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

Liste des Algorithmes

Listings

Introduction

En août 2012, l'Université du Michigan détecte une vulnérabilité critique sur les certificats RSA et DSA d'Internet [3]. Certains ont, en effet, des facteurs premiers communs avec d'autres certificats, et sont donc sujets à une attaque par factorisation. Une telle attaque permet de retrouver très rapidement les clefs privées de ces certificats vulnérables.
Les chiffres sont de plus très élevés, sur 12 828 613 certificats :

- 714 243 utilisent des clefs vulnérables dont 43 852 sont dues à une insuffisance d'entropie ;
- **64 081 utilisent des clefs RSA pouvant être factorisées** ;
- **4 147 utilisent (encore) des clefs prévisibles générées par Debian lors du bug de 2006**
- 123 038 utilisent (encore) des clefs RSA de 512 bits.

Partant de cette étude, notre client souhaitait voir si la sécurité d'Internet avait été améliorée ces deux dernières années en marchant sur les traces de l'Université du Michigan. Dans un second temps il souhaitait que l'on analyse 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 utilisées, 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 ferons 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 HTTPS où nous identifierons certaines faiblesses.

Remerciements

Nous tenons tout d'abord à remercier notre client et professeur M. Ayoub Otmani, pour nous avoir donné 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 orienter vers de bons choix dans notre étude afin de récupérer des informations pertinentes.

Nous remercions également M. Arnaud Citerin pour nous avoir donné accès au serveur de calcul 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 dispensé la formation nécessaire afin d'atteindre cet objectif.

Chapitre 1

Audit de clefs cryptographiques

1.1 Introduction

Cette première partie du projet consiste à faire une étude des clefs cryptographiques RSA circulant sur Internet. Nous avons donc dans un premier temps récupéré une liste d'adresses IP dont le port 443 (HTTPS) était ouvert.

Ensuite, nous avons tenté de récupérer les certificats de toutes les adresses listées. Cette récupération s'occupait également d'extraire les moduli et les clefs publiques des certificats, mais aussi de stocker une liste de certificats doublons.

Enfin, avec tous les moduli que nous avons extraits, nous avons pu construire deux arbres permettant de définir si des moduli contiennent des facteurs premiers communs.

1.2 ZMAP

ZMAP est un outil permettant de faire des scans réseau. Il est capable de scanner toutes les adresses IPv4 possibles, avec une moyenne de 1.4 millions de paquets envoyés par seconde sur les connexions à très haut débit. Nous l'avons utilisé pour scanner l'ensemble des adresses IPv4, en envoyant simplement des paquets SYN aux adresses visées sur le port 443 pour SSL.

Pour l'installer sur nos machines, voici la procédure à suivre :

1. Installation des dépendances :

   ```
   ~ $ sudo apt-get install cmake libgmp3-dev libpcap-dev gengetopt byacc flex git
   ```

2. Récupération des sources de ZMap :

   ```
   ~ $ git clone git://github.com/zmap/zmap.git
   ```

3. Compilation :

   ```
   ~ $ cd zmap
   ~/zmap $ mkdir build
   ~/zmap $ cd build
   ~/zmap/build $ cmake .. -DENABLE_HARDENING=ON
   ~/zmap/build $ make
   ```

```
~/zmap/build $sudo make install
```

Il y a plusieurs options disponibles, nous avons utilisé la commande suivante pour effectuer notre scan :

```
# zmap -p 443 -o scan_output_zmap
```

1.3 Application RC

Une fois que l'on avait la liste d'adresses fournie par ZMAP, il nous restait à récupérer les certificats sur les serveurs dont le port HTTPS était ouvert, en établissant une connexion sur chacun d'eux.

1.3.1 Récupération des certification

Nous avons codé un script perl pour la récupération de certificats SSL (ssl_collector.pl), suivant l'algorithme 1.

Entrées : Fichier f, contenant les adresses ayant le port 443 ouvert
Sorties : Certificats et clef de session des adresses
Données : log, certificat, clef de session
pour tous les *adresses de f* **faire**
  Se connecter au serveur;
  si *echec* **alors**
    si *le log contient protocol* **alors**
      on incrémente le nombre d'échecs de protocoles;
    fin
    si *le log contient handshake* **alors**
      on incrémente le nombre d'échecs de poignées de mains;
    fin
  sinon
    On capture la session (dont le certificat serveur);
    si *!echec de capture de session* **alors**
      on extrait le certificat et la clef de session;
    fin
  fin
fin
retourner *certificats et clés de session*

Algorithme 1 : Récupération des certificats

Commandes OpenSSL utilisées :

— pour l'établissement de la connexion (avec s_client) :
```
$ echo  \" \" | timeout 20 openssl s_client -connect $addr:443 -sess_out tmp_$addr.pem
-ignore_critical -showcerts -CApath /etc/ssl/certs > /dev/null 2> tmp_$addr.log
```
— pour la capture de session (avec sess_id) :
```
$ openssl sess_id -in tmp_$addr.pem -cert > certs/$addr.pem 2> tmp_$addr.log
```
— pour la récupération du certificat et de la clé privée :
```
$ openssl x509 -in certs/$addr.pem -noout -pubkey > keys/$addr.pem 2> tmp_$addr.log
```

1.3.2 Gestion des doublons

Pour ce qui est de la gestion des doublons, nous avons également créé un autre script perl (`clean.pl`). Le concept est simple, nous allons stocker l'ensemble des empreintes des certificats dans un dossier spécifique (certs_doublons). Si une empreinte se trouve déjà dans le dossier, on la stocke dans un autre dossier contenant les doublons (à l'aide de comparaisons sur les liens symboliques).

Entrées : Pré-requis : Exécution de l'algorithme ssl_collector
Création d'un dossier D contenant les certificats à traiter
Sorties : Dossiers : certs_doublons, certs_links ; Fichier : moduli
Données : Certificats C ; Fingerprint F ; Chaîne de caractères S ; Modulo M
certs_doublons ← dossier_vide;
certs_links ← dossier_vide;
moduli ← fichier_vide;
pour tous les $C \in D$ **faire**
  F ← *fingerprint*(C);
  S ← *nom_fichier*(C);
  M ← *modulo*(C);
  si *échec(F)* **alors**
    continue;
  fin
  si $F \in$ *certs_links* **alors**
    certs_doublons/F ← concat(certs_doublons/F, S);
  sinon
    moduli ← concat(moduli, M);
    certs_links ← F;
  fin
fin

Algorithme 2 : Gestion des doublons

Commandes OpenSSL utilisées :
— pour la récupération du *fingerprint* :
```
$ openssl x509 -noout -in $file -modulus 2> /dev/null | cut -d'=' -f2 |
 sha512sum | cut -d' ' -f1 2> /dev/null
```
— pour savoir le *fingerprint* se trouve dans *certs_links* :
```
$ ls -l certs_links | grep $hash 2> /dev/null
```
— pour ajouter le modulo dans le fichier *moduli* :
```
$ openssl x509 -noout -in $file -modulus 2> /dev/null | cut -d'=' -f2 >> moduli
```
— Pour créer un lien symbolique dans *certs_links*
```
$ ln -s ../$file certs_links/$hash > /dev/null 2> /dev/null
```

1.4 Factorisation

1.4.1 Algorithmes

L'objectif final de cette première partie était de déterminer s'il existait des facteurs premiers communs parmi les clefs récupérées. Pour cela nous avons développé deux arbres de factorisation. Lors du scan nous avons récupéré 85 000 certificats uniques, mais nous avons également souhaité faire la factorisation avec les données du projet Sonar de Rapid7 Labs avec 523 000 certificats.

Le premier arbre se contente de multiplier l'ensemble des moduli par groupe de deux. Le second arbre calcule deux moduli du résultat produit, selon le carré des résulats stockés par l'arbre des produits (aux étapes intermédiaires).

Une fois le second arbre calculé, les dernières feuilles des arbres indiquent soit un $PGCD(\text{res\_final}, N_i) = 1$ (donc avec des entiers premiers uniques), ou un $PGCD(\text{res\_final}, N_i) = p$ (entier premier non unique).

Voici les algorithmes de ces deux arbres :

Entrées : Tableau des moduli des clefs publiques : T
Sorties : Hauteur de l'arbre, produits des moduli des clefs publiques
Données : Tableaux v, tmp ; Entier i, level
$v \leftarrow T$;
$level \leftarrow 0$;
tant que $|v| > 1$ **faire**
  $tmp \leftarrow \emptyset$;
  pour chaque $i \in \{0, .., |v|/2\}$ **faire**
    $tmp[i] \leftarrow v[i \times 2] \times v[i \times 2 + 1]$;
  fin
  $storeProductLevel(v, level)$;
  $v \leftarrow tmp$;
  $level \leftarrow level + 1$;
fin
retourner *level*

Algorithme 3 : Construction de l'arbre des produits

Entrées : Hauteur de l'arbre des produits : level
Sorties : PGCDs des moduli des clefs publiques
Données : Tableaux P, v, w ; Entier i
$P \leftarrow getProductLevel(level)$;
tant que $level > 0$ **faire**
  $level \leftarrow level - 1$;
  $v \leftarrow getProductLevel(level)$;
  pour chaque $i \in \{0, .., |v|\}$ **faire**
    $v[i] \leftarrow P[i/2] \pmod{v[i]^2}$;
  fin
  $storeRemainderLevel(v, level + 1)$;
  $v \leftarrow tmp$;
fin
$w \leftarrow \emptyset$;
pour chaque $i \in \{0, .., |v|\}$ **faire**
  $w[i] \leftarrow P[i/2] \pmod{v[i]^2}$;
  $w[i] \leftarrow w[i]/v[i]$;
  $w[i] \leftarrow pgcd(w[i], v[i])$;
fin
retourner w

Algorithme 4 : Construction de l'arbre des restes

1.4.2 Implémentation

Lors de nos tâches d'optimisation, nous avons réalisé qu'il était possible de construire l'arbre entier (*product tree + remainder tree*), en mémoire vive (nous avons donc créé une option fullRAM à notre algo).

L'algorithme est le même, seul le choix de la structure change. Dans la première méthode, nous utilisions des fichiers binaires GMP à chaque étape de construction, dans la seconde nous stockons l'intégralité de l'arbre des produits en mémoire.

Pour accélérer la construction de l'arbre, nous utilisons des *threads*, qui parallélisent le traitement de chaque niveau d'arbre (sur la largeur de l'arbre). Nous avons tenté d'améliorer encore le procédé, en parallélisant le calcul sur la hauteur de l'arbre. Il s'est avéré que ce n'était pas aussi efficace à cause des dépendances entre les niveaux, nous avons donc abandonné cette partie.

Nous allons dans cette partie expliciter les composantes principales implémentées en mémoire de l'algorithme de factorisation. Nous développerons notamment la méthode `computeSuperSpeed`, qui prend en entrée un nom de fichier, contenant la liste des moduli.

1.4.2.1 Initialisation

```
1  vec_t v = {0};
2  transformFile(input);
3  char *moduli_filename = "PInterm1";
```

```
  input_bin_array(&v, moduli_filename);

  int count = v.count;
  int levels_count = (int) ceil (log (count) / log (2)) + 1;
  product_tree.levels_count = levels_count;

  product_tree.levels = (vec_t *) malloc (levels_count * sizeof (vec_t));
  product_tree.levels[0] = v;
  int count_i = count / 2;
  for (int i = 1; i < levels_count; i++) {
    init_vec (product_tree.levels + i, count_i);
    product_tree.levels[i].count = count_i;
    count_i = count_i / 2;
  }
  printf ("Done.\n\n");
```

Listing 1.1 – fact_superspeed.c - partie 1

Dans cette fraction de méthode, on initialise les différentes variables :

— le fichier contenant les moduli au format binaire est chargé, dans le vecteur `v`. Le vecteur est une structure décrite par son nombre d'éléments et par une liste de `mpz_t` ;

— on calcule le nombre de niveaux à dérouler, considérant que l'arbre est binaire ;

— on initialise enfin les vecteurs de la structure `product_tree`. Il s'agit d'une structure représentant l'arbre des produits, composée d'une hauteur d'arbre et d'une liste de vecteurs.

1.4.2.2 Calcul de l'arbre des produits

```
  for (int i = 1; i < levels_count; i++) {
    vec_t *w = product_tree.levels + i;
    vec_t *v2 = product_tree.levels + i - 1;
    void mul_job(int i) {
      mpz_mul(w->el[i], v2->el[2*i], v2->el[2*i+1]);
    }
    iter_threads(0, v2->count/2, mul_job);

    if (v2->count & 1)
      mpz_set(w->el[v2->count/2], v.el[v2->count-1]);

  }
```

Listing 1.2 – fact_superspeed.c - partie 2

Le déroulement de l'arbre des produits est plutôt simple. Les éléments connexes sont multipliés à chaque niveau puis le résultat est stocké dans le niveau suivant correspondant. On remarque également la méthode de parallélisation sur cette boucle afin d'accélérer la multiplication.

Nous avons pensé à intégrer des améliorations pour couper des branches inutiles, ou encore accélérer le développement de l'arbre en parallélisant également en hauteur, mais ces opérations se sont avérées inutiles car elles nécessitaient la mise en place de tests qui ralentissaient l'exécution du programme. L'opération de multiplication n'est en effet que très légèrement gourmande en processus, nous avons décidé de garder un

développement d'arbre le plus épuré possible.

1.4.2.3 Calcul de l'arbre des restes

```
int secL;
int sizeP=1;
vec_t *modPre = product_tree.levels + product_tree.levels_count - 1;
vec_t modCur,*prodL,gcd;
secL = product_tree.levels_count;

do {
  sizeP *= 2;
  init_vec(&modCur,sizeP);


  prodL = product_tree.levels + secL - 2;
  void mul_job(int j){
    mpz_pow_ui(prodL->el[j],prodL->el[j],2);
    mpz_mod(modCur.el[j],modPre->el[j/2],prodL->el[j]);
  }
  iter_threads(0, prodL->count, mul_job);

  *modPre = modCur;
  free_vec (prodL);
} while (--secL > 1);
```

Listing 1.3 – fact_superspeed.c - partie 3

Le calcul de l'arbre des restes se fait aussi efficacement. On récupère de l'arbre des restes, le produit correspondant au niveau équivalent de celui de l'arbre des restes, produit que l'on élève au carré. On effectue ensuite le calcul modulaire requis suivant l'algorithme. On remarque également le *multi-threading* effectué. Durant ce calcul, on ne conserve en mémoire que le niveau j et $j-1$ de l'arbre des restes, afin de ne pas surcharger la mémoire.

1.4.2.4 Dernière étape de l'arbre des restes

```
init_vec(&gcd,modCur.count);

  input_bin_array (&v, moduli_filename);
  void mul_job(int j){
    mpz_divexact(modCur.el[j],modCur.el[j],v.el[j]);
    mpz_gcd (gcd.el[j],modCur.el[j],v.el[j]);
  }
  iter_threads(0, v.count, mul_job);
```

Listing 1.4 – fact_superspeed.c - partie 4

Afin de ne pas avoir à ajouter des tests dans la boucle principale, nous avons séparé la dernière étape du calcul de l'arbre des restes. En effet, cette étape s'accompagne d'une division et d'un calcul de PGCD qui peuvent eux aussi être parallélisés.

1.4.2.5 Identification des moduli vulnérables

Il convient enfin d'analyser les PGCD obtenus. Dans un premier temps, on récupère tous les PGCD étant différents de 1, que l'on stocke dans un vecteur de moduli potentiellement vulnérables. De plus, si celui-ci est premier, alors on stocke ce premier dans un vecteur spécifique : `potentielVuln`.

```
vec_t potentielVuln, premiers;
init_vec (&potentielVuln, gcd.count);
init_vec (&premiers, gcd.count);
mpz_t q;
mpz_init (q);

int pvuln = 0, ppremier = 0;
for (int i = 0; i < gcd.count; i++) {
  if (mpz_cmp_ui (gcd.el[i], 1) != 0) {
    mpz_set (potentielVuln.el[pvuln++], v.el[i]);
    if (mpz_probab_prime_p (gcd.el[i], 25) != 0) {
      mpz_set (premiers.el[ppremier++], gcd.el[i]);
    }
  }
}
```

Listing 1.5 – fact_superspeed.c - partie 5a

Enfin, on récupère tous les moduli vulnérables potentiels étant divisibles par un premier récupéré. On en profite également pour récupérer q où moduli $= p * q$ avec p étant le premier récupéré. On écrit ces éléments dans un fichier : `moduli_p_q`.

```
    potentielVuln.count = pvuln;
premiers.count = ppremier;

FILE *result = fopen ("moduli_p_q", "w");
start = now ();
for (int i = 0; i < potentielVuln.count; i++) {
  for (int j = 0; j < premiers.count; j++) {
    if (mpz_divisible_p (potentielVuln.el[i], premiers.el[j]) != 0) {
      mpz_divexact (q, potentielVuln.el[i], premiers.el[j]);
      if (mpz_probab_prime_p (q, 25) != 0) {
        if (input_type == INPUT_HEXA)
          gmp_fprintf (result, "%ZX, %ZX, %ZX\n", potentielVuln.el[i], premiers.el[j
], q);
        else
          gmp_fprintf (result, "%Zd, %Zd, %Zd\n", potentielVuln.el[i], premiers.el[j
], q);
      }
    }
  }
}
```

Listing 1.6 – fact_superspeed.c - partie 5b

1.4.2.6 Compilation

Pour compiler le programme :

```
~ $ sudo apt-get install libgmp-dev cmake git
~ $ git clone https://github.com/RandomGuys/fact.git
~ $ cd fact
~/fact $ mkdir build
~/fact $ cd build
~/fact/build $ cmake ..
~/fact/build $ make
~/fact/build $ sudo make install
```

Il y a également un *man* qui s'installe, vous pouvez donc consutler l'aide avec la commande `man fact`. Pour utiliser ce programme, la ligne de commande est la suivante :

```
$ fact -m mes_moduli --fullram|--files [--format hexa|decimal]
```

1.5 Conclusion des résultats produits

1.5.1 Fichier de résultats

Les résultats se trouvent dans le fichier `moduli_p_q` construit de la façon suivante :

```
modulus, facteur commun, facteur déduit
...
```

1.5.2 Premiers chiffres

Voici les premiers chiffres obtenus des scans :

	Scan 1	**Scan Sonar**
moduli	85 636	523 017
vulnérables	104 (0,12%)	2 382 (0,46%)
facteurs communs	11	757
autres facteurs	104	2 382
total facteurs	115	3 139

TABLE 1.1 – Résultats globaux obtenus

En comparant avec les résultats du projet *factorable* de l'Université de Californie et de l'Université du Michigan, nous obtenons un pourcentage de clefs vulnérables assez proche. En effet, ils ont obtenu 0,50% de clefs vulnérables et nous en avons trouvées 0,46%

Pour la suite de l'analyse des résultats, nous nous sommes focalisés sur le scan Sonar.

1.5.3 Entropie

Le graphe suivant représente le pourcentage d'entropie des 757 facteurs premiers trouvés. Ce niveau d'entropie a été calculé selon l'entropie de Shannon (représentation binaire des clefs). Nous nous sommes basés sur la bibliothèque `libdisorder` (`http://libdisorder.freshdefense.net/`) qui fournit un programme de tests. Ce dernier a été modifié pour nos besoins et pour pouvoir injecter les résultats dans `gnuplot` pour obtenir le graphe suivant :

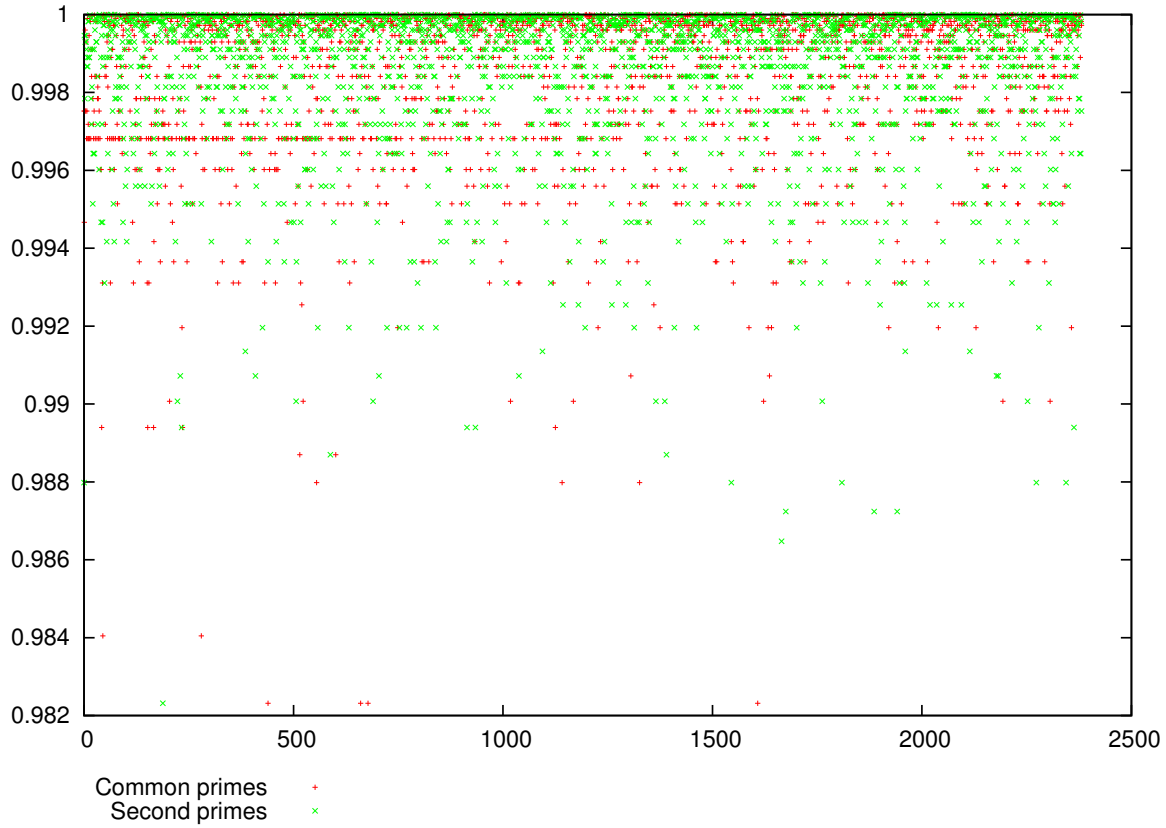


FIGURE 1.1 – Entropie moyenne des facteurs communs : 0.998518 - Entropie moyenne des seconds facteurs : 0.998551

1.5.4 Tailles des clefs

Dans nos résultats, nous pouvons constater que seules deux tailles de clefs apparaissent : 512 et 1024 bits. Voici les chiffres :

	512 bits	**1024 bits**
Total	2 345	318 558
Vulnérables	6 (0,26%)	2 376 (0,75%)

TABLE 1.2 – Tailles de clefs obtenues

Nous avions également d'autres tailles de clefs en entrée mais leur nombre devait être trop petit par rapport à l'espace des clefs de cette taille pour obtenir des facteurs communs.

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 (successeur de SSL).

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.

2.1.1 Entropie

L'entropie est la base pour générer des nombres pseudo-aléatoire. Elle joue donc un rôle prépondérant dans la sécurité de la cryptographie qui sera mise en place. Son générateur est composé de trois principaux éléments :

le bruit source : il doit être non déterministe, et renvoie de façon aléatoire des bits grâce à des processus non déterministes ;

le composant de conditionnement : il permet d'augmenter ou diminuer le taux d'entropie reçu ;

une batterie de tests : elle fait 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.

Plusieurs standards parlent de l'entropie, dont principalement la RFC 4086 **??** et le FIPS 140 **?? ??**. Mais malgré les recommandations, quelques failles ont été trouvées. Pour plus d'informations, nous vous invitons à aller consulter le rapport d'audit.

2.1.2 Génération des clefs

Les clefs cryptographiques sont un principe fondamental en cryptographie. En effet, elles sont censées être le seul secret à garder précieusement par leur possesseurs : il est conseillé que les algorithmes et tous les outils ne soient pas secret afin d'être certain de ne pas insérer de faille. Les algorithme reconnus et éprouvés sont beaucoup plus fiables que les algorithmes secrets et personnels.

Par conséquent les clefs se doivent d'être les plus solides possible. Mais malgré le soin qu'on peut y apporter, et les différentes recommandations qui sont faites par les normes, des failles existent. Nous vous invitons à vous référer au rapport d'audit pour de plus amples informations.

2.1.3 Chiffrement et protocoles

De nombreux protocoles de chiffrement existent :
- symétrique :
  - AES ;
  - Blowfish ;
  - DES, ou aujourd'hui le triple-DES ;
  - etc.

- asymétrique :
  - DSA ;
  - El Gamal ;
  - RSA ;
  - etc.

RSA étant l'un des algorithmes les plus utilisés en cryptographie asymétrique, et le temps nous ayant été limité, nous nous sommes concentrés dessus.
Cependant, RSA seul n'est plus d'une sécurité absolue. Il est combiné à OAEP, qui constitue une sorte de prétraitement à RSA lui même. Nous parlons de RSA-OAEP. Grâce à OAEP, un padding est rajouté avant l'application de RSA. La norme princiaple qui se charge des recommandations sur RSA est la PKCS#1, qui a aussi le nom de RFC 3447 **?? ??**. Plusieurs failles ont été trouvées et suite à ces découvertes, des recommandations ont été faites. Pour de plus amples informations nous vous invitons à parcourir le rapport d'audit.

2.1.4 Signature et chiffrement

La signature électronique est de plus en plus utilisée de nos jours, de façon plus ou moins transparente. Un mécanisme cmplexe est mis en place derrière, avec toute une chaîne d'autorités de certification et de révocation des certificats. Ils permettent de faire un lien fiable entre la clef et son possesseur. Processus de signature :

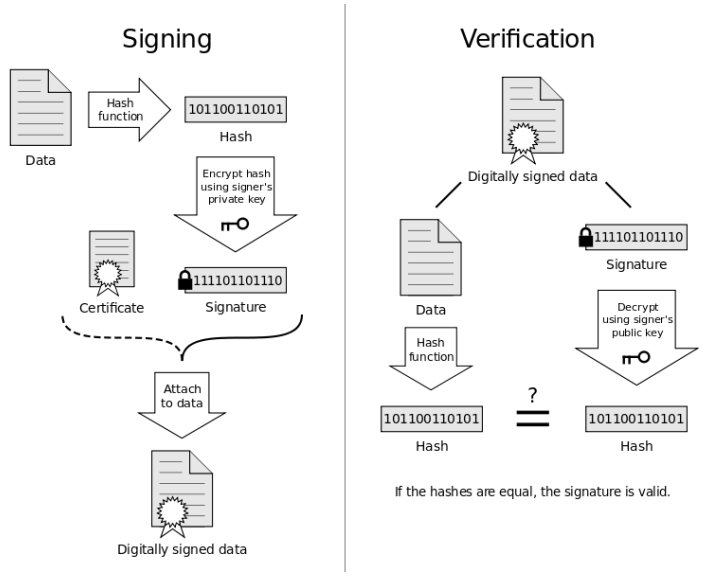


FIGURE 2.1 – Signature électronique

La RFC 3447 **??** **??** apporte aussi des recommandations à ce propos, ainsi que la RFC 979 **??**. Pour consulter les principales failles que nous avons trouvées à ce sujet, nous vous invitons à aller consulter le rapport d'audit.

2.1.5 Protocole SSL/TLS

SSL et TLS (successeur de SSL) sont des protocoles de sécurisation des échanges sur internet. Les versions 2 et 3 de SSL ont été développées par Netscape puis le brevet a été racheté par l'IETF en 2001 qui a publié une évolution de ce protocole en TLS. Ce protocole fonctionne selon un mode client-serveur et fournit les objectifs de sécurité suivants :

- authentification serveur/client ;
- confidentialité des données échangées ;
- intégrité des données échangées.

Du point de vue réseau, ce protocole se situe dans la couche session du modèle OSI et entre transport et application dans le modèle TCP.

Pour de plus amples informations, nous vous invitons à aller consulter le rapport d'audit.

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 [1] 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.

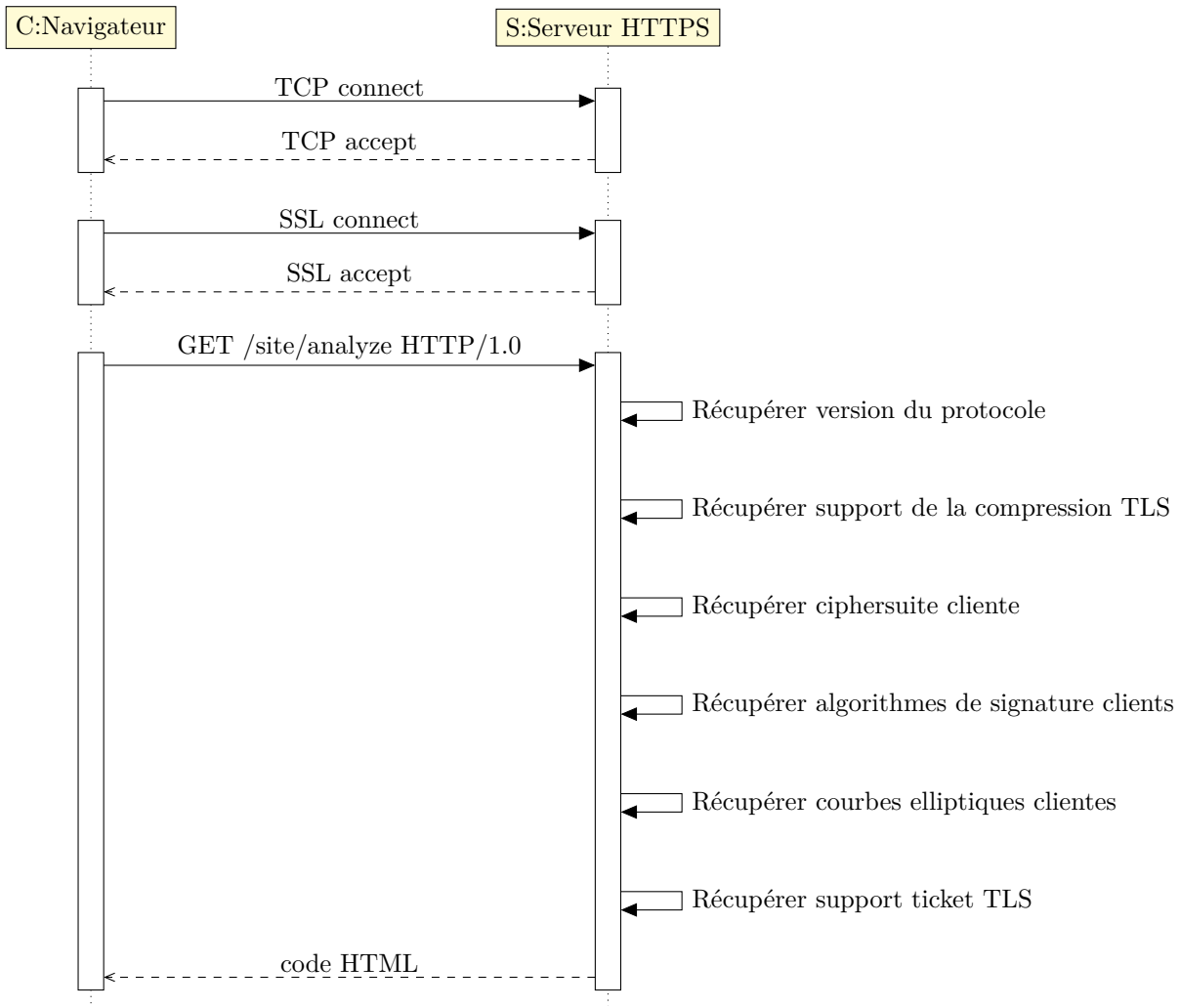


FIGURE 3.1 – Schéma de l'analyse dynamique du navigateur client

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

Site Web

4.1 Objectifs

Le site web fut principalement un moyen de présenter, de façon graphique, le résultat des différentes parties abordées durant ce projet. Pour la première partie, il consistait à afficher, grâce à l'utilisation des diagrammes, les statistiques de notre étude sur les certificats. Pour la deuxième partie, nous avons souhaité présenter le rapport d'audit au format HTML, celui-ci a également été livré lors de la deuxième livraison au format LaTeX. Enfin, notre site fut aussi une plate-forme pour l'intégration de la troisième partie du projet qui consiste à tester le navigateur du client.

4.2 Statistiques

Après quelques recherches, nous nous sommes mis d'accord sur l'utilisation de l'application javascript *Highcharts* pour la représentation des différents diagrammes sur le site. L'utilisation est simple et varié, nous l'avons utilisé de trois façons différentes :

— **Statique** : données sont récupérées directement depuis un tableau HTML ;
— **Dynamique** : données sont récupérées depuis la base de données ;
— **AJAX** : données sont récupérées par des appels AJAX.

Chaque diagramme possède une fonctionnalité *tooltip* qui offre plus d'informations sur les données représentées. Il est actionné lors du passage de la souris sur une part du cadre du diagramme circulaire (cf. figure 4.2)

4.2.1 Statistiques Générales

Nous avons souhaité faire une comparaison entre les deux scans réalisés durant le développement de la première partie du projet. De ce fait, nous présentons sur un même diagramme la proportion des certificats récupérés et ceux qui sont vulnérables pour le premier et deuxième scan. On y affiche également un tableau contenant, en chiffres, les valeurs du diagramme ainsi que le pourcentage de certificats vulnérables sur l'ensemble récupéré.

FIGURE 4.1 – Site Web - Statistique Général

4.2.2 Taille des clefs

On représente ici, avec un diagramme circulaire, les tailles des clefs des certificats récupérés. De tout les certificats se trouvant sur notre base de données, on identifie ceux possédant une taille de clef allant de 512 octets à 16.384 octets.

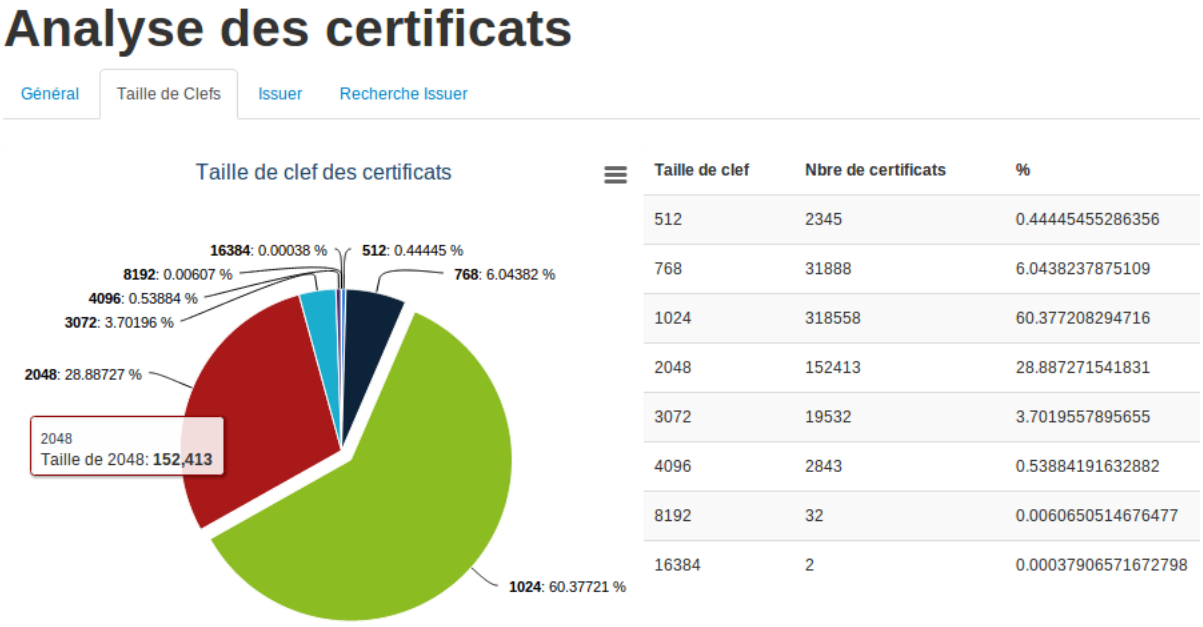


FIGURE 4.2 – Site Web - Taille de clef

4.2.3 Les émetteurs

Cette partie représente un tableau de tout les émetteurs se trouvant sur notre base de données. On y récupère aussi, à l'aide de deux requêtes SQL, le nombre total de certificats que possède chacun des émetteur ainsi que le nombre de certificats vulnérables.

FIGURE 4.3 – Site Web - Emetteurs

4.2.4 Onglet de recherche

Nous avons également intégré une barre de recherches. L'utilisateur peut ainsi faire une recherche avancée, en choisissant parmi les critères suivants :

— Le sujet (*subject* ;
— L'émetteur (*issuer*) ;
— La taille de la clef.

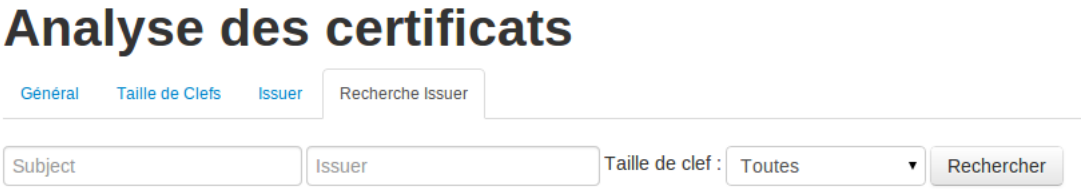


FIGURE 4.4 – Site Web - Bar de recherche

La requête est envoyée sur le serveur via un appel en AJaX. Le résultat est alors affiché, avec un diagramme circulaire, pour représenter le nombre total de certificats et de certificats vulnérables contenus dans la base.

FIGURE 4.5 – Site Web - Résultat d'une recherche

4.3 Résumé d'audit

Nous souhaitions, pour cette partie, intégrer le rapport d'audit directement sur le site web. De ce fait, nous avons utilisé un outil de conversion de fichier *TeX* dans un dossier de fichiers *HTML* que nous avons ensuite rejoins dans le dossier racine de notre site. Afin de garder la même mise en forme des pages web du site, nous avons modifié les différentes balises en y rajoutant les styles nécessaires.

4.4 Test Navigateur Client

Cette partie intègre l'outil permettant d'analyser les différentes *ciphersuite* utilisées à l'établissement de connexion entre un client et un serveur, comme expliqué au chapitre 3. La page est divisé en 4 sections :

- Version et Algorithmes de chiffrements supportés ;
- Algorithmes de signatures supportés ;
- Courbes elliptiques ;
- Légende.

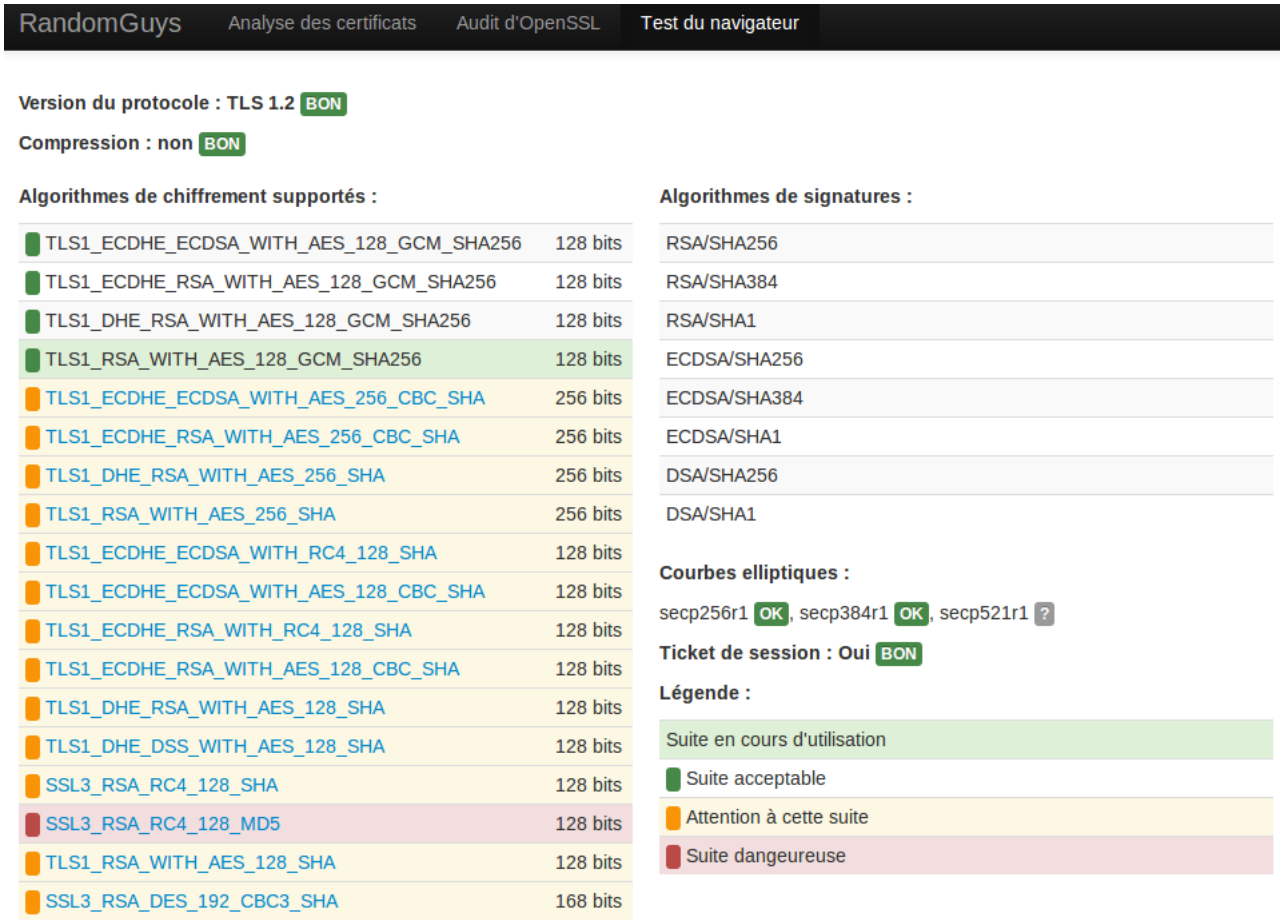


FIGURE 4.6 – Site Web - Test du Navigateur

Chapitre 5

Vie de projet

5.1 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 de Rouen. 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 à implémenter 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'étude, de tests, d'interfaces, etc...).

5.2 Le client

M. Otmani est notre client pour ce projet. Le cahier des charges se divise en trois grandes parties. Pour la première, le client attend de nous un audit solide sur les clefs cryptographiques, présentable, et intuitif. Pour cela, il nous a conseillé d'étudier un article publié par des universitaires de Californie et du Michigan, ainsi que le site `factorable.net`.

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, et le chiffrement.

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 clef primaire, le contrôle des certificats, le respect du protocole, le choix de l'algorithme etc.

Cette dernière partie est optionnelle, elle sera effectuée si le temps nous le permet. Le projet se déroule sur six semaines, le temps de travail est de 8h/jour (pour pouvoir palier le temps perdu par chacun pour la recherche de stage, les charges administratives, etc...)

5.3 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 ces sous-parties nous allons revenir sur le choix de la méthode, puis nous allons détailler son application tout au long du projet.

5.3.1 Pourquoi Scrum ?

Les méthodes agiles ont fait leur preuve dans leur efficacité et leur qualité de développement. De plus, Scrum permet un suivi et une transparence totale avec le client. Le découpage en sprints est adapté à notre projet puisqu'il se déroule en différentes parties et sur une période relativement courte.

5.3.2 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 avec de bonnes phases de tests, des réunions techniques en cas de doute, ... ;
- rentrer efficacement dans les sujets les plus importants du projet, surtout pour la partie de l'audit de code OpenSSL ;
- exposer efficacement nos travaux au client, les possibilités qui s'offrent 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 partis du principe que le client faisait partie intégrante de l'équipe. Son avis nous intéressant, nous lui avons montré lors de toutes nos réunions nos travaux afin qu'il puisse s'assurer du bon avancement du projet et qu'il puisse nous aiguiller pour la suite.

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

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

5.3.3 Réunions

5.3.3.1 Brainstorming et Stand-Up Meeting

Chaque matin tout le monde se réunit autour d'un café, et partage son ressenti sur le projet, et sur 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.

5.3.3.2 Réunions hebdomadaires

Nous nous réunissons deux fois par semaine, le jeudi. En début d'après-midi, ou en fin de matinée, nous passons dans une salle au calme, pour parler des difficultés techniques rencontrées, des axes d'améliorations, de la réorganisation ou du découpage des tâches si besoin.

Nous évoquons également les points importants à apporter au client pour la réunion qui suit. Ces réunions s'apparentent aux "Planifications du Sprint" et à la "Rétrospective de Sprint" du SCRUM.

L'après-midi, selon la disponibilité du client, nous nous réunissons pour parler de notre avancée, relever les remarques et les demandes du client, présenter nos résultats ou livrer une partie de projet finie.
Ces réunions s'apparentent aux "Revue de Sprint" du SCRUM.

5.3.3.3 Réunions d'urgences

Plus rarement, il a pu nous arriver d'avoir des réunions d'urgence. Nous nous sommes ainsi réunis avec le client lorsque nous étions bloqués sur la récupération des certificats SSH, ou lorsque l'on s'est aperçus qu'une amélioration majeure était possible.

5.3.3.4 Audit

Les audits 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 amélioré nos outils de gestion de projet, par exemple en passant notre planning sur le GantterProject afin de mieux visualiser les tenants et les aboutissants d'une tâche, et perfectionner nos documents livrables notamment au niveau de la traçabilité.

5.4 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.

5.4.1 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.

Lorsqu'un membre a terminé sa tâche, il ajoute les fichiers modifiés sur la branche concerné, en plaçant 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.

5.4.2 Redbooth

Redbooth (anciennement *Teambox*) est un outil de gestion de projet en ligne pour des équipes de quelques membres. Il nous permet de suivre l'évolution des tâches, telles que les tâches en cours de réalisation et celles à venir. Ces tâches peuvent être de différentes sortes :

- tâches du projet ;
- tâches de gestion de projet (outils, documents livrables) ;
- tâches pour la gestion des réunions (comptes-rendus, livraisons, signatures, ...) ;
- autres tâches (ex : recherche d'informations sur le choix des langages de développements, état de l'art, analyse de documents spécifiques).

5.4.3 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 pêle-mêle, afin de pouvoir les tester sur nos machines (listes d'adresses IP, certificats, fichiers d'insertion en base de données, etc.)

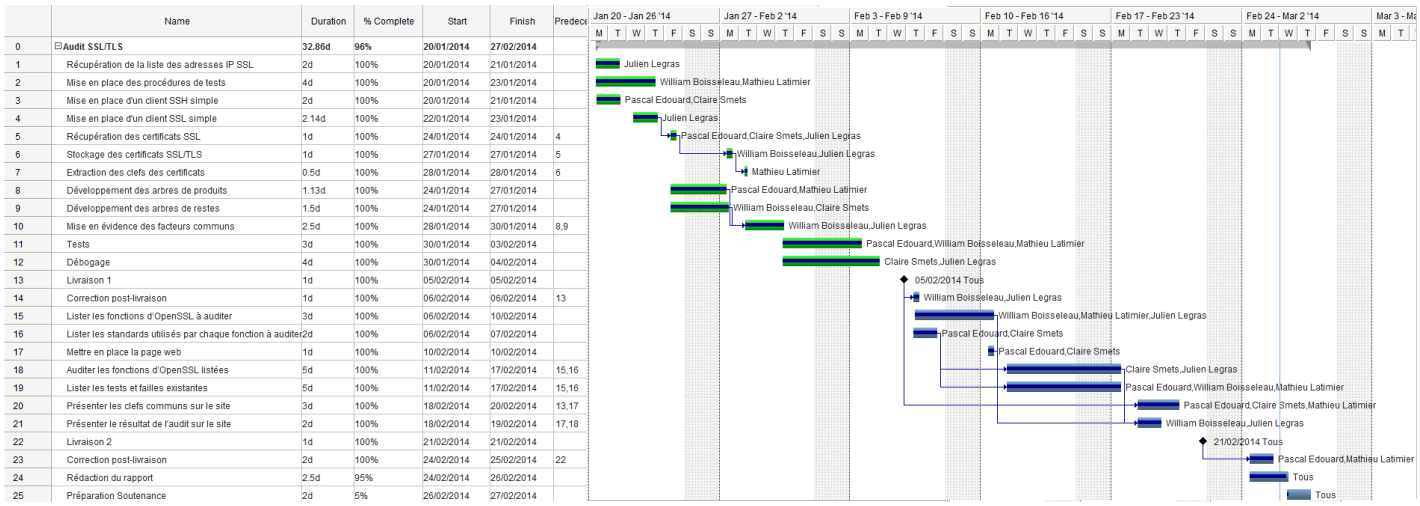
Une fois les fichiers validés ils pouvaient être déplacés vers le Git si l'on pensait qu'ils avaient une importance pour le projet final. Il nous permettait également de synchroniser nos résultats notamment lors de notre état de l'art pour la partie 2 du projet.

5.4.4 Google Hangouts

Hangouts est une plate-forme de messagerie instantanée qui nous permettait de partager nos ressentis, d'indiquer notre progression sur une tâche aux autres, proposer de l'aide si besoin.
Ce service est très utile car il nous permettait également de rester sur la même plate-forme que le Drive et nos mails, ce qui était un gain de temps non négligeable.

5.4.5 GanttProject et Gantter

Gantter [4] 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).



	Name	Duration	% Complete	Start	Finish	Predec
0	Audit SSL/TLS	32.86d	96%	20/01/2014	27/02/2014	
1	Récupération de la liste des adresses IP SSL	2d	100%	20/01/2014	21/01/2014	
2	Mise en place des procédures de tests	4d	100%	20/01/2014	23/01/2014	
3	Mise en place d'un client SSH simple	2d	100%	20/01/2014	21/01/2014	
4	Mise en place d'un client SSL simple	2.14d	100%	22/01/2014	23/01/2014	
5	Récupération des certificats SSL	1d	100%	24/01/2014	24/01/2014	4
6	Stockage des certificats SSL/TLS	1d	100%	27/01/2014	27/01/2014	5
7	Extraction des clefs des certificats	0.5d	100%	28/01/2014	28/01/2014	6
8	Développement des arbres de produits	1.13d	100%	24/01/2014	27/01/2014	
9	Développement des arbres de restes	1.5d	100%	24/01/2014	27/01/2014	
10	Mise en évidence des facteurs communs	2.5d	100%	28/01/2014	30/01/2014	8,9
11	Tests	3d	100%	30/01/2014	03/02/2014	
12	Débogage	4d	100%	30/01/2014	04/02/2014	
13	Livraison 1	1d	100%	05/02/2014	05/02/2014	
14	Correction post-livraison	1d	100%	06/02/2014	06/02/2014	13
15	Lister les fonctions d'OpenSSL à auditer	3d	100%	06/02/2014	10/02/2014	
16	Lister les standards utilisés par chaque fonction à auditer	2d	100%	06/02/2014	07/02/2014	
17	Mettre en place la page web	1d	100%	10/02/2014	10/02/2014	
18	Auditer les fonctions d'OpenSSL listées	5d	100%	11/02/2014	17/02/2014	15,16
19	Lister les tests et failles existantes	5d	100%	11/02/2014	17/02/2014	15,16
20	Présenter les clefs communs sur le site	3d	100%	18/02/2014	20/02/2014	13,17
21	Présenter le résultat de l'audit sur le site	2d	100%	18/02/2014	19/02/2014	17,18
22	Livraison 2	1d	100%	21/02/2014	21/02/2014	
23	Correction post-livraison	2d	100%	24/02/2014	25/02/2014	22
24	Rédaction du rapport	2.5d	95%	24/02/2014	26/02/2014	
25	Préparation Soutenance	2d	5%	26/02/2014	27/02/2014	

FIGURE 5.1 – Gantter

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

5.4.6 LaTeX et BibTeX

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

Nous avons choisi de réaliser l'ensemble de nos documents (comptes-rendus, rapports, livrables) sous LaTeX pour qu'ils soient homogènes (nous partions sur la même base), réutilisables et modulables (découpage sous forme de briques).

Les éditeurs/compilateurs de LaTeX sont nombreux, nous avons opté pour deux d'entres eux : Gummi et TexMaker.

5.5 Tests

Lors de notre projet, nous avons mis en place des procédures de test en amont, permettant de valider le code développé. Ceci n'étant pas requis en tant que livrable, nous les décrirons rapidement dans cette section. Les procédures de tests sont disponibles sur le dépôt git [8]. Nous avons eu besoin de quelques outils que nous détaillons ci-dessous.

5.5.1 Ressources de Tests

5.5.1.1 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 scan, 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 l'outil Netkit.

5.5.1.2 Pencil

Pencil est un logiciel de création de maquettes typographiques libre et gratuit développé par *Evolution Solutions*. Nous nous sommes servis 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.

5.5.2 Procédures de tests

Pour la configuration du réseau minimaliste, possédant différentes configurations que nous pourrons rencontrer lors du scan, puis de la récupération du certificat, les différentes configurations sont les suivantes :

- une machine avec un certificat SSL présent et valide ;
- une machine avec un certificat SSL présent et révoqué ;
- une machine avec un certificat SSL modifié (donc incorrect) ;
- une machine avec un certificat SSL présent et valide mais doublé ;
- une machine avec un certificat SSL présent et valide, mais tournant sur un autre port que 443 ;
- une machine avec un certificat SSL présent et valide avec changement de configuration (tests sur la 1e machine - SSLCiphers, SSLProtocols, ...).

Pour créer un certificat d'autorité SSL :

- On génère un couple clé privée, clé publique RSA de 1024 bits
  `$ openssl genrsa 1024 > cert.key`
- On créer un certificat auto-signé (peu nous importe la nature du certificat ici) - ce sera notre certificat racine
  `$ openssl req -new -x509 -days 365 -key cert.key > cert.crt`
- Ensuite, on fait une demande de certificat auprès du certificat autorité
  `$ openssl req -new -key test.key > test.csr`
- Enfin, on peut signer ce dernier avec notre certificat racine
  `$ openssl x509 -req -in test.csr -out test.crt -CA cert.crt -CAkey cert.key -CAcreateserial`

Pour révoquer un certificat :

- `$ openssl ca -revoke test.crt -cert test.crt -keyfile test.key -config openssl.cnf`
- `$ openssl ca -gencrl -config openssl.cnf -crldays 7 -out listcrl.crl`

Il ne faut pas oublier de concaténer les certificats, il sera nécessaire dans la configuration de notre serveur web :

— `$ cat test.crt cert.crt > concatcert.crt`

Pour ce qui est de la preuve de F, nous avons un jeu de tests avec des arbres comportant un nombre de nœuds impairs ou pairs, et comportant des nombres premiers communs ou non.

5.6 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 scan 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 scan, 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. Ainsi nous avons lancé les scans importants les vendredi soirs, en utilisant la prise ethernet extérieure.

Serveur de calcul de l'Université : ce serveur nous a permis de factoriser les moduli lors du deuxième scan (plus conséquent - environ 500 000 certificats), et de gagner du temps sur l'avancement de notre projet.

5.7 Choix des langages de développement

5.7.1 Langage C et librairie GnuMP

Nous avons décidé avant de débuter le projet de faire une étude en benchmark-test [6] [5] [7] (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 parmi lesquelles :
— C ;
— C++ ;
— Java ;
— Perl ;
— Python ;
— Ruby.

Nous avons au final décidé d'utiliser le langage C avec la librairie GnuMP [2] pour la gestion des grands entiers, et le compilateur CMAKE. Le langage C est l'un des plus rapides en temps d'exécution, il est également l'un de ceux qui consomment le moins de mémoire. Il est également très performant sur la gestion des grands entiers.

Le python et le C++ étaient également de très bons choix, mais les développeurs ont une meilleure maîtrise du C.

5.7.2 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.

5.7.3 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.

5.8 IHM

5.8.1 Site Web

Nous avons établi 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 réalisé 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).

5.8.2 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.

5.9 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 plusieurs 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

Rétrospective :

Nous pouvons retenir plusieurs points importants sur ce projet.
Premièrement, malgré les nombreuses études démontrant les vulnérabilités des certificats circulant sur Internet, elles persistent toujours et il devient urgent de bien sécuriser sa machine contre des attaques aussi efficaces et rapides (quelques jours à peine) face à des serveurs mal sécurisés.

Deuxièmement, nous avons en l'espace de quelques jours trouvé plusieurs vulnérabilités sur le code d'OpenSSL, et nous pouvons en conclure qu'il est nécessaire d'adopter rapidement une politique d'action plutôt qu'une politique de réaction.

Troisièmement, nous avons fait face à un code très mal documenté, avec un paramétrage par défaut contestable, et aucune classe de haut niveau. Il est au minimum important de retravailler la documentation.

Enfin, sur deux semaines nous n'avons pu faire qu'un audit de surface. Un audit complet, en profondeur, prendrait beaucoup plus de temps mais serait vraiment un sujet de projet intéressant.

Apports du projet :

Le projet nous a apportés plus d'expériences dans plusieurs domaines (développement applicatif, scripts, web, base de données, réseaux, cryptographie, ...). Nous avons cherché à rendre un projet soigné, dont chaque partie serait liée et serait directement accessible sur une même plateforme.
Notre étude sur la deuxième partie nous a permis d'analyser en détail des articles scientifiques ; il s'agissait d'un bon exercice de style pour notre futur professionnel. De plus, ce projet a attisé notre curiosité, et sachant qu'il ne faut pas se fier aux systèmes de sécurité mis en place, cela nous a donné l'envie à tous de continuer nos recherches sur les problématiques liés à la sécurité.

Axes d'améliorations :

Bibliographie

[1] 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.

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

[3] 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.

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

[5] 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.

[6] 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.

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

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