informatique, dans le but de me l'accaparer.

Je certifie donc que toutes formulations, idées, recherches, raisonnements, analyses, programmes, schémas ou autre créations, figurant dans le document et empruntés à un tiers, sont clairement signalés comme tels, selon les usages en vigueur.

Je suis conscient(e) que le fait de ne pas citer une source ou de ne pas la citer clairement et complètement est constitutif de plagiat, que le plagiat est considéré comme une faute grave au sein de l'Université, et qu'en cas de manquement aux règles en la matière, j'encourrais des poursuites non seulement devant la commission de discipline de l'établissement mais également devant les tribunaux de la République Française.

Fait à Villers-lès-Nancy, le 27 août 2014

Signature :

Rapport de stage

Étude du filtrage de flux HTTPS

Antoine Goichot

Année 2013–2014

Stage de 2e année réalisé chez Inria

Antoine Goichot
4 impasse du Vauloin
21370 LANTENAY
+33 (0)6 10 13 53 50
antoine.goichot@telecomnancy.eu

TELECOM Nancy
193 avenue Paul Muller,
CS 90172, VILLERS-LÈS-NANCY
+33 (0)3 83 68 26 00
contact@telecomnancy.eu

Inria Nancy - Grand Est
615 rue du Jardin Botanique
CS20101, 54603 VILLERS-LÈS-NANCY
+33 (0)3 83 59 30 00

Maître de stage : Isabelle Chrisment

Encadrant universitaire : Van-Phuc Do

Remerciements

Je tiens à adresser mes remerciements à Mme Isabelle CHRISMENT pour m'avoir permis de réaliser mon stage au sein de son équipe mais également pour son temps accueil.

Je remercie également M. Thibault CHOLEZ pour son aide.

Je remercie également M. Wazen SHBAIR pour ses découvertes qui ont fourni la base de mon travail.

Je remercie aussi tous les membres de l'équipe Madynes pour m'avoir accueilli.

Pour finir, j'adresse mes remerciements à tous les orateurs des différentes conférences auxquelles j'ai pu assister pendant mon séjour chez Inria.

Résumé

Ce rapport a pour but de présenter le travail que j'ai réalisé chez Inria, dans l'équipe MADYNES, pendant mes neuf semaines de stage. Ce dernier s'inscrit à la suite de ma deuxième année à TELECOM Nancy et a porté sur l'étude du filtrage de flux HTTPS [1]. Le protocole HTTPS est de plus en plus utilisé pour accéder, de manière sécurisée, à du contenu sur le web. Il permet de s'assurer que les données n'ont pas été altérées, ni même lues par un tiers, entre l'envoi du message et sa réception par un serveur.

Les récentes découvertes d'un des membres de l'équipe MADYNES ont montré que les systèmes de filtrage actuels ont des limites et pouvaient être facilement contournées, notamment dans le cas du filtrage sur le protocole HTTPS mis en place dans des pare-feux.

Pour commencer, j'ai étudié les différentes techniques de filtrage sur le web, et notamment dans le cas du web chiffré. J'ai analysé le fonctionnement du protocole HTTPS, et en particulier le protocole TLS (Transport Layer Security) qui fournit la couche de sécurité.

J'ai ensuite implanté une extension pour le navigateur Firefox permettant à un utilisateur d'accéder à des services malgré la présence de filtres.

Puis, j'ai développé un collecteur de flux HTTPS qui devrait fournir une base pour créer un nouveau filtrage, reposant sur des techniques automatiques.

Mots-clés : HTTPS, Filtrage, Contournement

Abstract

The purpose of this report is to present my internship done at Inria, in the MADYNES research team, during nine weeks. This internship is the continuation of my second year at TELECOM Nancy, and was about the study of HTTPS filtering. HTTPS is more and more used to securely get content from the web. Thereby, a user can be sure that data has not be changed during the communication between him and the server.

Recent discoveries form one member of the team MADYNES have shown that current filtering systems are limited and could be bypassed easily, particularly systems which HTTPS traffic filtering.

First, I studied filtering techniques and particularly HTTPS filtering. I analyzed how HTTPS works and especially I focused on the TLS (Transport Layer Security) protocol.

Then, I implemented a Firefox add-on in allowing a user to easily access services despite filtering rules.

Finally, I designed and set up a network architecture to collect HTTPS trafic. This system should help to create a new filtering software based on machine learning.

Keywords : HTTPS, Filtering, Bypassing

1. HyperText Transfer Protocol Secure

Table des matières

Résumé **iv**

Abstract **iv**

Table des matières **v**

Introduction **1**

1 Présentation de la structure d'accueil **2**

1.1 Présentation d'Inria 2

1.2 Le laboratoire de recherche : le LORIA 3

1.3 L'équipe MADYNES 3

2 Contexte du stage **4**

2.1 Objectifs 4

2.2 Les techniques de filtrage sur Internet 6

2.3 Le protocole HTTPS 8

2.4 Un message particulier : le ClientHello 10

2.5 Le filtrage sur le protocole HTTPS 10

3 Développement d'une extension pour le navigateur Firefox **13**

3.1 Contournement de filtres sur HTTPS 13

3.2 Contournement de filtres au niveau du serveur DNS 17

3.3 Tests 19

4 Mise en place d'une architecture de collecte de trafic web chiffré **22**

4.1 Architecture réseau 22

4.2 Mise en œuvre 23

4.3 Résultats 25

Conclusion **27**

Bibliographie / Webographie **28**

Liste des illustrations **29**

Annexes **31**

A Organigramme d'Inria **31**

B Organigramme du Loria **32**

C Scripts employés pour l'architecture de collecte de flux chiffrés **33**

D Scenarii de contournement **35**

D.1 Facebook bloqué par un filtrage DNS, contourné en utilisant le fichier hosts interne à l'extension (IP de Facebook : 173.252.110.27) 35

D.2 Facebook bloqué par un filtrage DNS, contourné en contactant le serveur DNS public de Google 36

D.3 Facebook bloqué par un filtrage SNI, contourné en utilisant un SNI factice . 37

D.4 Facebook bloqué par un filtrage DNS et SNI, contourné en contactant le serveur DNS public de Google et en utilisant un SNI factice 38

D.5 YouTube bloqué par un filtrage SNI, contourné en utilisant la technique de l'encapsulation (décrite en 3.1.1), Handshake avec `maps.google.com` 38

Introduction

Depuis ces quelques dernières années, l'usage d'Internet s'est intensifié et diversifié. Il n'est plus rare d'utiliser le web pour gérer ses comptes, faire des achats en ligne ou même déclarer ses impôts. Il est évident que ces données, qui transitent entre l'utilisateur et un serveur, sont sensibles et ne doivent pas être lues par une tierce personne. C'est là qu'intervient le protocole HTTPS.

Le protocole HTTPS (HyperText Transfer Protocol Secure) est la combinaison du protocole HTTP (utilisé pour naviguer sur le web) et de la couche de chiffrement TLS (Transport Layer Security). Les buts de cette dernière sont principalement l'authentification du serveur, la confidentialité des données échangées et l'intégrité de ces mêmes données.

D'autre part, pour certaines raisons, les flux HTTPS peuvent être filtrés. Certes, ces blocages peuvent aller à l'encontre de la vie privée, nous pouvons citer les événements du mois de mars en Turquie [2] où Twitter a été bloqué pendant deux semaines. Mais ces filtrages peuvent être totalement légitimes, comme par exemple un contrôle parental ou alors des restrictions d'accès, dans le cadre d'usage professionnel, aux ressources informatiques d'une entreprise.

Ce stage est lié également à des recherches en cours sur les techniques de contournement de filtrage HTTPS, par l'un des membres de MADYNES, équipe de recherche d'Inria qui m'a accueilli pendant neuf semaines.

Après une présentation de la structure d'accueil, nous nous intéresserons au contexte de ce stage, notamment aux techniques de filtrage existantes, en particulier à l'ère du web chiffré, ainsi qu'à leurs limites. Par la suite, nous présenterons l'outil développé au cours du stage, permettant ainsi de prouver ces limites et ainsi de démontrer que les systèmes actuels sont facilement contournables. Enfin, nous décrirons la mise en place d'une architecture de collecte de trafic chiffré qui devrait fournir une base de travail pour l'élaboration d'un moyen de filtrage plus robuste.

2. http://www.huffingtonpost.fr/2014/03/21/twitter-turquie-bloque-erdogan_n_5005075.html

1 Présentation de la structure d'accueil

Cette partie décrit l'organisme de recherche qu'est Inria, et le laboratoire d'accueil, le LORIA. Enfin sera présentée l'équipe dans laquelle j'ai effectué mon stage, l'équipe MADYNES.

1.1 Présentation d'Inria

Inria, créé en 1967, est un établissement public de recherche à caractère scientifique et technologique (EPST) sous l'autorité des ministères en charge de la Recherche et de l'Industrie. L'institut a pour missions de produire une recherche dans les sciences du numérique et d'assurer l'effet de cette recherche.

Au 31 décembre 2013 [1], Inria rassemblait 4 471 personnes dont 3 449 scientifiques. Ces derniers regroupés en 172 « équipes-projets » (dont 139 en collaboration avec des universités, et d'autres établissements de recherche), ont rédigé près de 4 500 publications scientifiques l'année dernière. Ensemble, ils inventent les technologies numériques de demain, en partenariat avec les acteurs de la recherche (publique et privée), en France et à l'étranger. Mais c'est également 304 brevets actifs, 120 logiciels déposés à l'Agence pour la Protection des Programmes et aussi plus de 110 start-ups. Tout cela avec un budget total de 233 M d'euros (dont 37 % de ressources propres).

Géographiquement parlant, Inria est composé d'un siège social à Rocquencourt (près de Paris) et de 8 centres de recherche répartis sur tout le territoire français : Bordeaux - Sud-Ouest, Grenoble - Rhône-Alpes, Lille - Nord Europe, Nancy - Grand Est, Paris - Rocquencourt, Rennes - Bretagne Atlantique, Saclay - Île-de-France, et Sophia Antipolis - Méditerranée .

L'organigramme d'Inria est joint en annexe A, Il est à noter que le président-directeur général est Antoine Petit, depuis le 25 août 2014.

C'est dans le centre de Nancy - Grand Est que j'ai réalisé mon stage, dans l'équipe MADYNES, équipe projet Inria commune avec le LORIA.

1. Source : `http://www.inria.fr/institut/inria-en-bref/chiffres-cles`

1.2 Le laboratoire de recherche : le LORIA

Le LORIA, Laboratoire Lorrain de Recherche en Informatique et ses Applications est une Unité Mixte de Recherche (UMR 7503), commune à trois établissements : le CNRS (Centre National de la Recherche Scientifique), l'Université de Lorraine et Inria.

Depuis sa création, en 1997, le LORIA a pour mission la recherche fondamentale et appliquée en sciences informatiques, mais le transfert technologique se place également au cœur des priorités du laboratoire au travers de collaborations industrielles à l'échelle nationale ou internationale.

Les recherches scientifiques sont menés au sein de 28 équipes (dont 15 sont communes avec Inria) structurées en cinq départements : « Algorithmique, calcul, image et géométrie », « Méthodes formelles », « Réseaux, systèmes et services », « Traitement automatique des langues et des connaissances », et, « Systèmes complexes et intelligence artificielle ».

Le laboratoire abrite près de 500 personnes et est structuré suivant l'organigramme fourni en annexe B.

C'est dans l'équipe MADYNES, du département « Réseaux, systèmes et services », que j'ai réalisé ce stage.

1.3 L'équipe MADYNES

L'« équipe-projet » MADYNES (« MAnagement of DYnamic NEtworks and Services ») vise la conception, la validation et la mise en œuvre de nouveaux paradigmes et architectures de supervision et de contrôle capables de maîtriser la « dynamicité » croissante des infrastructures et services de télécommunications, et de résister au facteur d'échelle induit par l'Internet ubiquitaire.

Leurs recherches s'organisent principalement autour de deux axes :
– L'axe **Gestion Autonome** : il porte sur l'évolution des paradigmes de supervision afin de définir les fondements et l'infrastructure de base de la gestion autonome (élaboration de méthodes d'auto-organisation des entités de gestion, évaluation et mise en œuvre d'architectures de communication distribuées exploitant le modèle pair-à-pair et le routage applicatif, recherche de nouvelles approches pour la représentation de l'information . . .).
– L'axe **Aires Fonctionnelles** : il contribue à étendre ces fondements au travers de : la sécurité, la configuration et la provision de services, et également, la mesure et l'analyse.

L'équipe coopère avec de nombreux laboratoires de recherche publics et industriels à travers le monde, en particulier avec les laboratoires universitaires suivants : Télécom ParisTech (Paris), l'Institut National des Sciences Appliquées (INSA) de Lyon, l'Université du Québec à Montréal (UQAM) (Montréal, Canada) ou encore Macquarie University (Sydney, Australie) et avec des partenaires industriels tels que : Alcatel, Orange Labs ou Thalès.

Ces coopérations sont renforcées aussi par la participation à de nombreux programmes nationaux et internationaux ainsi qu'à des groupes de recherche et de standardisation.

Maintenant que l'ensemble de la structure d'accueil a été décrit, nous allons présenter le contexte dans lequel le stage s'est déroulé.

2 Contexte du stage

La problématique de ce stage est d'inspecter le trafic des utilisateurs pour les protéger ainsi que protéger l'infrastructure réseau tout en leur laissant la possibilité d'utiliser des flux chiffrés. Or ces deux volontés semblent être incompatibles, car un flux chiffré signifie que seul l'émetteur et le destinataire du message peuvent le lire.

Le filtrage sur Internet n'est pas quelque chose de nouveau, il peut avoir plusieurs buts et volontés différents. C'est un outil de sécurité nécessaire, qui peut aller du simple contrôle parental à son domicile (pour protéger ses enfants contre du contenu inapproprié) à la censure d'Internet par un gouvernement, en passant par exemple par des restrictions d'accès à certains sites, dans le cadre d'un usage professionnel des ressources informatiques.

Un certain nombre de techniques existent déjà pour contourner ces filtrages. Cependant, des recherches sur de nouveaux moyens de contourner un filtrage sur le flux HTTPS ont été réalisées par Wazen SHBAIR, doctorant dans l'équipe MADYNES, (cf [1] et [2]) dont les travaux portent sur le protocole HTTPS. Ses travaux permettent de montrer les limites des filtrages existants et proposent de nouveaux moyens de filtrage. M. SHBAIR envisage de mettre en œuvre une méthode de filtrage basée sur l'apprentissage automatique (*machine learning*) reposant sur un ensemble de capture de flux de trafic HTTPS.

Dans cette partie, nous allons présenter les objectifs du stage et l'organisation du travail pour les remplir. Puis nous décrirons les différentes méthodes de filtrage, et expliquerons pourquoi l'utilisation du protocole HTTPS rend obsolète un certain nombre de ces méthodes. Enfin nous nous intéresserons aux nouvelles techniques de filtrage, liées à HTTPS.

2.1 Objectifs

Les objectifs de ce stage étaient au nombre de deux : développer un outil pour contourner le filtrage de flux HTTPS et mettre en place un collecteur de traces.

Nous allons donc retrouver dans cette partie les objectifs de ces deux travaux ainsi que la répartition du temps de travail entre ceux-ci.

Un outil pour contourner le filtrage

Le but était, à partir des travaux de Wazen SHBAIR, de développer un outil, simple à installer et à utiliser, capable de mettre en évidence les faiblesses des filtres utilisés aujourd'hui sur Internet, notamment dans le cas du protocole HTTPS.

Pour cela, j'ai implanté une extension pour le navigateur web Firefox. En effet, Mozilla laisse la possibilité d'ajouter des options à son navigateur relativement facilement. De plus, la fondation Mozilla fournit une large documentation et un certain nombre d'interfaces pour certaines spécificités. Un autre avantage est le fait d'être compatible multi plate-formes ; en principe une extension pour ce navigateur est installable sur tous les systèmes où Firefox est déployable. Et du côté utilisateur, l'installation d'une extension ne requiert aucune connaissance, juste quelques clics de souris.

Un collecteur de trafic HTTPS

Le deuxième objectif était de permettre à Wazen SHBAIR de créer un filtrage plus intelligent et plus efficace, utilisant des techniques d'apprentissage automatique (*machine learning*).

Pour avancer ses travaux, il avait besoin de capturer en grand nombre des flux HTTPS. Pour cela, un collecteur de trafic HTTPS a été mis en place. Ce dernier sauvegarde de manière journalière tout le trafic HTTPS des utilisateurs (volontaires) dans un fichier.

Répartition du travail

Pendant une semaine, j'ai dû me familiariser avec le protocole HTTPS, les différentes techniques de filtrage et avec le travaux de Wazen SHBAIR.

Puis, pendant les cinq semaines suivantes, j'ai développé l'extension pour le navigateur de Firefox.

Les trois jours d'après ont été consacrés à la mise en place du collecteur et au développement des scripts utilisés pour son fonctionnement.

Après cela, j'ai effectué des modifications sur l'extension pendant deux semaines et demie.

Enfin, La dernière semaine de ce stage m'a permis d'effectuer les ajustements finaux sur le collecteur et les finitions de l'extension.

2.2 Les techniques de filtrage sur Internet

Nous avons déjà évoqué la nécessité de filtrer le trafic pour protéger les utilisateurs. Bien souvent, cet outil de sécurité est mis en place par l'intermédiaire d'une pare-feu (*firewall* en anglais), un logiciel et/ou un matériel, permettant de faire respecter une politique de sécurité par rapport au réseau. Celle-ci définit quelles sont les communications autorisées sur ce réseau.

Dans cette section, nous allons présenter les différentes techniques de filtrages intervenant sur Internet. Le premier type de blocage que nous allons décrire se situe au niveau du serveur DNS. Les suivants sont réalisés par un équipement, type pare-feu, situé entre le client et le serveur, et concernent : le filtrage sur IP et nom de domaine et le filtrage sur le contenu des messages échangés.

2.2.1 Filtrage au niveau du serveur DNS

Le « Domain Name System » (ou DNS) est un service permettant de traduire un nom de domaine en une adresse IP. La résolution du nom de domaine se fait de manière itérative en parcourant la hiérarchie du DNS.

Quand un utilisateur souhaite accéder à une ressource par son nom de domaine, le fichier « hosts » est consulté. Puis, si l'adresse IP correspondante n'est pas présente, le système effectue une requête DNS.

Le filtrage DNS est assez simple à comprendre et à mettre en place : supposons que nous voulons interdire à l'utilisateur l'accès au site `www.twitter.com` par filtrage DNS.

Si la structure est vraiment réduite, il est possible d'ajouter une ligne dans le fichier « hosts » de chaque poste, sinon il faut mettre en place un DNS local. Et, dans les deux cas, il faut faire correspondre à `www.twitter.com` l'adresse IP désirée (par exemple, 192.168.1.254, qui correspondrait à une page signalant à l'utilisateur le blocage).

Cet exemple est illustré avec la figure 2.1 :

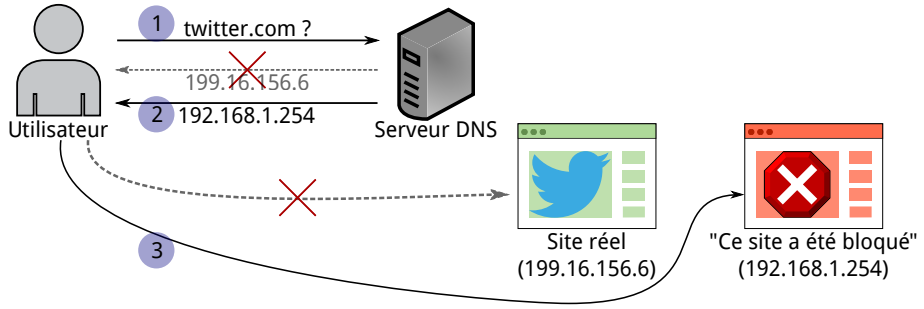


FIGURE 2.1 – Filtrage au niveau du serveur DNS

1. L'utilisateur cherche à accéder à Twitter, pour cela il demande l'adresse IP de ce site au serveur DNS.
2. Au lieu de lui répondre par l'adresse du serveur demandé, le serveur DNS lui transmet l'adresse 192.168.1.254 (adresse IP privée).
3. L'utilisateur se connecte à l'adresse que le serveur DNS lui a fournie, et au lieu de voir apparaître une page du réseau social, il voit une page lui signalant que ce site a été bloqué.

Remarque : à partir de maintenant et ce jusqu'à la fin de la section 2.2, le dispositif de filtrage qui sera évoqué, dans chaque sous-section, se situera entre l'utilisateur et le serveur, typiquement le filtrage est réalisé par un pare-feu.

2.2.2 Filtrage sur IP et sur les requêtes HTTP GET

Il est possible de filtrer des sites par leur IP et/ou leur nom de domaine (sites placés sur « liste noire »). Quand l'utilisateur va faire une requête pour une page, le système va vérifier si le nom de domaine (ou son adresse IP) est interdit ou non. Si ce site est interdit, l'usager va recevoir une page d'erreur avec éventuellement la raison du blocage. De nombreux services permettent de récupérer des listes contenant plusieurs millions d'entrées.

Il est possible également d'interdire tout le trafic, sauf une liste de sites explicitement autorisés (sites placés sur « liste blanche »).

Dans le protocole HTTP, le nom de domaine du site visité est disponible dans le champ « Host » d'une requête GET (méthode la plus courante pour demander une ressource). Par exemple, si nous faisons un filtre sur le nom de domaine de YouTube, le paquet dont est extrait la figure 2.2 n'est pas transmis à son destinataire.

```
83 89 47 45 54 20 2f 20  48 54 54 50 2f 31 2e 31   ..GET /  HTTP/1.1
0d 0a 48 6f 73 74 3a 20  77 77 77 2e 79 6f 75 74   ..Host:  www.yout
75 62 65 2e 63 6f 6d 0d  0a 55 73 65 72 2d 41 67   ube.com. .User-Ag
65 6e 74 3a 20 4d 6f 7a  69 6c 6c 61 2f 35 2e 30   ent: Moz illa/5.0
20 28 58 31 31 3b 20 55  62 75 6e 74 75 3b 20 4c    (X11; U buntu; L
69 6e 75 78 20 78 38 36  5f 36 34 3b 20 72 76 3a   inux x86 _64; rv:
33 31 2e 30 29 20 47 65  63 6b 6f 2f 32 30 31 30   31.0) Ge cko/2010
30 31 30 31 20 46 69 72  65 66 6f 78 2f 33 31 2e   0101 Fir efox/31.
```

FIGURE 2.2 – Paquet comportant le nom de domaine YouTube

Cependant le fitrage sur IP peut être problématique notamment quand plusieurs serveurs virtuels (installés sur la même machine physique) se partagent la même IP, puisque nous allons bloquer tous les serveurs et non pas seulement celui qui va à l'encontre de la politique du firewall.

2.2.3 Filtrage sur le contenu

Il est également possible d'inspecter le contenu des paquets (« Deep Packet Inspection »). Dans ce cas là, le dispositif de filtrage étudie chaque paquet avant de l'envoyer à son destinataire. Cette opération est coûteuse, mais permet de bloquer une page si elle contient un ou des mots jugés inappropriés.

Par exemple, si nous faisons un filtre sur le mot « informatique », le paquet dont est extrait la figure 2.3 n'est pas transmis à son destinataire.

```
6e 22 20 43 4f 4e 54 45  4e 54 3d 22 54 6f 75 74   n" CONTE NT="Tout
65 20 6c 27 61 63 74 75  61 6c 69 74 e9 20 69 6e   e l'actu alit. in
66 6f 72 6d 61 74 69 71  75 65 20 65 74 20 68 69   formatiq ue et hi
67 68 20 74 65 63 68 20  3a 20 65 6d 70 6c 6f 69   gh tech  : emploi
2c 20 74 65 73 74 20 64  65 20 70 72 6f 64 75 69   , test d e produi
74 73 20 68 69 67 68 2d  74 65 63 68 20 65 74 20   ts high- tech et
6a 65 75 78 20 76 69 64  e9 6f 73 2c 20 61 73 74   jeux vid .os, ast
75 63 65 73 20 65 74 20  6c 6f 67 69 63 69 65 6c   uces et  logiciel
73 20 e0 20 74 e9 6c e9  63 68 61 72 67 65 72 2e   s . t.l. charger.
```

FIGURE 2.3 – Paquet comportant le mot « informatique »

Nous avons désormais fait le point sur les différentes techniques utilisées pour filtrer le trafic sur Internet. Nous allons maintenant nous pencher sur le protocole HTTPS, qui permet d'établir des communications chiffrées entre un client et un serveur.

2.3 Le protocole HTTPS

Le protocole HTTPS (HyperText Transfer Protocol Secure) est la combinaison de HTTP (protocole de communication client-serveur développé pour le web) avec une couche de chiffrement SSL ou TLS.

TLS (Transport Layer Security), et son prédécesseur SSL (Secure Sockets Layer), sont des protocoles de sécurisation des échanges sur Internet. TLS dans sa version 1 correspond à la version 3.1 de SSL, le changement de nom correspond à la date du rachat du brevet. La version actuelle de TLS est la version 1.2 (décrite par la RFC 5246[3] d'août 2008).

TLS fournit les objectifs de sécurité suivants :
- l'authentification du serveur,
- la confidentialité des données échangées,
- l'intégrité des données échangées,
- de manière optionnelle, l'authentification du client,
- la transparence : les protocoles de la couche applicative ne sont pas modifiés pour utiliser une connexion sécurisée par TLS.

Le protocole HTTPS permet donc au visiteur de vérifier l'identité du site web auquel il accède (grâce à un certificat d'authentification). Il garantit la confidentialité et l'intégrité des données envoyées par l'utilisateur et reçues par le serveur.

Pour établir une communication privée avec le serveur, le protocole TLS réalise un « Handshake » entre le client et le serveur. La figure 2.4 présente le handshake « classique » de TLS. Il existe également une version plus courte, en cas de reprise d'une session débutée auparavant et une autre plus longue, dans le cas où le client doit s'authentifier auprès du serveur.

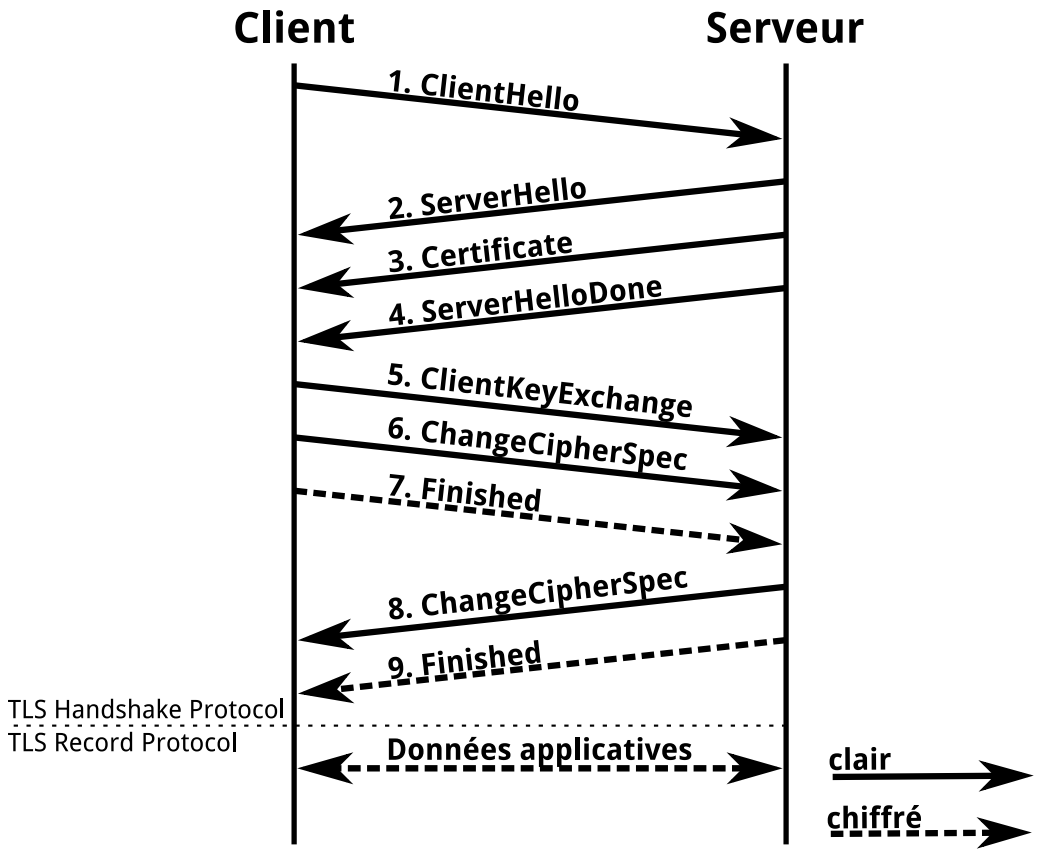


FIGURE 2.4 – Le Handshake TLS

Les étapes du Handshake TLS « classique » sont les suivantes :

1. Le client envoie un « ClientHello » au serveur. Ce message contient une valeur aléatoire (générée par le client) et la suite des algorithmes (Cipher Suite) supportés (par ordre de préférence décroissant).
2. Le serveur répond au client par un « ServerHello ». Ce message contient une valeur aléatoire (générée par le serveur).
3. Le serveur envoie son certificat au client pour authentification.
4. Le serveur envoie un « ServerHelloDone » pour indiquer la fin de ses émissions.
5. Le client envoie sa clé de session chiffrée avec la clé publique du serveur.
6. Le client envoie un « ChangeCipherSpec » et passe en mode chiffré.
7. Le client envoie un « Finished » pour indiquer la fin de ses émissions. À noter que ce message est chiffré.
8. Le serveur envoie également un message « ChangeCipherSpec » et passe à son tour en mode chiffré.
9. Le serveur envoie un « Finished » (premier message chiffré par le serveur).

Le tunnel TLS est établi, c'est maintenant le « TLS Record Protocol » qui prend le relais pour chiffrer les données. Tous les échanges futurs entre le client et le serveur seront donc chiffrés.

Après la descriptions des étapes de la mise en place du tunnel TLS, nous allons détailler le premier message du Handshake TLS : le ClientHello.

2.4 Un message particulier : le ClientHello

Le premier message envoyé lors d'un Handshake TLS est le « ClientHello » qui contient une valeur aléatoire (générée par le client) et la suite des algorithmes (Cipher Suite) supportés (par ordre de préférence décroissant). Ces caractéristiques sont obligatoires, mais le client peut ajouter dans son message, dans un champ d'extensions, la liste des fonctionnalités supplémentaires qu'il demande au serveur.

L'extension qui nous intéresse particulièrement ici, est celle nommée « Server Name Indication », abrégée en SNI dans la suite. Elle a été introduite en juin 2003 dans la RFC 3546[4] (la dernière actualisation date de janvier 2006, RFC 6066[5]).

Dans son mécanisme classique, TLS ne permet pas au client de dire au serveur le nom de domaine qu'il cherche à joindre. Ceci est particulièrement problématique quand plusieurs serveurs virtuels sont hébergés sur la même machine physique, qu'ils se partagent la même adresse IP (celle de la machine physique) mais n'ont pas le même certificat à fournir au client. Et avec la montée du Cloud, cette configuration est très courante, pour des raisons évidentes de limitations de coûts.

L'extension SNI règle ce problème, puisque le client peut ajouter le nom du domaine qu'il veut contacter dans le Handshake TLS. Ainsi, un serveur peut utiliser une seule adresse IP et héberger plusieurs serveurs qui ne se partagent pas forcément le même certificat.

D'une manière générale, les extensions TLS sont conçues pour être rétrocompatibles, c'est-à-dire qu'un client supportant une extension peut communiquer avec un serveur ne la supportant pas et vice versa. Cependant, certains serveurs n'acceptent pas les communications de clients non compatibles avec le SNI.

Le champ SNI est notamment utilisé dans le filtrage appliqué aux flux chiffrés, comme montré ci-après.

2.5 Le filtrage sur le protocole HTTPS

Dans cette partie, nous allons faire le point sur les différentes techniques adaptées aux flux chiffrés. Pour commencer, nous regarderons si les techniques fonctionnelles sur HTTP le sont toujours sur HTTPS, puis nous nous intéresserons à deux autres techniques spécifiques au flux chiffrés : le filtrage sur le certificat envoyé par le serveur et le filtrage sur le SNI.

2.5.1 Application des techniques de filtrage HTTP sur HTTPS

Dans la partie concernant les techniques de filtrage sur Internet, nous avons vu des filtrages :
- au niveau du serveur DNS,
- sur l'adresse IP du serveur de destination,
- sur le nom de domaine du serveur,
- sur le contenu des paquets.

Nous allons passer en revue ces quatre filtrages pour voir s'ils peuvent s'appliquer au protocole HTTPS.

Étant donné que la résolution DNS se passe avant le moindre échange entre le client et le serveur, donc en particulier avant le Handshake TLS permettant d'établir le tunnel TLS entre le client et le serveur, il est donc possible de mettre en œuvre ce filtrage dans le cas du protocole HTTPS.

Bien que le contenu des messages échangés entre un client et un serveur soit chiffré, les adresses IP source et IP de destination sont en clair, et elle doivent l'être, sinon il est impossible d'acheminer le trafic d'un point à un autre. Comme ces adresses IP ne sont pas chiffrées, un dispositif peut donc les utiliser pour effectuer un filtrage.

Dans le cadre du protocole HTTP, il était possible de filtrer les communications entre le client et le serveur, en utilisant le nom de domaine du serveur. Ce nom de domaine se trouve dans le champ « Host » d'une requête GET. Avec le protocole HTTPS, cette requête est effectuée après que le Handshake TLS ait eu lieu, elle est donc chiffrée. De ce fait, il est impossible pour un dispositif de filtrage de lire le contenu de cette requête.

Cependant, il est tout de même possible d'appliquer ce type de filtrage, si le dispositif fonctionne comme un « Man-In-The-Middle », décrit ci-dessous :

1. le client veut initier la connexion avec le serveur
2. le système intercepte et accepte la connexion puis il initie la connexion avec le serveur (Handshake TLS)
3. le client réalise un Hanshake TLS avec ce système (et reçoit donc le certificat du dispositif à la place de celui du serveur)
4. le système déchiffre le contenu chiffré du client et l'envoie (de manière chiffrée) au serveur et vice versa.

Ainsi, le système a accès en clair aux données du client, en particulier le champ « Host » d'une requête GET et peut donc bloquer le trafic si le nom de domaine fait partie d'une liste noire.

Dans la section sur le protocole HTTP, nous avions évoqué un autre type de filtrage, basé sur le contenu des paquets. Pour les mêmes raisons que le filtrage précédent, il n'est pas possible de repérer un quelconque mot interdit sans réaliser un « Man-In-The-Middle ».

Pour résumer, les filtrages au niveau du serveur DNS et sur l'adresse IP du serveur sont applicables pour le protocole HTTPS. Par contre, les filtres sur le nom de domaine (au niveau d'une requête GET) et sur le contenu nécessitent des procédés plus lourds à mettre en place (et également l'acceptation du certificat du pare-feu par le client).

2.5.2 Filtrage sur le certificat du serveur

Nous avons vu dans la partie sur le protocole HTTPS (Section 2.3) que lors du Hanshake TLS, le serveur envoyait son certificat au client. Ce message est en clair, le firewall peut donc l'intercepter et vérifier que le client ne tente pas de se connecter sur un site interdit par la politique du firewall.

Cependant, il y a un risque de bloquer plus de contenu que nécessaire. En effet, de nombreux sites hébergeant plusieurs services se partageant le même certificat. C'est par exemple le cas de Google : le certificat reçu en allant sur YouTube est le même qu'en allant sur le moteur de recherche de Google. Et donc en filtrant YouTube par le certificat du serveur, nous risquons de filtrer tous les services Google par la même occasion.

Pour cette dernière raison, le filtrage sur le SNI est de plus en plus appliqué.

2.5.3 Filtrage sur le SNI

Pendant un certain nombre d'années, le SNI n'était pas utilisé pour la moindre application puisqu'un trop grand nombre de clients ne supportaient pas cette extension (en particulier, les personnes utilisant Internet Explorer sous Windows XP). Désormais, la quasi-totalité des clients, en particulier les navigateurs, supportent cette extension.

Du côté des serveurs, nous retrouvons également la compatibilité avec cette extension pour la plupart des nouvelles versions de leurs systèmes d'exploitation (Apache, Cherokee, Nginx, Tomcat ...).

De plus, le SNI peut être extrait simplement de la liste des extensions à l'intérieur du ClientHello et il indique le serveur contacté pendant l'établissement de la connexion entre le client et celui-ci. De ce fait, le SNI est un paramètre intéressant pour identifier le serveur que le client cherche à joindre.

Ainsi, certains dispositifs mettent en place ce filtrage, simple et rapide, qui a l'avantage d'être parfaitement transparent pour l'utilisateur.

Dans cette partie nous avons vu tous les éléments constituant le contexte du stage, à savoir : les objectifs, les différentes techniques de filtrage existantes, en particulier celles adaptées au flux chiffrés. Dans la partie qui va suivre, nous allons présenter le premier travail réalisé, à savoir, le développement de l'outil de contournement de blocages afin de démontrer la fragilité des systèmes de filtrages.

3 Développement d'une extension pour le navigateur Firefox

Nous allons présenter l'extension développée pour le navigateur Firefox. Cet outil permet de montrer les faiblesses des filtrages réalisés à l'aide du SNI et de ceux intervenant au niveau du serveur DNS. Nous commencerons par nous intéresser au contournent de filtres sur HTTPS, puis le contournent de filtres effectués au niveau du serveur DNS. Enfin, nous donnerons un récapitulatif des tests effectués.

3.1 Contournement de filtres sur HTTPS

Nous allons monter les limites d'un filtrage réalisé sur le SNI (champ du ClientHello, indiquant le serveur à contacter). Tout d'abord, nous présenterons le principe du contournement, puis nous décrirons son implantation au sein de l'extension. Enfin nous donnerons les possibilités de réglages du module, laissées aux utilisateurs.

3.1.1 Principe

Le principe de contournement a été découvert par Wazen SHBAIR. Il a trouvé trois stratégies de contournement (décrites dans [1] et [2]) :

– La première peut paraître triviale (cf. Figure 3.1) : utiliser un navigateur qui n'est pas compatible avec l'extension SNI. Ainsi, le dispositif de filtrage basé sur le SNI ne peut pas bloquer nos communications. De plus, les extensions TLS sont conçues pour être rétrocompatibles (un client supportant une extension peut communiquer avec un serveur ne la supportant pas et vice versa). Cependant, certains serveurs n'accepteraient pas les communications de clients non compatibles avec le SNI, cette technique n'a pas été mise en œuvre.

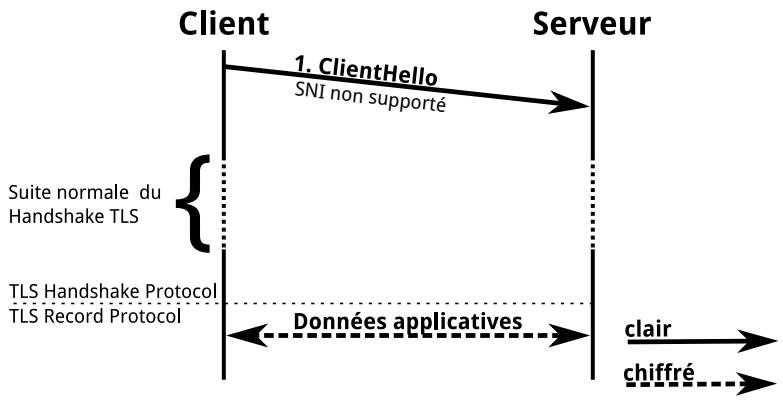


FIGURE 3.1 – Première technique de contournement de filtrages sur le SNI

– La seconde technique repose sur une recommandation de la RFC 6066. Nous pouvons y lire (voir [5], p. 5) que si le serveur comprend l'extension SNI du ClientHello, mais qu'il ne reconnait pas le nom de domaine, il faut alors qu'il entreprenne une de ces deux actions : soit abandonner le Handshake, mais ce n'est pas recommandé, soit le continuer. Donc le client peut envoyer un SNI factice et si le serveur applique la recommandation de cette RFC (ce qui est très souvent le cas), il continue de communiquer avec ce client.

Une illustration de cette technique se trouve en figure 3.2.

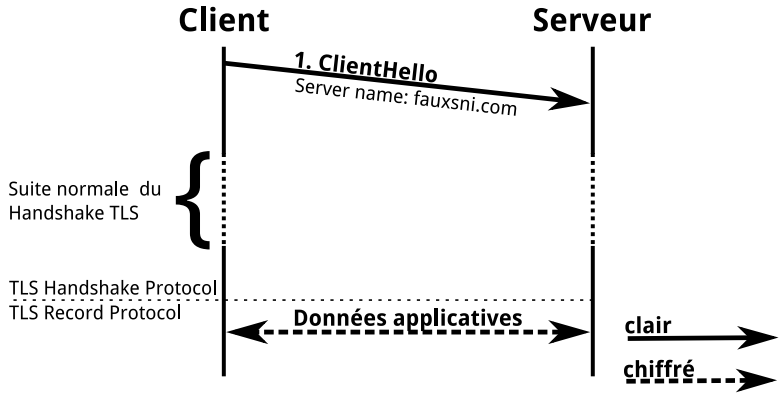


FIGURE 3.2 – Seconde technique de contournement de filtrages sur le SNI

– La troisième et dernière stratégie est d'encapsuler le contenu du site bloqué en réalisant le Handshake TLS avec un autre site.

Si le service YouTube est bloqué (par filtrage sur le SNI) mais Gmail ne l'est pas. Le Handshake TLS est effectué avec Gmail (le SNI n'est pas modifié et donc le nom du serveur est `mail.google.com`).

Une fois le Handshake effectué, nous effectuons une requête GET et nous y faisons figurer la ligne `Host:www.youtube.com:443`. Ce message étant chiffré, il est impossible de savoir ce que nous avons envoyé, donc en particulier que nous cherchons à recevoir une page YouTube. Et nous obtenons bien la page YouTube demandée.

Nous pouvons retrouver un schéma correspondant à cet exemple en figure 3.3.

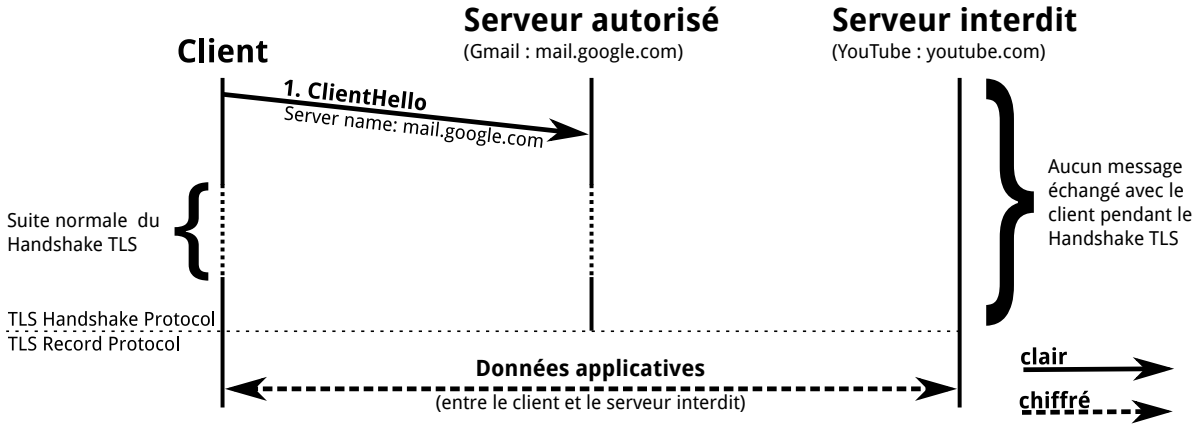


FIGURE 3.3 – Troisième technique de contournement de filtrages sur le SNI

3.1.2 Implantation

L'implantation (en JavaScript) repose sur une idée trouvée dans une autre extension, nommée Convergence [1] (disponible sous licence GPL v3). Cette idée consiste à intercepter la demande de l'utilisateur et d'envoyer depuis l'extension une requête similaire, avec les changements nécessaires.

L'extension que j'ai développée fonctionne comme un Man-In-The-Middle mais **localement**, l'extension interceptant le trafic HTTPS.

Le fonctionnement détaillé est le suivant :

1. L'utilisateur veut se connecter à un site en HTTPS.
2. L'extension récupère le site à contacter puis réalise le Handshake TLS avec celui-ci, en appliquant une des deux techniques décrites en 3.1.1.

   D'un point de vue plus technique, nous créons un socket (TCP), nous récupérons l'adresse IP de l'hôte à partir de son nom de domaine (avec une fonction de l'interface de programmation de Netscape Portable Runtime (NSPR)) et nous effectuons le Handshake TLS avec le serveur (en précisant le SNI).
3. L'extension envoie au client un message pour lui signaler que le Handshake avec le serveur a été fait.
4. Le client réalise un Handshake TLS avec l'extension (le certificat reçu par l'utilisateur est donc le certificat de l'extension). Pour cela, il a fallu créer un socket au préalable.
5. L'extension transmet les données du client au serveur et vice versa. (Cela revient à écouter sur un socket et à écrire sur un autre.)

Nous retrouverons un schéma correspondant à ces étapes en figure 3.4.

Le système de filtrage est donc désormais entre l'extension et le serveur. Il voit passer un SNI ne faisant pas partie de sa liste noire, il laisse donc passer le message.

1. `http://convergence.io/` : Le développeur Moxie Marlinspike, expert en sécurité, part du principe que les autorités de certifications SSL ne sont plus dignes de confiance (en 2011, des certificats avaient été piratés), et propose, à l'aide de l'extension Convergence, de ne plus vérifier les certificats de la feuille à la racine comme c'est actuellement le cas mais avec un maillage de plusieurs « experts », pour valider les certificats.

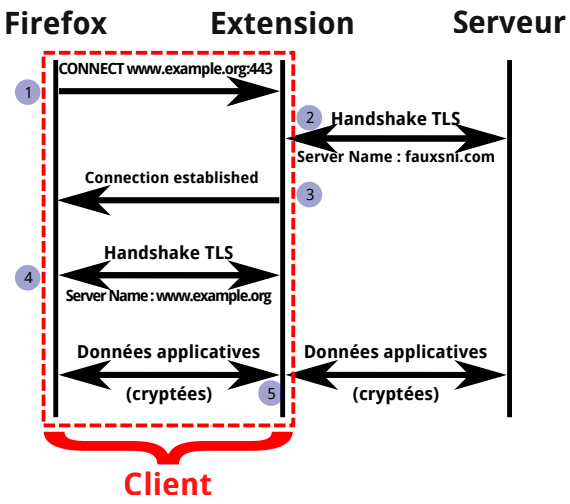


FIGURE 3.4 – Extension : Fonctionnement du Man-In-The-Middle

Ce schéma est à adapter dans le cas de l'utilisation de la technique de l'« encapsulation ». Dans ce cas, l'étape 2 se fait entre l'extension et le serveur autorisé (Gmail) et les données applicatives (étape 5) sont échangées entre l'extension et le serveur bloqué (YouTube).

3.1.3 Interface

Pour des raisons de vie privée, le Man-In-The-Middle se met en fonctionnement uniquement si le site que l'utilisateur cherche à contacter fait partie d'une liste de sites autorisés par celui-ci. Il peut également gérer cette liste : ajouter un site, en enlever un ou tous les enlever. Après cela, il peut sauvegarder ses modifications. Nous retrouverons en figure 3.5 une capture de la fenêtre permettant de gérer cette liste.

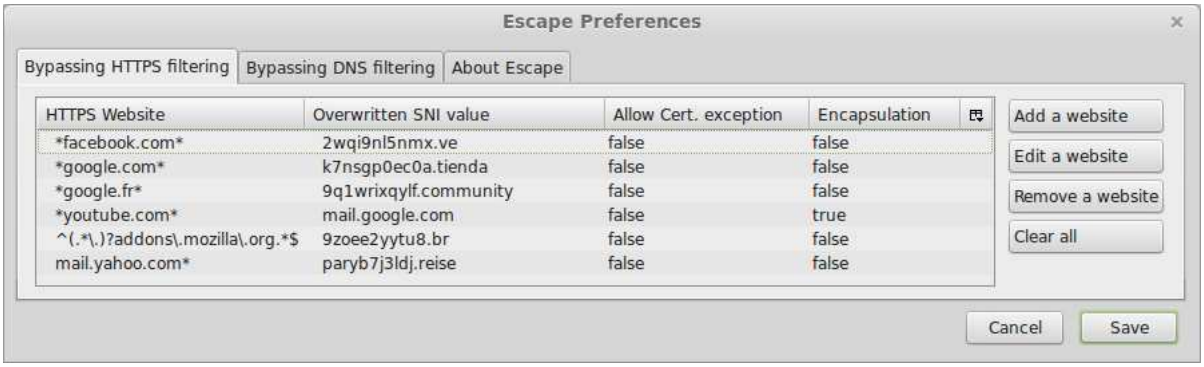


FIGURE 3.5 – Extension : onglet « Bypassing HTTPS filtering »

Quand l'utilisateur ajoute un site, il doit préciser le nom de celui ci, soit son adresse complète, soit sous forme d'une expression régulière, ou encore utilisant le symbole * qui correspond à « tout » (par exemple : `*.google.fr` correspond à tous les sous-domaines de `google.fr`).

Il peut préciser un SNI particulier, sinon une chaîne de caractères aléatoires sera générée.

Il peut également spécifier si l'extension doit faire de l'encapsulation (expliquée en section 3.1.1).

Enfin, il peut aussi préciser s'il est prêt à accepter un certificat invalide de la part du serveur (mauvais nom de domaine, certificat périmé . . .).

Dans cette partie nous avons donc vu les différentes techniques de contournement de filtres liés au SNI, leur implantation ainsi que les réglages laissés aux utilisateurs pour retrouver un accès aux sites filtrés.

Nous allons désormais nous intéresser au contournement de filtres réalisés au niveau du serveur DNS.

3.2 Contournement de filtres au niveau du serveur DNS

Dans cette section, nous allons montrer les limites d'un filtrage réalisé sur les requêtes DNS. Tout d'abord, nous allons présenter le principe de contournement, puis nous décrirons son implantation au sein de l'extension. Enfin nous donnerons les possibilités de réglages du module, laissées aux utilisateurs.

3.2.1 Principe

Nous considèrons qu'un blocage DNS a été mis en place (décrit en 2.2.1). Pour contourner un tel filtrage, l'utilisateur peut modifier son fichier **hosts**, mais cela nécessite de connaître l'adresse IP du site bloqué, ou alors il peut changer ses paramètres réseaux pour utiliser un autre serveur DNS (par exemple le serveur DNS public de Google : `8.8.8.8`). Dans les deux cas, il faut souvent être administrateur de son système pour faire ces changements.

L'idée a donc été de mettre en place ces possibilités dans les options de l'extension (pour des sites utilisant le protocole HTTPS).

Ainsi l'utilisateur peut :
- soit utiliser la configuration de son système (et le filtrage DNS ne sera pas contourné à moins que l'utilisateur change lui même sa configuration),
- soit utiliser un autre serveur DNS,
- soit utiliser un fichier **hosts** interne à l'extension (l'utilisateur doit donc connaître l'adresse IP du serveur à contacter).

3.2.2 Implantation

Pour mettre en œuvre les possibilités évoquées précédemment, j'ai utilisé quelques fonctions JavaScript, trouvées sur GitHub (auteur : `https://github.com/JoshData`), permettant de réaliser une requête DNS et d'extraire la ou les adresses IP de la réponse du serveur.

Dans tous les cas, la résolution DNS se passe entre les étapes 1 et 2 sur le schéma en figure 3.4 (entre l'interception par l'extension de la page demandée par l'utilisateur et le handshake TLS avec le serveur).

– Si l'utilisateur utilise la configuration de son système, alors la résolution DNS n'est pas modifiée par l'extension.
–S'il choisit un autre serveur DNS, alors l'extension envoie une requête DNS à ce serveur, attend sa réponse puis initie le Handshake TLS avec l'IP reçue.
D'un point de vue technique, au lieu d'utiliser l'interface de programmation de NSPR pour obtenir l'adresse IP du serveur, un socket UDP est créé (avec comme adresse IP, celle du serveur DNS et comme port, le numéro 53) et une requête DNS est envoyée. Enfin, on initie le Handshake TLS avec l'adresse IP reçue dans la réponse DNS.

– S'il utilise le fichier **hosts** interne et qu'il va sur un site qui est ajouté à cette partie, alors aucune requête DNS n'est effectuée et le Handshake TLS est initié avec l'adresse IP correspondante.

3.2.3 Interface

Dans l'onglet « Bypassing DNS filtering », l'utilisateur peut choisir sa configuration DNS. Il retrouve les possibilités évoquées plus haut, à savoir : soit utiliser la configuration du système, soit changer de serveur DNS, soit contacter les sites web directement via leur IP (par l'intermédiaire d'un fichier host interne à l'extension).

Par défaut, un certain nombre de serveurs DNS et de correspondances site/adresse IP sont pré-configurés, mais l'utilisateur peut modifier comme il le souhaite ces paramètres.

Encore une fois, il peut sauvegarder ou non ses modifications en fermant la fenêtre d'options.

Nous retrouverons un aperçu de ces possibilités dans la figure 3.6.

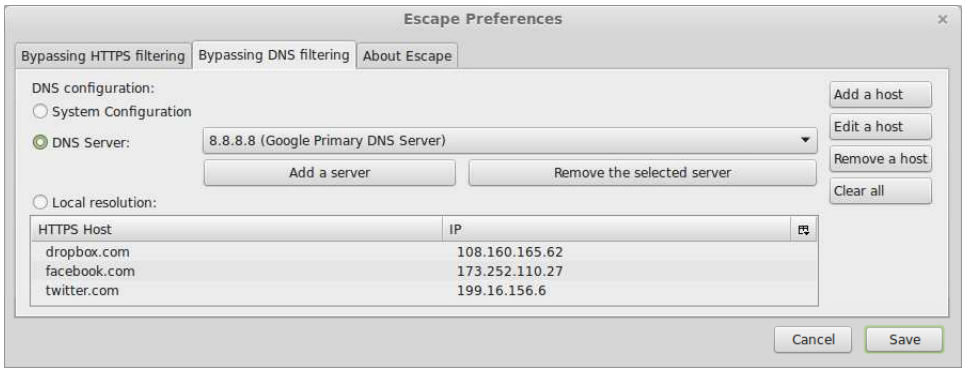


FIGURE 3.6 – Extension : onglet « Bypassing DNS filtering »

Dans ces deux sections, nous avons démontré les faiblesses de filtrages réalisés sur le SNI ou sur les requêtes DNS. Nous avons vu également que dans les deux cas, le contournement est basé sur des méthodes relativement simples à mettre en œuvre.

3.3 Tests

Dans cette partie, nous allons décrire les différents tests réalisés. Pour cela, nous commencerons par présenter la méthode employée, puis les trois pare-feux que nous avons utilisés pour réaliser un filtrage, à savoir : Sphirewall, Untangle NG Firewall et IPFire. Enfin, nous terminerons par les résultats obtenus.

3.3.1 Méthode

Le filtrage du protocole HTTPS s'est fait au moyen de l'installation d'un pare-feu logiciel sur une machine dédiée. L'architecture réseau employée a été la suivante :

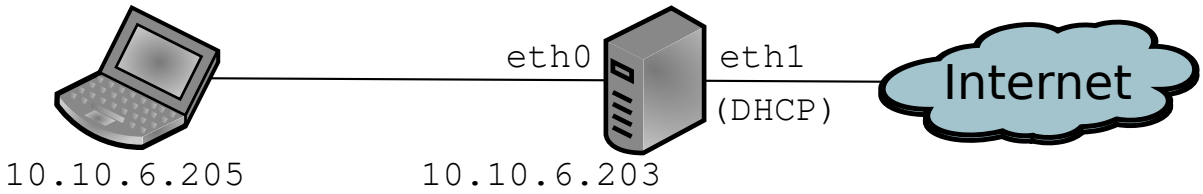


FIGURE 3.7 – Architecture réseau

L'accès à Internet du client se fait donc par l'intermédiaire de la machine jouant le rôle du pare-feu. De cette manière, tous les flux réseaux transitent par le système de filtrage.

Nous allons maintenant passer à la description de chacun des trois pare-feux.

3.3.2 Pare-feux testés

a) Sphirewall (version 0.9.9.27)

Sphirewall est un firewall/routeur (logiciel) opensource qui permet entre autres :
- de gérer des utilisateurs et la bande passante qui leur est allouée,
- de filtrer suivant les adresses IP, le contenu HTTP, ou encore les les URL (protocoles HTTP et HTTPS).

Tout ceci est paramétrable par l'intermédiaire d'une interface web. Une particularité notable de ce firewall est que le SNI est utilisé pour filtrer le protocole HTTPS.

b) Untangle NG Firewall (version 10.2.1)

Untangle est une distribution GNU-linux basée sur Debian, spécialisée pour monter un firewall et/ou une passerelle. Ce firewall permet de gérer chaque aspect de contrôle du réseau : du filtrage de contenu à la mise en cache de documents Web. Il est possible d'installer, de désinstaller et de configurer des modules directement. Il est gratuit dans sa version basique mais alors les fonctionnalités sont limitées.

Un module payant (mais disponible à l'essai pendant 14 jours) permet de faire du filtrage HTTPS sur SNI (et également sur le certificat du serveur) et ceci de manière transparente.

c) IPFire (version 2.15)

IPFire est une distribution Linux servant de routeur et de firewall, configurable via une interface web. Son architecture modulaire permet de l'adapter à ses besoins. Sans installation de modules supplémentaires, IPFire fournit, entre autre, un système de proxy (avec mise en cache), un service de VPN, un serveur DHCP ... La fonctionnalité qui nous intéresse est celle permettant de filtrer le trafic HTTPS.

3.3.3 Résultats

Les conditions d'expérimentation ont été les suivantes :
– Pour le premier pare-feu, l'extension a été testée avec les configurations suivantes :
  — Linux Mint 17 : Firefox 20, 21, 30, 31 et Firefox Nightly 33.
  — Ubuntu 12.04 : Firefox 31
  — Windows 7 : Firefox 31
– Pour les deux autres pare-feux, l'architecture réseau de la figure 3.7 a été virtualisée, et dans les deux cas, ils ont été testés avec Windows XP SP3 et Ubuntu 12.04. Sur ces deux machines virtuelles, la version 31 de Firefox a été utilisée .

Le filtrage au niveau du serveur DNS a été simulé en modifiant le fichier **hosts** des clients. Par exemple, pour bloquer Twitter, il suffit d'ajouter la ligne `127.0.0.1 twitter.com` à ce fichier.

Les résultats obtenus ayant été les mêmes quel que soit le pare-feu utilisé et quelle que soit la configuration du client.

Nous allons synthétiser une partie des tests effectués dans le tableau suivant :

Site	Type de filtrage	Type de déblocage	Succès/Échec	Remarques
Facebook	DNS	Fichier hosts interne	Succès	Trace en annexe D.1
Facebook	DNS	Serveur DNS : `8.8.8.8` (Google)	Succès	Trace en annexe D.2
Facebook	SNI	SNI factice	Succès	Trace en annexe D.3
Facebook	SNI et DNS	SNI factice et `8.8.8.8`	Succès	Trace en annexe D.4
Twitter	DNS	Fichier hosts interne	Succès	
Twitter	DNS	Serveur DNS : `216.146.35.35` (DynDNS)	Succès	
Twitter	SNI	SNI factice	Succès	
Dropbox	DNS	Fichier hosts interne	Succès	
Dropbox	SNI	SNI factice	Succès	

Table 3.1 – Récapitulatif des tests effectués

Site	Type de filtrage	Type de déblocage	Succès/Échec	Remarques
YouTube	SNI	SNI factice	Succès	
YouTube	SNI	« Encapsulation » Handshake : `maps.google.com`	Succès	Trace en annexe D.5
Google.fr	SNI	SNI factice	Succès	
LinkedIn	SNI	SNI factice	Succès	
GitHub	SNI	SNI factice	Succès	
Tumblr	SNI	SNI factice	Succès	
Flickr	SNI	SNI factice	Succès	
Flickr	SNI	« Encapsulation » Handshake : `yahoo.com`	Échec	Redirection vers `hbrb.yahoo.com`
Yahoo ! Sports	SNI	« Encapsulation » Handshake : `mail.yahoo.com`	Échec	Redirection vers `login.yahoo.com`
Yahoo ! Sports	SNI	« Encapsulation » Handshake : `fr.mail.yahoo.com`	Échec	Erreur 404

Table 3.2 – Récapitulatif des tests effectués (suite)

Tout comme Google, les services Yahoo ! se partagent le même certificat. Cependant leur comportement est différent par rapport à la technique consistant à réaliser le handshake TLS (« Encapsulation ») avec un autre site.

Si nous faisons un bilan, les deux techniques de contournement de filtrages réalisés au niveau du serveur DNS semblent être fonctionnelles dans tous les cas. Nous tirons la même conclusion par rapport au contournement de filtres sur le SNI en utilisant un SNI factice. Par contre, la technique de l' « encapsulation », semble fonctionner uniquement avec les services Google.

Nous allons désormais présenter la deuxième tâche réalisée.

4 Mise en place d'une architecture de collecte de trafic web chiffré

Cette partie présente l'architecture de collecte de trafic web chiffré mise en place. Celle-ci a pour but de fournir une base pour les futurs travaux de Wazen SHBAIR (évoqués en 2) et donc de lui permettre de mettre en place un filtrage plus intelligent, basé sur l'apprentissage automatique.

Tout d'abord, nous allons nous intéresser à l'architecture réseau de ce système, puis nous décrirons sa mise en œuvre, avant de détailler les résultats obtenus.

4.1 Architecture réseau

Pour avoir une architecture réseau pleinement fonctionnelle, il est nécessaire d'avoir deux cartes réseaux. En effet, avec une seule carte réseau, le trafic serait collecté uniquement dans le sens utilisateur vers serveur HTTPS mais pas la réponse du serveur. Si l'utilisateur spécifiait l'adresse IP du collecteur comme passerelle, le trafic passerait bien par la machine à l'aller, mais au retour le réseau (routeurs et commutateurs) connaîtrait l'utilisateur (adresses IP et MAC) et donc la réponse du serveur n'aurait pas à passer par le collecteur, et par ce fait ne serait pas capturée.

Or la réponse du serveur nous intéresse autant que la requête du client, c'est pourquoi le collecteur comporte deux cartes réseaux. L'architecture mise en place est schématisée en figure 4.1.

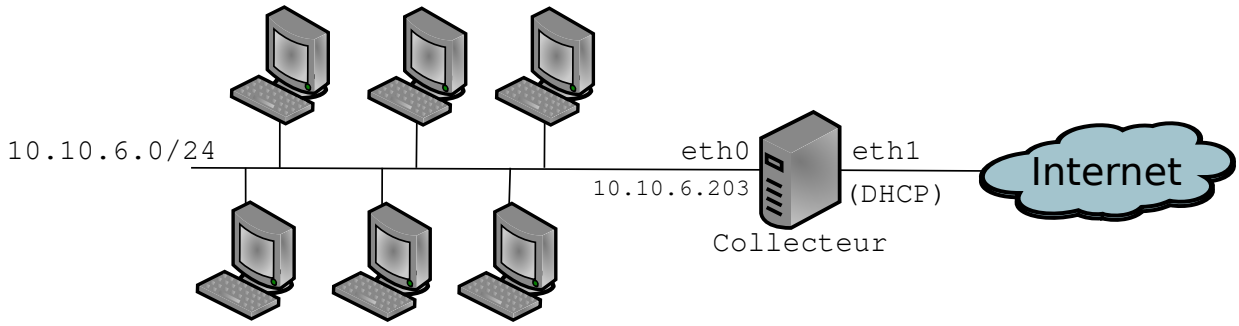


FIGURE 4.1 – Architecture réseau du collecteur

Nous avons donc mis en place l'architecture du collecteur, nous allons maintenant nous intéresser à la mise en œuvre de celui-ci.

4.2 Mise en œuvre

Dans cette section, nous allons décrire les différentes étapes de l'élaboration du collecteur : l'accès à Internet des utilisateurs, la capture du trafic chiffré, la division du fichier de capture pour une analyse plus rapide et enfin l'automatisation de ces tâches.

4.2.1 Accès des utilisateurs à Internet

La première fonction du collecteur est de permettre aux utilisateurs l'accès à Internet. Pour cela, un NAT (Network Address Translation) a été mis en place. Ainsi les adresses IP privées (en `10.10.6.0/24`) sont invisibles pour le reste du réseau.

Pour mettre en place ce système, **iptables** est utilisé : une interface en ligne de commande permettant de configurer Netfilter (un module du noyau Linux offrant la possibilité de contrôler, modifier et filtrer les paquets IP, et de suivre les connexions).

Le but ici n'étant pas de filtrer le trafic des utilisateurs, mais de sorte d'être le moins restrictif que possible. Pour cela, nous utilisons les deux règles suivantes :

```
iptables -t nat -A POSTROUTING -o eth1
iptables -A INPUT -j ACCEPT
```

La première règle spécifie qu'un NAT est effectué sur l'interface `eth1` (interface reliée à Internet) et la seconde ligne permet d'accepter tout le trafic entrant.

Maintenant que les utilisateurs peuvent accéder à Internet, il s'agit maintenant de réaliser la capture du trafic chiffré.

4.2.2 Capture du trafic chiffré

Pour limiter la taille des fichiers de captures, et aussi parce que les travaux de Wazen SHBAIR portent sur HTTPS, nous avons décidé de restreindre l'écoute du réseau au port TCP 443. Ce dernier étant le port utilisé par défaut par HTTPS.

Cette restriction étant faite, nous avons encore deux possibilités pour effectuer la capture : soit sur l'interface `eth0` soit sur l'interface `eth1`.

Notre choix s'est porté sur l'interface `eth0`, de cette manière, dans nos captures, les IP des utilisateurs seront les « vraies ». En effet, sur l'autre interface, tous les utilisateurs auraient la même IP : celle du collecteur sur l'interface `eth1` (c'est le rôle du NAT mis en place).

Si dans le futur, il s'avère que l'adresse IP de l'utilisateur n'a pas de rôle dans l'analyse des captures, nous pouvons décider d'effectuer la capture sur l'autre interface, et ainsi nous rendons anonyme l'utilisateur.

Pour réaliser l'écoute du réseau, l'outil de capture et d'analyse réseau, **tcpdump** est utilisé. Il permet de réaliser une analyse en direct du réseau ou d'enregistrer la capture dans un fichier. Des filtres peuvent être appliqués pour ne voir que certains types de paquets. Il est

basé sur la libpcap pour la capture, ce qui permet à ses sauvegardes d'être compatibles avec d'autres analyseurs réseaux, comme Wireshark.

Dans notre cas, le mode capture est utilisé pour conserver ce qui se passe sur le réseau et pour permettre une analyse plus tard.

Nous allons donc réaliser la capture avec cette commande :
`tcpdump -i eth0 port 443 -w fichier.pcap`
Nous retrouvons bien :
- l'interface de capture `eth0`, grâce à l'option `-i`,
- la limitation de la capture au port 443,
- l'enregistrement de la capture, grâce à l'option `-w`, dans le fichier `fichier.pcap`.

Cependant, le fait d'avoir un seul fichier de capture est devenu problématique. En effet, le fichier devenait volumineux assez rapidement, ce qui ralentissait son ouverture dans des outils comme Wireshark (pour l'analyser). Il a fallu trouver un critère pour scinder ce fichier en plusieurs fichiers plus légers, plus simples et donc plus rapides à analyser.

4.2.3 Division du fichier de capture

Le premier test a été de diviser le fichier de capture par rapport aux utilisateurs mais cela ne réduisait guère le problème, les fichiers étaient toujours trop volumineux.

L'idée suivante a été de considérer les flux comme critère. Un flux est caractérisé par un quadruplet : adresse IP source, port source, adresse IP de destination et port de destination.

Pour cela nous avons utilisé un petit utilitaire, écrit en C, nommé **pkt2flow** [1].

Une fois compilé, ce programme s'exécute avec la commande :
`./pkt2flow -o dossier_de_sortie fichier.pcap`

Cet utilitaire va extraire chaque flux et l'enregistrer dans un fichier *pcap* (packet capture) sous la forme :
`IPSource_portSource_IPDestination_portDestination_timestamp.pcap`.
Le timestamp correspond à la date et à l'heure du premier paquet du flux.

Ce petit programme est très rapide, mais cependant dans certains cas il divise en plusieurs fichiers des paquets appartenant au même flux. Ce phénomène correspond au timeout de **pkt2flow**, réglé à 30 minutes.

Dans ce cas là, nous exécutons le programme **mergecap** qui permet de fusionner plusieurs fichiers *pcap* en un seul, avec la commande :
`mergecap -w resultat.pcap fichier1.pcap fichier2.pcap`

Le fait d'utiliser **pkt2flow** (et de fusionner les fichiers « trop divisés »), à la fin de la capture, permet une analyse de capture plus rapide. L'étape suivante consiste à automatiser la capture et la division du fichier résultant que nous venons de détailler.

1. `https://github.com/caesar0301/pkt2flow`

4.2.4 Automatisation des tâches

Le processus correspondant aux deux sections précédentes (capture et division du fichier de capture) doit être effecué quotidiennement.

Pour débuter le processus d'automatisation, deux scripts ont été réalisés :

- **start-tcpdump** (annexe C) : qui va créer un dossier avec comme nom la date du jour puis va lancer **tcpdump** (sur l'interface `eth0` en écoute sur le port 443 et écriture dans un fichier)
- **stop-tcpdump** (annexe C) : qui va arrêter **tcpdump** puis diviser le(s) fichier(s) du jour en flux (à l'aide de **pkt2flow**)

Avec ces deux scripts, il suffit de lancer **start-tcpdump** pour lancer la capture puis d'exécuter **stop-tcpdump** pour l'arrêter et extraire les flux.

Mais pour aller encore plus loin dans le processus d'automatisation, le programme **cron** permet d'exécuter automatiquement une tâche à une date et à une heure spécifiées à l'avance, ou selon un cycle défini à l'avance.

Nous éditons la table de configuration de **cron** pour y ajouter ces lignes :

```
@reboot /root/cron-scripts/start-tcpdump
01 00 * * 1-5 /root/cron-scripts/start-tcpdump
59 23 * * 1-5 /root/cron-scripts/stop-tcpdump
```

Ces trois lignes permettent de démarrer la capture soit au (re)démarrage de la machine soit à 00h01 (quand la machine reste allumée jour et nuit), puis, à 23h59, d'arrêter la capture et de diviser le(s) fichier(s) de capture du jour pour permettre l'analyse de la trace. Et ceci tous les jours de la semaine (hors samedi et dimanche).

4.3 Résultats

Les résultats qui vont suivre correspondent au trafic de 4 utilisateurs entre le 30 juillet et le 12 août.

Chaque jour, nous retrouvons un dossier structuré de cette manière :

```
2014-08-01
|-- 2014-08-01--00-01-01.pcap
+-- streams [3107 fichiers]
    |-- 10.10.6.123_32768_173.252.100.27_443_1406910754.pcap
    |-- 10.10.6.254_32876_108.160.165.84_443_1406873492.pcap
    |-- 10.10.6.66_32789_199.16.156.70_443_1406884792.pcap
```

Nous retrouvons, dans ce dossier, la capture du jour (ici, démarrée le 1er août à `0:01:01`) ainsi que l'ensemble des flux correspondants à la capture dans le dossier « streams » (ici, 3107 flux ce jour là, nous en avons sélectionné trois pour l'exemple).

Des informations peuvent être obtenues juste avec le nom des fichiers (et en utilisant une résolution DNS inverse).

Ainsi, pour les trois fichiers affichés plus haut, nous pouvons déduire que :
– À `18:32:34`, l'utilisateur ayant comme IP 10.10.6.123 a visité facebook.com.
– À `8:11:32`, l'utilisateur ayant comme IP 10.10.6.254 est allé sur dropbox.com.
– À `11:19:52`, l'utilisateur ayant comme IP 10.10.6.66 s'est rendu sur twitter.com.

Ci-dessous, nous trouverons un tableau récapitulant la taille du fichier de capture ainsi que le nombre de flux extraits de celui-ci.

Date	Taille du fichier de capture (Mo)	Nombre de flux
30/07	159	2392
31/07	985	5344
01/08	120	3107
04/08	371	4156
05/08	306	7742
06/08	214	4878
07/08	599	6962
08/08	364	4628
11/08	153	4274
12/08	600	2462

Table 4.1 – Récapitulatif de la capture entre le 30/07 et le 12/08

Nous pouvons voir qu'il n'y a pas de réelle corrélation entre la taille du fichier de capture et le nombre de flux et ceci peut s'expliquer. En effet, la visualisation d'une vidéo va prendre de la place dans le fichier de capture mais ne représente qu'un seul flux. À l'inverse, une page utilisant un grand nombre de ressources (images par exemple) hébergées sur d'autres serveurs ne représentera pas un grand pourcentage en terme de volume, mais provoquera une multitude de flux (au moins un flux par hébergeur externe).

Les fichiers *pcap* correspondants aux flux sont d'ores et déjà exploités par Wazen SHBAIR. Après l'exécution des scripts mis en place, il lance un programme qui extrait, de ces fichiers, un certain nombre d'informations sur les communications entre les utilisateurs et les serveurs.

L'objectif sera de mettre en place un apprentissage automatique (*machine learning*) et ainsi, dans un premier temps, de pouvoir retrouver sur quel site l'utilisateur est allé (via le protocole HTTPS), sans avoir accès au SNI, et à terme de construire un outil de filtrage robuste, reposant sur cet apprentissage.

Dans cette partie, nous avons donc vu l'architecture du collecteur de trafic web chiffré mise en place puis les étapes de sa mise en œuvre. Et enfin les résultats de la collecte du trafic de 4 utilisateurs sur une période de deux semaines.

Conclusion

Dans ce rapport, nous avons présenté les différents filtrages existants sur Internet et en particulier les outils de filtrage adaptés aux flux chiffrés. Ces moyens de filtrages sont impératifs car ils représentent un dispositif de sécurité. Cependant, nous avons montré que les systèmes actuels ont des limites et sont facilement contournables et qu'il faut donc penser à mettre en place des pare-feux plus robustes, plus « intelligents ».

Les faiblesses du filtrage basé sur le SNI et de celui situé au niveau du serveur DNS, ainsi que leurs méthodes de contournement ont été illustrés au travers d'une extension pour le navigateur Firefox. Celle-ci est capable de déjouer :

— un filtrage DNS (soit en contactant un autre serveur DNS, soit en utilisant un fichier **hosts** interne à l'extension),

— un filtrage sur le SNI (Server Name Indication, un champ du ClientHello du Handshake TLS). Pour cela, soit un SNI factice est envoyé au serveur, soit le Handshake TLS est initié avec un autre serveur (par exemple, nous récupérons une page du site YouTube en réalisant le Handshake avec Gmail)

J'ai également élaboré une architecture de collecte de trafic HTTPS, elle repose principalement sur **iptables** pour la partie routage, et **tcpdump** pour la partie capture de paquets. Ce collecteur fournit un support pour l'élaboration d'un futur moyen de filtrage reposant sur l'apprentissage automatique (*machine learning*).

Ce stage a permis de renforcer mes connaissances en réseau, mais aussi de découvrir en détail le protocole HTTPS (et en particulier la couche TLS). Avec la mise en place du collecteur de trafic HTTPS, j'ai eu l'occasion de faire de l'administration système. Le développement de l'extension pour le navigateur Firefox m'a permis d'apprendre le JavaScript en même temps. En outre, j'ai eu l'occasion de voir de plus près, le métier de chercheur dans un laboratoire de recherche en informatique.

Pour poursuivre ces travaux, nous pourrions ajouter le contournement de filtrage DNS pour le protocole HTTP dans l'extension Firefox et ainsi augmenter son champ d'actions.

Bibliographie / Webographie

[1] Wazen M. Shbair, Isabelle Chrisment et Thibault Cholez : HTTPS Traffic Filtering and Bypassing Techniques. Rapport technique, Inria, avril 2014. (p. 4, 13)

[2] Wazen M. Shbair, Isabelle Chrisment et Thibault Cholez : How to bypass SNI-based HTTPS Filtering ?, mai 2014. (Draft). (p. 4, 13)

[3] T. Dierks et E. Rescorla : The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), août 2008. `http://tools.ietf.org/html/rfc5246` (consulté en aout 2014). (p. 8)

[4] S. Blake-Wilson, M. Nystrom, D. Hopwood, J. Mikkelsen et T. Wright : Transport Layer Security (TLS) Extensions. RFC 3546 (Proposed Standard), juin 2003. `http://tools.ietf.org/html/rfc3546` (consulté en aout 2014). (p. 10)

[5] D. Eastlake 3rd : Transport Layer Security (TLS) Extensions : Extension Definitions. RFC 6066 (Proposed Standard), janvier 2011. `http://tools.ietf.org/html/rfc6066` (consulté en aout 2014). (p. 10, 14)

[6] Inria en quelques mots : `http://www.inria.fr/institut/inria-en-bref/inria-en-quelques-mots` (consulté en juillet 2014).

[7] Présentation d'Inria en quelques chiffres : `http://www.inria.fr/institut/inria-en-bref/chiffres-cles` (consulté en juillet 2014).

[8] Equipes de recherche du centre Nancy : `http://www.inria.fr/recherches/structures-de-recherche/rechercher-une-equipe/(center)/1166` (consulté en juillet 2014).

[9] Présentation du Loria : `http://www.loria.fr/la-recherche` (consulté en juillet 2014).

[10] Présentation de l'équipe de recherche MADYNES : `http://www.inria.fr/equipes/madynes` (consulté en juillet 2014).

[11] Rapport d'activité de l'année 2013 de l'équipe MADYNES : `http://raweb.inria.fr/rapportsactivite/RA2013/madynes/uid0.html` (consulté en juillet 2014).

[12] Documentation Ubuntu sur iptables : `http://doc.ubuntu-fr.org/iptables` (consulté en juillet 2014).

[13] Red Hat Enterprise Linux 4 : Guide de sécurité - Chapitre 7. Pare-feu : `http://web.mit.edu/rhel-doc/4/RH-DOCS/rhel-sg-fr-4/s1-firewall-ipt-fwd.html` (consulté en juillet 2014).

[14] Programmer des tâches avec CRON : `http://doc.ubuntu-fr.org/cron` (consulté en juillet 2014).

Liste des illustrations

2.1 Filtrage au niveau du serveur DNS 6

2.2 Paquet comportant le nom de domaine YouTube 7

2.3 Paquet comportant le mot « informatique » 8

2.4 Le Handshake TLS 9

3.1 Première technique de contournement de filtrages sur le SNI 14

3.2 Seconde technique de contournement de filtrages sur le SNI 14

3.3 Troisième technique de contournement de filtrages sur le SNI 15

3.4 Extension : Fonctionnement du Man-In-The-Middle 16

3.5 Extension : onglet « Bypassing HTTPS filtering » 16

3.6 Extension : onglet « Bypassing DNS filtering » 18

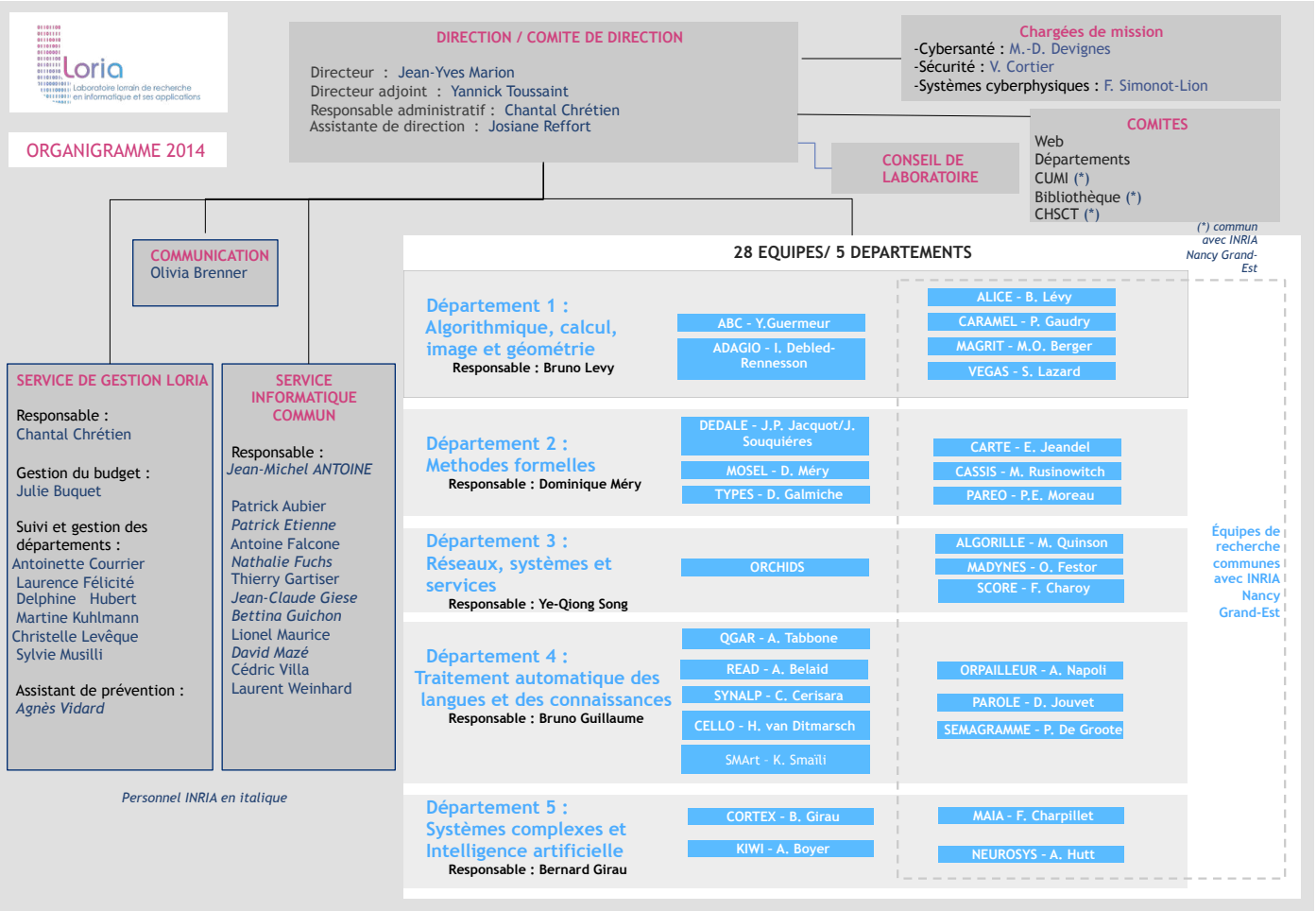
3.7 Architecture réseau 19

4.1 Architecture réseau du collecteur 22

Annexes

A Organigramme d'Inria

B Organigramme du Loria



C Scripts employés pour l'architecture de collecte de flux chiffrés

Script de début de capture

```
#!/bin/bash

folder=`date +%F`
filename=`date +%F--%H-%M-%S`

if [ ! -d "$folder" ]; then
  mkdir /root/dumps/$folder
fi


#Start tcpdump listening on eth0 (port 443) and
#dump the output to /root/dumps/Y-M-D/Y-M-D--HH-MM-SS.pcap

/usr/sbin/tcpdump -i eth0 port 443 \
                  -w /root/dumps/$folder/$filename.pcap
```

Script de fin de capture

```
#!/bin/bash

folder=`date +%F`
syn="tcp_syn"
nosyn="tcp_nosyn"
streams="streams"



#Kill tcpdump
killall -15 tcpdump



cd /root/dumps/$folder

if [ ! -d "$streams" ]; then
  mkdir $streams
fi
```

```
#For each pcap
for f in *.pcap
do
  #Exctract all streams
  /root/pkt2flow/pkt2flow -o . $f
  cd  $nosyn/
  for file in *.pcap
  do
    str=`echo $file | cut -d '_' -f -4`
    src=`echo $file | cut -d '_' -f -2`
    dst=`echo $file | cut -d '_' -f 3-4`

    cd ../tcp_syn

    #Merge duplicate files (src_dst)
    find . -name "$str*.pcap" -exec \
          mergecap ../tcp_nosyn/$file {} -w {}2 \;
    find . -name "*.pcap2" -exec bash \
      -c 'mv $0 ${0/pcap2/pcap}; rm ../tcp_nosyn/$1' {} $file \;

    #Merge duplicate files (dst_src)
    find . -name "$dst$_$src*.pcap" -exec \
      mergecap ../tcp_nosyn/$file {} -w {}2 \;
    find . -name "*.pcap2" -exec bash \
      -c 'mv $0 ${0/pcap2/pcap}; rm ../tcp_nosyn/$1' {} $file \;


    cd  ../$nosyn/
  done
  cd /root/dumps/$folder

  #Gather files
  if [ -d "$syn" ]; then
    mv "$syn"/* $streams
    rm -rf "$syn"
  fi
  if [ -d "$nosyn" ]; then
    mv "$nosyn"/* $streams
    rm -rf "$nosyn"
  fi

done
```

D Scenarii de contournement

D.1 Facebook bloqué par un filtrage DNS, contourné en utilisant le fichier hosts interne à l'extension (IP de Facebook : 173.252.110.27)

4	0.008849000	127.0.0.1	127.0.0.1	HTTP	269	CONNECT facebook.com:443 HTTP/1.1
5	0.008892000	127.0.0.1	127.0.0.1	TCP	68	52186 > 36287 [ACK] Seq=1 Ack=202 Win=44800 Len=0 TSval=3248735 TSecr=3248735
6	0.083489000	192.168.0.14	173.252.110.27	TCP	76	45659 > https [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=3248754 TSecr
7	0.235531000	173.252.110.27	192.168.0.14	TCP	76	https > 45659 [SYN, ACK] Seq=0 Ack=1 Win=14480 Len=0 MSS=1412 SACK_PERM=1 TSval=99
8	0.235587000	192.168.0.14	173.252.110.27	TCP	68	45659 > https [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=3248792 TSecr=990048240
9	0.236656000	192.168.0.14	173.252.110.27	TLSv1.2	234	Client Hello
10	0.389797000	173.252.110.27	192.168.0.14	TCP	68	https > 45659 [ACK] Seq=1 Ack=167 Win=15616 Len=0 TSval=990048394 TSecr=3248792
11	0.394057000	173.252.110.27	192.168.0.14	TLSv1.2	1468	Server Hello
12	0.394084000	192.168.0.14	173.252.110.27	TCP	68	45659 > https [ACK] Seq=167 Ack=1401 Win=32128 Len=0 TSval=3248831 TSecr=990048394
13	0.396683000	173.252.110.27	192.168.0.14	TCP	1468	[TCP segment of a reassembled PDU]
14	0.396697000	192.168.0.14	173.252.110.27	TCP	68	45659 > https [ACK] Seq=167 Ack=2801 Win=35072 Len=0 TSval=3248832 TSecr=990048394
15	0.398025000	173.252.110.27	192.168.0.14	TLSv1.2	724	Certificate
16	0.398036000	192.168.0.14	173.252.110.27	TCP	68	45659 > https [ACK] Seq=167 Ack=3457 Win=37888 Len=0 TSval=3248832 TSecr=990048394
17	0.402200000	192.168.0.14	173.252.110.27	TLSv1.2	194	Client Key Exchange, Change Cipher Spec, Hello Request, Hello Request
18	0.555061000	173.252.110.27	192.168.0.14	TCP	68	https > 45659 [ACK] Seq=3457 Ack=293 Win=15616 Len=0 TSval=990048559 TSecr=3248833
19	0.556054000	173.252.110.27	192.168.0.14	TLSv1.2	326	New Session Ticket, Change Cipher Spec, Encrypted Handshake Message
22	0.595146000	192.168.0.14	173.252.110.27	TCP	68	45659 > https [ACK] Seq=293 Ack=3715 Win=40704 Len=0 TSval=3248882 TSecr=990048559
32	0.699273000	127.0.0.1	127.0.0.1	HTTP	107	HTTP/1.1 200 Connection established
33	0.699327000	127.0.0.1	127.0.0.1	TCP	68	36287 > 52186 [ACK] Seq=202 Ack=40 Win=43776 Len=0 TSval=3248908 TSecr=3248908
34	0.699600000	127.0.0.1	127.0.0.1	TLSv1.2	247	Client Hello
35	0.699619000	127.0.0.1	127.0.0.1	TCP	68	52186 > 36287 [ACK] Seq=40 Ack=381 Win=45952 Len=0 TSval=3248908 TSecr=3248908
36	0.761458000	127.0.0.1	127.0.0.1	TLSv1.2	1197	Server Hello, Certificate, Server Key Exchange, Server Hello Done
37	0.764220000	127.0.0.1	127.0.0.1	TLSv1.2	194	Client Key Exchange, Change Cipher Spec, Hello Request, Hello Request
38	0.764244000	127.0.0.1	127.0.0.1	TCP	68	52186 > 36287 [ACK] Seq=1169 Ack=507 Win=45952 Len=0 TSval=3248924 TSecr=3248924
39	0.766202000	127.0.0.1	127.0.0.1	TLSv1.2	280	New Session Ticket, Change Cipher Spec, Hello Request, Hello Request
40	0.766466000	127.0.0.1	127.0.0.1	TLSv1.2	653	Application Data

```
    Cipher Suites Length: 46
  ▸ Cipher Suites (23 suites)
    Compression Methods Length: 1
  ▸ Compression Methods (1 method)
    Extensions Length: 70
  ▾ Extension: server_name
      Type: server_name (0x0000)
      Length: 17
    ▾ Server Name Indication extension
        Server Name list length: 15
        Server Name Type: host_name (0)
        Server Name length: 12
        Server Name: facebook.com
```

D.2 Facebook bloqué par un filtrage DNS, contourné en contactant le serveur DNS public de Google

No.	Time	Source	Destination	Protocol	Length	Info
35	5.388502000	127.0.0.1	127.0.0.1	HTTP	269	CONNECT facebook.com:443 HTTP/1.1
37	5.462542000	192.168.0.14	8.8.8.8	DNS	74	Standard query 0x3030 A facebook.com
38	5.502483000	8.8.8.8	192.168.0.14	DNS	90	Standard query response 0x3030 A 173.252.110.27
39	5.505013000	192.168.0.14	173.252.110.27	TCP	76	44934 > https [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=
40	5.658242000	173.252.110.27	192.168.0.14	TCP	76	https > 44934 [SYN, ACK] Seq=0 Ack=1 Win=14480 Len=0 MSS=1412
41	5.658295000	192.168.0.14	173.252.110.27	TCP	68	44934 > https [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=1329635
42	5.658933000	192.168.0.14	173.252.110.27	TLSv1.2	234	Client Hello
44	5.812966000	173.252.110.27	192.168.0.14	TCP	68	https > 44934 [ACK] Seq=1 Ack=167 Win=15616 Len=0 TSval=16629
45	5.815907000	173.252.110.27	192.168.0.14	TLSv1.2	1468	Server Hello
46	5.815933000	192.168.0.14	173.252.110.27	TCP	68	44934 > https [ACK] Seq=167 Ack=1401 Win=32128 Len=0 TSval=13
47	5.818898000	173.252.110.27	192.168.0.14	TCP	1468	[TCP segment of a reassembled PDU]
48	5.818915000	192.168.0.14	173.252.110.27	TCP	68	44934 > https [ACK] Seq=167 Ack=2801 Win=35072 Len=0 TSval=13
49	5.819337000	173.252.110.27	192.168.0.14	TLSv1.2	724	Certificate
50	5.819354000	192.168.0.14	173.252.110.27	TCP	68	44934 > https [ACK] Seq=167 Ack=3457 Win=37888 Len=0 TSval=13
51	5.826230000	192.168.0.14	173.252.110.27	TLSv1.2	194	Client Key Exchange, Change Cipher Spec, Hello Request, Hello
52	5.978761000	173.252.110.27	192.168.0.14	TCP	68	https > 44934 [ACK] Seq=3457 Ack=293 Win=15616 Len=0 TSval=16
53	5.980229000	173.252.110.27	192.168.0.14	TLSv1.2	326	New Session Ticket, Change Cipher Spec, Encrypted Handshake M
56	6.017684000	192.168.0.14	173.252.110.27	TCP	68	44934 > https [ACK] Seq=293 Ack=3715 Win=40704 Len=0 TSval=13
66	6.155947000	127.0.0.1	127.0.0.1	HTTP	107	HTTP/1.1 200 Connection established
67	6.156006000	127.0.0.1	127.0.0.1	TCP	68	54087 > 37374 [ACK] Seq=202 Ack=40 Win=342 Len=0 TSval=132975
68	6.156298000	127.0.0.1	127.0.0.1	TLSv1.2	247	Client Hello
69	6.156317000	127.0.0.1	127.0.0.1	TCP	68	37374 > 54087 [ACK] Seq=40 Ack=381 Win=359 Len=0 TSval=132975
70	6.183007000	127.0.0.1	127.0.0.1	TLSv1.2	1197	Server Hello, Certificate, Server Key Exchange, Server Hello
71	6.188290000	127.0.0.1	127.0.0.1	TLSv1.2	194	Client Key Exchange, Change Cipher Spec, Hello Request, Hello
72	6.191041000	127.0.0.1	127.0.0.1	TLSv1.2	280	New Session Ticket, Change Cipher Spec, Hello Request, Hello
73	6.191475000	127.0.0.1	127.0.0.1	TLSv1.2	653	Application Data
74	6.192855000	192.168.0.14	173.252.110.27	TLSv1.2	653	Application Data
75	6.229622000	127.0.0.1	127.0.0.1	TCP	68	37374 > 54087 [ACK] Seq=1381 Ack=1092 Win=368 Len=0 TSval=132
76	6.350303000	173.252.110.27	192.168.0.14	TCP	68	https > 44934 [ACK] Seq=3715 Ack=878 Win=18688 Len=0 TSval=16
77	6.369619000	173.252.110.27	192.168.0.14	TLSv1.2	521	Application Data
78	6.369654000	192.168.0.14	173.252.110.27	TCP	68	44934 > https [ACK] Seq=878 Ack=4168 Win=43392 Len=0 TSval=13
79	6.370156000	127.0.0.1	127.0.0.1	TLSv1.2	521	Application Data

```
▸ Frame 42: 234 bytes on wire (1872 bits), 234 bytes captured (1872 bits) on interface 0
▸ Linux cooked capture
▸ Internet Protocol Version 4, Src: 192.168.0.14 (192.168.0.14), Dst: 173.252.110.27 (173.252.110.27)
▸ Transmission Control Protocol, Src Port: 44934 (44934), Dst Port: https (443), Seq: 1, Ack: 1, Len: 166
▾ Secure Sockets Layer
  ▾ TLSv1.2 Record Layer: Handshake Protocol: Client Hello
      Content Type: Handshake (22)
      Version: TLS 1.0 (0x0301)
      Length: 161
    ▾ Handshake Protocol: Client Hello
        Handshake Type: Client Hello (1)
        Length: 157
        Version: TLS 1.2 (0x0303)
      ▸ Random
        Session ID Length: 0
        Cipher Suites Length: 46
      ▸ Cipher Suites (23 suites)
        Compression Methods Length: 1
      ▸ Compression Methods (1 method)
        Extensions Length: 70
      ▾ Extension: server_name
          Type: server_name (0x0000)
          Length: 17
        ▾ Server Name Indication extension
            Server Name list length: 15
            Server Name Type: host_name (0)
            Server Name length: 12
            Server Name: facebook.com
```

D.3 Facebook bloqué par un filtrage SNI, contourné en utilisant un SNI factice

4	0.013304000	127.0.0.1	127.0.0.1	HTTP	269	CONNECT facebook.com:443 HTTP/1.1
6	0.081669000	127.0.0.1	127.0.1.1	DNS	74	Standard query 0x68a5 A facebook.com
7	0.081753000	192.168.0.14	212.27.40.240	DNS	74	Standard query 0xfbe6 A facebook.com
8	0.081784000	192.168.0.14	212.27.40.241	DNS	74	Standard query 0xfbe6 A facebook.com
9	0.126916000	212.27.40.240	192.168.0.14	DNS	157	Standard query response 0xfbe6 A 173.252.110.27
10	0.127054000	127.0.1.1	127.0.0.1	DNS	157	Standard query response 0x68a5 A 173.252.110.27
11	0.127519000	192.168.0.14	173.252.110.27	TCP	76	45029 > https [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=
12	0.142973000	212.27.40.241	192.168.0.14	DNS	157	Standard query response 0xfbe6 A 173.252.110.27
14	0.327052000	173.252.110.27	192.168.0.14	TCP	76	https > 45029 [SYN, ACK] Seq=0 Ack=1 Win=14480 Len=0 MSS=1412
15	0.327115000	192.168.0.14	173.252.110.27	TCP	68	45029 > https [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=1560106
16	0.327972000	192.168.0.14	173.252.110.27	TLSv1.2	233	Client Hello
17	0.505111000	173.252.110.27	192.168.0.14	TCP	68	https > 45029 [ACK] Seq=1 Ack=166 Win=15616 Len=0 TSval=15686
18	0.508303000	173.252.110.27	192.168.0.14	TLSv1.2	1468	Server Hello
19	0.508337000	192.168.0.14	173.252.110.27	TCP	68	45029 > https [ACK] Seq=166 Ack=1401 Win=32128 Len=0 TSval=15
20	0.510791000	173.252.110.27	192.168.0.14	TCP	1468	[TCP segment of a reassembled PDU]
21	0.510813000	192.168.0.14	173.252.110.27	TCP	68	45029 > https [ACK] Seq=166 Ack=2801 Win=35072 Len=0 TSval=15
22	0.511828000	173.252.110.27	192.168.0.14	TLSv1.2	719	Certificate
23	0.511857000	192.168.0.14	173.252.110.27	TCP	68	45029 > https [ACK] Seq=166 Ack=3452 Win=37888 Len=0 TSval=15
24	0.518825000	192.168.0.14	173.252.110.27	TLSv1.2	194	Client Key Exchange, Change Cipher Spec, Hello Request, Hello
25	0.692543000	173.252.110.27	192.168.0.14	TCP	68	https > 45029 [ACK] Seq=3452 Ack=292 Win=15616 Len=0 TSval=15
26	0.692576000	173.252.110.27	192.168.0.14	TLSv1.2	326	New Session Ticket, Change Cipher Spec, Encrypted Handshake M
29	0.729249000	192.168.0.14	173.252.110.27	TCP	68	45029 > https [ACK] Seq=292 Ack=3710 Win=40704 Len=0 TSval=15
39	0.822940000	127.0.0.1	127.0.0.1	HTTP	107	HTTP/1.1 200 Connection established
40	0.822989000	127.0.0.1	127.0.0.1	TCP	68	54182 > 37374 [ACK] Seq=202 Ack=40 Win=342 Len=0 TSval=156023
41	0.823286000	127.0.0.1	127.0.0.1	TLSv1.2	585	Client Hello
42	0.823307000	127.0.0.1	127.0.0.1	TCP	68	37374 > 54182 [ACK] Seq=40 Ack=719 Win=359 Len=0 TSval=156023
43	0.861273000	127.0.0.1	127.0.0.1	TLSv1.2	211	Server Hello, Change Cipher Spec, Hello Request, Hello Reques
44	0.861658000	127.0.0.1	127.0.0.1	TLSv1.2	119	Change Cipher Spec, Hello Request, Hello Request
45	0.861679000	127.0.0.1	127.0.0.1	TCP	68	37374 > 54182 [ACK] Seq=183 Ack=770 Win=359 Len=0 TSval=15602
46	0.862346000	127.0.0.1	127.0.0.1	TLSv1.2	827	Application Data

```
▸ Frame 16: 233 bytes on wire (1864 bits), 233 bytes captured (1864 bits) on interface 0
▸ Linux cooked capture
▸ Internet Protocol Version 4, Src: 192.168.0.14 (192.168.0.14), Dst: 173.252.110.27 (173.252.110.27)
▸ Transmission Control Protocol, Src Port: 45029 (45029), Dst Port: https (443), Seq: 1, Ack: 1, Len: 165
▾ Secure Sockets Layer
  ▾ TLSv1.2 Record Layer: Handshake Protocol: Client Hello
      Content Type: Handshake (22)
      Version: TLS 1.0 (0x0301)
      Length: 160
    ▾ Handshake Protocol: Client Hello
        Handshake Type: Client Hello (1)
        Length: 156
        Version: TLS 1.2 (0x0303)
      ▸ Random
        Session ID Length: 0
        Cipher Suites Length: 46
      ▸ Cipher Suites (23 suites)
        Compression Methods Length: 1
      ▸ Compression Methods (1 method)
        Extensions Length: 69
      ▾ Extension: server_name
          Type: server_name (0x0000)
          Length: 16
        ▾ Server Name Indication extension
            Server Name list length: 14
            Server Name Type: host_name (0)
            Server Name length: 11
            Server Name: fauxsni.com
```

D.4 Facebook bloqué par un filtrage DNS et SNI, contourné en contactant le serveur DNS public de Google et en utilisant un SNI factice

26	5.340748000	127.0.0.1	127.0.0.1	HTTP	269	CONNECT facebook.com:443 HTTP/1.1
27	5.340794000	127.0.0.1	127.0.0.1	TCP	68	58964 > 33515 [ACK] Seq=1 Ack=202 Win=44800 Len=0 TSval=1703747 TSecr=1703747
28	5.408068000	192.168.0.14	8.8.8.8	DNS	74	Standard query 0x3030 A facebook.com
29	5.447211000	8.8.8.8	192.168.0.14	DNS	90	Standard query response 0x3030 A 173.252.110.27
30	5.449753000	192.168.0.14	173.252.110.27	TCP	76	45146 > https [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=1703774 TSecr=0
31	5.601526000	173.252.110.27	192.168.0.14	TCP	76	https > 45146 [SYN, ACK] Seq=0 Ack=1 Win=14480 Len=0 MSS=1412 SACK_PERM=1 TSval=1390
32	5.601577000	192.168.0.14	173.252.110.27	TCP	68	45146 > https [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=1703812 TSecr=1390386377
33	5.602594000	192.168.0.14	173.252.110.27	TLSv1.2	233	Client Hello
34	5.754758000	173.252.110.27	192.168.0.14	TCP	68	https > 45146 [ACK] Seq=1 Ack=166 Win=15616 Len=0 TSval=1390386531 TSecr=1703812
35	5.757909000	173.252.110.27	192.168.0.14	TLSv1.2	1468	Server Hello
36	5.757945000	192.168.0.14	173.252.110.27	TCP	68	45146 > https [ACK] Seq=166 Ack=1401 Win=32128 Len=0 TSval=1703851 TSecr=1390386531
37	5.760969000	173.252.110.27	192.168.0.14	TCP	1468	[TCP segment of a reassembled PDU]
38	5.761004000	192.168.0.14	173.252.110.27	TCP	68	45146 > https [ACK] Seq=166 Ack=2801 Win=35072 Len=0 TSval=1703852 TSecr=1390386531
39	5.761925000	173.252.110.27	192.168.0.14	TLSv1.2	720	Certificate
40	5.761942000	192.168.0.14	173.252.110.27	TCP	68	45146 > https [ACK] Seq=166 Ack=3453 Win=37888 Len=0 TSval=1703852 TSecr=1390386531
41	5.772432000	192.168.0.14	173.252.110.27	TLSv1.2	194	Client Key Exchange, Change Cipher Spec, Hello Request, Hello Request
42	5.924037000	173.252.110.27	192.168.0.14	TCP	68	https > 45146 [ACK] Seq=3453 Ack=292 Win=15616 Len=0 TSval=1390386700 TSecr=1703855
43	5.925002000	173.252.110.27	192.168.0.14	TLSv1.2	326	New Session Ticket, Change Cipher Spec, Encrypted Handshake Message
46	5.963527000	192.168.0.14	173.252.110.27	TCP	68	45146 > https [ACK] Seq=292 Ack=3711 Win=40704 Len=0 TSval=1703903 TSecr=1390386700
48	5.965958000	127.0.1.1	127.0.0.1	DNS	132	Standard query response 0xbba9 CNAME cs9.wac.edgecastcdn.net A 93.184.220.29
56	6.215215000	127.0.0.1	127.0.0.1	HTTP	107	HTTP/1.1 200 Connection established
57	6.215270000	127.0.0.1	127.0.0.1	TCP	68	33515 > 58964 [ACK] Seq=202 Ack=40 Win=43776 Len=0 TSval=1703965 TSecr=1703965
58	6.215591000	127.0.0.1	127.0.0.1	TLSv1.2	247	Client Hello
59	6.215617000	127.0.0.1	127.0.0.1	TCP	68	58964 > 33515 [ACK] Seq=40 Ack=381 Win=45952 Len=0 TSval=1703966 TSecr=1703966
60	6.273786000	127.0.0.1	127.0.0.1	TLSv1.2	1198	Server Hello, Certificate, Server Key Exchange, Server Hello Done
61	6.279537000	127.0.0.1	127.0.0.1	TLSv1.2	194	Client Key Exchange, Change Cipher Spec, Hello Request, Hello Request
62	6.279585000	127.0.0.1	127.0.0.1	TCP	68	58964 > 33515 [ACK] Seq=1170 Ack=507 Win=45952 Len=0 TSval=1703982 TSecr=1703982
63	6.281558000	127.0.0.1	127.0.0.1	TLSv1.2	280	New Session Ticket, Change Cipher Spec, Hello Request, Hello Request
64	6.281811000	127.0.0.1	127.0.0.1	TLSv1.2	653	Application Data

```
  Compression Methods Length: 1
  ▸ Compression Methods (1 method)
    Extensions Length: 69
  ▾ Extension: server_name
      Type: server_name (0x0000)
      Length: 16
    ▾ Server Name Indication extension
        Server Name list length: 14
        Server Name Type: host_name (0)
        Server Name length: 11
        Server Name: fauxsni.com
```

D.5 YouTube bloqué par un filtrage SNI, contourné en utilisant la technique de l'encapsulation (décrite en 3.1.1), Handshake avec `maps.google.com`

73	5.322637000	127.0.0.1	127.0.0.1	HTTP	275	CONNECT www.youtube.com:443 HTTP/1.1
74	5.322678000	127.0.0.1	127.0.0.1	TCP	68	52186 > 36386 [ACK] Seq=1 Ack=208 Win=44800 Len=0 TSval=3345628 TSecr=3345628
75	5.324062000	127.0.0.1	127.0.1.1	DNS	77	Standard query 0x0098 A maps.google.com
78	5.359625000	127.0.1.1	127.0.0.1	DNS	253	Standard query response 0x0098 A 173.194.45.66 A 173.194.45.71 A 173.194.45.78 A 1
79	5.360412000	192.168.0.14	173.194.45.66	TCP	76	59749 > https [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=3345638 TSecr=
80	5.395763000	173.194.45.66	192.168.0.14	TCP	76	https > 59749 [SYN, ACK] Seq=0 Ack=1 Win=42540 Len=0 MSS=1412 SACK_PERM=1 TSval=719
81	5.395805000	192.168.0.14	173.194.45.66	TCP	68	59749 > https [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=3345647 TSecr=719737212
82	5.396616000	192.168.0.14	173.194.45.66	TLSv1.2	237	Client Hello
83	5.433179000	173.194.45.66	192.168.0.14	TCP	68	https > 59749 [ACK] Seq=1 Ack=170 Win=42432 Len=0 TSval=719737249 TSecr=3345647
84	5.436366000	173.194.45.66	192.168.0.14	TLSv1.2	1468	Server Hello
85	5.436395000	192.168.0.14	173.194.45.66	TCP	68	59749 > https [ACK] Seq=170 Ack=1401 Win=32128 Len=0 TSval=3345657 TSecr=719737249
86	5.438860000	173.194.45.66	192.168.0.14	TCP	1468	[TCP segment of a reassembled PDU]
87	5.438878000	192.168.0.14	173.194.45.66	TCP	68	59749 > https [ACK] Seq=170 Ack=2801 Win=35072 Len=0 TSval=3345657 TSecr=719737249
88	5.440898000	173.194.45.66	192.168.0.14	TLSv1.2	1140	Certificate
89	5.440919000	192.168.0.14	173.194.45.66	TCP	68	59749 > https [ACK] Seq=170 Ack=3873 Win=37888 Len=0 TSval=3345658 TSecr=719737249
90	5.448799000	192.168.0.14	173.194.45.66	TLSv1.2	194	Client Key Exchange, Change Cipher Spec, Hello Request, Hello Request
91	5.484917000	173.194.45.66	192.168.0.14	TLSv1.2	314	New Session Ticket, Change Cipher Spec, Hello Request, Hello Request
94	5.523490000	192.168.0.14	173.194.45.66	TCP	68	59749 > https [ACK] Seq=296 Ack=4119 Win=40704 Len=0 TSval=3345679 TSecr=719737301
104	5.615511000	127.0.0.1	127.0.0.1	HTTP	107	HTTP/1.1 200 Connection established
105	5.615560000	127.0.0.1	127.0.0.1	TCP	68	36386 > 52186 [ACK] Seq=208 Ack=40 Win=43776 Len=0 TSval=3345702 TSecr=3345702
106	5.615841000	127.0.0.1	127.0.0.1	TLSv1.2	250	Client Hello
107	5.615865000	127.0.0.1	127.0.0.1	TCP	68	52186 > 36386 [ACK] Seq=40 Ack=390 Win=45952 Len=0 TSval=3345702 TSecr=3345702
108	5.653618000	127.0.0.1	127.0.0.1	TLSv1.2	976	Server Hello, Certificate, Server Key Exchange, Server Hello Done

```
    Length: 20
  ▾ Server Name Indication extension
      Server Name list length: 18
      Server Name Type: host_name (0)
      Server Name length: 15
      Server Name: maps.google.com
```