MISC

LE MAGAZINE DE LA SÉCURITÉ INFORMATIQUE MULTIPLATEFORME !

N° 98
JUILLET / AOÛT
2018

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

L 19018 - 98 - F: 8,90 € - RD

CRYPTOGRAPHIE : Contre-mesures / SCA

Attaques side channel contre le standard de chiffrement AES
p. 72

CODE : CORS / Pentest

Sécurité du Web : découvrir le fonctionnement de Cross Origin Ressource Sharing p. 67

ORGANISATION : Définition / IA

Analyse de risques : comprendre et modéliser la notion de surface d'attaque
p. 78

DOSSIER : 2FA / MFA

AUTHENTIFICATION :
ENFIN LA FIN DES MOTS DE PASSE ? p. 30

1 - Tour d'horizon de l'authentification multi-facteurs
2 - Retour d'expérience en bug bounty sur des failles d'authentification
3 - Découvrez le nouveau protocole d'authentification du W3C : WebAuthn
4 - OpenID Connect : anatomie d'une usurpation d'identité

MALWARE CORNER

Analyse du bootloader de NotPetya avec Miasm
p. 12

FORENSIC CORNER

Création d'alertes personnalisées dans votre SIEM avec ElasticAlert
p. 22

IoT CORNER

Explorez PiRanhaLysis, le framework pour l'analyse d'IoT
p. 06

esiea GROUPE

FORMATIONS OPÉRATIONNELLES EN SÉCURITÉ

/ Formations présentielles - Campus Paris V^e

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

/ Candidatures MS-SIS : en cours

FORMATION À PLEIN TEMPS

6 mois de pédagogie, puis 6 mois en entreprise

Prochaine rentrée : octobre 2018

MASTÈRE SPÉCIALISÉ SÉCURITÉ DE L'INFORMATION ET DES SYSTÈMES

(MS-SIS : 740 heures de cours)

Accrédité par la Conférence des Grandes Écoles

_Réseaux
_Sécurité des réseaux, des systèmes d'information et des applications
_Modèles et Politiques de sécurité
_Cryptologie

Mastère Spécialisé labellisé SecNumedu par l'ANSSI

ASM / C / crypto / firewalling / forensic / GPU / Java / malware / OSINT / pentest / python / reverse / SCADA / scapy / SDR / suricata / vlro / vuln / web...

/ Candidatures BADGE-RE et BADGE-SO : en cours

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

Prochaine rentrée : février 2019

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 / kernel / mlasm...

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

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

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 : https://www.miscmag.com
https://www.ed-diamond.com

Directeur de publication : Arnaud Metzler
Chef des rédactions : Denis Bodor
Rédacteur en chef : Cédric Foll
Rédacteur en chef adjoint : Émilien Gaspar
Secrétaire de rédaction : Aline Hof
Responsable service infographie : Kathrin Scali
Réalisation graphique : Thomas Pichon
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

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

Membre April
www.april.org

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.

https://www.miscmag.com

RETROUVEZ-NOUS SUR :

@MISCleMag

@miscredac et/ou @editionsdiamond

p.65/66 **Découvrez TOUS NOS abonnements MULTI-SUPPORTS !**

ÉDITO

LA SÉCURITÉ EST UN SUCCÈS (ET UN PROCESSUS D'AMÉLIORATION CONTINU)

Pour rebondir sur la note d'optimisme de bon aloi de Pappy à propos des 15 ans du SSTIC **[I]**, ne nous enfermons pas dans la posture du berger criant « au loup » en matière de sécurité des systèmes d'information, car globalement le niveau de protection des systèmes d'information n'a jamais été aussi élevé et il est en constante progression.

Au risque de passer pour un vieux con, quand je vois les jeunes s'écrier « la sécurité est un échec » parce que leur brosse à dents se connecte sur un serveur dont le certificat n'est pas certifié A+ chez SSL Lab, je mesure le chemin parcouru ces quinze dernières années. Vous n'imaginiez pas à quel point le niveau de sécurité aujourd'hui semblerait stratosphérique pour un informaticien du début des années 2000 faisant un saut de 15 ans dans le futur.

Imaginez qu'à cette époque pas si lointaine, toutes les applications clients serveurs développées en Delphi ou Access, les bouts de fichiers Excel, se retrouvaient du jour au lendemain propulsées sur le Web, écrites sur un coin de table par un collègue qui avait acheté « PHP pour les nuls » quelques jours plus tôt. Certaines tenaient sans se faire p0wner quelques semaines grâce à magic_quotes_gpc laissé activé par défaut... jusqu'à ce que quelqu'un se plaigne de la présence d'antislash sur les messages déposés dans le livre d'or.

Imaginez qu'il était possible de faire un TP sur les stack-overflows à des étudiants disposant de rudiments d'assembleurs. Pas besoin de se compliquer la vie avec l'adresse aléatoire de la stack, le bit NX ou les canaris. Un coup de gdb pour la stack, un shellcode trouvé sur le net et en trois heures tout le monde réussissait son remote exec.

Côté réseau, c'était encore l'opulence des adresses IPv4 et tout le monde se fichait des problèmes de vie privée. Il était possible dans beaucoup de structures d'être en adressage public complet afin de ne pas trop avoir à se compliquer la vie avec des vues DNS et du NAT. Les utilisateurs naviguaient l'esprit léger pendant leur pause méridienne sur les TGP (*Thumbnail Gallery Post*, un peu l'ancêtre de Pornhub) depuis leur IP fixe dont le reverse DNS contenait parfois, accolé au nom de domaine de l'entreprise, leur identifiant. La fonctionnalité « navigation privée » n'existait pas encore et dans l'imaginaire collectif, comme à Vegas, ce qui se passait sur Internet restait (et pour cause !) sur Internet.

Ajoutons à cela aucun chiffrement, des outils d'administration se basant uniquement sur l'adresse IP de l'administrateur (les fameux r* tellement plus pratiques que telnet, cet outil pour paranos de la sécurité) et un réseau bien à plat avec tout le monde dans un gros LAN, serveurs compris. Rappelons-nous qu'en 2000, 802.1q c'était un peu comme Shortest Path Bridge aujourd'hui, tout le monde en avait entendu parler, mais seuls les plus barbus d'entre nous y avaient goûté. Et pour faire bonne figure quelques ACL stateless sur le routeur WAN pour éviter que le port 139 soit accessible sur Internet et que le patrimoine informationnel, les mp3 sur le PC du secrétariat, soit dispersé à tous les vents.

Pour compléter, côté système, nous étions en pleine mode des pizza box (serveurs 1U) rackés à la chaîne sur lesquels tournaient dans le DC moyen au moins un exemplaire de toutes les distributions Linux de l'époque (il fallait bien les comparer) avec un niveau de mise à jour très variable. La virtualisation n'existait pas, encore moins les conteneurs, et faute d'Ansible les mises à jour, quand il y en avait, se faisaient en Expect, merveilleuse extension du non moins magnifique langage TCL. Un dialecte propre à chaque équipe système s'était aussi constitué pour la gestion du versionning des fichiers de configuration tels que « .backup », « .sos », « .dernierereversionquifonctionne », « .anesurtoutpaseffacer ».

Alors je veux bien entendre que l'on croise encore des horreurs et qu'il reste des anachronismes qui nous font bondir. Qu'en 2018 il est désespérant que des langages aussi complexes et dangereux que le C soient toujours autant utilisés sur des briques critiques. Côté réseau, qu'Ethernet, ce protocole bête à manger du foin, et sur lequel il faut empiler des briques à n'en plus finir pour combler les lacunes de sécurité (l' « arp cache poisoning » putain !) soit toujours là et qu'aucun protocole ne semble pouvoir lui succéder à moyen terme. Ou encore que des décideurs soient prompts à déranger leurs équipes en plein week-end pour leur faire patcher immédiatement une vulnérabilité médiatisée sur WPA, mais ne voient aucun problème à se ruer sur le premier portail captif disponible dans les hôtels et aéroports de tous pays.

Alors globalement, dans notre mission qui est la réduction de risque au travers de mesures techniques, il n'y a pas de quoi rougir. Des SI se font toujours pirater, et ce sera certainement encore très longtemps le cas, mais imaginons un instant le carnage que serait le cyber espace avec les besoins de sécurité actuels (à peu près tout est informatisé) ainsi que le niveau des menaces si nous étions restés au niveau de sécurité des années 2000 ?

Cédric FOLL / cedric@miscmag.com / @follc

[I] https://www.linkedin.com/pulse/diff-u-sstic2003-sstic2018-pourquoi-les-vieux-sont-encore-raynal/

SOMMAIRE

MISC MAGAZINE
N°98

IoT CORNER

MALWARE CORNER

FORENSIC CORNER

CODE

CRYPTOGRAPHIE

ORGANISATION & JURIDIQUE

30 DOSSIER

AUTHENTIFICATION : ENFIN LA FIN DES MOTS DE PASSE ?

PIRANHALYSIS : RENDRE L'ANALYSE IOT FACILE ET REPRODUCTIBLE

Rayna STAMBOLIYSKA – rayna@rs-strategy.consulting
Consultante gestion des risques et investigation numérique

mots-clés : IOT / OPEN SOURCE / AUTOMATISATION / KALI / RASPBERRY PI

Ce n'est un secret pour personne : les objets connectés (IoT) et la sécurité, ça fait 10. Cependant, malgré un marché en plein essor et des fabricants toujours aussi peu versés dans les fondamentaux de la sécurité IT, peu d'outils existent permettant d'assurer des analyses de qualité, régulières et faciles de ces objets. Cet article présente une suite de tels outils libres laquelle, même si elle est encore jeune, s'annonce prometteuse et invite les contributions.

Quand des casinos se font plumer grâce à leurs aquariums connectés **[0]**, on comprend que la sécurité prend l'eau... Le point de départ et la motivation pour une telle suite d'outils est le projet personnel de sensibilisation à la sécurité des IoT que votre serviteure a initié il y a déjà plus d'un an, autour des sextoys connectés **[1]** et l'IoT à visée médicale. Ces objets, à la sécurité trop souvent défaillante, sont de plus en plus présents dans notre quotidien et permettent la collecte et autres traitements de données à caractère personnel souvent sensibles. La méconnaissance habituelle des pratiques et exigences de sécurité de la part des fabricants fait que, même lorsqu'une analyse est faite et des vulnérabilités décrites et qualifiées, la correction et le suivi ne sont pas au rendez-vous.

Nous ne parlerons pas de l'acculturation nécessaire de ces start-ups et autres vendeurs ici. De par ses contraintes de marché, l'industrie de l'IoT repose sur le bon sens et la bonne culture générale de ses développeurs, soit sur des efforts beaucoup trop aléatoires pour être considérés comme systématiques. L'ambition du projet technique présenté ci-dessous est de fournir des moyens simples et peu chers d'analyse d'IoT permettant à un plus grand nombre d'acteurs de s'en saisir et de les inclure dans leur processus de release. Ce projet est un effort commun d'Esther Onfroy, experte systèmes embarqués et ingénierie inverse, et votre serviteure **[2]**.

1 Étudier pour mieux protéger : oui. Mais comment ?

Quand on se lance dans l'analyse IoT, on réalise que les outils adéquats font défaut. Sans aborder l'ingénierie inverse de matériel, appréhender et correctement décrire les éléments logiques est un défi à part entière. Plus précisément, il s'agit de saisir la complexité de l'écosystème : le plus souvent, un IoT ne se connecte pas directement à Internet, mais passe par une connexion *Bluetooth Low Energy* (BLE) pour se connecter à un smartphone à partir duquel, *via* une application, l'objet est « connecté » à Internet.

Ainsi, lorsque l'on souhaite étudier la sécurité d'un IoT, on ne peut qu'étudier l'écosystème. Pour ceux et celles qui se demandent pourquoi cette complexité : le modèle économique d'un IoT repose sur l'optimisation des coûts de fabrication, de l'autonomie et d'une simplicité d'usage. Implémenter les composants matériels et logiciels du modèle OSI pour permettre une connectivité directe à Internet représente ainsi une charge trop importante pour chacun de ces trois facteurs.

1.1 L'approche classique

Donc, analyser un IoT, c'est comprendre la façon dont l'objet se connecte en Bluetooth, comment il est contrôlé par l'application mobile compagnonne et les interactions de celle-ci avec le serveur distant. Pour les plus courageux-euses, une étape complémentaire d'analyse dynamique peut intervenir, où vous goûtez au plaisir d'ingénierie inverse de paquets BLE. Récupérer les logs BLE ne suffisant pas, il faut aussi décompiler l'application et capturer les trames réseaux correspondantes à ses communications avec son serveur distant **[3]**. Vu le focus de cet article, on ne parlera ni de BLE, ni de décompilation.

À l'heure actuelle, la façon la plus simple et probablement la plus utilisée de capturer du trafic réseau d'un IoT, connecté à travers du BLE à un smartphone et donc à Internet, est la suivante **[4]** :

1. Connecter le smartphone en Wi-Fi sur le même réseau que l'ordinateur ;
2. Lancer `mitmproxy` sur l'ordinateur ;
3. Côté smartphone, éditer les paramètres de connexion Wi-Fi pour préciser l'adresse IP et le port de `mitmproxy` dans la section `proxy http`.
4. Accéder depuis le smartphone à http://mitm.it, installer le certificat X.509.
5. Lancer la capture depuis l'ordinateur, suer à grosses gouttes qu'elle ne soit pas polluée, que les terminaux mobilisés soient correctement configurés, que l'on n'ait pas bousillé la configuration réseau de l'endroit où on est, puis se plonger dans Wireshark, etc.

L'insuffisance d'outils appropriés et le manque de documentation de la part des fabricants et éditeurs, combinés avec la complexité de l'écosystème de test et d'étude rendent même les approches plutôt simples, incertaines et ajoutent de l'impondérable.

En effet, comment s'assurer que les mêmes effets sont produits par les mêmes causes alors qu'on utilise des outils de capture, d'analyse et de conservation différents à chaque étude et ce, à différents moments de l'année ? Dans des conditions si peu contrôlées, comment s'assurer que les insuffisances décrites ne résultent pas d'un effet de bord quelconque, d'une pollution des trames, etc. ? Le problème ne se simplifie pas si l'on prend en compte la difficulté d'émuler du Bluetooth et de simuler le comportement d'un objet connecté à un smartphone. C'est faisable, mais c'est du bricolage et les résultats sont plutôt médiocres. Bien sûr, acheter un smartphone reconditionné est une solution, mais c'est un objet avec plein d'autres fonctionnalités (sans même entrer dans la discussion de l'hétérogénéité des plateformes Android existantes et leurs potentiels effets). Enfin, cette approche a un coût, notamment si on commence à aborder des analyses iOS : même reconditionnés, les iPhone restent onéreux.

À ces contraintes s'ajoute le fait d'avoir une approche boîte noire sur chaque élément de l'écosystème : le site du fabricant, l'objet, le smartphone et souvent l'application. Tous ces éléments font que nous ne pouvons garantir la reproductibilité des analyses, encore moins les faire évoluer en études longitudinales (*i.e.*, suivre avec la même méthodologie exigeante l'évolution d'un écosystème IoT donné dans le temps). C'est à la question de la reproductibilité et de suivi à bas coût que la suite Pi*, décrite dans cet article, vise à répondre.

1.2 La suite Pi*

La suite d'outils Pi* présentée dans cet article fournit des outils simples, appropriés et adaptables pour faciliter l'analyse IoT. Elle se construit autour de trois modules principaux :

- la collecte : PiRogue ;
- l'automatisation de l'installation de l'application et de la capture : PiRanha ;
- la définition d'expérimentations avec un smartphone, une application et un IoT ainsi que l'analyse des trames réseau : PiPrecious.

Ces trois modules sont présentés de façon détaillée ci-dessous et illustrés à travers quelques exemples issus de l'analyse d'un sextoy connecté et son application associée. Comme le premier module à être créé et permettant de débuter une analyse est le PiRogue, nous avons décidé d'assumer la terrible propension à la blague pourrie et garder le « Pi » comme élément incontournable de la convention de nommage.

La suite Pi* présente quelques avantages intéressants par rapport aux outils plus ou moins bricolés que l'on peut mobiliser. Déjà, il s'agit d'un ensemble de modules construits à partir de logiciels libres et open source. Ils sont imbriqués de façon logique et adaptable permettant d'automatiser une partie non négligeable d'une analyse, laissant du jus de cerveau disponible pour les points qui nécessitent une attention humaine particulière (désoffuscation, documentation d'API REST, etc.).

Étant de formation scientifique, la reproductibilité des résultats est une exigence fondamentale de mes projets. Pour pouvoir prétendre à une étude de qualité, il est primordial d'assurer que la méthodologie mobilisée pour la collecte et les analyses puisse être déployée par n'importe qui et produire les mêmes résultats sur le même objet d'analyse. Ainsi, la formalisation d'une suite d'outils, la Pi* en l'occurrence, est primordiale, car elle permet d'asseoir une méthodologie et d'appliquer les mêmes approches à n'importe quel écosystème IoT étudié. *In fine*, si des effets de bords venaient à être identifiés, ils seraient systémiques : donc, connus et potentiellement corrigeables. Une telle approche est évidemment à envisager avec ses limites : le comportement d'une application peut différer à l'inscription et après, limitation importante, car on sait qu'il est impossible de refaire l'inscription une 2ème fois. De même, il faut garder à l'esprit la non-reproductibilité des opérations matérielles sur l'objet lui-même.

Enfin, il est illusoire d'imaginer que tous les fabricants d'IoT vont automagiquement se doter de conditions spécifiques de tests. L'IoT est une industrie où l'on capitalise sur le volume : ajouter une question de coût des équipements de sécurisation est une barrière supplémentaire. D'où la nécessité d'avoir une infrastructure d'analyse à bas coût, mobilisant des logiciels et du matériel open source.

2 Pour l'amour du geek

Le PiRogue est un boîtier basé sur Raspberry Pi 3 permettant d'intercepter le trafic HTTP(S) et d'en assurer la collecte. Il permet de verser l'activité réseau d'un appareil dans les modules d'analyse.

Fabriquer le boîtier à l'aide d'une imprimante 3D est possible en suivant les instructions **[5]**. Au Raspberry Pi s'ajoute un dongle, reconnu par le système, qui fait office de carte Wi-Fi. Il ajoute une deuxième interface Wi-Fi au PiRogue (la `wlan1`, la première étant `wlan0`, nativement fournie par le Raspberry Pi). L'écran OLED,

situé sur le dessus du PiRogue, permet d'afficher quelques éléments basiques. Enfin, une carte SD de 32GB permet d'installer la distribution Kali Linux.

Par défaut, le PiRogue crée un point d'accès Wi-Fi avec le SSID PiRogue utilisable sans mot de passe (et on prévient les voisins d'aller jouer ailleurs, oui). Il est diffusé sur l'interface **wlan1**. La connexion Internet est dynamiquement partagée avec ce réseau Wi-Fi : une fois configuré, le PiRogue partagera automatiquement la connexion active, soit **wlan0**, soit **eth0**. Autrement dit, on peut choisir de le connecter au réseau local en Wi-Fi ou en Ethernet.

2.1 De la configure pour le petit-déjeuner

Pour configurer votre PiRogue, c'est relativement simple. Tout d'abord, installez Kali Linux sur une carte microSD en suivant la documentation officielle **[6]**. Pour préparer le PiRogue, il faut le connecter en Ethernet sur votre réseau local. Une fois n'est pas coutume : pour vous y connecter, le login est **root** et le mot de passe est **toor**.

Pour installer la personnalisation, il est nécessaire de retenir l'adresse IP de l'interface **eth0**, celle connectée à votre réseau local :

```
ip a s eth0
```

Afin de disposer de l'intégralité de la carte SD, il est nécessaire de redimensionner la partition principale :

```
parted /dev/mmcblk0 resizepart 2 -- -1
resize2fs /dev/mmcblk0p2
```

Il faut maintenant mettre à jour la distribution :

```
apt update
apt dist-upgrade
```

Et, bien sûr, il faut regénérer les clés SSH par défaut :

```
apt install sshpass
rm /etc/ssh/ssh_host_*
dpkg-reconfigure openssh-server
systemctl restart ssh
```

Enfin, n'oubliez pas de changer le mot de passe administrateur :

```
passwd root
```

Ensuite, on installera Ansible (logiciel libre de configuration et gestion automatisée) sur l'ordinateur (suivre la doc officielle **[7]**). Vous pouvez désormais personnaliser le PiRogue : sur votre ordinateur, clonez le dépôt **PiRogue-ansible** :

```
git clone https://github.com/PiRanhaLysis/PiRogue-ansible.git /tmp/PiRogue
cd /tmp/PiRogue
```

```
// éditez le fichier inventory.ini pour préciser l'adresse IP de votre
PiRogue juste après
// ansible_host=
// Le résultat devrait donner :

pirogue_1 ansible_host=192.128.0.12  ansible_ssh_user=root

// si vous souhaitez installer la personnalisation sur plusieurs
PiRogue, cela se fait en une fois // en ajoutant une ligne par
instance :

pirogue_1 ansible_host=192.168.1.123  ansible_ssh_user=root
pirogue_2 ansible_host=192.168.1.124  ansible_ssh_user=root
pirogue_3 ansible_host=192.168.1.125  ansible_ssh_user=root
```

Vous pouvez désormais lancer l'installation :

```
ansible-playbook -i inventory.ini --ask-pass pirogue.yml
```

Le mot de passe de l'utilisateur **root** de votre PiRogue sera demandé à l'installation.

Il faut désormais redémarrer la station :

```
reboot
// tous les fichiers de personnalisation sont dans le dossier /usr/
share/PiRogue.
```

L'écran OLED affichera :

- les informations d'utilisation disque et RAM ;
- les adresses IP des interfaces **eth0**, **wlan0** et **wlan1**, respectivement ;
- le SSID du Wi-Fi dédié à la capture ;
- l'indication de capture (*not capturing* si pas de capture ; *capturing* dans le cas contraire).

La station est prête à l'utilisation.

2.2 MITM, moi non plus

L'interception, la collecte et l'exploitation de trafic en clair sont simples. Cependant, comme les fabricants d'IoT découvrent de plus en plus le HTTPS, il est également nécessaire d'avoir un outil permettant l'étude de trames chiffrées.

Afin de pouvoir intercepter et exploiter du trafic chiffré (HTTPS donc), le PiRogue utilise l'outil **mitmproxy** **[8]**. Pour rappel, il s'agit de simuler une attaque MITM (*man-in-the-middle* ou personne-au-milieu), soit de se substituer au destinataire légitime du trafic en ayant ainsi les moyens de le récupérer en clair et de l'exploiter. Ainsi, **mitmproxy** permet d'intercepter, étudier, modifier et rejouer du trafic HTTP/1, HTTP/2, etc. ainsi que divers autres protocoles protégés par une couche de chiffrement SSL et/ou TLS.

L'utilisation de **mitmproxy** sur un smartphone Android n'est pas toujours aisée. C'est pourquoi nous nous sommes attachées à ce que le PiRogue en permette un usage fluide grâce à une configuration simplifiée. Pour ce faire, il utilise un « proxy transparent » pour rediriger le trafic HTTP(S) du réseau local vers le port dédié en permettant la capture :

```
iptables -t nat -A PREROUTING -i wlan1 -p tcp --dport 80 -j
REDIRECT --to-port 8080
iptables -t nat -A PREROUTING -i wlan1 -p tcp --dport 443 -j
REDIRECT --to-port 8080
// pour l'IPv6, on utilise les mêmes commandes en remplaçant
iptables par ip6tables
```

L'utilisation de ce « proxy transparent » enlève la nécessité de paramétrer le Wi-Fi du smartphone et de préciser l'adresse IP et le port que **mitmproxy** utilise (cf. section 1.1). Afin de visualiser le trafic en cours de capture, le PiRogue intègre **mitmweb**, une interface dédiée.

Ainsi, le cheminement d'interaction et d'interception à la première utilisation est comme suit :

1. Le PiRogue est connecté en Ethernet au réseau local ; son adresse IP est par ex. de type **192.168.0.42** (dans mon cas) ;
2. Le PiRogue crée un réseau Wi-Fi sur lequel il est identifié par une adresse IP qui lui est propre (par ex., **10.0.0.1**) ;
3. La connexion au PiRogue se fait en SSH. La création du « proxy transparent » se fait en étant qu'administrateur :

```
root@pirogue:~#  sh /usr/share/PiRogue/proxy/transparent.sh
```

Le script exécute les commandes **iptables** (cf. *supra*).

4. Pour finaliser la configuration, lancer **mitmproxy** :

```
root@pirogue:~#  mitmproxy -T -z
```

On connecte ensuite son smartphone de test sur le Wi-Fi du PiRogue et on se rend sur http://mitm.it/. Cliquez sur l'icône Android pour récupérer le certificat. Une fenêtre de dialogue sur le smartphone vous invite à lui donner un petit nom. Une fois cela fait, le certificat est installé sur le smartphone et la capture du trafic peut commencer.

5. L'écran du PiRogue indiquera si vous faites une capture (*capturing*) ou pas ainsi que les différentes adresses IP assignées. Grâce à l'interface de **mitmproxy**, vous pouvez suivre les requêtes en temps réel dans votre console.
6. L'utilisation du « proxy transparent » peut également être arrêtée ; pour cela, exécuter :

```
root@pirogue:~#  sh /usr/share/PiRogue/proxy/stop_transparent.sh
```

Une vidéo de démo est disponible, en anglais et en français **[9]**.

3 Une plate-forme pour les capter tous et correctement les gérer

Une fois le PiRogue installé et configuré, on peut jouer. On verra donc comment automatiser la capture et la façon dont on stocke les données versionnées.

3.1 C'est une PiRanha, mais il n'y a pas lieu de s'inquiéter

PiRanha est l'outil (en ligne de commandes) qui permet d'exécuter une expérimentation définie à l'aide de l'interface de PiPrecious. Autrement dit, PiRanha se charge de reproduire de façon automatique des conditions d'expérimentation en prenant comme base les références stockées dans PiPrecious. Ce faisant, PiRanha effectue aussi la capture complète du trafic réseau. Pour le moment, une expérimentation est définie par un smartphone, une application mobile et un objet connecté (optionnel). Il est possible d'exécuter une même expérimentation plusieurs fois, chaque exécution étant nommée une session. À chaque session sont associés les fichiers de capture PCAP, FLOW et Bluetooth. Ainsi donc, avec PiRanha, on peut lancer les captures réseau, démarrer l'application mobile, etc.

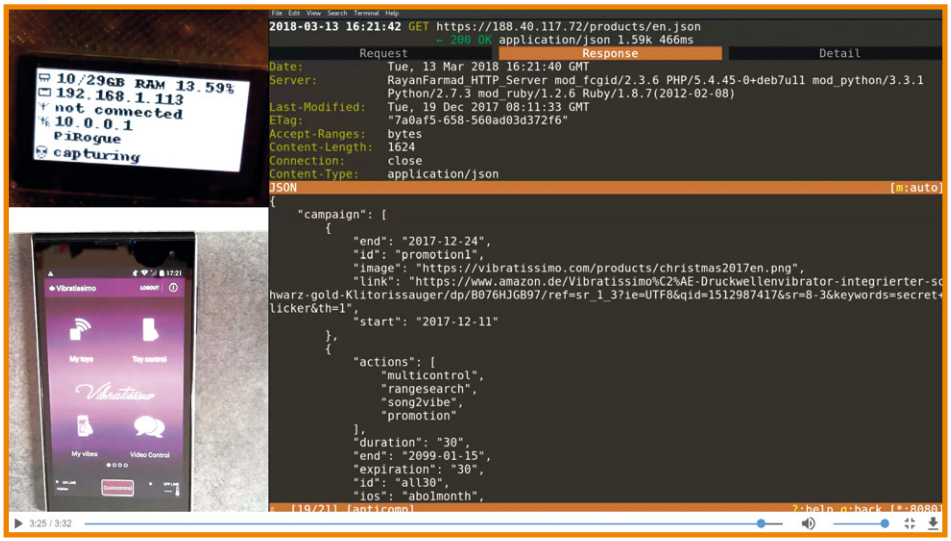


Fig. 1. Exemple d'utilisation du PiRogue en phase de capture de trafic (en haut à gauche) provenant de l'application Vibratissimo installée sur un smartphone Android de test (en bas à gauche). Le reste de l'écran est la sortie console.

Domain	Secure	Leaked Data	Country	Region	City	Organization
data.flurry.com	True	• smartphone_aid • application_handle • application_version • smartphone_attributes	United States	California	Sunnyvale	AS26101 Yahoo!
vibratissimo.com	True	• application_version	Germany			AS24940 Hetzner Online GmbH
app-measurement.com	True	• advertiser_id • application_handle • application_version	United States	California	Mountain View	AS15169 Google LLC

Fig. 2 : Interface PiPrecious présentant un résumé des éléments principaux. La colonne Leaked Data indique les résultats d'exécution des règles. On voit que les éléments identifiants du smartphone utilisé sont consommés par le destinataire data.flurry.com, soit Flurry, une société de publicité programmatique sur mobile détenue par Yahoo! et AOL.

PiRanha permet, entre autres, de récupérer de nombreuses informations sur le smartphone pour l'ajouter à PiPrecious. Ainsi, chaque composant d'une expérimentation (smartphone, application, capture de trafic, etc.) est richement décrit et peut être identifié avec une sorte d'empreinte unique. Cette empreinte permet de formuler des règles grâce à l'intégration du moteur Yara (cf. ci-dessous) et fluidifie la détection de données transmises et de leurs destinataires.

Pour ne pas s'encombrer d'un client lourd, mais permettre une prise en main plus simple y compris par des personnes peu habituées à l'exercice, une interface web permet d'accéder aux données et de naviguer dans les différents composants et leurs métadonnées associées. Toutes les informations générées et collectées sont envoyées à PiPrecious.

3.2 PiPrecious

La gestion des données collectées se fait par le module PiPrecious et est inspirée du modèle récemment présenté par le chercheur belge Xavier Mertens **[10]**. Ce modèle est entièrement basé sur des logiciels libres (Moloch, ElasticSearch, etc.), permettant de versionner, indexer et requêter un grand nombre de trames réseaux. À l'heure actuelle, PiPrecious n'intègre pas Moloch **[11]** : ce dernier ne permet de manipuler que des PCAP. Or, la suite Pi* s'intéresse avant tout aux informations pouvant être extraites des captures utilisant **mitmproxy**. Le choix technique d'utiliser un serveur Minio pour le stockage des informations collectées est fait pour privilégier la simplicité et pour éviter de lier structurellement la suite Pi* à un outil tel que Moloch qui est extrêmement exigeant en termes de ressources. Ainsi, nous réfléchissons actuellement à la meilleure façon d'assurer la fonctionnalité proposée par Moloch (le plus probablement par l'implémentation d'une passerelle vers cet outil).

Comme mentionné plus haut, PiPrecious récupère les données captées par le PiRogue (trames réseau) et assemblées par PiRanha (métadonnées d'un smartphone, les détails d'une application Android et ceux d'un IoT) dans une expérimentation. Le suivi de versions est assuré par une labellisation de l'intégralité des ressources d'une expérimentation. À l'heure actuelle, le requêtage n'est pas implémenté, pour éviter d'alourdir inutilement l'environnement d'étude. PiPrecious permet également l'analyse des trames réseau, précisant notamment quelles données sont envoyées à quel destinataire. Enfin, PiPrecious intègre une adaptation de Yara **[12]**, un moteur open source de règle de détection initialement créé pour la détection de malware.

PiPrecious génère des règles automatiques pour le smartphone et pour l'application. Par ex., la règle de détection de numéro de série :

```
rule smartphone_aid: mobile android_id pii {
  strings:
    $aid = "af309445xxxxxxxx"
  condition:
    $aid
}
```

La règle, appelée **smartphone_aid**, est également dotée de tags (ici : **mobile**, **android_id**, **pii**). Ceux-là peuvent être définis par l'utilisateur. La règle pré-citée permet de détecter la fuite de numéro de série, c.-à-d. de savoir si un service tiers consomme cette donnée et, le cas échéant, de quel service il s'agit. PiPrecious appliquera les règles sur l'URL, les en-têtes et le corps de chaque requête HTTP. Si une correspondance est trouvée, elle sera indiquée dans le rapport final, par ex. :

Des règles sur mesure peuvent être ajoutées directement *via* l'interface d'administration. Par ex., il existe déjà une règle automatique détectant le nom de l'application, **application_handle** ; d'une façon similaire à la règle détectant qui consomme le numéro de série d'un smartphone, cela nous permet de savoir si, par exemple, Facebook reçoit cette information alors que la personne n'a pas utilisé son compte du réseau social sur l'application. On peut ajouter une règle sur mesure, par exemple suivre quels destinataires reçoivent l'identifiant de connexion à l'application (ce dernier est « ratatouf » dans notre cas) :

```
rule application_login: application login {
  strings:
    $h = "ratatouf"
```

Domain	Secure	Leaked Data	Country	Region	City	Organization
data.flurry.com	True	• smartphone_aid • application_handle • application_version • smartphone_attributes	United States	California	Sunnyvale	AS26101 Yahoo!
vibratissimo.com	True	• application_login • application_version	Germany			AS24940 Hetzner Online GmbH
app-measurement.com	True	• advertiser_id • application_handle • application_version	United States	California	Mountain View	AS15169 Google LLC

Fig. 3. Seule l'application Vibratissimo consomme la donnée login, ce qui est logique, et surtout qu'il n'y a apparemment pas de fuite de cette même donnée vers les destinataires tiers embarqués avec l'application.

```
  condition:
    $h
}
rule application_handle: application handle {
  strings:
    $h = "ag.amor.vibratissimo"
  condition:
    $h
}
```

Et le résultat est visible en figure 3.

Conclusion

La suite Pi* permet de capturer du trafic réseau, de stocker et versionner ainsi que de rejouer une analyse IoT statique. Le PiRogue est la première étape de la suite Pi*. Les modules complémentaires permettent de conserver et versionner les traces collectées.

Le PiRogue peut paraître un effort disproportionné : pourquoi créer une station physique d'interception alors que capturer du trafic avec mitmproxy est facile ? Certes, le PiRogue n'invente rien de fondamentalement différent par rapport à la méthodologie classique. Toutefois, l'avantage certain d'un point de collecte spécifique est de créer un réseau dédié rendant ainsi la capture de trafic et sa manipulation vierges de toute interférence. Ainsi, même si vous êtes plusieurs à vous connecter au même réseau local, le PiRogue isole chaque utilisateur : chacun dispose donc d'un équipement dédié à cet usage avec des outils adéquats. Ce point n'est pas à négliger, les réseaux Wi-Fi dédiés à la capture d'analyse n'existent pas nécessairement dans toutes les entreprises, notamment les start-ups commercialisant des IoT. Même si certain-es personnes lisant cet article mangent du réseau au petit-déjeuner, configurer un point d'accès Wi-Fi, faire le routage, administrer et sécuriser ce réseau ne sont pas à la portée de tout le monde. Enfin, disposer d'un réseau dédié à la capture et séparé de celui de l'entreprise, du co-working ou du domicile permet de n'observer que le trafic passant sur ce réseau dédié et de ne pas polluer les trames par celles du frigo connecté, de la bouilloire connectée postant des selfies sur Instagram et d'autres ampoules-sur-IP.

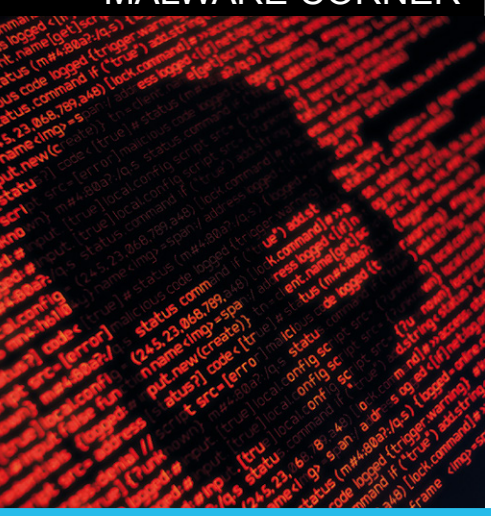
Des limites dans l'analyse du trafic sont présentes, inhérentes aux outils mobilisés. À l'heure actuelle, on ne peut analyser que du HTTP(S) ; si l'on souhaite analyser d'autres trames (du trafic UDP, des requêtes DNS, etc.), mitmproxy est inapproprié. Cependant, PiRanha envoie déjà un .pcap : l'une des prochaines étapes de développement est d'appliquer les règles dessus ou, à défaut ou par impatience, de ressortir le bon vieux Wireshark. De même, de par le fonctionnement de mitmproxy, on ne pourra pas analyser les trames réseau d'une application qui fait du *certificate pinning* (c.-à-d., s'il y a une vérification formelle que le serveur qui répond est celui légitimement attendu). Enfin, mitmproxy ne capturant que du HTTP(S), s'il permet de comprendre le fonctionnement des éventuelles API REST (ou SOAP pour les masos de l'IoT ou de l'HTTP en général) mises en jeu, il ne sera pas en mesure de capturer les protocoles spécifiques IoT (par ex., MQTT). Quant à PiPrecious et l'intégration de Yara, pour l'instant, l'exécution d'une règle ne portera pas ses fruits si le message est codé (en base64, sous forme de JSON compressé dans un gzip, etc.).

Afin de compléter les fonctionnalités des outils, nous implémentons un connecteur BLE et d'adjoindre les traces produites au reste des éléments déjà collectés. Cet ajout rendra possibles l'analyse dynamique et la complétude de l'écosystème IoT, étapes nécessaires à l'implémentation de simulateur de cet écosystème. De même, nous avons prévu la collecte de traces Android afin de mieux appréhender l'impact des appels système sur le comportement de l'écosystème IoT, l'écosystème applicatif unique à chaque utilisateur et les éventuelles fuites de données à caractère personnel pouvant avoir lieu. Bien évidemment, de tels paramètres seront extrêmement utiles à la création d'un environnement d'analyse reproductible. ■

Remerciements

Merci beaucoup à Benoît Sibaud et Axelle Apvrille pour les questions rigoureuses et qui donnent des idées.

Retrouvez toutes les références de cet article sur le blog de MISC : https://www.miscmag.com/ .



ÉMULATION DU BOOTLOADER DE NOTPETYA AVEC MIASM

Adrien GUINET – adrien@guinet.me
lafouine

mots-clés : *MIASM / MBR / BOOTLOADER / ÉMULATION / JIT / BIOS / INTERRUPTION / PETYA / NOTPETYA*

NotPetya est un célèbre malware issu de la famille Petya, apparu en juin 2017. La partie s'exécutant depuis le MBR a souvent été étudiée en statique ou en dynamique grâce au débogueur Bochs pour IDA. Une autre approche d'analyse est-elle possible ? Nous proposons ici d'émuler pas à pas le bootloader de NotPetya en utilisant Miasm.

1 Rappel

NotPetya est un malware issu de la famille bien connue des ransomwares Petya. Cette variante est apparue pour la première fois au cours du mois de juin 2017 en Ukraine. D'après Mikko Hyppönen, Chief Research Officer chez F-Secure, le vecteur d'infection serait le mécanisme de mise à jour du logiciel de comptabilité M.E.Doc, largement déployé dans les pays de l'Est.

Cette famille de malware a la particularité d'écraser le bootloader de la machine compromise afin de chiffrer une partie du disque dur à son redémarrage. Cet article utilise ce bootloader comme prétexte à un tutoriel sur l'émulation et le reverse de ces petites bêtes grâce au framework de reverse engineering Miasm. Le code associé est disponible à cette adresse : https://github.com/aguinet/miasm-bootloader/. Il contient une implémentation en Python d'un sous-ensemble des interfaces d'un BIOS de PC x86 classique. Le code a été écrit de façon à ce qu'il soit réutilisable facilement pour d'autres cas, voire pour aider au développement/débogage de bootloaders.

1.1 Travaux associés

Des articles ainsi que de nombreux blogposts ont déjà étudié les mécanismes de compromission et d'infection du MBR de NotPetya, ainsi que l'étude de l'implémentation de ses différents mécanismes cryptographiques (et de leurs défauts). En voici quelques-uns notables :

- *MISC n°86* : « Pleased to meet you, my name is Petya ! » écrit en juillet 2016 par Damien Schaeffer ;
- *MISC n°93* : « Petya or Not Petya, that is the question » écrit en septembre 2017 par Teddy et Benjamin. Il aborde de manière précise les mécanismes de démarrage d'un système et propose une rétroconception du code du bootloader ;
- Crowstrike : « Full Decryption of Systems Encrypted by Petya/NotPetya » **[1]**. Étude d'une erreur d'implémentation de l'algorithme Salsa20 dans le bootloader.

1.2 NotPetya

Cette section n'aborde que d'une manière très généraliste le cycle de vie du malware. Cela permet de mettre en lumière la partie étudiée dans cet article.

Une fois que NotPetya s'est exécuté sur la machine, il génère une clé de chiffrement AES qui va être utilisée pour réaliser la première phase de chiffrement des données de l'utilisateur. Cette clé est elle-même chiffrée avec une clé publique RSA.

Le malware vérifie ensuite que le système utilise un schéma de partition classique et, s'il en a les droits, inscrit ses propres données sur les premiers secteurs du disque (de 1 à 18, puis 32 à 34), dont le MBR dans le premier secteur. Si le système utilise UEFI (avec donc un schéma de partition GPT), le malware ne continue pas les opérations. La machine est ensuite redémarrée et le bootloader de NotPetya est exécuté : une clé Salsa20 et un nonce sont générés. Ces secrets sont utilisés pour chiffrer la *Master File Table* (MFT)**[2]** du système de fichiers NTFS. Cette structure de donnée contient les métadonnées permettant de retrouver les données

associées à chaque fichier. Cette opération prend l'allure d'un « chkdsk ». Une fois cette opération réalisée, la machine redémarre une dernière fois et s'affiche alors le message de demande de rançon.

1.3 Miasm

Miasm est un framework de reverse engineering développé en Python. Brièvement, il permet :

- d'ouvrir, modifier et générer des binaires au format PE, ELF 32, 64 LE, BE ;
- d'assembler/désassembler du code x86, ARM, MIPS, SH4, PPC et MSP430 ;
- représenter la sémantique des instructions via une représentation intermédiaire ;
- émuler du code avec un moteur de JIT (*Just In Time compilation*) ;
- simplifier des expressions, utilisées par exemple pour la désobfuscation de code obfusqué.

1.4 Pourquoi émuler NotPetya avec Miasm ? (dit autrement, pourquoi se faire autant de mal ?)

Il existe différentes façons d'émuler un bootloader. Une approche classique est d'utiliser QEMU (ou toute autre solution de virtualisation) en écrivant le bootloader sur un disque dur virtuel, mais cela permet difficilement d'instrumenter le code de ce bootloader. Une telle chose est cependant possible via le débogueur *Bochs* pour IDA. Cette approche a été adoptée par Teddy et Benjamin dans *MISC n°93*, mais aussi par Saurabh Sharma **[3]**. Cette méthode a fait ses preuves et permet de déboguer un bootloader assez facilement.

Dans l'article associé à la présentation de son outil Miasm à SSTIC en 2012 **[4]**, Fabrice Desclaux en présentait les possibilités offertes. L'une des applications proposées était l'émulation d'un bootloader.

Pouvoir émuler complètement le bootloader (jusqu'aux interruptions du BIOS) avec un framework comme Miasm donne un contrôle plus fin sur ce qu'il se passe, permet éventuellement de le désobfusquer, et/ou d'utiliser tous les outils développés dans Miasm pour aider le reverseur dans cette tâche. Il devient par exemple très simple d'instrumenter le code afin de voir les données/lues écrites sur le disque, les secrets générés, etc.

Enfin, le code du bootloader de Petya étant succinct, non obfusqué et extrêmement simple (il tourne en mode réel, en 16 bits et ne fait appel qu'à quelques interruptions du BIOS), c'est un cas d'école de choix pour prendre en main Miasm !

2 Fonctionnement d'un bootloader PC/x86

2.1 Généralités

Nous n'abordons ici que le fonctionnement d'un bootloader avec un BIOS « à l'ancienne ». Nous n'aborderons pas ici les mécanismes de démarrage utilisant l'UEFI.

Sur un PC x86, lorsque la machine démarre, le BIOS va charger le premier secteur du disque (*Master Boot Record* ou MBR) à l'adresse **0x7C00**, puis y transférer le flot d'exécution. Le MBR contient donc le code du bootloader. Pour le moment, le processeur ne supporte que les instructions 16 bits et ne peut adresser la mémoire qu'en mode réel **[5]**.

Pour rappel, un secteur de disque contient 512 octets. Par conséquent, il n'est pas possible de stocker beaucoup de code sur celui-ci. C'est pourquoi les bootloaders sont généralement conçus en plusieurs parties, aussi appelées *stages*. Ainsi le bootloader stocké dans le MBR va, par exemple, charger et exécuter le *stage* 2 contenu dans le secteur 2.

2.2 Dans le cas de NotPetya

NotPetya fonctionne exactement de cette manière. Le *bootstrap code* correspond au code assembleur ci-dessous. La primitive à l'adresse **0x7C38**, c'est-à-dire **disk_read_stage2**, va écrire le contenu des secteurs 2 à 34 (inclus) à l'adresse **0x8000** puis transférer l'exécution à cette adresse.

```
seg000:7C00 cli
seg000:7C01 xor     ax, ax
seg000:7C03 mov     ds, ax
seg000:7C05 mov     ss, ax
seg000:7C07 mov     es, ax
seg000:7C09 lea     sp, start
seg000:7C0D sti
seg000:7C0E mov     eax, 32
seg000:7C14 mov     byte ptr ds:word_7C93, dl
seg000:7C18 mov     ebx, 1
seg000:7C1E mov     cx, 8000h
seg000:7C21 call    disk_read_stage2
seg000:7C24 dec     eax
seg000:7C26 cmp     eax, 0
seg000:7C2A jnz     short loc_7C21
seg000:7C2C mov     eax, dword ptr ds:8000h
seg000:7C30 jmp     far ptr 0:8000h
```

3 Émulation avec Miasm

3.1 Installation

Le système hôte utilisé pour ces tests est basé sur Linux. Les utilisateurs de Windows 10 devraient pouvoir faire fonctionner tout cela en utilisant le très pratique *Windows*

Subsystem for Linux (WSL pour les intimes), en installant par exemple Ubuntu à travers le Windows Store **[6]**.

Nous conseillons d'utiliser la version de Miasm spécifiée dans le fichier **README** du dépôt GitHub. À l'heure où sont écrites ces lignes, la version utilisée est associée à l'empreinte **dadfaabc3fff5edb9bf4ef7e7e8c4cfc4baccb94**. Pour récupérer cette version spécifique, faire :

```
$ git clone --depth=1 -o dadfaabc3fff5edb9bf4ef7e7e8c4cfc4baccb94
https://github.com/cea-sec/miasm/
```

Nous utilisons le moteur de JIT de Miasm basé sur LLVM. Pour cela, le paquet python **llvmlite** est nécessaire. D'autres dépendances sont nécessaires à Miasm et au projet en soi. Elles sont installables directement grâce au fichier **requirements.txt** fourni :

```
$ cd /path/to/src && pip install -r requirements.txt
```

Il suffit ensuite d'installer Miasm :

```
$ cd /path/to/miasm && python ./setup.py install
```

3.2 Implémentation

Toutes les techniques décrites ici peuvent être testées grâce au script **src/emulate_mbr.py**, présent dans le dépôt GitHub cité au début de cet article. Les différentes options disponibles sont accessibles en utilisant le drapeau **--help**.

3.3 Création d'un disque de test

Nous avons réalisé nos tests avec des machines virtuelles sous Windows XP et Windows 10. L'hyperviseur utilisé importe peu (VMWare, VirtualBox), tant que le disque créé est de taille fixe et au format VMDK. Ainsi, l'émulation du bootloader se fait directement sur le disque de la machine virtuelle. Un avantage à cette méthode est qu'il n'y a pas besoin d'extraire le bootloader de la DLL d'origine du malware ou bien depuis le disque à la main.

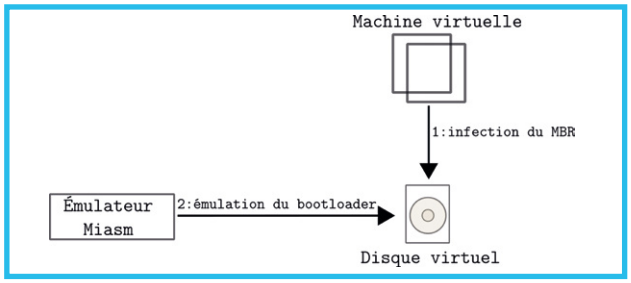


Fig. 1 : Scénario d'émulation.

Le scénario de test est le suivant :

1. Infection volontaire de la machine virtuelle avec NotPetya ;
2. Attente d'environ 10s (la machine ne doit pas redémarrer seule au risque que le chiffrement effectué par le bootloader soit lancé) ;
3. Arrêt de la machine virtuelle : le MBR est maintenant infecté ;
4. Lancement de l'émulation : chiffrement de la MFT par le bootloader puis affichage du message de rançon.

Si votre machine virtuelle n'est pas au bon format de disque, vous pouvez toujours le convertir au format RAW grâce à QEMU :

```
$ qemu-img convert -f vmdk mydisk.vmdk -O raw mydisk.raw
```

Nous fournissons de plus une image de test disponible dans le dépôt Git référencé en introduction (fichier **disk.raw.bz2**). Cette image décompressée fait environ 1Go, et contient une simple partition NTFS avec quelques fichiers de test.

Nous pouvons maintenant émuler le bootloader de NotPetya. Pour cela, il faut émuler un BIOS capable de :

- lecture/écriture sur les secteurs du disque ;
- l'affichage des caractères sur le moniteur ;
- la saisie des caractères au clavier ;
- démarrer sur un MBR (re/démarrage léger).

La suite de l'article décrit comment implémenter tout cela à travers Miasm.

3.4 Abstraction système

Nous implémentons une abstraction d'un système simple tel que vu par le BIOS. Il comprend :

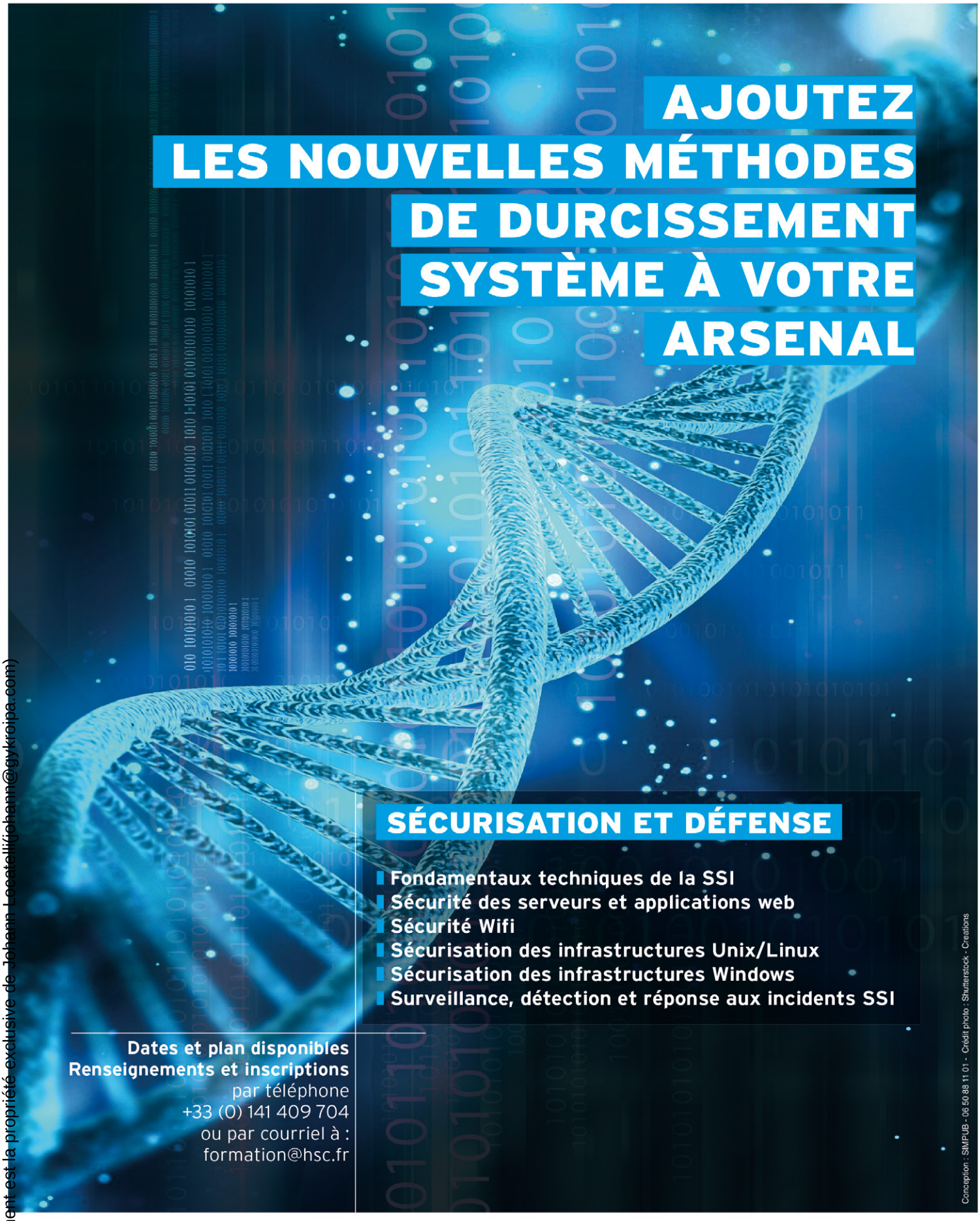
- un disque dur virtuel (classe **HardDrive**) ;
- un affichage vidéo, qui passe par un terminal classique Unix, à travers le pipe **stdout** ;
- un clavier, qui utilise le pipe **stdin** pour récupérer les frappes clavier (fonctions de **async_kb.py**).

L'abstraction est soutenue par la classe **System**, dont une instance est utilisée tout au long de l'émulation. Celle-ci est initialisée en même temps que la VM Miasm.

3.5 Initialisation de la VM Miasm

Comme annoncé en introduction, le code du MBR est chargé et exécuté à l'adresse **0x7C00** par le BIOS, qui écrira et exécutera son *stage* 2 à l'adresse **0x8000**. L'espace restant est dédié à la pile et commence à l'adresse **0x500** pour terminer à l'adresse **0x07C00** **[7]**, c'est-à-dire l'espace **[0x00000500:0x00007BFF]**.

Nous initialisons donc dans un premier temps la VM Miasm pour déclarer ces espaces mémoire, en mettant le premier secteur (MBR) à l'adresse **0x7C00** :



www.hsc-formation.fr

HSC by Deloitte.

```
HD0 = HardDrive(hd_path)
sys_ = System([HD0])
mbr = HD0.read_sector(0)
stage1_addr = 0x07C00
stage2_addr = 0x08000
jitter.vm.add_memory_page(stage1_addr, PAGE_READ | PAGE_WRITE |
PAGE_EXEC, mbr, "NotPetyaS1")
jitter.vm.add_memory_page(stage2_addr, PAGE_READ | PAGE_WRITE |
PAGE_EXEC, "\x00"*SECTOR_LEN*32, "NotPetyaS2")
jitter.vm.add_memory_page(0x500, PAGE_READ | PAGE_WRITE, "\
x00"*(0x7C00-0x500+1), "Stack")
# Affiche l'état de la mémoire
print(jitter.vm)
```

L'état de la mémoire au sein de la VM Miasm est ainsi le suivant :

```
Addr            Size            Access Comment
0x500           0x7700          RW_    Stack
0x7C00          0x200           RWX    NotPetyaS1
0x8000          0x4000          RWX    NotPetyaS2
```

NotPetya charge 32 secteurs du disque en mémoire lors de l'exécution du *stage* 1. C'est pourquoi l'espace mémoire alloué au *stage* 2 est de 32 secteurs.

3.6 Gestion des interruptions dans Miasm

Miasm permet de spécifier un gestionnaire d'interruption qu'il appellera à chaque fois que l'instruction **INT** est rencontrée. Pour cela, il suffit d'appeler la fonction **add_exception_handler** du *jitter* utilisé :

```
jitter.add_exception_handler(EXCEPT_INT_XX, lambda jitter:
exception_int(jitter, sys_))
```

Nous pouvons ensuite appeler notre implémentation Python du BIOS à partir de la fonction **exception_int**.

3.7 Support des différentes interruptions

Il convient maintenant d'écrire les gestionnaires d'interruption du BIOS. Nous distinguons 4 familles d'interruptions à implémenter pour émuler NotPetya :

- **INT 10h** : accès au moniteur (écriture de caractères, couleurs...) ;
- **INT 13h** : accès au disque (lecture, écriture, géométrie...) ;
- **INT 16h** : accès au clavier (lecture des frappes) ;
- **INT 19h** : démarrage sur le MBR d'un disque.

3.7.1 INT 13h

Pour illustrer nos propos, voici une implémentation de l'interruption **13h** avec le code de fonction **0x43** (*Extended Read Sectors From Drive*). Ce code correspond au chargement d'un ou plusieurs secteurs du disque vers la mémoire RAM. Le code Python présenté ici ne contient pas la gestion des erreurs pour des raisons de lisibilité. L'objet **sys_** correspond à l'abstraction du système proposée en section 3.4.

```
@func(disk_interrupts, 0x42)
def extended_read_sectors(jitter, sys_):
    drive_idx = get_xl(jitter.cpu.DX)
    print "Extended read sectors, drive idx 0x%x" % drive_idx

@func(disk_interrupts, 0x42)
def extended_read_sectors(jitter, sys_):
    drive_idx = get_xl(jitter.cpu.DX)
    print "Extended read sectors, drive idx 0x%x" % drive_idx

    dap = jitter.vm.get_mem((jitter.cpu.DS << 4)+jitter.cpu.SI, 16)
    dap_size, _, num_sect, buff_addr, abs_sect = struct.
unpack("<BBHIQ", dap)

    hd = sys_.hd(drive_idx)

    print("  Read %d sectors from sector %d" % (num_sect, abs_
sect))
    size = num_sect * SECTOR_LEN
    data = hd.read(abs_sect * SECTOR_LEN, size)

    jitter.cpu.cf = 0 # No error
    # AL is the number of sectors read
    # AH is the return code, 0 = successful completion
    jitter.cpu.AX = set_16bit_reg(low=int(len(data) / SECTOR_LEN),
high=0)
    jitter.vm.set_mem(buff_addr, data)
```

Il existe deux manières d'accéder à des données sur le disque, en utilisant deux adressages différents pour la même interruption **13h** :

1. L'adressage CHS (*Cylinder, Head, Sector*), utilisé par le code **02h**/**03h**. Il permet de lire/écrire un ou plusieurs secteurs en spécifiant le numéro de cylindre et de tête.
2. L'adressage LBA (*Logical Bloc Adressing*), utilisé par le code **42h**/**43h** permet de lire/écrire un ou plusieurs secteurs en spécifiant le secteur de manière absolue, c'est-à-dire à partir du début du disque en faisant abstraction des têtes et des cylindres.

NotPetya utilise l'adressage LBA nécessitant l'usage d'un DAP (*Disk Address Packet*). Il s'agit d'une structure de données décrivant quels secteurs lire et où placer les données lues.

Nous noterons qu'il existe une structure étendue permettant de charger plus de secteurs (mode LBA étendu).

Offset	Taille	Description
0	1	Taille du paquet
1	1	Toujours à zéro
2	2	Nombre de secteurs à charger
4	4	Buffer où charger les données (seg:off)
8	8	Numéro absolu du secteur à lire

Pour résumer :

1. Le DAP est parsé ;
2. Les données sont lues depuis le disque virtuel ;
3. Les données lues sont inscrites dans la page mémoire de la VM Miasm instanciée.

Le mécanisme d'écriture sur le disque est l'exact opposé : l'adresse mémoire spécifiée contient les données à écrire sur le disque.

3.7.2 INT 19h

Le second exemple choisi est l'interruption **19h** (*diskboot*), qui permet de redémarrer la machine **[8] [9]**. Elle est utilisée à deux endroits :

- dans la procédure à l'adresse **0x892E**, appelée lorsqu'une erreur fatale survient ;
- au redémarrage après le chiffrement des entrées de la MFT (adresse **0x820D**).

L'interruption **19h** est exécutée après la procédure POST (*Power On Self Test*) par le BIOS. Le MBR est alors chargé depuis le disque et le BIOS donne le contrôle au code situé à l'adresse **0x7C00**. Ce n'est donc pas un redémarrage complet, mais plutôt une instruction permettant de démarrer à partir d'un disque. Cette instruction fait partie du processus de boot après exécution du BIOS. Certains BIOS permettent de gérer les priorités de boot, tandis que d'autres bouclent sur les différents disques disponibles jusqu'à trouver sur lequel il peut démarrer.

Nous allons donc émuler cette instruction en chargeant à nouveau le MBR dans la page mémoire dédiée au *stage* 1, et en sautant ensuite sur l'adresse **0x7C00** :

```
diskboot_interrupts = FuncTable("INT 19h (diskboot)")
@func(diskboot_interrupts, 0x02)
def reboot(jitter, sys_):
    # Here, we assume only one bootable disk (index 0)
    hd = sys_.hd(0)
    mbr = hd.read_sector(0)
    jitter.vm.set_mem(0x7C00, mbr)
    jitter.pc = 0x7C00
```

3.8 Et pour quelques hacks de plus...

L'instruction **STI** (*Set Interrupt Flag*) est utilisée à l'adresse **0x7C0D**. Elle permet d'activer les interruptions masquables (*flag* **IF**, offset 9 du registre **FLAGS**). Ce *flag* n'a pas d'effet sur les interruptions non masquables. Les interruptions provenant du matériel étant entièrement émulées, Miasm ne contient pas (légitimement) de sémantique pour cette instruction.

Nous décidons donc de l'ignorer en posant un *breakpoint* à l'adresse spécifiée :

```
jitter.add_breakpoint(0x7C0D, handle_sti)
```

puis de faire passer PC à l'instruction suivante. Sachant que l'*opcode* de l'instruction **STI** (0xFB) ne fait qu'un octet, une simple incrémentation de **PC** suffit :

```
def handle_sti(jitter):
    jitter.pc += 1
    return True
```

3.9 Yippie kay yay motherfucker !

Maintenant que les différents *gestionnaires* ont été écrits, et que le code du MBR est chargé puis *mappé* dans Miasm, l'émulation peut commencer :

```
jitter.init_run(stage1_addr)
jitter.continue_run()
```

Avec le drapeau **--verbose-bios-data** (voir section 3.2), la sortie de notre programme affiche directement le contenu des lectures/écritures du disque. Par exemple, voici le contenu du second secteur (sur les 32 chargés par le bootloader à l'adresse **0x8000**) :

```
Extended read sectors, drive idx 0x0
  Read 1 sectors from sector 2
00000000: 50 FF 76 04 E8 91 0A 83  C4 0A E8 3B 07 CD 19 5E  P.v........;...^
00000010: C9 C3 6A 0E E8 39 07 5B  68 70 9C E8 C0 03 5B C3  ..j..9.[hp....[.
00000020: C8 04 04 00 56 6A 00 6A  01 6A 00 6A 20 8D 86 FC  ....Vj.j.j.j ...
00000030: FD 50 8A 46 06 50 E8 21  0A 83 C4 0C 6A 00 68 8E  .P.F.P.!....j.h.
[...]
```

Le code chargé correspond à celui du *stage* 2. Nous observons aussi facilement le chargement du secteur 32 :

```
Extended read sectors, drive idx 0x80
  Read 1 sectors from sector 32
00000000: 00 AA 92 E7 82 11 15 D3  20 96 A7 75 51 C0 36 08  ........ ..uQ.6.
00000010: E8 65 42 8C 73 9F 06 53  77 CB C5 95 60 C8 38 69  .eB.s..Sw...`.8i
00000020: 9B 0D A4 99 E0 13 12 30  79 31 4D 7A 37 31 35 33  .......0y1Mz7153
00000030: 48 4D 75 78 58 54 75 52  32 52 31 74 37 38 6D 47  HMuxXTuR2R1t78mG
00000040: 53 64 7A 61 41 74 4E 62  42 57 58 00 00 00 00 00  SdzaAtNbBWX.....
00000050: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000060: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000070: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000080: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000090: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
000000A0: 00 00 00 00 00 00 00 00  00 48 34 79 5A 73 77 56  .........H4yZswV
000000B0: 54 64 43 6B 43 77 55 68  72 31 4D 52 6D 4A 65 69  TdCkCwUhr1MRmJei
000000C0: 76 31 34 46 4B 39 6A 5A  6A 4D 36 36 4C 44 79 65  v14FK9jZjM66LDye
000000D0: 71 52 4C 64 6B 38 53 58  53 53 73 53 53 45 78 34  qRLdk8SXSSsSSEx4
000000E0: 44 51 57 4E 47 00 00 00  00 00 00 00 00 00 00 00  DQWNG...........
000000F0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
[...]
000001F0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
```

D'après la description faite de ce secteur dans **[MISC 93]**, nous pouvons en déduire :

- l'indicateur de chiffrement du disque : **0x00** ;

- la clé Salsa20 de 32 octets : **AA 92 E7 82 11 15 D3 20 96 A7 75 51 C0 36 08 E8 65 42 8C 73 9F 06 53 77 CB C5 95 60 C8 38 69** ;
- un *nonce* de 8 octets : **0D A4 99 E0 13 12 30 79** ;
- la chaîne aléatoire générée par le malware lors de son lancement sous Windows, qui est affichée dans la rançon.

Après la phase de chiffrement réalisée par le bootloader, la clé et le *nonce* sont ensuite effacés du disque via 32 réécritures successives de zéros. En outre, nous observons bien le *flag* de chiffrement du disque passé à **0x01** après le chiffrement :

```
Extended write sectors, drive idx 0x80
  Write 1 sectors at offset 32 (from memory at 0x776A)
00000000: 01 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000010: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000020: 00 0D A4 99 E0 13 12 30  79 31 4D 7A 37 31 35 33  .......0y1Mz7153
00000030: 48 4D 75 78 58 54 75 52  32 52 31 74 37 38 6D 47  HMuxXTuR2R1t78mG
00000040: 53 64 7A 61 41 74 4E 62  42 57 58 00 00 00 00 00  SdzaAtNbBWX.....
00000050: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000060: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000070: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000080: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000090: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
000000A0: 00 00 00 00 00 00 00 00  00 48 34 79 5A 73 77 56  .........H4yZswV
000000B0: 54 64 43 6B 43 77 55 68  72 31 4D 52 6D 4A 65 69  TdCkCwUhr1MRmJei
000000C0: 76 31 34 46 4B 39 6A 5A  6A 4D 36 36 4C 44 79 65  v14FK9jZjM66LDye
000000D0: 71 52 4C 64 6B 38 53 58  53 53 73 53 53 45 78 34  qRLdk8SXSSsSSEx4
000000E0: 44 51 57 4E 47 00 00 00  00 00 00 00 00 00 00 00  DQWNG...........
000001E0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
[...]
000001F0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
```

Nous remarquons aussi que le secteur 35 sert au stockage du nombre d'entrées de la MFT chiffrées, par exemple lors du chiffrement de l'entrée suivant celle de la MFT :

```
Extended write sectors, drive idx 0x80
  Write 1 sectors at offset 35 (from memory at 0x5C74)
00000000: 01 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
[...]
```

3.10 Récupération de secrets en mémoire

L'avantage de l'émulation est qu'il est possible de facilement analyser les pages mémoires. Par exemple, il est possible de récupérer la clé de chiffrement Salsa20 utilisée, et ce, même après que la machine ait fini le chiffrement de la MFT (durant le « faux chkdsk », cf. introduction).

En effet, comme vu en section 3.7.2, après que le chiffrement ait fini, le bootloader « redémarre » grâce à l'interruption **19h**. Cette interruption n'effectue pas un redémarrage complet de la machine, car le BIOS n'est pas exécuté comme dans le cas d'un *reset* ou d'un redémarrage complet. Ainsi, sur les BIOS testés, la mémoire actuelle du bootloader est conservée. Dans les cas où le BIOS s'exécuterait quand même, il y aurait malheureusement de grandes chances qu'il écrase la clé Salsa20 en mémoire par des données de sa propre pile.

Ceci étant dit, il se révèle intéressant de voir si la clé Salsa20 se trouve toujours en mémoire. Pour ce faire, nous lisons celle stockée dans le secteur 32 (voir section 3.9). Nous ajoutons ensuite un breakpoint au moment où le message de rançon est affiché, à l'adresse **0x85AF** :

```
key = HD0.read(32*SECTOR_LEN + 1, 32)
jitter.add_breakpoint(0x85AF, functools.partial(find_key_in_mem, key=key))
```

La fonction **find_key_in_mem** parcourt la mémoire de la VM Miasm et cherche ladite clé :

```
def find_key_in_mem(jitter, key):
    # Find if the salsa20 key is still in memory!
    mem = jitter.vm.get_all_memory()
      print >>sys.stderr, "\n[+] Looking for key %s in memory..." % key.
encode("hex")
    for addr,v in mem.iteritems():
        idx = v['data'].find(key)
        if idx == -1:
            continue
        print >>sys.stderr, "[+] Key found at address %s!" % hex(addr + idx)
        break
    else:
        print >>sys.stderr, "[-] Key not found in memory!"
    return True
```

Le drapeau **--hook=find_key** du programme **emulate_mbr** effectue cette manipulation.

En lançant ce bootloader instrumenté grâce à Miasm, nous obtenons :

```
$ python ./emulate-mbr.py --hook=find_key disk.raw

Repairing file system on C:
[... encryption happends here ...]

[+] Looking for key [your salsa20 key] in memory...
[+] Key found at address 0x674a!
```

Pour accélérer le processus, vous pouvez utiliser l'option **--skip-encryption** (voir section 3.1). À noter que même avec ce drapeau, l'octet de statut du chiffrement du disque sera quand même mis à vrai à la fin du chiffrement. Le drapeau **--dry** peut être utilisé pour éviter toute modification sur le disque original.

Pour savoir à quel moment la clé a été copiée sur la pile, nous pouvons mettre un point d'arrêt mémoire en écriture :

```
def print_ip(jitter):
    print(hex(jitter.pc))
    return False
jitter.exceptions_handler.callbacks[EXCEPT_BREAKPOINT_MEMORY] = []
jitter.add_exception_handler(EXCEPT_BREAKPOINT_MEMORY, print_ip)
jitter.vm.add_memory_breakpoint(0x674a, 1, PAGE_WRITE)
```

Miasm gère par défaut les exceptions mémoires en arrêtant le *jitter*. Il enregistre pour cela un *callback* sur l'exception **EXCEPT_BREAKPOINT_MEMORY**. Nous devons donc le supprimer pour le remplacer par notre propre fonction qui va afficher le pointeur d'instruction, puis mettre l'exécution en pause.

HS2
Hervé Schauer Sécurité

Formation cybersécurité technique

PROGRAMME

Introduction à la cybersécurité

ESSCYBER :
Essentiels techniques de la cybersécurité
SECUCYBER :
Fondamentaux techniques de la cybersécurité

Sécurité défensive et Réponse aux incidents

SECUWEB :
Sécurité des serveurs et des applications Web
SECUWIN :
Sécurisation des infrastructures Windows
SECULIN :
Sécurité Linux
SECUARCH :
Conception d'architectures sécurisées
SECUBLUE :
Surveillance, détection et réponse aux incidents de sécurité

Sécurité des réseaux et des infrastructures

SECUINDUS :
Cybersécurité des systèmes industriels
SECURSF :
Sécurité des réseaux sans fil
DNSSEC :
DNSSEC
SECUPKI :
Infrastructures de clés publiques
SECUPKIWIN :
Infrastructure de clés publiques Windows

Inforensique

FORENSIC1 :
Analyse inforensique Windows
FORENSIC2 :
Analyse inforensique avancée
REVERSE1 :
Rétroingénierie de logiciels malfaisants

Sécurité offensive

PENTEST1 :
Test d'intrusion
PENTEST2 :
Test d'intrusion et développement d'exploits

+33 644 014 072

www.hs2.fr

formations@hs2.fr

Nous trouvons ainsi que le code responsable de cette copie se trouve à l'adresse **0x816B**. Il est important de noter que, lorsque la fonction **print_ip** est appelée, **PC** pointe vers la prochaine instruction à exécuter, soit **0x816F**.

En analysant le code, nous voyons que l'instruction à l'adresse **0x816B** fait partie d'une boucle qui copie 32 octets provenant du secteur 32 (où se trouve la clé Salsa20). En regardant le code complet de la fonction associée, nous pouvons voir que ce *buffer* sur la pile n'est en effet jamais effacé.

De plus, comme il n'y a aucun mécanisme de type ASLR sur le bootloader, cette adresse sera toujours la même.

3.11 Modification du bootloader pour déchiffrer la MFT

Pour quelqu'un disposant d'un mécanisme permettant d'écrire directement dans la mémoire de la machine (en utilisant par exemple une carte *PCI Express* **[10]** au d'autres interfaces comme *FireWire* ou *Thunderbolt* **[11]**), il serait donc possible de déchiffrer les données. L'attaque consiste à patcher le bootloader en mémoire de manière à ce qu'il utilise la clé restée sur la pile. Cette section simule cette attaque grâce à Miasm.

Pour cela, nous allons injecter du code à l'adresse **0x82A8**. Cette fonction vérifie que la clé entrée est celle attendue. Étant donné qu'elle a été effacée du disque, et que le texte de demande rançon est complètement aléatoire **[12]**, le bootloader n'a en théorie aucun moyen de savoir si la clé entrée est bonne. Cette fonction renverra toujours 0 (clé incorrecte). Le code injecté va copier la clé Salsa20 depuis l'adresse **0x674A** vers un endroit spécifique de la pile, pour que la fonction de déchiffrement à l'adresse **0x835A** utilise cette clé. Nous sauterons ensuite sur cette fonction.

Le code associé est le suivant :

```
    ; Sauvegarde des registres sur la pile
    PUSHA
    LEA DI, WORD PTR [BP-0x44]
    LEA BX, WORD PTR [key_addr]
    XOR CX,CX

    ; Copie de la clé sur la pile à l'adresse bp-0x44
loop:
    MOV EAX, DWORD PTR [BX]
    MOV DWORD PTR [DI], EAX
    ADD DI, 4
    ADD BX, 4
    INC CX
    CMP CX,8
    JNZ loop

    ; Restauration des registres
    POPA
    ; Saut sur la fonction de déchiffrement (avec une adresse
absolue CS:OFFSET)
    JMP 0000:0x835A
```

Nous utilisons Miasm pour l'assembler grâce à la fonction suivante :

```
def asm_shellcode(asm, labels = None):
    machine = Machine("x86_16")
    symbol_pool = asmblock.AsmSymbolPool()

    # Assemble
    blocks, symbol_pool = parse_asm.parse_txt(machine.mn, 16, asm,
symbol_pool)

    # Set custom labels
    if not labels is None:
        for name,value in labels.iteritems():
            sym = symbol_pool.getby_name(name)
            symbol_pool.set_offset(sym, value)

    # Resolve all the labels
    patches = asmblock.asm_resolve_final(machine.mn,
                                         blocks,
                                         symbol_pool)

    # Patch the final code with the label values
    shellcode = StrPatchwork()
    for offset, raw in patches.items():
        shellcode[offset] = raw
    return str(shellcode)
```

Attardons-nous un peu sur cette fonction. Le code est tout d'abord assemblé en 16 bits. Les labels passés par argument sont ensuite associés à une valeur concrète avec la fonction **symbol_pool.set_offset**. Tous les labels restants (par exemple ici **loop**) sont résolus grâce à la fonction **asmblock.asm_resolve_final**, qui retourne le code assemblé par blocs. Nous utilisons finalement la classe utilitaire **StrPatchwork** pour assembler le *shellcode* final.

La fonction **read_key_and_patch** se charge de lire la clé en mémoire, l'afficher et d'écrire ce code fraîchement assemblé en mémoire :

```
def read_key_and_patch(jitter):
    # Key is still in the stack, at 0x674A. You can find this value
by activating the
    # find_key_in_mem breakpoint!
    key_addr = 0x674A
    key = jitter.vm.get_mem(key_addr, 32)
    print >>sys.stderr, "\n[+] Key from memory: %s" % key.
encode("hex")

    # Assemble our "shellcode" thanks to Miasm!
    shellcode = """
...
    """
    shellcode = asm_shellcode(shellcode, {"key_addr": key_addr})

    # Patch the bootloader in memory to decrypt using the key
    jitter.vm.set_mem(0x82A8, shellcode)
    return True
```

Il ne reste plus qu'à poser un point d'arrêt à la même adresse qu'à la section ci-dessus (**0x85AF**) afin d'appeler cette fonction :

```
jitter.add_breakpoint(0x85AF, read_key_and_patch)
```

Tout est maintenant en place. Il suffira ensuite d'appuyer sur « Entrée » lorsque le bootloader demandera la clé pour lancer le processus de déchiffrement. Le drapeau **--hook=patch_bootloader** du script **emulate_mbr** permet d'activer cette attaque.

3.12 Étude du keystream lors du chiffrement

L'algorithme de chiffrement utilisé est *Salsa20*, un algorithme de chiffrement par flot (autrement appelé *stream cipher*). Le principe général est qu'un flux de données aléatoire basé sur une clé - appelé communément *keystream* - est généré, et ce flux est XORé avec les données à chiffrer. Un avantage des *stream ciphers* est que les données à chiffrer n'ont pas besoin d'être *paddées*. Il faut en revanche faire attention à ne pas utiliser deux fois les mêmes parties de ce flux de données aléatoires.

Nous pouvons vérifier cela avec Miasm, en regardant les données avant et après chiffrement, et en affichant la différence (XOR).

Pour cela, nous savons déjà poser des breakpoints. Le début de la fonction de chiffrement est à l'adresse **0x9798**, et la fin à l'adresse **0x9877**. Nous allons poser le premier breakpoint juste après l'instruction **enter**, et le deuxième juste avant l'instruction **leave**, afin d'avoir la pile proprement alignée pour récupérer les données avant et après chiffrement. Le code associé est le suivant :

```
_last_buf = None
def encrypt_start(jitter, options):
    global _last_buf
    buf_ptr  = upck16(jitter.vm.get_mem((jitter.cpu.SS << 4) + jitter.cpu.BP
+ 0xC, 2))
    buf_size = upck16(jitter.vm.get_mem((jitter.cpu.SS << 4) + jitter.cpu.BP
+ 0xE, 2))
    _last_buf = jitter.vm.get_mem((jitter.cpu.DS << 4) + buf_ptr, buf_size)
    return True

def encrypt_end(jitter, options):
    global _last_buf
    buf_ptr  = upck16(jitter.vm.get_mem((jitter.cpu.SS << 4) + jitter.cpu.BP
+ 0xC, 2))
    buf_size = upck16(jitter.vm.get_mem((jitter.cpu.SS << 4) + jitter.cpu.BP
+ 0xE, 2))
    encr_buf = jitter.vm.get_mem((jitter.cpu.DS << 4) + buf_ptr, buf_size)
    keystream = ''.join(chr(ord(a)^ord(b)) for a,b in zip(_last_buf,encr_
buf)).encode("hex")
    keystream = ' '.join(keystream[i:i+4] for i in xrange(0,len(keystream),4))
    print >>sys.stderr, "Keystream for next 2 sectors: %s" % keystream
    return True
jitter.add_breakpoint(0x979C, functools.partial(encrypt_start,
options=options))
jitter.add_breakpoint(0x9876, functools.partial(encrypt_end, options=options))
```

Le drapeau **--dump-keystream** du script **emulate_mbr** permet d'activer cette fonctionnalité.

En regardant la sortie, nous nous apercevons que, entre deux secteurs (2*512 octets), le *keystream* est seulement décalé de deux octets, au lieu des 2x512 octets normalement requis. Ce décalage est schématisé en figure 3.

Ainsi, des parties du *keystream* sont réutilisées entre plusieurs secteurs, ce qui peut permettre de récupérer certaines de ses composantes.

En effet, si l'on considère **p** le texte clair, **k** le *keystream* et **c** le texte chiffré, alors la fonction de chiffrement **E** se définie comme **E(p) = p xor k = c**. Une partie des structures de la MFT étant invariantes et connues, il est donc possible, sur deux secteurs, de retrouver une partie du *keystream* utilisé pour ces secteurs. Celui-ci étant réutilisé pour les deux secteurs suivants en étant simplement décalé de deux octets, une partie du clair de ces autres secteurs peut être retrouvée.

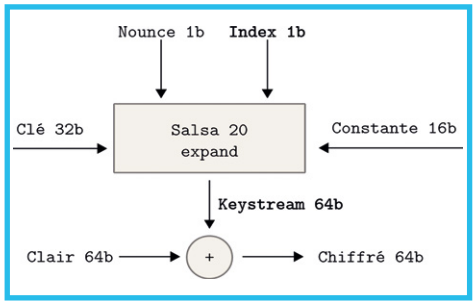


Fig. 2 : Chiffrement des données avec Salsa20.

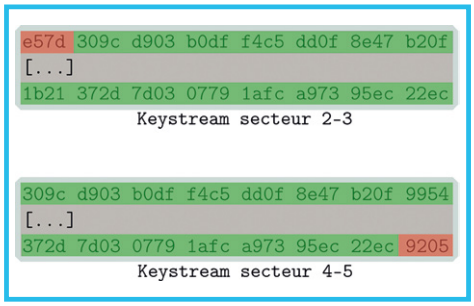


Fig. 3 : Décalage du keystream.

Cette vulnérabilité dans l'implémentation de *Salsa20* du bootloader a été exploitée par *CrowdStrike* pour récupérer une grande partie des données d'origine de la MFT (entre 98.10% et 99.97% selon la méthode **[13]**).

Conclusion

L'émulation du code du bootloader NotPetya a permis de vérifier des hypothèses et de comprendre de manière très concrète les différentes étapes liées au chiffrement des entrées de la MFT. En outre, elle nous a permis de vérifier la validité de méthode de recouvrement de données, de trouver assez facilement le biais dans l'implémentation du *keystream Salsa20* (sans avoir à reverser l'algorithme statiquement), ou encore de simuler la récupération de la clé restant en mémoire après le chiffrement.

Cet article ne fait que gratter la surface des possibilités de Miasm, et nous espérons que l'approche didactique adoptée dans les étapes d'émulation encouragera peut-être les lecteurs non initiés à Miasm à jouer avec :) ■

Remerciements

Nous tenons à remercier gapz pour ses encouragements, ainsi que Camille Mougey et Fabrice Desclaux pour leur aide et la relecture de cet article ! Nous remercions également Thomas Chauchefoin et zerk pour leur relecture attentive.

Retrouvez toutes les références de cet article sur le blog de MISC : https://www.miscmag.com/ .

VOTRE SIEM À PORTÉE DE MAIN AVEC ELASTICSEARCH/ ELASTALERT

Laurent BUTTI

mots-clés : ***ANALYSE DE JOURNAUX D'ACTIVITÉ / SIEM / ALERTES COMPORTEMENTALES***

Nous présentons dans cet article une étude préalable de l'outil ElastAlert en tant que SIEM dans un environnement avec stockage d'événements dans ElasticSearch. Nous aborderons les principes de fonctionnement et les principales fonctionnalités rendues par l'outil ainsi qu'un retour d'expérience de son utilisation opérationnelle.

Les technologies ElasticSearch/LogStash/Kibana sont de plus en plus populaires pour traiter des masses d'informations très importantes. Pour ce faire, il est nécessaire de structurer les événements en documents JSON, qui seront sémantiquement normalisés et éventuellement enrichis avec des données tierces (localisation géographique, résolution inverse DNS...). Dans le cadre de traitement de journaux d'activité (événements), cela peut être réalisé grâce à un outil open source tel que LogStash (mais non exclusivement, d'autres outils populaires existent, e.g. Heka). L'indexation et le stockage de ces documents structurés en JSON ainsi que la mise à disposition d'un langage de requêtes avec des fonctionnalités puissantes sont dévolus à ElasticSearch. Enfin, Kibana est un système de visualisation des données présentes dans ElasticSearch qui rend extrêmement aisé la construction de tableaux de bord et autres graphes de tendances sur les données stockées. Il est à noter que Kibana réalise tout un ensemble de traitements des données collectées via ElasticSearch et que donc il est nécessaire de bien connaître le langage utilisé par ElasticSearch si nous souhaitons arriver aux mêmes données restituées uniquement par des requêtes directement à ElasticSearch.

Bien que ces outils soient simples d'utilisation, flexibles et puissants, ils ne sont pas conçus pour l'analyse de journaux d'activité en temps réel. En conséquence, l'outillage fournit par ElasticSearch/Kibana n'est en standard pas suffisant pour réaliser les primitives d'un SIEM.

Nous pouvons cependant noter que le X-Pack (par Elastic) dispose de fonctionnalités utiles pour implémenter un SIEM (suite logicielle appelée « watcher »), mais est sous licence commerciale, malgré les évolutions récentes où certaines parties historiquement privées du X-Pack (monitoring, grok debugger...) sont depuis la 6.3 distribuées par défaut via la suite logicielle ELK.

Concernant les initiatives open source basées sur ElasticSearch pouvant apporter des fonctionnalités de type SIEM, seules deux sont disponibles : MozDef par Mozilla et ElastAlert par Yelp.

La première initiative, MozDef, est utilisée opérationnellement par Mozilla. Elle est développée de manière très active et propose tout un ensemble de fonctionnalités pour disposer d'un SIEM complet (règles de détection, WebUI...) basé sur un stockage des événements dans ElasticSearch.

La deuxième initiative, ElastAlert (*Easy & Flexible Alerting With Elasticsearch*), est un outil de remontée d'alertes sur différents critères (anomalies comportementales) à partir de données présentes dans ElasticSearch.

Ces deux outils ont des approches plutôt différentes : MozDef se voulant complet en terme de fonctionnalités tout en mettant à disposition une architecture de collecte d'événements à base de RabbitMQ et une WebUI évoluée, tandis qu'ElastAlert se veut léger et flexible pour être adaptable dans de multiples contextes (et donc pas nécessairement en tant que brique SIEM).

Après étude de ces deux outils, nous avons souhaité nous focaliser sur ElastAlert qui dispose de plusieurs avantages : léger, modulable, fiable et peu contraignant sur la structure des données stockées dans ElasticSearch.

Nous présentons alors dans cet article le mode de fonctionnement d'ElastAlert tout en décrivant certaines des fonctionnalités intéressantes pour arriver à mettre en œuvre un SIEM tout à fait convenable au niveau des fonctionnalités, et surtout, le plus allégé possible au niveau des composants, apportant alors une fiabilité sur la chaîne de traitement des événements.

Un prochain article est prévu pour aborder certains principes avancés d'ElastAlert.

1 Architecture de collecte, normalisation, enrichissement et traitement

1.1 Collecte, normalisation et enrichissement des événements

La donnée brute est la matière nécessaire en premier lieu. Il faut cependant la structurer dans une forme suffisamment intelligible pour l'exploiter de manière correcte. En effet, dans le cadre de la collecte de journaux d'activité, les formats peuvent être très multiples (texte, JSON, Common Event Format...), mais aussi les structures sémantiques très variées (journaux applicatifs, journaux d'accès de serveur Web, journaux d'accès de serveur SSH...). Il faut alors normaliser ces événements pour les structurer à la fois sur la forme et sur la sémantique, c'est un élément primordial ! Par ailleurs, dans le cadre de l'utilisation de corrélation d'événements entre données ou services intrinsèquement différents, il faudra aussi réaliser un travail (non négligeable) afin de structurer les données de manière à ce que, par exemple, une adresse IP source issue d'un journal d'accès à un serveur Web, soit déposée dans la même structure de donnée sémantique qu'une adresse IP source issue d'un journal d'un pare-feu applicatif.

Nous voyons alors toute la quantité de travail à réaliser si nous disposons de très nombreuses sources de données qu'elles soient standards (journaux d'accès Web...) ou spécifiques (journaux d'activité applicatifs).

Afin de limiter cette problématique, l'idéal serait que la source émettrice des journaux d'activité envoie des données déjà structurées, préférablement en JSON. Les journaux applicatifs devraient en effet tous journaliser en données structurées syntaxiquement et sémantiquement.

Par exemple, il est possible d'utiliser des fonctionnalités pour journaliser directement en JSON pour les journaux d'accès Web (apache et nginx) avec les directives suivantes :

```
LogFormat "{ \"time\":\"%t\", \"remoteIP\":\"%a\", \"host\":\"%V\",
\"request\":\"%U\", \"query\":\"%q\", \"method\":\"%m\",
\"status\":\"%>s\", \"userAgent\":\"%{User-agent}i\",
\"referer\":\"%{Referer}i\" }" json_apache
CustomLog /var/log/apache/access.log apache_json
```

```
log_format nginx_json '{ "time": "$time_iso8601", ' '"remote_addr":
"$remote_addr", ' '"remote_user": "$remote_user", ' '"body_bytes_
sent": "$body_bytes_sent", ' '"request_time": "$request_time",
' '"status": "$status", ' '"request": "$request", ' '"request_
method": "$request_method", ' '"http_referrer": "$http_referer",
' '"http_user_agent": "$http_user_agent" }'; access_log /var/log/
nginx/access.log nginx_json;
```

L'outil open source LogStash est tout à fait adapté dans le cadre-là. Il est capable de se greffer à de très nombreuses sources d'événements (fichier, syslog, RabbitMQ, Kafka, HTTP...) grâce à ses « input plugins », de normaliser et d'enrichir grâce à ses « filter plugins », et de transmettre grâce à ses « output plugins ».

LogStash est capable de monter en performances grâce à une scalabilité verticale (multi-thread) et horizontale (multi-hôtes), en conséquence, sauf dans le cadre d'environnements avec plusieurs dizaines/centaines de milliers d'événements par seconde, il devrait tout à fait convenir. Si tel n'est pas le cas, Heka est réputé beaucoup plus performant, mais au prix d'une courbe d'apprentissage plus pentue pour l'opérateur devant mettre en place la normalisation et l'enrichissement des événements à traiter.

Exemple de normalisation concernant les logs d'accès SSH :

```
# logs bruts
Feb 20 21:54:44 localhost sshd[3402]: Accepted publickey for
vagrant from 10.0.2.2 port 63673 ssh2: RSA 39:33:99:e9:a0:dc:f2:33:
a3:e5:72:3b:7c:3a:56:84
Feb 21 00:13:35 localhost sshd[7483]: Accepted password for vagrant
from 192.168.33.1 port 58803 ssh2
Feb 21 08:35:22 localhost sshd[5774]: Failed password for root
from 116.31.116.24 port 29160 ssh2 Feb 21 19:19:26 localhost
sshd[16153]: Failed password for invalid user aurelien from
142.0.45.14 port 52772 ssh2
Feb 21 21:56:12 localhost sshd[3430]: Invalid user test from
10.0.2.2

# règle GROK à utiliser dans configuration logstash
%{SYSLOGTIMESTAMP:system.auth.timestamp} %{SYSLOGHOST:system.auth.
hostname} sshd(?:\\[%{POSINT:system.auth.pid}\\])?: %{DATA:system.
auth.ssh.event} %{DATA:system.auth.ssh.method} for (invalid user
)?%{DATA:system.auth.user} from %{IPORHOST:system.auth.ip} port
%{NUMBER:system.auth.port} ssh2(: %{GREEDYDATA:system.auth.ssh.
signature})?
```

Les données se retrouvant alors dans un document JSON en sortie de LogStash après avoir transité par tous les filtres.

Dans le cadre de la supervision sécurité, il est primordial de disposer des événements pertinents afin d'y appliquer alors les règles de détection qui pourront utiliser des enrichissements. Il convient alors d'utiliser les fonctionnalités de LogStash afin d'enrichir les événements à la volée sur de nombreux critères : informations GeoIP grâce aux bases de connaissance MaxMind (pays, *Service Provider*, numéro d'Autonomous System...), résolution inverse DNS, présence dans une liste donnée (utilisable pour enrichir par exemple si adresse IP connue dans une liste de réputation)...

```
/etc/logstash/logstash.yml
# This defaults to the number of the host's CPU cores.
#
pipeline.workers: 64
```

Nous constatons que sur les trois exemples de normalisation (apache, nginx et SSH), l'adresse IP source de l'événement concerné se retrouve respectivement dans **remoteip**, **remote_addr** et **system.auth.ip**. Il sera très difficile de corréler ces événements entre eux

sans structure commune. Une approche est de disposer d'une structure commune pour tous les événements, dans laquelle les données représentant un socle commun pourront y être déposées.

Note

Attention lors de l'utilisation de plugins tels que la résolution DNS. Cela est bloquant pour l'événement en cours de traitement, en conséquence, cela pénalise très fortement votre capacité à encaisser un volume d'événements par seconde important. Il faut alors jouer sur les paramètres du plugin DNS (`failed_cache_size`, `failed_cache_ttl`, `hit_cache_size`, `hit_cache_ttl`, `timeout`) ainsi que sur le pipeline. En effet, les filtres traitent un événement à la fois, donc si le filtre DNS prend (par exemple) 10 millisecondes par résolution DNS alors un *worker* sera capable d'encaisser 100 événements par seconde. Il faut alors dimensionner le nombre de *workers* total sachant que la configuration par défaut de LogStash est de prendre un nombre de *workers* égal au nombre de CPU de la machine hôte.

Par exemple, mais de manière non exclusive :

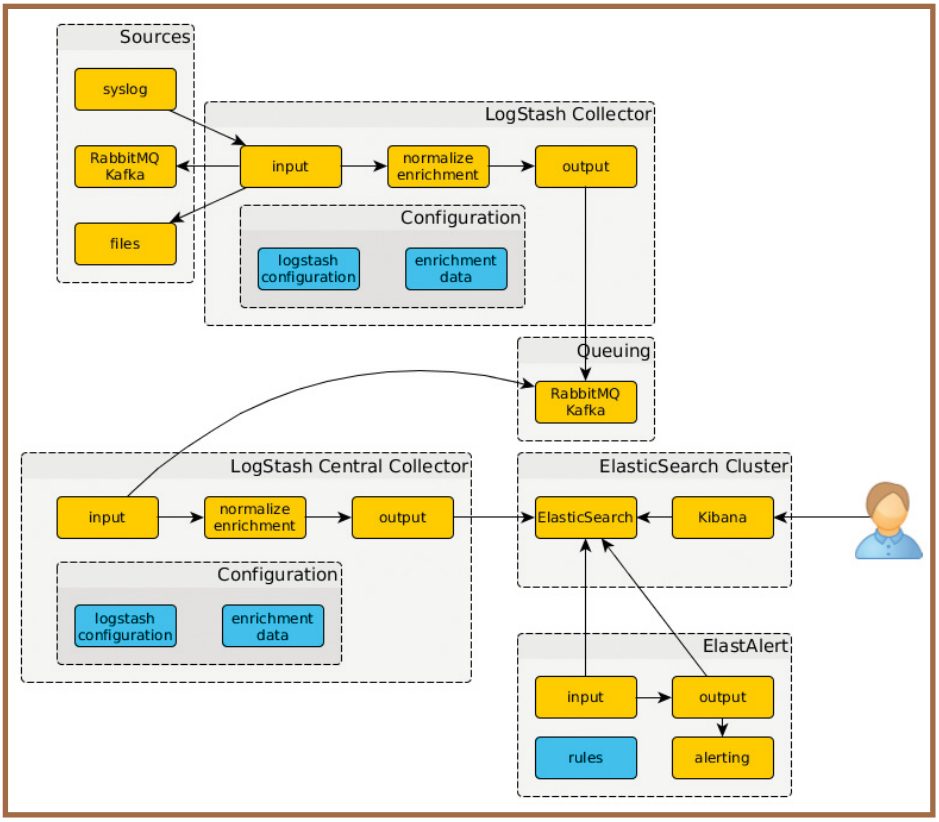
- `data.source.ip` : pour l'adresse IPv4 ;
- `data.source.country` : pour la résolution GeoIP du pays ;
- `data.source.reverse_dns` : pour la résolution inverse DNS ;
- `data.source.reputation` : pour un marqueur de présence dans une liste de (mauvaise) réputation...

Ensuite, au niveau de la visualisation via Kibana ou via des requêtes ElasticSearch, nous pourrons tirer avantage de la construction d'index adaptés pour récupérer des événements dissociés, mais disposant d'un socle commun, par exemple, récupérer l'ensemble des événements ayant une certaine adresse IP source, que ce soient des événements de type accès web, pare-feu applicatif web ou autres...

```
# index possibles par type d'événement
events.type.subtype.date_format
events.web.apache-error.%Y.%m.%d
events.web.apache-access.%Y.%m.%d
events.sys.ssh.%Y.%m.%d

# un index général qui sera utilisé si nous souhaitons corréler des
données entre elles grâce au
# socle commun
events.*
```

La collecte, la normalisation et l'enrichissement sont essentiels afin de mettre en place une supervision de sécurité efficace. En effet, les règles de détection pourront alors tirer avantage des données structurées et enrichies, améliorant alors naturellement la flexibilité et l'efficacité de la solution.



Architecture générale

1.2 Architecture générale

Dans cette partie, nous présentons un exemple d'architecture de collecte, normalisation, enrichissement et de remontée d'alertes grâce aux technologies citées dans l'article. Il est à noter qu'il est important de disposer d'un bus de transport d'événements qui soit réputé performant et fiable comme par exemple RabbitMQ ou Kafka, ceci pour disposer de fiabilité sur la chaîne de collecte des événements. Ce n'est bien sûr qu'un exemple, d'autres approches sont possibles.

Afin de traiter une forte volumétrie, nous pouvons nous reposer sur plusieurs collecteurs LogStash qui réaliseront la normalisation et l'enrichissement préalable. Puis nous pouvons utiliser un collecteur central LogStash pour la centralisation de la collecte des événements pré-traités avec un éventuel enrichissement en a posteriori. Le choix d'enrichissement en a priori et a posteriori peut être défini selon des contraintes

de performances. En effet, lors du premier étage de normalisation, il est envisageable de rejeter des événements non conformes ou non souhaités. En conséquence, il est intéressant de ne réaliser l'enrichissement qu'en a posteriori, pour peu bien sûr que le rejet des événements ne soit pas sur des critères issus d'enrichissements.

La partie SIEM étant alors dévolue à la brique ElastAlert que nous allons décrire dans les prochains paragraphes.

2 ElastAlert

Si vous écrivez des données dans ElasticSearch en temps réel et que vous souhaitez intégrer un système de détection et d'alertes simple et flexible, ElastAlert est probablement une solution intéressante à considérer ! En effet, ElastAlert repose sur un système de règles et d'alertes. ElasticSearch est interrogé régulièrement et les données récupérées sont passées aux différentes catégories de règles qui déterminent si les critères de détection d'événements sont déclenchés. Si cela est le cas, le ou les modules d'alertes sont appelés, réalisant alors une ou plusieurs actions selon ce qui est décrit dans les règles concernées. L'avantage de cette approche est de lier la règle concernée à un ensemble de réactions possibles, ce qui est très flexible puisqu'en pratique nous avons souvent besoin de plusieurs réactions possibles à un même événement.

Une règle est composée des éléments suivants :

- le type de règle utilisé ;
- la définition de la requête en langage ElasticSearch ;
- éléments de configuration spécifiques au type de règle ;
- la ou les alerte(s) lorsque les critères de détection sont atteints ;
- éléments de configuration spécifiques aux types d'alertes.

Nous pouvons noter un point très pratique dans cette approche : le fait de lier une requête ElasticSearch avec la règle de détection. En effet, lors des investigations suite à une alerte issue d'une des règles, il suffit alors de récupérer la requête ElasticSearch concernée à appliquer sur la période donnée afin de faciliter très grandement l'investigation en a posteriori (typiquement grâce à la puissance de l'interface de visualisation Kibana). Faciliter le travail d'investigation de l'opérateur est un élément primordial pour disposer d'un processus de supervision de sécurité efficace, et nous ne saurons que conseiller au lecteur de se référer au lien suivant qui apporte un éclairage très pertinent à la gestion des événements de sécurité **[I]**.

2.1 Types de règles

Différents types de règles sont disponibles dans ElastAlert, ce qui rend l'outil très pertinent dans un grand nombre de contextes. Par ailleurs, il est assez aisé de rajouter une autre type de règle en développant un module pour ElastAlert. Bien entendu, cela requiert de bien être rompu aux principes de fonctionnement d'ElastAlert, ce qui nécessite naturellement un temps d'adaptation.

Les types de règles possibles sont :

- **blacklist** et **whitelist** : détection lorsque certains champs correspondent à des listes noires ou blanches sur une durée donnée ;
- **change** : détection lorsqu'un champ d'un événement a des valeurs différentes sur une durée donnée ;

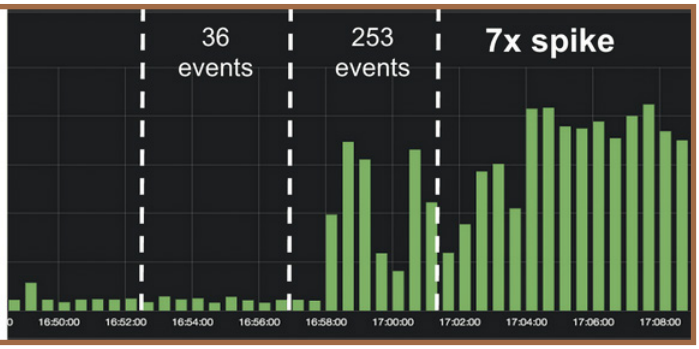
```
type: change
compare_field: country
query_key: username
timeframe:
  days: 1
```

@timestamp	username	remote_ip	country
2015-07-03T06:40:00.000-07:00	user123	[redacted]	Romania
2015-07-03T06:35:00.000-07:00	denvercoder9	[redacted]	Canada
2015-07-03T06:15:00.000-07:00	coolguy7	[redacted]	Germany
2015-07-03T05:30:00.000-07:00	user123	[redacted]	United States
2015-07-03T05:20:00.000-07:00	coolguy7	[redacted]	Germany
2015-07-03T05:00:00.000-07:00	user123	[redacted]	United States

Source : Yelp

- **frequency** : détection lorsque nous avons le dépassement d'un nombre d'événements sur une durée donnée ;
- **any** : détection lorsqu'un événement correspond à un filtre donné sur une durée donnée ;
- **spike** : détection lorsque nous avons le changement d'un taux (à la hausse ou à la baisse) du nombre d'événements sur une durée donnée ;

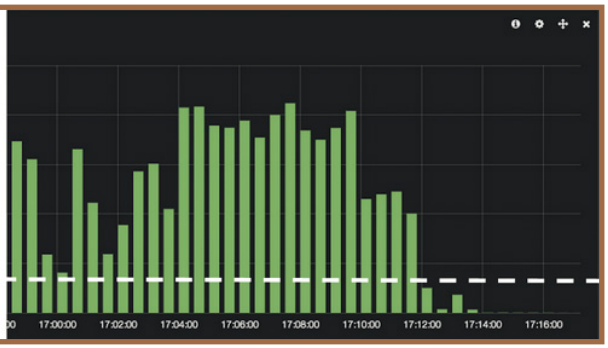
```
type: spike
spike_height: 7
spike_type: up
timeframe:
  minutes: 5
threshold_ref: 25
```



Source : Yelp

- **flatline** : détection lorsque nous avons le dépassement à la baisse d'un nombre d'événements sur une durée donnée ;

```
type: flatline
threshold: 150
timeframe:
  minutes: 5
```



Source : Yelp

- **new terms** : détection lorsque nous avons une nouvelle valeur dans le champ des événements sur une durée donnée ;

```
type: new_term
fields:
  -  kernel_extension
```

@timestamp	kernel_extension
2015-07-27T21:31:15.000-07:00	com.apple.driver.AppleUSBCDC
2015-07-27T21:31:15.000-07:00	com.apple.driver.AppleUSBHIDKeyboard
2015-07-27T21:31:15.000-07:00	com.apple.iokit.IOUSBHIDDriver
2015-07-27T21:31:15.000-07:00	com.evil.rootkit.trojan
2015-07-27T21:30:15.000-07:00	com.apple.iokit.IOUSBMassStorageClass
2015-07-27T21:30:15.000-07:00	com.apple.driver.AppleMikeyHIDDriver
2015-07-27T21:30:15.000-07:00	com.apple.iokit.SCSITaskUserClient

Source : Yelp

- **cardinality** : détection lorsque la cardinalité d'un champ est supérieure ou inférieure à un seuil donné sur une durée donnée ;
- **metricaggregation** : détection lorsque nous avons un dépassement à la hausse ou à la baisse sur une agrégation spécifiée (moyenne, cardinalité, somme...) sur une durée donnée ;
- **percentagematch** : détection lorsque nous avons un dépassement à la hausse ou à la baisse sur un pourcentage d'événements selon un critère de sélection sur une durée donnée.

Nous constatons avoir un nombre suffisant de fonctionnalités pour disposer d'un SIEM tout à fait honorable, car tous ne sont pas capables d'autant !

En effet, il nous manque principalement des règles de corrélation d'événements entre eux par des principes de machines à état, e.g. détection si séquence ordonnée d'événements sur une durée donnée, ce qui effectivement nécessiterait une logique plus complexe de récupération de l'historique des événements sur une durée donnée, puis ensuite d'effectuer les traitements souhaités.

2.2 Types d'alertes

Différents types d'alertes sont disponibles dans ElastAlert, ce qui rend l'outil très pertinent dans un grand nombre de contextes. Par ailleurs, il est assez aisé de rajouter une autre type de règle en développant un module pour ElastAlert.

Les modules actuellement disponibles dans ElastAlert sont très variés : l'exécution de commandes, l'envoi d'e-mail, une sortie de type débogage, l'envoi HTTP en POST, JIRA, MsTeams, PagerDuty, Exotel, Twilio, VictorOps, Gitter, ServiceNow, OpsGenie, SNS, HipChat, Slack, Telegram...

Il est à noter qu'il est possible d'utiliser plusieurs alertes pour une même règle, ce qui est très utile en pratique puisqu'une détection doit pouvoir générer plusieurs actions de contre-mesures (par exemple : envoi d'un mail, création d'un ticket d'incident, pilotage d'un équipement de sécurité pour contre-mesure...).

D'autres fonctionnalités présentes dans ElastAlert sont en pratique très utiles, car les outils d'alertes pourront disposer de données supplémentaires exploitables par les opérateurs facilitant d'autant plus leurs investigations :

- liens vers des tableaux de bord Kibana : l'opérateur pourra facilement analyser les événements concernés par la remontée d'alerte ;
- consolidation de l'ensemble des événements ayant déclenché une règle de détection : l'opérateur aura à disposition l'ensemble des événements concernés par la remontée d'alerte ;
- calcul d'autres indicateurs comme le top des données (**top_count_keys**) selon la clé d'agrégation utilisée.

Note

Il est aussi possible de détourner le mode de fonctionnement du déclenchement des alertes pour s'en servir d'un outil de reporting régulier par période de temps donnée.

2.3 Configuration globale

ElastAlert dispose d'une configuration globale **config.yaml** qui définit certains modes de fonctionnement pour l'ensemble de l'exécution de son instance, ces paramètres seront globaux à l'ensemble des règles qui seront exécutées.

Exemple de configuration :

```
# fichier de dépôt des règles
rules_folder: rules

# fréquence d'exécution
run_every:
  seconds: 10
# fenêtre de temps maximale sur laquelle les requêtes ES seront
réalisées
```

```
buffer_time:
  minutes: 60

# informations de connectivité à l'ES
es_host: 127.0.0.1
es_port: 9200
es_username: someusername
es_password: somepassword
# informations de connectivité TLS à l'ES
use_ssl: True
verify_certs: True
ca_certs: /path/to/cacert.pem

# nom de l'index ES pour sauver l'état de fonctionnement
writeback_index: elastalert_status

# délai de réémission des alertes si erreur d'envoi
alert_time_limit:
  minutes: 30

# informations spécifiques pour les alertes par e-mail (possible
soit en global, soit dans les règles)
smtp_host: 127.0.0.1
smtp_port: 25
```

Pour des raisons de performance ou pour séparer les différents types de règles, il est envisageable de disposer plusieurs fichiers de configuration globaux qui seront alors appelés par plusieurs instances ElastAlert, comme ci-dessous :

```
# python elastalert/elastalert.py --config configA.yaml
# python elastalert/elastalert.py --config configB.yaml
```

Par ailleurs, cela apporte une flexibilité sur les répertoires des règles qui sont spécifiés dans les fichiers de configuration globaux, nous pouvons avantageusement en tirer partie pour avoir des règles de ce type :

```
rules.prod
rules.preprod
rules.dev
```

Enfin, il est indispensable d'utiliser une brique comme supervisord afin de s'assurer que l'exécution des instances d'ElastAlert soit correctement réalisée et qu'elles soient redémarrées en cas de plantage d'ElastAlert. Nous constatons alors ici toute la flexibilité dont ElastAlert fait preuve.

2.4 Exemple de conception d'une règle de type « fréquence »

```
# nommage de la règle
name: ModSecurity.Error.High

# type de règle
type: frequency

# index à utiliser pour la recherche
index: events.modsecurity-error.%Y.%m.%d

# fonctionnalité pour optimiser la sélection des index (très
recommandée)
use_strftime_index: true

# nombre d'événements devant être dépassé pour déclenchement alerte
num_events: 3

# durée sur laquelle les calculs de seuils sont réalisés
timeframe:
  minutes: 15

# filtre de requête ES pour sélection des documents concernés par
la règle
filter:
- query:
    query_string:
      query: 'modsecurity.error.score:[20 TO *]'

# clé sur laquelle l'agrégation sera réalisée par ElastAlert
query_key: data.source.ip

# inclusion des informations suivantes dans l'alerte générée par
ElastAlert
# à noter que l'ensemble de ces informations sera préfixé par
match_body
include:
- modsecurity.error
- data

# configuration des modules d'alerte
alert:
- "email"
```

Dans l'exemple ci-dessus, nous pouvons noter le processus de construction d'une règle :

- connaître l'objectif i.e. le besoin fonctionnel pour sélectionner le type de règle le plus adapté (**frequency**) ;
- définir les critères de détection, dans ce cas-là, nombre d'événements à dépasser (**num_events**) sur une période donnée (**timeframe**), sur un ensemble de documents donné (**filter**/**query**/**query_string**/**query**) par une clé d'agrégation par adresse IP source (**query_key**) ;
- définir les données à inclure lors du stockage de l'alerte et d'éventuellement de sa transmission aux modules d'alerte (**include**) ;
- définir le processus d'alerte à réaliser (**alert**) avec l'ensemble des modules à appeler et leur configuration spécifique.

Pour d'autres exemples de conception de règles, nous invitons le lecteur à se reporter à deux articles des concepteurs d'ElastAlert **[2]** et **[3]**.

Concernant les spécificités des mécanismes d'alertes, deux paramètres de configuration sont essentiels :

- **realert** : temps minimum pendant lequel toute nouvelle alerte (concernant un événement « similaire », la similarité étant définie par ElastAlert via une clé associant le nom de la règle et la **query_key** si elle est présente) ne déclenchera pas les modules d'alertes, cette option est essentielle pour éviter de se retrouver submergé d'alertes similaires sur une fenêtre de temps donnée ;

- **aggregation** : agrégation des alertes en une alerte sur une période donnée, ce qui aura pour effet de n'alerter que toutes les périodes données avec la liste des alertes concernées, cela peut être utile dans un contexte où l'on souhaite disposer de rapports périodiques.

Lorsqu'une alerte a été déclenchée, il est possible soit de passer l'ensemble des documents ayant déclenché l'alerte (via **pipe_match_json**), soit uniquement l'alerte en elle-même avec les métadonnées. Tout dépend du traitement qui en sera réalisé ensuite et de l'aide à l'opérateur souhaitée pour lui faciliter le travail d'investigation. Il est tout à fait possible de transformer ce qui est transmis au module d'alerte de manière à rendre l'alerte plus intelligible pour un humain, par exemple :

```
alert_text:
    {1}: {0} IP address has triggered an alert: {2},{3},{4}
alert_text_args:
- data.source.ip
- timestamp
- data.rule_name
- data.source.reverse_dns
- data.source.reputation
```

2.4.1 Conception et maintenance des règles

En plus des aspects classiques de convention de nommage des règles, la conception des règles nécessite une attention toute particulière vis-à-vis des aspects performance. En effet, ElastAlert exécute tous les temps « t », l'ensemble des règles de manière séquentielle et en mono-thread. Toute règle trop consommatrice en temps aura un impact sur les autres règles et donc sur l'ensemble du système. Sachant que la volumétrie à traiter peut aussi jouer un rôle critique dans le temps de traitement requis au niveau d'ElastAlert, il s'agit de s'assurer de l'ensemble des verrous nécessaire au bon fonctionnement de la solution (cf. partie limitations techniques).

Concernant le développement et la maintenance des règles, une approche est de faire du classique : disposer de plusieurs environnements de règles (production, préproduction, développement) et se baser sur des outils d'intégration et de déploiement continus tels que GitLab-CI afin de faciliter et de fiabiliser l'ensemble du processus de développement, maintien et déploiement des règles. Il ne faudrait pas qu'une erreur syntaxique ait le malheur de casser l'environnement de production, cela ne peut pas arriver si les verrous adéquats sont positionnés correctement aux endroits adaptés.

3 Autres considérations

3.1 Fiabilité

ElastAlert dispose de plusieurs fonctionnalités pour améliorer sa fiabilité de fonctionnement. Par exemple, afin de pallier au cas de non-disponibilité d'ElasticSeach, ElastAlert aura sauvé préalablement son état de fonctionnement dans un index dans ElasticSearch, ce qui lui sera utile pour reprendre son fonctionnement nominal lors du retour à la normale d'ElasticSearch.

Note

Après des semaines d'utilisation dans un contexte opérationnel avec de fortes volumétries de plusieurs centaines d'événements par seconde, nous n'avons pas constaté de défauts de fonctionnement d'ElastAlert et nous sommes satisfaits de la très bonne stabilité de fonctionnement de l'outil.

De la même manière, ElastAlert sauve son état de fonctionnement dans des index dans ElasticSearch dont toutes les erreurs rencontrées dans l'index **elastalert_status_errors**, ce qui donne un indicateur fiable du fonctionnement d'ElastAlert. Il est alors possible de superviser son mode de fonctionnement et d'y rajouter aussi un système d'alerte comme il se doit pour de la supervision de bon fonctionnement.

Note

À noter qu'ElastAlert se repose sur python-elasticsearch pour interroger ElasticSearch, cette librairie est reconnue comme étant flexible, fiable et fonctionnant pour toutes les versions d'ES (qui évoluent très rapidement).

3.2 Limitations techniques

Nous avons constaté que les règles implémentées dans ElastAlert n'utilisent pas par défaut les fonctionnalités d'ElasticSearch concernant les agrégations. Bien entendu, utiliser les agrégations calculées par ElasticSearch entraîne forcément une perte d'information puisque que seuls les résultats du calcul de l'agrégation seront restitués dans les « buckets », mais le gain en terme de performance est clairement inégalable.

Par exemple, dans le fonctionnement normal d'ElastAlert, sur une règle de type **cardinality**, alors le calcul de la cardinalité est réalisé par ElastAlert lui-même sur le champ qui est défini par l'option **cardinality_field**, ce qui sera très coûteux en temps de calcul (et donc en pratique potentiellement inutilisable) si des milliers de documents sont restitués par ElasticSearch à ElastAlert.

Afin de pallier à cette problématique, il faut soit utiliser le type de règle **MetricAggregation** ou développer son propre module de règle pour ElastAlert.

De la même manière, les règles complexes nécessitant une gestion d'état (détection si état 1 puis état 2 puis état 3) ne sont pas naturelles en ElasticSearch, et même si cela est réalisable en développant un module de règle pour ElastAlert, cela est loin d'être trivial. La bonne nouvelle est que cela est théoriquement implémentable avec toutefois un peu d'huile de coude !

3.3 Limitations générales

Bien que disposant de très nombreux suiveurs et contributeurs dans la communauté open source, il est assez perturbant de voir que plus de 500 « issues » ne sont pas résolues et que plus de 60 « pull requests » sont encore ouvertes à ce jour... Plutôt inquiétant pour le futur d'une telle solution, donc, si vous vous orientez vers ce choix technique, il doit se faire en connaissance de cause : reprise à son propre compte des évolutions fonctionnelles et correctives de la version d'ElastAlert que vous devrez réaliser par vous-même dans votre branche dédiée. C'est tout à fait envisageable, pour quiconque apprécie de développer en Python et souhaite acquérir de bonnes compétences en ElasticSearch, même si cela est dommage d'éprouver autant de difficultés pour reverser dans la communauté open source.

3.4 Perspectives

Malgré des limitations tout à fait naturelles, les perspectives sont très prometteuses de par la flexibilité d'ElastAlert qui doit être considéré comme un framework de gestion d'événements et d'alertes. Les fonctionnalités ElasticSearch sans cesse en évolution pourront alors être tirées à profit par de nouveaux modules pour ElastAlert.

Conclusion

ElastAlert est tout à fait adapté en tant que solution de type SIEM « allégé ». Il dispose d'une grande flexibilité et simplicité d'utilisation. Cependant le choix d'aller vers l'utilisation d'un tel outil doit être fait en connaissance de cause : s'assurer de la maîtrise de l'outil et s'approprier les évolutions à réaliser pour satisfaire les besoins fonctionnels de votre solution à déployer. Si tel est le cas, alors nous vous souhaitons de belles aventures dans le monde de la gestion des alertes de sécurité avec ElasticSearch ! ■

Remerciements

Je tiens à remercier Thanat0s pour sa relecture et ses commentaires pertinents.

Références

[1] https://www.linkedin.com/pulse/socless-detection-team-netflix-alex-maestretti/

[2] https://engineeringblog.yelp.com/2015/10/elastalert-alerting-at-scale-with-elasticsearch.html

[3] https://engineeringblog.yelp.com/2016/03/elastalert-part-two.html

DOSSIER

VERS LA FIN DES MOTS DE PASSE ?

Ces dernières années ont été riches concernant les solutions d'authentification, avec notamment le développement des fameux « facteurs » supplémentaires afin de valider l'identité d'un utilisateur. Cette évolution importante, que l'on nomme désormais 2FA ou MFA (pour « two-factor authentication » ou « multi-factor authentication »), n'a rien de très nouveau, des dispositifs de ce genre existent depuis des années, à l'instar du vénérable RSA secure ID. Mais avec le développement des services en ligne (pour ne pas dire le fait de pouvoir manipuler de l'argent) et face à l'échec de la simple authentification par mot de passe, il devint nécessaire d'apporter une réponse en ajoutant un élément d'authentification plus robuste (et de supprimer les aides à la récupération des mots de passe, qui vous demandaient l'âge de votre grand-mère, cela va sans dire).

L'idée a donc été d'ajouter naturellement une étape supplémentaire dans la partie authentification, telle que la saisie d'un code généré par son « token » ou bien reçu par SMS. Seulement voilà, même si l'idée est simple, implémenter correctement une seconde étape d'authentification n'est pas trivial, certains vecteurs étant plus fiables que d'autres (un SMS peut par exemple être intercepté) et la bonne validation des deux facteurs n'a pas toujours été faite correctement. De plus, aucun standard adapté n'existait alors, empêchant toute interopérabilité ainsi que la mise en place de solutions uniformisées pour l'utilisateur final. Un consortium d'entreprises privées, la FIDO (« Fast Identity Online ») Alliance, portée notamment par Google, NXP et Yubico, s'est donné pour tâche il y a quelques années de développer une norme, dont l'objet était notamment de simplifier l'authentification à deux facteurs, ce qui conduira au protocole U2F, désormais bien connu, que l'on peut aujourd'hui utiliser avec une bonne partie des géants du Web et des navigateurs modernes (et même sur mobile, en utilisant NFC).

Au-delà des protocoles pour l'authentification forte, l'expérience utilisateur par rapport aux mots de passe a elle aussi clairement évolué, depuis le développement des gestionnaires de mots de passe à l'enregistrement de ces derniers dans le navigateur. Pour l'utilisateur initié, les mots de passe sont presque devenus transparents dans l'expérience de navigation sur le Web, une fois les bons outils installés. De la même manière, il pourra utiliser un token d'authentification U2F de manière très simple lorsque c'est supporté. Bref, quelques pas vont dans la bonne direction pour mettre un terme aux mots de passe, et la future norme « WebAuthn » du W3C annonce des évolutions encore plus intéressantes.

Ce dossier, qui porte tant sur les attaques que sur les solutions récentes apportées pour l'authentification, est composé des articles suivants : un tour d'horizon des fameux « facteurs » des méthodes d'authentification forte, un retour d'expérience (via l'intermédiaire de programmes de bug bounty) sur les problèmes habituels de sécurité rencontrés sur la partie authentification de services en ligne. Un article sera consacré ensuite à donner un aperçu du fonctionnement du futur standard du W3C, qui va notamment permettre des authentifications sans mot de passe. Sera enfin détaillée une vulnérabilité sur OpenIDconnect aboutissant à une usurpation d'identité.

Bonne lecture !

Émilien Gaspar / gapz / eg@miscmag.com

AU SOMMAIRE DE CE DOSSIER :

TOUR D'HORIZON DE L'AUTHENTIFICATION FORTE (MFA)

Yann CAM
Security Consultant @SYNETIS
Security Researcher @ASafety

mots-clés : *AUTHENTIFICATION / IDENTIFICATION / PASSWORD / OTP / BIOMÉTRIE / MULTI-FACTOR*

L'authentification forte, 2FA, 3FA, MFA pour « Multi-Factor Authentication » se démocratise ces dernières années. Le simple couple « login/password » ne suffit plus, notamment pour les accès sensibles à privilèges. Mais de quels « facteurs » parle-t-on ?

1 L'authentification dite « forte »

Les systèmes d'authentification sont un élément essentiel d'un SI et des services en ligne. Clé de voûte de l'accès aux ressources les plus critiques, l'utilisation d'un système d'authentification forte est de plus en plus demandée au sein des SI modernes et proposée par les majors du Web **[TFA]**.

Mais qu'appelle-t-on une authentification forte ? C'est un processus d'authentification incluant au minimum deux facteurs différents permettant d'accéder à une ressource (une application web, un dossier, un accès administrateur...).

Le processus d'authentification doit combiner (a minima) deux facteurs différents afin d'être qualifié de « fort », les différents facteurs communément utilisés sont les suivants :

- Qu'est ce que l'on sait ? (un mot de passe, une phrase secrète...) ;
- Que possède-t-on ? (une carte magnétique, un hard-token, une clé USB, un smartphone...) ;
- Qu'est ce que l'on est ? (utilisation de l'empreinte digitale, empreinte rétinienne...) ;
- Que savons-nous faire ? (un *selfie* avec grimace, de la reconnaissance vocale...).

Une combinaison de ces facteurs permet d'indiquer que vous êtes authentifié fortement pour accéder à une ressource critique, l'intérêt ici étant de complexifier la tâche de l'usurpateur d'identité en combinant plusieurs facteurs décorrélés rendant le vol d'identité particulièrement complexe.

Mais à ces facteurs plutôt communs en termes d'authentification forte, d'autres plus exotiques peuvent s'y ajouter :

- Où nous trouvons-nous ? (géolocalisation depuis des lieux jugés « de confiance ») ;
- À quelle date/heure sommes-nous ? (accès autorisé en jours/heures ouvrés uniquement) ;
- Qu'entendons-nous ? (corrélation du bruit ambiant entre l'ordinateur et le smartphone) ;
- Que portons-nous ? (vêtements intelligents, bijoux, babioles...) ;
- Qu'avons-nous fait récemment ? (SMS reçus, activités des réseaux sociaux, etc.).

Ce présent article est voué à présenter de manière non exhaustive, un florilège d'actualité de facteurs communs et atypiques permettant de renforcer les phases d'authentification des systèmes jugés sensibles.

1.1 L'impuissance du couple « login/password »

Il est à présent avéré que le simple couple traditionnel « identifiant » (login, e-mail, matricule) et le mot de passe « password » associé n'est plus jugé suffisant pour sécuriser convenablement un accès.

Nombreux sont les « *leaks* », les piratages massifs médiatisés de bases de données en ligne, faisant fuiter ces précieux sésames que bien trop utilisent à la fois sur des services personnels comme professionnels **[HIB]**.

Pour renforcer la sécurité de ces crédentiels primaires, la notion de « *strong-authentication* » induisant de nouveaux facteurs est apparue.

L'attaquant ne peut plus usurper votre identité sur vos comptes et services en simple connaissance de votre couple login/password ; celui-ci doit de plus vous dérober votre carte magnétique, votre smartphone, se trouver au même endroit que vous ou encore falsifier votre voix ou vos empreintes digitales.

1.2 Du physique à la dématérialisation (hard/soft-token)

Historiquement, l'authentification forte s'est démocratisée (notamment en entreprise, pour les accès à privilèges) via un équipement dédié : le token physique (ou hard-token). Ce petit équipement, se greffant à votre porte-clés ou tenant dans votre poche, génère un nombre fini de chiffres que l'utilisateur doit recopier sur la mire d'authentification protégée (suite à son login/password). Ce token physique, dédié à la génération de mot de passe à usage unique (OTP) est lié à l'identité qui en est le propriétaire tout en induisant un second facteur « *quelque chose que l'on possède* ».

En fonction des points de vue ce boîtier peut être considéré comme un appareil très sécurisé, car dédié et limité à cette fonction ou bien quelque chose d'impersonnel que les utilisateurs se prêtent s'ils ont besoin d'un accès renforcé. La notion d'hard-token est intimement liée au fait d'avoir un équipement spécifique au second facteur d'authentification : un boîtier générateur d'OTP, une puce RFID ou un badge quelconque (figure 1).

Les hard-tokens représentent toutefois un coût certain en termes de déploiement et de gestion pour les DSI, et sont rarement destinés à un usage par des particuliers.

La démocratisation de l'utilisation des smartphones (que ce soit dans un domaine professionnel ou personnel) a permis de fournir une alternative crédible et relativement robuste aux mécanismes d'authentification forte utilisant des hard-tokens dédiés ou autres badges.

De nombreux éditeurs, ainsi que des communautés open source mettent à disposition des applications logicielles génératrices d'OTP (applications mobiles sur les stores iOS / Google) permettant de simuler la génération d'OTP tels que le faisaient les hard-tokens. Ces applications, que l'on nomme des « soft-tokens », entrent parfaitement dans les mœurs et dans l'ère du BYOD, se démocratisent (rares sont les usagers oubliant leur smartphone par rapport à un token physique) et réduisent les coûts de déploiement/gestion pour les entreprises (plus besoin d'une flotte de tokens physiques, les employés sont tous équipés d'un smartphone, gestion via un MDM, etc.).

L'activation en *self-service* du soft-token par l'usager lui-même, via un enrôlement par une clé secrète, un QRCode ou équivalent, est moins contraignant et mieux accepté avec de très bons retours d'expérience utilisateurs (UX). D'un autre côté, l'utilisation d'un équipement non dédié explicitement à cette fonction

Fig. 1 : Hard tokens RSA.

est potentiellement dangereuse (désavantages inhérents au BYOD), l'appareil mobile étant plus assujetti aux menaces extérieures que les hard-tokens.

1.3 La démocratisation et simplification de la MFA

Adopter l'authentification forte, qu'elle soit imposée par votre employeur pour des accès critiques ou tout simplement, car vous y êtes sensibilisé dans le cadre professionnel comme personnel engendre généralement plusieurs contraintes :

- Le mot de passe à usage unique et temporaire (OTP) n'arrive pas à temps ou pas du tout sur votre smartphone (hors couverture réseau) ;
- Vous vous êtes blessé à la main, empêchant vos empreintes d'être lues convenablement ;
- Le token physique ou la puce RFID est restée à votre domicile...

D'une manière générale, l'utilisation d'autres facteurs que le traditionnel couple « login/password » est chronophage et déplaît aux usagers en termes d'UX.

Ainsi, les solutions d'authentification forte visent à :

- Être portatives voire s'intégrant avec les équipements que vous portez habituellement sans induire de surcoût (le hard-token se voit délaissé par rapport aux soft-tokens ces dernières années) ;
- Limiter les actions de l'usager pour impacter le moins possible son expérience utilisateur et le temps consacré à ses phases d'authentification : recopier un code reçu par SMS ou généré par un soft-token se voit remplacé par un simple « *swipe* ».

- Être totalement transparent, sans nécessiter d'action de l'usager (pilule ingurgitée, analyse du son ambiant entre le smartphone et l'ordinateur, port de vêtements intelligents, etc.).
- Assurer une sécurité à toute épreuve, tâche la plus complexe.

2 Les One-Time Password (OTP)

Les OTP sont la forme la plus répandue en termes de mot de passe à usage unique et limité dans le temps faisant office de second facteur.

Qui n'a pas reçu par SMS un code de vérification à 6 ou 8 chiffres pour valider un paiement réalisé en ligne ?

Deux grandes familles d'OTP se distinguent : les TOTP basés sur le temps (epoch Unix) synchronisés entre l'ordinateur et le smartphone et les HOTP qui se fondent sur un compteur incrémental.

HOTP (*HMAC-based One-Time Password*) est un algorithme défini par l'OATH puis par l'IETF au sein de la RFC 4226 **[HOT]**. HOTP a pour objectif de définir un algorithme d'OTP simple et robuste basé sur une fonction d'HMAC, elle-même basée sur SHA-1. HOTP vise à répondre aux objectifs suivants :

- être basé sur un compteur ou une séquence ;
- être simple à implémenter même avec des ressources limitées (carte à puce) ;
- être utilisable sur des dispositifs ne proposant pas d'entrée utilisateur ;
- produire une valeur générée facilement lisible et manipulable par l'utilisateur ;
- être simple à resynchroniser pour l'utilisateur ;
- être basé sur un secret partagé fort (ie. >= 160 bits).

TOTP (*Time-based One-Time Password*) RFC6238 (draft) **[TOT]** : ces authentifieurs utilisent, en plus du secret partagé, un dénominateur commun qui est le temps. Chaque partie est synchronisée sur le temps universel (UTC). On utilise alors un Code NIP comme deuxième facteur d'authentification. Ces authentifieurs sont définis comme une technologie dite synchrone. Chaque minute ou toutes les 30 secondes, par exemple, ces authentifieurs affichent un nouveau « Token Code », le *One Time Password*.

Une des contraintes fortes qu'ajoute l'authentification forte, sous sa forme la plus répandue et démocratisée à savoir les OTP, concerne l'aspect « connecté » ou non au réseau.

L'avantage principal de ces OTP est qu'ils peuvent être utilisés sans couverture réseau (figure 2).

Dès lors que votre équipement (smartphone) est synchronisé temporellement de la même manière que

Fig. 2 : One Time Password

le service sur lequel vous cherchez à vous authentifier (NTP), et que les deux parties disposent du secret partagé, les TOTP générés seront valides. Toutefois l'usager aura tout de même la tâche de consulter son token/smartphone pour recopier manuellement l'OTP dans la mire d'authentification (UX dégradée).

2.1 Les OTP indirects connectés (swipe)

Pour améliorer l'expérience utilisateur, et afin de lui éviter la « fastidieuse tâche » de recopier manuellement un code à 6 ou 8 chiffres de son téléphone à l'ordinateur, la technique du « *swipe* » est de plus en plus implémentée. Impliquant un accès au réseau, et après avoir renseigné son login/password, le smartphone (connecté) reçoit une notification demandant de valider la connexion (simple bouton « ok », ou glissière à déplacer pour valider l'authentification, figure 3).

Fig. 3 : Swipe

Derrière ce mécanisme, bien plus rapide et simple pour l'usager, une requête de validation à usage unique est transmise depuis le smartphone vers le serveur pour valider l'authentification. Google a récemment mis en place ce mécanisme en plus de son « Google Authenticator » générateur d'OTP.

2.2 Sécurité, génération et mode de transport

Les OTP et en particulier leur générateur (soft/hard) ont en conséquence de nombreuses contraintes sécuritaires : ils génèrent des mots de passe qui vous sont propres après tout, même s'ils sont limités dans le temps et à usage unique.

Les applications (soft-tokens) sur smartphone se doivent d'être robustes et convenablement implémentées. Une récente étude menée par des experts en sécurité a révélé de nombreuses faiblesses d'implémentation dans des applications de génération d'OTP répandues **[HST]**.

En 2017, le NIST (*National Institute of Standards and Technology*) a officiellement considéré les OTP transmis par SMS comme étant obsolètes, désuets et jugés insécurisés **[NIS]** ; notamment via des attaques de re-routage ou de *Man in The Middle* (SS7) **[FBK]**.

Les applications mobiles permettant le « *swipe* » doivent disposer de protection anti-rejeu et anti-capture du trafic.

Dans tous les cas, il convient que tout OTP soit à usage unique et limité temporellement ; caractéristiques normalement intrinsèques et indispensables parfois délaissées par les éditeurs.

Mais au-delà des OTP, des soft/hard-tokens et des SMS d'authentification reçus, voyons d'autres facteurs d'authentification qui peuvent être employés.

3 Les facteurs comportementaux

Les facteurs comportementaux peuvent s'ajouter aux autres facteurs explicites comme le couple login/password et un OTP. Ces facteurs sont souvent vérifiés lors de mécanisme d'authentification adaptative :

- si une tentative d'authentification est réalisée depuis un poste de confiance, dans un lieu de confiance, à une heure normale, seul le premier facteur est nécessaire ;
- si la tentative est faite depuis un autre pays, en pleine nuit, à partir d'un poste et d'un navigateur inconnus jusqu'à présent, alors l'authentification va s'adapter au contexte en imposant d'autres facteurs (OTP par exemple).

3.1 Géolocalisation

Le facteur de géolocalisation n'est pas un facteur direct que l'usager renseigne suite à son login/password. Ce facteur entre en jeu pour déclencher une authentification adaptative en analysant la provenance de l'usager (à partir de sa position GPS, de la géolocalisation de son adresse IP, etc.).

Si la position courante diffère des habitudes déjà enrôlées et connues de l'application, alors le schéma d'authentification va se renforcer en demandant d'autres facteurs explicites.

Garder toutefois à l'esprit que des positions GPS peuvent être falsifiées aisément sur smartphone, tout comme l'IP de provenance via un VPN ou proxy par exemple.

3.2 Créneaux temporels

Tout comme le facteur influant sur l'authentification adaptative lié à la géolocalisation, le créneau temporel durant lequel la tentative d'authentification a lieu peut être considéré comme un facteur.

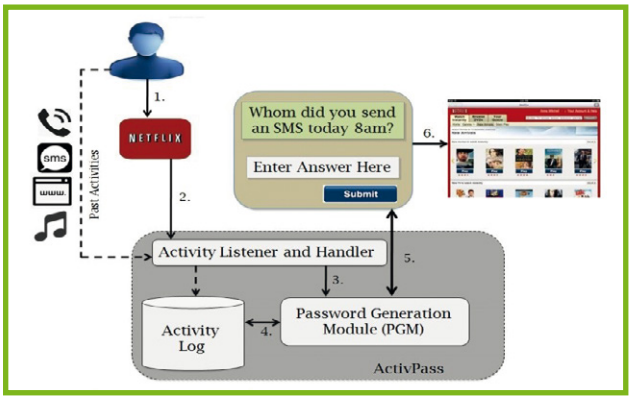


Fig. 4 : ActivPass

Si la tentative d'authentification a lieu au milieu de la nuit, un week-end, hors jours ouvrés, celle-ci peut être directement bloquée ou déclencher un autre schéma d'authentification demandant d'autres facteurs.

3.3 Activités passées

Suite à son identification par login/password, un second facteur explicite lié à son comportement passé peut s'y ajouter. C'est ce que la solution « ActivPass » propose. Ce système consiste à collecter quotidiennement divers journaux et logs en provenance d'un tas de ressources autorisées par l'usager, puis génère un mot de passe qui se traduira comme la réponse à une question de sécurité pour le lendemain **[ACT]**.

Le principe d'ActivPass se fonde sur des activités non fréquentes, mais mémorables pour l'utilisateur final. Le mot de passe généré dynamiquement peut s'inspirer d'événements et d'activités sur les réseaux sociaux, les appels téléphoniques reçus, les SMS ou encore la navigation Internet réalisée (figure 4).

Un exemple simple fourni par l'un des chercheurs est de fournir le nom de la personne qui a transmis le premier SMS de la journée. Cette information est non-récurrente, mais mémorable dans le laps de temps nécessaire à l'authentification.

Une expérimentation a été conduite sur la base de 70 individus, et 95% d'entre eux ont pu répondre à la question.

Les données personnelles nécessaires à la génération des questions quotidiennes d'authentification se fondent sur des services tiers, tels les réseaux sociaux. ActivPass extrait ces informations et les organise. À partir de là, plusieurs questions sont générées en arrière-plan. Dès qu'un utilisateur nécessite de s'authentifier, une (ou plusieurs) question lui est posée sur la base de sa récente activité. Les questions peuvent être textuelles ou une liste de choix multiples. Des indices peuvent également accompagner les questions.

ActivPass est d'ores et déjà disponible à l'essai (application Android, Facebook, Chrome...). Cette nouvelle méthode d'authentification, innovante, implique toutefois que de nombreuses informations personnelles soient brassées par la solution, et que cette collecte

doit être strictement définie et régulée pour le respect des données personnelles des utilisateurs, surtout à l'avènement de la RGPD.

4 Les facteurs biométriques

Quand on parle d'authentification renforcée, l'usage de la biométrie et les films de James Bond nous viennent immédiatement en tête. Ces principes se démocratisent, en particulier avec les smartphones qui s'équipent de lecteur d'empreintes digitales et de caméra pour la reconnaissance faciale. La biométrie devient accessible pour le commun des mortels.

4.1 Apparences physiques morphologiques

4.1.1 Reconnaissance faciale

La biométrie via reconnaissance faciale n'est plus à présenter. L'analyse des traits du visage par un équipement doté d'une caméra permet d'assurer l'identification et l'authentification d'un usager. De nombreuses solutions existent, plus ou moins sécurisées (déjouables avec une simple photo par exemple).

Amazon l'a récemment remis au goût du jour pour procéder à des paiements sur sa plateforme, via le « *selfie* » **[AMA]**, cette mode de s'auto-photographier.

L'idée est donc d'employer la caméra d'un smartphone ou la webcam d'un poste pour réaliser l'authentification de l'identité qui se présente à elle. Ce « *payment-via-facial-contorsion* » se veut robuste face aux attaques que l'on a pu observer dans l'actualité.

Toute la difficulté de la reconnaissance faciale est de détecter si c'est bien un « humain » et non pas une « photo ». Pour cela, plusieurs principes ont vu le jour, notamment proposés par Google.

En juin 2013, Google propose les « *funny faces* » afin d'améliorer le côté de reconnaissance « humaine ». Vous pouvez enregistrer un pattern d'une grimace (tirer la langue, sourire, etc.) pour accroître la sécurité de l'authentification.

La technique du « *Liveness Check* », toujours proposée par Google, consiste à demander à l'utilisateur cherchant à s'authentifier de cligner des yeux à un moment donné. Toutefois cette technique peut être rapidement contournée en modifiant une photo via un simple éditeur pour générer une animation...

En jonglant avec ces recherches et constatations, Amazon souhaite plutôt s'orienter vers le « *head-tracking* », suivant les mouvements de la tête et du visage, la détection infrarouge, l'imagerie thermique ou encore une combinaison de toutes ces méthodes.

4.1.2 Reconnaissance oculaire

L'iris est un des éléments d'authentification biométrique les plus utilisés dans tous les films d'espionnage hollywoodiens. En effet, les caractéristiques oculaires sont uniques pour chaque individu ; toutefois l'adoption en masse de ce procédé d'authentification reste encore éloignée du commun des mortels.

Sous réserve qu'un vairon se souvienne de l'œil enrôlé, et qu'il ne soit pas atteint de conjonctivite, la reconnaissance oculaire est l'une des plus robustes en termes d'authentification biométrique.

4.1.3 Reconnaissance labiale

« *Lip motion password* » ou « *lip password* » est le nom de ce mécanisme de 2FA inventé par le professeur Cheung Yiu-ming de l'Université Baptist de Hong-Kong (*Department of Computer Science, Hong Kong Baptist University (HKBU)*) **[LIP]**. Cette méthode innovante se base sur les mouvements des lèvres de l'individu, jugés uniques. En corrélant ces mouvements vis-à-vis du mot de passe indiqué textuellement, un *framework* appliquant une série d'algorithmes de reconnaissance permet de prouver ou non l'identité de l'usager tout en l'authentifiant (figure 5).

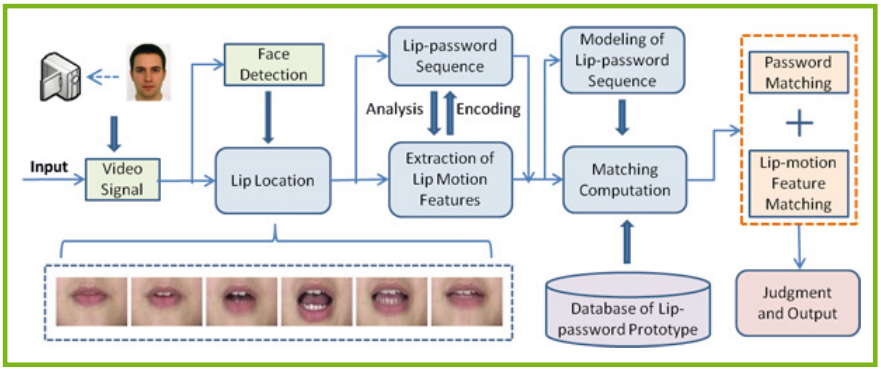


Fig. 5 : Lip Motion

Le professeur indique que cette nouvelle technique dispose de plusieurs avantages :

- La caractéristique dynamique des mouvements labiaux est résistante aux mimétismes, ce qui induit qu'un même mot de passe « prononcé » par le mauvais interlocuteur ne sera pas validé.
- Cette seconde vérification basée sur la combinaison des mouvements labiaux assure une robustesse du contrôle d'accès doublement sécurisée.
- Comparé à la traditionnelle authentification vocale, « *lip motion password* » n'est pas sujet aux bruits

environnants et à la distance (du microphone). De plus, ce mécanisme peut également être utilisé par les usagers muets !

- La réinitialisation de la séquence labiale peut être renouvelée régulièrement et simplement.
- Il n'y a pas de frontière de langue, en d'autres termes, quiconque le souhaite peut utiliser ce mécanisme.

Les « *duck-faces* » ont peut-être un intérêt après tout… Vous sentez-vous prêt à vous lécher les babines pour vous connecter sur vos réseaux sociaux favoris ?

4.1.4 Empreintes digitales (doigts/mains)

Classique, et de plus en plus adopté notamment au travers des smartphones ou des ordinateurs équipés de lecteurs d'empreintes digitales, ce mécanisme d'authentification s'avère difficilement falsifiable bien que de nombreuses approches et techniques ont fait leurs preuves.

4.2 Caractéristiques biologiques

4.2.1 Reconnaissance vocale

Comme pour la reconnaissance faciale, des empreintes digitales, ou de l'iris, la reconnaissance vocale est bien connue en tant que facteur d'authentification.

Souvent illustrée dans les films, celle-ci se doit d'être résistante au simple rejeu par magnétophone de la voix d'une victime.

C'est pourquoi plusieurs formes de reconnaissances vocales de différents niveaux de sécurité sont apparues avec le temps :

- La simple répétition d'une phrase « Bond, James Bond », pré-enrôlée et falsifiable via un magnétophone ;
- La lecture à l'instant T d'une phrase soumise par le système authentifiant, qui sur la base de la réponse (et d'un certain nombre de mots/sons pré-enrôlés) permet de valider l'authentification : bien plus robuste.

4.2.2 Reconnaissance cognitive

Les étudiants et chercheurs sous la direction de John Chuang de l'Université de Californie à Berkeley ont découvert une méthode d'identification/authentification par le biais d'une électro-encéphalographie (EEG). L'idée est d'analyser les ondes électriques cérébrales via un casque « *Neurosky Mindset* » puis de déchiffrer les signaux au travers d'un programme **[BRA]**.

Le fonctionnement du casque consiste à penser à quelque chose et l'outil enregistre le signal correspondant, avec

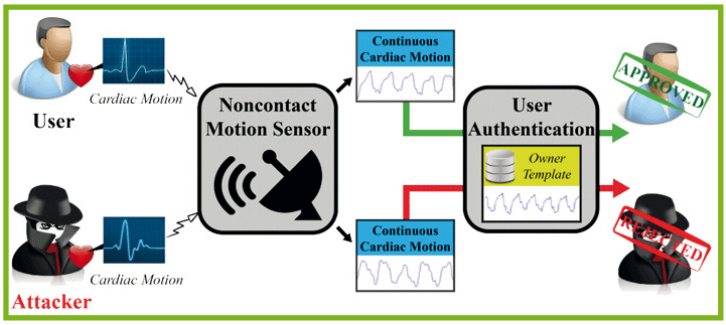


Fig. 6 : CardiacScan

un taux d'erreur inférieur à 1% selon les chercheurs. Le principe est aussi bien exploité pour autoriser l'accès à des fichiers sur une machine, que pour l'utilisation au travers de jeux vidéos.

4.2.3 Reconnaissance cardiaque

Nommé « *Cardiac Scan* », ce système d'authentification utilise un radar Doppler bas niveau pour analyser par onde et cartographier continuellement les dimensions de votre cœur, vous autorisant en conséquence l'accès à une application, un équipement ou à une zone restreinte dès lors que vous êtes suffisamment proche du capteur **[CAR]**.

Le système *Cardiac Scan* nécessite 8 secondes pour le scan de l'empreinte initiale (enrôlement) du cœur. Par la suite, le système analyse et cartographie continuellement l'enveloppe (pour être certain qu'un usager n'a pas usurpé votre équipement protégé à côté de vous), voir la figure 6.

Afin de valider cette nouveauté, 78 personnes ont éprouvé le système avec une précision de l'ordre de 98.61% et un taux d'erreur (EER) de 4.42% induisant une robustesse notable de la sécurité.

À cela, les chercheurs répondent que les effets sur la santé sont minimes puisque la force du signal est bien plus faible que les signaux Wifi et que les autres mécanismes d'authentification sur smartphone à base de radiation SAR (*Specific Absorption Rate*).

À l'heure actuelle, *Cardiac Scan* n'est pas réellement utilisable/commercialisable/industrialisable en raison de sa taille (*Proof-of-concept*), mais les chercheurs espèrent une adoption et une optimisation de celui-ci pour le voir, peut-être, un jour, présent sur chaque clavier d'ordinateur.

Une autre solution exploitant le rythme cardiaque est le bracelet Nimy **[NIM]**, conçu par la startup Bionym, qui est capable de récupérer l'électrocardiogramme d'un utilisateur pour une authentification multi-facteurs : le port physique du bracelet ainsi que l'empreinte du cœur.

4.2.4 Cavité auditive

Un autre modèle d'identification/authentification a vu le jour récemment, et celui-ci porte sur l'audition. L'idée est d'exploiter la caractéristique unique de la cavité auditive des utilisateurs, par exemple via un casque audio qu'ils porteraient.

Des chercheurs de l'entreprise NEC ont mis en place ce tout nouveau mécanisme d'authentification auditif avec plus de 99% de précision, via les cavités auditives qui diffèrent entre chaque individu. En émettant un signal audio et en enregistrant le signal retour (l'écho), l'appareil est capable de déterminer l'identité du porteur **[NEC]**.

Le principe est un simple « echo »/« réponse » et une analyse spectrale du signal. L'écho est basé sur la réflexion du canal auditif externe et de la membrane du tympan. La réponse est convertie en un signal unique, identifiant l'utilisateur (figure 7).

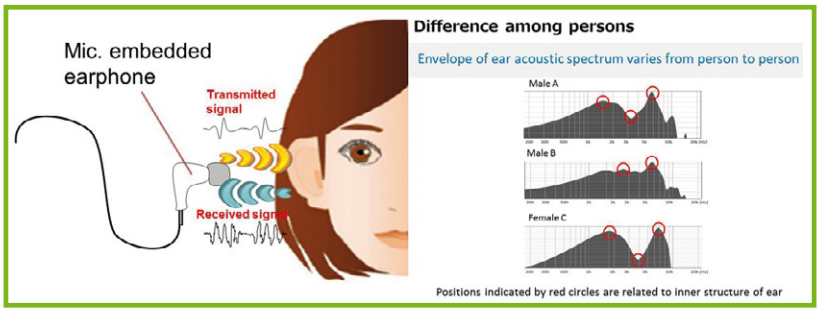


Fig. 7 : Cavité auditive

4.3 Marquage volontaire

4.3.1 Pilule bleue ou pilule rouge ?

Une alternative plus expérimentale a été mise en avant lors de la conférence WSJ/All Things Digital, Google et Motorola ont présenté une pilule à avaler permettant de vous transformer en mot de passe humain pour votre smartphone ou autres applications. La bien-nommée « *vitamin authentication* » pilule s'active dans l'estomac via les acides contenus dans celui-ci. Le *chipset* de la pilule génère ensuite un signal 18-bits relayé dans tout votre corps permettant l'authentification par le touché (le corps humain servant de conducteur pour le signal) **[PIL]**. Cette pilule, encore au stade expérimental, présente cependant des inconvénients, notamment de revoir votre petit-déjeuner.

4.3.2 Tatouage biométrique

Un tatouage sous forme de code-barres, sur l'avant-bras, a également été présenté lors de la conférence WSJ/All Things Digital. Conçu par l'entreprise MC10 **[BST]**, l'utilisateur n'a qu'à présenter son tatouage à un scanner pour s'authentifier auprès d'un système.

Cette autre technique d'authentification, généralement appelée « BioStamps » (figure 8) est principalement destinée au monde médical pour faciliter l'authentification des patients.

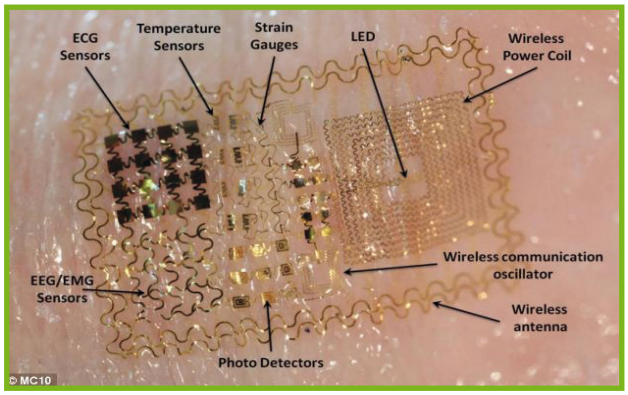


Fig. 8 : BioStamp

5 Les facteurs atypiques

5.1 Les vêtements

Les « SMART textiles » se démocratisent, notamment dans les défilés de mode. Toutefois savez-vous qu'il est à présent possible de stocker de la donnée dans des vêtements ? En effet, des chercheurs de l'Université de Washington arrivent à présent à manipuler la polarité de tissu magnétisé dans le but d'y stocker de la donnée ou des informations visuelles **[VET]**.

Les mots de passe pourraient en conséquence être contenus dans vos tenues, et ces données pourraient être aisément lues via le smartphone (technologie NFC, figure 9).

Cette technologie n'est pas nouvelle. Elle est déjà manipulée dans la mode ou encore dans la conception d'animaux en peluche ou des accessoires. Mais son rattachement au milieu de la sécurité et en particulier sur la notion de mot de passe peut permettre de considérer cette avancée comme un nouveau facteur d'authentification.

Résistant au lavage, au séchage et au repassage, ces textiles intelligents agrémentent une nouvelle fois la diversité des moyens d'identification/authentification/second-facteur que l'on constate à l'heure d'aujourd'hui.

Je ne peux m'empêcher de me projeter avec ce mécanisme d'authentification... Quelle tenue devrais-je en conséquence porter pour réaliser une connexion FTP en « *anonymous* » ?

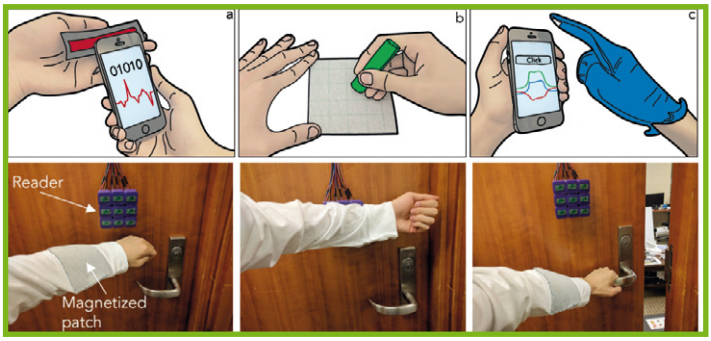


Fig. 9 : Smart textil

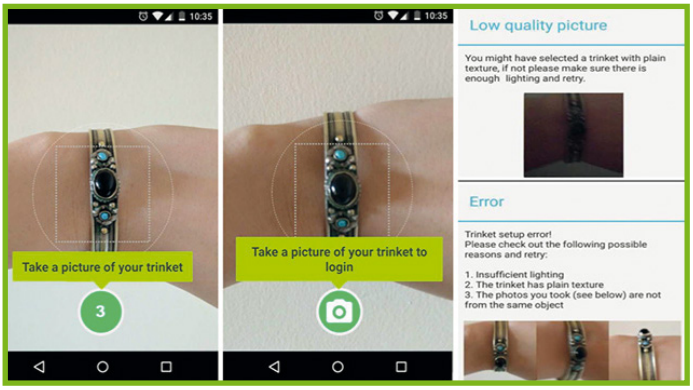


Fig. 10 : Pixie

5.2 Les bijoux et babioles

Un groupe de chercheurs de la *Florida International University* et Bloomberg LP a créé Pixie **[PIX]**, une solution basée sur une caméra (smartphone) qui se fonde sur ce que l'utilisateur porte : il est question de *trinket* (babiole) et sur ce que l'utilisateur sait (quelle babiole photographier/filmer, quel angle de visibilité, quel point de vue, etc.). Pixie repose donc sur un objet, quel qu'il soit, présent sur le porteur et bien évidemment en dehors du smartphone qui généralement fait office de second facteur.

Un collier, des boucles d'oreilles, une montre, un porte-clé, des chaussures, un tatouage, des lunettes... Toutes ces caractéristiques « portées » en dehors du smartphone permettent d'identifier/authentifier l'usager si la photo est correctement prise (figure 10). L'usager a la charge d'enrôler une première fois la photo qui servira de comparatif de sa « babiole » (distance de prise de vue, angle, etc.).

Comme pour l'authentification morphologique-biométrique du faciès, Pixie est vulnérable si un attaquant dispose d'une image correspondant à la « babiole » prise avec l'angle de vue et le recul de l'enrôlement initial. Mais Pixie s'avère résilient aux attaques par *shoulder-surfing*. Bien évidemment, les performances de Pixie peuvent être amoindries par manque de lumière ou lors d'authentification avec beaucoup de monde présent.

5.3 Ambiance sonore

« *Sound-Proof* » ! C'est une innovation de chercheurs suisses en termes d'authentification forte (2FA). Cette nouvelle technique ne nécessite aucune interaction supplémentaire de l'utilisateur tout en assurant un second facteur d'authentification.

Un groupe de chercheurs du *Swiss Federal Intitute of Technology* de Zurich a présenté à la conférence de sécurité USENIX leur nouvelle solution 2FA qui se fonde sur les sons ambiants **[SOU]**.

Cette solution opère comme suit : lorsqu'une ressource sur un poste est protégée par l'authentification forte « *Sound-Proof* », le micro du poste s'active automatiquement pour détecter les sons ambiants. En parallèle, une notification est envoyée sur le smartphone de l'individu identifié, qui de son côté (doté de l'application « *Sound-Proof* »), active également son micro pour comparer les sons ambiants.

S'il y a concordance entre les signatures audio enregistrées sur le poste et celles de l'équipement mobile (smartphone) de l'usager, alors l'authentification est réalisée avec succès. Si les signatures ne correspondent pas, l'authentification bascule vers une 2FA classique/de secours (authentification adaptative).

D'après les chercheurs, « *Sound-Proof* » permet d'économiser près de 25 secondes sur les phases d'authentification par rapport aux solutions de 2FA classiques via OTP.

En réalité, « *Sound-Proof* » n'est pas seulement une nouvelle méthode d'authentification double-facteur, mais on peut la considérer comme du triple-facteur : en effet, il est nécessaire de disposer d'un équipement portatif (« *ce que l'on a* ») et que celui-ci soit situé dans la même pièce/ambiance sonore que le porteur (« *où l'on est* » / « *ce que l'on entend* »).

Difficile pour quelqu'un de malintentionné d'usurper l'identité d'un usager équipé de « *Sound-Proof* ». Il faudrait par exemple que par chance, l'assaillant soit sur la même chaîne télévisée que la victime, au même instant.

Conclusion

D'une manière générale, il est fortement recommandé d'adopter l'authentification forte en ajoutant un (voire plusieurs) facteur(s) en plus du traditionnel couple login/password ; aussi bien pour vos comptes à privilèges, sensibles, services en ligne, professionnels et personnels.

L'authentification adaptative couplée à ces facteurs aussi divers que variés complexifie grandement la tâche des attaquants souhaitant usurper votre identité et vos comptes.

Ces assaillants n'en restent pas moins imaginatifs, en réalisant des attaques par ingénierie sociale particulièrement poussées, comme ce fût le cas avec des SMS malicieux visant les OTP générés par Google **[FAK]**, l'utilisation des attaques SS7 à l'encontre des OTP de Facebook **[FBK]**, ou encore l'utilisation de *frameworks* de *Social Engineering* dédiés au vol de facteurs 2FA présentés très récemment par le célèbre Kevin Mitnick **[BYP]**.

Restez vigilants, adoptez l'authentification forte pour tous les services qui le supportent et que vous utilisez **[TFA]**, et ne négligez surtout pas le renouvellement régulier (et robuste !) de vos crédentiels primaires : vos mots de passe.

Cette grande diversité de seconds facteurs s'explique peut-être aussi par le fait qu'il n'y en a toujours pas eu de trouvé avec un excellent compromis sécurité/utilisabilité.

PS : Je remercie Estelle pour ses relectures attentives et salutations à toute l'équipe de SYNETIS pour nos échanges constructifs :-) !

Retrouvez toutes les références de cet article sur le blog de MISC : https://www.miscmag.com/ .

WEB AUTHENTIFICATION / PASSWORD RESET : REX DE BUG BOUNTY

Yann CAM
Security Consultant @SYNETIS
Security Researcher @ASafety

mots-clés : *AUTHENTIFICATION / IDENTIFICATION / PASSWORD RESET / SESSION / TOKEN / PASSWORD POLICY / BRUTE-FORCE / ÉNUMÉRATION*

Présentation de faiblesses communément observées lors de recherches de vulnérabilités dans le cadre de Bug Bounty publics et privés, à l'encontre des modules web d'authentification et de réinitialisation de mot de passe.

Mettre en place un mécanisme d'authentification (d'utilisateurs, administrateurs, clients) sur un site web va généralement de pair avec une fonctionnalité de « *mot de passe oublié* » permettant la réinitialisation du *password*. Le présent article est voué à dresser une liste (non-exhaustive) des faiblesses communément observées sur de tels modules en agrémentant via des cas concrets issus de Bug Bounty.

1 Authentification web et faiblesses communes

1.1 Énumération passive / OSINT

L'énumération est une des faiblesses les plus répandues au sein des formulaires d'authentification bien que non critique. En effet, il est souvent possible de les exploiter pour déterminer si un login (généralement sous forme d'une adresse e-mail) est associé à un compte ou non. L'intérêt pour l'attaquant est de concevoir une liste de logins valides (sans disposer, encore, de leurs mots de passe respectifs).

Un compte est d'autant plus sécurisé, et donc contribue à la sécurité globale de l'applicatif, lorsque ses deux principales composantes associées à la confidentialité sont inconnues de l'attaquant, à savoir le mot de passe, mais également le login.

L'énumération passive consiste à dresser une liste de logins potentiels (*wordlist*) à partir de ressources tierces (réseaux sociaux, OSINT), et de les tester un à un (via un script) à l'encontre du formulaire d'authentification.

L'énumération est rendue possible via une distinction comportementale de la réponse du formulaire entre un login existant et inexistant (avec un mauvais mot de passe dans les deux cas).

Cas typique :

- Tentative avec le login inexistant « **unknown@target.com** », retour : « *Identifiant inconnu* » ou « code HTTP 302 » ;
- Tentative avec le login existant « **admin@target.com** », retour de l'application : « *Mot de passe incorrect* » ou « code HTTP 403 ».

Cette distinction de message d'erreur ou de code retour suffit à l'attaquant pour déterminer si un login soumis est associé à un compte ou non. Il réalise une première fuite d'informations sensibles : une liste d'identifiants pouvant faciliter des attaques ultérieures (*spear-phishing*, brute-force, *social-engineering*, *post-exploitation* sur un périmètre plus large, etc.). C'est une exfiltration indirecte de données du *backend*.

De nombreux outils facilitent de telles énumérations, tel que **theharvester** **[01]**, capable de dresser des listes d'adresses e-mails valides associées à un domaine (@target.com) via des techniques d'OSINT (*Open Source INTelligence*) sur les moteurs de recherche.

Ou encore **linkedin2username** **[02]**, permettant à partir du réseau social professionnel LinkedIn, de générer des logins probables sur une application *corporate* via les collaborateurs présents sur ce réseau. Les logins sont généralement sous la forme « pnom », « p-nom », « prénom.nom », etc.

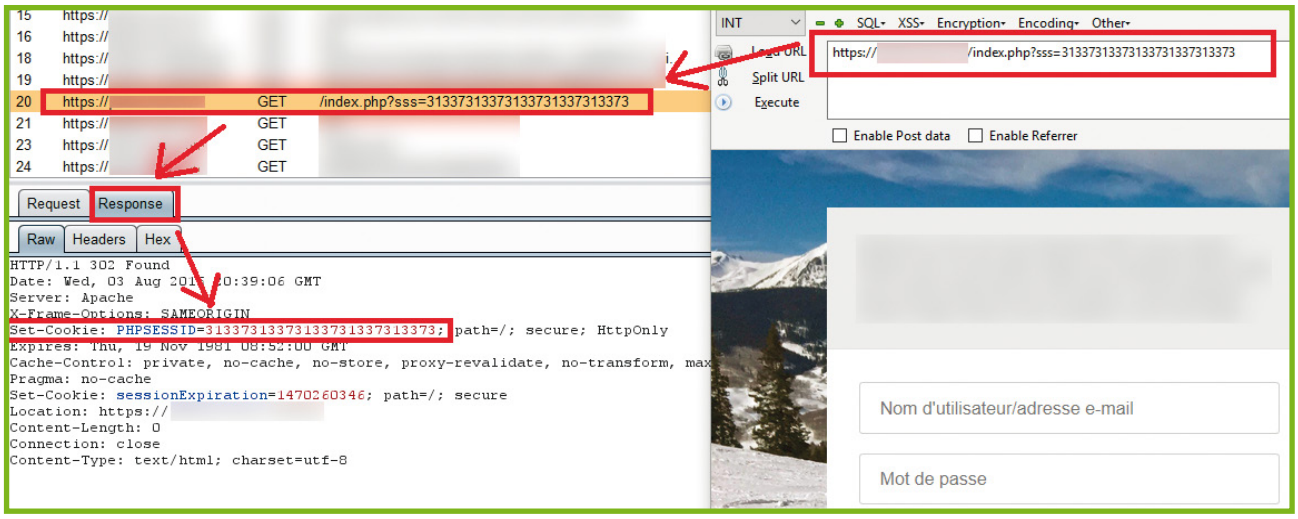


Fig. 1 : Illustration d'une session fixation attack rencontrée sur un bug bounty.

1.2 Énumération active/ incrémentale ou déductible

L'énumération active, incrémentale ou déductible permet de gagner en exhaustivité des logins associés à des comptes sur une plateforme ciblée. Si entre deux inscriptions (création de compte), l'attaquant observe que les logins évoluent avec une suite logique (M290690 puis M290691) il peut automatiser la génération de logins pour une énumération.

Celui-ci automatisera les tests d'énumération depuis le login M000000 jusqu'à M999999, et dressera une liste bien plus complète, car le format des identifiants s'avère prédictible.

L'attaquant dispose de l'intervalle des logins de l'applicatif ainsi que du format de ceux-ci, information qui n'est pas d'une forte criticité, mais facilitant grandement ses futures attaques.

1.3 Gestion des sessions

L'identifiant de session, généralement échangé au travers des cookies, est une des données les plus critiques qui doit être soigneusement protégée entre l'usager et l'applicatif.

Dès lors que celui-ci est dérobé, l'attaquant est en mesure de se connecter sous le compte de la victime sans avoir connaissance de son mot de passe. Il usurpe la session (*hijacking*).

1.3.1 Non-renouvellement avant/après authentification

La valeur du cookie de session doit disposer de plusieurs caractéristiques, hélas, trop peu souvent mises en pratique :

- Unicité, d'où une entropie/un aléa robuste utilisé lors de sa génération ;
- Générée et définie côté serveur, sans qu'un attaquant ne puisse définir sa valeur (*session fixation attack*, *HTTP Response Spliting*...) ;
- Limitée dans le temps (ne pas avoir des sessions de 10 ans) ;
- Être gérée correctement à la création, destruction et renouvellement.

C'est sur ce dernier point qu'une observation souvent faite est à souligner : les applicatifs attribuent un cookie de session dès lors qu'une quelconque page (non-authentifiée) de l'application est atteinte ; et cette valeur du cookie de session est conservée post-authentification.

Intrinsèquement à la notion de « session », il convient d'en générer pour chaque espace. Un cookie pour une session non-authentifiée, renouveler celui-ci (ou générer un nouveau cookie) pour l'espace authentifié standard, et renouveler à nouveau pour des espaces à forts privilèges.

À titre d'exemple, Amazon, la plateforme commerciale, implémente ces mécanismes : si vous êtes un habitué de Amazon, vous devez être constamment authentifié (cookie de session « client simple »). Toutefois, dès lors que vous procédez à un achat et que vous entrez dans le processus de paiement (choix de la carte bancaire, etc.), une réauthentification est nécessaire générant une autre session plus sensible et donc plus courte.

Pour une application, il est donc conseillé de détruire le cookie de session obtenu en mode « non-authentifié », pour en générer un nouveau post-authentification.

1.3.2 Session fixation attack / HTTP Response Spliting

L'attaque par fixation de session, également réalisable par HRS (*HTTP Response Spliting*), se synthétise par la capacité de l'attaquant à forger une URL ou une requête GET/POST vers un applicatif, pour y définir lui-même la valeur du cookie de session.

Ainsi, l'attaquant génère une URL telle que **https://target.com/index.php?sessid=133713371337** et dès lors qu'il accède

à cette URL, le serveur lui retourne dans les en-têtes un **Set-cookie : SESSID=133713371337**.

La dangerosité de cette faiblesse est qu'en incitant une victime à se rendre à cette URL, l'attaquant prédéfinit la valeur de la session de sa victime. L'usager légitime (avec le cookie **SESSID=133713371337**) s'authentifie sur l'applicatif et l'attaquant n'a plus qu'à utiliser la même valeur du cookie pour usurper son identité (figure 1).

Si les valeurs des cookies de session étaient renouvelées constamment entre un espace non-authentifié et authentifié, ce type d'attaque ne serait pas exploitable.

De plus, permettre de définir via un paramètre **GET** ou **POST** la valeur d'un cookie (ou un tout autre *header* via une **HRS**, pouvant engendrer des attaques XSS) est une pratique à proscrire.

1.3.3 Prédiction et faiblesse des tokens de sessions

Dans certains cas, la valeur du cookie de session est prédictible, ce qui peut amener à des attaques d'une grande criticité comme le vol de n'importe quel compte utilisateur.

Pour un autre bug bounty, il m'est arrivé de créer plusieurs comptes utilisateurs et de m'authentifier via différents navigateurs en parallèle sur ces comptes. En observant les cookies affectés à chacun d'eux, je me suis rendu compte de leur caractère prédictible :

```
R4RGMGX2XKFC
R4RGMGX2XKFD
R4RGMGX2XKFE
[...]
```

Ces valeurs de session sont clairement incrémentées et en conséquence un attaquant peut facilement réaliser un script qui itère sur ces valeurs (**R4RGMGX00000** jusqu'à **R4RGMGXZZZZZ**) en observant les retours de l'application (code HTTP ou code source indiquant « *Bonjour* **login** »). Ainsi, il a été possible de dérober l'accès à de nombreux comptes actuellement connectés à l'applicatif.

La génération de la valeur d'une session doit nécessairement se réaliser à partir d'un aléa (**PRNG** robuste) tel que **/dev/random**. L'utilisation du *timestamp* courant, d'un hachage du login ou de toute autre information d'identité est à bannir.

1.4 Brute-force actif (online)

Le brute-force actif, sollicitant directement l'application cible et le réseau est à différencier du brute-force passif (*offline*), qui à l'inverse, effectue ses traitements localement lorsque les hashs des mots de passe sont en possession de l'attaquant.

Avec le brute-force actif, l'attaquant teste des couples d'inconnus : un login associé à un mot de passe. Si l'attaquant connaît un login existant, il va se focaliser sur le mot de passe uniquement (via des *wordlists* par exemple). Si l'attaquant connaît un mot de passe, il va s'attarder à trouver le login correspondant. C'est là où l'intérêt d'avoir un login unique tout aussi confidentiel qu'un mot de passe s'avère robuste.

Les comptes « admin », « administrateur » ou « test » sont souvent des logins préexistants. Couplés à une *wordlist* issue de *leak* ou la traditionnelle Rockyou **[03]**, un attaquant peut employer des outils tels que Burp Intruder **[04]** ou Hydra **[05]** pour dérouler un brute-force à l'encontre d'une application jusqu'à trouver des mots de passe valides.

Il est impératif de bloquer les tentatives de brute-force au niveau d'une page d'authentification. Soumettre une grande quantité de mots de passe pour un même compte dans un court intervalle, ou tenter de multiples identifications/authentifications à partir d'une même IP source sont des tâches suspicieuses.

Mais que faire en termes d'implémentation lors de détection de brute-force ?

1.4.1 Blocage automatique du compte

« *Pour X tentatives d'authentification échouées pour un même identifiant, bloquer le compte* ». Cette méthode est déconseillée, car en cas de brute-force par un attaquant, elle impacte l'utilisateur légitime qui ne peut plus s'authentifier. De plus, comment le déverrouillage du compte est-il réalisé ?

Si l'utilisateur peut se débloquer en *self-service* (envoi d'un e-mail de déblocage), cette solution peut limiter les impacts. Toutefois, l'attaquant peut spécifiquement cibler cet utilisateur, et bombarder son compte pour le bloquer indéfiniment malgré les tentatives de déblocage de l'usager : déni de service du compte.

L'attaquant pourrait automatiser cela sur tous les logins qu'il a pu déterminer par énumération : pour chaque login, réaliser X+1 authentifications en échec pour bloquer les comptes, et ce, continuellement. L'ensemble des comptes serait bloqué, et les usagers n'arriveraient pas à les déverrouiller en *self-service* : déni de service global.

Si le déverrouillage nécessite une action du support/helpdesk (appeler tel numéro), les équipes pourraient être submergées d'appels, causant un déni de service de l'applicatif et du standard téléphonique également : critique.

Les mires ADFSv2 proposent une authentification web sans anti-brute-force. En conséquence, il est aisé de bloquer en masse des comptes d'un domaine Windows en bombardant une telle page avec des logins valides et mots de passe invalides.

1.4.2 Blocage temporaire du compte

Un blocage temporaire (pendant M minutes) permet de limiter grandement les attaques par brute-force. Mais cela impacte également la productivité de l'utilisateur légitime et son expérience (UX).

Ce type de blocage est à implémenter côté applicatif. En effet, si une application est rattachée à un référentiel d'authentification centralisé (AD, LDAP), le blocage du compte peut être intrinsèque à la technologie d'annuaire (3 échecs de *bind* successifs bloquent le compte). Ce qui en plus d'impacter l'accès à l'application, impacte également toutes les autres applications exploitant l'annuaire : le *scope* de l'attaque s'étend.

Quid du temps de blocage ? Si un attaquant a la capacité de tester 100 mots de passe à la seconde, mais qu'il est bloqué tous les 10 essais pendant 10 minutes, il n'arrivera qu'à tester 432 mots de passe à la journée. S'il est bloqué tous les 3 essais pendant 10 secondes, il arrivera à tester 25 920 mots de passe dans la journée.

1.5 Pavés numériques aléatoires

Le milieu bancaire implémente souvent ce type de mécanisme d'authentification : les pavés numériques aléatoires. Partant d'une bonne intention afin de contrer les *keyloggers* (puisque le mot de passe numérique est entré en « cliquant » sans utiliser le clavier) ainsi que les *mouse-recorders* (enregistrement des clics de souris impossible puisque le pavé numérique est affiché aléatoirement), les implémentations existantes peuvent néanmoins engendrer des faiblesses de sécurité.

Ces pavés numériques aléatoires indiquent que les mots de passe ne sont constitués que de caractères numériques. Leur longueur est aussi souvent indiquée dans un message explicite « *merci d'indiquer votre code à 6 chiffres* » : l'attaquant à toutes les informations pour optimiser son brute-force (figure 2).

Bien que côté client le mot de passe semble entré de manière sécurisée, la requête POST qui émane du navigateur à destination du serveur contient bien trop souvent ce *password* en clair, et donc aisé à brute-forcer.

Les solutions de pavés numériques aléatoires jugées sécurisées sont celles qui transmettent l'identifiant du pavé généré pour la session courante, ainsi que la combinaison des cases (chiffres) qui ont été cliquées. La vérification est réalisée côté serveur. Trop peu d'implémentations exploitent ce principe.

1.6 Autocomplete, maxlength, pattern et réflexion : éradiquer les XSS

L'attribut HTML **autocomplete** est conseillé d'être mis sur **off** pour les champs **input** de connexion (*login* et *password*). Cet attribut empêche le navigateur de conserver une version en clair des crédentiels pour les réinjecter automatiquement à la prochaine connexion.

Pour des applications critiques, dont les mots de passe ne sont pas à éparpiller, cet attribut évite qu'ils soient enregistrés dans les navigateurs (et donc dérobés si la machine est compromise par un attaquant).

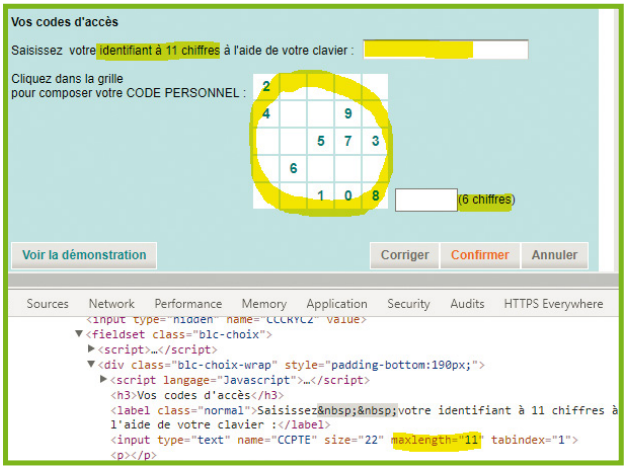


Fig. 2 : Pavé numérique aléatoire avec détails de la password policy.

Les attributs de la balise **input** du mot de passe, **maxlength** ou encore **pattern**, permettent d'indiquer des indices précieux à l'attaquant sur la politique de mot de passe appliquée (figure 2) :

- **maxlength=6** : le champ du mot de passe n'accepte que 6 caractères maximum ;
- **pattern=[A-Za-z]{3,6}** : mot de passe alpha uniquement (*lower*/*upper*) de 3 à 6 caractères.

Une autre observation régulièrement faite concernant les pages d'authentification est la « réflexion » du login dans le code source. Si l'usager se trompe de mot de passe, il est redirigé vers la page d'authentification avec un message d'erreur, et son login est souvent pré-renseigné. Seulement, cette réinjection de données POST peut engendrer des vulnérabilités de type XSS directement dans la page d'authentification.

Une telle vulnérabilité XSS permet à un attaquant de modifier le comportement du formulaire d'authentification (vol des crédentiels en clair via un *hook*), d'activer un *keylogger*, compromettre tout le navigateur (framework BeEF **[06]**) et de contourner toutes les protections anti-CSRF en place dans l'application.

Éradiquer les XSS dans l'applicatif tout en équipant tous les formulaires de protection anti-CSRF est une action obligatoire pour tous les applicatifs web d'aujourd'hui.

1.7 Password Policy disclosure et contrôles client-side

Parfois, les contrôles du login ou du mot de passe sont directement implémentés côté client en JavaScript. Intrinsèquement insécurisés (puisque l'attaquant peut visualiser ces contrôles et les modifier), l'application se doit également d'appliquer ces mêmes vérifications côté serveur.

Exposer *client-side* de tels contrôles (la politique de mot de passe) permet à l'attaquant d'affiner ses attaques brute-force (figure 2).

Dans certains cas, lors de l'absence de contrôle côté serveur, il est possible d'altérer les requêtes de changement de mot de passe (ou d'inscription), pour définir un mot de passe allant totalement à l'encontre de la politique en place, tel que « **x** » (1 caractère alpha). Ce contournement de politique ne devrait pas être possible.

2 Réinitialisation de mot de passe perdu ou oublié et faiblesses communes

2.1 Énumération active/passive

Au même titre que les formulaires de connexion, les pages de « *mot de passe oublié* » sont très fréquemment sujettes à des énumérations de comptes actives ou passives.

Lorsque l'usager est invité à indiquer son adresse e-mail, en vue de réinitialiser son mot de passe, l'application peut indiquer un message tel que « *un e-mail vient de vous être envoyé* » ou « *identifiant inconnu* ».

Comme détaillé précédemment, cette différenciation permet de déterminer l'existence ou non d'un compte utilisateur sur le système.

Une différence (notable) est toutefois à préciser, contrairement à l'énumération via les pages de connexion : cette présente énumération par le biais du « *reset password* » n'est pas totalement transparente et invisible pour la victime, puisque celle-ci reçoit un e-mail.

Une telle énumération s'automatise aisément, même en présence de jeton anti-CSRF.

```
import requests
import sys
import json
if len(sys.argv) > 1:
        file = sys.argv[1]
else:
        print "[*] Usage : python " + sys.argv[0] + " <file>"
        exit(0)
session = requests.Session()
URL = "https://target.fr/api/auth/forgetPassword"
for login in open(file,'r').readlines():
        login = login.strip()
        r = session.post(URL, json={"email": login})
        if "user_not_found" not in r.text:
                print "[+] ["+login+"] exists !"
```

2.2 Réinitialisation via e-mail

2.2.1 Mailbombing et saturation

La réinitialisation de mot de passe par e-mail implique de fait l'envoi d'e-mails par l'applicatif et la réception de ceux-ci sur la boîte-mail des usagers. Cette boîte-mail est censée vous appartenir et ne pas être partagée/compromise. Autrement dit, l'étape d'identification est assurée, par essence, par l'accès à votre boîte-mail personnelle.

Dès lors qu'un mécanisme de réinitialisation de mot de passe génère des e-mails, plusieurs points de contrôle sont à vérifier, notamment si le formulaire de réinitialisation est appelé plusieurs fois pour un même usager : la victime (utilisateur légitime) ne devrait recevoir qu'un unique e-mail de réinitialisation, même si le formulaire est appelé plusieurs fois. Ou un e-mail par jour par exemple. Dans tous les cas, une limitation d'envoi doit être mise en place.

Si le formulaire est soumis en boucle pour un même usager, les effets de bord suivants peuvent survenir :

- DoS / saturation de la boîte-mail de l'usager ;
- DoS / saturation du serveur SMTP applicatif qui se charge de délivrer les e-mails ;
- Mise sous liste noire (*blacklisting*) de vos serveurs d'envoi pour activité inhabituelle, entachant l'image de marque, la qualité du service, et la confiance des utilisateurs.

2.2.2 Qualité des e-mails / spam

La qualité des e-mails envoyés par l'applicatif est primordiale. En particulier concernant les e-mails de « réinitialisation de mot de passe » ou de « déverrouillage de compte ». Cette qualité, pour que ces e-mails n'arrivent pas dans les *spams* de l'usager, dépend d'en-têtes et d'entrées DNS qui peuvent être configurés, notamment DKIM, DMARC et SPF.

Au cours d'un autre bug bounty, j'ai pu observer que les e-mails de « Bienvenue » (lors de l'inscription) arrivaient bien dans ma boîte de réception. Mais à l'inverse, les e-mails de réinitialisation de mot de passe (mot de passe perdu) étaient catégorisés comme *spam*.

L'attaque mise en œuvre pour exploiter cette faiblesse consistait à :

- Cibler une victime **victim@target.com** ;
- Dérouler le processus de « *mot de passe perdu* » pour cette victime ;
- La victime reçoit le mail avec la procédure de renouvellement de mot de passe dans ses *spams* ;
- En parallèle, en tant qu'attaquant, j'envoie un e-mail à cette victime via une adresse proche du domaine cible « **@target.fr** » par exemple, aux couleurs des e-mails légitimes, invitant l'usager à confirmer/redéfinir son mot de passe sur une page que je contrôle.

Mon e-mail en tant qu'attaquant est fiable, car les en-têtes DKIM, DMARC et SPF sont convenablement définies pour mon domaine « **target.fr** », ce qui n'est pas le cas du domaine ciblé « **target.com** ». En conséquence, mon e-mail frauduleux arrivera dans la boîte de réception de la victime.

Par déduction, quel e-mail sera le plus crédible aux yeux de notre victime crédule ? Le légitime dans les spams, ou le frauduleux dans la boîte de réception ?

2.2.3 Réception du mot de passe en clair par e-mail

Lors du déroulement d'un processus de « *réinitialisation de mot de passe* », si l'e-mail reçu sur votre boîte contient un message du type « *Votre mot de passe est : MyInitialPassword* » avec votre mot de passe à vous, défini lors de l'inscription, alors vous avez du souci à vous faire (tout comme les responsables de l'applicatif).

En effet, si votre propre mot de passe vous est renvoyé en clair :

- Cela signifie que l'applicatif les conserve en clair, sans hachage quelconque, ou chiffrés de manière réversible ;
- Les responsables de l'applicatif peuvent connaître tous les mots de passe des usagers/clients ;
- Si un attaquant met la main sur la base de données de l'application, il dispose d'un *leak* sensible avec tous les mots de passe en clair des inscrits ;
- Si vous avez la mauvaise habitude d'utiliser toujours le même mot de passe, considérez tous vos autres comptes *online* voire votre boîte-mail comme compromis ;
- En ces heures sombres avec la RGPD, conserver des mots de passe en clair côté *backend* n'est pas recommandé pour une application...

À l'heure d'aujourd'hui, votre boîte-mail principale est clairement le service le plus sensible dont vous disposez : si celle-ci est compromise, un attaquant pourra y dénicher un très grand nombre d'informations et étendra son périmètre de compromission :

- Un accès à votre boîte-mail permet de changer les mots de passe (réinitialisation) de la plupart des services en ligne où vous êtes inscrit ;
- Beaucoup d'e-mails de « *rappel de votre mot de passe* » s'y trouvent, l'attaquant n'a qu'à fouiller.

Ainsi, dans l'intérêt de conserver la confidentialité de vos informations, les mots de passe d'une application tierce ne sont pas voués à être dupliqués/transférés ailleurs que dans l'application elle-même. Envoyer un mot de passe en clair sur une boîte-mail n'est pas une bonne pratique.

2.2.4 Réception d'un mot de passe auto-généré

La réception d'un mot de passe auto-généré est considérée comme « mieux sécurisée » que la retransmission du mot de passe originel en clair, mais engendre de nouvelles problématiques de sécurité et pas des moindres.

Les mots de passe auto-générés transmis par e-mail induisent toujours le souci de déporter une information hautement confidentielle sur une source tierce, qui n'est pas forcément de confiance (votre boîte-mail), centralisant une nouvelle fois en un même endroit toutes sortes de crédentiels.

Les mots de passe auto-générés informent également de la politique de mot de passe en vigueur. Casse des caractères, chiffres, symboles, longueur prédéfinie : toutes ces informations permettent à l'attaquant d'affûter ses prochaines attaques par brute-force. Par exemple, suite à 3 réinitialisations de mot de passe successives et réception des passwords « XOLIMX », « TOLRAP » puis « MACYAN », on en déduit que les mots de passe auto-générés sont tous en majuscule, sans chiffre ni symbole, et de longueur 6 (brute-force a posteriori aisé).

L'autre problématique bien plus critique avec les mots de passe auto-générés engendre un déni de service d'un ou plusieurs comptes (en fonction de l'énumération réalisée au préalable). Si un attaquant réalise un script qui génère en boucle pour un ou plusieurs comptes des requêtes de « *mot de passe oublié* », alors :

- Les usagers légitimes vont recevoir un très grand nombre d'e-mails avec des nouveaux mots de passe auto-générés ;
- Les mots de passe auto-générés ne seront utilisables qu'un temps très court (tant qu'un nouveau auto-généré n'est pas de nouveau envoyé) ;
- Le/les usager(s) légitime(s) sont donc dans l'impossibilité de se connecter à leur compte, étant donné que leur mot de passe est auto-généré et change continuellement : déni de service ;
- La régénération d'un mot de passe par le serveur, et son hachage en base sont des opérations pouvant être coûteuses en ressources (**bcrypt**). À grande échelle et pour beaucoup de comptes, les performances du serveur peuvent être impactées sévèrement.

2.2.5 Réception d'un lien de réinitialisation

Une des meilleures pratiques consiste à envoyer des e-mails de réinitialisation de mot de passe contenant un lien pointant vers l'application pour précéder au renouvellement. Ces liens disposent généralement d'un *token* aléatoire via un paramètre **GET**.

Ce *token* **GET** doit nécessairement être valable qu'une seule fois (unicité) et limité dans le temps (quelques dizaines de minutes). Hélas, beaucoup d'implémentations ne respectent pas cela :

- liens de réinitialisation réutilisables à l'infini avec le même *token* sans limitation temporelle ;
- *token* « aléatoire » qui n'est autre que le MD5 du login (donc calculable pour tous les utilisateurs, avec vol de n'importe quel compte) ;
- *token* prédictible basé sur le **timestamp** au moment de la demande de réinitialisation.

Pour un autre bug bounty d'un site d'e-commerce de renom, j'ai pu observer que le formulaire atteint par le biais d'un tel lien unique, ne validait ni n'exploitait plus du tout ce *token* ni l'identité l'ayant généré. Il suffisait de soumettre les données POST **userid** (arbitraire,

n'importe quel compte), `passwd1` et `passwd2` au *endpoint* `/resetpassword` et le tour était joué. Quid de l'utilité d'un *token* aléatoire à usage unique si le formulaire final permet de voler n'importe quel compte client ?

2.3 Autres méthodes de réinitialisation

2.3.1 Questions d'identification

Les « *questions personnelles* » que vous avez pré-enrôlées sur l'applicatif avec vos propres réponses ne sont plus considérées comme sûres en 2018. En plus d'être un mécanisme de réinitialisation boudé par les usagers en termes d'UX, cette méthode d'identification dépend fortement des questions et des réponses.

C'est justement ces questions et réponses qui peuvent aujourd'hui être mises à mal, notamment avec des approches OSINT et les réseaux sociaux.

- *Quel est le prénom de votre animal de compagnie ?*

- *Quel est le nom de jeune fille de votre mère ?*

- *Dans quel lycée étiez-vous ?*

Ces types de questions peuvent facilement trouver leurs réponses sur des sites tels que Facebook ou Copainsdavant, mettant à mal la sécurité de votre mot de passe en conséquence.

Ces formulaires de questions anodines bien que personnelles, n'étant pas suffisamment suspectes pour vous alerter, peuvent également être exploités via des attaques par *clickjacking* pour *gagner le dernier iPhone* si l'applicatif n'est pas protégé contre ce type d'attaque via des en-têtes HTTP tels que `X-Frame-Options`.

2.3.2 « Merci de contacter le support au numéro suivant »

Lorsque le processus de réinitialisation de mot de passe vous invite à « *contacter le support* » pour procéder au renouvellement, notamment par téléphone, imaginez la surcharge d'appels téléphoniques du *helpdesk* si l'ensemble des comptes s'avère bloqué.

Il est recommandé de permettre une autonomie à l'usager final dans sa réinitialisation de mot de passe (*self-service*), pour soulager les ressources internes de support et éviter les pics d'activité/saturation du réseau téléphonique en cas d'attaque.

2.3.3 Courrier postal avec code d'accès

Cette méthode est également répandue. Déjà rencontrée lors de bug bounty, je me suis retrouvé avec plusieurs lettres similaires reçues quelques jours après mes tests, contenant pour chacune d'elles un « *nouveau code d'accès* ».

Cette méthode de réinitialisation peut engendrer des coûts supplémentaires particulièrement onéreux pour les propriétaires de l'application en cas de bombardement de demande de réinitialisation pour de multiples usagers.

3 Mitigation, prévention et sécurité

3.1 Anti-robotisation

Pour éviter toutes les attaques et faiblesses détaillées précédemment et automatisables via des scripts (brute-force, blocage de comptes en masse, *mailbombing*, énumération, etc.), l'anti-robotisation est conseillée. Comprendre : implémenter un Test de Turing, une solution de CAPTCHA.

Une idée reçue, et totalement fausse, concerne les mécanismes anti-CSRF (jeton en session, analyse du *referer*) : ces protections bien qu'indispensables, ne permettent pas de protéger un module d'authentification contre la robotisation.

Les CAPTCHAs sont rarement appréciés. C'est pourquoi, l'idée est d'implémenter un CAPTCHA qui n'apparaît qu'après X tentatives de soumission d'un formulaire, ainsi l'expérience utilisateur ne serait pas dégradée au départ, mais seulement après plusieurs essais infructueux.

3.2 Limitation de requêtes

En complément du CAPTCHA, appliquer des limites au niveau des requêtes (par utilisateur, ou par adresse IP source) est une excellente pratique :

- Un même usager ne doit pouvoir réinitialiser son mot de passe qu'une fois par jour ;
- Une même IP ne doit pouvoir réaliser que X tentatives d'authentifications successives en 1h.

Éviter les blocages automatiques des comptes, en particulier si l'usager n'est pas autonome pour le déverrouiller.

3.3 Limitation et qualité des e-mails

Pour tous les formulaires de réinitialisation exploitant les e-mails :

- Envoyer des e-mails de qualité qui ne finissent pas dans les spams (DKIM / DMARC / SPF) ;
- Limiter le nombre d'envoi d'e-mails par utilisateur et par jour ;
- Ne jamais envoyer un mot de passe en clair (l'initial ou un auto-généré) par e-mail ;
- Privilégier les liens à usage unique et limités dans le temps pour réinitialiser un mot de passe (bien vérifier que le *token* est associé au bon utilisateur, et que ce *token* est généré avec un PRNG robuste).

3.4 En-têtes, cookies et transit

Généraliser l'utilisation de SSL/TLS avec HTTPS, surtout avec la facilité actuelle (LetsEncrypt). Une fois une application accessible en HTTPS, forcer l'utilisation de ce canal (redirection automatique HTTP vers HTTPS), puis activer les sécurités complémentaires :

- **HSTS** : *HTTP Strict Transport Security* (pour forcer le chargement de toute ressource tierce via HTTPS) ;
- **Flag Secure** sur les cookies en particulier le cookie de session (pour que les cookies critiques ne transitent jamais en clair sur le réseau) ;
- **Flag HttpOnly** sur les cookies en particulier le cookie de session (pour prévenir le vol de cookie par XSS) ;
- **Flag SameSite** sur les cookies en particulier le cookie de session (pour bloquer toutes les CSRF) ;
- **X-Frame-Options** : pour contrer les attaques par *clickjacking*.

Tous les autres en-têtes de sécurité sont bien évidemment conseillés, tels que X-XSS-Protection, CSP, X-Content-Type, mais sortent du présent cadre de l'article.

3.5 Messages génériques et politique

Diffuser le moins d'informations publiques possible à propos de la politique de mot de passe en place, en particulier si celle-ci est jugée faible.

De plus, afin de contrer les énumérations, harmoniser les messages retours et messages d'erreur sur les pages d'authentification et formulaires de réinitialisation de mot de passe avec un message générique et unique :

- Page d'authentification :
  - Éviter les différenciations « *Identifiant inconnu* » ou « *Mot de passe incorrect* » ;
  - Privilégier le message unique : « *Échec de connexion* » ;
- Page de réinitialisation de mot de passe :
  - Éviter les différenciations « *Adresse e-mail inconnue* » ou « *Un mail a été envoyé* » ;
  - Privilégier le message unique : « *Un e-mail a été envoyé à l'adresse indiquée si elle est associée à un compte* ».

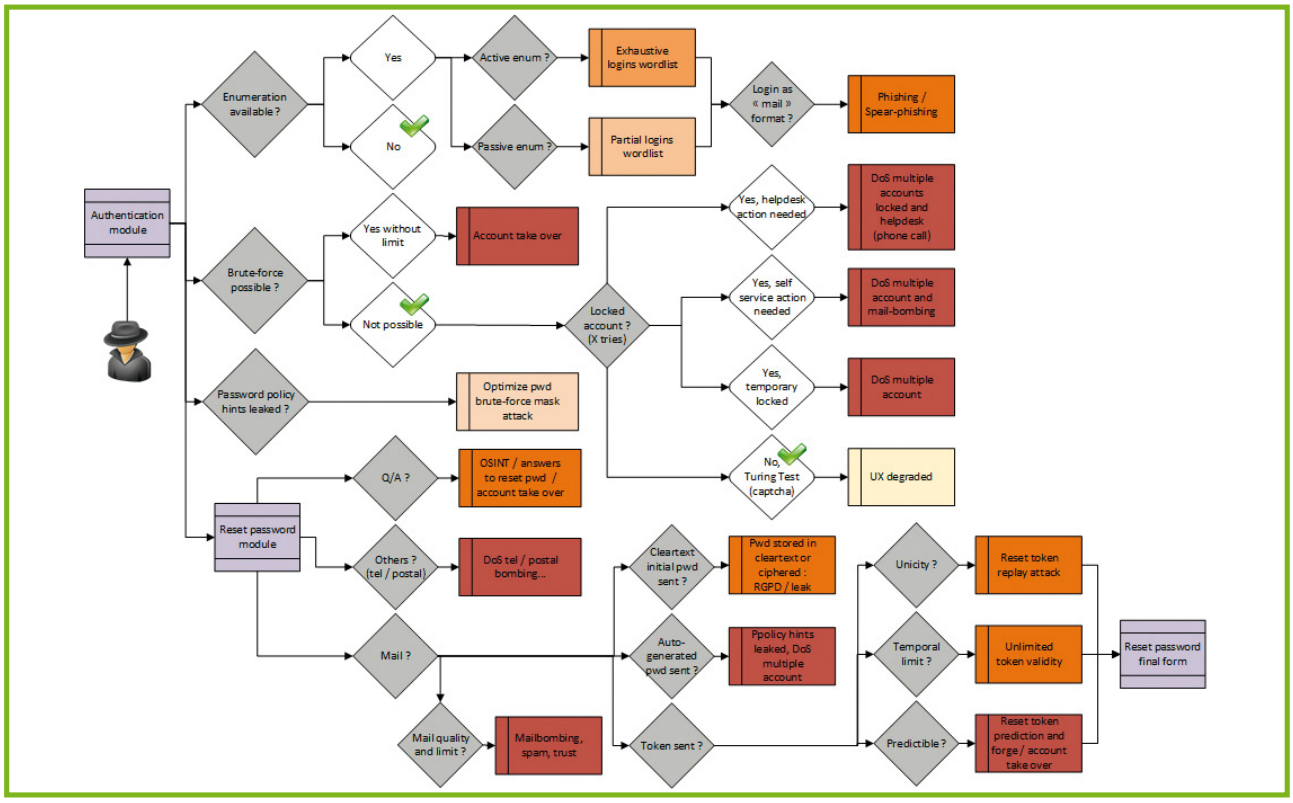


Fig. 3 : Arbre des faiblesses en fonction des approches fonctionnelles.

3.6 Identifiants et mots de passe

Privilégier des identifiants uniques non-incrémentals (pour contrer les énumérations actives). Pour les logins au format « e-mail », appliquer correctement la RFC5233 en autorisant le caractère **+**, alias natif de n'importe quelle boîte-mail, permettant d'avoir des logins uniques (en plus de mot de passe unique évidemment) ; tout en pointant vers la même boîte-mail :

- *yann.cam***+linkedin***@domain.com* (pointant intrinsèquement vers *yann.cam@domain.com*) ;
- *yann.cam***+facebook***@domain.com* (pointant également vers *yann.cam@domain.com*).

Cette technique permet d'avoir des couples (*login*/*password*) uniques pour chaque service, tout en ayant une seule adresse e-mail de centralisation. Ainsi, dans les *leaks* massifs médiatisés (HaveIBeenPwned **[07]**), vos différents comptes seront difficiles à recouper.

Concernant le choix des mots de passe, je ne détaillerais pas le sujet ici, mais en bref : utiliser un Password Manager **[08]**, générez des mots de passe robuste, aléatoires, longs et toujours uniques.

Le service HaveIBeenPwned a mis en place une API permettant de vérifier si un compte ou un mot de passe a déjà fuité dans un *leak*. Automatiser la vérification du *leak* d'un mot de passe lors de son changement au travers de ces API est d'intérêt, tout comme confronter le mot de passe à des listes noires connues.

3.7 Approche fonctionnelle robuste

Adopter une approche de « *security by design* » pour les scénarios fonctionnels. La figure 3 représente les cheminements menant à des faiblesses à éviter.

Conclusion et remerciements

Concevoir un module d'authentification/réinitialisation de mot de passe n'est pas chose aisée pour le rendre sécurisé. Avoir conscience des attaques possibles et des contres-mesures associées est indispensable pour tous les développements web actuels, l'OWASP est un très bon guide à ce sujet **[09] [10]**.

Je tenais à saluer les diverses plateformes de Bug Bounty et les programmes publics et privés associés, qui m'ont fait confiance pour éprouver leurs systèmes, et qui m'ont ainsi permis de synthétiser plusieurs observations au travers de cet article.

Salutations à toute l'équipe de SYNETIS et à la team Leffe au passage, pour leurs échanges, relectures et motivations ! ■

Retrouvez toutes les références de cet article sur le blog de MISC : https://www.miscmag.com/ .

« WEBAUTHN » : ENFIN LA FIN DES MOTS DE PASSE ?

Clément NOTIN – clement@notin.org – @cnotin
Pentester & auditeur

mots-clés : *WEB / IDENTITÉ / AUTHENTIFICATION / MOTS DE PASSE / PASSWORDLESS / WEBAUTHN*

« WebAuthn » pour « Web Authentication » est le nouveau standard du W3C, dont l'objectif est de remplacer U2F pour enfin permettre de s'authentifier sans mot de passe. Comment fonctionne-t-il ? Et va-t-il réussir à s'imposer ?

Le mot de passe est à la fois le mécanisme d'authentification le plus populaire et un échec en termes de sécurité, si bien que des tentatives d'amélioration ont vu le jour, telles des pansements sur une jambe de bois : captcha et verrouillage de compte pour tenter de contrer les attaques par force brute, gestionnaire de mots de passe pour rendre possible la recommandation d'avoir un mot de passe différent par site ou usage, imposition de règles de complexité et de renouvellement pour tenter d'interdire les mots de passe faibles, introduction d'un second facteur (2FA) pour limiter la casse en cas de découverte ou de vol par phishing, passage aux phrases de passe pour faciliter la saisie et augmenter la complexité. Mais le cœur du problème demeure... Et la multiplication des fuites de bases de mots de passe ne fait que le confirmer. Et si nous faisions, enfin, sans ? Ou plutôt : autrement !

Au risque de se répéter, la théorie mentionne trois familles majeures de facteurs d'authentification : ce que l'on sait (mot de passe, code PIN...), ce que l'on détient (token physique, smartphone...) et ce que l'on est (biométrie : empreintes, rétine...).

Nous observons aujourd'hui une transition de « ce que l'on sait » vers « ce que l'on détient », éventuellement complété par « ce que l'on est ».

Après tout, selon le modèle de menace d'un utilisateur lambda, la probabilité est plus forte de se faire pirater, car son mot de passe est *Toto123* et le même de partout, que parce qu'on lui a volé son portefeuille ou son trousseau de clés.

1 Solution WebAuthn

1.1 Présentation

« WebAuthn » est le diminutif de *Web Authentication*, il s'agit d'une API en cours de standardisation par le **[W3C]** (*World Wide Web Consortium*). Les entreprises majeures du Web contribuent à l'effort (Google, Paypal, Mozilla, Microsoft, Yubico, Qualcomm...) ainsi que l'alliance FIDO (à laquelle nous devons le standard U2F, voir ci-après) avec ses 250 organisations membres. La spécification a atteint le niveau de *Candidate Recommendation* (ce qui est presque l'étape finale).

Le principe est de permettre aux utilisateurs de s'authentifier sur des applications web à l'aide d'un périphérique externe : en simple ou en multifacteur, et potentiellement même sans mot de passe.

En coulisse, plus de mots de passe, mais du chiffrement asymétrique avec paires de clés : privés et publiques.

Attention !

Attention, cet article n'a pour objectif que de donner un aperçu de WebAuthn. Si vous l'implémentez, nous vous recommandons fortement de lire au moins la description sur [MDN] et la spécification [W3C].

1.2 Différence avec FIDO U2F

WebAuthn est une évolution de l'U2F de l'alliance FIDO et en reprend certains concepts, notamment celui de l'*authenticator* que nous traduirons par « authentificateur ». C'est un périphérique externe spécialisé dans la gestion des clés d'authentification. Les authentificateurs U2F les plus populaires sont les clés de sécurité YubiKey de la société Yubico qui fonctionnent en USB et NFC.

Le projet **[FIDO2]** a donné naissance à WebAuthn pour l'API côté application, et à *Client-to-Authenticator Protocol 2* **[CTAP2]** qui définit le protocole de communication avec les authentificateurs.

U2F avait quelques limitations. Tout d'abord, comme le nom *Universal 2nd Factor* l'indique, ce standard n'est pas utilisable en tant que premier et unique facteur d'authentification. De plus, il a peiné à sortir de la sphère technophile, en partie, car il n'était pas supporté par tous les sites et navigateurs, ni par les smartphones dépourvus de port USB compatible (et sans NFC à l'époque).

WebAuthn est porté par un consortium plus large et les navigateurs majeurs le supportent déjà (Firefox 60, Chrome 67) ou annoncent le support futur (Edge 18). Apple n'a pour le moment (fin mai 2018) fait aucune annonce quant au support dans son navigateur Safari et dans OS X et iOS.

La promesse pour l'utilisateur est aussi plus alléchante : fini les mots de passe !

Début mai 2018, Dropbox a annoncé la compatibilité avec WebAuthn. Le site de partage de fichiers faisait déjà partie des premiers à supporter U2F en 2015. Le support est toutefois limité à l'usage en tant que second facteur d'authentification, et non en remplacement du mot de passe pour des raisons « de sécurité et d'utilisabilité » **[DROPBOX]**.

1.3 Authentificateurs

Les authentificateurs peuvent prendre plusieurs formes selon les niveaux de sécurité et de facilité d'utilisation souhaités. Nous pouvons retrouver des clés de sécurité physiques, comme les YubiKey, que nous pourrons connecter via USB, NFC ou Bluetooth Low Energy (le sans-fil étant plus adapté aux smartphones). Les smartphones, via leurs composants de sécurité (ex. : *secure element* d'Apple), et les TPM et autres environnements d'exécution de confiance (*trusted execution environment*) des ordinateurs peuvent aussi être exploités. Les montres connectées pourraient aussi remplir ce rôle. Enfin, des implémentations purement logicielles sont aussi possibles, par exemple simplement au niveau du système d'exploitation sans composant matériel dédié.

Le rôle de l'authentificateur est de générer, stocker et gérer des paires (privé & publique) de clés d'authentification. En particulier, la clé privée qui ne doit pas en sortir. Il a aussi le rôle de recueillir le consentement de l'utilisateur par un geste d'autorisation (*authorization gesture*) : simple appui sur un bouton, saisie d'un PIN, biométrie...

Il est à noter que WebAuthn est rétrocompatible, de manière dégradée, avec les authentificateurs U2F existants. Les clés existantes pourront servir de second facteur via WebAuthn, et les sites pourront passer de U2F à WebAuthn sans avoir à reconfirmer tous les authentificateurs existants. Il faudra néanmoins changer de matériel pour profiter des nouveautés comme l'authentification sans mot de passe.

La spécification permet d'avoir plusieurs authentificateurs, et autant de clés privées, associés à son compte. Le serveur doit le prendre en compte et ainsi permettre de stocker plusieurs clés publiques. Un identifiant est associé à chacune, stocké dans l'authentificateur et côté serveur, pour que ce dernier identifie facilement quel authentificateur a été utilisé lors de l'authentification.

L'authentificateur peut avoir un écran pour afficher à l'utilisateur les informations du site auquel il est en train de s'inscrire ou se connecter. Mais cela n'est pas requis, c'est alors au navigateur d'afficher ces informations.

1.4 Séquence d'authentification haut-niveau

Attention, cette description haut-niveau omet volontairement les subtilités, les cas particuliers et certaines des données échangées et validations requises.

La norme utilise le terme de *relying party*. Il est important de comprendre qu'il désigne l'application web ou le service auquel nous cherchons à nous authentifier. Dans la suite, nous le désignerons simplement par « serveur ».

WebAuth utilise une authentification dite de « challenge-réponse ». Le client prouve sa connaissance d'un secret, sans jamais l'envoyer au serveur, mais en fournissant une réponse valide que seul celui qui détient le secret peut créer. Ceci via une signature cryptographique, ce qui est très similaire à une authentification par certificat. Le challenge est toujours généré par le serveur et n'est utilisé que pour une opération (inscription ou authentification). Il doit aussi être suffisamment long (ex. 16, 32 ou 100 octets selon les sources...) et d'un bon niveau d'entropie.

1.4.1 Inscription

L'utilisateur demande à s'inscrire dans son application cliente (JavaScript...). Le serveur génère un challenge aléatoire retourné à l'application cliente. Le navigateur ajoute les informations sur le domaine et envoie le tout à l'authentificateur. Ce dernier recueille le consentement de l'utilisateur (appui sur bouton, PIN, biométrie...) puis génère une nouvelle paire de clés. Il stocke la privée, signe le challenge et le renvoie avec la clé publique au navigateur qui les transmet à l'application cliente. Elle les envoie au serveur, qui vérifie la réponse au challenge et enregistre la clé publique avec les informations de l'utilisateur. Cette clé devient le pendant de l'empreinte du mot de passe qu'utilise une application classique aujourd'hui.

Et visuellement, dans Firefox 60, nous obtenons :

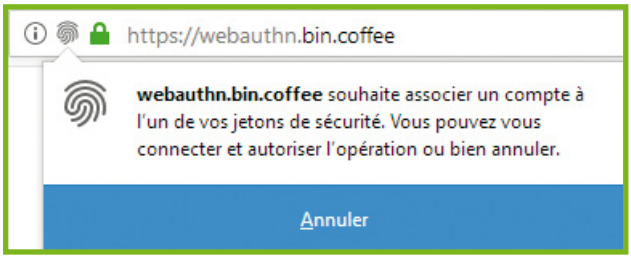


Figure 1

1.4.2 Authentification

L'utilisateur demande à s'authentifier dans son application cliente (JavaScript). Le serveur génère un challenge aléatoire retourné à l'application cliente. Le navigateur ajoute les informations sur le domaine et envoie le tout à l'authentificateur. Ce dernier recueille

le consentement de l'utilisateur (appui sur bouton, PIN, biométrie...) puis vérifie que le domaine est bien connu. L'authentificateur signe le challenge avec la clé privée qu'il contient et renvoie cette réponse au navigateur qui transmet à l'application cliente. Elle l'envoie au serveur, qui récupère dans sa base la clé publique de l'utilisateur et vérifie avec la réponse au challenge. Le serveur retourne en échange un jeton de session (cookie, JWT...).

C'est ainsi que l'utilisateur prouve son identité : par la détention de la clé privée associée à la clé publique que le serveur connaît.

Et visuellement, dans Firefox 60, nous obtenons :

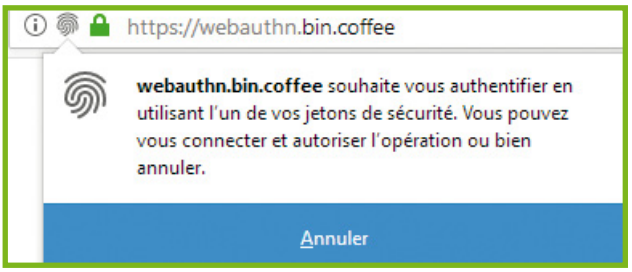


Figure 2

1.5 API

WebAuthn ne spécifie pas les échanges entre l'application et le serveur : il peut s'agir de JSON envoyé avec **XMLHttpRequest** ou l'API plus récente **Fetch** **[FETCH]**, de formulaires **POST** classiques, d'un envoi au format **SOAP**, etc.

La seule obligation est d'utiliser HTTPS afin d'être dans un *Secure Context* **[SECURECONTEXT]** comme pour de nombreuses autres API modernes et puissantes.

1.5.1 Credential Management API

L'API *Credential Management* **[CREDENTIAL MANAGEMENT]** du W3C permet à un site web de stocker et de récupérer des identifiants pour l'utilisateur, qu'il s'agisse de classiques noms d'utilisateur et mots de passe, ou d'identifiants fédérés (ex. *OpenID Connect*). JavaScript peut interagir avec cette API par l'intermédiaire de **navigator.credentials**.

WebAuthn étend cette API pour ajouter le type d'identifiant **PublicKeyCredential**.

1.5.2 Inscription

L'inscription utilise la méthode **navigator.credentials.create()** avec le type d'identifiant **publicKey**. Voici un exemple court de code JavaScript partagé par Google **[GOOGLE]** :

```
let credential = await navigator.credentials.create({ publicKey: {
  // challenge aléatoire fourni par le serveur
  challenge: Uint8Array(32) [117, 61, 252, 231, 191, 241,...]
  // informations d'identification du serveur " Relying-Party "
  rp: { id: "acme.com", name: "ACME Corporation" },
  user: {
    // identifiant technique de l'utilisateur, fourni par le serveur
    id: Uint8Array(8) [79, 252, 83, 72, 214, 7, 89, 26]
    // informations visuelles de l'utilisateur : peuvent être
affichées par l'authentificateur
    name: "jamiedoe",
    displayName: "Jamie Doe"
  },
  // exigences techniques sur l'identifiant à créer, fournies par le
serveur
  pubKeyCredParams: [ {type: "public-key", alg: -7} ]
}});
```

L'algorithme indiqué par « -7 » correspond à « ES256 », c'est-à-dire l'algorithme ECDSA basé sur la courbe elliptique P-256 et l'algorithme de hachage SHA256. Le nombre « -7 » est défini dans le registre des algorithmes COSE (*CBOR Object Signing and Encryption*) maintenu par l'**[IANA]**.

1.5.3 Authentification

L'authentification utilise la méthode **navigator.credentials.get()** avec le type d'identifiant **publicKey**. Voici un exemple court de code JavaScript partagé par Google **[GOOGLE]** :

```
let credential = await navigator.credentials.get({ publicKey: {
  // challenge aléatoire fourni par le serveur, différent de celui
de l'inscription et à chaque authentification
  challenge: Uint8Array(32) [139, 66, 181, 87, 7, 203,...]
  // identifiant technique du serveur " Relying-Party "
  rpId: "acme.com",
  allowCredentials: [{
    type: "public-key",
    // identifiant technique de la clé précédemment enregistrée et
stockée dans l'authentificateur actuellement utilisé
    id: Uint8Array(80) [64, 66, 25, 78, 168, 226, 174,...]
  }],
  // le serveur requiert que l'utilisateur consente explicitement à
cette opération
  userVerification: "required",
}});
```

1.5.4 Manipulations des clés

L'API ne propose pas de méthodes permettant de gérer les clés stockées, et encore moins leur partie privée. Elle ne permet que d'en créer.

1.5.5 Sites de test

Vous pouvez déjà tester la technologie sur plusieurs sites de test, qui partagent leur code source :

- Mozilla (pas de backend) : https://webauthn.bin.coffee/
- Google (backend en Java) : https://webauthndemo.appspot.com/
- Duo Security (backend en Go) : https://webauthn.io/
- Alliance FIDO (backend en Node.js) : https://webauthn.org/

www.hsc-formation.fr

HSC by Deloitte.

1.6 Propriétés de sécurité

Il est à noter que ces propriétés de sécurité étaient déjà présentes dans U2F, mais il est intéressant de les rappeler.

1.6.1 Détection de faux domaine (anti-phishing)

Le nom de domaine du site sur lequel nous tentons de nous connecter est extrait par le navigateur et transmis à l'authentificateur comme décrit dans le protocole **[CTAP2]**. Si un pirate crée un site de phishing avec un nom de domaine similaire au site ciblé (ex. *mabanque.fr.pirate.com* ou *mabamque.fr*), l'authentificateur ne parviendra pas à trouver la clé associée à ce faux site et refusera donc l'authentification.

Cette propriété permet de pallier les faiblesses de l'humain.

1.6.2 Anti-relai (anti-phishing)

Avec l'avènement des jetons OTP utilisés comme second facteur, les pirates ont adapté leurs techniques d'ingénierie sociale et pensent désormais à collecter ces codes auprès des victimes. Ils peuvent le faire de manière astucieuse en les demandant simplement à leur victime (oralement, ou dans la page de phishing) ou via une application malveillante qui intercepte leur envoi par SMS. Dans le cas d'une attaque ciblée, ils peuvent aussi appeler leur victime et utiliser un prétexte pour demander le code affiché sur son jeton. Nous observons alors que ces seconds facteurs ne sont plus qu'un frein, mais pas un blocage.

WebAuthn se base sur des nombres qui sont beaucoup plus grands et complexes (autant les clés, que les challenges et leurs réponses). Il apparaît dès lors difficile de demander à une victime de les saisir ou de les fournir manuellement au pirate d'une manière ou d'une autre.

1.6.3 Canal de communication protégé

WebAuthn requiert l'utilisation d'un canal de communication protégé en confidentialité et intégrité tel que HTTPS (SSL/TLS).

Ce protocole n'est pas parfait, mais il a un niveau de sécurité bien plus élevé que ce que nous pouvons obtenir par les canaux de téléphonie mobile classiques. Il a été démontré que certains criminels s'en prennent aux protocoles **[SS7]** des opérateurs afin de rediriger les communications de leurs cibles et finalement de dérober des codes de confirmation envoyés par SMS.

1.6.4 Anti-rejeu

Comme nous venons de le voir, WebAuthn utilise forcément HTTPS. Il peut arriver cependant qu'un attaquant parvienne à capturer un échange d'authentification. Ceci peut arriver dans plusieurs cas rares tels que l'obtention d'une position active d'interception, dite de *Man-in-the-Middle*, avec un faux certificat, ou exploitation laborieuse d'une vulnérabilité SSL/TLS, ou logiciel malveillant d'interception installé sur le poste...

Cette exploitation est dérangeante sur le coup, mais il faut noter que la clé privée de l'utilisateur n'est pas compromise. Et vu que le challenge est aléatoire et différent à chaque opération, l'attaquant ne pourra donc pas réutiliser les informations obtenues.

L'attaque n'est donc pas persistante.

1.6.5 Anti-bruteforce

Un mot de passe peut s'attaquer par force brute. Soit de manière verticale, en testant plusieurs mots de passe probables pour un compte utilisateur donné, soit de manière horizontale, en testant une poignée de mots de passe triviaux, mais sur un grand nombre de comptes (technique dite de *password spraying*).

L'utilisation de la cryptographie asymétrique avec des clés longues permet un niveau d'entropie des secrets impossible à manipuler de tête pour un humain. Il sera donc quasiment impossible pour un attaquant de deviner la clé secrète. C'est pour la même raison que les authentifications SSH par clé sont préférées à celles par mot de passe.

Ceci permet aussi de se passer des bricolages destinés à pallier les faiblesses des mots de passe, comme les captchas et les verrouillages de comptes.

1.6.6 Authentification silencieuse et pistage

WebAuthn permet à des sites web d'enregistrer une information persistante sur leur authentificateur. Ceci serait immédiatement détourné à des fins de pistage au regard de la créativité des sociétés spécialisées dans le domaine, et qui sont à la recherche permanente d'alternatives aux cookies.

Ce risque a été pris en compte dans la conception. Notez au passage que WebAuthn est donc conforme à la règle de *privacy by design* au cœur du RGPD !

Ainsi, les authentificateurs requièrent systématiquement un consentement explicite de l'utilisateur. Que ce soit au niveau de l'authentificateur, ou de l'interface du navigateur. Ce consentement s'applique aussi lorsqu'une application tente de collecter les informations techniques de l'authentificateur connecté (ce qui pourrait faciliter le *fingerprinting*).

Une application malveillante ne pourra pas en théorie créer des identifiants, ou provoquer une authentification, de manière silencieuse. C'est aussi la raison pour laquelle l'authentificateur ou le navigateur peuvent afficher le domaine qui a déclenché l'opération.

1.6.7 Protection des clés privées

Toute la sécurité de l'identité de l'utilisateur repose sur ses clés privées présentes dans l'authentificateur. Elles sont donc stockées dans des composants électroniques aux propriétés de sécurité renforcées, tels que des cartes à puces ou des TPM. Ces composants ont aussi le rôle d'effectuer les vérifications biométriques, ou de PIN, le cas échéant. Ils ont aussi la charge de générer des clés privées qui doivent avoir un bon niveau d'entropie (et donc très difficilement devinables). Ces exigences de sécurité sont contrôlées par l'alliance FIDO via son processus de certification **[FIDO2-CERTIF]**.

L'utilisation d'un composant externe, renforcé, et à la surface d'attaque réduite, rend plus difficiles les attaques pour dérober les clés privées et donc usurper la victime.

Aujourd'hui, une application mobile malveillante, disposant des droits nécessaires, peut aisément intercepter les codes OTP envoyés par SMS. Mais il est plus difficile d'aller dérober les secrets d'une autre application, grâce aux mécanismes de cloisonnement du système d'exploitation mobile (tant qu'il n'a pas été débridé par une opération de *jailbreak/root*).

1.6.8 Révocation d'un authentificateur

Un authentificateur peut être révoqué pour plusieurs raisons et par chacune des parties. L'utilisateur peut déclarer une perte, un vol ou un changement de matériel. Il accède à une page du site et sélectionne dans une liste l'authentificateur concerné. Le serveur peut aussi choisir de retirer régulièrement ceux qui ne sont plus utilisés.

La spécification parle de « décommissionner », mais ne donne pas plus d'indications, car il s'agit d'une opération purement côté serveur. L'authentificateur n'est pas informé qu'il a été révoqué, même si nous y avons toujours accès. Les tentatives d'authentification avec l'authentificateur révoqué seront ensuite refusées.

À l'opposé, il est aussi possible de retirer les identifiants de l'authentificateur. Le moyen à utiliser n'est pas non plus spécifié. Il peut s'agir d'une interface sur l'authentificateur, ou d'un logiciel de gestion dédié, etc.

1.7 Propriétés de sécurité

WebAuthn évoque d'autres cas d'usage étendus qui vont plus loin que la simple inscription et connexion à un site (liste non exhaustive).

Un utilisateur peut accéder à un site depuis son ordinateur portable et être guidé pour la création et le stockage d'identifiants sur son smartphone.

Le serveur peut demander à l'utilisateur de confirmer une action sensible (ex. paiement) par son *authorization gesture* qui est le geste effectué habituellement auprès de l'authentificateur pour s'enregistrer et se connecter.

En effet, la spécification est générique et le consentement que l'utilisateur donne pour s'authentifier, peut aussi s'appliquer à d'autres opérations.

Ceci peut permettre d'implémenter de façon standard ce que certaines banques commencent à proposer avec une application mobile dédiée.

La spécification distingue les *platform credentials* qui sont issus d'authentificateurs attachés à l'appareil en cours d'utilisation (ex. TPM d'un ordinateur portable), des *roaming credentials* qui sont détachables et utilisables sur plusieurs machines (ex. clé USB, Bluetooth ou NFC qui peuvent passer d'un ordinateur portable à un smartphone).

Le serveur peut par exemple requérir une authentification initiale par un *roaming credential* d'un authentificateur précédemment associé au compte depuis un autre ordinateur, stocker un *platform credential* sur l'ordinateur actuellement utilisé, puis autoriser l'utilisateur à se reconnecter directement avec son *platform credential* sans avoir à sortir son authentificateur externe de sa poche.

Cela est laissé libre selon le compromis souhaité entre la sécurité et l'utilisabilité.

2 Interaction avec les systèmes existants

Microsoft travaille et communique beaucoup sur le sujet de l'authentification sans mot de passe, comme l'illustre l'article de blog « construire un monde sans mots de passe » **[MS-PASSWORDLESS]**. Ils ont remarqué que les mots de passe avaient quelques avantages comme la compatibilité, la portabilité et la facilité de distribution, mais qu'en contrepartie, ils étaient peu sécurisés ni pratiques et coûteux au final. Ils proposent une stratégie en quatre étapes pour s'en débarrasser :

1. Développer des solutions de remplacement.
2. Réduire l'utilisation des mots de passe dans tout le cycle de vie d'une identité (création, association à un appareil, connexion à des applications et sites, récupération, etc.).
3. Tester et déployer progressivement ces solutions.
4. Éliminer complètement les mots de passe du référentiel d'identité.

Nous aurions aimé pouvoir rendre compte d'alternatives chez les autres éditeurs majeurs, y compris dans le monde du logiciel libre, mais nous n'avons pas trouvé de ressources équivalentes.

2.1 Windows Hello

Windows Hello **[MS-HELLO]** est un nouveau système d'authentification de Microsoft développé pour Windows 10 et qui permet de s'authentifier avec la biométrie, comme la reconnaissance faciale ou d'iris et les empreintes digitales. Windows est évidemment le premier à s'en

servir, mais l'idée est que les applications, notamment web, puissent se baser sur cette fonctionnalité du système d'exploitation pour leur authentification. Un code PIN est aussi créé comme moyen de secours. Un peu à la manière de Touch ID d'Apple, les données biométriques restent stockées dans l'appareil et ne sont pas envoyées dans le cloud.

Ce système permet aussi d'étendre l'authentification à d'autres mécanismes récents. Microsoft s'est donc associé aux efforts de FIDO2 et WebAuthn et a annoncé **[MS-HELLO-WEBAUTHN]** en avril 2018 que les clés FIDO2 permettaient désormais de s'authentifier, sans nom d'utilisateur ni mot de passe, sur les ordinateurs Windows 10 joints à Azure AD via Windows Hello. Trois constructeurs de clés compatibles FIDO2 ont été mis en avant (Yubico, HID et Feitian).

Microsoft a aussi annoncé **[MS-EDGE-WEBAUTHN]** le 31 mai 2018 que sa dernière version *Insider Preview Build 17682* de Windows 10 activait le support de WebAuthn dans son navigateur Edge. L'utilisateur peut choisir un authentificateur externe, ou Windows Hello en tant qu'authentificateur.

2.2 Microsoft Authenticator

Il s'agit d'une application mobile disponible sur Android et iOS. Au début, elle servait uniquement de second facteur OTP, comme Google Authenticator, mais ses fonctionnalités ont été étendues pour la connexion aux services Microsoft. Elle permettait déjà de s'authentifier sur Azure AD et Office 365 sans mot de passe.

Il suffit de saisir son nom d'utilisateur puis une notification s'affiche sur le smartphone associé au compte. L'utilisateur n'a qu'à confirmer l'opération sur Microsoft Authenticator pour être connecté, sans avoir à saisir son mot de passe **[MS-AUTHENTICATOR]**.

La même procédure est utilisable pour se connecter sur un ordinateur Windows 10 S avec un compte Azure AD.

Le protocole interne utilisé par cette fonctionnalité, jusqu'alors propriétaire, va pouvoir être remplacé par WebAuthn et donc devenir plus interopérable.

Conclusion

WebAuthn est une API prometteuse, qui se base sur l'existant U2F en l'améliorant. Ce dernier a pêché par une faible diffusion et WebAuthn semble mieux préparé pour rencontrer un plus large succès. Essentiellement grâce à deux facteurs : une compatibilité avec plus de navigateurs et systèmes d'exploitation, et à un bénéfice clair et immédiat pour l'utilisateur final qui pourra enfin se débarrasser de ses mots de passe.

Nous remarquons néanmoins que l'API est légèrement complexe, autant côté client que serveur. En tout cas, l'ensemble est plus complexe que d'envoyer simplement un mot de passe et le comparer avec son empreinte (ou pire en clair) stockée en base. Espérons donc que WebAuthn soit rapidement pris en charge par les frameworks de développement client et serveur, ainsi que par les fournisseurs tiers de service d'authentification (comme Auth0, Duo Security, etc.). Charge ensuite aux fameux mastodontes du Web d'informer leurs utilisateurs.

Enfin, les puristes regretteront que cette solution ne soit pas parfaite. Oui, un authentificateur physique peut être volé, oui une faille logicielle pourrait mener au vol de la clé privée, oui un utilisateur peut être amené à dévoiler son PIN ou à prêter son doigt à un tiers (y compris de manière forcée), oui les systèmes de biométrie peuvent être trompés... Mais le gain en sécurité espéré est important néanmoins, surtout pour la majorité de la population et des cas d'utilisation. ■

Remerciements

Merci à Émilien de m'avoir proposé d'écrire cet article et à Arthur pour la relecture.

Références

[W3C] https://www.w3.org/TR/webauthn/

[MDN] https://developer.mozilla.org/en-US/docs/Web/API/Web_Authentication_API

[FIDO2] https://fidoalliance.org/fido2/

[CTAP2] https://fidoalliance.org/specs/fido-v2.0-id-20180227/fido-client-to-authenticator-protocol-v2.0-id-20180227.html

[DROPBOX] https://blogs.dropbox.com/tech/2018/05/introducing-webauthn-support-for-secure-dropbox-sign-in/

[FETCH] https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API

[SECURECONTEXT] https://developer.mozilla.org/en-US/docs/Web/Security/Secure_Contexts

[CREDENTIALMANAGEMENT] https://developers.google.com/web/fundamentals/security/credential-management/

[GOOGLE] https://developers.google.com/web/updates/2018/05/webauthn

[IANA] https://www.iana.org/assignments/cose/cose.xhtml#algorithms

[SS7] https://www.silicon.fr/lancienne-faille-dans-ss7-braque-des-comptes-bancaires-par-sms-174033.html

[FIDO2-CERTIF] https://fidoalliance.org/certification/fido-certified-products/

[MS-PASSWORDLESS] https://cloudblogs.microsoft.com/microsoftsecure/2018/05/01/building-a-world-without-passwords/

[MS-HELLO] https://www.microsoft.com/fr-fr/windows/windows-hello

[MS-HELLO-WEBAUTHN] https://blogs.windows.com/business/2018/04/17/windows-hello-fido2-security-keys/

[MS-EDGE-WEBAUTHN] https://blogs.windows.com/windowsexperience/2018/05/31/announcing-windows-10-insider-preview-build-17682/

[MS-AUTHENTICATOR] https://cloudblogs.microsoft.com/enterprisemobility/2017/04/18/no-password-phone-sign-in-for-microsoft-accounts/

HS2
Hervé Schauer Sécurité

Formation cybersécurité organisationnelle

PROGRAMME

Gouvernance de la sécurité

RSSI :
Formation RSSI
CISSP :
Préparation au CISSP
CISA :
Préparation au CISA
SECUHOMOL :
Homologation de la SSI
SECUCRISE :
Gestion de crise IT/SSI
EBIOS2010 :
EBIOS 2010 Risk Manager

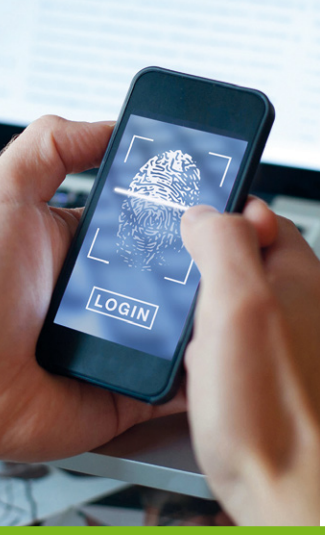
Gouvernance de la sécurité avec les normes ISO270XX

ESS27 :
Essentiels ISO27001 & ISO27002
ISO27LA :
ISO27001 Lead Auditor
ISO27LI :
ISO27001 Lead Implementer
ISO27RM :
ISO27005 Risk Manager
ISO27004 :
ISO27004 / Indicateurs et tableaux de bord cybersécurité
ISO27035 :
ISO27035 / Gestion des incidents de sécurité

+33 644 014 072

www.hs2.fr

formations@hs2.fr



OPENID CONNECT : PRÉSENTATION DU PROTOCOLE ET ÉTUDE DE L'ATTAQUE BROKEN END-USER AUTHENTICATION

Rémi CASSAM CHENAÏ – remi.cassam@protonmail.com
Officier de la Marine nationale - Diplômé ESSI
Olivier LEVILLAIN – olivier.levillain@ssi.gouv.fr
ANSSI - Responsable du centre de formation

mots-clés : OPENID CONNECT / FÉDÉRATION D'IDENTITÉ

L'emploi quotidien de nombreux services sur le Web rend l'utilisation de méthodes d'authentification unifiées très utile. La fédération d'identité avec OpenID Connect est une manière de mettre en œuvre cette authentification unique. Cependant, ce jeu à trois acteurs (utilisateur, fournisseur d'identité, fournisseur de service) ne fonctionne que si tout le monde a la même vision de la situation !

1 La fédération d'identité dans le monde Web

Les applications web sont aujourd'hui devenues omniprésentes dans nos vies. Que ce soit dans la sphère personnelle ou dans un contexte professionnel, nous accédons à de nombreuses applications chaque jour, depuis nos ordinateurs, nos tablettes, nos téléphones ou encore nos télévisions connectées.

1.1 Authentification classique par mot de passe

L'une des problématiques soulevées par la multitude des services disponibles est certainement la multiplication des comptes utilisateurs. Pour utiliser ces applications, il est en effet généralement nécessaire de s'authentifier. Or, la solution la plus courante pour justifier de son identité reste encore aujourd'hui l'authentification par mot de passe. Dans ce cas de figure, l'utilisateur s'identifie avec un identifiant (nom, adresse électronique, pseudonyme…) puis prouve son identité en fournissant le mot de passe associé pour se connecter.

Le schéma par mot de passe est simple et facile à déployer. Il présente néanmoins plusieurs inconvénients de taille. Tout d'abord, il faut bien choisir son mot de passe pour qu'il ne soit pas prédictible facilement (rappel : ni le nom de votre chien, ni votre date de naissance, ni *azerty12* ne sont de bons mots de passe).

Cependant, cela n'est pas suffisant, car il arrive que les fournisseurs de service ne protègent pas correctement votre mot de passe et que ce précieux sésame se retrouve dans la nature. Pour éviter qu'une telle compromission affecte l'ensemble de votre activité numérique, il est donc nécessaire de ne pas réutiliser le même secret sur des services différents, ce qui vous amène à devoir définir (et retenir) des dizaines de mots de passe.

Enfin, la nécessité de s'authentifier individuellement pour chacune de ses applications web est fondamentalement pénible. Or, derrière ce problème d'ergonomie, il faut comprendre que l'utilisateur va rapidement se lasser d'oublier ou de se tromper fréquemment pour finalement réutiliser le même mot de passe (ou le même schéma de mots de passe).

Une manière classique de s'en sortir est d'utiliser un coffre-fort de mots de passe : un outil qui génère, conserve et protège ces précieux sésames, et que l'on déverrouille à partir d'un mot de passe maître. Il devient en effet alors possible de gérer l'authentification de ses différentes identités numériques de manière plus ou moins automatique, en n'ayant qu'un seul mot de passe à retenir. Cette méthode atteint tout de même ses limites quand l'utilisateur change fréquemment d'équipements, et que ceux-ci n'acceptent pas de tels outils (la télévision connectée par exemple).

1.2 L'authentification unique

Heureusement, une autre méthode est disponible pour l'authentification des utilisateurs auprès des services auxquels on souhaite se connecter : l'authentification

unique (SSO pour *Single Sign On* en anglais). Dans le monde bureautique classique de l'entreprise, un moyen répandu d'implémenter ce concept est Kerberos. L'utilisateur s'authentifie la première fois auprès d'un serveur central (KDC, *Key Distribution Center*). À chaque fois qu'il souhaite ensuite accéder à un service (le serveur de messagerie, une imprimante, etc.), il retourne voir le KDC pour obtenir un ticket à présenter au fournisseur du service.

Dans le monde web, ce modèle très centralisé ne peut généralement pas fonctionner et n'est d'ailleurs pas bien adapté. C'est pourquoi un autre modèle d'authentification unique est généralement rencontré : la fédération d'identité.

Si ce n'est pas déjà fait, l'utilisateur commence par se créer un compte auprès d'un fournisseur d'identité (Facebook, Google...). Cette étape consiste la plupart du temps à choisir un identifiant et un mot de passe, mais aussi à renseigner d'autres informations personnelles telles que sa date de naissance.

Ensuite, lorsque l'utilisateur souhaite utiliser un service sur Internet (musique en *streaming*, presse en ligne, réseau social professionnel, etc.), il doit s'enregistrer avec un compte. Cependant, au lieu d'en créer un ad hoc, spécifique au service souhaité, il va lui indiquer qu'il souhaite se connecter via son fournisseur d'identité.

L'utilisateur sera alors redirigé vers celui-ci pour s'authentifier. Une fois cette étape réalisée, les informations d'identité (son identifiant au minimum) seront transmises par le fournisseur d'identité au fournisseur de service auquel souhaite accéder l'utilisateur.

Le modèle contient donc trois types d'acteurs pour lesquels il existe généralement plusieurs instances (voir figure 1) : les utilisateurs (humains ou systèmes automatisés), les fournisseurs d'identité (FI) et les fournisseurs de service (FS). Les principaux acteurs du Web favorisent aujourd'hui l'utilisation de ce modèle qui connaît une forte croissance depuis quelques années.

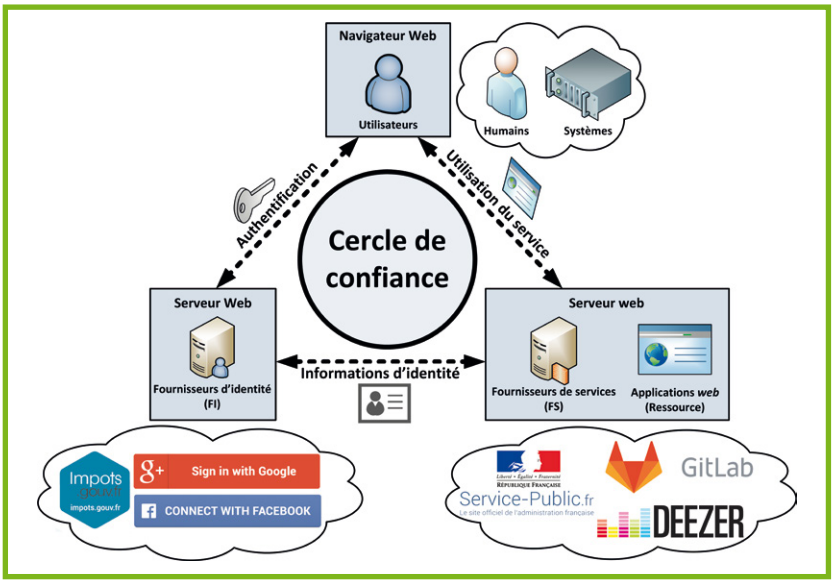


Fig. 1 : Modèle de gestion d'identité fédérée : certains fournisseurs de services peuvent déléguer l'identification et l'authentification de leurs utilisateurs à d'autres fournisseurs d'identité.

L'avantage de ce schéma est que l'utilisateur n'a plus qu'à gérer qu'un seul compte, auprès de son fournisseur d'identité. Il peut donc choisir un mot de passe robuste, et en théorie mieux maîtriser les informations personnelles qu'il accepte de transmettre aux fournisseurs de services. Enfin, les services auxquels l'utilisateur accède n'ont pas accès aux accréditations de ce dernier, ce qui limite a priori les conséquences d'une éventuelle compromission.

Le lecteur attentif remarquera néanmoins que dans ce schéma, la perte du mot de passe devient catastrophique, puisque le fournisseur d'identité se trouve être un acteur critique du dispositif. Cependant, il devient également possible de renforcer la sécurité de l'authentification dans ce cas : choisir un mot de passe très robuste, activer l'authentification à double facteur, utiliser des jetons de sécurité U2F. Enfin, en cas de compromission du compte, il suffit de contacter un opérateur pour reprendre la main sur l'ensemble des services.

2 Standards existants

Pour mettre en place une telle fédération d'identité, il existe essentiellement deux protocoles : SAML et OpenID Connect.

2.1 SAML

SAML (*Security Assertion Markup Language*) est un protocole soutenu par le consortium mondial à but non lucratif OASIS. Il est très populaire dans les fédérations d'identité universitaires, comme celui opéré par RENATER en France. Les spécifications de la version 2.0 **[SAML]** datent de 2005, mais ont fait l'objet d'amendements réguliers depuis.

SAML s'appuie sur le format XML pour transmettre les informations d'identité, aussi appelées assertions. Ces assertions peuvent être chiffrées ou signées à l'aide de XML Encryption et de XML-DSig. Il existe différentes manières pour communiquer les assertions SAML entre les fournisseurs d'identité et de services : en passant par l'utilisateur (*HTTP Redirect Binding* ou *HTTP POST Binding*), ou en utilisant un canal de communication dédié entre le FI et le FS (*HTTP Artifact Binding*, qui utilise le protocole SOAP). En pratique, c'est la première méthode (*HTTP Redirect Binding*), qui repose sur une simple requête GET, qui est la plus employée.

Bien que SAML soit aujourd'hui un standard éprouvé, documenté et encore très utilisé, il souffre de plusieurs limitations (gestion du consentement de l'utilisateur hors spécification, déconnexion unique difficile à gérer, etc.) et est techniquement complexe à mettre en œuvre. De plus, avec l'arrivée de protocoles plus récents, SAML dispose désormais d'une communauté un peu moins active.

2.2 OpenID Connect

OpenID Connect, publié en 2014 **[OIDC]**, est une évolution du standard OpenID et s'appuie sur le protocole OAuth 2.0 **[OAUTH]** en y ajoutant une couche d'authentification.

OpenID Connect repose sur le standard JSON Web Token (JWT) pour échanger des informations d'identité ou toute autre information pertinente pour le FS. De manière similaire aux assertions SAML, ces jetons peuvent être signés et chiffrés.

Il existe plusieurs manières pour le protocole de fonctionner (appelées « cinématiques »). La méthode dite du code d'autorisation (*Authorization Code Mode*) est aujourd'hui la plus répandue et reste généralement très simple à mettre œuvre.

L'avantage de cette cinématique est qu'elle réduit les risques de compromission. En effet, bien que certaines informations passent par l'intermédiaire de l'utilisateur, le jeton d'identité ne transite en revanche pas par celui-ci. Ces informations sensibles sont directement échangées entre le FI et le FS, ce qui permet de restreindre les possibilités d'un utilisateur (ou d'un attaquant ayant compromis celui-ci) qui chercherait à modifier ce jeton. C'est pour cela que cette méthode est recommandée, notamment par l'OWASP, dès qu'une communication entre les fournisseurs (*back channel*) est réalisable..

Bien qu'OpenID Connect soit un standard relativement jeune, il est porté par la fondation OpenID, soutenue par de nombreuses entreprises influentes comme Google, Microsoft ou Oracle.

Si SAML semble adapté à des fédérations d'identité dans des environnements plus cadrés, OpenID Connect a vocation à rendre possible la fédération d'identité dans un environnement très ouvert. Certaines fonctionnalités optionnelles du protocole témoignent en particulier de cette philosophie : la détermination à la volée du FI associé à l'utilisateur ou l'enregistrement dynamique d'un FS.

Nous avons donc choisi de nous intéresser plus en détail à OpenID Connect.

3 Description d'OpenID Connect

Comme indiqué précédemment, il existe plusieurs cinématiques OpenID Connect. Nous décrivons ici uniquement la méthode du code d'autorisation (*Authorization Code Mode*), la plus répandue et la plus recommandée.

3.1 Terminologie du protocole

Avant de présenter le fonctionnement technique du protocole, voici quelques définitions propres à OIDC :

- *OpenID Provider* (OP) : il s'agit du fournisseur d'identité (FI).
- *Relying Party* (RP) : c'est le fournisseur de services (FS).
- *End-User* : ce terme désigne l'utilisateur.

Pour faciliter la compréhension de la suite de l'article, les termes génériques fournisseur d'identité (FI), fournisseur de service (FS) et utilisateur seront préférés aux appellations spécifiques du protocole.

- *ID Token* : jeton d'identité au format *JSON Web Token* (JWT) signé et/ou chiffré permettant au FI de certifier l'authentification de l'utilisateur au FS.
- *Access Token* : jeton d'autorisation permettant au FS d'accéder à une ressource de l'utilisateur protégée et généralement hébergée par le FI. Ces ressources sont la plupart du temps des informations d'identité (attributs) supplémentaires concernant l'utilisateur authentifié. Le consentement de l'utilisateur avant toute transmission de ces informations est obligatoire.

3.2 Détermination du fournisseur d'identité

La première étape consiste pour le FS à déterminer le FI associé à l'utilisateur. Il existe plusieurs façons d'y arriver.

Une première méthode consiste à associer tous les utilisateurs au même FI. C'est évidemment la solution la plus simple, mais elle ne permet pas au sens strict la mise en œuvre de la fédération d'identité.

Il est également possible de proposer une page statique contenant la liste des FI reconnus par le FS. C'est alors l'utilisateur lui-même qui choisit le FI auprès de qui il est enregistré. On parle alors d'*association statique*.

Enfin, la dernière manière, appelée « *Issuer discovery* » dans le protocole, consiste à associer dynamiquement un FI à l'utilisateur. Pour cela, le FS utilise les éléments d'identité qu'il a en sa possession (typiquement, une adresse électronique), et en déduit le FI correspondant.

La seconde option (l'association statique) est celle qui est le plus souvent retenue, comme le montre la figure 2.

Dans tous les cas, le FS va récupérer certaines métadonnées propres au FI. Les *endpoints* que l'on peut citer en particulier sont les URI déclarées par le FI et qui permettent par exemple :

- la bonne redirection de l'utilisateur pour son authentification (`authorization_endpoint`) ;
- la récupération par le FS des données personnelles de l'utilisateur authentifié (`token_endpoint`) ;
- l'enrôlement dynamique du FS auprès du FI (`registration_endpoint`).

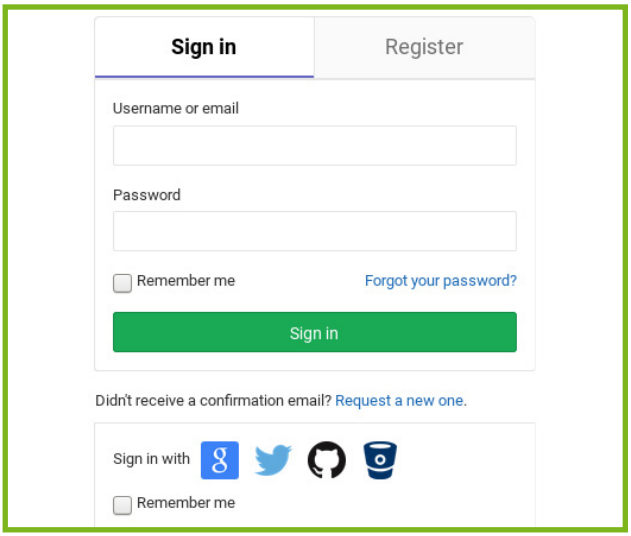


Fig. 2 : Exemple de formulaire mettant en œuvre l'association statique du FI (GitLab propose de s'authentifier avec un compte GitLab, ou via OIDC en choisissant l'un des FI listés en bas du formulaire).

3.3 Enrôlement du fournisseur de service

La seconde étape consiste si ce n'est déjà fait à faire enrôler le FS auprès du FI. Cette phase peut encore une fois s'effectuer de manière statique ou dynamique. Dans le cas statique, les administrateurs du FI et du FS renseignent les informations en amont de l'ouverture du service aux utilisateurs. Dans le cas dynamique (*Dynamic Client Registration*), les informations sont échangées à la volée, de façon automatique, entre le FI et le FS, lorsqu'un utilisateur renseigne un FI pour la première fois auprès du FS.

Quelle que soit l'option choisie, l'établissement de ce lien, assimilable à un contrat, donne lieu à l'échange des informations suivantes :

- **`client_id`** : identifiant unique du FS auprès du FI. Il s'agit d'une valeur publique, généralement générée aléatoirement par le FI ;
- **`client_secret`** : cette valeur est optionnelle, mais est généralement utilisée pour authentifier le FS lorsqu'il récupère un code d'autorisation auprès du FI. Elle doit rester secrète et n'être connue que par les deux acteurs ;
- **`redirect_uri`** : URL appartenant au domaine du FS utilisée par le FI après l'authentification de l'utilisateur pour renvoyer la réponse d'autorisation au FS (via l'utilisateur).

3.4 Cinématique du « code d'autorisation »

Une fois le couple FS / FI établi lors de la phase d'enrôlement, la phase d'authentification de l'utilisateur peut commencer. La figure 3 fait apparaître la cinématique

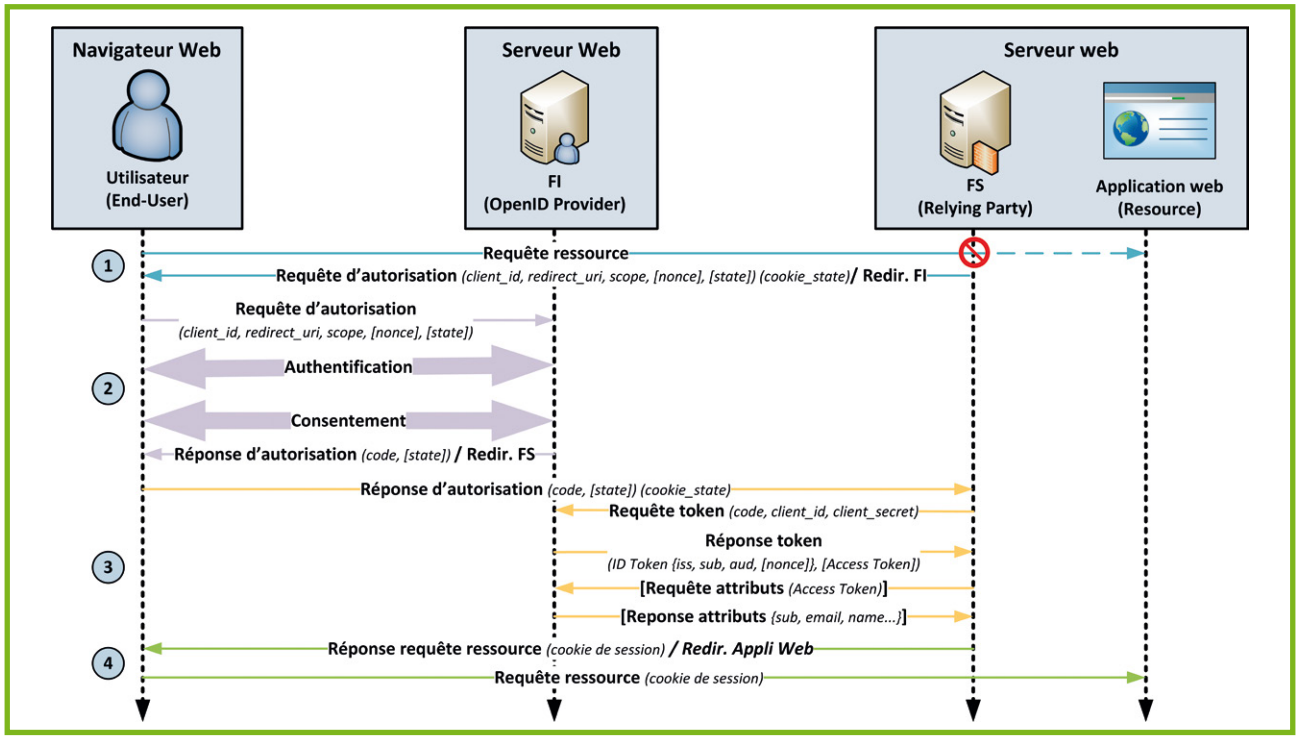


Fig. 3 : Cinématique du code d'autorisation.

avec les principaux paramètres échangés (ceux considérés dans le standard comme non obligatoires - seulement recommandés ou optionnels - sont encadrés entre crochets). Afin de comprendre le processus mis en place par OIDC et l'attaque présentée au paragraphe suivant, détaillons leur signification.

- **`client_id`**, **`client_secret`**, **`redirect_uri`** : paramètres explicités précédemment et identiques à ceux échangés lors de la phase d'enrôlement ;
- **`scope`** : liste les ressources de l'utilisateur protégées par le FI auxquelles souhaite accéder le FS. Il s'agit généralement d'informations d'identité supplémentaires tels que l'adresse électronique ;
- **`nonce`** : valeur opaque générée par le FS lors de la requête d'autorisation. Le FI la reçoit via l'utilisateur lors du renvoi de cette requête et l'intègre ensuite à l'*ID Token*. Le FS se charge de comparer la valeur stockée dans la session de l'utilisateur à la valeur reçue dans l'*ID Token* (mécanisme anti-rejeu) ;
- **`state`** : valeur opaque générée par le FS lors de la requête d'autorisation. Bien que cela ne soit pas imposé, mais seulement mentionné par le protocole, ce paramètre est généralement accompagné d'un cookie (appelé dans la suite de cet article **`cookie_state`**) contenant une autre valeur opaque. Ces deux valeurs sont ensuite renvoyées in fine par le navigateur web de l'utilisateur avec le code d'autorisation. À travers la vérification de la cohérence du **`state`** et du **`cookie_state`**, le FS s'assure ainsi qu'il s'agit bien du même utilisateur entre la requête initiale et la réponse d'autorisation (mécanisme anti *cross-site request forgery*) ;
- **`cookie_session`** : cookie permettant de faire le lien entre l'utilisateur authentifié et sa session sur le FS.

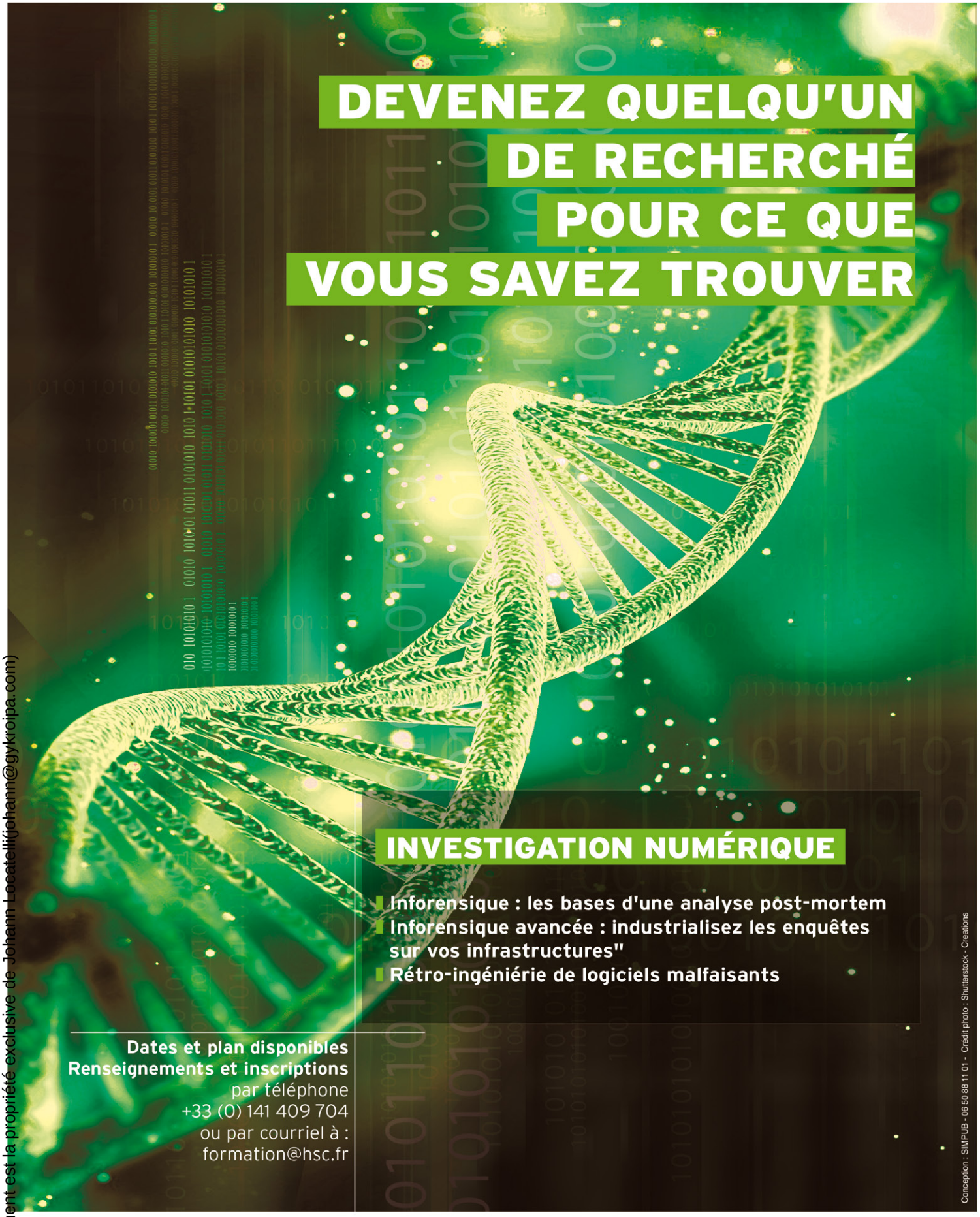
Bien que ce mécanisme ne fasse pas à proprement parlé partie du protocole OIDC, il est aujourd'hui largement utilisé. Le navigateur de l'utilisateur le reçoit classiquement une fois que le FS a validé les informations d'identité récupérées auprès du FI et qu'il a effectué son contrôle d'accès.

Étape 1. L'utilisateur non authentifié essaye d'accéder à une ressource protégée. Sa requête est interceptée par le FS qui prépare une requête d'autorisation. Celle-ci est renvoyée à l'utilisateur qui la présente à son tour à son fournisseur d'identité via l'**`authorization_endpoint`**.

Étape 2. Le FI vérifie la requête d'autorisation puis invite l'utilisateur à s'authentifier. Ce dernier est ensuite sollicité pour autoriser la transmission des informations demandées par le FS dans le paramètre **`scope`**. Son consentement acquis, le FI transmet au FS via le navigateur de l'utilisateur une réponse d'autorisation contenant notamment le code d'autorisation.

Étape 3. Le FS vérifie la réponse d'autorisation fournie par l'utilisateur puis en extrait le code d'autorisation. Le FS transmet alors ce code via une requête de token au FI. Ce message, authentifié à l'aide du *client_ID* et du *client_secret*, est envoyé sur le **`token_endpoint`**. Le FI authentifie le FS, vérifie le code d'autorisation et envoie au FS le jeton d'identité (*ID Token*) correspondant à l'utilisateur authentifié. Un *Access Token* peut également être fourni à ce stade pour récupérer des informations d'identité supplémentaires. Cette démarche s'effectuera le cas échéant par le FS via une requête d'attributs.

Étape 4. Le FS vérifie l'*ID Token* et retrouve l'identité de l'utilisateur pour effectuer son contrôle d'accès. Si l'utilisateur a le droit d'accéder au service désiré, le FS lui délivre la ressource demandée initialement.



www.hsc-formation.fr

HSC by Deloitte.

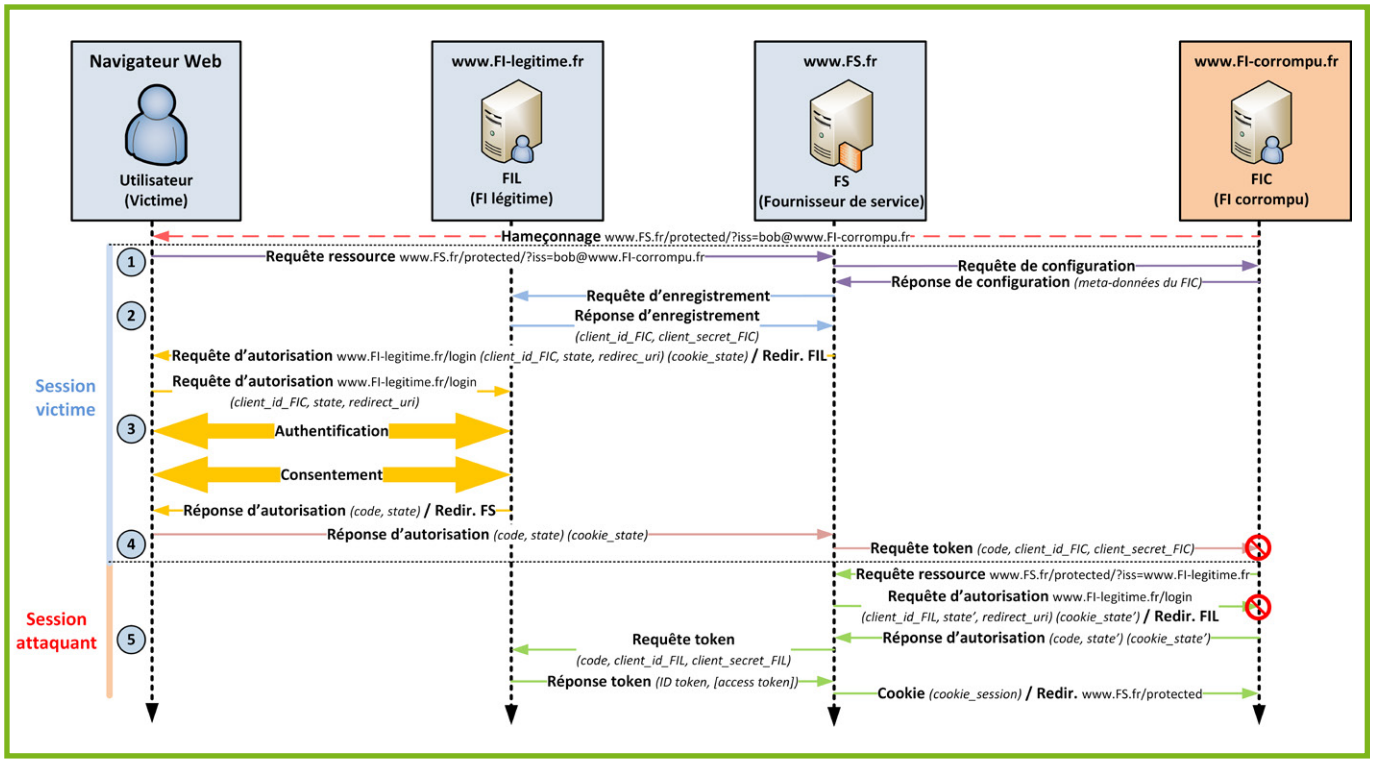


Fig. 4 : Déroulement de l'attaque Broken End-User Authentication.

4 L'attaque Broken End-User Authentication

En 2016, une équipe de chercheurs de l'université de la Ruhr à Bochum a publié une analyse sur la sécurité du protocole OIDC **[BEA]**. Cette étude a en particulier mis au jour une attaque intitulée *Broken End-User Authentication*.

Cette attaque est intéressante, car elle montre que dans un standard complexe comme OpenID Connect, le diable se cache dans les détails.

4.1 Hypothèse et modèle de l'attaquant

Avant toute chose, on suppose que l'attaquant évolue dans le contexte suivant :

- TLS est systématiquement utilisé lors des échanges entre les différents acteurs et l'attaquant ne peut intercepter ou modifier ces communications ;
- les logiciels utilisés (système d'exploitation, navigateurs, serveurs web, etc.) ne présentent pas de faille exploitable ;
- les secrets utilisés dans OIDC sont générés correctement : l'attaquant n'a pas les moyens d'effectuer en temps raisonnable une attaque générique pour retrouver ces valeurs ;
- le champ `state` (et le cookie associé) est utilisé par le FS dans sa requête d'autorisation, mais pas le champ `nonce` qui reste optionnel dans les spécifications.

On s'intéresse à une victime (un utilisateur final), enregistré auprès d'un fournisseur d'identité légitime (FIL). Il utilise régulièrement un fournisseur de services donné, FS, auprès duquel il s'authentifie en utilisant son identité du FIL.

L'objectif de l'attaquant est d'accéder aux ressources appartenant à la victime et stockées par le FS.

L'attaquant peut mettre en place un fournisseur d'identité corrompu (FIC) et initier des communications vers les acteurs légitimes. De plus, on suppose que l'attaquant peut réaliser des campagnes d'hameçonnage auprès de sa victime pour l'inviter à suivre un lien corrompu.

On suppose enfin que les mécanismes de découvertes de FI (*Issuer discovery*) et d'enrôlement automatique (*Dynamic Client Registration*) sont activées chez les acteurs légitimes.

4.2 Déroulement de l'attaque

Étape 1 (association de la victime avec le FI corrompu). L'attaquant envoie par hameçonnage un lien pointant vers le FS. Il contient dans les paramètres URL une information d'identité (une adresse mail par exemple) associée au domaine de l'attaquant. Lorsque l'utilisateur

suit le lien, le FS légitime reçoit sa requête et utilise l'information d'identité reçue pour associer, via le service *Issuer discovery* , l'utilisateur au FI corrompu. Un dialogue propre au service *Issuer discovery* s'installe ensuite entre le FS et le serveur de l'attaquant pour récupérer les métadonnées. Le FI corrompu lui transmet les informations suivantes (liste non exhaustive), mêlant des informations relatives au FI légitime et d'autres concernant le FI corrompu :

```
{
  "issuer" : "https://www.FI-corrompu.fr",
  "registration_endpoint" : "https://www.FI-legitime.fr/register" ,
  "authorization_endpoint" : "https ://www.FI-legitime.fr/login" ,
  "token_endpoint" : "https://www.FI-corrompu.fr/token"
}
```

Étape 2 (enrôlement du FI corrompu). Bien que le FS soit déjà enregistré auprès du FI légitime, il se base sur le champ **`issuer`** des métadonnées pour identifier le FI. En l'occurrence, il ne s'est jamais enrôlé auprès du FI corrompu. Le FS lance donc une procédure d'enregistrement à travers son service *Dynamic Client Registration* à l'URL indiquée dans le champ **`registration_endpoint`** (https://www.FI-legitime.fr/register). Le FI légitime lui envoie un identifiant (**`client_id_FIC`**) et un secret (**`client_secret_FIC`**) dont l'attaquant n'a pas connaissance.

Étape 3 (authentification de la victime). Une fois enregistré, le FS génère sa requête d'autorisation et renvoie la victime s'authentifier à l'adresse indiquée dans le champ **`authorization_endpoint`** (https://www.FI-legitime.fr/login). Les phases d'authentification et de consentement qui s'en suivent s'effectuent sans difficulté puisque la victime fait confiance au FI légitime qui gère son identité. L'utilisateur reçoit ensuite un code d'autorisation qu'il renvoie au FS.

Étape 4 (vol du code d'autorisation). Afin de recevoir l'*ID Token* lié à l'utilisateur, le FS envoie ce code d'autorisation, ainsi que le **`client_id_FIC`** et le **`client_secret_FIC`** à l'adresse indiquée dans le champ **`token_endpoint`** des métadonnées (https://www.FI-corrompu.fr/token). À ce stade, l'attaquant reçoit donc trois informations : le **`client_id_FIC`**, le **`client_secret_FIC`** ainsi que le code d'autorisation. Ces deux dernières informations sont particulièrement sensibles, car elles ne sont pas supposées être révélées à une tierce personne.

Étape 5 (usurpation d'identité). L'attaquant initie sa propre session avec le FS en lui envoyant une requête spécifiant qu'il est associé au FI légitime. Le FS est déjà enrôlé auprès du FI légitime et a déjà reçu ses métadonnées. Il génère donc directement une requête d'autorisation qu'il envoie à l'attaquant. Celui-ci ignore la redirection vers le FI légitime et renvoie au FS une réponse d'autorisation contenant le code d'autorisation précédemment récupéré. Lorsque le FS envoie au FI légitime une requête token contenant ce code, il reçoit en retour l'*ID Token* associé à la victime. À ce moment, l'attaquant a usurpé l'identité de la victime auprès du FS et peut donc accéder à ses ressources.

4.3 Analyse des causes

4.3.1 Absence de mécanisme anti-rejeu

L'usurpation d'identité est notamment rendue possible par la réutilisation par l'attaquant du code d'autorisation de la victime précédemment récupéré par l'intermédiaire du FS. L'utilisation dans la requête d'autorisation du paramètre **`nonce`** généré par le FS permet à celui-ci de corréler l'*ID Token* reçu en échange du code d'autorisation avec la session de l'utilisateur. Dans notre scénario, l'usurpation d'identité n'aurait donc pas pu être réalisée dans ces conditions puisque les **`nonces`** des deux sessions auraient été différents.

4.3.2 Contrôle de la phase d'enrôlement

La cause initiale provient d'une manipulation des métadonnées envoyées par le FIC au FS. Certains *endpoints* font référence au FIL et d'autres au FIL, ce qui engendre l'enregistrement du FS auprès du FIL alors que l'utilisateur est associé au FIC... Il aurait été utile de vérifier que les métadonnées envoyées par le FI lors de son enrôlement, et notamment les *endpoints*, correspondent bien au domaine du FI déclaré, ce qui aurait contré l'attaque. Cette vérification est évidemment plus compliquée à faire lorsque l'enregistrement peut être fait de manière dynamique.

4.3.3 Absence de garanties d'intégrité et de confidentialité sur certains échanges

Une fois son FIC enrôlé, deux étapes permettent à l'attaquant de réaliser son attaque si l'on met de côté la campagne d'hameçonnage :

- la première consiste à récupérer lors de l'étape 4 le code d'autorisation de la victime généré par le FIL. Garantir la confidentialité de ce code dans les échanges entre les acteurs concernés (soit du point de vue du FIL : lui-même et le FS) n'aurait pas permis à l'attaquant d'accéder à cette information capitale ;
- la seconde consiste pour l'attaquant à fabriquer à l'étape 5 une réponse d'autorisation ad hoc contenant le code d'autorisation de la victime récupéré à l'étape précédente. Garantir l'intégrité de cette réponse d'autorisation, normalement générée par le FI, aurait permis d'éviter une telle manipulation.

Malheureusement, le protocole ne permet pas en l'état actuel de garantir l'intégrité et la confidentialité sur ces deux échanges, bien que le FI et le FS disposent d'un secret partagé pour réaliser ces deux opérations.

Au-delà de l'attaque présentée dans cette section, d'autres scénarios possibles ont été décrits pour mettre

en cause les propriétés de sécurité d'OpenID Connect. Ils reposent sur des hypothèses différentes et suivent des cheminements différents. Le lecteur intéressé pourra notamment consulter l'attaque IdP Confusion **[SOK]**.

5 Contremesures et recommandations

5.1 Cela va sans dire...

Tout d'abord, les recommandations classiques liées à la mise en œuvre de ce type de mécanismes sur le Web doivent être appliquées.

En premier lieu, l'ensemble des communications doit être protégé par TLS.

Il est également essentiel de s'assurer de la bonne entropie de l'ensemble des secrets générés : ceux-ci doivent être tirés aléatoirement, et être d'une taille suffisamment longue. Cette recommandation s'applique évidemment au code d'autorisation, mais également aux champs `state`, `nonce` et au `client_secret`.

Afin de pouvoir détecter et déverminer les problèmes éventuels, il faut enfin journaliser les messages échangés. Cela permet dans certains cas de détecter des anomalies, au moins a posteriori. Ici, le FS reçoit coup sur coup deux codes d'autorisation identiques, issus de FI distincts, ce qui ne devrait a priori pas arriver.

5.2 Durcissement d'OpenID Connect

Il faut d'abord utiliser tous les mécanismes mis à notre disposition dans la spécification pour garantir que la vision des différents acteurs soit cohérente. Il est donc indispensable d'utiliser les champ `nonce`, `state` et de mettre en place un `cookie_state` correspondant.

Le protocole prévoit également une phase d'authentification du FS auprès du FI dans la requête token. Au lieu d'envoyer le mot de passe `client_secret` directement, une méthode d'authentification plus sécurisée doit être privilégiée. Cela peut passer soit par l'utilisation de clefs asymétriques, soit par l'utilisation du mécanisme symétrique `client_secret_jwt` qui repose sur HMAC et qui permet de ne pas divulguer le `client_secret.`.

Au-delà de la protection du canal de communications, il est également utile de protéger les objets transportés, les jetons JWT. Pour cela, il est fortement recommandé de signer et de chiffrer les jetons JWT. Comme indiqué dans la section 4.3, certains paramètres ne peuvent pas être incorporés dans un JWT comme le code d'autorisation présent dans la requête token et dans la réponse d'autorisation.

En revanche, la spécification permet d'utiliser les JWT pour protéger les paramètres de la requête d'autorisation, notamment les aléas comme `state` ou `nonce`. Il faut donc imposer que ces éléments ne soient transmis que chiffrés et signés, uniquement au sein de jetons JWT. Ainsi, on protège ces éléments sensibles et on prive l'attaquant de leur connaissance, ce qui peut compliquer une usurpation d'identité.

5.3 Un mot sur l'enregistrement à la volée

Enfin, dans l'attaque décrite ci-dessus, une des causes est l'autorisation pour les FS de s'enregistrer de manière automatique auprès d'un FI. En pratique, l'enrôlement est une opération importante qui correspond à un contrat entre les deux fournisseurs. Il semble donc important que les informations utilisées / recueillies lors de cette étape soient vérifiées scrupuleusement, idéalement via un contrôle humain. Mais sans aller jusque-là, l'utilisation par un FI donné du même `registration_endpoint` qu'un autre FI, comme c'est le cas ici, devrait au moins lever une alerte !

De manière plus générale, il faut se méfier des modes automatiques de découverte et d'enregistrement de fournisseurs.

Conclusion

Le concept d'authentification unique est très utile dans le monde du Web. Combiné avec d'autres outils (coffres forts de mots de passe, authentification à deux facteurs), la fédération d'identité peut aider à améliorer de manière notable la sécurité des communications et des accès. OpenID Connect est aujourd'hui la solution la plus en vogue, en particulier dans des contextes de déploiement largement ouverts.

Cependant, si ce standard présente de bonnes propriétés, le fonctionnement du protocole est assez complexe, et peut mettre en jeu un nombre important d'acteurs, pas forcément tous de confiance. Il est donc important d'appliquer de bonnes pratiques à son déploiement OpenID Connect et d'utiliser si possible une implémentation certifiée par la fondation OpenID. L'objectif est d'assurer que l'ensemble des acteurs légitimes (utilisateur, FI et FS) ait une vision cohérente de la situation, et ce malgré l'utilisation d'échanges bilatéraux. ■

Remerciements

Les auteurs tiennent à remercier Vincent Lancino qui a bien voulu relire cet article.

Retrouvez toutes les références de cet article sur le blog de MISC : https://www.miscmag.com/ .

Abonnez-vous !

MISC
LE MAGAZINE DE LA SÉCURITÉ INFORMATIQUE MULTIPLATEFORME !

M'abonner !

Me réabonner !

Compléter ma collection !

Pouvoir lire en ligne mon magazine préféré !

PARTICULIERS,

➡ Rendez-vous sur :

www.ed-diamond.com

pour consulter toutes les offres !

➡ ...ou renvoyez-nous le document au verso complété !

PROFESSIONNELS,

➡ Rendez-vous sur :

proboutique.ed-diamond.com

pour consulter toutes les offres dédiées !

➡ ...ou renvoyez-nous le document au verso complété !

CONNECT

DÉCOUVREZ CONNECT
LA PLATEFORME DE LECTURE EN LIGNE !

Tous les articles du magazine sont disponibles dès la parution !
Pour plus de renseignements, contactez-nous :
par téléphone au +33 (0) 3 67 10 00 28 ou par mail à connect@ed-diamond.com

À découvrir sur : **connect.ed-diamond.com**

VOICI LES OFFRES D'ABONNEMENT AVEC MISC !

CHOISISSEZ VOTRE OFFRE

ou retrouvez toutes nos offres sur www.ed-diamond.com !

Prix TTC en Euros / France Métropolitaine*

*Les tarifs hors France Métropolitaine, Europe, Asie, etc. sont disponibles en ligne !

Offre	ABONNEMENT	PAPIER Réf	PAPIER Tarif TTC*	PAPIER + CONNECT (1 connexion Connect) Réf	PAPIER + CONNECT Tarif TTC*
MC	6n° MISC	☐ MC1	45 €	☐ MC13	259 €
MC+	6n° MISC + 2n° HS	☐ MC+1	65 €	☐ MC+13	279 €
	LES COUPLAGES AVEC NOS AUTRES MAGAZINES				
B	6n° MISC + 11n°GLMF	☐ B1	109 €	☐ B13	499 €
B+	6n° MISC + 2n° HS + 11n°GLMF + 6n° HS	☐ B+1	185 €	☐ B+13	629 €
C	6n° MISC + 6n° LP + 11n°GLMF	☐ C1	149 €	☐ C13	669 €
C+	6n° MISC + 2n° HS + 6n° LP + 3n° HS + 11n°GLMF + 6n° HS	☐ C+1	249 €	☐ C+13	769 €
I	6n° MISC + 6n° HK	☐ I1	79 €	☐ I13	419 €
I+	6n° MISC + 2n° HS + 6n° HK	☐ I+1	99 €	☐ I+13	439 €
L	6n° MISC + 6n° HK + 6n° LP + 11n° GLMF	☐ L1	189 €	☐ L13	839 €
L+	6n° MISC + 2n° HS + 6n° HK + 6n° LP + 3n° HS + 11n°GLMF + 6n° HS	☐ L+1	289 €	☐ L+13	939 €

Les abréviations des offres sont les suivantes : GLMF = GNU/Linux Magazine France | HS = Hors-Série | LP = Linux Pratique | HK = Hackable

J'indique l'offre si différente que celles ci-dessus :

J'indique la somme due (Total) : €

Je choisis de régler par :

❑ **Chèque bancaire ou postal à l'ordre des Éditions Diamond (uniquement France et DOM TOM)**

❑ **Pour les règlements par virements, veuillez nous contacter via e-mail : cial@ed-diamond.com ou par téléphone : +33 (0)3 67 10 00 20**

SÉLECTIONNEZ VOTRE OFFRE DANS LA GRILLE CI-DESSUS 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.

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 :

.....

.....

RETROUVEZ TOUTES NOS OFFRES SUR : www.ed-diamond.com !

*Les tarifs hors France Métropolitaine, Europe, Asie, etc. sont disponibles en ligne !

COMPRENDRE LE FONCTIONNEMENT DES CORS

Christophe RIEUNIER – @sdkddk

mots-clés : PENTEST / WEB / SOP / CORS / CSRF / XSS

Introduits afin d'assouplir la Same Origin Policy pour permettre au Web 2.0 de donner sa pleine mesure, les Cross Origin Ressource Sharing (CORS) souffrent de spécifications pour le moins absconses et d'une dissymétrie d'implémentation particulièrement dommageable à leur bonne mise en œuvre. En conséquence, leur activation côté serveur conduit très souvent à la création de vulnérabilités exploitables susceptibles de faire le bonheur du pentesteur.

Cet article est découpé en deux parties : la première présente les CORS, la seconde s'attache à décrire les cas fréquents de mauvaises configurations débouchant sur des vulnérabilités exploitables, puis détaille plusieurs d'entre elles et suggère quelques contrôles et recommandations à destination des pentesteurs et de leurs clients.

1 Genèse des CORS

En 1995, le navigateur Netscape 2 introduit JavaScript, et avec lui la possibilité d'exécuter du code en provenance d'un site web. Il devient dès lors possible de réaliser des actions à l'insu de l'utilisateur et d'agir de manière silencieuse à sa place.

Un mécanisme nommé *Same Origin Policy* (désigné par l'acronyme SOP dans la suite de cet article) est alors implémenté au sein des agents. Ce mécanisme garantit l'impossibilité d'agir à l'insu d'un internaute sur un site (non vulnérable) en empêchant tout contenu provenant d'une origine A d'accéder au résultat d'une requête qu'il émettrait vers une origine B.

La SOP devient ainsi la pierre angulaire de la sécurité du web. Pour une révision de ses principes détaillés, le lecteur pourra se reporter à la RFC6454 « The Web Origin Concept » **[1]** et à l'excellent article de *MISC* consacré au sujet **[2]**. Comme son nom l'indique, la SOP repose sur la notion d'origine. Il est donc primordial de définir clairement le concept d'origine, lequel est très simple : une origine est constituée d'un protocole, d'un nom d'hôte et d'un port. Ainsi, les ressources `https://www.domaine.com` et `https://www.domaine.com/index.html` relèvent de la même origine. Alors que `http://www.domaine.com` et `https://www.domaine.com` relèvent de deux origines différentes.

On notera que pour ce qui concerne les CORS, et comme pour Internet Explorer, une origine est constituée uniquement d'un protocole et d'un nom d'hôte.

Autre point important pour la suite, la SOP est exclusivement implémentée par les agents, c'est-à-dire les navigateurs.

En 1999, Microsoft invente avec la première version d'Outlook Web Access (le client web d'Outlook) ce qui deviendra le fondement du Web 2.0, c'est-à-dire Ajax et l'objet `XMLHttpRequest` **[3]** ou XHR pour les intimes.

L'objet `XMLHttpRequest` permet d'envoyer des requêtes HTTP depuis un script JavaScript vers n'importe quel serveur, y compris des requêtes authentifiées. Il permet ainsi d'accéder à des ressources exposées par des Web Services et de concevoir des clients dits « riches » proposant une expérience utilisateur beaucoup plus agréable. Cependant, la SOP interdisant par construction d'accéder au contenu de la réponse d'une requête inter-origines, elle devient un problème dans le cadre de web services destinés à être consommés par un agent plutôt que par un serveur en back-to-back.

À partir de mi-2005, le W3C a donc planché sur une possibilité d'assouplir la SOP afin de supporter les nouveaux usages, sans toutefois mettre en péril la sécurité du Web. Ces travaux ont donné lieu à l'extension CORS ou *Cross Origin Ressource Sharing* **[4]**, laquelle autorise les requêtes inter-origines selon des règles précises, mais malheureusement pas franchement claires.

À ce jour, et depuis quelques années, tous les navigateurs implémentent les CORS.

Pour mémoire, l'extension CORS fait suite à JSONP qui n'est plus vraiment d'actualité et ne sera pas abordé dans cet article.

1.1 Rappels concernant les vulnérabilités de type CSRF et XSS

Pour la bonne compréhension de la suite de cet article, on insistera sur le fait que la SOP n'empêche pas une requête d'être émise, mais seulement de consulter la réponse reçue du serveur lorsque la requête est émise depuis une origine différente.

Ainsi la SOP ne protège pas d'attaques de type CSRF (*Cross Site Request Forgery*) consistant à émettre une requête simple ou via un POST en *autosubmit* qui sera forcément honorée par le serveur, d'autant qu'elle sera le cas échéant accompagnée du ou des cookies de session existant au sein du navigateur. Il appartient dès lors au serveur de se protéger des attaques CSRF en découpant les requêtes susceptibles de modifier des données en deux temps : une première destinée généralement à obtenir un formulaire de création/modification/suppression qui contiendra un aléa généré par le serveur pour l'occasion et une seconde destinée à envoyer le formulaire (donc l'ordre de modification) au serveur, lequel vérifiera la valeur de l'aléa (associé à la session et à usage unique ou regénéré lors de chaque ouverture de session) avant de réaliser les modifications demandées. La SOP empêchera alors tout script malveillant de récupérer la valeur de l'aléa et donc de générer des requêtes de modification valides à l'insu de l'utilisateur.

De la même manière, si la SOP limite la portée des vulnérabilités de type *Cross Site Scripting* (XSS) aux bornes de chaque origine, nous verrons plus bas que les CORS lorsqu'elles sont mal configurées permettent d'étendre la portée de ces attaques et d'utiliser les vulnérabilités de type XSS de n'importe quel site pour rendre inopérantes les protections anti-CSRF d'un site comportant des CORS mal configurées.

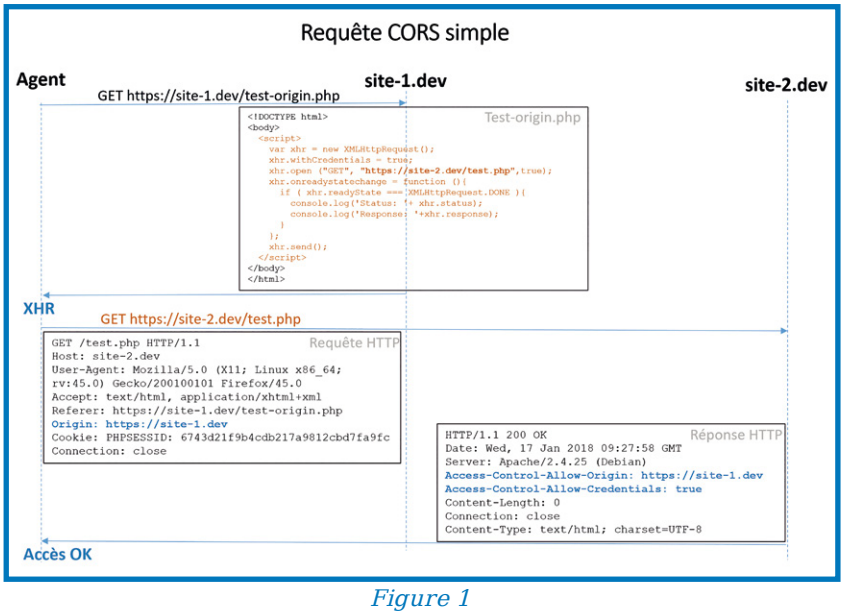


Figure 1

2 Fonctionnement des CORS

L'implémentation des CORS consiste à ajouter des en-têtes dans les requêtes émises par les agents et dans les réponses envoyées par les serveurs. Ce choix s'explique par la volonté de ne pas impacter le fonctionnement des couples (agent, serveurs délivrant des ressources) antérieurs à la définition des CORS.

Les CORS se découpent en deux types de requêtes : les requêtes dites simples consistent uniquement en l'ajout d'en-têtes aux requêtes et réponses « normales », alors que les requêtes de type préliminaires (ou « preflight ») nécessitent comme leur nom l'indique une étape préliminaire.

2.1 Requêtes simples

D'une manière générale, les requêtes simples recouvrent le type de requêtes qu'un agent ne supportant pas les CORS est susceptible d'émettre. Les requêtes simples concernent donc uniquement les verbes **`GET`**, **`HEAD`** et **`POST`** (avec une limitation sur les **`Content-type`** pour le **`POST`** où seuls les types **`application/x-www-form-urlencoded`**, **`multipart/form-data`** et **`text/plain`** sont autorisés).

Le principe d'une requête simple est le suivant :

- l'agent ajoute à la requête un en-tête **`Origin:`** spécifiant l'origine du script ayant généré la requête ;
- le serveur répond :
  - sans en-tête **`Access-Control-Allow-Origin:`** s'il refuse l'origine ou ne supporte pas les CORS ;
  - avec un en-tête **`Access-Control-Allow-Origin: <origine>`** reprenant strictement l'origine envoyée dans la requête si le serveur accepte cette origine ;
  - avec un en-tête **`Access-Control-Allow-Origin: *`** si le serveur accepte toutes les origines.

La réponse est accessible du script ayant généré la requête si l'origine retournée par le serveur correspond strictement à l'origine envoyée ou si le serveur accepte toutes les origines.

Bien que l'on rencontre très souvent ce type de réponse, le serveur ne peut pas renvoyer d'origines multiples, que ce soit sous la forme d'une liste d'origines, de jokers ou d'expressions régulières. Dans ce cas, l'agent considèrera que le serveur refuse l'origine soumise.

Le synoptique d'une requête simple peut être illustré par le schéma suivant où un script renvoyé par la ressource

https://site1.dev/test-origin.php génère une requête **GET** vers la ressource https://site2.dev/test.php. L'agent ajoute à la requête **GET** un en-tête **Origin: https://site1.dev** et le serveur du site 2 répond avec les en-têtes **Access-Control-Allow-Origin: https://site1.dev** et **Access-Control-Allow-Credentials: true** autorisant de ce fait le script à accéder au contenu de la réponse à la requête authentifiée qu'il a générée (cf **xhr.withCredentials** à **true** et le cookie **PHPSESSID** envoyé dans le **GET**) (Figure 1).

2.2 Requêtes préliminaires (preflight)

Pour tout type de requête autre que simple (i.e. requêtes **DELETE**, **PUT** ou **PATCH** et requêtes avec un **content-type** différent des trois listés ci-dessus), l'agent va d'abord envoyer une requête préliminaire via le verbe **OPTIONS** accompagné d'un en-tête **Origin:** spécifiant l'origine de la requête.

L'agent pourra aussi envoyer les en-têtes optionnels suivants :

- **Access-Control-Request-Method :** indiquant la méthode qu'il souhaite employer, par exemple **PUT**, **DELETE**, **PATCH**, etc.
- **Access-Control-Request-Headers :** indiquant la liste des en-têtes particuliers qui seront envoyés par l'agent.

Le serveur répond alors avec les en-têtes suivants :

- **Access-Control-Allow-Origin :** dont l'usage est strictement identique au cas des requêtes simples ;
- **Access-Control-Allow-Credentials :** dont l'usage est strictement identique au cas des requêtes simples et sera détaillé au paragraphe suivant ;
- **Access-Control-Allow-Max-Age :** indique la durée pendant laquelle l'agent peut mettre la réponse à la requête **OPTIONS** en cache ;
- **Access-Control-Allow-Methods :** spécifie la liste des verbes HTTP acceptés par le serveur lors de requêtes CORS ;
- **Access-Control-Allow-Headers :** spécifie la liste des en-têtes particulières HTTP que l'agent pourra envoyer au serveur lors de requêtes CORS. Tout autre en-tête que **Accept:**, **Accept-Language:**, **Content-Language:**, **Content-Type: application/x-www-form-urlencoded**, **Content-Type: multipart/form-data** ou **Content-Type: text/plain** est considéré comme un en-tête particulier ;
- **Access-Control-Expose-Headers :** spécifie la liste des en-têtes de réponses particuliers dont le contenu pourra être accessible du script ayant généré la requête. Tout autre en-tête de réponse que **Cache-Control:**, **Content-Language:**, **Content-Type:**, **Expires:**, **Last-Modified:** et **Pragma:** est considéré comme un en-tête particulier.
- **Vary: origin** : si le serveur accepte plusieurs origines, comme il ne peut en spécifier qu'une dans sa réponse, il doit absolument dans ce cas renvoyer un en-tête **Vary: origin** pour indiquer à l'agent, mais aussi à des équipements intermédiaires tels que des proxy que sa réponse dépend de l'origine soumise et éviter la mise en cache de celle-ci et sa réutilisation lors de requêtes provenant d'origines différentes. De nombreuses implémentations des CORS négligent ce point (voir par exemple la vulnérabilité CVE-2017-7674 récente d'Apache Tomcat).

Si la réponse à la requête **OPTIONS** correspond à la demande du script, alors la vraie requête est envoyée par l'agent, sinon aucune requête supplémentaire n'est émise.

Le synoptique d'une requête préliminaire peut être illustré par le schéma suivant où un script renvoyé par la ressource https://site1.dev/test-origin-preflight.php génère une requête **DELETE** vers la ressource https://site2.dev/test.php. L'agent va générer une requête **OPTIONS** en préalable à l'envoi de la requête **DELETE**, le serveur acceptant les méthodes **DELETE** et **PATCH** va répondre à la requête **OPTIONS** avec un en-tête supplémentaire **Access-Control-Allow-Methods: DELETE, PATCH**, l'agent enverra donc la requête **DELETE** générée par le script (Figure 2).

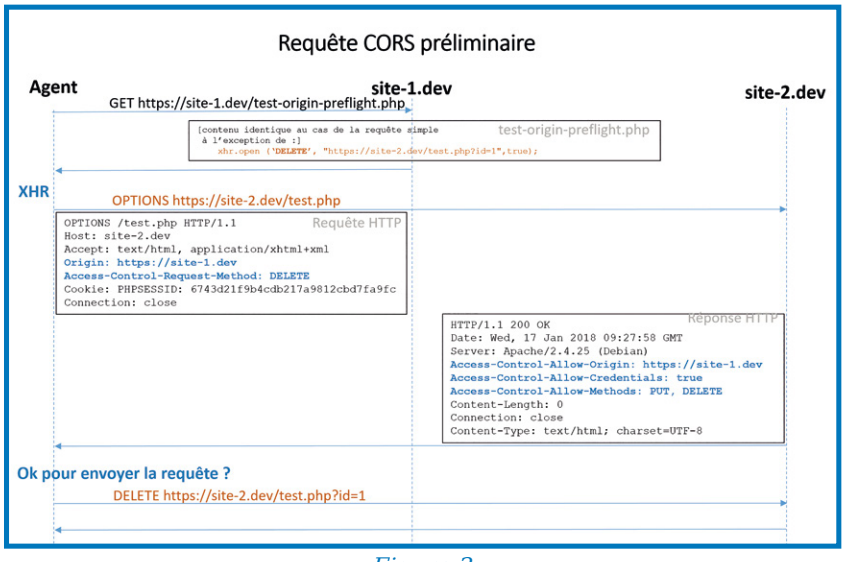


Figure 2

2.3 Requêtes avec credentials

Si le client souhaite envoyer une requête authentifiée (i.e. avec *credentials* sous la forme de cookies, d'en-tête **Authorization** ou de certificat), alors il met

Cas d'une requête simple				
xhr.withCredentials	Credentials envoyés (cookies, authorization)	En-têtes reçus		Réponse accessible par le script
		ACAC	ACAO	
False	Non	[pas d'ACAC]	[Sans objet]	NON
		http://script.com		OUI
		*		OUI
True	Oui	[pas d'ACAC]	[Peu importe]	NON
		http://script.com	true	OUI
			[absent]	NON
		*	true	NON
			[absent]	NON

Tableau 1

Cas d'une requête préliminaire					
xhr.withCredentials	Résultat requête OPTIONS (1)	Credentials envoyés (cookies, authorization)	En-têtes reçus		Réponse accessible par le script
			ACAC	ACAO	
False	OK	Non	http://script.com	[Sans objet]	OUI
	NOK	[Sans objet : la requête demandée ne sera pas envoyée]			
True	OK	Oui	http://script.com	true	OUI
	NOK	[Sans objet : la requête envoyée ne sera pas envoyée]			

Tableau 2

l'attribut `.withCredentials` de l'objet `XHR` à `true`. Dans le cas d'une requête simple, les *credentials* seront envoyés s'ils existent et le serveur autorisera alors le script à consulter sa réponse en émettant un en-tête `Acces-Control-Allow-Origin:` reprenant strictement l'origine soumise et un en-tête `Access-Control-Allow-Credentials: true`. Si le serveur refuse au script l'accès à sa réponse, il n'enverra pas ce dernier en-tête. Dans le cas d'une requête préliminaire, l'agent enverra la requête `OPTIONS` sans les *credentials*, puis enverra la requête demandée avec ceux-ci uniquement si le serveur accepte la méthode demandée, l'origine source et les requêtes avec *credentials*.

Il est important de noter que les spécifications interdisent à un serveur d'accepter les requêtes authentifiées depuis toutes les origines. Si un serveur renvoie à la fois les en-têtes `Access-Control-Allow-Origin: *` et `Access-Control-Allow-Credentials: true`, alors l'agent devra ignorer ce dernier en-tête et refuser l'accès à la réponse du serveur pour toute requête authentifiée. Les pères des CORS ont ainsi posé un garde-fou pour éviter la probable prolifération de ce combo magique, lequel aurait alors mis totalement à bas la *Same Origin Policy* !

2.4 Particularités

Il est bon de savoir qu'en cas de redirections, l'agent les suivra toutes avant de renvoyer un résultat au script ayant généré la requête. Si l'une des redirections refuse l'origine, le script n'aura pas accès à la réponse.

2.5 Récapitulatif des différents cas de figure

Le tableau 1 récapitule les différents cas de figure dans le cas d'une requête simple et permet de comprendre dans quels cas un script provenant d'une origine http://script.com pourra accéder à la réponse du serveur http://victime.com, sachant que dans tous les cas, l'agent enverra une requête avec un en-tête `Origin: http://script.com`.

Dans le cas d'une requête préliminaire, les choses sont plus simples puisque la requête demandée est envoyée uniquement si le résultat de la requête `OPTIONS` est conforme aux attentes (voir tableau 2).

Rendez-vous dans le prochain numéro de votre magazine préféré pour la suite de cet article où seront abordées les conséquences du fonctionnement des CORS, les configurations vulnérables, les principales exploitations qui peuvent en être faites et quelques suggestions à destination des pentesteurs... ■

■ Références

[1] RFC 6454, « The Web Origin Concept » : https://tools.ietf.org/html/rfc6454

[2] Mariem El Gharbi, « La SOP et ses contournements », *MISC HS n°4*, octobre 2011

[3] W3C XMLHttpRequest Living Standard : https://xhr.spec.whatwg.org/

[4] W3C « CORS » : https://www.w3.org/TR/cors/

shutterstock

HS2
Hervé Schauer Sécurité

Formation Vie privée, Droit de la cybersécurité et Continuité d'activité

PROGRAMME

Vie privée et droit de la cybersécurité

RGPD :
RGPD/GDPR / Réglement Européen sur la Protection des Données personelles
DPO :
Formation DPO / Privacy Implementer
PIA :
PIA / ISO29134 / Appréciation des impacts sur la vie privée
SECUSANTE :
Protection des données de santé et vie privée
SECUDROIT :
Droit de la cybersécurité
SECUCLOUD :
Sécurité du cloud

Continuité d'activité

RPCA :
Formation RPCA
ISO22LA :
ISO22301 Lead Auditor
ISO22LI :
ISO22301 Lead Implementer

+33 644 014 072

www.hs2.fr

formations@hs2.fr

ATTAQUES PAR CANAUX AUXILIAIRES SUR AES
PARTIE 2

Melissa AZOUAOUI – melissa.azouaoui@nxp.com
PhD Student
Vincent VERNEUIL – vincent.verneuil@nxp.com
Security Analyst

mots-clés : SIDE-CHANNEL / CRYPTOGRAPHIE / COMPOSANTS EMBARQUÉS / AES

Nous avons présenté dans la première partie de cet article le principe des attaques par canaux auxiliaires sur les algorithmes de cryptographie à clef privée en prenant pour exemple l'AES. Dans cette seconde partie, nous nous intéressons aux contre-mesures utilisées pour s'en protéger et poursuivons notre exposé avec deux classes d'attaques plus puissantes : les attaques multivariées et les attaques profilées.

Dans la première partie de cet article **[Azo18]**, nous avons fait un tour d'horizon des attaques univariées sur une implémentation non protégée de l'AES. En pratique, différentes contre-mesures peuvent être appliquées afin de protéger une implémentation contre les attaques par canaux auxiliaires (*Side-Channel Attacks*, *SCA*). Dans cette seconde partie, nous rappelons les principes fondamentaux des contre-mesures, en particulier le principe du masquage, qui permet de garantir la résistance d'une implémentation contre les attaques univariées. Nous présentons ensuite une classe d'attaques plus complexes, les attaques multivariées, qui permettent d'attaquer une implémentation masquée. Finalement, nous décrivons une classe d'attaques plus puissantes, les SCA profilées, qui requièrent une étape préalable de caractérisation du composant à attaquer.

1 Contre-mesures

Les contre-mesures permettant de se prémunir des SCA avancées peuvent être divisées en deux catégories principales : la première regroupe les techniques visant à diminuer le rapport signal-bruit des observations, tandis que la seconde catégorie consiste à masquer les valeurs intermédiaires manipulées – c'est-à-dire rendre les valeurs physiquement manipulées par le composant indépendantes dans une certaine mesure des valeurs logiques que l'on veut calculer.

Les contre-mesures de la première catégorie peuvent agir à différents niveaux : directement au niveau matériel par l'utilisation de circuits limitant les fuites par leur conception telle que la technique dual-rail **[Che06]**, ou simplement par l'ajout de bruit (qui peut prendre la forme de *jitter*, de bruit généré par un autre composant, etc.). Au niveau logiciel, le rapport signal-bruit peut-être également diminué par l'implémentation de techniques visant à désynchroniser les observations dans le domaine temporel. Cela est habituellement implémenté sous forme de délais (pseudo-)aléatoires, ou bien de permutations (pseudo-)aléatoires d'opérations sensibles si celles-ci sont indépendantes les unes des autres – technique éventuellement utilisée en conjonction avec des opérations factices pour cacher une « vraie » opération parmi n « fausses ».

L'étude des techniques de masquage a donné lieu à de nombreuses recherches, dont nous ne présentons ici que les aspects les plus généraux. Un schéma de masquage *à l'ordre d* consiste à masquer une valeur de l bits en utilisant une représentation redondante de cette valeur sur $(d + 1) \cdot l$ bits dont la valeur peut être rendue partiellement aléatoire et telle que la totalité des $(d + 1) \cdot l$ bits est nécessaire pour retrouver la valeur originale. D'une manière équivalente, on dit que la valeur originale est divisée en $d + 1$ *partages*. Si le composant manipule chaque partage indépendamment, l'attaquant requiert au moins $d + 1$ observations (ou points de fuite) pour exploiter une fuite d'information sur la valeur originale.

Par exemple, le schéma booléen est un schéma de masquage classique en cryptographie à clef privée : pour masquer une variable x de l bits, on tire de manière (pseudo-)aléatoire d valeurs de l bits $m_1, m_2, \ldots, m_d$, puis on calcule la valeur masquée : $x \oplus m_1 \oplus m_2 \oplus \ldots \oplus m_d$

où ⊕ représente le x-or, ou « ou exclusif » bit-à-bit. La valeur de x ne peut être retrouvée qu'en combinant l'ensemble des partages, c'est-à-dire en calculant le x-or de la valeur masquée de x avec l'ensemble des masques.

L'implémentation d'un schéma de masquage dans un algorithme tel que l'AES s'avère en pratique très simple pour les parties linéaires de l'algorithme, telles que `AddRoundKey`, `ShiftRows`, et `MixColumn`, puisque ces opérations sont transparentes pour le schéma de masquage – i.e. $f(x \oplus m) = f(x) \oplus f(m)$. La difficulté et le coût de l'implémentation d'un schéma de masquage proviennent donc des opérations non-linéaires de l'algorithme – `SubBytes` dans le cas de l'AES. Le principe général du masquage de la S-box de l'AES est représenté en Fig. 1, où x représente un octet du message, m le masque de cet octet appliqué avant l'addition de l'octet de clef k, z la sortie (non masquée) de la S-box, m' le masque en sortie de la S-box et S' la S-box modifiée telle que $S'(y \oplus m) = S(y) \oplus m'$ pour toute valeur y. Différentes techniques d'implémentation ont été proposées dans la littérature : une méthode générale consiste à recalculer la S-box pour un masque donné **[Gou99]**, d'autres techniques exploitent la structure spécifique de la S-box de l'AES **[Can05]**. En pratique, le coût de l'implémentation augmente rapidement avec l'ordre du schéma de masquage – typiquement de manière quadratique pour la plupart des opérations arithmétiques et logiques.

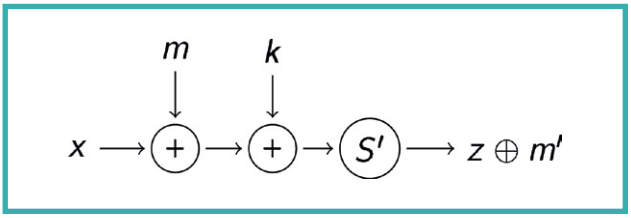


Fig. 1 : Addition de la clef et S-box masquées dans la première ronde de l'AES.

2 Attaques multivariées non profilées

Comme nous l'avons présenté dans la première partie de cet article, le principe général d'une SCA sur un algorithme de chiffrement par bloc consiste à répéter l'opération suivante pour chaque mot de la clef k que l'on cible : pour chaque valeur k' possible de k, on calcule les valeurs intermédiaires v' qui seraient manipulées par le composant si k' était la valeur du mot k. Si l'attaque prend en compte un modèle de fuite L, on calcule $L(v')$ puis l'on calcule $s_{k'} = D(T, L(v'))$ le score associé à k', où D est le distingueur de l'attaque et T la matrice des observations telle que chaque ligne correspond à une exécution de l'opération ciblée et chaque colonne à un instant de la mesure. Les scores obtenus permettent de classer les différentes valeurs k' par vraisemblance décroissante.

Dans le cas d'une attaque univariée, nous avons vu que les colonnes de T sont utilisées indépendamment pour le calcul du score. En d'autres termes, l'attaque est répétée pour chaque colonne de T et l'on conserve en principe le score maximum atteint pour chaque valeur k'.

Une attaque multivariée fonctionne sur le même principe qu'une attaque univariée, mais combine les points de mesure de différents instants de manière à exploiter de l'information contenue dans les traces qui serait divisée entre différents instants dans le temps, comme c'est le cas lorsqu'un schéma de masquage est employé.

2.1 DPA à l'ordre d

On appelle *DPA à l'ordre d* (ou *High-Order* DPA, *HODPA*), une SCA avancée où d points de mesure sont combinés par une fonction de combinaison C dont la sortie est passée au distingueur de l'attaque. Le calcul du score associé à une hypothèse k' devient donc $s_{k'} = D(C(T), L(v'))$ où C prend en entrée d colonnes de T et retourne un vecteur colonne.

En combinant d points de mesure, il est théoriquement possible d'attaquer un schéma de masquage à l'ordre d - 1 si l'on combine ensemble les points de mesure correspondant à la manipulation de la valeur secrète masquée et des d - 1 masques.

Les fonctions de combinaison habituellement utilisées sont (en prenant l'exemple d'une attaque d'ordre 2 combinant les points de mesure correspondant aux instants $j_1 = a$ et $j_2 = b$) :

- le produit centré $(T_{i,a} - \mu_a)(T_{i,b} - \mu_b)$ où μ_a, resp. μ_b, représente la moyenne du vecteur colonne d'indice a, resp. b, de T **[Cha99]** ;
- la valeur absolue de la différence $|T_{i,a} - T_{i,b}|$ **[Mes00]**.

Note

Il a été démontré que le produit centré est la meilleure fonction de combinaison pour la CPA à condition que le niveau de bruit présent dans les mesures soit suffisamment élevé [Pro09].

Pour illustrer le principe de cette attaque, prenons l'exemple du schéma de masquage booléen au premier ordre : une valeur intermédiaire x est masquée par un masque m (pseudo-)aléatoire et uniformément distribué. Soit j_1, resp. j_2, un instant où $x \oplus m$, resp. m, est manipulé par le composant cible. Le schéma de masquage résiste aux attaques univariées, car la valeur $x \oplus m$, d'une part, et celle de m, d'autre part, sont chacune indépendantes de la valeur de x. Leurs observations, prises indépendamment, ne peuvent donc pas révéler d'informations sur x. Cependant, l'observation de la manipulation du couple $(x \oplus m, m)$ est clairement liée à la valeur de x, ainsi la combinaison des observations aux instants j_1 et j_2 peut permettre de retrouver x.

Notons que le choix du distingueur utilisé par une HODPA est indépendant du choix de la fonction de combinaison. En pratique, on retrouve donc les distingueurs déjà présentés pour les attaques univariées (différence des moyennes, coefficient de corrélation de Pearson, information mutuelle, etc.).

En théorie, il est toujours possible d'attaquer une implémentation masquée à l'ordre d avec une DPA d'ordre $d + 1$ ou plus. En pratique, le masquage est pourtant une contre-mesure efficace en raison des deux problèmes qui se posent lorsqu'on doit appliquer une attaque d'ordre élevé :

1) si les instants j_1, j_2, ..., j_d ne sont pas connus à l'avance par l'attaquant, la complexité calculatoire de l'attaque augmente exponentiellement avec l'ordre du schéma de masquage, puisqu'un attaquant doit tester toutes les combinaisons d'instants possibles ;

2) le rapport signal-bruit de la combinaison des observations décroît exponentiellement avec le nombre d'observations combinées, c'est-à-dire avec l'ordre du schéma de masquage.

Cette dernière raison permet de garantir en théorie la protection d'une implémentation masquée jusqu'à un nombre maximum d'exécutions en supposant une borne supérieure sur le rapport signal-bruit, comme illustré en Fig. 2.

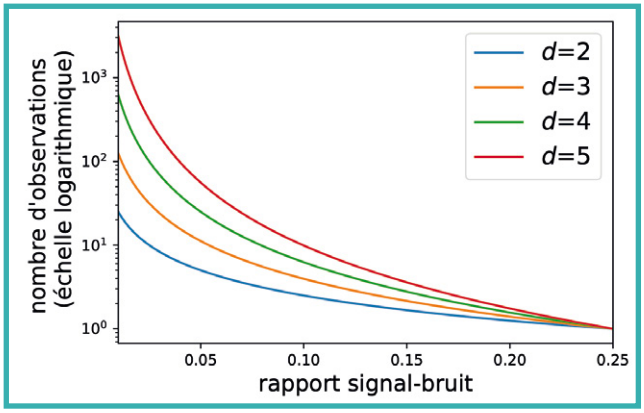


Fig. 2 : Évolution du nombre d'observations nécessaires pour attaquer une valeur masquée en fonction du rapport signal-bruit et de l'ordre de masquage.

2.2 Attaques par collision

Les SCA par collision ont été introduites en 2003 sur le DES **[Sch03]** et en 2004 sur l'AES **[Sch04]**. Le principe de ces attaques est de comparer directement la fuite de deux opérations afin d'en déduire si les opérandes manipulés sont égaux ou non. Elles appartiennent donc à la catégorie des attaques multivariées puisque différents points de mesures sont utilisés par l'attaque, mais diffèrent de la HODPA par leur principe.

Une propriété intéressante de ces attaques est qu'elles ne requièrent pas d'hypothèse sur le modèle de fuite puisqu'on ne fait pas d'hypothèse sur les valeurs manipulées. En contrepartie, l'information obtenue ne permet pas en général de retrouver la clef complète, mais seulement des relations entre les mots de la clef.

De nombreuses variantes ont été proposées dans la littérature, mais nous ne présentons ici que l'une d'entre elles par souci de simplicité. Considérons deux octets distincts x_i et x_j du message clair, les octets de clef correspondants k_i et k_j, et les sorties de S-box correspondantes z_i et z_j dans la première ronde (contrairement au reste de l'article, les indices i et j ne désignent pas les lignes et les colonnes de T, mais sont des indices des mots du message, de la clef et des valeurs intermédiaires correspondantes). Si un attaquant parvient à identifier une exécution pour laquelle les deux S-box manipulent des valeurs identiques, i.e. $z_i = z_j$, il obtient la relation $x_i \oplus k_i = x_j \oplus k_j$ car la S-Box de l'AES est une fonction bijective, et peut en déduire que $k_i \oplus k_j = x_i \oplus x_j$. Comme x_i et x_j sont connus, l'attaquant obtient une relation directe entre k_i et k_j.

Il est ainsi possible de trouver des relations entre tous les octets de la clef (15 relations indépendantes sont suffisantes), ce qui fournit 120 bits d'information sur la clef – la valeur d'un seul octet détermine alors la valeur entière de la clef. Les 8 bits restants peuvent être simplement retrouvés par recherche exhaustive.

Les techniques de masquage que nous avons présentées précédemment permettent de se prémunir des attaques par collision, à condition que les masques soient tirés de manière (pseudo-)aléatoire uniforme. En contrepartie, lorsque les masques sont utilisés plus d'une fois pour des raisons d'efficacité (temps d'exécution ou espace mémoire), une implémentation peut rester vulnérable aux attaques par collision. Par exemple, dans l'exemple décrit ci-dessus, si les valeurs manipulées pour les octets i et j sont masquées par un même masque m, l'attaque reste applicable, car $z_i \oplus m = z_j \oplus m$ est équivalent à $z_i = z_j$. Ceci souligne le soin particulier qui doit être apporté à l'implémentation d'un schéma de masquage pour garantir la sécurité d'une implémentation.

2.3 Attaques à l'aveugle

Les attaques à l'aveugle sont appelées ainsi parce qu'elles ne nécessitent pas de connaître l'entrée de l'opération ciblée, permettant ainsi d'attaquer n'importe quelle ronde de l'algorithme (ou bien des protocoles particuliers où l'entrée de l'algorithme est inconnue ou masquée par exemple).

Ce type d'attaque part de l'observation que la distribution jointe du poids de Hamming de certains couples de variables intermédiaires de l'AES dépend d'un octet de la clef de ronde. Par exemple, si l'on considère une variable aléatoire X correspondant à un octet x_i de l'entrée d'une ronde donnée et Z la variable aléatoire correspondant à $z_i = S(x_i \oplus k_i)$, la sortie de la S-box de cette ronde, alors la distribution jointe des variables aléatoires HW(X), HW(Z) dépend de la valeur de k_i (où HW

est la fonction retournant le poids de Hamming) comme illustré en Fig. 3. Il est ainsi possible de retrouver k_i si le modèle de fuite du composant ciblé est connu et que l'attaquant parvient à localiser préalablement les manipulations de x_i et z_i et estimer leur poids de Hamming **[CRI7]**.

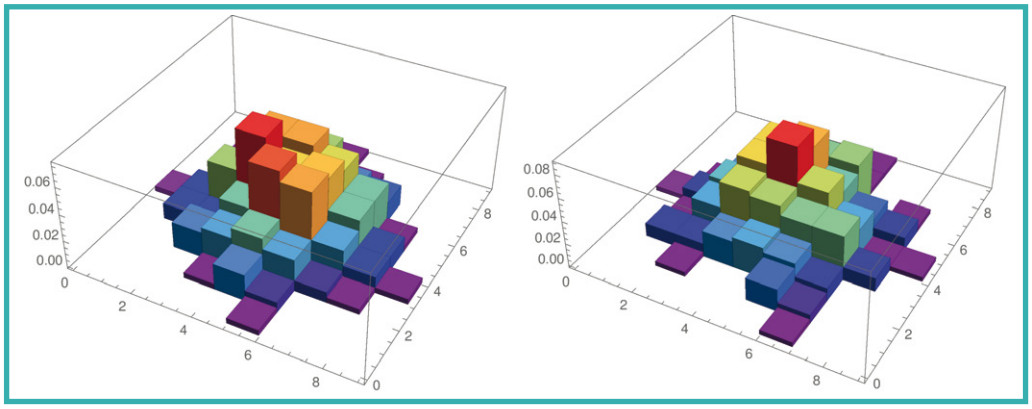


Fig. 3 : Distributions jointes de HW(X) et HW(Z) pour k_i = 39 (à gauche) et k_i = 167 (à droite).

Bien qu'elles soient par nature multivariées, il est possible de se protéger de ces attaques par l'utilisation d'un schéma de masquage au premier ordre. Il est cependant nécessaire que celui-ci soit implémenté avec précaution, car, comme dans le cas des attaques par collision, la réutilisation des masques peut rendre la contre-mesure inefficace.

3 Attaques profilées

Les SCA profilées sont plus puissantes que les attaques précédentes, car elles permettent de caractériser le modèle de fuite du composant, alors que les attaques non profilées nécessitent le plus souvent de faire une hypothèse sur celui-ci. Une SCA profilée se déroule en deux étapes :

1. Le *profilage* dont le but est de modéliser le comportement du composant pendant le calcul cryptographique. En d'autres termes, cette étape correspond à la caractérisation des fuites par canaux auxiliaires du composant en fonction de la valeur manipulée. Il est pour cela nécessaire de disposer d'un composant ouvert identique à celui que l'on cible.
2. L'*exploitation* est l'étape qui permet d'extraire de l'information sur la valeur manipulée par le composant ciblé, en comparant les observations physiques à la prédiction fournie par le modèle obtenu à l'étape précédente pour chaque hypothèse sur cette valeur.

Les attaques profilées les plus documentées dans la littérature sont les attaques par modèle **[Cha02]** et les attaques basées sur un modèle stochastique, plus particulièrement la régression linéaire **[Lem05]**.

3.1 Attaques par modèle

Les attaques par modèle (*Template Attack*, TA) introduites par Chari et al. **[Cha02]**, supposent que le bruit qui perturbe les observations physiques générées par le composant ciblé est distribué selon une loi normale. Cette hypothèse est basée sur le théorème central limite, qui établit que la somme de variables aléatoires indépendantes est normalement distribuée. Lorsque l'on cible une valeur manipulée par le composant, toutes les fuites provenant de calculs ne dépendant pas de cette valeur sont considérées comme étant aléatoires. Le bruit correspondant à la somme de ces fuites et sa loi de probabilité peuvent donc être approximés par une loi gaussienne.

L'étape de profilage d'une TA consiste à estimer la moyenne du signal et la distribution du bruit gaussien pour chaque valeur v' d'une variable ciblée. En utilisant un ensemble de N observations, on estime la moyenne $\mu_{v'} = N^{-1} \Sigma\ T_i$ du signal et la variance du bruit $\sigma_{v'}^2 = (N - 1)^{-1} \Sigma\ (T_i - \mu_{v'})^2$ dans le cas d'une attaque univariée ou, dans le cas d'une attaque multivariée, sa matrice de covariance $\Lambda_{v'} = (N - 1)^{-1} \Sigma\ (T_i - \mu_{v'})^T (T_i - \mu_{v'})$. Le couple $(\mu_{v'}, \sigma_{v'}^2)$ – ou $(\mu_{v'}, \Lambda_{v'})$ – est appelé le modèle – ou le *template* – de v'.

L'étape d'exploitation qui vise à retrouver la valeur de la variable ciblée pendant un calcul manipulant un secret est basée sur le principe de maximum de vraisemblance. La probabilité que le bruit d'une trace T soit généré par un calcul manipulant la valeur v' est $P(v' | T) = Norm(T | \mu_{v'}, \sigma_{v'})$ pour une attaque univariée ou $Norm(T | \mu_{v'}, \Lambda_{v'})$ pour une attaque multivariée. Ainsi, l'attaquant sélectionne la valeur la plus vraisemblable $v = \text{argmax}_{v'} P(v' | T)$. L'attaquant peut combiner plusieurs observations de v en estimant la probabilité de leur moyenne pour diminuer l'influence du bruit s'il est possible de mesurer plusieurs traces où la même valeur v est manipulée.

Une TA peut donc être utilisée pour attaquer directement la valeur d'un mot de la clef secrète lorsque celui-ci est manipulé, par exemple pendant une opération de chargement, de démasquage... Cependant, si le modèle de fuite du composant est non-injectif, l'attaquant ne peut obtenir, au mieux, qu'une information partielle sur v. Par exemple, si le modèle de fuite correspond au poids de Hamming et si v est un mot de 8 bits, 16 bits ou 32 bits, l'identification de la classe de v permet en moyenne de retrouver respectivement environ 2,5 bits, 3 bits ou 3,5 bits d'information sur v.

Au lieu de profiler selon les hypothèses sur un mot de clef, on peut caractériser la fuite d'une variable intermédiaire d'un calcul dépendant de la clef et d'une valeur connue. On peut alors appliquer une attaque

similaire à une DPA en utilisant le principe du maximum de vraisemblance comme distingueur. Comme pour la DPA, il est alors possible de retrouver complètement la clef, même si le composant ne fuit que le poids de Hamming des valeurs manipulées.

Note

Pour mener à bien une TA, il est nécessaire de sélectionner un petit nombre de points des observations (de l'ordre de quelques dizaines) appelés *points d'intérêt*. Si trop peu de points sont sélectionnés, toute l'information contenue dans l'observation de la fuite n'est pas exploitée et l'attaque est sous-optimale. Si trop de points sont utilisés, la complexité de l'attaque est inutilement élevée et peut mener à des problèmes de calcul numérique, en particulier lors de l'inversion de la matrice de covariance nécessaire à l'évaluation de $Norm(T \mid \mu_{v'}, \Lambda_{v'})$.

Différentes méthodes de sélection des points d'intérêts ont été proposées, comme par exemple l'analyse de la variance des traces ou du rapport signal-bruit [Cho13].

3.2 Régression linéaire

Comme précédemment, on considère que les observations du composant ciblé sont la somme d'un terme déterministe dépendant des valeurs manipulées et d'un terme non-déterministe correspondant aux autres sources ne dépendant pas de v que l'on considère comme du bruit. Le terme déterministe est noté $h(v)$, où v est la variable manipulée par le composant ; l'attaque consiste alors à estimer la fonction de fuite h et le bruit avec deux ensembles de traces différents T_1 et T_2 **[Lem05]**.

On approxime h par une combinaison linéaire de fonctions g_j de la variable v. Un modèle simple consiste à choisir g_j comme étant le j-ième bit de v. Ainsi, h est approximée par $h'(v) = \Sigma_j \ \beta_j \ . \ v[j]$. Pour estimer les coefficients β_j, on applique la méthode des moindres carrés de la manière suivante. Soit A la matrice dont les lignes correspondent à la représentation binaire de v pour chaque valeur possible et T_1 un vecteur contenant les fuites correspondant à la manipulation de chaque valeur possible de v. On calcule alors le vecteur b des coefficients β_j, $b = [\beta_0, \beta_1, ..., \beta_7] = (A^T \ . \ A)^{-1} \ . \ A^T \ . T_1$.

Comme pour la TA, le bruit est supposé suivre une loi normale. La matrice de covariance Λ reflétant le bruit est calculée à l'aide d'un autre ensemble de traces T_2 de N traces par $\Lambda = N^{-1} \ (T_2 - A.b)^T \ (T_2 - A.b)$.

La phase d'exploitation est similaire à l'attaque précédente : l'attaquant ayant acquis une observation T à partir du composant ciblé, sélectionne la valeur v suivant le principe du maximum de vraisemblance $v = \text{argmax}_{v'} \ Norm(T \mid h'(v') \ , \Lambda)$.

Contrairement à la TA qui ne fait aucune hypothèse sur la fonction de fuite, le modèle stochastique présuppose la forme de cette fonction – i.e. les monômes g_j. Il est possible de choisir un modèle plus complexe que le modèle linéaire présenté ici, par exemple un modèle incluant tous les termes d'ordre deux $v[i] \ . \ v[j]$ ou même d'ordre supérieur. Des modèles plus complexes peuvent permettre une caractérisation plus fine du modèle de fuite, mais en contrepartie l'augmentation du nombre de paramètres du modèle rend la phase d'estimation plus complexe en pratique (phénomène de sur-ajustement ou *overfitting*).

3.3 Apprentissage automatique

L'apprentissage automatique (*Machine Learning*, ML) est un domaine qui englobe différentes méthodes pour effectuer des tâches variées telles que la classification, le partitionnement, la régression, etc. Une attaque profilée s'apparentant à un problème de classification, il est naturel que les techniques de ML de partitionnement de données puissent également être utilisées pour effectuer une attaque. Parmi les méthodes proposées, citons le partitionnement en k-moyennes (*k-means*), les machines à vecteurs de support (*SVM*), les arbres de décision et les forêts aléatoires (*RF*), ou encore les cartes auto adaptatives (*SOM*) **[Hos11, Mar11, Heu12]**.

Note

Les RF et les SVM sont des modèles d'apprentissage supervisés utilisés pour la classification et la régression.

De manière très concise, les SVM effectuent une projection des observations dans un espace de dimension supérieure où elles sont linéairement séparables en fonction de leur classe, qui correspondent le plus souvent à la valeur de l'hypothèse de clef ou à la valeur d'une variable intermédiaire dépendante de la clef.

Un arbre de décision prédit la valeur d'une variable en apprenant des règles de décision simples inférées depuis les données d'apprentissage. Le principe des RF est d'effectuer un apprentissage sur de multiples arbres de décision afin d'aboutir à une décision par consensus, réduisant ainsi la variance des résultats.

Les méthodes basées sur les SVM et les RF paraissent les plus prometteuses dans le domaine des SCA. Leur efficacité pour attaquer une implémentation masquée est supérieure à celle des méthodes profilées classiques comme la TA ou la régression linéaire, car aucune hypothèse n'est faite sur la fonction de densité de probabilité des fuites observées, contrairement aux méthodes précédentes qui supposent une distribution gaussienne **[Ler13]**.

L'apprentissage profond (*Deep Learning*, DL) est un ensemble de méthodes qui permettent d'analyser

les données grâce à une modélisation avec un haut niveau d'abstraction en utilisant différentes couches de réseaux de neurones artificiels. Différents types de réseaux de neurones ont été utilisés pour pratiquer des SCA, tels que les auto-encodeurs (*AE*), les réseaux longue mémoire à court terme (*LSTM*), les réseaux de neurones récurrents (*RNN*) et les réseaux de neurones convolutifs (*CNN*) **[Gil15, Mag16, Cag17, Pic18]**. En particulier, il a été montré expérimentalement que les CNN, combinés à des méthodes d'augmentation du jeu de traces de profilage, sont significativement plus robustes au désalignement des traces que les TA dont la réussite dépend intrinsèquement des points de la trace que l'on cible **[Cag17]**. Ce résultat est encourageant, car l'alignement des traces et la sélection des points d'intérêts sont des problèmes qui peuvent s'avérer complexes en pratique.

Conclusion

Le niveau de sophistication des SCA n'a cessé de progresser depuis leur apparition en 1996. Aux premières attaques essentiellement empiriques ont succédé des attaques aux concepts plus généraux, comme l'illustre l'exemple de la DPA dont ont découlé la CPA, puis la MIA.

Le masquage est la pierre angulaire des contre-mesures contre ces attaques. Bien que la sécurité apportée par les schémas de masquage soit aujourd'hui largement étudiée et bien comprise, l'évaluation de la sécurité des implémentations reste une question complexe en pratique – voir l'article dédié à ce sujet dans ce numéro.

Les SCA profilées requièrent un accès ouvert à un composant identique au composant ciblé, mais constituent la forme d'attaque la plus puissante. Ce domaine, comme beaucoup d'autres, est aujourd'hui en pleine ébullition depuis que des techniques de DL ont montré un intérêt significatif sur les méthodes classiques comme les TA. Il est probable que ces techniques se perfectionnent encore dans les années à venir et posent de nouveaux défis pour la sécurisation des composants embarqués. Par exemple, un travail très récent propose l'utilisation des techniques de DL dans le cadre d'attaques non-profilées **[Tim17]**. ■

Remerciements

Nous souhaitons vivement remercier nos relecteurs Christophe Clavier et Jean-Baptiste Bédrune, ainsi que Léo Reynaud pour l'illustration des distributions jointes. Cette série d'articles a été financée en partie par la Commission Européenne à travers le programme H2020 par le projet REASSURE (réf. 731591), plus de détails sur http://reassure.eu.

Retrouvez toutes les références de cet article sur le blog de MISC : https://www.miscmag.com/ .

LA SURFACE D'ATTAQUE : SON UTILITÉ, SES LIMITES, SES NOUVEAUX DÉFIS

Daniel VENTRE
Laboratoire CESDIP

mots-clés : SURFACE D'ATTAQUE, DÉFINITION, RÉDUCTION/EXPANSION, IA

La surface d'attaque est une notion essentielle en cybersécurité. Nous proposons dans cet article un rappel des divers éléments de sa définition (partie 1), puis formulons quelques hypothèses relatives à l'objectif de réduction de la surface d'attaque, qui se heurte à plusieurs difficultés (partie 2). Enfin, dans une visée plus prospective, nous interrogeons-nous sur les effets potentiels de l'intelligence artificielle sur la définition de la surface d'attaque (partie 3).

1 Définitions, enjeux

C'est au tournant des années 1990-2000 que l'on voit apparaître la notion de « surface d'attaque ». Ainsi apparaît-elle depuis dans des articles qui en proposent définitions, modèles formels **[1]**, taxonomies **[2]**, métriques **[3]**, méthodes d'analyses (pour des environnements et applications particuliers tels que BlackBerry **[4]**, Windows **[5]**, Cloud, web browser **[6]**, mobiles **[7]** ; ou pour des dimensions spécifiques telles que la surface d'attaque humaine **[8]**...), méthodes de réduction **[9]**.

1.1 Définitions

La notion de surface d'attaque est aujourd'hui centrale en cybersécurité, gestion du risque et de la menace. Les évolutions technologiques incessantes conjuguées à l'expansion accélérée du cyberespace (cloud computing, Internet des objets **[10]**) contribuent par ailleurs à une remise en question constante de la notion, de son approche.

La « surface d'attaque » désigne « *les potentielles vulnérabilités qui peuvent être exposées par un système par le biais de la somme de ses dépendances avec d'autres sous-systèmes. Le terme est habituellement utilisé pour désigner l'ensemble des ressources sur lesquelles repose un système, et qui pourraient être utilisées pour lancer une attaque contre ce dernier. Les systèmes avec des surfaces d'attaques plus larges sont jugés comme potentiellement plus vulnérables* » **[11]** ; « *la surface d'attaque est notre niveau d'exposition, l'ensemble de nos vulnérabilités atteignables, exploitables par des tiers* » **[12]**, « *il s'agit de tous les points d'entrée et les points de communication qu'un système d'information possède avec l'extérieur* » **[13]**. « *La surface d'attaque est la somme totale des vulnérabilités d'un ordinateur (computing device) ou d'un réseau, qui sont accessibles à un hacker* » **[14]**.

Les « points » de vulnérabilité sont alors de nature diverse, selon le type de système considéré (une application, un ordinateur, un serveur, une infrastructure critique, voire l'ensemble des réseaux d'un pays, etc.). Les « points » peuvent alors être les failles d'un logiciel, celles des protocoles de communication, dans les droits d'accès, etc.

On considère habituellement trois catégories de surface d'attaque : réseau, logicielle et physique (correspondant aux trois dimensions constitutives du cyberespace, à savoir réseau/hardware, software, meatware). La « surface d'attaque humaine » est également intégrée dans ces approches.

La mesure de la surface d'attaque d'un système est un indicateur de la sécurité du système **[15]**. La métrique de la surface d'attaque a fait l'objet de nombreuses recherches **[16]**. Nous renvoyons le lecteur aux travaux de Michael Howard (Microsoft), qui a imaginé des méthodes de mesure de la surface d'attaque d'une application et de suivi des modifications de la surface d'attaque dans le temps (*Relative Attack Surface Quotient* – RSQ), ainsi que ceux de Pratyusa K. Manadhata étendant la méthode à l'étude de systèmes de plus grande taille **[17]**.

1.2 Difficultés

Plusieurs obstacles s'opposent à une vision précise de cette surface d'attaque, tels que l'hétérogénéité des systèmes en réseau, leurs dépendances complexes, le dynamisme de leurs interactions **[18]**.

Convenons que si l'identification de tous les points de vulnérabilités peut être un objectif, il demeure en réalité difficilement atteignable. Un recensement exhaustif de tous les points de vulnérabilité semble hors de portée. On devrait plutôt distinguer :

- Une surface d'attaque réelle, totale, mais dont la connaissance parfaite est hors de portée.
- Une surface d'attaque identifiable, sous-ensemble de la surface totale réelle. Cette surface d'attaque qui servira à la définition de mesures de sécurité idoines, résultera de choix (il peut y avoir hiérarchisation des points de vulnérabilité, des priorités). Pour un système donné, on peut donc envisager plusieurs définitions ou périmètres de sa surface d'attaque.
- La différence entre la zone d'attaque totale/réelle et la surface d'attaque identifiée constitue une zone de risque, car elle ne fait pas l'objet de mesures de cybersécurité.

La surface d'attaque est dotée de quelques caractéristiques :

- Mal sécurisée, elle demeure surface d'attaque à part entière. La définition de la surface d'attaque n'est qu'un préalable, un prérequis.
- Étant difficilement mesurable dans sa totalité, la surface d'attaque identifiée est un compromis, et ce d'autant plus que sa définition peut résulter d'un processus d'analyse empreint de subjectivité, d'erreurs, d'oublis, de choix. Est-on jamais certain d'avoir parfaitement cartographié le système, d'avoir identifié tous les points d'attaques possibles ? Les points considérés comme sûrs, le sont-ils vraiment ou suffisamment ?
- Elle n'a pas de périmètre, de limite, de frontière strictement définissable : la complexité du cyberespace rend les systèmes interdépendants, les effets peuvent s'enchaîner les uns aux autres, produire des résultats imprévisibles.
- Elle est évolutive, différente à l'instant t0 et tn. La découverte, publication et exploitation de failles 0days font varier brutalement la dimension de la surface d'attaque par exemple. Elle doit donc être réévaluée constamment, une fois les attaques survenues, des failles mises à jour, des correctifs déployés.
- Pour tout système, on peut considérer que sa surface d'attaque diffère selon l'environnement dans lequel il est placé. Ainsi, un ordinateur portable est-il plus vulnérable quand il se connecte à un réseau public qu'au réseau sécurisé d'une d'entreprise ? L'environnement détermine la taille de la surface d'attaque d'un système.

1.3 La surface d'attaque de l'agresseur

Le cyber-agresseur a, lui aussi, ses surfaces d'attaques, ses propres points de vulnérabilité. Tout ce qui pourrait trahir sa présence ou son identité réelle, tout ce qui permettrait de remonter jusqu'à lui, de tracer son attaque, doit être considéré comme point de vulnérabilité.

La surface d'attaque de l'agresseur doit être reconstituée, quand ce dernier devient à son tour la « cible » de la police, de la justice, de la défense active (contre-attaques, représailles). Les points de vulnérabilité de l'agresseur résident dans les failles des outils ou méthodes d'anonymisation qu'il aura mobilisés pour son action ; dans sa tactique ou stratégie, qui révélera de lui ses modes opératoires; dans toute forme de trace de son passage ; par ce qu'il révélera de lui-même dans des forums, sur des médias sociaux ; par le niveau de confiance qu'il aura créé, maintenu ou altéré auprès de ses communautés de hackers ou de ses réseaux criminels, etc.

1.4 Quelques variables non techniques impactant la dimension de la surface d'attaque

Un système S plongé dans un milieu M1 ne supportera pas les mêmes risques, menaces, niveaux d'exposition, que dans un milieu M2. Il ne sera pas exposé de la même manière à l'instant T1 et T2. En raison de l'interconnexion et l'interdépendance des systèmes, S peut hériter des effets des vulnérabilités associées au milieu M. La surface d'attaque de S doit être appréciée en fonction de celle de M. Plusieurs variables contextuelles interviennent, qui peuvent être de nature spatiale, temporelle, politique/sociale/économique/juridique.

La législation d'un État impacte directement la dimension de la surface d'attaque. Le droit peut imposer des normes de sécurité dont l'application permet de réduire la surface d'attaque : obligations faites par exemple aux hébergeurs de données à caractère personnel d'en assurer la protection ; obligation faite aux entreprises de certains secteurs (infrastructures critiques notamment) de déclarer aux autorités les attaques dont elles sont victimes... Ces contraintes permettent soit de réduire a priori (avant les attaques) soit a posteriori (prise en compte des vulnérabilités révélées par les attaques) la surface d'attaque des systèmes concernés par la loi. Mais le droit peut au contraire élargir la surface d'attaque quand, par exemple en Chine, les autorités contraignent les entreprises étrangères qui y exercent leurs activités, à fournir les codes sources de leurs applications. De même, les pratiques visant à réduire la surface d'attaque, telles que le chiffrement des données, peuvent être encadrées par la loi, qui en pose les limites.

2 Réduire la surface d'attaque : défis

Se fondant sur la mesure de la surface d'attaque se construisent ensuite les mesures de cybersécurité. L'un des enjeux principaux réside dans la réduction de la surface d'attaque.

2.1 Réduction versus expansion inexorable

Le niveau d'exposition aux attaques est défini par la taille ou dimension de la surface d'attaque. Plus la surface est grande, plus vulnérable est le système. D'où le postulat suivant : en réduisant la taille de la surface, on réduit les risques d'attaque, l'exposition, la vulnérabilité du système dans son ensemble. Réduire la surface d'attaque vise à limiter les possibilités d'attaques en réduisant le nombre de points d'entrée, de prises en main par l'agresseur ; mais aussi à limiter les impacts, réduire le risque de propagation des effets des attaques à l'intérieur du système ou entre systèmes interdépendants.

Réduire la surface d'exposition est donc une tâche qui peut s'avérer à la fois coûteuse, complexe et insatisfaisante (car elle est un compromis entre ce qui est d'une part théoriquement souhaitable et de l'autre ce que les capacités permettent de faire). Tâche complexe, car il faut prendre en compte aussi bien les vulnérabilités logicielles (applications web, erreurs de programmation, failles 0days, etc.), réseaux, matérielles et humaines.

La réduction de la surface d'attaque est appliquée dans l'ensemble des secteurs d'activité. Le Département de la Défense américaine inscrit la réduction de la surface d'attaque au rang des 4 axes de la cyber-sécurisation de ses systèmes. Les 3 autres sont l'authentification forte, le renforcement des outils (devices) pour réduire le nombre de vecteurs d'attaques internes et externes, l'amélioration de la détection des activités adverses et des réponses à y apporter. La réduction de la surface d'attaque vise ici « *à réduire les vecteurs d'attaques externes* » **[19]** et consiste à réduire le nombre de connexions directes avec l'Internet public **[20]** (serveurs et applications telles que les sites internet publics) en déconnectant tout ce qui n'est pas opérationnellement nécessaire ; et pour ce qui doit être maintenu, mettre en place des hébergements dans des zones démilitarisées.

Mais cet exercice de réduction se heurte à une dynamique inverse : l'expansion continue du cyberespace qui se traduit par le développement de l'Internet des Objets, la complexité et l'interdépendance des systèmes croissante, le volume de lignes de codes produites, etc.

Or cette expansion du cyberespace signifie une expansion de la surface d'attaque. Le défi consiste donc à faire en sorte que, pour un système donné, sa surface d'attaque ne croisse pas proportionnellement à celle du cyberespace.

2.2 Limites : ce qui échappe à l'effort de réduction

On ne peut pas tout réduire. Il y a des parties du système sur lesquelles un acteur qui en a la responsabilité, n'a pas totalement prise. Plusieurs conditions peuvent contribuer à cette absence de maîtrise : quand les données sont hébergées sur des serveurs tiers, dans le cloud ; quand le recours à la sous-traitance dilue les responsabilités et la maîtrise des données, quand les accès sont multipliés ; quand le nombre d'acteurs intervenant sur un système croît, élargissant le périmètre de la surface d'attaque humaine ; quand des failles de type 0days ne sont pas divulguées et restent à l'usage de quelques acteurs ; quand les hardwares embarquent des vulnérabilités auxquelles il peut être difficile de pallier par des actions logicielles (la console Switch embarque une vulnérabilité de niveau hardware, qui a été qualifiée d'« impossible » à traiter ; faille Meltdown des processeurs x86-64 d'Intel **[21]** et faille Spectre concernant les processeurs de tous les fabricants) ; quand la surface d'attaque est trop grande pour être prise en compte dans sa globalité (« *La surface d'attaque totale du Département de la Défense est trop large pour être défendue contre toutes les menaces et trop vaste pour écarter toutes les vulnérabilités* ») **[22]** ; la dépendance technologique, vis-à-vis d'industries étrangères notamment, l'absence de souveraineté technologique, expose les systèmes à des failles indétectables (*backdoors*), ou plus simplement au risque de non-approvisionnement (mésaventure qui touche l'entreprise chinoise ZTE en 2018, contrainte de cesser la fabrication et commercialisation de ses téléphones portables. Cette fragilisation de l'entreprise est une autre facette de la « surface d'attaque », car il y a impossibilité de construire les systèmes eux-mêmes). Enfin, la réduction de la surface d'attaque peut inciter à la prise de mesures radicales, mais ces dernières s'avérer difficilement tenables dans les faits : le gouvernement de Singapour avait envisagé en 2016 d'interdire à tous les fonctionnaires l'usage de l'Internet dans le cadre professionnel. L'objectif était de réduire la surface d'exposition humaine (risques de *phishing*, d'ingénierie sociale, consultation de sites infectés, etc.). Des experts ont qualifié cette mesure de radicale, excessive et inefficace **[23]**. La déconnexion totale réduit certes la surface d'attaque (sans la réduire à zéro toutefois), mais elle a aussi des effets sur les pratiques de l'entreprise/administration, et dans des sociétés qui prônent le tout connecté, la mesure peut sembler à contre-courant.

2.3 Peut-il exister des zones immunes dans le cyberespace?

Existe-t-il dans le cyberespace des zones échappant à la notion de risques et menaces, où la mesure de la surface d'attaque serait égale à zéro ?

La manière dont se distribuent les attaques dans l'espace (observons par exemple les cartographies publiées en ligne par diverses entreprises de cybersécurité) permet de mettre en évidence des zones de concentration. Les attaques, bien que concernant l'ensemble des pays connectés, ne sont pas également distribuées. Ces répartitions, qui peuvent varier dans le temps, illustrent l'importance du contexte : des pays en conflit peuvent être plus sensibles à des cyberattaques massives que d'autres ; les cyberattaques d'espionnage touchent des cibles privilégiées par les agresseurs ; les zones plus fortement touchées sont peut-être aussi celles où la densité du cyberespace est plus élevée (plus d'adresses IP, d'internautes, de réseaux haut débit, etc.) La cartographie

des attaques virales révèle la surface de l'ensemble des systèmes qui étaient vulnérables. Les attaques par malwares ne visent ni n'atteignent tous les ordinateurs connectés. Les malwares peuvent être conçus pour ne toucher que des systèmes en particulier. Tous les systèmes ne sont donc pas exposés tout le temps de la même manière. On pourrait donc formuler l'hypothèse que des espaces, des territoires particuliers échappent durablement aux risques et menaces qui touchent les autres. Et donc que la surface d'attaque des systèmes se trouvant dans ces espaces puisse être considérée comme approchant de zéro. On aurait d'un côté des zones d'exposition de forte intensité, des routes du cybercrime, et de l'autre des zones d'hostilité moindre, relativement à l'abri, voire immunes. Réduire la surface d'attaque pourrait résider en un déplacement physique du système.

2.4 Quand les exigences de la sécurité étatique accroîtraient la surface d'attaque

Le rapport d'un collectif de chercheurs américains publié en 2015 **[24]** a qualifié de dangereuses les prétentions des États qui demandent à pouvoir accéder aux contenus des applications, des plateformes des médias sociaux, aux communications chiffrées. La systématisation de la mise en œuvre de *backdoors*, d'accès privilégiés, « exceptionnels », n'est pas selon eux souhaitable, pas acceptable. Les États, au nom de l'efficacité de la lutte contre le crime et le terrorisme, demandent depuis plus de 20 ans, à disposer des moyens d'accès aux contenus chiffrés. Le crime ne doit pas, selon eux, être en mesure d'échapper au regard de la police, des services de renseignements, de la justice. Les « Crypto Wars » des années 1990 désignaient l'opposition entre, d'un côté, les États soucieux d'accéder à tous les contenus, à tous les flux, à toutes les communications, en clair, à l'aide de backdoors ou de la mise à disposition des clefs de chiffrement ; et de l'autre leurs opposants, revendiquant le droit au recours à des solutions de chiffrement de haut niveau, l'enjeu majeur étant la défense du droit à la protection de la vie privée et plus largement des libertés individuelles. Les arguments des deux camps n'ont cessé de s'opposer depuis, sans trouver de véritable compromis. À plusieurs reprises les acteurs étatiques de la sécurité ont demandé à ce que soit levé l'obstacle du chiffrement. Les déboires de la justice américaine face à la résistance d'Apple, incapable elle-même de déchiffrer les contenus de ses utilisateurs, ont réactivé les débats.

Pour les détracteurs de la posture étatique, systématiser les accès exceptionnels serait préjudiciable à la cybersécurité, car reviendrait à augmenter la surface d'attaque de tous les systèmes concernés :

- Les États ne seraient probablement pas les seuls à exploiter les accès pour eux conçus. Le principe d'un accès exceptionnel exige une rigueur extrême dans la mise en œuvre des accès. Prévoir des systèmes/processus d'accès exceptionnels reviendrait à créer de nouvelles vulnérabilités

- Le principe même des accès exceptionnels va à l'encontre de la logique des acteurs de l'Internet qui, pour créer de la confiance et assurer la sécurité des transactions, sont engagés dans la voie du « toujours plus sécurisé », qui accepte peu de compromis.
- La mise en place d'accès exceptionnels augmenterait la complexité des systèmes. Or la complexité est l'un des ennemis majeurs de la cybersécurité

À la lumière des arguments de ce rapport, on doit s'interroger sur l'effet négatif qu'est susceptible d'avoir l'action étatique sur le dimensionnement de la surface d'exposition aux cyber-menaces.

3 Surface d'attaque et intelligence artificielle

Forte de son succès renouvelé, l'intelligence artificielle (IA) s'est naturellement invitée dans les considérations sur la surface d'attaque et plus généralement sur la cybersécurité. Si outils et méthodes de l'IA offrent potentiellement de nouvelles capacités à la cybersécurité, leur utilisation à des fins agressives est également possible. On qualifie les outils de l'IA de technologies duales. Les uns évoquent ainsi le recours aux capacités de l'IA pour rééquilibrer en faveur de la cybersécurité le rapport de force avec l'agresseur, habituellement considéré à l'avantage de ce dernier (effet de surprise, coût pour l'agresseur inférieur à ceux de la sécurité, surface d'attaque des cibles supérieure à la surface d'exposition des agresseurs, etc.) **[25]**. D'autres au contraire insistent sur les bénéfices que va tirer le cybercrime de ces nouvelles connaissances **[26]**.

Du côté de la cybersécurité, l'IA permettrait de :

- Réduire la surface d'attaque, en automatisant les recherches de vulnérabilités logicielles, détecter les comportements anormaux, superviser les trafics, détecter les menaces...
- Seconder les acteurs de la cybersécurité, dont les capacités peuvent être débordées par l'ampleur de la tâche (face aux volumes des attaques, à leur fréquence) **[27]**. L'IA pourrait être un multiplicateur de force pour les équipes de petite taille, devant répondre à des menaces de plus en plus nombreuses et rapides, des masses de données de plus en plus importantes **[28]**.
- L'IA ne réduirait pas la surface d'attaque, mais permettrait de la gérer de manière automatique, plus rapide, plus réactive, plus adaptative.

Du côté de l'agresseur, on évoque par exemple :

- le développement de botnets intelligents **[29]** ;
- les capacités de tromperie telles que le *deep fakes* (vidéos truquées) utilisables en ingénierie sociale, en désinformation **[30]** ;
- le recours à des algorithmes permettant de générer des campagnes de *spear-phishing* (forme de phishing ciblée), dans la langue natale des victimes par exemple, ou grâce à la synthèse vocale ;
- l'automatisation de la recherche de vulnérabilités logicielles ;
- l'utilisation des algorithmes de *deep-learning* pour améliorer l'efficacité des attaques ;
- l'adaptation automatique des malwares à chaque environnement attaqué (du « sur-mesure » en quelque sorte, mais possible à grande échelle) ;
- l'automatisation du développement de malwares ;
- le recours à des IA pour attaquer des systèmes cyber-physiques (exemple : hacker les systèmes de véhicules autonomes) ;
- la possibilité de propagande ciblée, de désinformation, de manipulation sociale ;
- plus généralement, l'IA permet d'envisager le développement de nouvelles « armes » cyber ;
- le recours à de l'IA pour attaquer la surface de vulnérabilités d'autres systèmes d'IA. Les systèmes d'IA ont leur propre surface d'exposition aux attaques et sont donc eux aussi vulnérables, des cibles. Comment mesurer, analyser et réduire la surface d'attaques des systèmes d'IA ?

De ces considérations, il ressort que :

- L'introduction de l'IA à la fois dans le champ de la sécurité et de l'agresseur, se soldera par des « affrontements » entre systèmes intelligents.
- Les capacités de l'IA au service de l'attaque ne sont pas encore généralisées. On est donc encore essentiellement dans la réflexion prospective. Mais la réalité pourrait évoluer très rapidement.
- L'IA repousse les frontières de la surface d'attaque « conventionnelle » en permettant de créer des vulnérabilités dans les dimensions logicielles, humaines, politiques. Les catégories de surfaces d'attaques que l'IA peut toucher sont multiples. La diversité des catégories de surface d'attaque existe déjà, comme nous le rappelions au début de cet article. L'IA en renouvelle cependant les possibilités d'exploitation. L'IA pourrait révéler de nouvelles composantes de la surface d'attaque.

Conclusion

Le principe de « *cybersecurity by design* » est confronté à ces évolutions technologiques en cours et à venir, dont nul ne sait exactement où elles peuvent mener. Face aux capacités de l'IA et du ML, la cybersécurité telle quelle fut pratiquée et imaginée jusqu'alors sera-t-elle suffisante ? La surface d'attaque ne va-t-elle pas voir ses limites déjà imprécises reculer encore davantage ? IA et ML vont-ils permettre au contraire de réduire drastiquement la dimension de toute surface d'attaque ? À ces questions il est encore impossible de répondre précisément. ■

Retrouvez toutes les références de cet article sur le blog de MISC : https://www.miscmag.com/ .

SANS EMEA

WWW.SANS.ORG

@SANSEMEA

The Largest and Most Trusted Source of Cyber Security Training, Certification, and Research in the World

TRAINING TOPICS INCLUDE:

- DIGITAL FORENSICS
- INCIDENT RESPONSE
- PEN TESTING
- SECURE SOFTWARE DEVELOPMENT
- SECURITY AWARENESS
- CYBER DEFENCE
- MANAGEMENT
- AUDIT
- INDUSTRIAL CONTROL SYSTEMS

Register now at SANS.org for one of our training events throughout Europe or the Middle East. Choose any training location from Amsterdam, Brussels, Copenhagen or Dubai to Oslo, Paris, Rome, Riyadh, Stockholm or Zurich.

TAKE SANS TRAINING AT A SANS TRAINING EVENT,
IN A PRIVATE CLASS OR ONLINE
WITH SANS ONDEMAND

SANS PARIS: JUNE 25-30 & NOV 19-24 2018

www.sans.org/emea emea@sans.org @SANSEMEA (+44) 203 384 3470

Quarkslab

SECURING EVERY BIT OF YOUR DATA

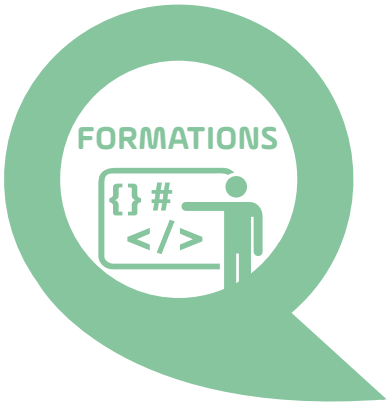
Les attaquants ciblent les données, et non les infrastructures qui sont régulièrement surveillées, testées et mises à jour. Quarkslab se concentre sur la sécurisation des données, au travers de 3 outils issus de notre R&D : IRMA (orchestrateur de threat intelligence), Epona (obfuscateur) et Ivy (reconnaissance 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**