Rapport Final
Audit des implantations SSL/TLS

Date	5 février 2014
Rédigé par	Claire Smets, William Boisseleau, Julien Legras, Mathieu Latimier, Pascal Edouard
À l'attention de	Ayoub Otmani

Table des matières

Table des figures

Liste des tableaux

Listings

Introduction

En Août 2012, l'Université du Michigan détecte une vulnérabilité critique sur les certificats RSA et DSA d'Internet [4]. Certains ont, en effet, des facteurs premiers communs avec d'autres certificats, et sont donc sujet à une attaque par factorisation. Une tel attaque permet de retrouver très rapidement les clés privées de ces certificats vulnérables.
Les chiffres sont de plus très élevés, sur 12.828.613 certificats :

- 714.243 utilisent des clés vulnérables dont 43.852 sont dues à une insuffisance d'entropie.
- **64.081 utilisent des clés RSA pouvant être factorisées**.
- **4.147 utilisent (encore) des clés prévisibles générées par Debian lors du bug de 2006**
- 123.038 utilisent (encore) des clés RSA de 512 bits.

Partant de cette étude, notre client souhaitait voir si la sécurité d'Internet avait été amélioré ces deux dernières années en marchant sur les traces de l'Université du Michigan. Puis d'analyser le code d'OpenSSL ainsi que sa sécurité au niveau des primitives cryptographiques. Enfin, il nous a été demandé d'évaluer le niveau de sécurité de nos navigateurs actuels.

Ce rapport présentera chacune des trois parties de notre projet dans des chapitres distincts, pour conclure sur un chapitre concernant la vie du projet, nos méthodes de travail, les outils et les ressources utilisés, les choix de développements et notre gestion des risques.

Dans la partie concernant l'audit de clefs cryptographiques nous détaillerons l'ensemble des algorithmes utilisés lors des phases de récupération, de factorisation et de traitement des certificats SSL.

Dans la partie concernant l'audit statique d'OpenSSL, nous feront un résumé du rapport d'audit en soulevant les points importants concernant le contexte d'utilisation des primitives cryptographiques, les normes visées, les failles ou vulnérabilités trouvées, et les recommandations.

Enfin, dans la partie concernant l'analyse dynamique, nous parlerons des objectifs fixés, nous expliquerons la mise en place du serveur, nous identifierons certaines faiblesses, pour finir sur l'implémentation en C d'une bibliothèque de gestion de socket TCP.

Remerciements

Nous tenons tout d'abord à remercier notre client et professeur M. Ayoub Otmani, pour nous avoir donner un sujet de projet passionnant et complet. Il nous a en effet permis de mieux nous placer dans un contexte de projet professionnel, et il a su nous donner de bons choix dans notre étude afin de récupérer des informations pertinentes.

Nous le remercions également ainsi que M. Olivier Quelquechose pour nous avoir donné l'accès au serveur local de l'Université afin de faire de récupérer plus rapidement nos résultats lors de la factorisation.

Nous remercions finalement l'ensemble du corps enseignant pour nous avoir donné la formation nécessaire afin d'atteindre cet objectif.

Chapitre 1

Audit de clefs cryptographiques

1.1 Ex section

1.1.1 Ex subsection

1.1.1.1 Ex subsubsection

1.1.1.1.1 Ex subsubsubsection

Exemple item :
- item1 ;
- item2 ;
- itemFin.

Exemple enum :

1. enum1 ;
2. enum2 ;
3. enumFin.

FIGURE 1.1 – Exemple de figure avec titre raccourci

Identifiant	**KeyExch**	**Authn**	**Enc**	**MAC**
`TLS_SRP_SHA_WITH_3DES_EDE_CBC_SHA`	SRP SHA1	SRP SHA1	3DES CBC	SHA1

TABLE 1.1 – Exemple tableau

Chapitre 2

Analyse Statique : audit d'OpenSSL

2.1 But de cette partie

Nous nous sommes concentrés sur les parties qui peuvent être critiques :

l'entropie : un des problèmes les plus épineux lors de la génération de clef ;

la génération des clefs : c'est sur elles que reposent une bonne partie de la sécurité. Si leur secret venait à être découvert, les fonctions de chiffrement ne servent plus à rien (il sont en théorie connus et éprouvés) ;

les chiffrements et leur protocole : régulièrement, des failles sont trouvées dans ces protocoles, et les avancées technologiques poussent aussi à rendre des algorithme obsolètes (augmentation de la puissance de calcul des machines) ;

les signatures et leurs authentifications : parce que les certificats et les signatures électroniques sont bien plus présents qu'on ne pourrait le croire. Le mécanisme est souvent transparent à l'utilisateur, mais les certificats sont devenus indispensables ;

les protocoles SSL et TLS : OpenSSL est basé directement sur le protocole SSL puis ensuite sur celui de TLS.

Cette deuxième partie est un très court résumé du rapport d'audit réalisé pendant le projet. Pour de plus amples informations, nous vous conseillons de le consulter.

2.2 Entropie

2.2.1 Introduction

Cette partie explicite les notions d'entropie nécessaires pour la définition d'aléatoire de certains programmes. Il décrit aussi en particulier les sources de génération de bits aléatoires et leurs tests liés.

Trois axes principaux sont nécessaires à la mise en place d'un générateur cryptographique aléatoire de bits :

- Une source de bits aléatoires (source d'entropie)
- Un algorithme pour accumuler ces bits reçus et les faire suivre vers l'application en nécessitant.
- Une méthode appropriée pour combiner ces deux premiers composants

2.2.2 Estimation de l'entropie générée par la source

Il est tout d'abord important de vérifier que la source d'entropie choisie produit suffisamment d'entropie, à un taux égalant voire dépassant une borne fixée. Pour ce faire, il faut définir avec précision la quantité d'entropie générée par la source. Il est de plus important de considérer les différents comportements des composants de la source, afin d'éliminer les interactions qu'ils peut y avoir entre les composants. En effet, ceci peut provoquer une redondance dans la génération d'entropie si cela n'est pas considéré. Étant donné une source biaisée, l'entropie générée sera conditionnée et donc plus facilement prévisible/estimable.

La source d'entropie doit donc être minutieusement choisie, sans qu'aucune interaction et conditionnement ne soit possible.

2.2.3 Concept d'entropie

Définition.
Soit X est une V.A. discrète. On définit l'**entropie** de X comme suit :

$$H(X) = -\sum_{x} P(X = x) * log_2(P(X = x))$$

Le logarithme est dans notre cas de base 2. L'entropie se mesure en shannons ou en bits.

Définition.
On définit le **désordre** (ou incertitude) étant liée à cette expérience aléatoire. Si l'on considère l'ensemble fini des issues possibles d'une expérience $\{v_1, ..., v_n\}$, l'entropie de l'expérience vaudra :

$$H(\epsilon) = -\sum_{x} P(\{a_i\}) * log_2(P(\{a_i\}))$$

Propriété.
On constate que l'entropie est maximale lorsque X est équi-répartie. En effet, si l'on considère n éléments de X étant équi-répartie, on retrouve notre entropie de $H(X) = log_2(n)$.

Ainsi, on comprend qu'une variable aléatoire apporte en moyenne un maximum d'entropie lorsqu'elle peut prendre chaque valeur avec une équiprobabilité. D'un point de vue moins théorique, on considère que plus l'entropie sera grande, plus il sera difficile de prévoir la valeur que l'on observe.

Min-entropy.
La recommandation du NIST propose le calcul de *Min-entropy* pour mesurer au pire des cas l'entropie d'une observation.

Soit x_i un bruit de la source d'entropie. Soit $p(x_i)$ la probabilité d'obtenir x_i. On définit l'entropie au pire des cas telle que :

$$\text{Min-entropy} = -log_2(max(p(x_i))$$

La probabilité d'observer x_i sera donc au minimum $\frac{1}{2^{\text{Min-entropy}}}$.

2.2.4 Source d'entropie

Approche théorique.

La source d'entropie est composée de 3 éléments principaux :

- le **bruit source**, qui est la voûte de la sécurité du système. Ce bruit doit être non déterministe, il renvoie de façon aléatoire des bits grâce à des processus non déterministes. Le bruit ne vient pas nécessairement directement d'éléments binaires. Si ce bruit est externe, il est alors converti en données binaires. La taille des données binaires générées est fixée, de telle sorte que la sortie du bruit source soit déterminée dans un espace fixe.
- le **composant de conditionnement**, qui permet d'augmenter ou diminuer le taux d'entropie reçu. L'algorithme de conditionnement doit être un algorithme cryptographique approuvé.
- une **batterie de tests**, partie également intégrante du système. Des tests sont réalisés pour déterminer l'état de santé du générateur aléatoire, permettant de s'assurer que la source d'entropie fonctionne comme attendu. On considère 3 catégories de tests :
  - Les tests au démarrage sur tous les composants de la source
  - Les tests lancés de façon continue sur le bruit généré par la source
  - Les tests sur demande (qui peuvent prendre du temps)

  L'objectif principal de ces tests est d'être capable d'identifier rapidement des échecs de génération d'entropie, ceci avec une forte probabilité. Il est donc important de déterminer une bonne stratégie de détermination d'échec pour chacun de ces tests.

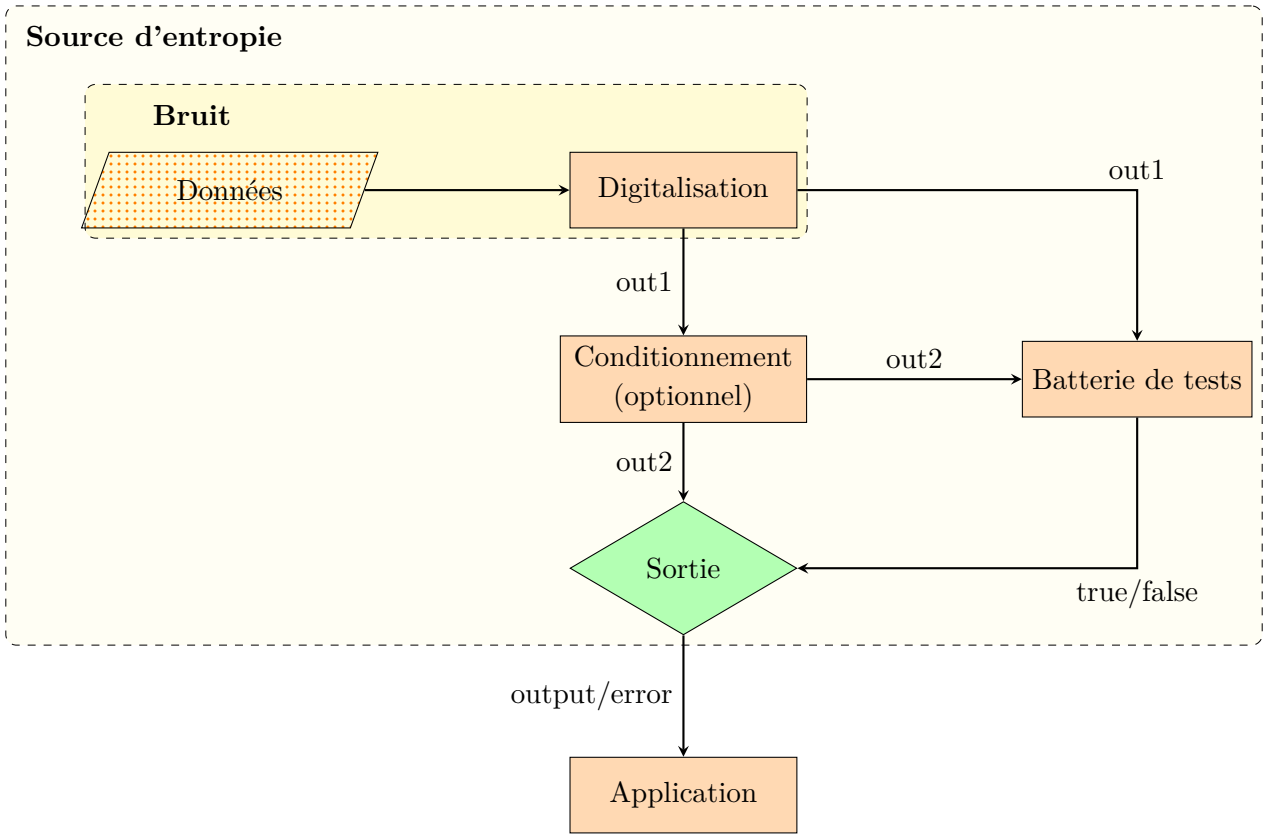

FIGURE 2.1 – Composants d'une source d'entropie. `out1` est une chaîne binaire de taille quelconque et `out2` est une chaîne binaire conditionnée de taille fixe.

2.2.5 Standards

2.2.5.1 RFC 4086

Nos machines utilisent ce qu'on appelle des PRNG (Pseudo Random Number Generator), qui sont des algorithmes qui génèrent une séquence de nombres s'apparentant à de l'aléatoire. En réalité rien n'est aléatoire car tout est déterminé par des valeurs initiales (état du PRNG) et des contextes d'utilisation.

Un bon PRNG se doit d'avoir une très forte entropie (proche de un), afin d'éviter de délivrer de l'information.
Comme l'entropie est fournie majoritairement (si ce n'est totalement) par l'OS, chacun présente ses forces et ses faiblesses. Dans le rapport d'audit sont détaillés les différentes méthodes de différents OS.

Nous nous basons sur la RFC 4086 [1] : *Randomness requirements for security* pour le choix des PRNG selon les différents systèmes.

2.2.5.2 FIPS 140

Le FIPS 140 (*Ferderal Information Processing Standards*) est un standard du gouvernement américain spécifique aux modules cryptographiques déployés par des éléments du gouvernement. Il inclue notamment des standards de tests qui permettent de décrire la qualité et/ou valider des générateurs d'entropie ou générateurs pseudo-aléatoires. La version la plus récente est le FIPS 140-2, qui décrit plusieurs niveaux de tests.

Les test du FIPS 140-1 permettent de s'assurer que les sources d'entropies produisent suffisamment de "bonnes" données, pourvu que les sources d'entropies n'utilisent pas quelques opérations cryptographiques internes. Si une source d'entropie en utilise, alors elle réussira les tests avec quasi-certitude, même si la source d'entropie est faible. De ce fait, ces jeux de tests ne sont pas bons pour tester des générateurs de nombres pseudo-aléatoires cryptographiques, car ceux-ci passeront facilement les tests même si les générateurs d'entropie sont faibles. Par exemple, si l'on hache une suite d'entiers (par pas de 1), les tests seront tous validés bien qu'ils n'auront pas été tirés aléatoirement, ceci en vertu de la fonction de hachage. Pourtant, la donnée est hautement prédictible.

Les tests du FIPS 140-2 ne sont également pas très efficaces, et permettent seulement de détecter si un matériel commence à produire un motif répété. Ils consistent à comparer différentes sorties consécutives d'un générateur. Ainsi, si le "générateur aléatoire" consiste à produire une simple incrémentation de nombres, les tests passeront sans problème.

Il est donc conseillé d'utiliser les tests du FIPS 140-1 et 140-2 pour vérifier uniquement si la source d'entropie produit de bonnes données, au démarrage et périodiquement lorsque c'est possible.

2.2.5.3 Quelques failles

2.2.5.3.1 Le cas Debian 4.0 et OpenSSL 0.9.8

C'est l'une des failles qui a fait grand bruit ces dernières années.
Le 13 mai 2008, Luciano Bello a découvert une faille critique du paquet d'OpenSSL sur les systèmes Debian. Un mainteneur Debian souhaitant corriger quelques bugs aurait malencontreusement supprimé une grosse source d'entropie lors de la génération des clefs. Il ne restait plus que le PID comme source d'entropie.

Comme celui-ci ne pouvait dépasser 32 768 (qui est le PID maximal par défaut atteignable), l'espace des clefs a été restreint à 264 148 clefs distinctes.

Il est donc fortement recommandé de mettre à jour sa version d'OpenSSL vers une version stable où le bug a été corrigé, puis de générer de nouvelles clefs de chiffrement, et de révoquer les certificats corrompus. Il reste malheureusement beaucoup de certificats touchés par la faille, la plupart été valides jusqu'en 2020-2030.

2.2.5.3.2 Le cas MinuxMintDebianEdition sous Android

Faille.

Récemment, en août 2013 précisément, un patch de sécurité pour les systèmes Android utilisant la version LinuxMintDebianEdition/OpenSSL, dévoile une réparation du générateur de nombres pseudo-aléatoires (PRNG) qui ne donnait pas suffisamment d'entropie.

Le patch indique que le PRNG de cette version d'OpenSSL utilise dorénavant une combinaison de données plus ou moins prévisibles associées à l'entropie générées par /dev/urandom. Mais sachant que le PRNG d'OpenSSL utilise lui-même /dev/urandom, on a du mal à comprendre pourquoi en rajouter davantage.

Eric Wong et Martin Boßlet apportent la solution sur leur site. Ils montrent que l'erreur provient d'un bug "à la Debian", une simple ligne diffère de la version officielle d'OpenSSL (utilisant SecureRandom) à celle de OpenSSL : Random se situant dans la fonction ssleay_rand_Toutefois. bytes, la conséquence n'est pas aussi lourde que celle de Debian, tout d'abord parce que le système Android est rarement utilisé pour du chiffrement de données sensibles, et une attaque par prédiction bien que plus rapide qu'une attaque par brute-force reste infaisable. Mais l'erreur est quand même là.

Recommandations.

Il est alors recommandé à court terme d'ajouter plus d'ntropie à notre PRNG sous Android, soit manuellement comme le script Ruby d'Eric Wong, ou avec des outils cités plus haut dans ce document.Sinon, le patch d'OpenSSL répare également l'erreur en rajoutant plus d'aléatoire, bien qu'une recommandation officielle serait l bienvenue, ainsi qu'une meilleure documentation.

2.2.5.3.3 NetBSD 6.0 et OpenSSH

Faille.

Autre faille du même genre sur les systèmes NetBSD 6.0 concernant OpenSSH/OpenSSL et datant de mars 2013. L'erreur provient d'une insuffisance d'entropie dans le PRNG, qui ne tient que sur 32 ou 64 bits (la taille d'un sizeof(int)). Un attaquant peut brute-forcer une clef générée par ce PRNG. Il est probable que les dégâts soient bien moins étendus que lors de l'affaire OpenSSL Debian car les systèmes NetBSD 6.0 sont moins fréquents.

Il est recommandé de passer à une version NetBSD supérieure à 5.1, et de générer de nouvelles données de chiffrement comme les clés SSH ayant étés générées avec ce noyau. Il faut également faire attention au patch de sécurité de janvier 2013 qui en tentant de régler le problème a généré une autre erreur produisant le même effet. L'erreur ne provient pas directement du code d'OpenSSL. Mais ici, OpenSSL se contente de

récupérer l'entropie fournie sans aucune vérification (fonction `RAND_bytes()`).

Recommandations.
Il est également recommandé d'utiliser `/dev/random` plutôt que `/dev/urandom` pour avoir une bonne entropie. On souligne quand même le fait que OpenSSL ne contrôle pas son entropie, si l'entropie du système est quasi-nulle les clefs sont tout de même générées. Un avertissement auprès de l'utilisateur serait un minimum.

De manière plus générale, le NIST propose des recommandations à plusieurs niveaux concernant la conception de la source d'entropie. Il est possible d'en trouver dans le rapport du NIST SP800-90B.

2.3 Génération des clefs

La génération de clefs est une notion fondamentale de cryptographie. En effet, les données sont protégées grâce à des algorithmes (ou des méthodes cryptographiques) et des clefs cryptographiques. Celles-ci sont notamment utilisées pour le chiffrement et le déchiffrement de données.
Relativement aux deux types de cryptographie, on compte deux types de clefs :

- les clefs symétriques pour le chiffrement symétrique ;
- les clefs publiques/privées pour le chiffrement asymétriques.

On se considérera dans ce chapitre dans un milieu optimal :

1. l'algorithme cryptographique considéré est optimal ;
2. la génération de bits aléatoire est optimale.

2.3.1 Clefs d'algorithmes symétriques

Les algorithmes symétriques utilisent pour leur chiffrement une clef symétrique qui est partagée par chacun des correspondants.

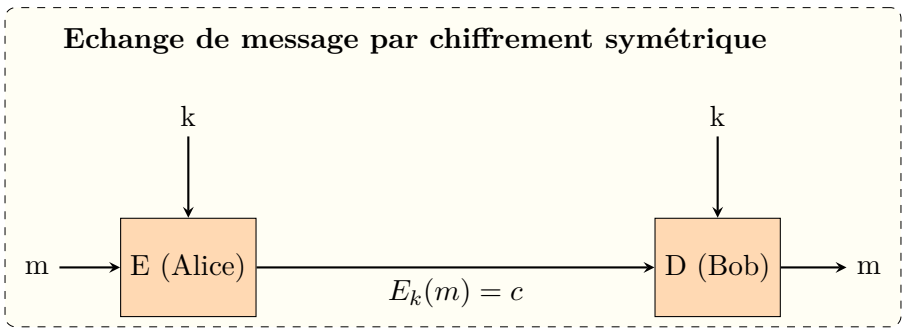

FIGURE 2.2 – Chiffrement symétrique - Alice envoie un message c chiffré à Bob qui le déchiffre

2.3.2 Clefs d'algorithmes asymétriques

Les clefs d'algorithmes symétriques sont de deux catégories, les clefs privées et les clefs publiques. Les clefs publiques sont disponibles pour tous, par demande à l'utilisateur, ou bien souvent par certificat.

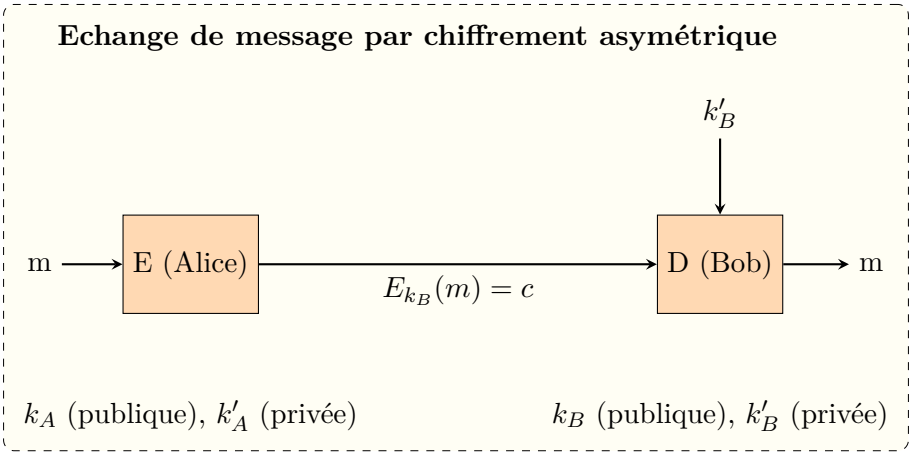

FIGURE 2.3 – Chiffrement asymétrique - Alice envoie un message c chiffré à Bob avec sa clef publique qui le déchiffre

Comme montré en figure 2.3, Alice chiffre avec la clef publique de Bob un message, qui lui-même déchiffre le message avec la clef privée.

2.3.3 Méthode de génération et sécurité

Les systèmes utilisent des entiers comme clef. Les clefs sont générées en utilisant des générateurs de nombres aléatoires (RNG) ou des générateurs de nombres pseudo aléatoires (PRNG).

2.3.3.1 Génération de clés quelconques

Un générateur de bits aléatoires (RGB) approuvé est utilisé pour générer la clef cryptographique. Le RGB doit fournir une entropie complète ou suffisante aux exigences de sécurité du système. Les RGB peuvent permettre de générer complètement et "directement" la clef. C'est le cas pour les clefs privées des algorithmes AES ou DSA. Les RGB peuvent aussi être utilisés comme *seed* pour générer des clefs, suivants des critères spécifiques. C'est le cas pour RSA, où habituellement le *seed* est utilisé comme point de départ pour trouver un nombre premier suivant les critères du FIPS 186-3.

2.3.3.2 Génération de paires de clefs asymétriques

Les algorithmes asymétriques demandent la création de paires de clefs asymétriques (privée/publique), dont chaque clef est associée à une seule entité, son possesseur. Les clefs sont généralement générées par leur possesseur direct, ou par une autorité de confiance qui fournit la clef au possesseur de façon sûre.
Il convient au moment de la génération de bien considérer deux paramètres principaux :

1. l'espace de clefs, qui doit être suffisamment grand pour éviter les doublons et les cycles ;
2. générer les premiers de façon la plus aléatoire possible. Ceci nécessite un bon choix de stratégie d'identification de premiers. On peut par exemple utiliser des algorithmes permettant de détecter le prochain premier suivant un entier tiré aléatoirement.

2.3.4 Exemples de faille

2.3.4.1 Le générateur Diffie-Hellman

Voici l'algorithme de génération de g, d'après la RFC 2631 datant de 1999 et dérivé de la FIPS-186 :
- 1- Soit $j = (p-1)/q$.
- 2- Choisir $h \in \mathbb{N}$, tel que $1 < h < p-1$
- 3- Calculer $g = h^j \mod p$
- 4- Si $g = 1$ recommencer l'étape 2

Mais depuis 2006, on peut lire comme recommandation dans la RFC 4419 (pour une utilisation SSH) : "*It is recommended to use 2 as generator, because it improves efficiency in multiplication performance. It is usable even when itis not a primitive root, as it still covers half of the space of possible residues.*"

Lorsque nous avons étudié le code de Diffie-Hellman dans OpenSSL, nous nous sommes penchés sur un choix plutôt étrange. La valeur du générateur est toujours fixé à 2 ou à 5.

Le générateur de Diffie-Hellman n'étant pas une racine primitive dans $\mathbb{Z}/\mathbb{Z}_p$, les conséquences sont :
- l'espace des clefs possibles est fortement réduit (Si $g = 2 \implies$ espace divisé par deux) ;
- deux clefs privées distinctes pourront avoir une clé publique commune ;
- la méthode de cryptanalyse Baby-step Giant-step peut s'en trouver facilité.

Évidemment, ce choix n'est pas une faille en soit, il n'est juste pas optimal et résulte d'un bon compromis entre vitesse et sécurité. Pour une sécurité optimale, il est conseillé de choisir un générateur qui soit une racine primitive, pour être certain que personne ne puisse signer, déchiffrer des messages à votre place !

Norme.

Ainsi selon la norme RFC-4419, le choix du générateur peut se résumer à un petit générateur qui ne serait pas une racine primitive. La criticité du risque est très grande, car si deux personnes possèdent la même clef publique pour deux clefs privées distinctes, ils pourront alors déchiffrer les messages, et signer à la place de l'autre. Mais, la probabilité d'une telle collision est quasiment nulle, il est simplement deux fois plus efficace de générer deux clés privées possédant la même clef publique, que de retrouver la clef privée en brute force. OpenSSL laisse en plus le choix au développeur de choisir une racine primitive comme générateur, pour des cas d'extrême sécurité.

2.3.4.2 Diffie-Hellman Ephémère en mode FIPS

Faille.

Une faille plus grave concerne le mode FIPS (Federal Information Processing Standard) d'OpenSSL, qui peut être compilé avec la commande `./config fipscanisterbuild`". En effet, un attaquant situé entre le client et le serveur connaissant la clef secrète du serveur peut déchiffrer une session SSL/TLS.

L'algorithme EDH/DHE (Diffie-Hellman Éphémère) permet de calculer une nouvelle clef connue uniquement du client et du serveur, donc l'attaquant intermédiaire ne peut plus déchiffrer la session. Cependant, en mode FIPS, OpenSSL ne rejette pas les paramètres P/Q faibles pour EDH/DHE. Lorsque OpenSSL est

compilé en mode FIPS, un attaquant en Man-in-the-middle peut donc forcer la génération d'un secret Diffie Hellman prédictible, en modifiant par exemple le traffic réseau..

La faille en elle même n'est pas suffisante pour faire l'attaque, elle requiert également une implémentation SSL faible.

Recommandations.

Il est tout d'abord recommandé de passer à la version OpenSSL supérieur, sinon de désactiver le mode FIPS, ou encore de configurer la *ciphersuite* afin de ne pas permettre au serveur d'utiliser DH comme algorithme d'échange de clefs.

2.4 Chiffrement et protocoles

2.4.1 Définitions et contexte

Nous allons voir qu'une combinaison entre un mode de chiffrement et un protocole de chiffrement peut générer des failles, souvent grave (i.e vols de mots de passes), que le padding est également une donnée sensible et qu'il faut donc le choisir intelligemment, et qu'une mauvaise documentation ou un mauvais paramétrage peuvent entraîner des failles au niveau protocolaire.

2.4.1.1 Les "Manger's attack" sur RSA-OAEP

OAEP : *Optimal Asymmetric Encryption Padding.*

Dans les chiffrements par blocs, cela nécessite généralement que tous les blocs soient d'une taille précise. Or ce n'est pas toujours le cas. Pour cela, on rajoute des bits de bourrage (padding).
OAEP est un schéma de remplissage, généralement utilisé avec RSA (en prétraitement). Il a été introduit en 1994 par Mihir Bellare et Phil Rogaway1. L'OAEP est une forme de réseau de Feistel qui nécessite une source d'aléa ainsi que deux fonctions de hachage.
RSA-OEAP peut être prouvé sûr dans un modèle théorique idéalisé, celui de l'oracle aléatoire. Il est recommandé par les PKCS.
OAEP a deux buts :

- insérer un élément d'aléatoire qui permet de passer d'un schéma déterministe à un schéma non déterministe (le même message clair chiffré deux fois avec la même clef et le même algorithme n'aura pas le même message chiffré.) ;
- prévenir un déchiffrement partiel en s'assurant que l'attaquant ne peut retrouver une portion du texte clair sans être capable d'inverser la fonction trapdoor (par exemple la factorisation de deux grands nombres premiers : il est facile de multiplier, mais quand on n'a que le produit il est très difficile de retrouver les facteurs).

Il n'est pas prouvé sûr pour une attaque IND-CCA (attaque à texte chiffré seulement). Victor Shoup a démontré qu'il n'existe pas de preuve générale. Il a montré que dans un cas IND-CCA, quelqu'un qui sait comme inverser partiellement une primitive d'insertion mais ne sait pas comment l'inverser complètement, pourrait bien être en mesure de casser le système. Par exemple, on peut imaginer quelqu'un qui peut attaquer RSAES-OAEP si on sait comment retrouver tous les octets exceptés les 20 premiers d'un entier généré aléatoirement chiffré avec RSAEP. Un tel attaquant n'a pas besoin d'être capable d'inverser entiè-

rement RSAEP (RSA Encryption Protocole), parce qu'il n'utilise pas les 20 premiers octets dans son attaque.

Norme.

Note de la PCKS# 1 (RFC 3447) : Il faut faire attention qu'un adversaire ne puisse distinguer les différentes erreurs renvoyées, que ce soit par un message d'erreur, ou un temps de réponse différent, ou, plus généralement, apprendre une information partielle à propos du message en clair `EM`. Sinon un adversaire peut être en mesure d'obtenir des informations utiles sur le déchiffrement du texte chiffré `C`, conduisant à une attaque à chiffré choisi telle que celle observée par Manger.

Description de la faille.

RSA-OAEP peut être soumis à une attaque nommée "Mangers Attack" selon son implantation. OpenSSL semble être vulnérable à une attaque de ce type, à base de "prédictions" par injections de fautes. La vulnérabilité semble être très récente puisqu'elle fonctionne sous OpenSSL 1.0.0.

Correction.

Le padding OAEP devait palier le problème d'insécurité que causait le padding PKCS#1 v1.5 (attaque à chiffré choisi). OpenSSL a tout de même pris en compte cette vulnérabilité et a placé des contres-mesures efficaces. La Technische Universität Darmstadt (Allemagne) explique en détails comment sont implémentées ces contres-mesures et montre que dans certains cas l'attaque reste possible. Enfin, elle apporte ses propres contre-mesures.

On peut noter que plusieurs librairies sont vulnérables à une attaque de Manger (voir paragraphe suivant) qui consiste à contrôler la taille des paramètres à hacher, mais que l'implantation de RSA-OAEP d'OpenSSL ne le permet pas. La raison est que le décodage OAEP est linéaire quelque soit la taille des paramètres et les erreurs survenues. Il semble également y avoir un problème avec l'OAEP_padding sur le chiffrement RSA. Bill Nickless recommande l'utilisation de PKCS_padding.

Recommandation.

Il n'y a pas vraiment de quoi s'alarmer, cette attaque est en pratique infaisable sur un serveur car il y a suffisamment de variations de délais (différences de CPU, opérations multi-tâches, connexions réseaux, etc...) pour éviter une attaque par timing. Cependant, sur des systèmes embarqués l'attaque peut être réalisable, et il serait plus prudent de palier ce problème.

2.4.1.2 Chiffrement SSLv3 ou TLS 1.0 en mode CBC

Faille.

En Septembre 2011, une attaque en *man in the middle* très efficace a vu le jour contre les protocoles SSLv3 et TLS 1.0. . L'attaque est à clair choisi. Le but étant d'insérer des morceaux de texte clair grâce au navigateur dans la requête chiffrée avec ces protocoles, ceci afin de récupérer les cookies de session.

La technique est basique, un individu enregistre plusieurs cookies de session auprès de divers sites officiels (banques, messageries, etc...). Puis, il clique malencontreusement sur du code Java malveillant (publicité,

image, etc...). et l'attaque se déroule automatiquement. L'ensemble des cookies est envoyé au serveur malveillant qui n'a plus qu'à déchiffrer les clés de session.

La cause viendrait du mode de chiffrement choisi : CBC. SSL/TLS est un protocole qui chiffre un canal de communication. De ce fait il ne chiffre pas un fichier unique, mais une série d'enregistrements. Il y a deux façons d'utiliser le mode CBC dans ce cas précis :

- prendre chacun de ces enregistrements indépendamment des autres. Générer un nouveau vecteur d'initialisation à chaque fois ;
- traiter ces enregistrements comme un seul objet en les concaténant. Le vecteur d'initialisation est donc choisi aléatoirement pour le premier enregistrement et pour les autres, il aura pour valeur le dernier bloc de l'enregistrement précédent.

Recommandations.

SSLv3 et TLS 1.0 utilisent ce deuxième choix, cela soulève un lourd problème de sécurité. En 2004, Moeller trouve une méthode pour exploiter ce mauvais choix afin de récupérer des morceaux de textes clairs. Il y a certes une faille immense, mais peu exploitable. Les grandes entreprises savent (normalement) qu'il ne faut pas utiliser le mode CBC pour du chiffrement SSL/TLS. Et, dans tous les cas, plusieurs navigateurs ne permettent pas ce type d'attaque (c'est le cas de Chrome par exemple).

La faille existe tant que l'association de ces protocoles avec le mode de chiffrement CBC existe. Même si l'attaque est infaisable sur les navigateurs les plus répandus (Chrome, Firefox, IE, Safari, ...), OpenSSL devrait pouvoir interdire cette association, et ne pas laisser le travail aux navigateurs. Mais rien n'empêche l'utilisation de ce chiffrement par un navigateur plus léger, nous pourrons tester cette vulnérabilité lors de notre partie 3 si nous trouvons un navigateur acceptant ce type de chiffrement.

2.4.1.3 Non-validation des certificats SSL

Norme.

La RFC 5246 stipule que le serveur doit toujours envoyer un message certificat dès lors que la méthode d'échange des clefs a été acceptée. Le certificat DOIT être approprié à la suite des chiffrements utilisés pour les échanges de clefs.

Faille.

Six chercheurs des universités de Stanford et d'Austin au Texas, analysent une attaque en Man in the Middle autour des certificats SSL sans utilisation d'un navigateur. Le titre est sans appel "Le code le plus dangereux du monde".

SSL doit permettre d'être sécurisé en toute circonstance, que le cache DNS soit empoisonné, que les attaquants contrôlent les points d'accès et les routeurs, etc. . Il assure théoriquement trois grands principes de la cryptologie : la confidentialité, l'intégrité et l'authentification. Nous connaissons certaines failles au niveau du navigateur et de l'implantation SSL (voir ci-dessus). Mais il existe également d'autres cas d'utilisation du protocole SSL. Par exemple :

- Administration à distance basé sur le cloud, stockage sécurisé sur le cloud en local.
- Transmissions de données sensibles (ex : e-commerce)

– Services en ligne comme les messageries électroniques
– Authentification via applications mobiles comme Android et iOS

L'étude montre que la validation des certificats SSL est casée sur plusieurs applications et librairies dont :
– OpenSSL
– JSSE
– CryptoAPI
– NSS
– GnuTLS
– etc...

En fait, un attaquant en *Man In The Middle* peut intercepter le secret entre un client et un serveur utilisant une connexion SSL. Il peut ainsi récupérer des numéros de carte bancaire, avoir accès à une messagerie, récupérer des mots de passes, etc... La cause principale vient du fait que les développeurs retouchent les librairies cryptographiques à leur façon. En voulant réparer un bug ou en souhaitant rendre SSL compatible avec leurs API, ils injectent de nouvelles vulnérabilités. De plus, l'application est souvent propriétaire et payante ce qui rend le déboggage difficile.

Que ce soit accidentel ou intentionnel, l'une des conséquences les plus graves est la non-validation de certificat sur des contexte où la sécurité est primordiale (e.g. payement en ligne). La faute ne revient pas directement au code d'OpenSSL, mais à une mauvaise utilisation des différentes fonctions et options.

2.4.1.3.1 Difficultés du code OpenSSL

OpenSSL ne déroge pas à la règle.
Voici quelques vulnérabilités du code :
– Les contraintes de nom x509 ne sont pas correctement validés.
– Les applications DOIVENT fournir elles même leur code de vérification de nom d'hôte. Or, des protocoles comme HTTPS, LDAP ont chacun leurs propres notions de validations. Ainsi, Apache Libcloud utilise les librairie Python eux-même utilisant des commandes OpenSSL. Et sa méthode de vérification du nom d'hôte comporte des vulnérabilités pouvant causer des attaques en man in the middle (e.g "google.com" et "oogle.com" vérifie la même expression régulière)
– Un programme utilisant OpenSSL peut exécuter la fonction `SSL_connect` pour le handshake SSL. Bien que certaines erreurs de validation soient signalées par `SSL_connect`, d'autres ne peuvent être vérifier qu'en appelant la fonction `SSL_get_verify_result`, alors que `SSL_connect` se contente de retourner "OK".

2.4.1.3.2 Exemple : Trillian

Trillian est une messagerie cliente instantanée reliée à OpenSSL pour la sécurisation de l'établissement de connexion. Par défaut OpenSSL ne soulève pas d'exception en cas de certificat auto-signé ou de non-confiance auprès de la chaîne de vérification. A la place, il envoie un drapeau. De plus, il ne vérifie jamais le nom d'hôte. Si l'application appelle la fonction `SSL_CTX_set` pour initialiser le drapeau `SSL_VERIFY_PEER`, alors `SSL_connect` se ferme et affiche un message d'erreur lorsque le certificat n'est pas valide. Mais Trillian n'initialise jamais ce drapeau. Par conséquent, `SSL_connect` va retourner 1 et le statut de la validation

du certificat peut être connu en appelant la fonction `SSL_get_verify_result`. Encore une fois, Trillian n'appelle pas cette fonction. Les conséquences sont très lourdes : vols de mots de passes, compromissions de services, révélations des paramètres de sécurité, etc...

L'étude montre que l'attaque est possible sur la version 5.1.0.19 et antérieure de Trillian.

2.4.2 Conclusion

Les chercheurs nous donnent alors plusieurs leçons à retenir, dont voici quelques points :
- Premièrement, les vulnérabilités doivent être trouvées et réparées lors des phases de tests. Certaines se trouvent très facilement si les procédures de tests sont bien réalisées.
- Deuxièmement, la plupart des librairies SSL ne sont pas **sûres par défaut**, laissant le choix de la sécurité aux applications de plus haut niveau avec choix des options, choix de la vérification de l'hôte, choix d'interprétation des résultats.
- Troisièmement, même les librairies SSL sûrs par défaut peuvent être mal utilisées par des développeurs changeant les paramètres par défaut par des paramètres non sécurisés. La cause peut venir d'une **mauvaise documentation** ou d'une mauvaise formalisation de la part de l'API. Les API devraient entre autre proposer des abstractions de haut niveau pour les développeurs comme des tunnels d'authentification, plutôt que de les laisser traiter des détails de bas niveau comme la vérification du nom d'hôte.

Nous conseillons surtout une meilleure documentation d'OpenSSL, et des rapports d'erreurs d'interfaces plus simples et plus consistants afin d'éviter les erreurs d'interprétation. L'idée des chercheurs de proposer des abstractions de haut niveau pour les applications semblent être une très bonne idée.

2.5 Signature et authentification

2.5.1 Définitions et contexte

Une signature électronique utilise le concept de la traditionnelle signature sur papier et la tourne en une empreinte électronique. Cette empreinte est un message encodé et est unique pour chaque document et chaque signataire. La signature permet alors de garantir l'authenticité du signataire pour son document. Toute modification dans le document après l'avoir signé rend la signature invalide, ce qui protège alors contre les fausses informations et la contrefaçon des signatures.

De ce fait, il est important de faire attention à toute les formes de vulnérabilités des signatures afin de comprendre les attaques possibles sur la contrefaçon des signatures. Cette partie est consacrée à la bonne compréhension et l'implémentation qui sont définies dans la RFC, afin de se prémunir des différentes attaques possibles. On verra, cependant, qu'il existe quand même des failles au niveau des injections, surtout lorsque la librairie dépend trop du matériel.

2.5.2 Attaque par injection de fautes sur les certificats RSA

Dans la RFC 3447, la signature est décrite telle une primitive de signature qui produit la représentation de la signature depuis un message sous le contrôle d'une clef privée. La vérification se fait alors en récupérant la représentation du message depuis la représentation de la signature sous le contrôle de la clef publique

correspondante.

La signature se déroule en deux opérations qui sont la génération et la vérification. L'opération de génération consiste donc à générer une signature depuis un message avec la clef privée de l'utilisateur (signataire) et l'opération de vérification consiste à vérifier la signature en se basant sur le message en utilisant la clef publique du signataire. Ce schéma peut être utilisé dans de multiples applications telles que les certificats X.509.

L'Université du Michigan a réussi l'exploit de récupérer la clé privée d'un certificat RSA en un peu plus de 100h. L'attaque fonctionne par injection de fautes sur la méthode d'authentification. La technique est donc très poussée, mais le résultat en vaut la chandelle. L'injection de faute doit se faire sur quelques bits pour ne pas faire dysfonctionner le système tout entier. Les signatures erronées produites révéleront de l'information sur la clé privée. Avec le bon matériel et 100h d'attente, la clef peut être reforgée.

La technique consite à renseigner de fausses signatures afin de vérifier les fautes avec la clef publique de la machine.

Lorsque le système est vulnérable, OpenSSL ne le détecte pas forcément. Le risque est donc très fort, et les contre-mesures sont parfois difficiles à trouver dans les phases de tests. Toutefois, cette étude soulève un choix de programmation qui semble à première vue anodin, mais qui peut avoir de lourdes conséquences.

Toutefois, il faut pouvoir contrôler la machine (en ayant un accès au BIOS par exemple) pour pouvoir exploiter cette faille car il faut pouvoir toucher directement à l'alimentation de la faille.

Cependant, cette erreur n'est pas à prendre à la légère, car une attaque à base de faiseaux lumineux est en cours de développement afin de réaliser cette attaque à distance.

2.5.3 Malformation des signatures DSA/ECDSA

La RFC R979 définit l'utilisation de DSA (*Digital Signature Algorithm*) et ECDSA (*Elliptic Curve Digital Signature Algorithm*) de façon déterministe. DSA et ECDSA sont deux standards de signature électronique qui offrent l'intégrité et authenticité dans de nombreux protocoles.

En 2008, une vulnérabilité sur la malformation des signatures survient sur OpenSSL (re-analysé en Novembre 2012).

La cause vient de plusieurs fonctions implémentant la fonction EVP_VerifyFinal() (cf.). Elles valident de fausses signatures au lieu de retourner des erreurs, parmis les signatures corrompues peuvent se trouver :

- des signatures DSA ;
- des signatures ECDSA.

En 2009, un cas similaire a été trouvé dans un autre protocole (NTP) avec la même fonction `EVP_VerifyFinal`.

La conséquence est très grave, car cette faille permet une attaque en *man in the middle*, en faisant par exemple une attaque par *phishing* en HTTPS où la validation de la chaîne des certificats serait valide.

Ici, la faille persistera tant que le serveur et le client resteront à une version antérieur à OpenSSL 0.9.8j, les clefs quant à elles ne sont pas vulnérables, et peuvent être conservées. Malheureusement, le nombre de

serveurs tournant sous OpenSSL 0.9.8 et versions antérieures est très élevé.

Il est également recommandé aux développeurs utilisant OpenSSL de faire des audits réguliers de la fonction EVP_VerifyFinal() pour s'assurer que les vérifications sont bonnes. Les tests étants assez simples à effectuer.

Chapitre 3

Analyse dynamique

3.1 Objectifs

L'objectif de ce projet consiste à étudier le comportement d'une machine cliente lorsqu'elle se connecte sur un serveur HTTPS. Bien qu'à la base considérée comme optionnelle, nous avons décidé de développer cette partie du projet car nous avions de l'avance sur les parties précédentes et que l'analyse du code d'OpenSSL nous a donné des idées sur l'implémentation de cet outil.

Nous avons développé un serveur HTTPS permettant d'analyser les différentes *ciphersuite* utilisées à l'établissement de connexion d'un client à un serveur, et d'établir un diagnostique sur la qualité des suites utilisées par le client. Pour rappel, une *ciphersuite* est une combinaison de noms pour l'échange des clefs, l'authentification, le chiffrement, et les algorithmes MAC utilisés pour négocier les paramètres de sécurité pour une connexion réseau par TLS (*Transport Layer Security*)/ SSL (*Secure Sockets Layer*). La structure de ces suites est définie dans la RFC 5246 [2] et fut également décrite dans le rapport d'audit d'OpenSSL (Chapitre 5, Protocoles SSL/TLS).

La figure 3.1 illustre le système à mettre en place.

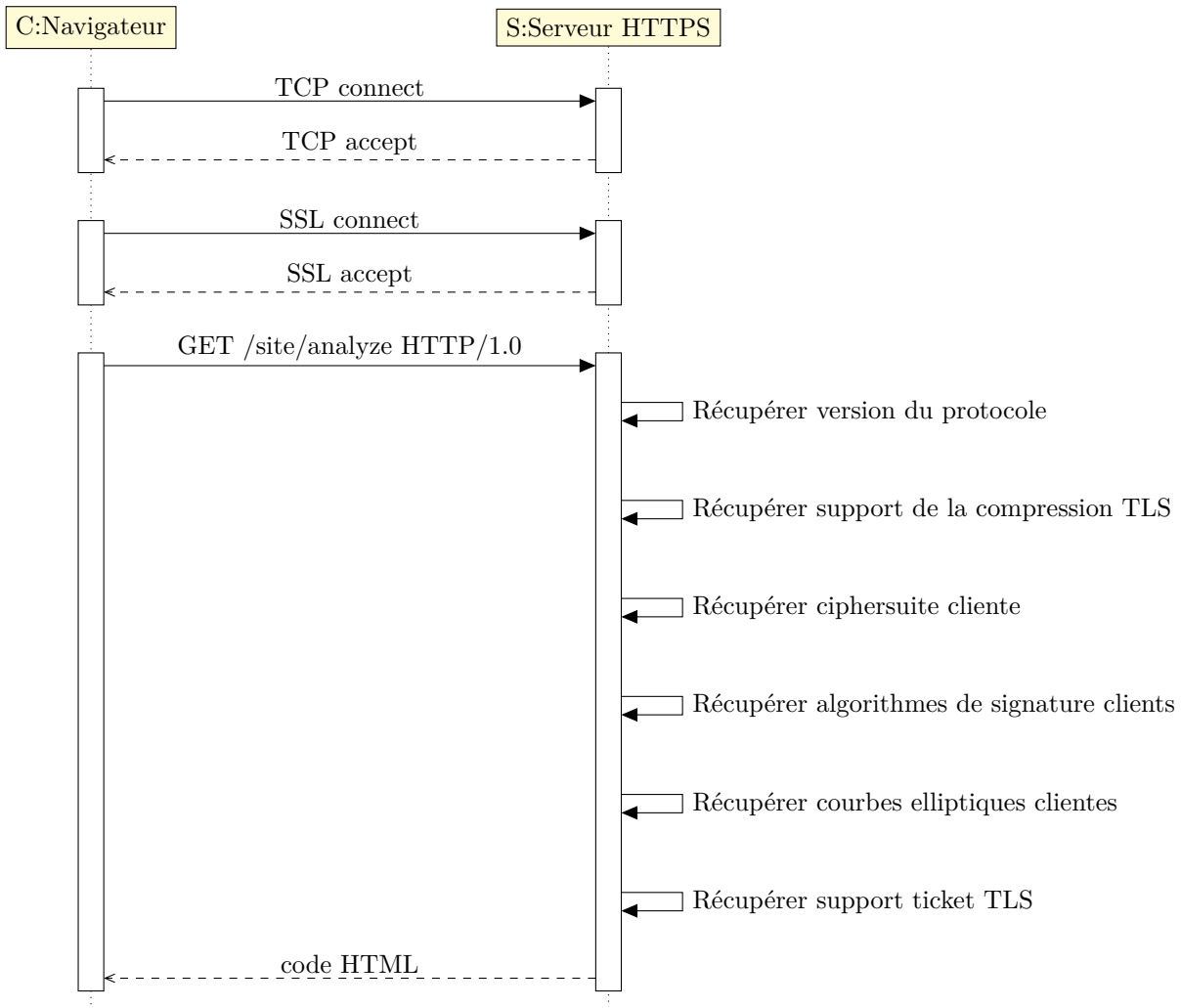

FIGURE 3.1 – Schéma de l'analyse dynamique du navigateur client

3.2 Mise en place du serveur

3.2.1 Exigences

Il convient de pouvoir identifier les caractéristiques de la connexion entre le client et le serveur à savoir :
- la version du protocole mise en œuvre ;
- les *ciphersuites* proposées par le client ;
- les courbes elliptiques supportées ;
- les algorithmes de signature ;
- la compression TLS, qui permet de compresser les données chiffrées afin d'accélérer le transfert de celles-ci ;
- l'activation ou non du ticket de session.

Suivant ces caractéristiques, nous pouvons émettre un jugement sur la qualité du client SSL/TLS utilisé. Si un client possède des attributs étant considérés comme vulnérables, alors il faut pouvoir les identifier rapidement sur le site, et expliciter les raisons de l'affaiblissement du système.

3.2.2 Faiblesses à identifier

3.2.2.1 Version du protocole

Si le client utilise une version `TLS 1.0` ou `SSLv3`, alors celui-ci sera considéré comme non sécurisé.

1. La version `SSLv3` précède la version `TLS 1.0` ; elle est considérée comme non sûre de nos jours.
2. La version `TLS 1.0` est la toute première version de TLS, et demande une pré-configuration client et serveur importante pour être utilisé de façon sécurisée. De plus, cette version ne permet pas d'utiliser les dernières *ciphersuites* récentes offrant une meilleure sécurité. De plus, cette version est vulnérable à l'attaque BEAST (*Browser Exploit Against SSL/TLS*).
3. Enfin, la version `TLS 1.1` n'étant pas la dernière version TLS, nous considérons qu'elle est moyennement sûre.

3.2.2.2 Ciphersuites

Nous avons pu identifier toutes les *ciphersuites* étant considérées comme peu sûres, à savoir :
- les *ciphersuites* utilisant des clefs de taille inférieure à 128 bits pour le chiffrement ;
- les *ciphersuites* ne spécifiant aucun chiffrement pour la connexion ;
- les *ciphersuites* sans authentification serveur, rendant l'attaque d'homme par le milieu possible ;
- autres *ciphersuites* étant supposées arrêtées après SSL 3.0, ou dont les caractéristiques de sûreté ne sont pas connues.

3.2.2.3 Courbes elliptiques

La RFC 5430 **??** décrit deux niveaux de sécurité pour les courbes elliptiques, les clefs de 128 bits et le clesf de 192 bits, référencés sur le tableau 3.1.

Ciphersuite	**Niveau de sécurité**
`TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256`	128
`TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256`	128
`TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA`	128
`TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384`	192
`TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384`	192
`TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA`	192

TABLE 3.1 – Niveaux de sécurité pour les *ciphersuites*

La RFC recommande l'utilisation de deux courbes :
- pour un niveau de sécurité à 128 bits, la courbe `secp256r1` ;
- pour un niveau de sécurité à 192 bits, la courbe `secp384r1`.

3.2.2.4 Compression des données chiffrées

La compression des données chiffrées permet d'accélérer le transfert des données. Toutefois, cette technique permet à un attaquant d'effectuer une attaque de type CRIME (*Compression Ratio Info-leak Made Easy*), qui permet de récupérer des données de session.

3.2.2.5 Ticket de session

Les tickets de session permettent d'accélérer la reprise de communication chiffrée avec le serveur. Le temps de la poignée de main peut prendre du temps et ce ticket permet d'établir une session entre le client et le serveur. Lorsqu'un client se déconnecte puis se reconnecte avec un ticket de session, les échanges du *handshake* sont passés, gagnant un temps considérable.

3.3 Implémentation

Pour la partie TCP, nous avons utilisé une bibliothèque écrite en Licence 3 et simplifiée pour notre utilisation : SocketTCP.

Le serveur HTTPS a été développé en C et il implémente les méthodes suivantes :

- `get_version (SSL *ssl)`, qui retourne la version de connexion utilisée ;
- `get_ecc_list (SSL *ssl)`, qui retourne la liste des courbes elliptiques ;
- `get_cipher_suite_string (SSL_CIPHER *c)`, qui retourne le texte associé à une *ciphersuite* ;
- `get_cipher_suite_list (SSL *ssl)`, qui permet l'affichage de toutes les *ciphersuites* ;
- `get_sig_algs (SSL *ssl)`, qui renvoie les algorithmes de signature ;
- `get_analyze_page (SSL *ssl)` qui génère la réponse complète (en-tête et corps HTTP) ;
- `handle_connection (void *param)` qui gère la connexion entre le serveur et le client : gestion des ressources statiques (CSS, JS) et de l'appel à l'analyse dynamique (`get_analyze_page`) ;
- `new_thread (SocketTCP *socket)` qui gère le lancement des threads pour chaque connexion.

Pour développer un tel serveur, il a simplement fallu s'appuyer sur un serveur HTTP basique puis rajouter la surcouche SSL dans `handle_connection` avec mise en place d'un contexte SSL et lecture/écriture chiffrée.

Pour le rendu de la page, nous sommes restés consistant avec le site développé lors du sprint 2 et avons donc utilisé le bootstrapt twitter. De plus, il nous permet d'indiquer les niveaux de sécurité avec des badges colorés qui permettent d'identifier visuellement et rapidement les problèmes.

Chapitre 4

Vie de projet

4.1 Présentation de l'équipe

Notre équipe est formée de cinq membres, tous étudiants à l'université de Rouen, dont un est également étudiant à l'INSA. La définition des rôles s'est faite assez rapidement, nous souhaitions changer de rôle par rapport au projet de l'année dernière, afin de mieux explorer la gestion de projet. Nous avons donc désigné **Pascal Edouard** comme chef de projet, **Julien Legras** comme Responsable technique, **Claire Smets** comme responsable client, et deux testeurs **William Boisseleau** et **Mathieu Latimier**. Le choix des deux testeurs vient du grand nombre de tests sur notre projet. Les résultats devant être sans faille.

Bien que chacun ait un rôle attitré, tous contribuent à différentes tâches du projet (tâches de développement, d'études, de tests, d'interfaces, etc...).

4.2 Le client

Notre client pour ce projet est : **M. Ayoub Otmani**.
Le cahier des charges se divise en trois grandes parties, pour la première, le client attend de nous un audit de clés cryptographique solide, présentable, et intuitif. Pour cela, il nous a conseillé d'étudier un article [4] publié par des universitaires de Californie et du Michigan, ainsi que le site factorable.net [5].

Pour la seconde partie, nous devions faire un audit du code source d'OpenSSL par rapport aux normes prescrites (ex : RFC, PKCS, NIST, ...) Le client souhaite notamment que l'on étudie la génération d'aléa, la génération des clefs, le chiffrement et les protocoles.

Enfin, dans la dernière partie, le client nous propose d'établir un diagnostic de connexion entre un client et un serveur sur plusieurs critères comme la génération des nonces, la génération de la clé primaire, le contrôle des certificats, le respect du protocole, le choix de l'algorithme etc...

Cette dernière partie a longtemps été optionnelle, elle a cependant pu être effectuée car le temps nous le permettait sur plusieurs thèmes (que nous avons développé plus haut dans ce rapport).

Le projet c'est déroulé sur six semaines, le temps de travail était de 8h/jour (pour pouvoir pallier le temps perdu par chacun pour la recherche de stage, les charges administratives, etc...)

4.3 Méthode agile : SCRUM

Nous avons choisi d'utiliser la méthode de développement agile SCRUM (comme indiqué dans le Plan de développement) afin d'apporter une discipline de développement et de délivrer les résultats dans les meilleures conditions.
Dans ses sous-parties nous allons revenir sur le choix de la méthodes, puis nous allons vous détailler son application tout au long du projet.

4.3.1 Pourquoi Scrum ?

Les méthodes agiles ont fait leurs preuves dans la gestion de projet, de par leurs grandes efficacité, et les valeurs qu'elles incarnent. De plus, SCRUM permet un suivi et une transparence totale avec le client. Le découpage en sprint est adapté à notre projet puisqu'il se déroule en trois parties et sur une période relativement courte.

4.3.2 Valeurs et principes

Les individus et leurs interactions plus que les processus et les outils : La méthodologie SCRUM correspond à la communication entre les collaborateurs à tous les niveaux (client/fournisseurs, testeurs/-programmeurs, ...) afin de ne pas perdre de temps ni d'énergie avec des malentendus ou de l'incompréhension.

Cette valeur a été primordiale pour nous, la force de notre projet réside dans cette bonne communication entre chacun des membres. Nous avons ainsi pu :

- détecter rapidement les problèmes grâce à de bonnes phases de tests, des réunions techniques régulières et une bonne communication.
- rentrer efficacement dans les sujets les plus importants de projet, surtout pour la partie de l'audit de code OpenSSL.
- exposer efficacement nos travaux au client, les possibilités qui s'offre à nous pour mieux satisfaire ses besoins.
- nous mettre tous à niveau sur chaque partie en résumant le travail de chacun lors de chaque réunion.

La collaboration avec les clients plus que la négociation contractuelle : Une approche directe avec le client qui se sent beaucoup plus impliqué dans le projet afin qu'il puisse apporter ses avis et remarques.

Nous sommes parti du principe que le client faisait partie intégrante de l'équipe. Lors de nos réunions, nous lui montrons un aperçu de nos travaux afin qu'il puisse s'assurer du bon avancement du projet, et qu'il puisse par la même occasion nous aiguiller pour la suite.

De plus, nous avons évité au maximum de faire des livraisons ou des demandes au client par messagerie électronique, préférant des rencontres interpersonnelles.

L'adaptation au changement plus que le suivi d'un plan : Être capable de s'adapter lorsqu'une modification importante est nécessaire.

Nous sommes parti d'un but général fixé sans détailler les étapes de chaque tâche afin de laisser libre cours au changement de contexte, au risques pouvant être rencontrés, aux sujets que le client souhaitait plus

approfondir, etc...
Les livraisons correspondent donc aux besoins du client, et sont modulaires afin d'approfondir l'étude.

4.3.3 Réunions

4.3.3.1 Brainstorming et Stand-Up Meeting

Chaque matin tout le monde se réunissait autour d'un café, et partageait son ressenti sur le projet et les tâches à venir. Ici, rien de technique, nous contrôlons juste la bonne avancée de chacun, et la compréhension générale du projet.
C'est également le bon moment pour définir les dates des prochaines réunions techniques.
Ces réunions s'apparentent aux "Mélées quotidiennes" du SCRUM.

4.3.3.2 Réunions hebdomadaires

Nous nous réunissions deux fois par semaine, le jeudi. La première en début d'après-midi ou en fin de matinée, nous passions dans une salle au calme, pour parler des difficultés techniques rencontrées, des axes d'améliorations, de la réorganisation ou du découpage des tâches.
Nous évoquions également les points importants à apporter au client pour la réunion suivante.
Ces réunions s'apparentent aux "Planification du Sprint" et à la "Rétrospective de Sprint" du SCRUM. La deuxième avait lieu dans l'après-midi, selon la disponibilité du client. Nous nous réunissions avec lui pour parler de notre avancée, relever ses remarques et ses nouvelles recommandations, présenter nos résultats ou livrer une partie du projet.
Ces réunions s'apparentent aux "Revue de Sprint" du SCRUM.

4.3.3.3 Réunions d'urgences

Plus rarement, il pouvait nous arriver d'avoir des réunions d'urgences. Nous nous sommes ainsi réuni avec le client lorsque nous étions bloqués sur la récupération des certificats SSH, ou lorsque l'on c'est aperçu qu'une amélioration majeure était possible (factorisation en full-RAM par exemple).

4.3.3.4 Audit

Les audit avec **M. Abdellah Godard**, nous ont permis d'avoir des remarques pertinentes sur nos documents livrables, et de progresser dans notre méthodologie de gestion de projet. Nous avons ainsi pu améliorer nos outils de gestion de projet. Par exemple, en adoptant GantterProject pour la gestion du planning, afin de mieux visualiser les tenants et les aboutissants d'une tâche, et perfectionner nos documents livrables (notamment au niveau de la traçabilité).

4.4 Outils pour la gestion de projet

Durant ces six semaines de travail, nous avons eu l'occasion de tester plusieurs logiciels outils pour notre projet.

4.4.1 Git

Git est un logiciel libre de gestion de versions décentralisé, créé par Linus Torvalds en 2005, accessible sur les systèmes Linux et Windows.

Un logiciel de gestion de versions est un logiciel qui permet de stocker un ensemble de fichiers en conservant la chronologie de toutes les modifications qui ont été effectuées dessus. Il permet notamment de retrouver les différentes versions d'un lot de fichiers connexes [13].
Notre répertoire se nomme RandomGuys [12] Lorsqu'un membre a terminé sa tâche, il ajoute les fichiers modifiés sur la branche concernée, accompagné d'un commentaire pour expliquer les modifications apportées. Il est ensuite plus simple de voir les différences entre chaque version, et de retrouver une ancienne version si besoin.

4.4.2 Redbooth

Redbooth (anciennement Teambox) [11] est un outil de gestion de projet en ligne pour des équipes de quelques membres. Il nous permet de suivre l'évolution des tâches en cours de réalisation, et celles à venir.

Ces tâches peuvent être de différentes sortes :

- Tâches concernant directement le contenu du projet
- Tâches de gestion de projet (outils, documents livrables)
- Tâches pour la gestion des réunions (comptes-rendus, livraisons, signatures, ...)
- Autres tâches (i.e, recherche d'informations sur le choix des langages de développements, état de l'art, analyse de documents spécifiques)

4.4.3 Google drive

Google Drive est un service cloud proposé par google en 2012 pour le stockage et le partage de fichier. C'est une bonne alternative au Git, qui nous permettait de partager des ressources sous toutes formes (articles, logiciels, scripts), et nos résultats en vrac afin de pouvoir les tester sur nos machines (listes d'adresses IP, certificats, fichiers d'insertions en base de données, etc...).
Une fois les fichiers validés ils peuvent être déplacés vers le Git s'ils ont une importance pour le produit final. Il nous permettait également de synchroniser nos résultats notamment lors de notre état de l'art pour la partie 2 du projet.

4.4.4 Google Hangouts

Hangouts est une plate-forme de messagerie instantanée qui nous permettait de partager nos ressentis, d'indiquer notre progression aux autres, et de proposer de l'aide si besoin.
Ce service est très utile car il nous offrait également la possibilité d'accéder immédiatement au Drive et à nos mails, ce qui était un gain de temps non négligeable.

4.4.5 GanttProject et GantterProject

Gantter [6] est une application outil pour le management de projet basé sur le web (que l'on peut d'ailleurs consulter sur le GoogleDrive). Nous avons tout d'abord réalisé notre diagramme de Gantt avec GanttProject, mais il s'est avéré qu'il fallait calculer les informations demandées par l'auditeur (qui jouait le rôle du client).
Parmis les informations non-visibles sur un diagramme réalisé avec GanttProject que l'on trouve sur un GantterProject nous avons :

–

De plus, le diagramme du GantterProject est plus esthétique que notre précédent diagramme, ce qui nous permettait de retrouver plus facilement nos informations.

4.4.6 LaTeX et BibTeX

Latex est un langage de structuration de documents créé en 1983 par Leslie Lamport. Il utilise des macro-commandes qui seront interprétés par le processeur de texte TeX.
BibTeX est un logiciel de gestion de références bibliographiques qui va nous servir à gérer et traiter notre base bibliographique à travers nos documents Latex.

Nous avons choisi de réalisé l'ensemble de nos documents (comptes-rendus, rapports, livrables) sous LaTeX pour qu'ils soient homogènes (nous partions sur la même base), réutilisables et modulaires (découpage sous forme de briques). Les éditeurs/compilateurs de LaTeX sont nombreux, nous avons optés pour deux d'entres eux : Gummi et TexMaker.

4.5 Ressources de tests

Pour nos tests nous avons eu besoin de quelques outils que nous détaillerons ci-dessous.

4.5.1 Netkit

Netkit est un environnement permettant de configurer et de tester un réseau virtuel rapidement et sans grandes ressources.
Lors des procédures de tests de la première partie, nous voulions générer un petit réseau comportant toutes les configurations possibles rencontrables lors du scannage, de la récupération de certificats et de la post-récupération (extraction de données, gestion des doublons, liens symboliques, etc...).
Nous avons donc décidé de réaliser ce petit réseau à l'aide de cet outil.

4.5.2 Pencil

Pencil [8] est un logiciel de création de maquettes typographiques libre et gratuit développé par Evolution Solutions. Nous nous sommes servi de ce logiciel afin de réaliser les maquettes des pages web attendues pour chacune des trois parties.
Le rendu final n'est pas exactement celui des maquettes car nous avons également utilisé plusieurs frameworks pour améliorer la qualité graphique (i.e. HighCharts, BootStraps), mais le contenu et la disposition est sensiblement la même. Ces tests nous ont permis de partir sur une base commune.

4.6 Ressources techniques

Pour ce qui est du développement de scripts, de la gestion du site web avec base de données, et de la mise en place du navigateur sécurisé de la troisième partie, nous avons utilisé les machines de l'université et nos ordinateurs portables. Mais pour les parties plus délicates comme, le scannage de ports Internet, la récupération de certificats, ou la factorisation des moduli pour la première partie nous avions des ressources techniques plus importantes.

Connexion internet : Pour la première partie, nous avions besoin d'une bonne connexion internet pour le scannage, ainsi que pour la récupération des certificats. Il fallait prendre certaines précautions afin de ne pas congestionner le réseau, et éviter également que le proxy ne dérange le déroulement des scripts. Par conséquent, nous avons lancer les scans importants les vendredi soirs, et en utilisant la prise ethernet extérieur.

Serveur de calcul de l'Université : Ce serveur nous a permis de factoriser les moduli lors du deuxième scan (plus de 500.000 certificats), et de gagner du temps sur l'avancement de notre projet.

4.7 Choix des langages de développement

4.7.1 Langage C et librairie GnuMP

Nous avons décidé avant de débuter le projet de faire une étude en benchmark-test [9] [7] [10] (test de performance CPU, RAM, taille de code), sans oublier deux principes fondamentaux qui sont la gestion des grands entiers et les préférences de chacun (degré de compétence, aisance).
Les codes sont basés sur l'utilisation combinée de structures et d'algorithmes complexes (sur arbres, ensembles, anneaux, etc...).

Plusieurs langages ont étés testés parmis lesquels :
- C ;
- C++ ;
- Java ;
- Perl ;
- Python ;
- Ruby ;

Nous avons au final décidé d'utiliser le langage C avec la librairie GnuMP [3] pour la gestion des grands entiers, et le compilateur CMAKE. Le langage C est l'un des plus rapide en temps d'exécution, il est également l'un de ceux qui consomme le moins de mémoire. Il est également très performant sur la gestion des grands entiers.
Le python et le C++ étés également de très bon choix, mais les développeurs ont une meilleure maitrise du C.

4.7.2 Langages de scripts : Bash et Perl

Nous avons opté également pour du développement de scripts pour des algorithmes peu complexes nécessitant plusieurs appels systèmes, surtout pour les commandes OpenSSL (extraction de moduli dans un certificat, connexion au serveur, récupération de champs dans un ensemble X509).
L'avantage des scripts est qu'ils sont facilement modifiables. On pouvait donc les réarranger pour les réutiliser à d'autres fins.

4.7.3 IDE : Eclipse pour C

Eclipse est un environnement de développement et gestionnaire de projets pouvant supporter plusieurs langages de développement comme le Java, l'Android, le C, le C++, etc...

Nous avons utilisé cet éditeur afin de mieux lire le code d'OpenSSL et d'identifier les failles à l'aide des différents commits de l'équipe OpenSSL. Nous l'avons également utilisé pour la troisième partie du projet.

4.8 IHM

4.8.1 Site Web

Nous avons établit un site web local afin de regrouper tout nos résultats sur une interface que nous pourrons offrir au client. Les trois parties sont certes indépendantes, mais elles se complètent parfaitement afin de mieux comprendre la sécurité actuelle de l'Internet, des protocoles SSL/TLS et du logiciel de génération de clés cryptographiques le plus répandu : OpenSSL.
L'ensemble du site web repose sur le framework **BootStrap**.

Sur le premier onglet nous avons réaliser des études statistiques sur les certificats SSL récupérés (certificats contenant des facteurs communs à d'autres certificats, regroupement par *issuer*, recherche utilisateur, tri par taille de clés, etc...).
Les certificats sont stockés en base **MySQL**, et les graphiques sont générés via le framework **HighCharts**.

Sur le second onglet nous listons quelques primitives cryptographiques d'OpenSSL à auditer. Nous avons intégré deux modes un en écriture et un en lecture seule, et chaque élément de cette page est enregistré dans notre base de données.

Enfin, sur le dernier onglet, nous étudions la sécurité du navigateur client en affichant les algorithmes de chiffrement, les algorithmes de signatures, les méthodes de compressions, les versions de protocoles, les courbes elliptiques et les tickets de sessions. Un code couleur est également défini afin d'identifier les *ciphersuites* faibles (Jaune), critiques (Rouge) et fortes (Verte).

4.8.2 Doxygen

Doxygen est un générateur de documentation sous licence libre capable de produire une documentation logicielle à partir du code source d'un programme. Il supporte entre autres le langage C qui est le langage de développement utilisé par OpenSSL. Nous avons décidé de rediriger le code source d'OpenSSL sous Doxygen pour faciliter sa compréhension, et pouvoir analyser plus facilement l'arborescence de la fonction.
La fonction de recherche de code de Doxygen nous a également fait gagner du temps, notamment sur le choix des primitives cryptographiques à étudier et l'identification de vulnérabilités.

4.9 Gestion des risques

Durant ces six semaines, nous avons été confrontés à deux risques que nous avions listés dans le document d'analyse des risques.

- Nous avons eu régulièrement des absences occasionnelles, en grande partie à cause des entretiens pour nos stages, mais aussi pour des problèmes d'administration, de cours, maladies, empêchements.
- Nous n'avons pas pu récupérer les adresses IP des serveurs SSH, le scan de port SSH étant interdit nous avons reçu une plainte d'Amazon, et nous avons décidé avec l'accord du client d'abandonner

cette partie.

Pour le premier point, nous avons bien suivi le plan d'action, en partant sur une base de travail de 8h par jour, et en rattrapant le retard accumulé si possible en semaine ou le samedi. Notre bonne organisation nous a permis de ne pas accumuler de retard, et de pouvoir prendre rapidement la tâche d'un autre en cas d'absence prolongée.

Aucun autre risque identifié n'est survenu durant le projet.

Conclusion

Conclusion rapport final

Bibliographie

[1] 3rd, D. E., Schiller, J., and Crocker, S. Randomness Requirements for Security. RFC 4086 (Best Current Practice), June 2005.

[2] Dierks, T., and Rescorla, E. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), Aug. 2008. Updated by RFCs 5746, 5878, 6176.

[3] GMP. The gnu multiple precision arithmetic library. https ://gmplib.org/.

[4] Heninger, N., Durumeric, Z., Wustrow, E., and Halderman, A. J. Mining your ps and qs : Detection of widespread weak keys in network devices. https ://factorable.net/weakkeys12.extended.pdf, Août 2012.

[5] Heninger, N., Durumeric, Z., Wustrow, E., and Halderman, A. J. Widespread weak keys in network devices. https ://factorable.net/, 2012.

[6] InQuest Technologies, I. Site officiel - gantter. http ://www.gantter.com/, 2014.

[7] MARCEAU, G. The speed, size and dependability of programming languages. http ://blog.gmarceau.qc.ca/2009/05/speed-size-and-dependability-of.html, Mai 2009.

[8] Naidon, P., and Corrieri, P. Site officiel - pencil. http ://www.pencil-animation.org/, 2006-2009.

[9] Primus. Choosing a programming language : So easy, a caveman can do it. http ://the-world-is.com/blog/2013/02/choosing-a-programming-language-so-easy-a-caveman-can-do-it/, Février 2013.

[10] PURER, K. Modern language war. https ://www.udemy.com/blog/wp-content/uploads/2012/01/PROGRAMMING-LANGUAGE-3.png, Janvier 2012.

[11] Redbooth. Site officiel. https ://redbooth.com/, 2010-2014.

[12] Smets, C., Boisseleau, W., Legras, J., Latimier, M., and Edouard, P. Github - randomguys. https ://-github.com/RandomGuys/, Mars 2014.

[13] Wikipedia. Git - wiki. http ://fr.wikipedia.org/wiki/Git, 2012.